Fusion and Extended Temperature Fusion FPGA Fabric User's Guide

Microsemi

Table of Contents

Introduction . . . 9
Contents . . . 9
Revision History . . . 9
Related Information . . . 9

1 FPGA Array Architecture in Low Power Flash Devices . . . 11
Device Architecture . . . 11
FPGA Array Architecture Support . . . 12
Device Overview . . . 13
Related Documents . . . 22
List of Changes . . . 22

2 Global Resources in Low Power Flash Devices . . . 23
Introduction . . . 23
Global Architecture . . . 23
Global Resource Support in Flash-Based Devices . . . 24
VersaNet Global Network Distribution . . . 25
Chip and Quadrant Global I/Os . . . 27
Spine Architecture . . . 33
Using Clock Aggregation . . . 36
Design Recommendations . . . 38
Conclusion . . . 50
Related Documents . . . 50
List of Changes . . . 51

3 Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs . . . 53
Introduction . . . 53
Overview of Clock Conditioning Circuitry . . . 53
CCC Support in Microsemi's Flash Devices . . . 55
Global Buffers with No Programmable Delays . . . 56
Global Buffer with Programmable Delay . . . 56
Global Buffers with PLL Function . . . 59
Global Input Selections . . . 63
Device-Specific Layout . . . 70
PLL Core Specifications . . . 76
Functional Description . . . 77
Software Configuration . . . 88
Detailed Usage Information . . . 96
Recommended Board-Level Considerations . . . 104
Conclusion . . . 105
Related Documents . . . 105
List of Changes . . . 105

4 Fusion Clock Resources . . . 109
Internal RC Oscillator . . . 110

Crystal Oscillator (XTLOSC) . . . 114
No-Glitch Multiplexer (NGMUX) . . . 123
Real-Time Counter (RTC) . . . 131
Related Documents . . . 136
List of Changes . . . 136

5 Fusion Embedded Flash Memory Blocks . . . 137
Using the Embedded Flash Memory for Initialization . . . 137
Using the Embedded Flash Memory for General Data Storage . . . 158
Microprocessor/Microcontroller Interface . . . 181
List of Changes . . . 190

6 FlashROM in Microsemi's Low Power Flash Devices . . . 191
Introduction . . . 191
Architecture of User Nonvolatile FlashROM . . . 191
FlashROM Support in Flash-Based Devices . . . 192
FlashROM Applications . . . 194
FlashROM Security . . . 195
Programming and Accessing FlashROM . . . 196
FlashROM Design Flow . . . 198
Custom Serialization Using FlashROM . . . 203
Conclusion . . . 204
Related Documents . . . 204
List of Changes . . . 204

7 SRAM and FIFO Memories in Microsemi's Low Power Flash Devices . . . 205
Introduction . . . 205
Device Architecture . . . 205
SRAM/FIFO Support in Flash-Based Devices . . . 208
SRAM and FIFO Architecture . . . 209
Memory Blocks and Macros . . . 209
Initializing the RAM/FIFO . . . 222
Software Support . . . 228
Conclusion . . . 231
List of Changes . . . 231

8 Designing the Fusion Analog System . . . 233
Introduction . . . 233
Analog-to-Digital Converter Background . . . 233
ADC Clock . . . 236
Sample Sequencing Overview . . . 239
Sample Rate and Sample Sequence Calculation . . . 240
Acquisition Time Calculation . . . 241
Prescaler Selection . . . 243
Analog Configuration MUX (ACM) . . . 243
List of Changes . . . 245

9 Fusion Design Solutions and Methodologies . . . 247
HDL Design with Analog System Soft IP . . . 247
Microprocessor/Microcontroller Design . . . 250

List of Changes . . . 252

10 Interfacing with the Fusion Analog System: Processor/Microcontroller Interface . . . 253
Objective . . . 253
CoreAI . . . 253
Clocking Scheme . . . 256
Analog Configuration MUX Initialization . . . 258
ADC Configuration and Calibration . . . 261
Implementing Voltage Monitoring Applications . . . 262
Implementing Current Monitor Applications . . . 265
Implementing Temperature Monitor Applications . . . 266
Implementing Gate Driver Applications . . . 267
Design Example . . . 268
Designing with the RTC . . . 272
List of Changes . . . 272

11 Interfacing with the Fusion Analog System: IP Interface . . . 273
Fusion Analog System Soft IP Design . . . 273
System Overview – Interface Components . . . 274
System Operation . . . 276
SmartGen Soft IP Blocks . . . 277
Basic Analog Block Settings . . . 283
Soft IP Implementation Options . . . 285
Analog Configuration MUX (ACM) . . . 290
Sample Code . . . 291
List of Changes . . . 293

12 Temperature, Voltage, and Current Calibration in Fusion FPGAs . . . 295
Introduction . . . 295
General Calibration Concept . . . 296
Calibration Measurements . . . 297
Calibration Solution . . . 299
Performing System-Level Calibration Using Fusion . . . 307
Conclusion . . . 309
Related Documents . . . 309
List of Changes . . . 309

13 I/O Software Control in Low Power Flash Devices . . . 311
Flash FPGAs I/O Support . . . 312
Software-Controlled I/O Attributes . . . 313
Implementing I/Os in Microsemi Software . . . 314
Assigning Technologies and VREF to I/O Banks . . . 324
Conclusion . . . 329
Related Documents . . . 329
List of Changes . . . 330

14 DDR for Microsemi's Low Power Flash Devices . . . 331
Introduction . . . 331
Double Data Rate (DDR) Architecture . . . 331
DDR Support in Flash-Based Devices . . . 332

I/O Cell Architecture . . . 333
Input Support for DDR . . . 335
Output Support for DDR . . . 335
Instantiating DDR Registers . . . 336
Design Example . . . 342
Conclusion . . . 344
List of Changes . . . 345

15 Prototyping With AFS600 for Smaller Devices . . . 347
Prototype Guideline . . . 348
Summary . . . 350

16 Programming Flash Devices . . . 351
Introduction . . . 351
Summary of Programming Support . . . 351
Programming Support in Flash Devices . . . 352
General Flash Programming Information . . . 353
Important Programming Guidelines . . . 359
Related Documents . . . 361
List of Changes . . . 362

17 Security in Low Power Flash Devices . . . 365
Security in Programmable Logic . . . 365
Security Support in Flash-Based Devices . . . 366
Security Architecture . . . 367
Security Features . . . 368
Security in Action . . . 372
FlashROM Security Use Models . . . 375
Generating Programming Files . . . 377
Conclusion . . . 388
Glossary . . . 388
References . . . 388
Related Documents . . . 389
List of Changes . . . 389

18 In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X . . . 391
Introduction . . . 391
ISP Architecture . . . 391
ISP Support in Flash-Based Devices . . . 392
Programming Voltage (VPUMP) and VJTAG . . . 393
Nonvolatile Memory (NVM) Programming Voltage . . . 393
IEEE 1532 (JTAG) Interface . . . 394
Security . . . 394
Security in ARM-Enabled Low Power Flash Devices . . . 395
FlashROM and Programming Files . . . 397
Programming Solution . . . 398
ISP Programming Header Information . . . 399
Board-Level Considerations . . . 401
Conclusion . . . 402
Related Documents . . . 402

List of Changes . . . 403

19 Microprocessor Programming of Microsemi's Low Power Flash Devices . . . 405
Introduction . . . 405
Microprocessor Programming Support in Flash Devices . . . 406
Programming Algorithm . . . 407
Implementation Overview . . . 407
Hardware Requirement . . . 410
Security . . . 410
Conclusion . . . 411
List of Changes . . . 412

20 Boundary Scan in Low Power Flash Devices . . . 413
Boundary Scan . . . 413
TAP Controller State Machine . . . 413
Microsemi's Flash Devices Support the JTAG Feature . . . 414
Boundary Scan Support in Low Power Devices . . . 415
Boundary Scan Opcodes . . . 415
Boundary Scan Chain . . . 415
Board-Level Recommendations . . . 416
Advanced Boundary Scan Register Settings . . . 417
List of Changes . . . 418

21 UJTAG Applications in Microsemi's Low Power Flash Devices . . . 419
Introduction . . . 419
UJTAG Support in Flash-Based Devices . . . 420
UJTAG Macro . . . 421
UJTAG Operation . . . 422
Typical UJTAG Applications . . . 424
Conclusion . . . 428
Related Documents . . . 428
List of Changes . . . 428

22 Fusion Board-Level Design Guidelines . . . 429
Objective . . . 429
Analog and Digital Plane Isolation . . . 429
Other Special Function Pins . . . 434
Application-Specific Recommendations . . . 436
List of Changes . . . 438

23 Power-Up/-Down of Fusion FPGAs . . . 439
Introduction . . . 439
Power Supply Requirements . . . 439
Power-Up/Down Sequence and Transient Current . . . 441
Digital I/O Behavior at Power-Up/Down . . . 441
Power-Up of Fusion FPGA Fabric Using Internal Voltage Regulator . . . 444
Power-Down of FPGA Fabric Using the Internal Voltage Regulator . . . 446
Periodic Power-Up/Down of FPGA Fabric . . . 446
Basic Operation Modes . . . 448
Conclusion . . . 448

List of Changes 449

24 Fusion Solutions, Design Examples, and Reference Designs 451
System Management Applications 451
Other Applications 452
Development System 453

A Fusion Glossary 455

B Summary of Changes 457
History of Revision to Chapters 457

C Product Support 459
Customer Service 459
Customer Technical Support Center 459
Technical Support 459
Website 459
Contacting the Customer Technical Support Center 459
ITAR Technical Support 460

Index 461

Introduction

Contents

This user's guide contains information to help designers understand and use Microsemi's Fusion® mixed signal FPGAs. Each chapter addresses a specific topic. Many of these chapters apply to other Microsemi device families as well. When a feature or description applies only to a specific device family, this is made clear in the text.

Revision History

The revision history for each chapter is listed at the end of the chapter. Most of these chapters were formerly included in device handbooks. Some were originally application notes or information included in device datasheets.

A "Summary of Changes" table at the end of this user's guide lists the chapters that were changed in each revision of the document, with links to the "List of Changes" sections for those chapters.

Related Information

Refer to the *Fusion Mixed Signal FPGAs* datasheet for detailed specifications, timing, and package and pin information. The website page is /www.microsemi.com/soc/products/fusion/default.aspx.

1 – FPGA Array Architecture in Low Power Flash Devices

Device Architecture

Advanced Flash Switch

Unlike SRAM FPGAs, the low power flash devices use a live-at-power-up ISP flash switch as their programming element. Flash cells are distributed throughout the device to provide nonvolatile, reconfigurable programming to connect signal lines to the appropriate VersaTile inputs and outputs. In the flash switch, two transistors share the floating gate, which stores the programming information (Figure 1-1). One is the sensing transistor, which is only used for writing and verification of the floating gate voltage. The other is the switching transistor. The latter is used to connect or separate routing nets, or to configure VersaTile logic. It is also used to erase the floating gate. Dedicated high-performance lines are connected as required using the flash switch for fast, low-skew, global signal distribution throughout the device core. Maximum core utilization is possible for virtually any design. The use of the flash switch technology also removes the possibility of firm errors, which are increasingly common in SRAM-based FPGAs.

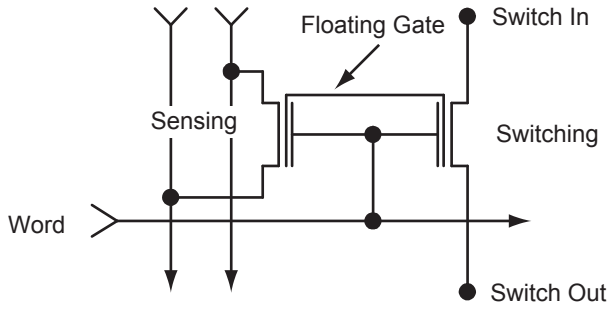


Figure 1-1 • **Flash-Based Switch**

FPGA Array Architecture Support

The flash FPGAs listed in Table 1-1 support the architecture features described in this document.

***Table 1-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO®	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC®3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 1-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

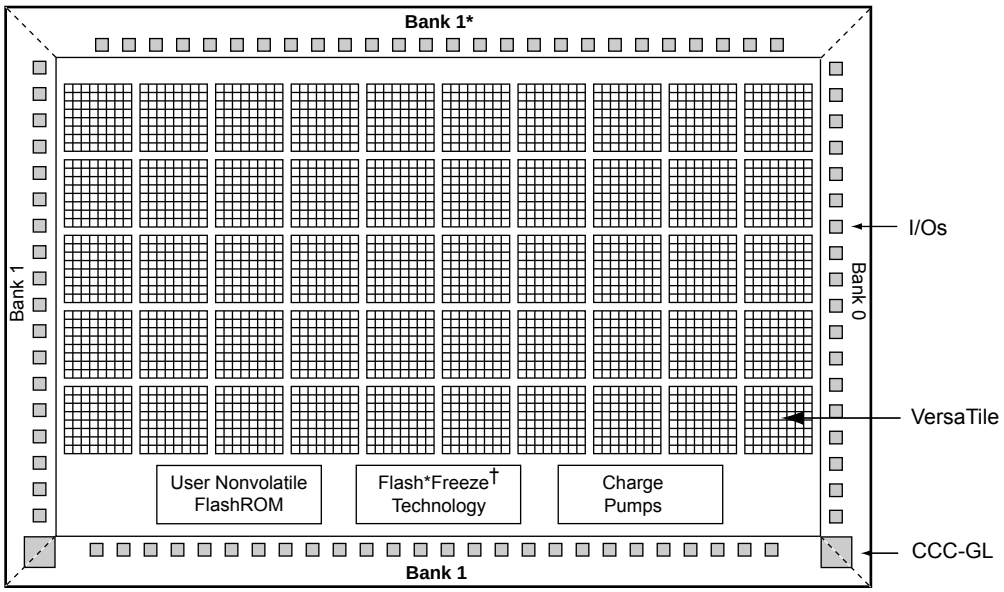
In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 1-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Device Overview

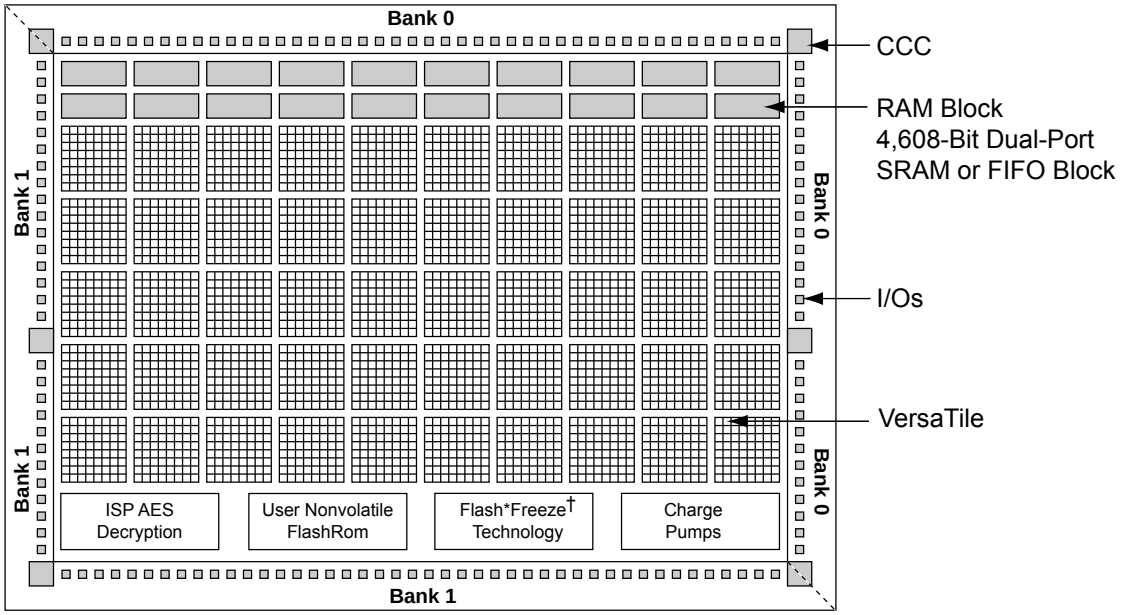
Low power flash devices consist of multiple distinct programmable architectural features (Figure 1-5 on page 15 through Figure 1-7 on page 16):

- FPGA fabric/core (VersaTiles)
- Routing and clock resources (VersaNets)
- FlashROM
- Dedicated SRAM and/or FIFO
  - 30 k gate and smaller device densities do not support SRAM or FIFO.
  - Automotive devices do not support FIFO operation.
- I/O structures
- Flash*Freeze technology and low power modes



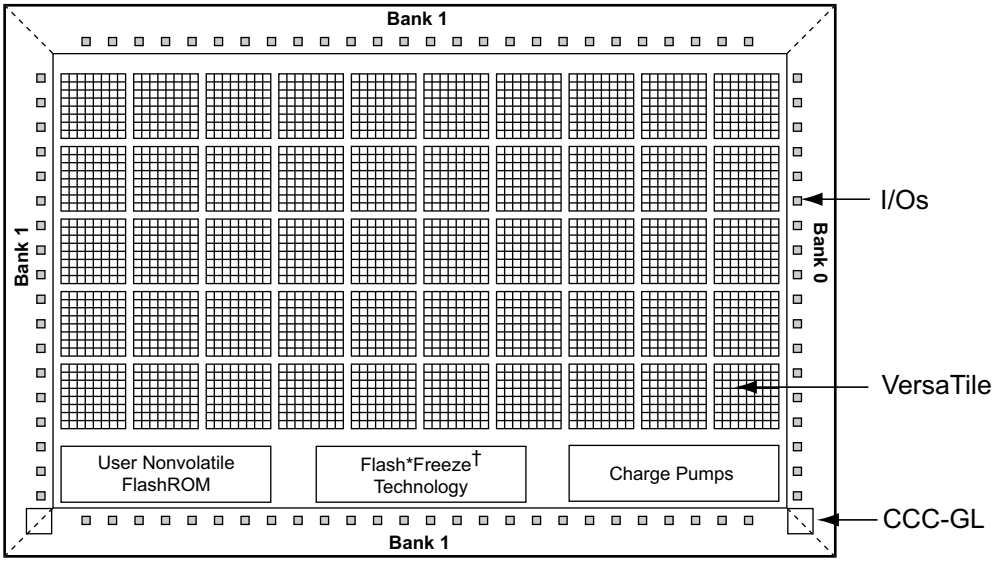
Notes: ** Bank 0 for the 30 k devices*
*† Flash*Freeze mode is supported on IGLOO devices.*

Figure 1-2 • **IGLOO and ProASIC3 nano Device Architecture Overview with Two I/O Banks (applies to 10 k and 30 k device densities, excluding IGLOO PLUS devices)**



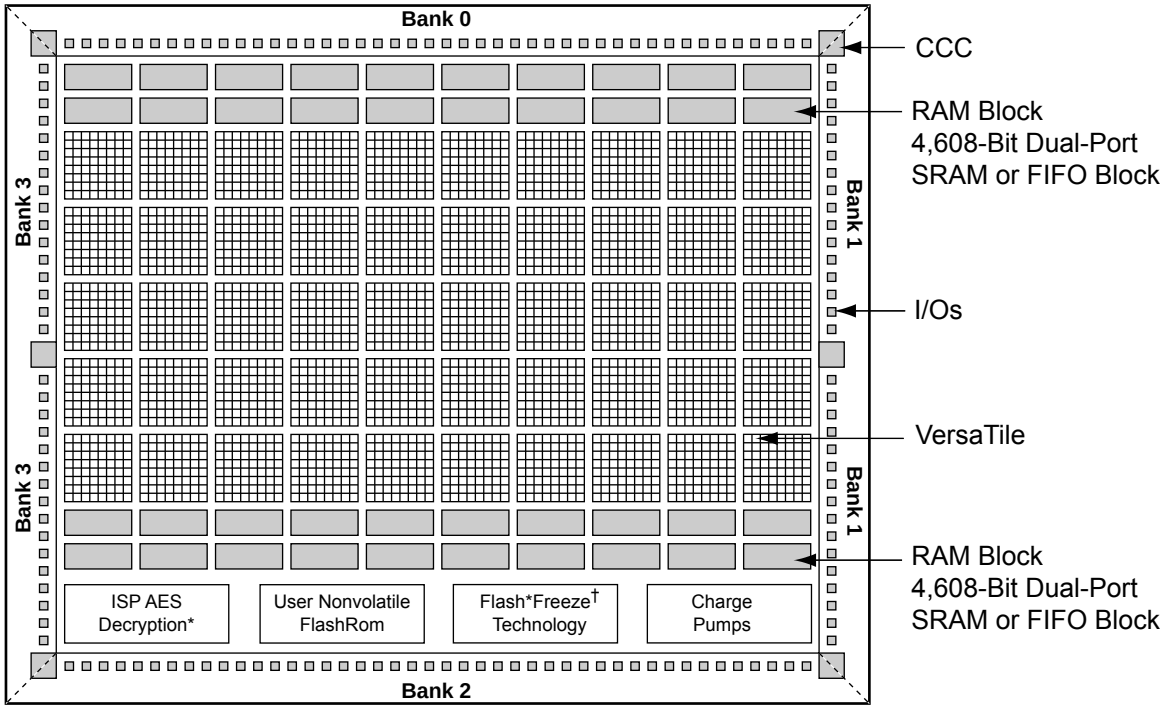
Note: † *Flash*Freeze mode is supported on IGLOO devices.*

Figure 1-3 • **IGLOO Device Architecture Overview with Two I/O Banks with RAM and PLL (60 k and 125 k gate densities)**



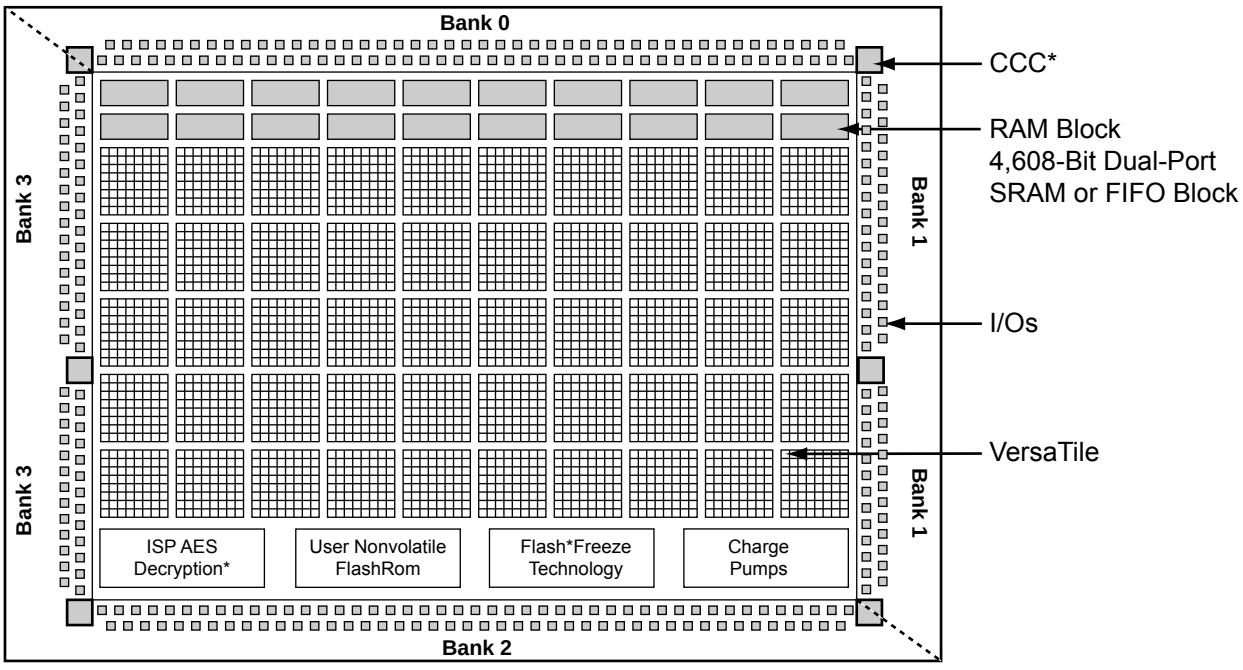
Note: † *Flash*Freeze mode is supported on IGLOO devices.*

Figure 1-4 • **IGLOO Device Architecture Overview with Three I/O Banks (AGLN015, AGLN020, A3PN015, and A3PN020)**



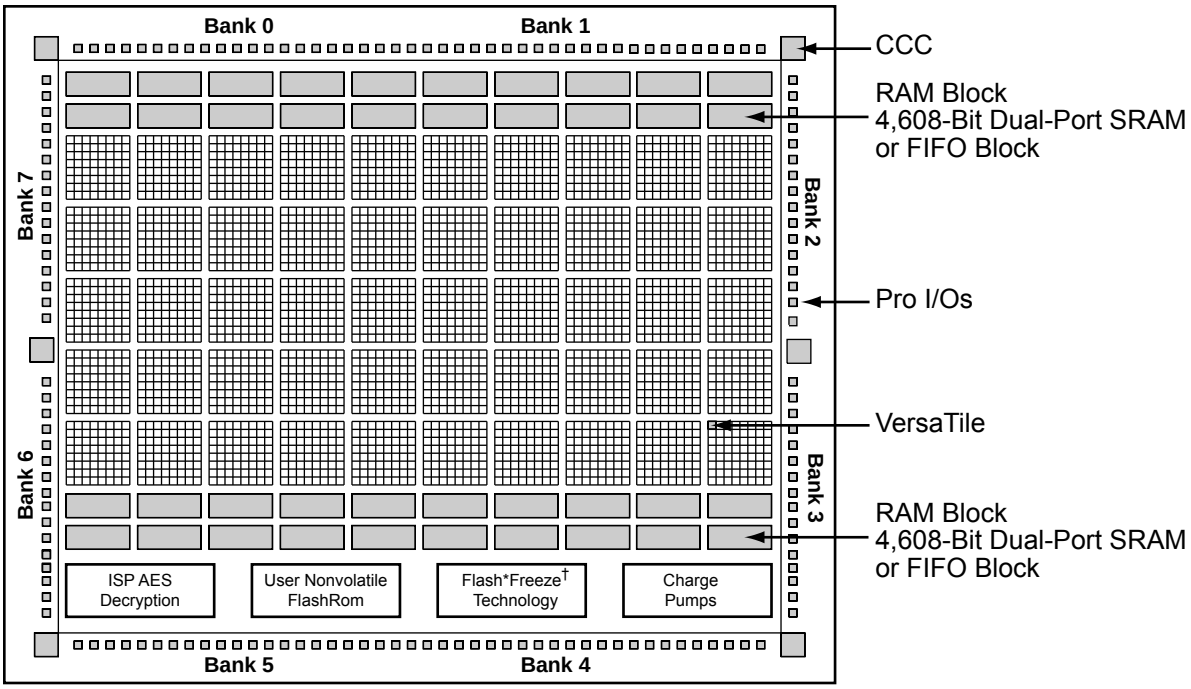
*Note: Flash*Freeze technology only applies to IGLOO and ProASIC3L families.*

Figure 1-5 • **IGLOO, IGLOO nano, ProASIC3 nano, and ProASIC3/L Device Architecture Overview with Four I/O Banks (AGL600 device is shown)**



*Note: * AGLP030 does not contain a PLL or support AES security.*

Figure 1-6 • **IGLOO PLUS Device Architecture Overview with Four I/O Banks**



*Note: Flash*Freeze technology only applies to IGLOOe devices.*

Figure 1-7 • **IGLOOe and ProASIC3E Device Architecture Overview (AGLE600 device is shown)**

I/O State of Newly Shipped Devices

Devices are shipped from the factory with a test design in the device. The power-on switch for VCC is OFF by default in this test design, so I/Os are tristated by default. Tristated means the I/O is not actively driven and floats. The exact value cannot be guaranteed when it is floating. Even in simulation software, a tristate value is marked as unknown. Due to process variations and shifts, tristated I/Os may float toward High or Low, depending on the particular device and leakage level.

If there is concern regarding the exact state of unused I/Os, weak pull-up/pull-down should be added to the floating I/Os so their state is controlled and stabilized.

Core Architecture

VersaTile

The proprietary IGLOO and ProASIC3 device architectures provide granularity comparable to gate arrays. The device core consists of a sea-of-VersaTiles architecture.

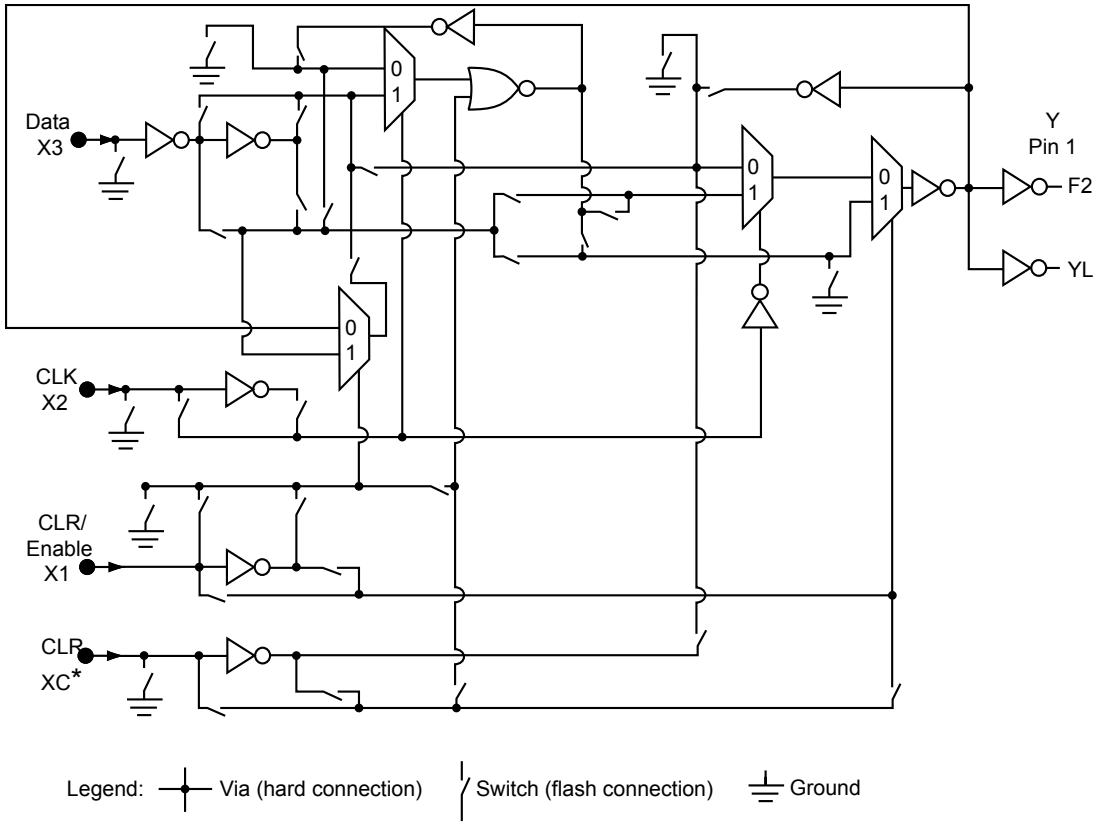
As illustrated in Figure 1-8, there are four inputs in a logic VersaTile cell, and each VersaTile can be configured using the appropriate flash switch connections:

- Any 3-input logic function
- Latch with clear or set
- D-flip-flop with clear or set
- Enable D-flip-flop with clear or set (on a 4th input)

VersaTiles can flexibly map the logic and sequential gates of a design. The inputs of the VersaTile can be inverted (allowing bubble pushing), and the output of the tile can connect to high-speed, very-long-line routing resources. VersaTiles and larger functions can be connected with any of the four levels of routing hierarchy.

When the VersaTile is used as an enable D-flip-flop, SET/CLR is supported by a fourth input. The SET/CLR signal can only be routed to this fourth input over the VersaNet (global) network. However, if, in the user's design, the SET/CLR signal is not routed over the VersaNet network, a compile warning message will be given, and the intended logic function will be implemented by two VersaTiles instead of one.

The output of the VersaTile is F2 when the connection is to the ultra-fast local lines, or YL when the connection is to the efficient long-line or very-long-line resources.



* *This input can only be connected to the global clock distribution network.*

Figure 1-8 • **Low Power Flash Device Core VersaTile**

Array Coordinates

During many place-and-route operations in the Microsemi Designer software tool, it is possible to set constraints that require array coordinates. Table 1-2 provides array coordinates of core cells and memory blocks for IGLOO and ProASIC3 devices. Table 1-3 provides the information for IGLOO PLUS devices. Table 1-4 on page 19 provides the information for IGLOO nano and ProASIC3 nano devices. The array coordinates are measured from the lower left (0, 0). They can be used in region constraints for specific logic groups/blocks, designated by a wildcard, and can contain core cells, memories, and I/Os.

I/O and cell coordinates are used for placement constraints. Two coordinate systems are needed because there is not a one-to-one correspondence between I/O cells and core cells. In addition, the I/O coordinate system changes depending on the die/package combination. It is not listed in Table 1-2. The Designer ChipPlanner tool provides the array coordinates of all I/O locations. I/O and cell coordinates are used for placement constraints. However, I/O placement is easier by package pin assignment.

Figure 1-9 on page 19 illustrates the array coordinates of a 600 k gate device. For more information on how to use array coordinates for region/placement constraints, see the *Designer User's Guide* or online help (available in the software) for software tools.

***Table 1-2* • IGLOO and ProASIC3 Array Coordinates**

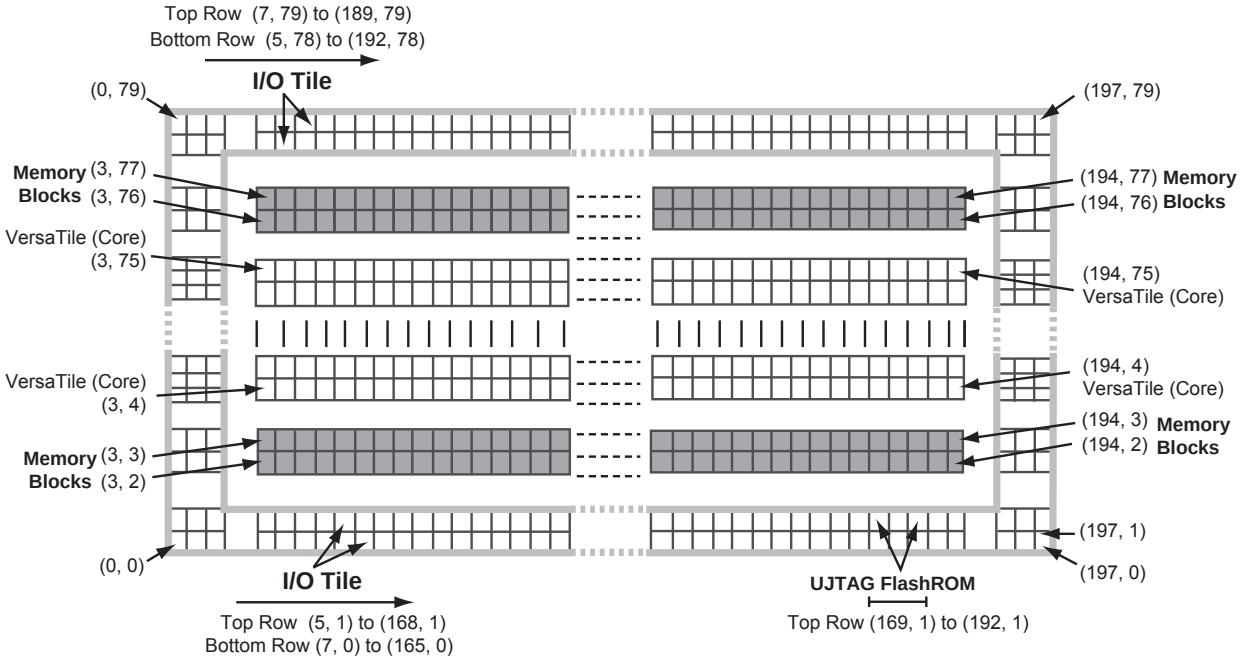
Device		VersaTiles				Memory Rows		Entire Die	
		Min.		Max.		Bottom	Top	Min.	Max.
IGLOO	ProASIC3/ ProASIC3L	x	y	x	y	(x, y)	(x, y)	(x, y)	(x, y)
AGL015	A3P015	3	2	34	13	None	None	(0, 0)	(37, 15)
AGL030	A3P030	3	3	66	13	None	None	(0, 0)	(69, 15)
AGL060	A3P060	3	2	66	25	None	(3, 26)	(0, 0)	(69, 29)
AGL125	A3P125	3	2	130	25	None	(3, 26)	(0, 0)	(133, 29)
AGL250	A3P250/L	3	2	130	49	None	(3, 50)	(0, 0)	(133, 53)
AGL400	A3P400	3	2	194	49	None	(3, 50)	(0, 0)	(197, 53)
AGL600	A3P600/L	3	4	194	75	(3, 2)	(3, 76)	(0, 0)	(197, 79)
AGL1000	A3P1000/L	3	4	258	99	(3, 2)	(3, 100)	(0, 0)	(261, 103)
AGLE600	A3PE600/L, RT3PE600L	3	4	194	75	(3, 2)	(3, 76)	(0, 0)	(197, 79)
	A3PE1500	3	4	322	123	(3, 2)	(3, 124)	(0, 0)	(325, 127)
AGLE3000	A3PE3000/L, RT3PE3000L	3	6	450	173	(3, 2) or (3, 4)	(3, 174) or (3, 176)	(0, 0)	(453, 179)

***Table 1-3* • IGLOO PLUS Array Coordinates**

Device	VersaTiles				Memory Rows		Entire Die	
	Min.		Max.		Bottom	Top	Min.	Max.
IGLOO PLUS	x	y	x	y	(x, y)	(x, y)	(x, y)	(x, y)
AGLP030	2	3	67	13	None	None	(0, 0)	(69, 15)
AGLP060	2	2	67	25	None	(3, 26)	(0, 0)	(69, 29)
AGLP125	2	2	131	25	None	(3, 26)	(0, 0)	(133, 29)

***Table 1-4* • IGLOO nano and ProASIC3 nano Array Coordinates**

Device		VersaTiles		Memory Rows		Entire Die	
		Min.	Max.	Bottom	Top	Min.	Max.
IGLOO nano	**ProASIC3 nano**	**(x, y)**	**(x, y)**	**(x, y)**	**(x, y)**	**(x, y)**	**(x, y)**
AGLN010	A3P010	(0, 2)	(32, 5)	None	None	(0, 0)	(34, 5)
AGLN015	A3PN015	(0, 2)	(32, 9)	None	None	(0, 0)	(34, 9)
AGLN020	A3PN020	(0, 2)	32, 13)	None	None	(0, 0)	(34, 13)
AGLN060	A3PN060	(3, 2)	(66, 25)	None	(3, 26)	(0, 0)	(69, 29)
AGLN125	A3PN125	(3, 2)	(130, 25)	None	(3, 26)	(0, 0)	(133, 29)
AGLN250	A3PN250	(3, 2)	(130, 49)	None	(3, 50)	(0, 0)	(133, 49)



Note: The vertical I/O tile coordinates are not shown. West-side coordinates are {(0, 2) to (2, 2)} to {(0, 77) to (2, 77)}; east-side coordinates are {(195, 2) to (197, 2)} to {(195, 77) to (197, 77)}.

***Figure 1-9 •* Array Coordinates for AGL600, AGLE600, A3P600, and A3PE600**

Routing Architecture

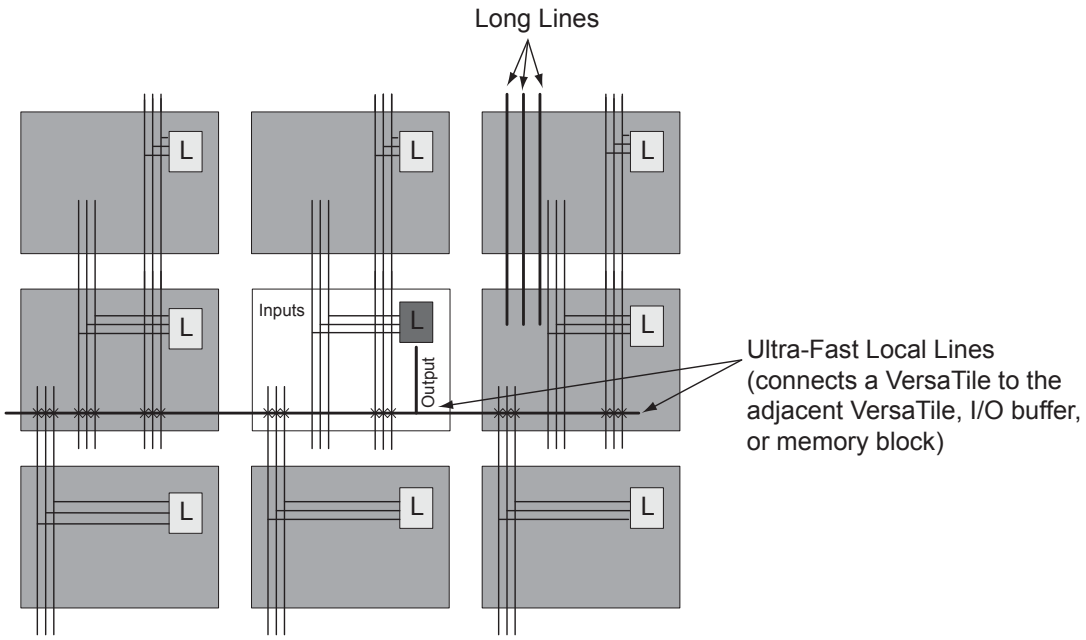
The routing structure of low power flash devices is designed to provide high performance through a flexible four-level hierarchy of routing resources: ultra-fast local resources; efficient long-line resources; high-speed, very-long-line resources; and the high-performance VersaNet networks.

The ultra-fast local resources are dedicated lines that allow the output of each VersaTile to connect directly to every input of the eight surrounding VersaTiles (Figure 1-10). The exception to this is that the SET/CLR input of a VersaTile configured as a D-flip-flop is driven only by the VersaTile global network.

The efficient long-line resources provide routing for longer distances and higher-fanout connections. These resources vary in length (spanning one, two, or four VersaTiles), run both vertically and horizontally, and cover the entire device (Figure 1-11 on page 21). Each VersaTile can drive signals onto the efficient long-line resources, which can access every input of every VersaTile. Routing software automatically inserts active buffers to limit loading effects.

The high-speed, very-long-line resources, which span the entire device with minimal delay, are used to route very long or high-fanout nets: length ±12 VersaTiles in the vertical direction and length ±16 in the horizontal direction from a given core VersaTile (Figure 1-12 on page 21). Very long lines in low power flash devices have been enhanced over those in previous ProASIC families. This provides a significant performance boost for long-reach signals.

The high-performance VersaNet global networks are low-skew, high-fanout nets that are accessible from external pins or internal logic. These nets are typically used to distribute clocks, resets, and other high-fanout nets requiring minimum skew. The VersaNet networks are implemented as clock trees, and signals can be introduced at any junction. These can be employed hierarchically, with signals accessing every input of every VersaTile. For more details on VersaNets, refer to the "Global Resources in Low Power Flash Devices" section on page 23.



Note: Input to the core cell for the D-flip-flop set and reset is only available via the VersaNet global network connection.

Figure 1-10 • **Ultra-Fast Local Lines Connected to the Eight Nearest Neighbors**

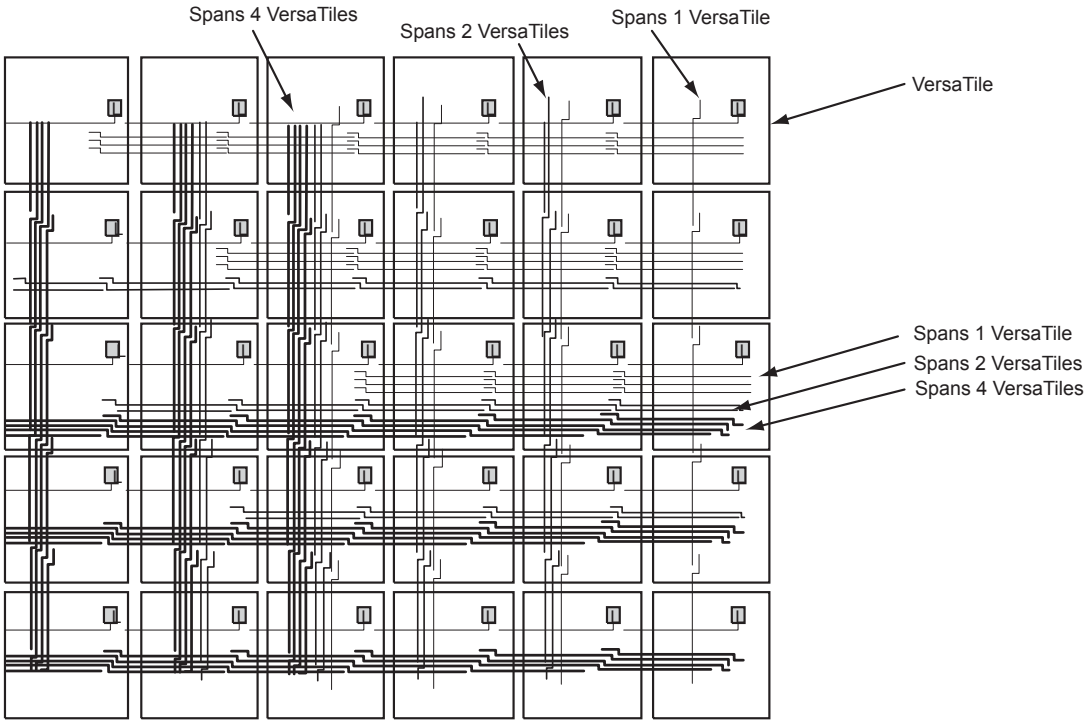


***Figure 1-11* • Efficient Long-Line Resources**

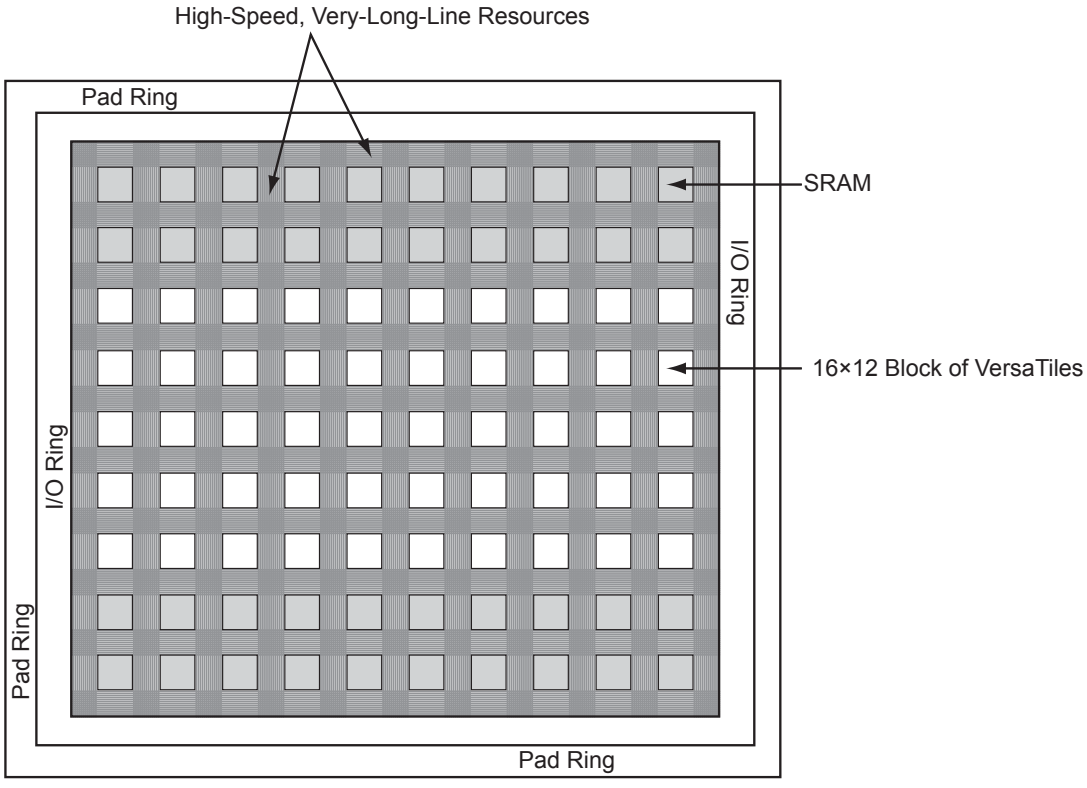


***Figure 1-12* • Very-Long-Line Resources**

Related Documents

User's Guides

Designer User's Guide

http://www.microsemi.com/soc/documents/designer_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
August 2012	The "I/O State of Newly Shipped Devices" section is new (SAR 39542).	16
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 1-1 • Flash-Based FPGAs.	12
	Figure 1-2 • IGLOO and ProASIC3 nano Device Architecture Overview with Two I/O Banks (applies to 10 k and 30 k device densities, excluding IGLOO PLUS devices) through Figure 1-5 • IGLOO, IGLOO nano, ProASIC3 nano, and ProASIC3/L Device Architecture Overview with Four I/O Banks (AGL600 device is shown) are new.	13, 14
	Table 1-4 • IGLOO nano and ProASIC3 nano Array Coordinates is new.	19
v1.3 (October 2008)	The title of this document was changed from "Core Architecture of IGLOO and ProASIC3 Devices" to "FPGA Array Architecture in Low Power Flash Devices."	11
	The "FPGA Array Architecture Support" section was revised to include new families and make the information more concise.	12
	Table 1-2 • IGLOO and ProASIC3 Array Coordinates was updated to include Military ProASIC3/EL and RT ProASIC3 devices.	18
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 1-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	12
v1.1 (March 2008)	Table 1-1 • Flash-Based FPGAs and the accompanying text was updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "Device Overview" section are new.	12
	The "Device Overview" section was updated to note that 15 k devices do not support SRAM or FIFO.	13
	Figure 1-6 • IGLOO PLUS Device Architecture Overview with Four I/O Banks is new.	15
	Table 1-2 • IGLOO and ProASIC3 Array Coordinates was updated to add A3P015 and AGL015.	18
	Table 1-3 • IGLOO PLUS Array Coordinates is new.	18

2 – Global Resources in Low Power Flash Devices

Introduction

IGLOO, Fusion, and ProASIC3 FPGA devices offer a powerful, low-delay VersaNet global network scheme and have extensive support for multiple clock domains. In addition to the Clock Conditioning Circuits (CCCs) and phase-locked loops (PLLs), there is a comprehensive global clock distribution network called a VersaNet global network. Each logical element (VersaTile) input and output port has access to these global networks. The VersaNet global networks can be used to distribute low-skew clock signals or high-fanout nets. In addition, these highly segmented VersaNet global networks contain spines (the vertical branches of the global network tree) and ribs that can reach all the VersaTiles inside their region. This allows users the flexibility to create low-skew local clock networks using spines. This document describes VersaNet global networks and discusses how to assign signals to these global networks and spines in a design flow. Details concerning low power flash device PLLs are described in the "Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" section on page 53. This chapter describes the low power flash devices' global architecture and uses of these global networks in designs.

Global Architecture

Low power flash devices offer powerful and flexible control of circuit timing through the use of global circuitry. Each chip has up to six CCCs, some with PLLs.

- In IGLOOe, ProASIC3EL, and ProASIC3E devices, all CCCs have PLLs—hence, 6 PLLs per device (except the PQ208 package, which has only 2 PLLs).
- In IGLOO, IGLOO nano, IGLOO PLUS, ProASIC3, and ProASIC3L devices, the west CCC contains a PLL core (except in 10 k through 30 k devices).
- In Fusion devices, the west CCC also contains a PLL core. In the two larger devices (AFS600 and AFS1500), the west and east CCCs each contain a PLL.

Refer to Table 3-6 on page 76 for details. Each PLL includes delay lines, a phase shifter (0°, 90°, 180°, 270°), and clock multipliers/dividers. Each CCC has all the circuitry needed for the selection and interconnection of inputs to the VersaNet global network. The east and west CCCs each have access to three chip global lines on each side of the chip (six chip global lines total). The CCCs at the four corners each have access to three quadrant global lines in each quadrant of the chip (except in 10 k through 30 k gate devices).

The nano 10 k, 15 k, and 20 k devices support four VersaNet global resources, and 30 k devices support six global resources. The 10 k through 30 k devices have simplified CCCs called CCC-GLs.

The flexible use of the VersaNet global network allows the designer to address several design requirements. User applications that are clock-resource-intensive can easily route external or gated internal clocks using VersaNet global routing networks. Designers can also drastically reduce delay penalties and minimize resource usage by mapping critical, high-fanout nets to the VersaNet global network.

Note: Microsemi recommends that you choose the appropriate global pin and use the appropriate global resource so you can realize these benefits.

The following sections give an overview of the VersaNet global network, the structure of the global network, access point for the global networks, and the clock aggregation feature that enables a design to have very low clock skew using spines.

Global Resource Support in Flash-Based Devices

The flash FPGAs listed in Table 2-1 support the global resources and the functions described in this document.

***Table 2-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
	IGLOO nano	The industry's lowest-power, smallest-size solution
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO products as listed in Table 2-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 2-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

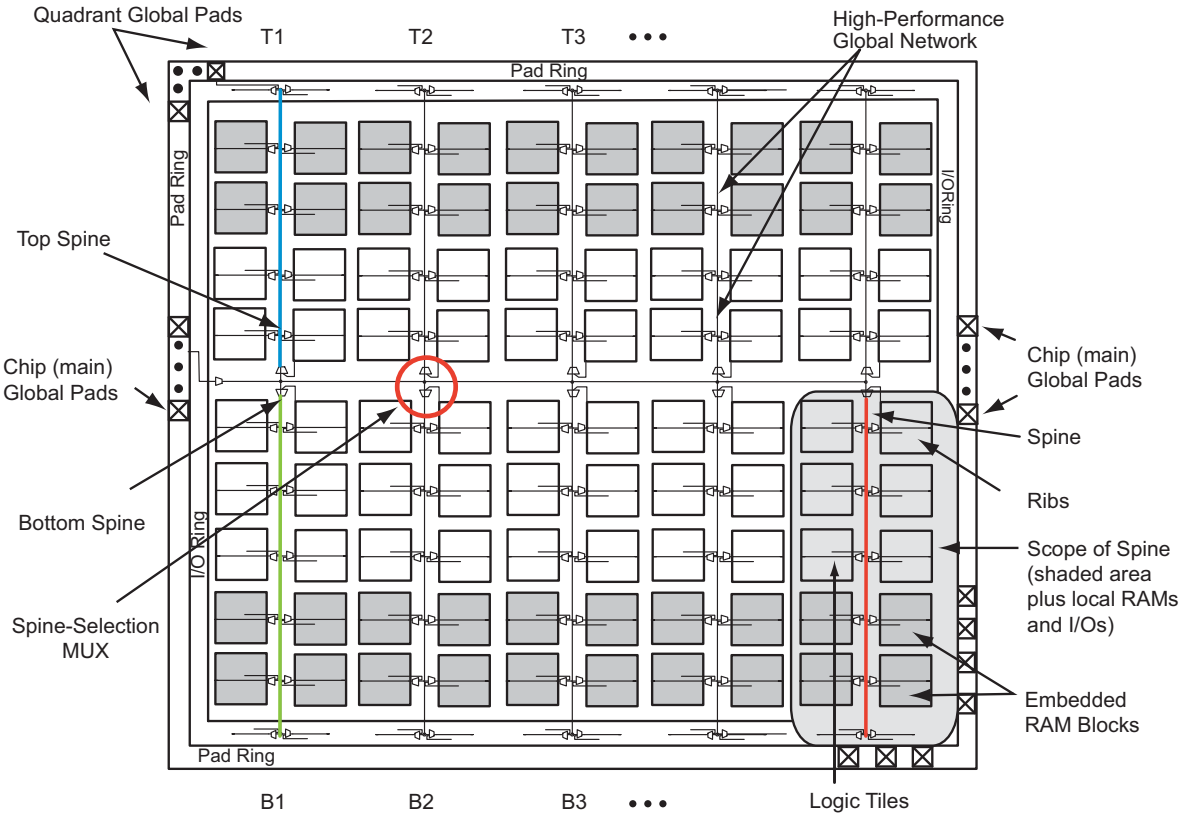
VersaNet Global Network Distribution

One of the architectural benefits of low power flash architecture is the set of powerful, low-delay VersaNet global networks that can access the VersaTiles, SRAM, and I/O tiles of the device. Each device offers a chip global network with six global lines (except for nano 10 k, 15 k, and 20 k gate devices) that are distributed from the center of the FPGA array. In addition, each device (except the 10 k through 30 k gate device) has four quadrant global networks, each consisting of three quadrant global net resources. These quadrant global networks can only drive a signal inside their own quadrant. Each VersaTile has access to nine global line resources—three quadrant and six chip-wide (main) global networks—and a total of 18 globals are available on the device (3 × 4 regional from each quadrant and 6 global).

Figure 2-1 shows an overview of the VersaNet global network and device architecture for devices 60 k and above. Figure 2-2 and Figure 2-3 on page 26 show simplified VersaNet global networks.

The VersaNet global networks are segmented and consist of spines, global ribs, and global multiplexers (MUXes), as shown in Figure 2-1. The global networks are driven from the global rib at the center of the die or quadrant global networks at the north or south side of the die. The global network uses the MUX trees to access the spine, and the spine uses the clock ribs to access the VersaTile. Access is available to the chip or quadrant global networks and the spines through the global MUXes. Access to the spine using the global MUXes is explained in the "Spine Architecture" section on page 33.

These VersaNet global networks offer fast, low-skew routing resources for high-fanout nets, including clock signals. In addition, these highly segmented global networks offer users the flexibility to create low-skew local clock networks using spines for up to 252 internal/external clocks or other high-fanout nets in low power flash devices. Optimal usage of these low-skew networks can result in significant improvement in design performance.



Note: Not applicable to 10 k through 30 k gate devices

***Figure 2-1* • Overview of VersaNet Global Network and Device Architecture**

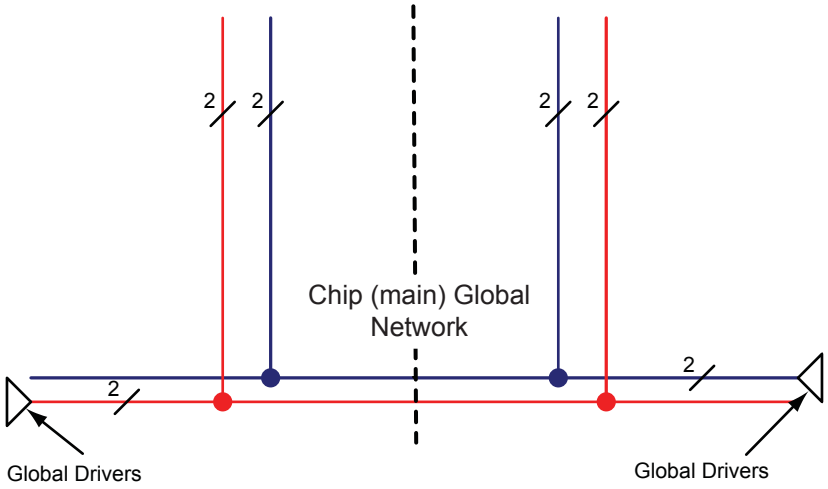


***Figure 2-2* • Simplified VersaNet Global Network (30 k gates and below)**

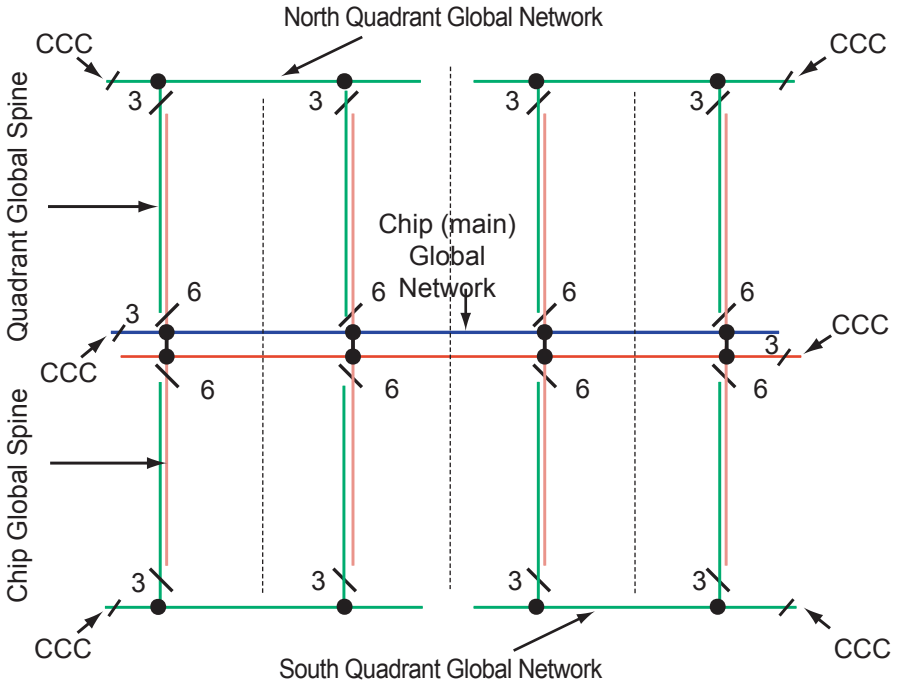


***Figure 2-3* • Simplified VersaNet Global Network (60 k gates and above)**

Chip and Quadrant Global I/Os

The following sections give an overview of naming conventions and other related I/O information.

Naming of Global I/Os

In low power flash devices, the global I/Os have access to certain clock conditioning circuitry and have direct access to the global network. Additionally, the global I/Os can be used as regular I/Os, since they have identical capabilities to those of regular I/Os. Due to the comprehensive and flexible nature of the I/Os in low power flash devices, a naming scheme is used to show the details of the I/O. The global I/O uses the generic name Gmn/IOuxwByVz. Note that Gmn refers to a global input pin and IOuxwByVz refers to a regular I/O Pin, as these I/Os can be used as either global or regular I/Os. Refer to the I/O Structures chapter of the user's guide for the device that you are using for more information on this naming convention.

Figure 2-4 represents the global input pins connection. It shows all 54 global pins available to access the 18 global networks in ProASIC3E families.

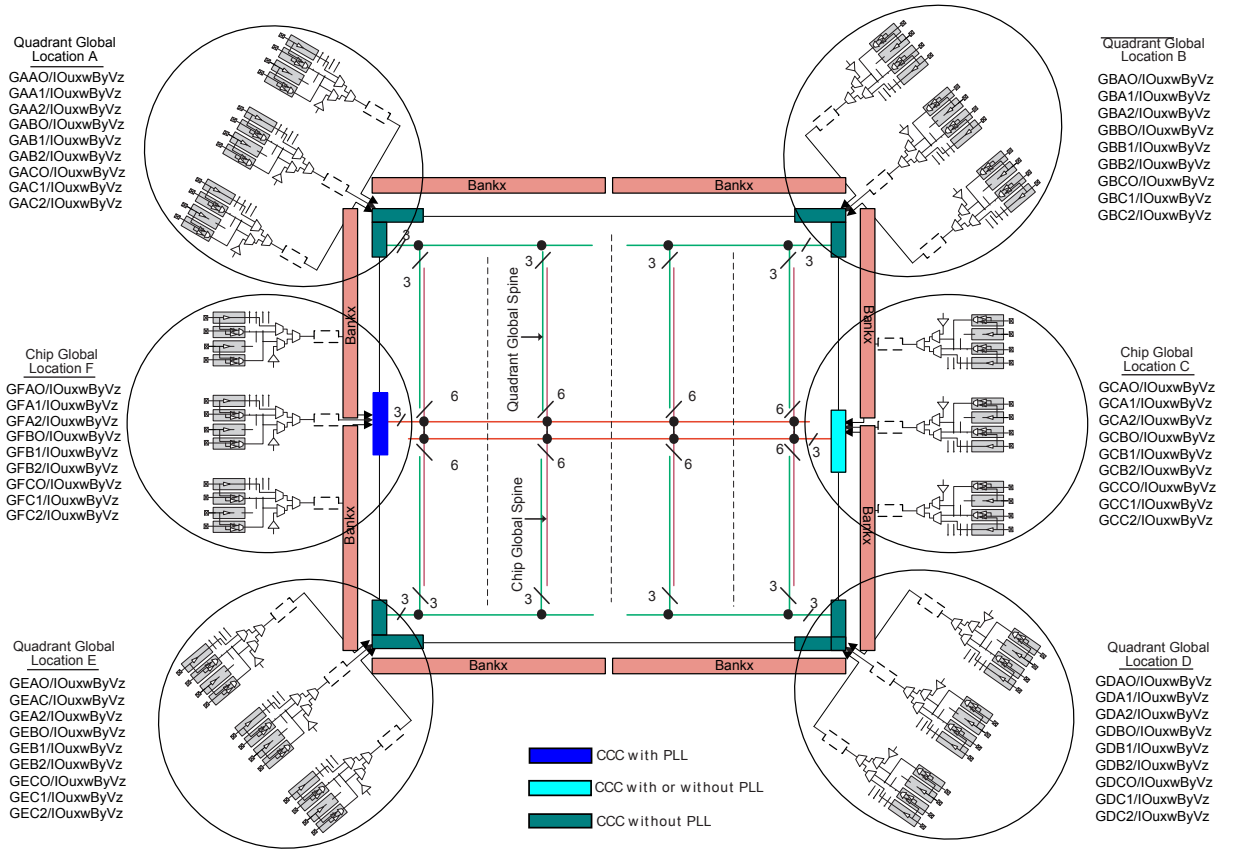
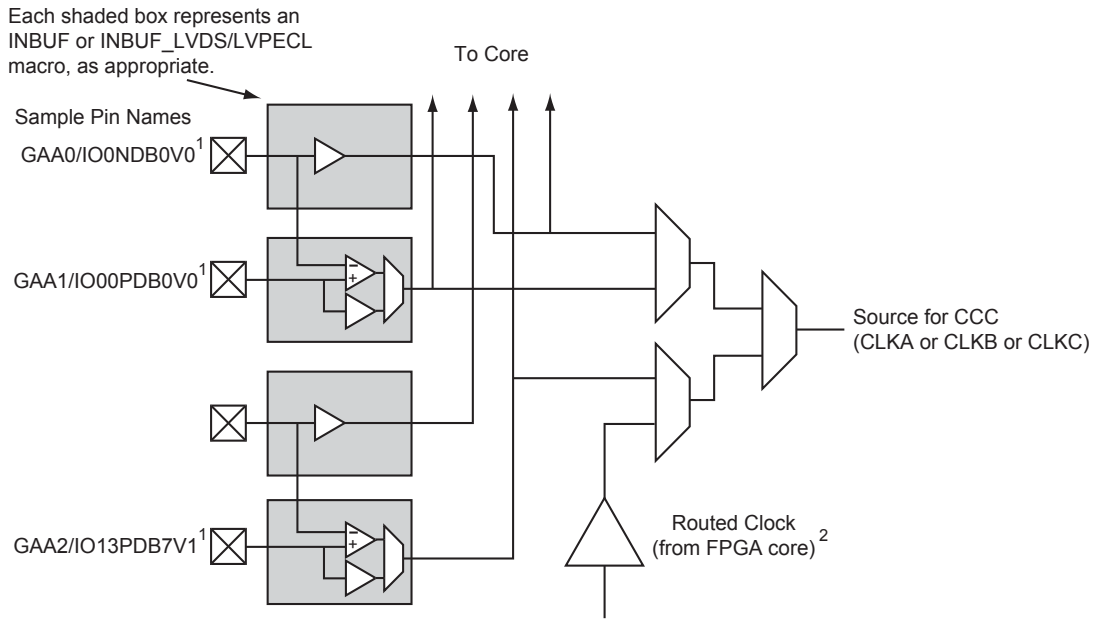


***Figure 2-4* • Global Connections Details**

Figure 2-5 shows more detailed global input connections. It shows the global input pins connection to the northwest quadrant global networks. Each global buffer, as well as the PLL reference clock, can be driven from one of the following:

- 3 dedicated single-ended I/Os using a hardwired connection
- 2 dedicated differential I/Os using a hardwired connection (not supported for IGLOO nano or ProASIC3 nano devices)
- The FPGA core



Note: Differential inputs are not supported for IGLOO nano or ProASIC3 nano devices.

***Figure 2-5* • Global I/O Overview**

Figure 2-6 shows all nine global inputs for the location A connected to the top left quadrant global network via CCC.

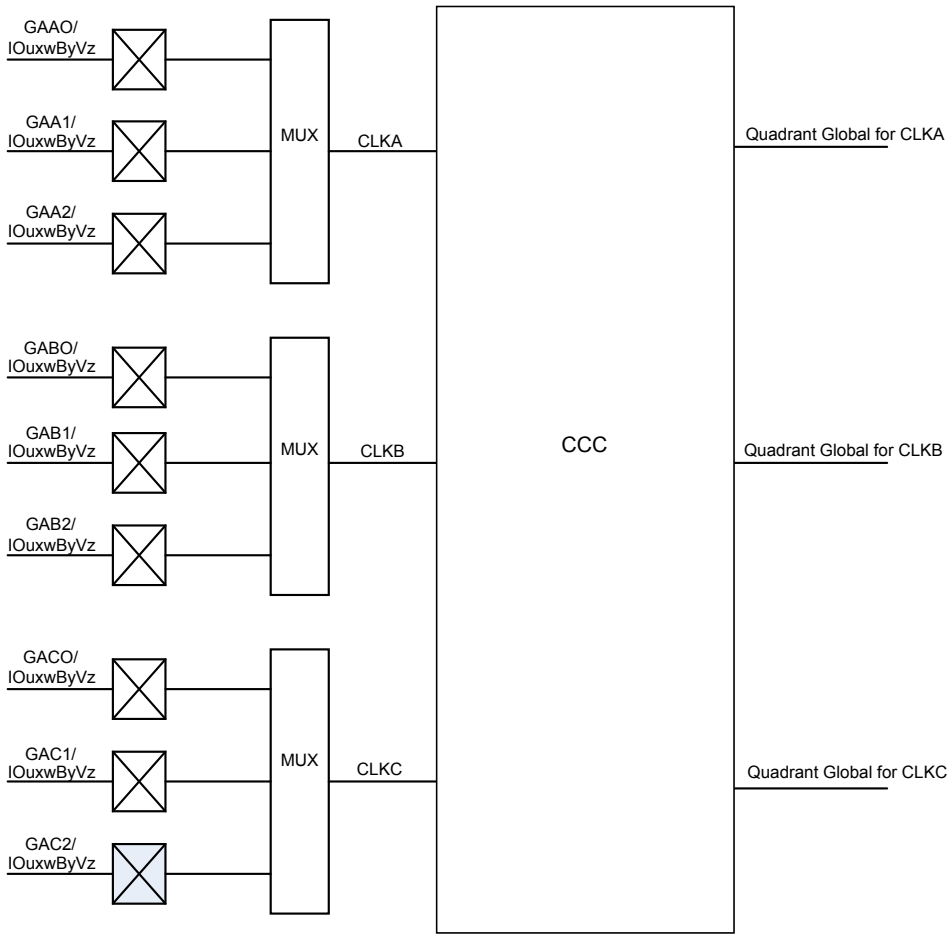


***Figure 2-6* • Global Inputs**

Since each bank can have a different I/O standard, the user should be careful to choose the correct global I/O for the design. There are 54 global pins available to access 18 global networks. For the single-ended and voltage-referenced I/O standards, you can use any of these three available I/Os to access the global network. For differential I/O standards such as LVDS and LVPECL, the I/O macro needs to be placed on (A0, A1), (B0, B1), (C0, C1), or a similar location. The unassigned global I/Os can be used as regular I/Os. Note that pin names starting with GF and GC are associated with the chip global networks, and GA, GB, GD, and GE are used for quadrant global networks. Table 2-2 on page 30 and Table 2-3 on page 31 show the general chip and quadrant global pin names.

***Table 2-2* • Chip Global Pin Name**

I/O Type	Beginning of I/O Name	Notes
Single-Ended	GFAO/IOuxwByVz GFA1/IOuxwByVz GFA2/IOuxwByVz	Only one of the I/Os can be directly connected to a chip global at a time.
	GFBO/IOuxwByVz GFB1/IOuxwByVz GFB2/IOuxwByVz	Only one of the I/Os can be directly connected to a chip global at a time.
	GFC0/IOuxwByVz GFC1/IOuxwByVz GFC2/IOuxwByVz	Only one of the I/Os can be directly connected to a chip global at a time.
	GCAO/IOuxwByVz GCA1/IOuxwByVz GCA2/IOuxwByVz	Only one of the I/Os can be directly connected to a chip global at a time.
	GCBO/IOuxwByVz GCB1/IOuxwByVz GCB2/IOuxwByVz	Only one of the I/Os can be directly connected to a chip global at a time.
	GCC0/IOuxwByVz GCC1/IOuxwByVz GCC2/IOuxwByVz	Only one of the I/Os can be directly connected to a chip global at a time.
Differential I/O Pairs	GFAO/IOuxwByVz GFA1/IOuxwByVz	The output of the different pair will drive the chip global.
	GFBO/IOuxwByVz GFB1/IOuxwByVz	The output of the different pair will drive the chip global.
	GFCO/IOuxwByVz GFC1/IOuxwByVz	The output of the different pair will drive the chip global.
	GCAO/IOuxwByVz GCA1/IOuxwByVz	The output of the different pair will drive the chip global.
	GCBO/IOuxwByVz GCB1/IOuxwByVz	The output of the different pair will drive the chip global.
	GCCO/IOuxwByVz GCC1/IOuxwByVz	The output of the different pair will drive the chip global.

Note: Only one of the I/Os can be directly connected to a quadrant at a time.

***Table 2-3* • Quadrant Global Pin Name**

I/O Type	Beginning of I/O Name	Notes
Single-Ended	GAAO/IOuxwByVz GAA1/IOuxwByVz GAA2/IOuxwByVz	Only one of the I/Os can be directly connected to a quadrant global at a time
	GABO/IOuxwByVz GAB1/IOuxwByVz GAB2/IOuxwByVz	Only one of the I/Os can be directly connected to a quadrant global at a time.
	GAC0/IOuxwByVz GAC1/IOuxwByVz GAC2/IOuxwByVz	Only one of the I/Os can be directly connected to a quadrant global at a time.
	GBAO/IOuxwByVz GBA1/IOuxwByVz GBA2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GBBO/IOuxwByVz GBB1/IOuxwByVz GBB2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GBC0/IOuxwByVz GBC1/IOuxwByVz GBC2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GDAO/IOuxwByVz GDA1/IOuxwByVz GDA2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GDBO/IOuxwByVz GDB1/IOuxwByVz GDB2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GDC0/IOuxwByVz GDC1/IOuxwByVz GDC2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GEAO/IOuxwByVz GEA1/IOuxwByVz GEA2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GEBO/IOuxwByVz GEB1/IOuxwByVz GEB2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.
	GEC0/IOuxwByVz GEC1/IOuxwByVz GEC2/IOuxwByVz	Only one of the I/Os can be directly connected to a global at a time.

Note: Only one of the I/Os can be directly connected to a quadrant at a time.

***Table 2-3* • Quadrant Global Pin Name (continued)**

Differential I/O Pairs	GAAO/IOuxwByVz GAA1/IOuxwByVz	The output of the different pair will drive the global.
	GABO/IOuxwByVz GAB1/IOuxwByVz	The output of the different pair will drive the global.
	GACO/IOuxwByVz GAC1/IOuxwByVz	The output of the different pair will drive the global.
	GBAO/IOuxwByVz GBA1/IOuxwByVz	The output of the different pair will drive the global.
	GBBO/IOuxwByVz GBB1/IOuxwByVz	The output of the different pair will drive the global.
	GBCO/IOuxwByVz GBC1/IOuxwByVz	The output of the different pair will drive the global.
	GDAO/IOuxwByVz GDA1/IOuxwByVz	The output of the different pair will drive the global.
	GDBO/IOuxwByVz GDB1/IOuxwByVz	The output of the different pair will drive the global.
	GDCO/IOuxwByVz GDC1/IOuxwByVz	The output of the different pair will drive the global.
	GEAO/IOuxwByVz GEA1/IOuxwByVz	The output of the different pair will drive the global.
	GEBO/IOuxwByVz GEB1/IOuxwByVz	The output of the different pair will drive the global.
	GECO/IOuxwByVz GEC1/IOuxwByVz	The output of the different pair will drive the global.

Note: Only one of the I/Os can be directly connected to a quadrant at a time.

Unused Global I/O Configuration

The unused clock inputs behave similarly to the unused Pro I/Os. The Microsemi Designer software automatically configures the unused global pins as inputs with pull-up resistors if they are not used as regular I/O.

I/O Banks and Global I/O Standards

In low power flash devices, any I/O or internal logic can be used to drive the global network. However, only the global macro placed at the global pins will use the hardwired connection between the I/O and global network. Global signal (signal driving a global macro) assignment to I/O banks is no different from regular I/O assignment to I/O banks with the exception that you are limited to the pin placement location available. Only global signals compatible with both the VCCI and VREF standards can be assigned to the same bank.

Spine Architecture

The low power flash device architecture allows the VersaNet global networks to be segmented. Each of these networks contains spines (the vertical branches of the global network tree) and ribs that can reach all the VersaTiles inside its region. The nine spines available in a vertical column reside in global networks with two separate regions of scope: the quadrant global network, which has three spines, and the chip (main) global network, which has six spines. Note that the number of quadrant globals and globals/spines per tree varies depending on the specific device. Refer to Table 2-4 for the clocking resources available for each device. The spines are the vertical branches of the global network tree, shown in Figure 2-3 on page 26. Each spine in a vertical column of a chip (main) global network is further divided into two spine segments of equal lengths: one in the top and one in the bottom half of the die (except in 10 k through 30 k gate devices).

Top and bottom spine segments radiating from the center of a device have the same height. However, just as in the ProASIC^{PLUS}® family, signals assigned only to the top and bottom spine cannot access the middle two rows of the die. The spines for quadrant clock networks do not cross the middle of the die and cannot access the middle two rows of the architecture.

Each spine and its associated ribs cover a certain area of the device (the "scope" of the spine; see Figure 2-3 on page 26). Each spine is accessed by the dedicated global network MUX tree architecture, which defines how a particular spine is driven—either by the signal on the global network from a CCC, for example, or by another net defined by the user. Details of the chip (main) global network spine-selection MUX are presented in Figure 2-8 on page 36. The spine drivers for each spine are located in the middle of the die.

Quadrant spines can be driven from user I/Os or an internal signal from the north and south sides of the die. The ability to drive spines in the quadrant global networks can have a significant effect on system performance for high-fanout inputs to a design. Access to the top quadrant spine regions is from the top of the die, and access to the bottom quadrant spine regions is from the bottom of the die. The A3PE3000 device has 28 clock trees and each tree has nine spines; this flexible global network architecture enables users to map up to 252 different internal/external clocks in an A3PE3000 device.

***Table 2-4* • Globals/Spines/Rows for IGLOO and ProASIC3 Devices**

ProASIC3/ ProASIC3L Devices	IGLOO Devices	Chip Globals	Quadrant Globals (4×3)	Clock Trees	Globals/ Spines per Tree	Total Spines per Device	VersaTiles in Each Tree	Total VersaTiles	Rows in Each Spine
A3PN010	AGLN010	4	0	1	0	0	260	260	4
A3PN015	AGLN015	4	0	1	0	0	384	384	6
A3PN020	AGLN020	4	0	1	0	0	520	520	6
A3PN060	AGLN060	6	12	4	9	36	384	1,536	12
A3PN125	AGLN125	6	12	8	9	72	384	3,072	12
A3PN250	AGLN250	6	12	8	9	72	768	6,144	24
A3P015	AGL015	6	0	1	9	9	384	384	12
A3P030	AGL030	6	0	2	9	18	384	768	12
A3P060	AGL060	6	12	4	9	36	384	1,536	12
A3P125	AGL125	6	12	8	9	72	384	3,072	12
A3P250/L	AGL250	6	12	8	9	72	768	6,144	24
A3P400	AGL400	6	12	12	9	108	768	9,216	24
A3P600/L	AGL600	6	12	12	9	108	1,152	13,824	36
A3P1000/L	AGL1000	6	12	16	9	144	1,536	24,576	48
A3PE600/L	AGLE600	6	12	12	9	108	1,120	13,440	35
A3PE1500		6	12	20	9	180	1,888	37,760	59
A3PE3000/L	AGLE3000	6	12	28	9	252	2,656	74,368	83

***Table 2-5* • Globals/Spines/Rows for IGLOO PLUS Devices**

IGLOO PLUS Devices	Chip Globals	Quadrant Globals (4×3)	Clock Trees	Globals/ Spines per Tree	Total Spines per Device	VersaTiles in Each Tree	Total VersaTiles	Rows in Each Spine
AGLP030	6	0	2	9	18	384*	792	12
AGLP060	6	12	4	9	36	384*	1,584	12
AGLP125	6	12	8	9	72	384*	3,120	12

*Note: *Clock trees that are located at far left and far right will support more VersaTiles.*

***Table 2-6* • Globals/Spines/Rows for Fusion Devices**

Fusion Device	Chip Globals	Quadrant Globals (4×3)	Clock Trees	Globals/ Spines per Tree	Total Spines per Device	VersaTiles in Each Tree	Total VersaTiles	Rows in Each Spine
AFS090	6	12	6	9	54	384	2,304	12
AFS250	6	12	8	9	72	768	6,144	24
AFS600	6	12	12	9	108	1,152	13,824	36
AFS1500	6	12	20	9	180	1,920	38,400	60

Spine Access

The physical location of each spine is identified by the letter T (top) or B (bottom) and an accompanying number (T*n* or B*n*). The number *n* indicates the horizontal location of the spine; 1 refers to the first spine on the left side of the die. Since there are six chip spines in each spine tree, there are up to six spines available for each combination of T (or B) and *n* (for example, six T1 spines). Similarly, there are three quadrant spines available for each combination of T (or B) and *n* (for example, four T1 spines), as shown in Figure 2-7.

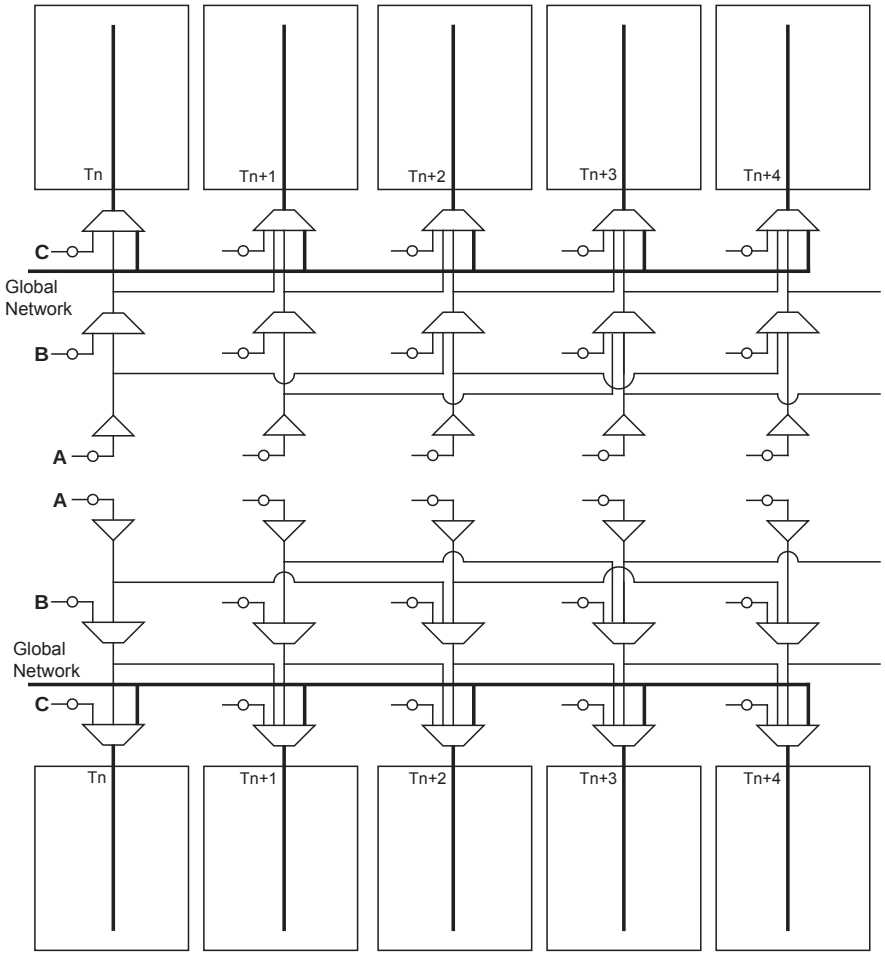


***Figure 2-7* • Chip Global Aggregation**

A spine is also called a local clock network, and is accessed by the dedicated global MUX architecture. These MUXes define how a particular spine is driven. Refer to Figure 2-8 on page 36 for the global MUX architecture. The MUXes for each chip global spine are located in the middle of the die. Access to the top and bottom chip global spine is available from the middle of the die. There is no control dependency between the top and bottom spines. If a top spine, T1, of a chip global network is assigned to a net, B1 is not wasted and can be used by the global clock network. The signal assigned only to the top or bottom spine cannot access the middle two rows of the architecture. However, if a spine is using the top and bottom at the same time (T1 and B1, for instance), the previous restriction is lifted.

The MUXes for each quadrant global spine are located in the north and south sides of the die. Access to the top and bottom quadrant global spines is available from the north and south sides of the die. Since the MUXes for quadrant spines are located in the north and south sides of the die, you should not try to drive T1 and B1 quadrant spines from the same signal.

Using Clock Aggregation

Clock aggregation allows for multi-spine clock domains to be assigned using hardwired connections, without adding any extra skew. A MUX tree, shown in Figure 2-8, provides the necessary flexibility to allow long lines, local resources, or I/Os to access domains of one, two, or four global spines. Signal access to the clock aggregation system is achieved through long-line resources in the central rib in the center of the die, and also through local resources in the north and south ribs, allowing I/Os to feed directly into the clock system. As Figure 2-9 indicates, this access system is contiguous.

There is no break in the middle of the chip for the north and south I/O VersaNet access. This is different from the quadrant clocks located in these ribs, which only reach the middle of the rib.

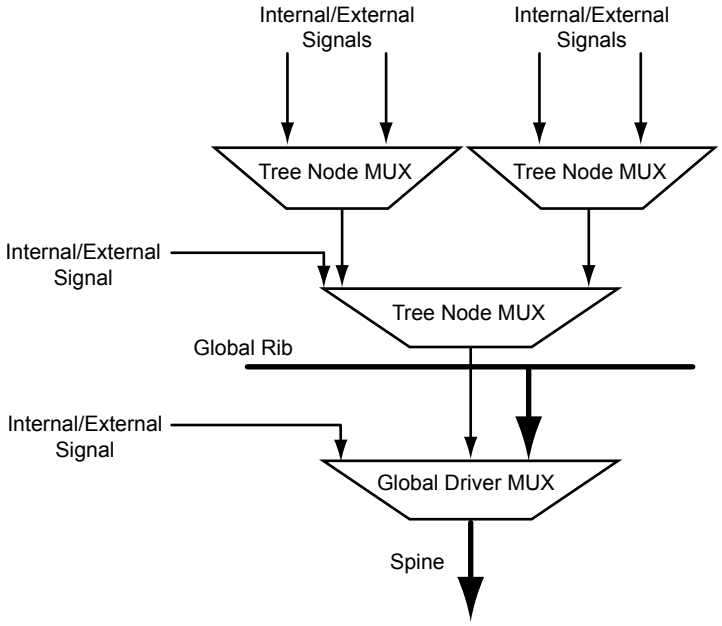


***Figure 2-8* • Spine Selection MUX of Global Tree**

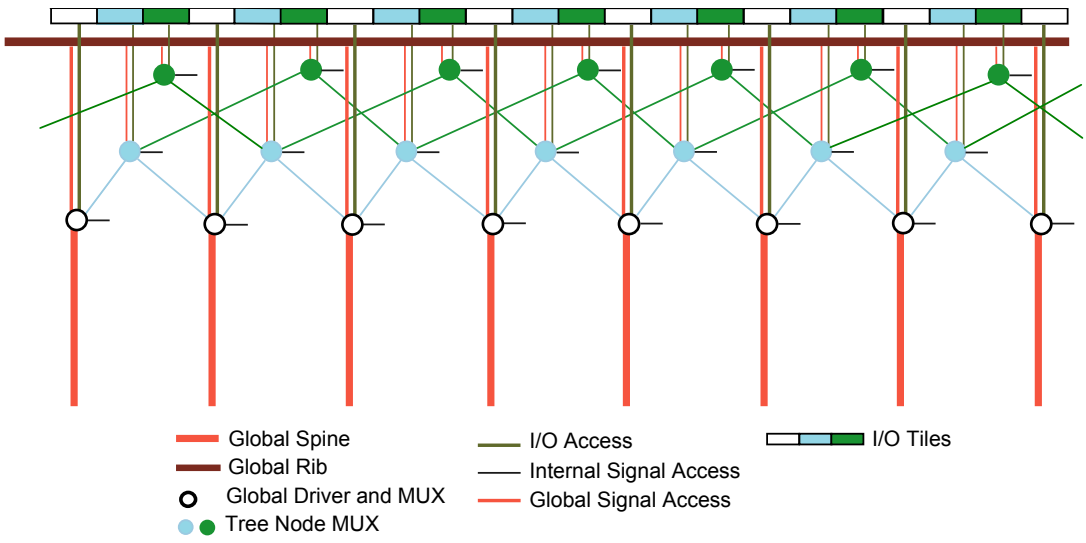


***Figure 2-9* • Clock Aggregation Tree Architecture**

Clock Aggregation Architecture

This clock aggregation feature allows a balanced clock tree, which improves clock skew. The physical regions for clock aggregation are defined from left to right and shift by one spine. For chip global networks, there are three types of clock aggregation available, as shown in Figure 2-10:

- Long lines that can drive up to four adjacent spines (A)
- Long lines that can drive up to two adjacent spines (B)
- Long lines that can drive one spine (C)

There are three types of clock aggregation available for the quadrant spines, as shown in Figure 2-10:

- I/Os or local resources that can drive up to four adjacent spines
- I/Os or local resources that can drive up to two adjacent spines
- I/Os or local resources that can drive one spine

As an example, A3PE600 and AFS600 devices have twelve spine locations: T1, T2, T3, T4, T5, T6, B1, B2, B3, B4, B5, and B6. Table 2-7 shows the clock aggregation you can have in A3PE600 and AFS600.

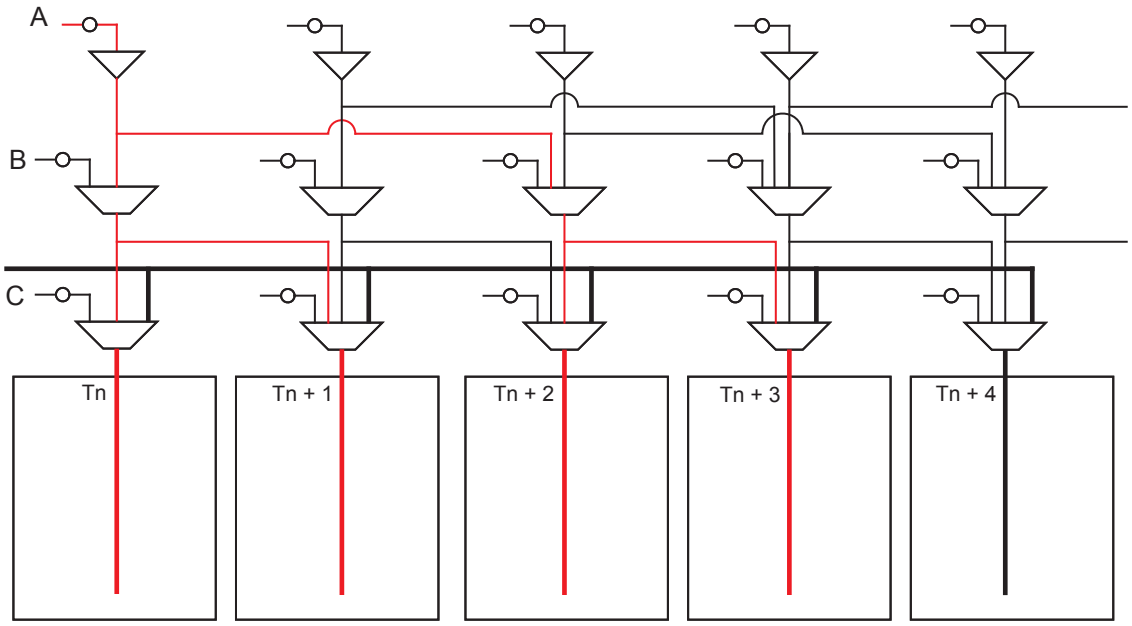


***Figure 2-10* • Four Spines Aggregation**

***Table 2-7* • Spine Aggregation in A3PE600 or AFS600**

Clock Aggregation	Spine
1 spine	T1, T2, T3, T4, T5, T6, B1, B2, B3, B4, B5, B6
2 spines	T1:T2, T2:T3, T3:T4, T4:T5, T5:T6, B1:B2, B2:B3, B3:B4, B4:B5, B5:B6
4 spines	B1:B4, B2:B5, B3:B6, T1:T4, T2:T5, T3:T6

The clock aggregation for the quadrant spines can cross over from the left to right quadrant, but not from top to bottom. The quadrant spine assignment T1:T4 is legal, but the quadrant spine assignment T1:B1 is not legal. Note that this clock aggregation is hardwired. You can always assign signals to spine T1 and B2 by instantiating a buffer, but this may add skew in the signal.

Design Recommendations

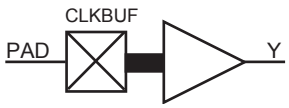
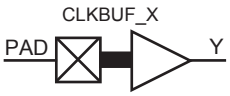
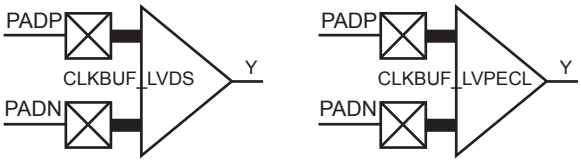
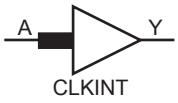
The following sections provide design flow recommendations for using a global network in a design.

- "Global Macros and I/O Standards"
- "Global Macro and Placement Selections" on page 40
- "Using Global Macros in Synplicity" on page 42
- "Global Promotion and Demotion Using PDC" on page 43
- "Spine Assignment" on page 44
- "Designer Flow for Global Assignment" on page 45
- "Simple Design Example" on page 47
- "Global Management in PLL Design" on page 49
- "Using Spines of Occupied Global Networks" on page 50

Global Macros and I/O Standards

The larger low power flash devices have six chip global networks and four quadrant global networks. However, the same clock macros are used for assigning signals to chip globals and quadrant globals. Depending on the clock macro placement or assignment in the Physical Design Constraint (PDC) file or MultiView Navigator (MVN), the signal will use the chip global network or quadrant network. Table 2-8 lists the clock macros available for low power flash devices. Refer to the *IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide* for details.

***Table 2-8* • Clock Macros**

Macro Name	Description	Symbol
CLKBUF	Input macro for Clock Network	
CLKBUF_x	Input macro for Clock Network with specific I/O standard	
CLKBUF_LVDS/LVPECL	LVDS or LVPECL input macro for Clock Network (not supported for IGLOO nano or ProASIC3 nano devices)	
CLKINT	Macro for internal clock interface	
CLKBIBUF	Bidirectional macro with input dedicated to routed Clock Network	

Use these available macros to assign a signal to the global network. In addition to these global macros, PLL and CLKDLY macros can also drive the global networks. Use I/O–standard–specific clock macros (CLKBUF_x) to instantiate a specific I/O standard for the global signals. Table 2-9 on page 39 shows the list of these I/O–standard–specific macros. Note that if you use these I/O–standard–specific clock macros, you cannot change the I/O standard later in the design stage. If you use the regular CLKBUF macro, you can use MVN or the PDC file in Designer to change the I/O standard. The default I/O

standard for CLKBUF is LVTTL in the current Microsemi Libero® System-on-Chip (SoC) and Designer software.

***Table 2-9* • I/O Standards within CLKBUF**

Name	Description
CLKBUF_LVCMOS5	LVCMOS clock buffer with 5.0 V CMOS voltage level
CLKBUF_LVCMOS33	LVCMOS clock buffer with 3.3 V CMOS voltage level
CLKBUF_LVCMOS25	LVCMOS clock buffer with 2.5 V CMOS voltage level[1]
CLKBUF_LVCMOS18	LVCMOS clock buffer with 1.8 V CMOS voltage level
CLKBUF_LVCMOS15	LVCMOS clock buffer with 1.5 V CMOS voltage level
CLKBUF_LVCMOS12	LVCMOS clock buffer with 1.2 V CMOS voltage level
CLKBUF_PCI	PCI clock buffer
CLKBUF_PCIX	PCIX clock buffer
CLKBUF_GTL25	GTL clock buffer with 2.5 V CMOS voltage level[1]
CLKBUF_GTL33	GTL clock buffer with 3.3 V CMOS voltage level[1]
CLKBUF_GTLP25	GTL+ clock buffer with 2.5 V CMOS voltage level[1]
CLKBUF_GTLP33	GTL+ clock buffer with 3.3 V CMOS voltage level[1]
CLKBUF_ HSTL _I	HSTL Class I clock buffer[1]
CLKBUF_ HSTL _II	HSTL Class II clock buffer[1]
CLKBUF_SSTL2_I	SSTL2 Class I clock buffer[1]
CLKBUF_SSTL2_II	SSTL2 Class II clock buffer[1]
CLKBUF_SSTL3_I	SSTL3 Class I clock buffer[1]
CLKBUF_SSTL3_II	SSTL3 Class II clock buffer[1]

Notes:

1. *Supported in only the IGLOOe, ProASIC3E, AFS600, and AFS1500 devices*
2. *By default, the CLKBUF macro uses the 3.3 V LVTTL I/O technology.*

The current synthesis tool libraries only infer the CLKBUF or CLKINT macros in the netlist. All other global macros must be instantiated manually into your HDL code. The following is an example of CLKBUF_LVCMOS25 global macro instantiations that you can copy and paste into your code:

VHDL

```
component clkbuf_lvcmos25
  port (pad : in std_logic; y : out std_logic);
end component

begin
-- concurrent statements
u2 : clkbuf_lvcmos25 port map (pad =>  ext_clk, y => int_clk);
end
```

Verilog

```
module design (______);

input _____;
output ______;

clkbuf_lvcmos25 u2 (.y(int_clk), .pad(ext_clk);

endmodule
```

Global Macro and Placement Selections

Low power flash devices provide the flexibility of choosing one of the three global input pad locations available to connect to a global / quadrant global network. For 60K gate devices and above, if the single-ended I/O standard is chosen, there is flexibility to choose one of the global input pads (the first, second, and fourth input). Once chosen, the other I/O locations are used as regular I/Os. If the differential I/O standard is chosen, the first and second inputs are considered as paired, and the third input is paired with a regular I/O. The user then has the choice of selecting one of the two sets to be used as the global input source. There is also the option to allow an internal clock signal to feed the global network. A multiplexer tree selects the appropriate global input for routing to the desired location. Note that the global I/O pads do not need to feed the global network; they can also be used as regular I/O pads.

Hardwired I/O Clock Source

Hardwired I/O refers to global input pins that are hardwired to the multiplexer tree, which directly accesses the global network. These global input pins have designated pin locations and are indicated with the I/O naming convention Gmn (m refers to any one of the positions where the global buffers is available, and n refers to any one of the three global input MUXes and the pin number of the associated global location, m). Choosing this option provides the benefit of directly connecting to the global buffers, which provides less delay. See Figure 2-11 for an example illustration of the connections, shown in red. If a CLKBUF macro is initiated, the clock input can be placed at one of nine dedicated global input pin locations: GmA0, GmA1, GmA2, GmB0, GmB1, GmB2, GmC0, GmC1, or GmC2. Note that the placement of the global will determine whether you are using chip global or quadrant global. For example, if the CLKBIF is placed in one of the GF pin locations, it will use the chip global network; if the CLKBIF is placed in one of the GA pin locations, it will use quadrant global network. This is shown in Figure 2-12 on page 41 and Figure 2-13 on page 41.

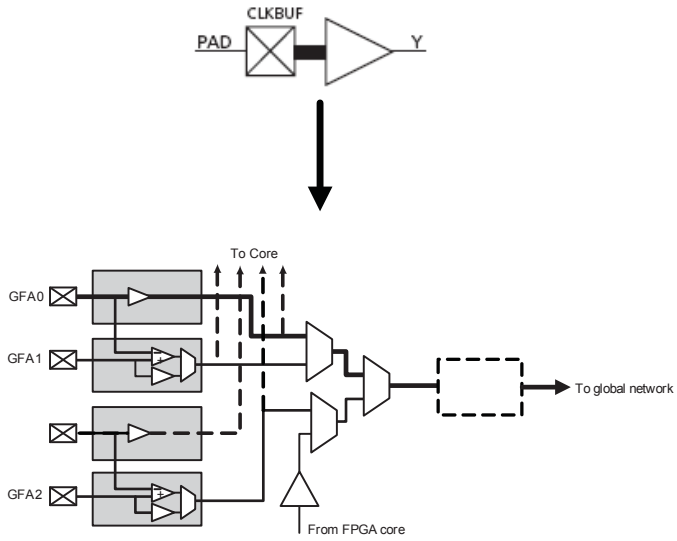


***Figure 2-11* • CLKBUF Macro**

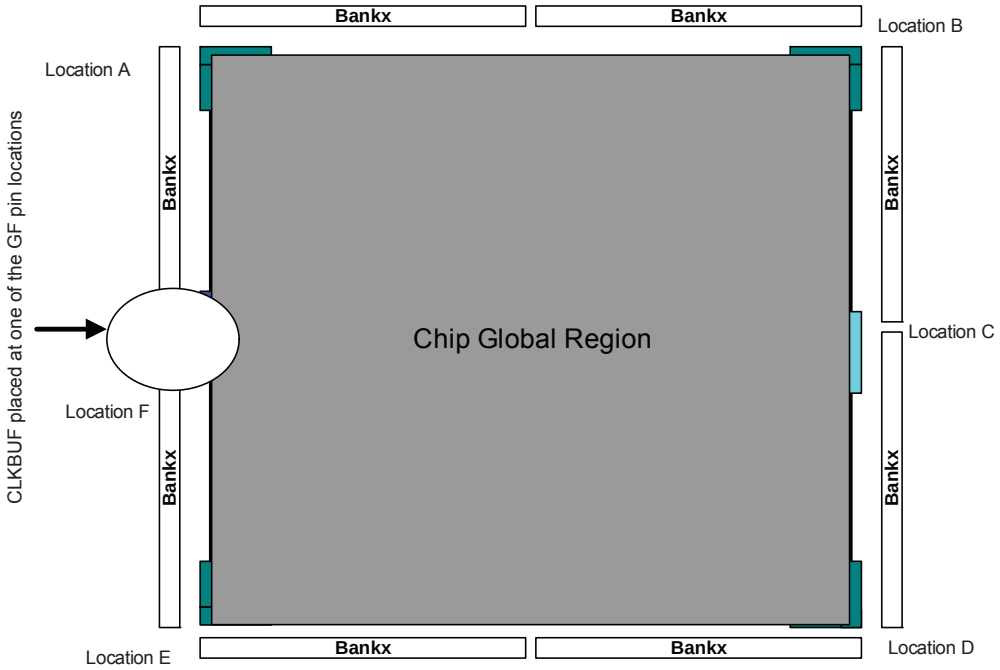


***Figure 2-12* • Chip Global Region**

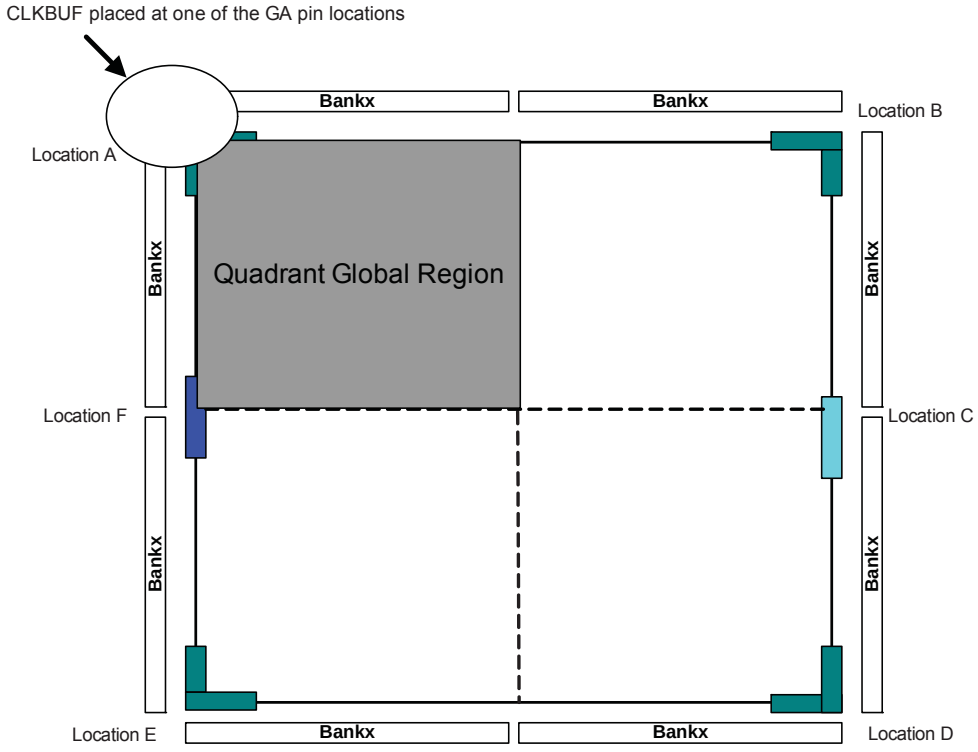


***Figure 2-13* • Quadrant Global Region**

External I/O or Local signal as Clock Source

External I/O refers to regular I/O pins are labeled with the I/O convention IOuxwByVz. You can allow the external I/O or internal signal to access the global. To allow the external I/O or internal signal to access the global network, you need to instantiate the CLKINT macro. Refer to Figure 2-4 on page 27 for an example illustration of the connections. Instead of using CLKINT, you can also use PDC to promote signals from external I/O or internal signal to the global network. However, it may cause layout issues because of synthesis logic replication. Refer to the "Global Promotion and Demotion Using PDC" section on page 43 for details.

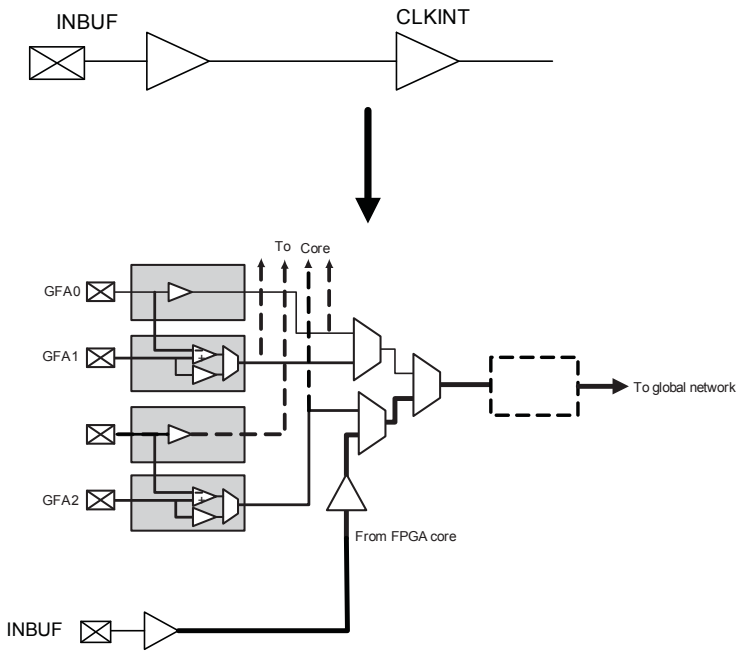
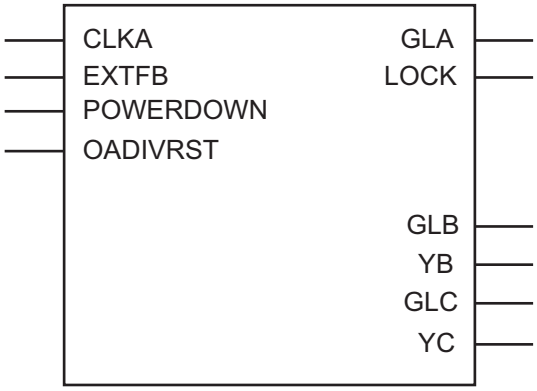


Figure 2-14 • **CLKINT Macro**

Using Global Macros in Synplicity

The Synplify® synthesis tool automatically inserts global buffers for nets with high fanout during synthesis. By default, Synplicity® puts six global macros (CLKBUF or CLKINT) in the netlist, including any global instantiation or PLL macro. Synplify always honors your global macro instantiation. If you have a PLL (only primary output is used) in the design, Synplify adds five more global buffers in the netlist. Synplify uses the following global counting rule to add global macros in the netlist:

1. CLKBUF: 1 global buffer
2. CLKINT: 1 global buffer
3. CLKDLY: 1 global buffer
4. PLL: 1 to 3 global buffers
   - GLA, GLB, GLC, YB, and YC are counted as 1 buffer.
   - GLB or YB is used or both are counted as 1 buffer.
   - GLC or YC is used or both are counted as 1 buffer.



Note: OAVDIVRST exists only in the Fusion PLL.

***Figure 2-15* • PLLs in Low Power Flash Devices**

You can use the syn_global_buffers attribute in Synplify to specify a maximum number of global macros to be inserted in the netlist. This can also be used to restrict the number of global buffers inserted. In the Synplicity 8.1 version or newer, a new attribute, syn_global_minfanout, has been added for low power flash devices. This enables you to promote only the high-fanout signal to global. However, be aware that you can only have six signals assigned to chip global networks, and the rest of the global signals should be assigned to quadrant global networks. So, if the netlist has 18 global macros, the remaining 12 global macros should have fanout that allows the instances driven by these globals to be placed inside a quadrant.

Global Promotion and Demotion Using PDC

The HDL source file or schematic is the preferred place for defining which signals should be assigned to a clock network using clock macro instantiation. This method is preferred because it is guaranteed to be honored by the synthesis tools and Designer software and stop any replication on this net by the synthesis tool. Note that a signal with fanout may have logic replication if it is not promoted to global during synthesis. In that case, the user cannot promote that signal to global using PDC. See Synplicity Help for details on using this attribute. To help you with global management, Designer allows you to promote a signal to a global network or demote a global macro to a regular macro from the user netlist using the compile options and/or PDC commands.

The following are the PDC constraints you can use to promote a signal to a global network:

1. PDC syntax to promote a regular net to a chip global clock:

   ```
   assign_global_clock –net netname
   ```

   The following will happen during promotion of a regular signal to a global network:

   – If the net is external, the net will be driven by a CLKINT inserted automatically by Compile.
   – The I/O macro will not be changed to CLKBUF macros.
   – If the net is an internal net, the net will be driven by a CLKINT inserted automatically by Compile.

2. PDC syntax to promote a net to a quadrant clock:

   ```
   assign_local_clock –net netname –type quadrant UR|UL|LR|LL
   ```

   This follows the same rule as the chip global clock network.

The following PDC command demotes the clock nets to regular nets.

```
unassign_global_clock -net netname
```

The following will happen during demotion of a global signal to regular nets:

- CLKBUF_x becomes INBUF_x; CLKINT is removed from the netlist.
- The essential global macro, such as the output of the Clock Conditioning Circuit, cannot be demoted.
- No automatic buffering will happen.

Since no automatic buffering happens when a signal is demoted, this net may have a high delay due to large fanout. This may have a negative effect on the quality of the results. Microsemi recommends that the automatic global demotion only be used on small-fanout nets. Use clock networks for high-fanout nets to improve timing and routability.

Spine Assignment

The low power flash device architecture allows the global networks to be segmented and used as clock spines. These spines, also called local clock networks, enable the use of PDC or MVN to assign a signal to a spine.

PDC syntax to promote a net to a spine/local clock:

```
assign_local_clock –net netname –type [quadrant|chip] Tn|Bn|Tn:Bm
```

If the net is driven by a clock macro, Designer automatically demotes the clock net to a regular net before it is assigned to a spine. Nets driven by a PLL or CLKDLY macro cannot be assigned to a local clock.

When assigning a signal to a spine or quadrant global network using PDC (pre-compile), the Designer software will legalize the shared instances. The number of shared instances to be legalized can be controlled by compile options. If these networks are created in MVN (only quadrant globals can be created), no legalization is done (as it is post-compile). Designer does not do legalization between non-clock nets.

As an example, consider two nets, net_clk and net_reset, driving the same flip-flop. The following PDC constraints are used:

```
assign_local_clock –net net_clk –type chip T3
assign_local_clock –net net_reset –type chip T1:T2
```

During Compile, Designer adds a buffer in the reset net and places it in the T1 or T2 region, and places the flip-flop in the T3 spine region (Figure 2-16).

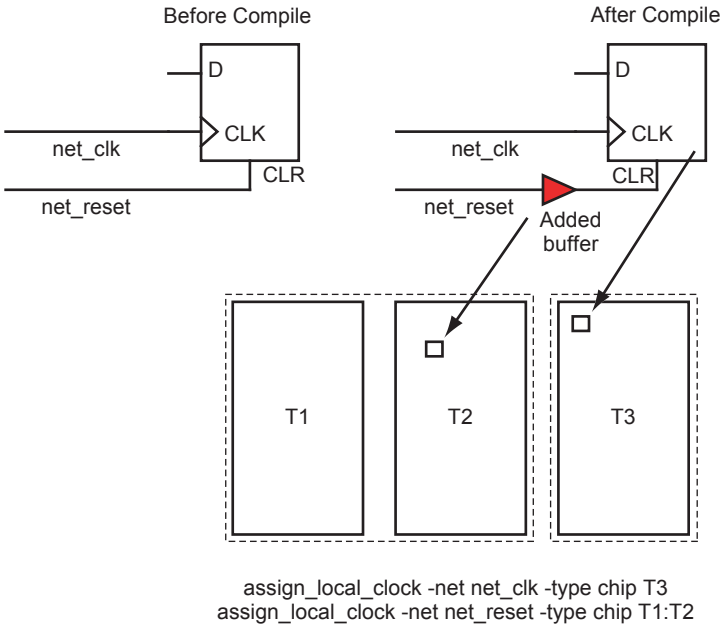


Figure 2-16 • **Adding a Buffer for Shared Instances**

You can control the maximum number of shared instances allowed for the legalization to take place using the Compile Option dialog box shown in Figure 2-17. Refer to Libero SoC / Designer online help for details on the Compile Option dialog box. A large number of shared instances most likely indicates a floorplanning problem that you should address.

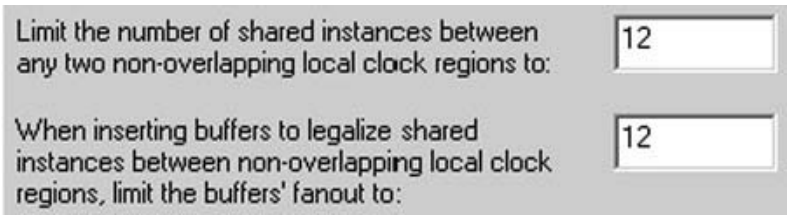


***Figure 2-17* • Shared Instances in the Compile Option Dialog Box**

Designer Flow for Global Assignment

To achieve the desired result, pay special attention to global management during synthesis and place-and-route. The current Synplify tool does not insert more than six global buffers in the netlist by default. Thus, the default flow will not assign any signal to the quadrant global network. However, you can use attributes in Synplify and increase the default global macro assignment in the netlist. Designer v6.2 supports automatic quadrant global assignment, which was not available in Designer v6.1. Layout will make the choice to assign the correct signals to global. However, you can also utilize PDC and perform manual global assignment to overwrite any automatic assignment. The following step-by-step suggestions guide you in the layout of your design and help you improve timing in Designer:

1. Run Compile and check the Compile report. The Compile report has global information in the "Device Utilization" section that describes the number of chip and quadrant signals in the design. A "Net Report" section describes chip global nets, quadrant global nets, local clock nets, a list of nets listed by fanout, and net candidates for local clock assignment. Review this information. Note that YB or YC are counted as global only when they are used in isolation; if you use YB only and not GLB, this net is not shown in the global/quadrant nets report. Instead, it appears in the Global Utilization report.
2. If some signals have a very high fanout and are candidates for global promotion, promote those signals to global using the compile options or PDC commands. Figure 2-18 on page 46 shows the Globals Management section of the compile options. Select **Promote regular nets whose fanout is greater than** and enter a reasonable value for fanouts.

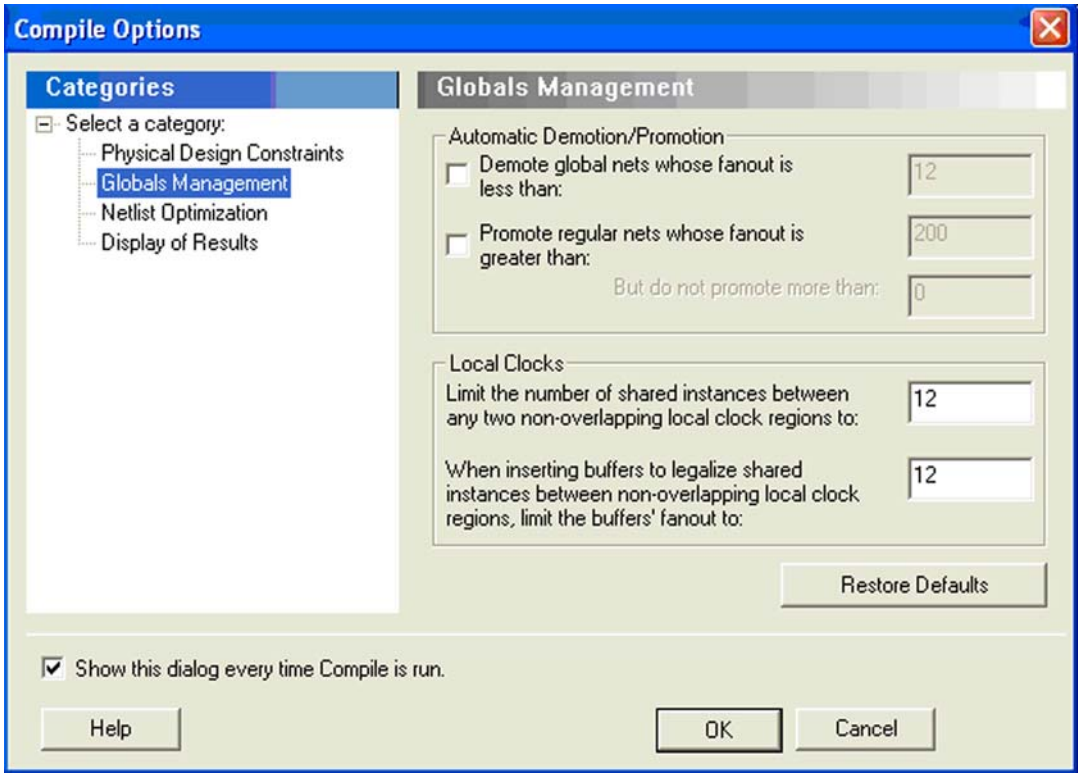


Figure 2-18 • **Globals Management GUI in Designer**

3. Occasionally, the synthesis tool assigns a global macro to clock nets, even though the fanout is significantly less than other asynchronous signals. Select **Demote global nets whose fanout is less than** and enter a reasonable value for fanouts. This frees up some global networks from the signals that have very low fanouts. This can also be done using PDC.
4. Use a local clock network for the signals that do not need to go to the whole chip but should have low skew. This local clock network assignment can only be done using PDC.
5. Assign the I/O buffer using MVN if you have fixed I/O assignment. As shown in Figure 2-10 on page 37, there are three sets of global pins that have a hardwired connection to each global network. Do not try to put multiple CLKBUF macros in these three sets of global pins. For example, do not assign two CLKBUFs to GAA0x and GAA2x pins.
6. You must click **Commit** at the end of MVN assignment. This runs the pre-layout checker and checks the validity of global assignment.
7. Always run Compile with the **Keep existing physical constraints** option on. This uses the quadrant clock network assignment in the MVN assignment and checks if you have the desired signals on the global networks.
8. Run Layout and check the timing.

Simple Design Example

Consider a design consisting of six building blocks (shift registers) and targeted for an A3PE600-PQ208 (Figure 2-16 on page 44). The example design consists of two PLLs (PLL1 has GLA only; PLL2 has both GLA and GLB), a global reset (ACLR), an enable (EN_ALL), and three external clock domains (QCLK1, QCLK2, and QCLK3) driving the different blocks of the design. Note that the PQ208 package only has two PLLs (which access the chip global network). Because of fanout, the global reset and enable signals need to be assigned to the chip global resources. There is only one free chip global for the remaining global (QCLK1, QCLK2, QCLK3). Place two of these signals on the quadrant global resource. The design example demonstrates manually assignment of QCLK1 and QCLK2 to the quadrant global using the PDC command.

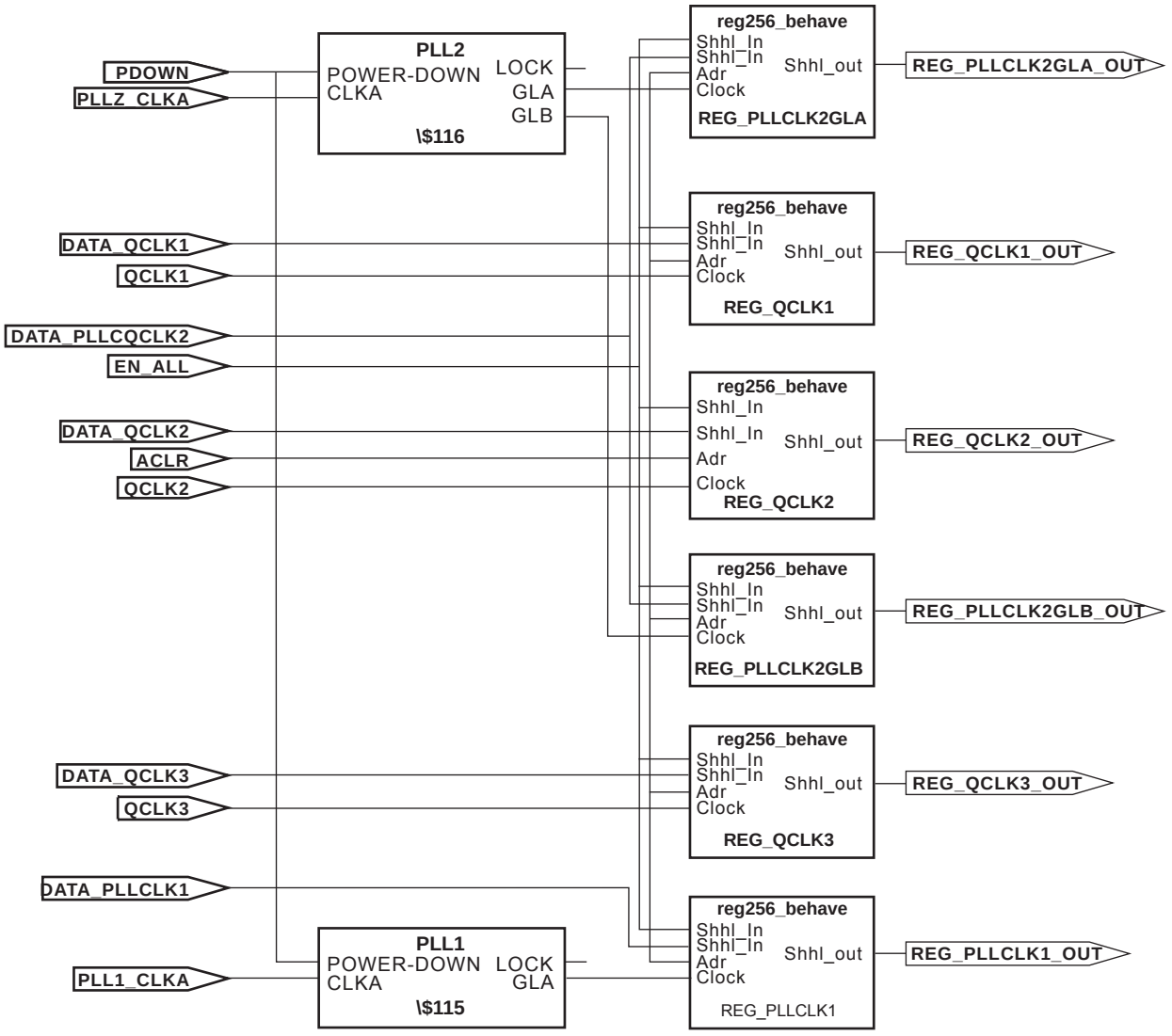


Figure 2-19 • **Block Diagram of the Global Management Example Design**

Step 1

Run Synthesis with default options. The Synplicity log shows the following device utilization:

Cell usage:

	cell count	area	count*area
DFN1E1C1	1536	2.0	3072.0
BUFF	278	1.0	278.0
INBUF	10	0.0	0.0
VCC	9	0.0	0.0
GND	9	0.0	0.0
OUTBUF	6	0.0	0.0
CLKBUF	3	0.0	0.0
PLL	2	0.0	0.0
TOTAL	1853		3350.0

Step 2

Run Compile with the **Promote regular nets whose fanout is greater than** option selected in Designer; you will see the following in the Compile report:

```
Device utilization report:
==========================
CORE               Used:   1536  Total:  13824   (11.11%)
IO (W/ clocks)     Used:     19  Total:    147   (12.93%)
Differential IO    Used:      0  Total:     65    (0.00%)
GLOBAL             Used:      8  Total:     18   (44.44%)
PLL                Used:      2  Total:      2  (100.00%)
RAM/FIFO           Used:      0  Total:     24    (0.00%)
FlashROM           Used:      0  Total:      1    (0.00%)
……….
The following nets have been assigned to a global resource:
Fanout  Type          Name
-------------------------
1536    INT_NET       Net   : EN_ALL_c
                      Driver: EN_ALL_pad_CLKINT
                      Source: AUTO PROMOTED
1536    SET/RESET_NET Net   : ACLR_c
                      Driver: ACLR_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK1_c
                      Driver: QCLK1_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK2_c
                      Driver: QCLK2_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK3_c
                      Driver: QCLK3_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : $1N14
                      Driver: $1I5/Core
                      Source: ESSENTIAL
256     CLK_NET       Net   : $1N12
                      Driver: $1I6/Core
                      Source: ESSENTIAL
256     CLK_NET       Net   : $1N10
                      Driver: $1I6/Core
                      Source: ESSENTIAL
```

Designer will promote five more signals to global due to high fanout. There are eight signals assigned to global networks.

During Layout, Designer will assign two of the signals to quadrant global locations.

Step 3 (optional)

You can also assign the QCLK1_c and QCLK2_c nets to quadrant regions using the following PDC commands:

```
assign_local_clock –net QCLK1_c  –type quadrant UL
assign_local_clock –net QCLK2_c  –type quadrant LL
```

Step 4

Import this PDC with the netlist and run Compile again. You will see the following in the Compile report:

```
The following nets have been assigned to a global resource:
Fanout  Type          Name
--------------------------
1536    INT_NET       Net   : EN_ALL_c
                      Driver: EN_ALL_pad_CLKINT
                      Source: AUTO PROMOTED
1536    SET/RESET_NET Net   : ACLR_c
                      Driver: ACLR_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK3_c
                      Driver: QCLK3_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : $1N14
                      Driver: $1I5/Core
                      Source: ESSENTIAL
256     CLK_NET       Net   : $1N12
                      Driver: $1I6/Core
                      Source: ESSENTIAL
256     CLK_NET       Net   : $1N10
                      Driver: $1I6/Core
                      Source: ESSENTIAL
The following nets have been assigned to a quadrant clock resource using PDC:
Fanout  Type          Name
--------------------------
256     CLK_NET       Net   : QCLK1_c
                      Driver: QCLK1_pad_CLKINT
                      Region: quadrant_UL
256     CLK_NET       Net   : QCLK2_c
                      Driver: QCLK2_pad_CLKINT
                      Region: quadrant_LL
```

Step 5

Run Layout.

Global Management in PLL Design

This section describes the legal global network connections to PLLs in the low power flash devices. For detailed information on using PLLs, refer to "Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" section on page 53. Microsemi recommends that you use the dedicated global pins to directly drive the reference clock input of the associated PLL for reduced propagation delays and clock distortion. However, low power flash devices offer the flexibility to connect other signals to reference clock inputs. Each PLL is associated with three global networks (Figure 2-5 on page 28). There are some limitations, such as when trying to use the global and PLL at the same time:

- If you use a PLL with only primary output, you can still use the remaining two free global networks.
- If you use three globals associated with a PLL location, you cannot use the PLL on that location.
- If the YB or YC output is used standalone, it will occupy one global, even though this signal does not go to the global network.

Using Spines of Occupied Global Networks

When a signal is assigned to a global network, the flash switches are programmed to set the MUX select lines (explained in the "Clock Aggregation Architecture" section on page 37) to drive the spines of that network with the global net. However, if the global net is restricted from reaching into the scope of a spine, the MUX drivers of that spine are available for other high-fanout or critical signals (Figure 2-20).

For example, if you want to limit the CLK1_c signal to the left half of the chip and want to use the right side of the same global network for CLK2_c, you can add the following PDC commands:

```
define_region -name region1 -type inclusive 0 0 34 29
assign_net_macros region1 CLK1_c
assign_local_clock –net CLK2_c –type chip B2
```

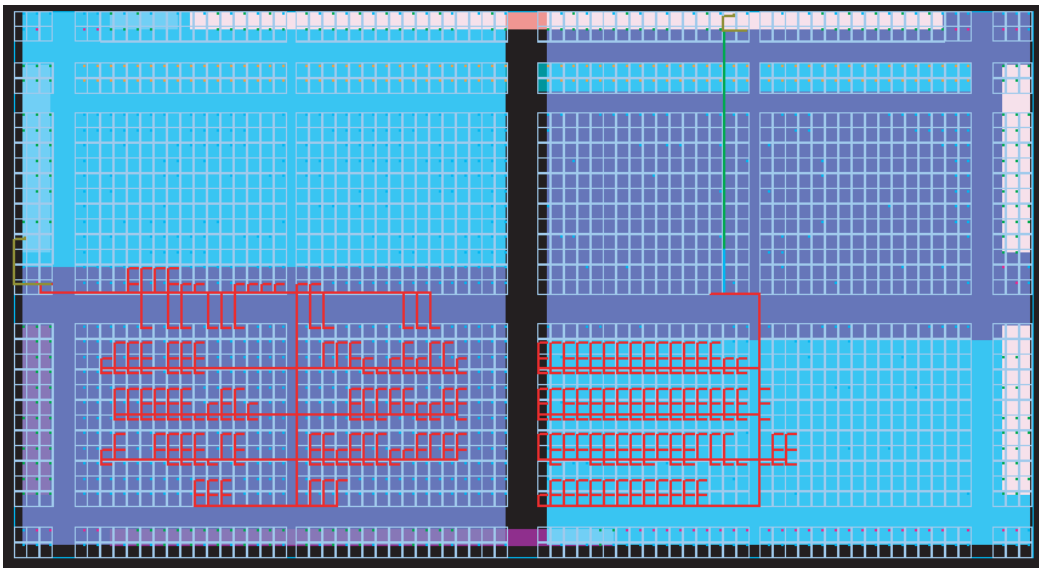


***Figure 2-20* • Design Example Using Spines of Occupied Global Networks**

Conclusion

IGLOO, Fusion, and ProASIC3 devices contain 18 global networks: 6 chip global networks and 12 quadrant global networks. These global networks can be segmented into local low-skew networks called spines. The spines provide low-skew networks for the high-fanout signals of a design. These allow you up to 252 different internal/external clocks in an A3PE3000 device. This document describes the architecture for the global network, plus guidelines and methodologies in assigning signals to globals and spines.

Related Documents

User's Guides

IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide

http://www.microsemi.com/soc/documents/pa3_libguide_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
	Notes were added where appropriate to point out that IGLOO nano and ProASIC3 nano devices do not support differential inputs (SAR 21449).	N/A
	The "Global Architecture" section and "VersaNet Global Network Distribution" section were revised for clarity (SARs 20646, 24779).	23, 25
	The "I/O Banks and Global I/Os" section was moved earlier in the document, renamed to "Chip and Quadrant Global I/Os", and revised for clarity. Figure 2-4 • Global Connections Details, Figure 2-6 • Global Inputs, Table 2-2 • Chip Global Pin Name, and Table 2-3 • Quadrant Global Pin Name are new (SARs 20646, 24779).	27
	The "Clock Aggregation Architecture" section was revised (SARs 20646, 24779).	33
	Figure 2-7 • Chip Global Aggregation was revised (SARs 20646, 24779).	35
	The "Global Macro and Placement Selections" section is new (SARs 20646, 24779).	40
v1.4 (December 2008)	The "Global Architecture" section was updated to include 10 k devices, and to include information about VersaNet global support for IGLOO nano devices.	23
	The Table 2-1 • Flash-Based FPGAs was updated to include IGLOO nano and ProASIC3 nano devices.	24
	The "VersaNet Global Network Distribution" section was updated to include 10 k devices and to note an exception in global lines for nano devices.	25
	Figure 2-2 • Simplified VersaNet Global Network (30 k gates and below) is new.	26
	The "Spine Architecture" section was updated to clarify support for 10 k and nano devices.	33
	Table 2-4 • Globals/Spines/Rows for IGLOO and ProASIC3 Devices was updated to include IGLOO nano and ProASIC3 nano devices.	33
	The figure in the CLKBUF_LVDS/LVPECL row of Table 2-8 • Clock Macros was updated to change CLKBIBUF to CLKBUF.	38
v1.3 (October 2008)	A third bullet was added to the beginning of the "Global Architecture" section: In Fusion devices, the west CCC also contains a PLL core. In the two larger devices (AFS600 and AFS1500), the west and east CCCs each contain a PLL.	23
	The "Global Resource Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	24
	Table 2-4 • Globals/Spines/Rows for IGLOO and ProASIC3 Devices was updated to include A3PE600/L in the device column.	33
	Table note 1 was revised in Table 2-9 • I/O Standards within CLKBUF to include AFS600 and AFS1500.	39
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 2-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	24

Date	Changes	Page
v1.1 (March 2008)	The "Global Architecture" section was updated to include the IGLOO PLUS family. The bullet was revised to include that the west CCC does not contain a PLL core in 15 k and 30 k devices. Instances of "A3P030 and AGL030 devices" were replaced with "15 k and 30 k gate devices."	23
v1.1 (continued)	Table 2-1 • Flash-Based FPGAs and the accompanying text was updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	24
	The "VersaNet Global Network Distribution" section, "Spine Architecture" section, the note in Figure 2-1 • Overview of VersaNet Global Network and Device Architecture, and the note in Figure 2-3 • Simplified VersaNet Global Network (60 k gates and above) were updated to include mention of 15 k gate devices.	25, 26
	Table 2-4 • Globals/Spines/Rows for IGLOO and ProASIC3 Devices was updated to add the A3P015 device, and to revise the values for clock trees, globals/spines per tree, and globals/spines per device for the A3P030 and AGL030 devices.	33
	Table 2-5 • Globals/Spines/Rows for IGLOO PLUS Devices is new.	34
	CLKBUF_LVCMOS12 was added to Table 2-9 • I/O Standards within CLKBUF.	39
	The "User's Guides" section was updated to include the three different I/O Structures chapters for ProASIC3 and IGLOO device families.	50
v1.0 (January 2008)	Figure 2-3 • Simplified VersaNet Global Network (60 k gates and above) was updated.	26
	The "Naming of Global I/Os" section was updated.	27
	The "Using Global Macros in Synplicity" section was updated.	42
	The "Global Promotion and Demotion Using PDC" section was updated.	43
	The "Designer Flow for Global Assignment" section was updated.	45
	The "Simple Design Example" section was updated.	47
51900087-0/1.05 (January 2005)	Table 2-4 • Globals/Spines/Rows for IGLOO and ProASIC3 Devices was updated.	33

3 – Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs

Introduction

This document outlines the following device information: Clock Conditioning Circuit (CCC) features, PLL core specifications, functional descriptions, software configuration information, detailed usage information, recommended board-level considerations, and other considerations concerning clock conditioning circuits and global networks in low power flash devices or mixed signal FPGAs.

Overview of Clock Conditioning Circuitry

In Fusion, IGLOO, and ProASIC3 devices, the CCCs are used to implement frequency division, frequency multiplication, phase shifting, and delay operations. The CCCs are available in six chip locations—each of the four chip corners and the middle of the east and west chip sides. For device-specific variations, refer to the "Device-Specific Layout" section on page 70.

The CCC is composed of the following:

- PLL core
- 3 phase selectors
- 6 programmable delays and 1 fixed delay that advances/delays phase
- 5 programmable frequency dividers that provide frequency multiplication/division (not shown in Figure 3-6 on page 63 because they are automatically configured based on the user's required frequencies)
- 1 dynamic shift register that provides CCC dynamic reconfiguration capability

Figure 3-1 provides a simplified block diagram of the physical implementation of the building blocks in each of the CCCs.

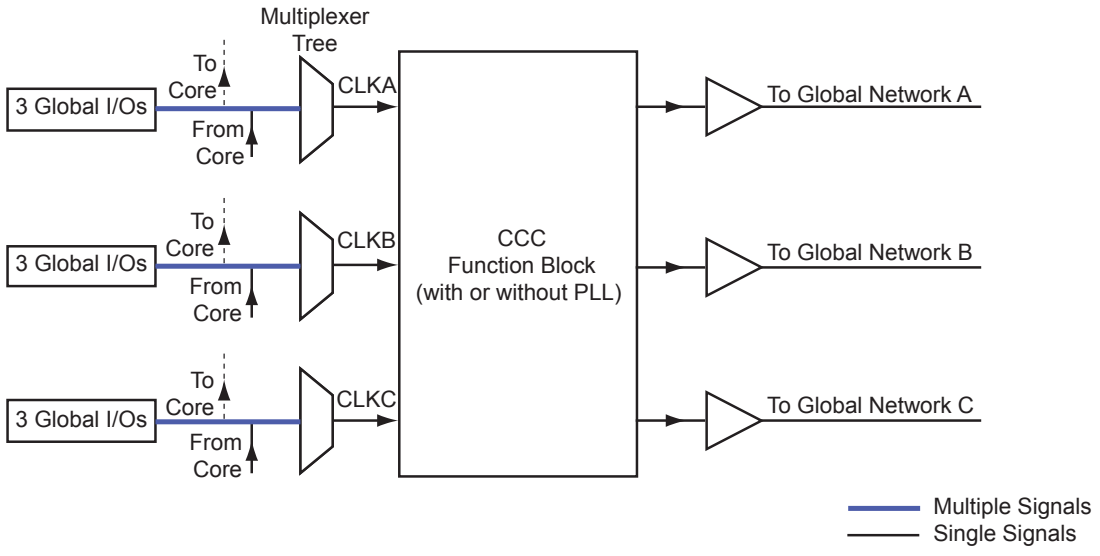


***Figure 3-1* • Overview of the CCCs Offered in Fusion, IGLOO, and ProASIC3**

Each CCC can implement up to three independent global buffers (with or without programmable delay) or a PLL function (programmable frequency division/multiplication, phase shift, and delays) with up to three global outputs. Unused global outputs of a PLL can be used to implement independent global buffers, up to a maximum of three global outputs for a given CCC.

CCC Programming

The CCC block is fully configurable, either via flash configuration bits set in the programming bitstream or through an asynchronous interface. This asynchronous dedicated shift register interface is dynamically accessible from inside the low power flash devices to permit parameter changes, such as PLL divide ratios and delays, during device operation.

To increase the versatility and flexibility of the clock conditioning system, the CCC configuration is determined either by the user during the design process, with configuration data being stored in flash memory as part of the device programming procedure, or by writing data into a dedicated shift register during normal device operation.

This latter mode allows the user to dynamically reconfigure the CCC without the need for core programming. The shift register is accessed through a simple serial interface. Refer to the "UJTAG Applications in Microsemi's Low Power Flash Devices" section on page 419 or the application note *Using Global Resources in Actel Fusion Devices*.

Global Resources

Low power flash and mixed signal devices provide three global routing networks (GLA, GLB, and GLC) for each of the CCC locations. There are potentially many I/O locations; each global I/O location can be chosen from only one of three possibilities. This is controlled by the multiplexer tree circuitry in each global network. Once the I/O location is selected, the user has the option to utilize the CCCs before the signals are connected to the global networks. The CCC in each location (up to six) has the same structure, so generating the CCC macros is always done with an identical software GUI. The CCCs in the corner locations drive the quadrant global networks, and the CCCs in the middle of the east and west chip sides drive the chip global networks. The quadrant global networks span only a quarter of the device, while the chip global networks span the entire device. For more details on global resources offered in low power flash devices, refer to the "Global Resources in Low Power Flash Devices" section on page 23.

A global buffer can be placed in any of the three global locations (CLKA-GLA, CLKB-GLB, or CLKC-GLC) of a given CCC. A PLL macro uses the CLKA CCC input to drive its reference clock. It uses the GLA and, optionally, the GLB and GLC global outputs to drive the global networks. A PLL macro can also drive the YB and YC regular core outputs. The GLB (or GLC) global output cannot be reused if the YB (or YC) output is used. Refer to the "PLL Macro Signal Descriptions" section on page 60 for more information.

Each global buffer, as well as the PLL reference clock, can be driven from one of the following:

- 3 dedicated single-ended I/Os using a hardwired connection
- 2 dedicated differential I/Os using a hardwired connection (not supported for IGLOO nano or ProASIC3 nano devices)
- The FPGA core

CCC Support in Microsemi's Flash Devices

The flash FPGAs listed in Table 3-1 support the CCC feature and the functions described in this document.

***Table 3-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
	IGLOO nano	The industry's lowest-power, smallest-size solution
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 3-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 3-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Global Buffers with No Programmable Delays

Access to the global / quadrant global networks can be configured directly from the global I/O buffer, bypassing the CCC functional block (as indicated by the dotted lines in Figure 3-1 on page 53). Internal signals driven by the FPGA core can use the global / quadrant global networks by connecting via the routed clock input of the multiplexer tree.

There are many specific CLKBUF macros supporting the wide variety of single-ended I/O inputs (CLKBUF) and differential I/O standards (CLKBUF_LVDS/LVPECL) in the low power flash families. They are used when connecting global I/Os directly to the global/quadrant networks.

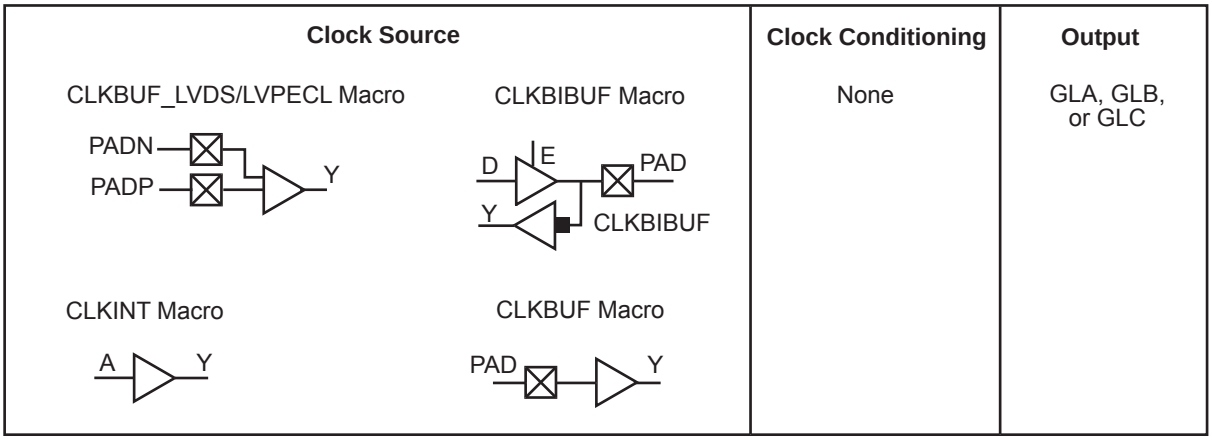
Note: IGLOO nano and ProASIC nano devices do not support differential inputs.

When an internal signal needs to be connected to the global/quadrant network, the CLKINT macro is used to connect the signal to the routed clock input of the network's MUX tree.

To utilize direct connection from global I/Os or from internal signals to the global/quadrant networks, CLKBUF, CLKBUF_LVPECL/LVDS, and CLKINT macros are used (Figure 3-2).

- The CLKBUF and CLKBUF_LVPECL/LVDS[1] macros are composite macros that include an I/O macro driving a global buffer, which uses a hardwired connection.
- The CLKBUF, CLKBUF_LVPECL/LVDS[1] and CLKINT macros are pass-through clock sources and do not use the PLL or provide any programmable delay functionality.
- The CLKINT macro provides a global buffer function driven internally by the FPGA core.

The available CLKBUF macros are described in the *IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide*.



Note: IGLOO nano and ProASIC nano devices do not support differential inputs.

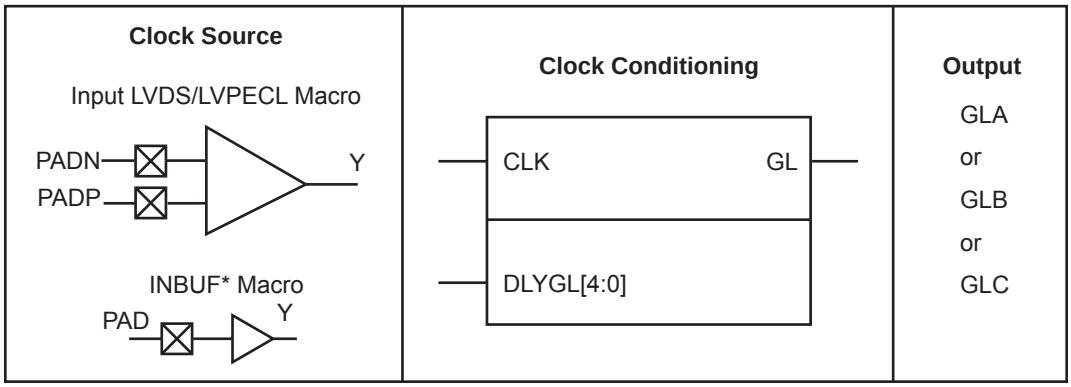
***Figure 3-2* • CCC Options: Global Buffers with No Programmable Delay**

Global Buffer with Programmable Delay

Clocks requiring clock adjustments can utilize the programmable delay cores before connecting to the global / quadrant global networks. A maximum of 18 CCC global buffers can be instantiated in a device—three per CCC and up to six CCCs per device.

Each CCC functional block contains a programmable delay element for each of the global networks (up to three), and users can utilize these features by using the corresponding macro (Figure 3-3 on page 57).

1. *B-LVDS and M-LVDS are supported with the LVDS macro.*



Notes:

1. *For INBUF* driving a PLL macro or CLKDLY macro, the I/O will be hard-routed to the CCC; i.e., will be placed by software to a dedicated Global I/O.*
2. *IGLOO nano and ProASIC3 nano devices do not support differential inputs.*

***Figure 3-3* • CCC Options: Global Buffers with Programmable Delay**

The CLKDLY macro is a pass-through clock source that does not use the PLL, but provides the ability to delay the clock input using a programmable delay. The CLKDLY macro takes the selected clock input and adds a user-defined delay element. This macro generates an output clock phase shift from the input clock.

The CLKDLY macro can be driven by an INBUF* macro to create a composite macro, where the I/O macro drives the global buffer (with programmable delay) using a hardwired connection. In this case, the software will automatically place the dedicated global I/O in the appropriate locations. Many specific INBUF macros support the wide variety of single-ended and differential I/O standards supported by the low power flash family. The available INBUF macros are described in the *IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide.*

The CLKDLY macro can be driven directly from the FPGA core. The CLKDLY macro can also be driven from an I/O that is routed through the FPGA regular routing fabric. In this case, users must instantiate a special macro, PLLINT, to differentiate the clock input driven by the hardwired I/O connection.

The visual CLKDLY configuration in the SmartGen area of the Microsemi Libero System-on-Chip (SoC) and Designer tools allows the user to select the desired amount of delay and configures the delay elements appropriately. SmartGen also allows the user to select the input clock source. SmartGen will automatically instantiate the special macro, PLLINT, when needed.

CLKDLY Macro Signal Descriptions

The CLKDLY macro supports one input and one output. Each signal is described in Table 3-2.

***Table 3-2* • Input and Output Description of the CLKDLY Macro**

Signal	Name	I/O	Description
CLK	Reference Clock	Input	Reference clock input
GL	Global Output	Output	Primary output clock to respective global/quadrant clock networks

CLKDLY Macro Usage

When a CLKDLY macro is used in a CCC location, the programmable delay element is used to allow the clock delays to go to the global network. In addition, the user can bypass the PLL in a CCC location integrated with a PLL, but use the programmable delay that is associated with the global network by instantiating the CLKDLY macro. The same is true when using programmable delay elements in a CCC location with no PLLs (the user needs to instantiate the CLKDLY macro). There is no difference between the programmable delay elements used for the PLL and the CLKDLY macro. The CCC will be configured to use the programmable delay elements in accordance with the macro instantiated by the user.

As an example, if the PLL is not used in a particular CCC location, the designer is free to specify up to three CLKDLY macros in the CCC, each of which can have its own input frequency and delay adjustment options. If the PLL core is used, assuming output to only one global clock network, the other two global clock networks are free to be used by either connecting directly from the global inputs or connecting from one or two CLKDLY macros for programmable delay.

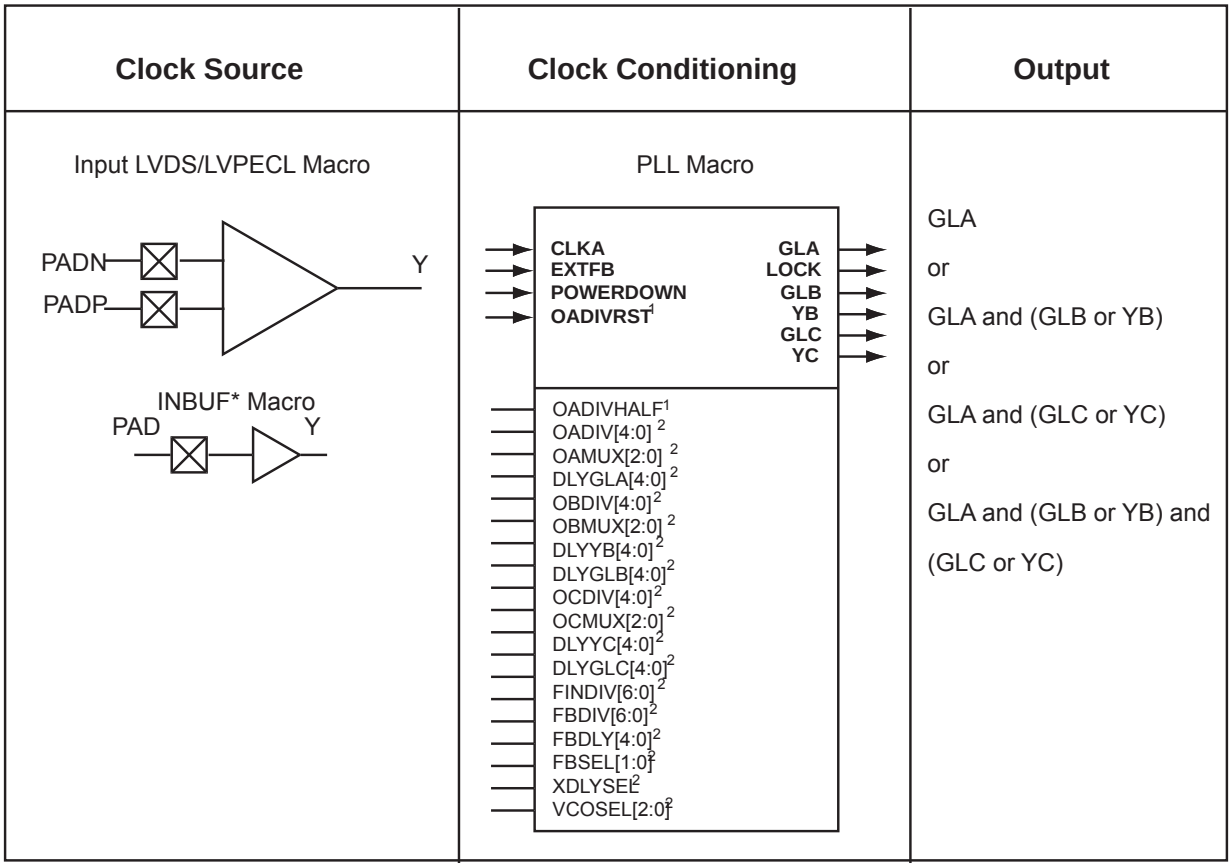
The programmable delay elements are shown in the block diagram of the PLL block shown in Figure 3-6 on page 63. Note that any CCC locations with no PLL present contain only the programmable delay blocks going to the global networks (labeled "Programmable Delay Type 2"). Refer to the "Clock Delay Adjustment" section on page 78 for a description of the programmable delay types used for the PLL. Also refer to Table 3-14 on page 86 for Programmable Delay Type 1 step delay values, and Table 3-15 on page 86 for Programmable Delay Type 2 step delay values. CCC locations with a PLL present can be configured to utilize only the programmable delay blocks (Programmable Delay Type 2) going to the global networks A, B, and C.

Global network A can be configured to use only the programmable delay element (bypassing the PLL) if the PLL is not used in the design. Figure 3-6 on page 63 shows a block diagram of the PLL, where the programmable delay elements are used for the global networks (Programmable Delay Type 2).

Global Buffers with PLL Function

Clocks requiring frequency synthesis or clock adjustments can utilize the PLL core before connecting to the global / quadrant global networks. A maximum of 18 CCC global buffers can be instantiated in a device—three per CCC and up to six CCCs per device. Each PLL core can generate up to three global/quadrant clocks, while a clock delay element provides one.

The PLL functionality of the clock conditioning block is supported by the PLL macro.



Notes:

1. *For Fusion only.*
2. *Refer to the IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide for more information.*
3. *For INBUF* driving a PLL macro or CLKDLY macro, the I/O will be hard-routed to the CCC; i.e., will be placed by software to a dedicated Global I/O.*
4. *IGLOO nano and ProASIC3 nano devices do not support differential inputs.*

***Figure 3-4* • CCC Options: Global Buffers with PLL**

The PLL macro provides five derived clocks (three independent) from a single reference clock. The PLL macro also provides power-down input and lock output signals. The additional inputs shown on the macro are configuration settings, which are configured through the use of SmartGen. For manual setting of these bits refer to the *IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide* for details.

Figure 3-6 on page 63 illustrates the various clock output options and delay elements.

PLL Macro Signal Descriptions

The PLL macro supports two inputs and up to six outputs. Table 3-3 gives a description of each signal.

***Table 3-3* • Input and Output Signals of the PLL Block**

Signal	Name	I/O	Description
CLKA	Reference Clock	Input	Reference clock input for PLL core; input clock for primary output clock, GLA
OADIVRST	Reset Signal for the Output Divider A	Input	For Fusion only. OADIVRST can be used when you bypass the PLL core (i.e., OAMUX = 001). The purpose of the OADIVRST signals is to reset the output of the final clock divider to synchronize it with the input to that divider when the PLL is bypassed. The signal is active on a low to high transition. The signal must be low for at least one divider input. If PLL core is used, this signal is "don't care" and the internal circuitry will generate the reset signal for the synchronization purpose.
OADIVHALF	Output A Division by Half	Input	For Fusion only. Active high. Division by half feature. This feature can only be used when users bypass the PLL core (i.e., OAMUX = 001) and the RC Oscillator (RCOSC) drives the CLKA input. This can be used to divide the 100 MHz RC oscillator by a factor of 1.5, 2.5, 3.5, 4.5 ... 14.5). Refer to Table 3-18 on page 87 for more information.
EXTFB	External Feedback	Input	Allows an external signal to be compared to a reference clock in the PLL core's phase detector.
POWERDOWN	Power Down	Input	Active low input that selects power-down mode and disables the PLL. With the POWERDOWN signal asserted, the PLL core sends 0 V signals on all of the outputs.
GLA	Primary Output	Output	Primary output clock to respective global/quadrant clock networks
GLB	Secondary 1 Output	Output	Secondary 1 output clock to respective global/quadrant clock networks
YB	Core 1 Output	Output	Core 1 output clock to local routing network
GLC	Secondary 2 Output	Output	Secondary 2 output clock to respective global/quadrant clock networks
YC	Core 2 Output	Output	Core 2 output clock to local routing network
LOCK	PLL Lock Indicator	Output	Active high signal indicating that steady-state lock has been achieved between CLKA and the PLL feedback signal

Input Clock

The inputs to the input reference clock (CLKA) of the PLL can come from global input pins, regular I/O pins, or internally from the core. For Fusion families, the input reference clock can also be from the embedded RC oscillator or crystal oscillator.

Global Output Clocks

GLA (Primary), GLB (Secondary 1), and GLC (Secondary 2) are the outputs of Global Multiplexer 1, Global Multiplexer 2, and Global Multiplexer 3, respectively. These signals (GLx) can be used to drive the high-speed global and quadrant networks of the low power flash devices.

A global multiplexer block consists of the input routing for selecting the input signal for the GLx clock and the output multiplexer, as well as delay elements associated with that clock.

Core Output Clocks

YB and YC are known as Core Outputs and can be used to drive internal logic without using global network resources. This is especially helpful when global network resources must be conserved and utilized for other timing-critical paths.

YB and YC are identical to GLB and GLC, respectively, with the exception of a higher selectable final output delay. The SmartGen PLL Wizard will configure these outputs according to user specifications and can enable these signals with or without the enabling of Global Output Clocks.

The above signals can be enabled in the following output groupings in both internal and external feedback configurations of the static PLL:

- One output – GLA only
- Two outputs – GLA + (GLB and/or YB)
- Three outputs – GLA + (GLB and/or YB) + (GLC and/or YC)

PLL Macro Block Diagram

As illustrated, the PLL supports three distinct output frequencies from a given input clock. Two of these (GLB and GLC) can be routed to the B and C global network access, respectively, and/or routed to the device core (YB and YC).

There are five delay elements to support phase control on all five outputs (GLA, GLB, GLC, YB, and YC).

There are delay elements in the feedback loop that can be used to advance the clock relative to the reference clock.

The PLL macro reference clock can be driven in the following ways:

1. By an INBUF* macro to create a composite macro, where the I/O macro drives the global buffer (with programmable delay) using a hardwired connection. In this case, the I/O must be placed in one of the dedicated global I/O locations.
2. Directly from the FPGA core.
3. From an I/O that is routed through the FPGA regular routing fabric. In this case, users must instantiate a special macro, PLLINT, to differentiate from the hardwired I/O connection described earlier.

During power-up, the PLL outputs will toggle around the maximum frequency of the voltage-controlled oscillator (VCO) gear selected. Toggle frequencies can range from 40 MHz to 250 MHz. This will continue as long as the clock input (CLKA) is constant (HIGH or LOW). This can be prevented by LOW assertion of the POWERDOWN signal.

The visual PLL configuration in SmartGen, a component of the Libero SoC and Designer tools, will derive the necessary internal divider ratios based on the input frequency and desired output frequencies selected by the user.

Implementing EXTFB in ProASIC3/E Devices

When the external feedback (EXTFB) signal of the PLL in the ProASIC3/E devices is implemented, the phase detector of the PLL core receives the reference clock (CLKA) and EXTFB as inputs. EXTFB must be sourced as an INBUF macro and located at the global/chip clock location associated with the target PLL by Designer software. EXTFB cannot be sourced from the FPGA fabric.

The following example shows CLKA and EXTFB signals assigned to two global I/Os in the same global area of ProASIC3E device.

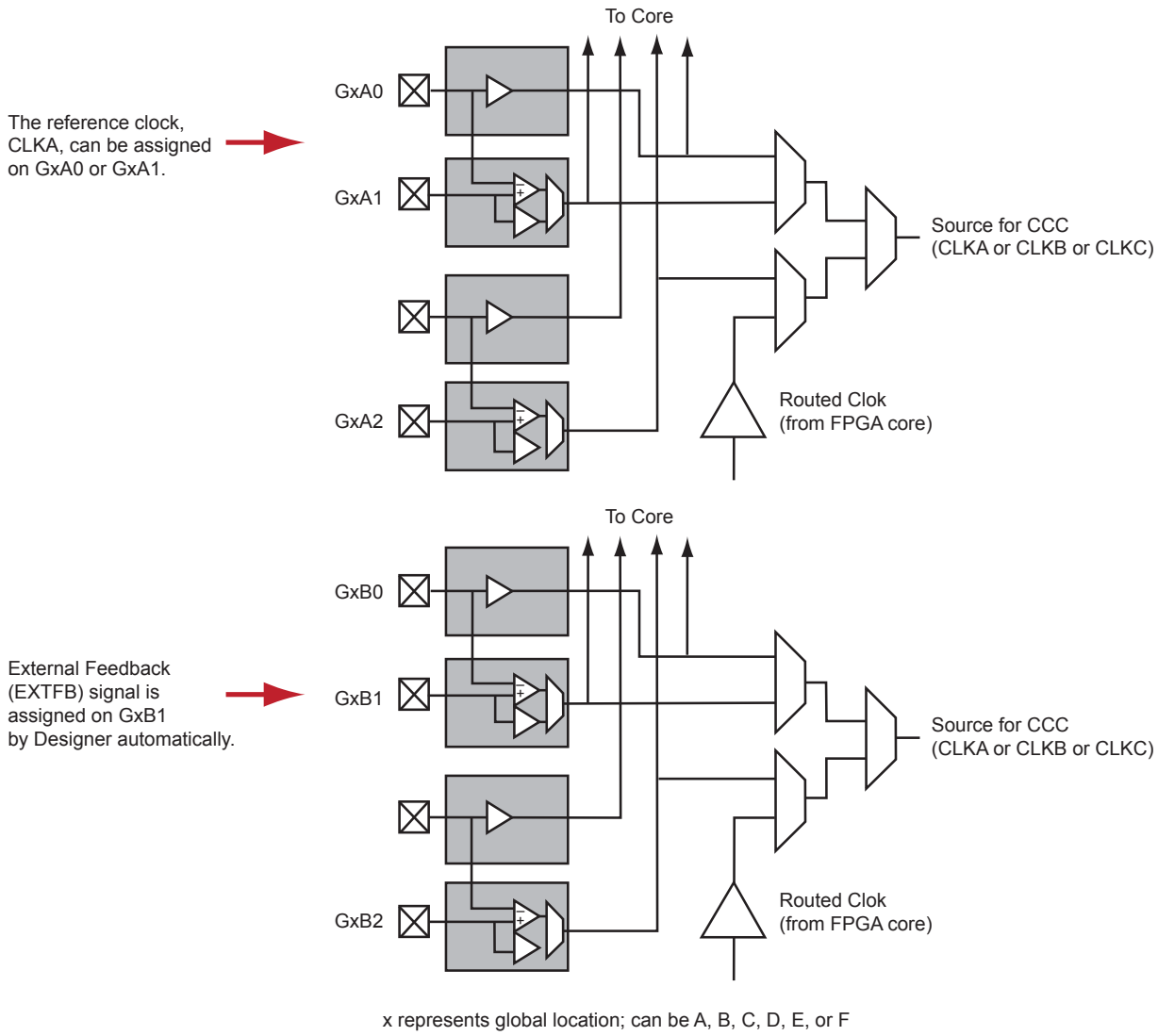
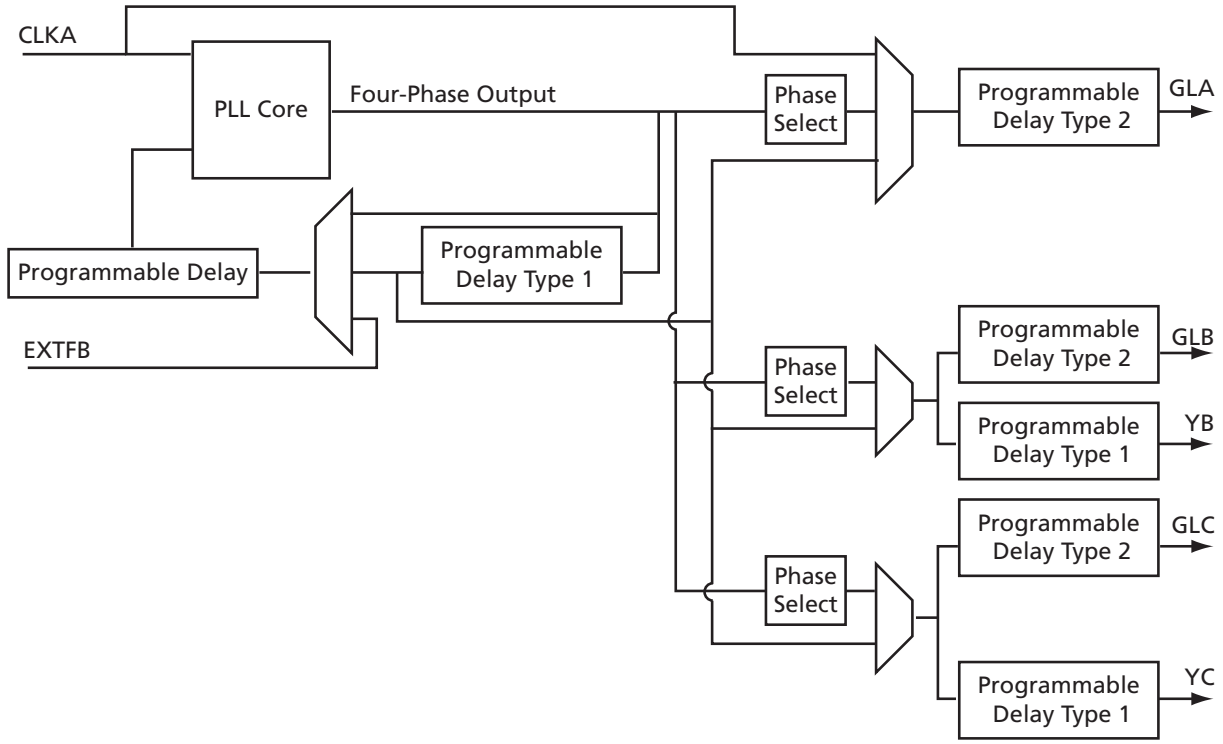


***Figure 3-5* • CLKA and EXTFB Assigned to Global I/Os**

SmartGen also allows the user to select the various delays and phase shift values necessary to adjust the phases between the reference clock (CLKA) and the derived clocks (GLA, GLB, GLC, YB, and YC). SmartGen allows the user to select the input clock source. SmartGen automatically instantiates the special macro, PLLINT, when needed.



Note: Clock divider and clock multiplier blocks are not shown in this figure or in SmartGen. They are automatically configured based on the user's required frequencies.

***Figure 3-6* • CCC with PLL Block**

Global Input Selections

Low power flash devices provide the flexibility of choosing one of the three global input pad locations available to connect to a CCC functional block or to a global / quadrant global network. Figure 3-7 on page 64 and Figure 3-8 on page 64 show the detailed architecture of each global input structure for 30 k gate devices and below, as well as 60 k gate devices and above, respectively. For 60 k gate devices and above (Figure 3-7 on page 64), if the single-ended I/O standard is chosen, there is flexibility to choose one of the global input pads (the first, second, and fourth input). Once chosen, the other I/O locations are used as regular I/Os. If the differential I/O standard is chosen (not applicable for IGLOO nano and ProASIC3 nano devices), the first and second inputs are considered as paired, and the third input is paired with a regular I/O.

The user then has the choice of selecting one of the two sets to be used as the clock input source to the CCC functional block. There is also the option to allow an internal clock signal to feed the global network or the CCC functional block. A multiplexer tree selects the appropriate global input for routing to the desired location. Note that the global I/O pads do not need to feed the global network; they can also be used as regular I/O pads.

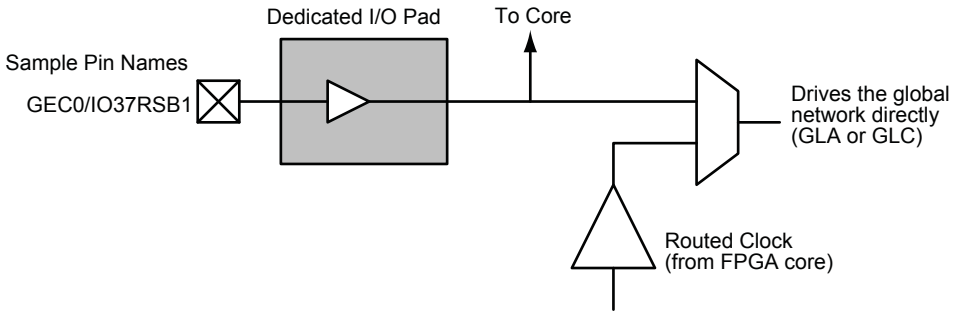
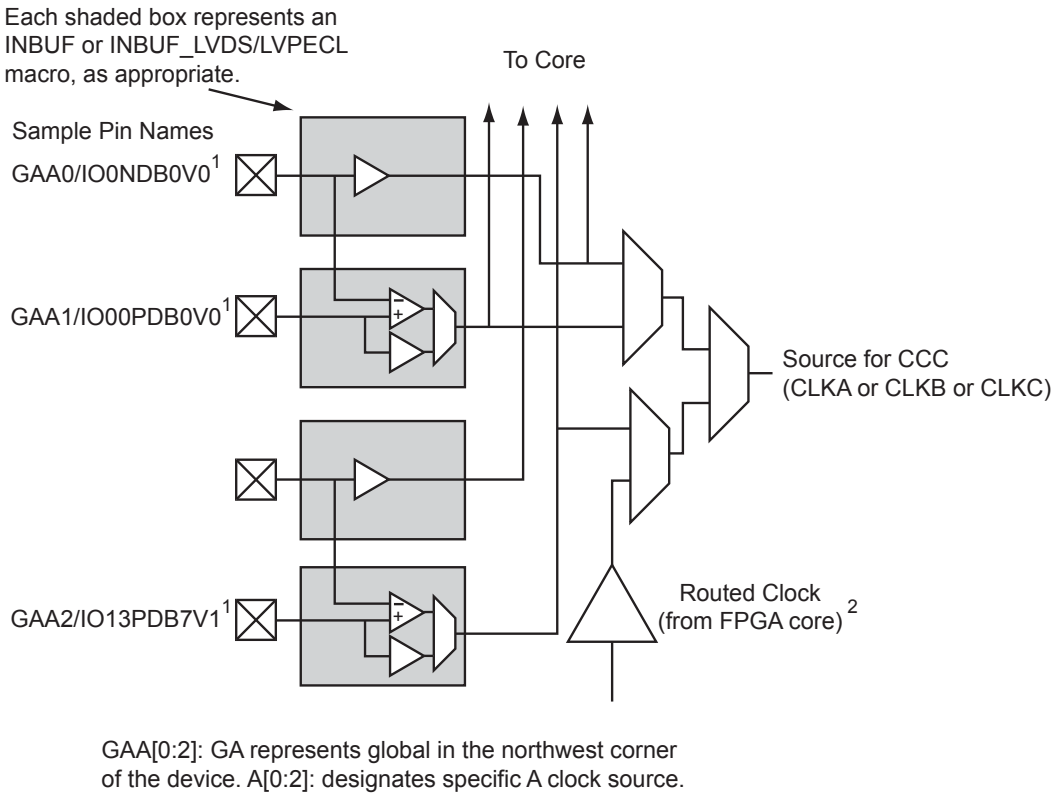


***Figure 3-7* • Clock Input Sources (30 k gates devices and below)**



Notes:

1. *Represents the global input pins. Globals have direct access to the clock conditioning block and are not routed via the FPGA fabric. Refer to the "User I/O Naming Conventions in I/O Structures" chapter of the appropriate device user's guide.*
2. *Instantiate the routed clock source input as follows:*
   a) *Connect the output of a logic element to the clock input of a PLL, CLKDLY, or CLKINT macro.*
   b) *Do not place a clock source I/O (INBUF or INBUF_LVPECL/LVDS/B-LVDS/M-LVDS/DDR) in a relevant global pin location.*
3. *IGLOO nano and ProASIC3 nano devices do not support differential inputs.*

***Figure 3-8* • Clock Input Sources Including CLKBUF, CLKBUF_LVDS/LVPECL, and CLKINT (60 k gates devices and above)**

Each global buffer, as well as the PLL reference clock, can be driven from one of the following:

- 3 dedicated single-ended I/Os using a hardwired connection
- 2 dedicated differential I/Os using a hardwired connection (not applicable for IGLOO nano and ProASIC3 nano devices)
- The FPGA core

Since the architecture of the devices varies as size increases, the following list details I/O types supported for globals:

IGLOO and ProASIC3

- LVDS-based clock sources are available only on 250 k gate devices and above (IGLOO nano and ProASIC3 nano devices do not support differential inputs).
- 60 k and 125 k gate devices support single-ended clock sources only.
- 15 k and 30 k gate devices support these inputs for CCC only and do not contain a PLL.
- nano devices:
  - 10 k, 15 k, and 20 k devices do not contain PLLs in the CCCs, and support only CLKBUF and CLKINT.
  - 60 k, 125 k, and 250 k devices support one PLL in the middle left CCC position. In the absence of the PLL, this CCC can be used by CLKBUF, CLKINT, and CLKDLY macros. The corner CCCs support CLKBUF, CLKINT, and CLKDLY.

Fusion

- AFS600 and AFS1500: All single-ended, differential, and voltage-referenced I/O standards (Pro I/O).
- AFS090 and AFS250: All single-ended and differential I/O standards.

Clock Sources for PLL and CLKDLY Macros

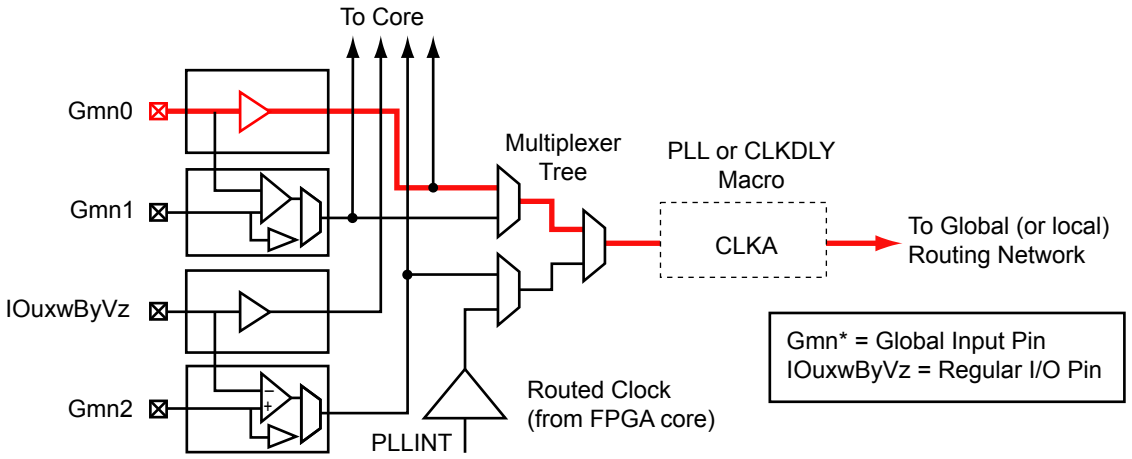
The input reference clock (CLKA for a PLL macro, CLK for a CLKDLY macro) can be accessed from different sources via the associated clock multiplexer tree. Each CCC has the option of choosing the source of the input clock from one of the following:

- Hardwired I/O
- External I/O
- Core Logic
- RC Oscillator (Fusion only)
- Crystal Oscillator (Fusion only)

The SmartGen macro builder tool allows users to easily create the PLL and CLKDLY macros with the desired settings. Microsemi strongly recommends using SmartGen to generate the CCC macros.

Hardwired I/O Clock Source

Hardwired I/O refers to global input pins that are hardwired to the multiplexer tree, which directly accesses the CCC global buffers. These global input pins have designated pin locations and are indicated with the I/O naming convention *Gmn* (*m* refers to any one of the positions where the PLL core is available, and *n* refers to any one of the three global input MUXes and the pin number of the associated global location, *m*). Choosing this option provides the benefit of directly connecting to the CCC reference clock input, which provides less delay. See Figure 3-9 on page 66 for an example illustration of the connections, shown in red. If a CLKDLY macro is initiated to utilize the programmable delay element of the CCC, the clock input can be placed at one of nine dedicated global input pin locations. In other words, if Hardwired I/O is chosen as the input source, the user can decide to place the input pin in one of the GmA0, GmA1, GmA2, GmB0, GmB1, GmB2, GmC0, GmC1, or GmC2 locations of the low power flash devices. When a PLL macro is used to utilize the PLL core in a CCC location, the clock input of the PLL can only be connected to one of three GmA* global pin locations: GmA0, GmA1, or GmA2.



Note: Fusion CCCs have additional source selections (RCOSC, XTAL).

***Figure 3-9* • Illustration of Hardwired I/O (global input pins) Usage for IGLOO and ProASIC3 devices 60 k Gates and Larger**

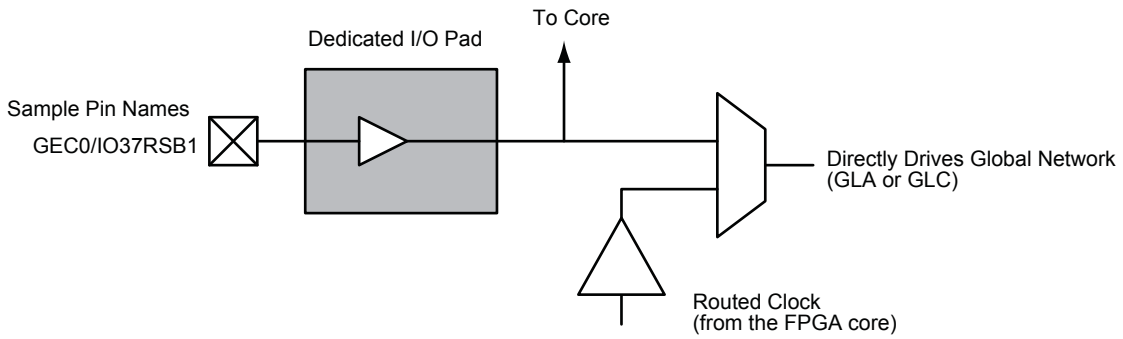


***Figure 3-10* • Illustration of Hardwired I/O (global input pins) Usage for IGLOO and ProASIC3 devices 30 k Gates and Smaller**

External I/O Clock Source

External I/O refers to regular I/O pins. The clock source is instantiated with one of the various INBUF options and accesses the CCCs via internal routing. The user has the option of assigning this input to any of the I/Os labeled with the I/O convention *IOuxwByVz*. Refer to the "User I/O Naming Conventions in I/O Structures" chapter of the appropriate device user's guide, and for Fusion, refer to the *Fusion Family of Mixed Signal FPGAs* datasheet for more information. Figure 3-11 gives a brief explanation of external I/O usage. Choosing this option provides the freedom of selecting any user I/O location but introduces additional delay because the signal connects to the routed clock input through internal routing before connecting to the CCC reference clock input.

For the External I/O option, the routed signal would be instantiated with a PLLINT macro before connecting to the CCC reference clock input. This instantiation is conveniently done automatically by SmartGen when this option is selected. Microsemi recommends using the SmartGen tool to generate the CCC macro. The instantiation of the PLLINT macro results in the use of the routed clock input of the I/O to connect to the PLL clock input. If not using SmartGen, manually instantiate a PLLINT macro before the PLL reference clock to indicate that the regular I/O driving the PLL reference clock should be used (see Figure 3-11 for an example illustration of the connections, shown in red).

In the above two options, the clock source must be instantiated with one of the various INBUF macros. The reference clock pins of the CCC functional block core macros must be driven by regular input macros (INBUFs), not clock input macros.

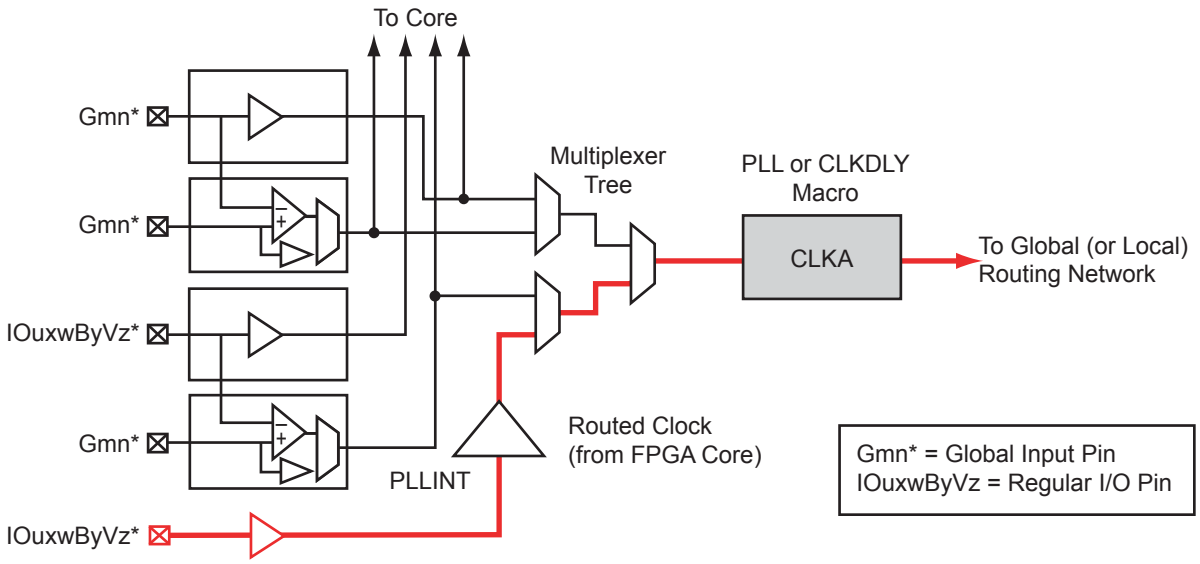


***Figure 3-11* • Illustration of External I/O Usage**

For Fusion devices, the input reference clock can also be from the embedded RC oscillator and crystal oscillator. In this case, the CCC configuration is the same as the hardwired I/O clock source, and users are required to instantiate the RC oscillator or crystal oscillator macro and connect its output to the input reference clock of the CCC block.

Core Logic Clock Source

Core logic refers to internal routed nets. Internal routed signals access the CCC via the FPGA Core Fabric. Similar to the External I/O option, whenever the clock source comes internally from the core itself, the routed signal is instantiated with a PLLINT macro before connecting to the CCC clock input (see Figure 3-12 for an example illustration of the connections, shown in red).

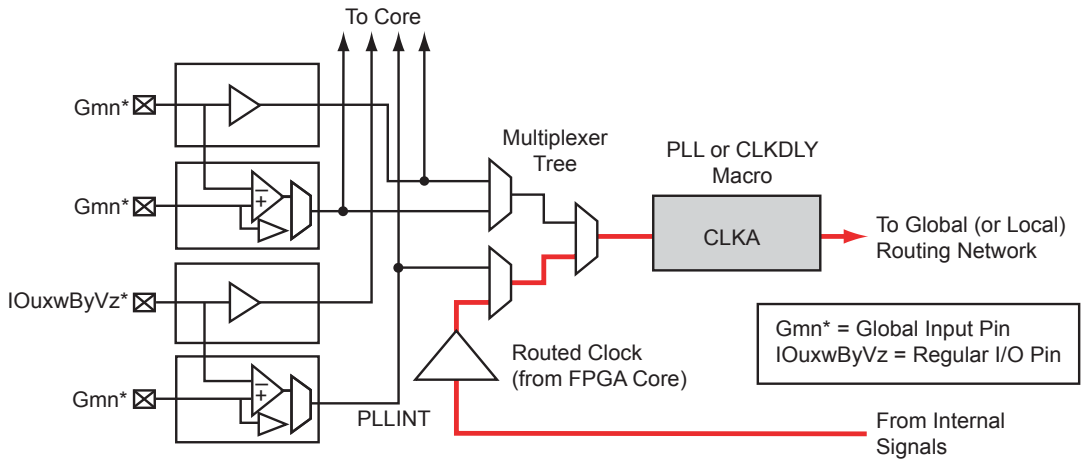


***Figure 3-12* • Illustration of Core Logic Usage**

For Fusion devices, the input reference clock can also be from the embedded RC oscillator and crystal oscillator. In this case, the CCC configuration is the same as the hardwired I/O clock source, and users are required to instantiate the RC oscillator or crystal oscillator macro and connect its output to the input reference clock of the CCC block.

Available I/O Standards

***Table 3-4* • Available I/O Standards within CLKBUF and CLKBUF_LVDS/LVPECL Macros**

CLKBUF_LVCMOS5
CLKBUF_LVCMOS33 [1]
CLKBUF_LVCMOS25 [2]
CLKBUF_LVCMOS18
CLKBUF_LVCMOS15
CLKBUF_PCI
CLKBUF_PCIX [3]
CLKBUF_GTL25 [2,3]
CLKBUF_GTL33 [2,3]
CLKBUF_GTLP25 [2,3]
CLKBUF_GTLP33 [2,3]
CLKBUF_HSTL_I [2,3]
CLKBUF_HSTL_II [2,3]
CLKBUF_SSTL3_I [2,3]
CLKBUF_SSTL3_II [2,3]
CLKBUF_SSTL2_I [2,3]
CLKBUF_SSTL2_II [2,3]
CLKBUF_LVDS [4,5]
CLKBUF_LVPECL[5]

Notes:

1. *By default, the CLKBUF macro uses 3.3 V LVTTL I/O technology. For more details, refer to the IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide.*
2. *I/O standards only supported in ProASIC3E and IGLOOe families.*
3. *I/O standards only supported in the following Fusion devices: AFS600 and AFS1500.*
4. *B-LVDS and M-LVDS standards are supported by CLKBUF_LVDS.*
5. *Not supported for IGLOO nano and ProASIC3 nano devices.*

Global Synthesis Constraints

The Synplify® synthesis tool, by default, allows six clocks in a design for Fusion, IGLOO, and ProASIC3. When more than six clocks are needed in the design, a user synthesis constraint attribute, syn_global_buffers, can be used to control the maximum number of clocks (up to 18) that can be inferred by the synthesis engine.

High-fanout nets will be inferred with clock buffers and/or internal clock buffers. If the design consists of CCC global buffers, they are included in the count of clocks in the design.

The subsections below discuss the clock input source (global buffers with no programmable delays) and the clock conditioning functional block (global buffers with programmable delays and/or PLL function) in detail.

Device-Specific Layout

Two kinds of CCCs are offered in low power flash devices: CCCs with integrated PLLs, and CCCs without integrated PLLs (simplified CCCs). Table 3-5 lists the number of CCCs in various devices.

***Table 3-5* • Number of CCCs by Device Size and Package**

Device		Package	CCCs with Integrated PLLs	CCCs without Integrated PLLs (simplified CCC)
ProASIC3	**IGLOO**			
A3PN010	AGLN010	All	0	2
A3PN015	AGLN015	All	0	2
A3PN020	AGLN020	All	0	2
	AGLN060	CS81	0	6
A3PN060	AGLN060	All other packages	1	5
	AGLN125	CS81	0	6
A3PN125	AGLN125	All other packages	1	5
	AGLN250	CS81	0	6
A3PN250	AGLN250	All other packages	1	5
A3P015	AGL015	All	0	2
A3P030	AGL030/AGLP030	All	0	2
	AGL060/AGLP060	CS121/CS201	0	6
A3P060	AGL060/AGLP060	All other packages	1	5
A3P125	AGL125/AGLP125	All	1	5
A3P250/L	AGL250	All	1	5
A3P400	AGL400	All	1	5
A3P600/L	AGL600	All	1	5
A3P1000/L	AGL1000	All	1	5
A3PE600	AGLE600	PQ208	2	4
A3PE600/L		All other packages	6	0
A3PE1500		PQ208	2	4
A3PE1500		All other packages	6	0
A3PE3000/L		PQ208	2	4
A3PE3000/L	AGLE3000	All other packages	6	0
Fusion Devices				
AFS090		All	1	5
AFS250, M1AFS250		All	1	5
AFS600, M7AFS600, M1AFS600		All	2	4
AFS1500, M1AFS1500		All	2	4

Note: nano 10 k, 15 k, and 20 k offer 6 global MUXes instead of CCCs.

This section outlines the following device information: CCC features, PLL core specifications, functional descriptions, software configuration information, detailed usage information, recommended board-level considerations, and other considerations concerning global networks in low power flash devices.

Clock Conditioning Circuits with Integrated PLLs

Each of the CCCs with integrated PLLs includes the following:

- 1 PLL core, which consists of a phase detector, a low-pass filter, and a four-phase voltage-controlled oscillator
- 3 global multiplexer blocks that steer signals from the global pads and the PLL core onto the global networks
- 6 programmable delays and 1 fixed delay for time advance/delay adjustments
- 5 programmable frequency divider blocks to provide frequency synthesis (automatically configured by the SmartGen macro builder tool)

Clock Conditioning Circuits without Integrated PLLs

There are two types of simplified CCCs without integrated PLLs in low power flash devices.

1. The simplified CCC with programmable delays, which is composed of the following:
   – 3 global multiplexer blocks that steer signals from the global pads and the programmable delay elements onto the global networks
   – 3 programmable delay elements to provide time delay adjustments
2. The simplified CCC (referred to as CCC-GL) without programmable delay elements, which is composed of the following:
   – A global multiplexer block that steer signals from the global pads onto the global networks

CCC Locations

CCCs located in the middle of the east and west sides of the device access the three VersaNet global networks on each side (six total networks), while the four CCCs located in the four corners access three quadrant global networks (twelve total networks). See Figure 3-13.

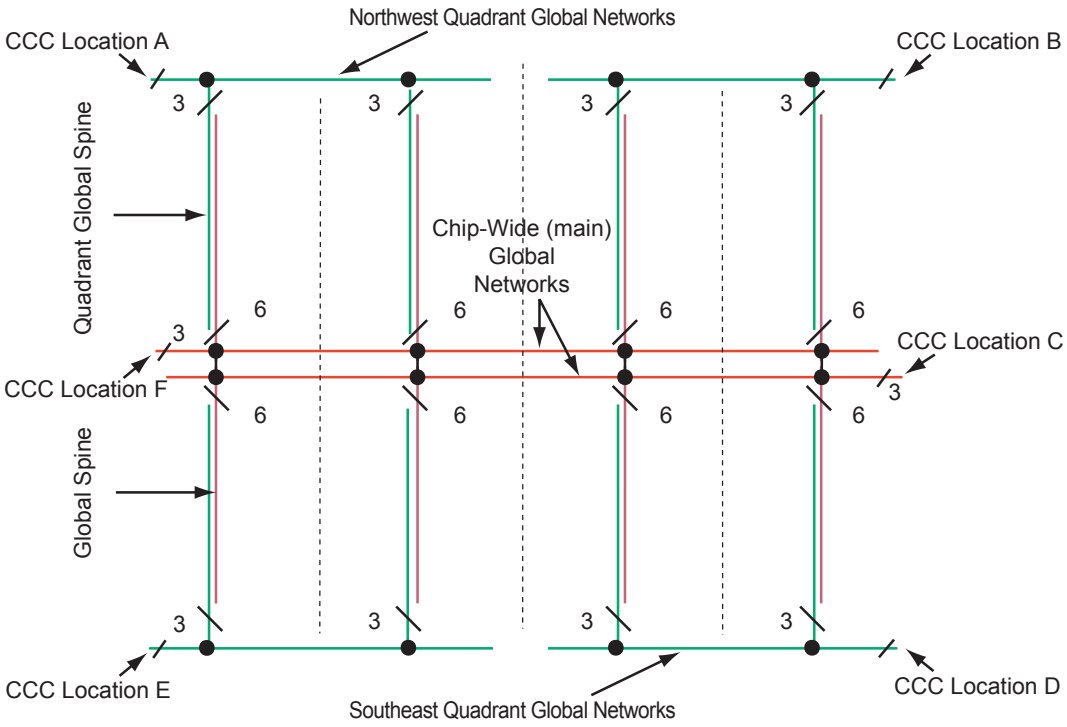


***Figure 3-13* • Global Network Architecture for 60 k Gate Devices and Above**

The following explains the locations of the CCCs in IGLOO and ProASIC3 devices:

In Figure 3-15 on page 74 through Figure 3-16 on page 74, CCCs with integrated PLLs are indicated in red, and simplified CCCs are indicated in yellow. There is a letter associated with each location of the CCC, in clockwise order. The upper left corner CCC is named "A," the upper right is named "B," and so on. These names finish up at the middle left with letter "F."

IGLOO and ProASIC3 CCC Locations

In all IGLOO and ProASIC3 devices (except 10 k through 30 k gate devices, which do not contain PLLs), six CCCs are located in the same positions as the IGLOOe and ProASIC3E CCCs. Only one of the CCCs has an integrated PLL and is located in the middle of the west (middle left) side of the device. The other five CCCs are simplified CCCs and are located in the four corners and the middle of the east side of the device (Figure 3-14).

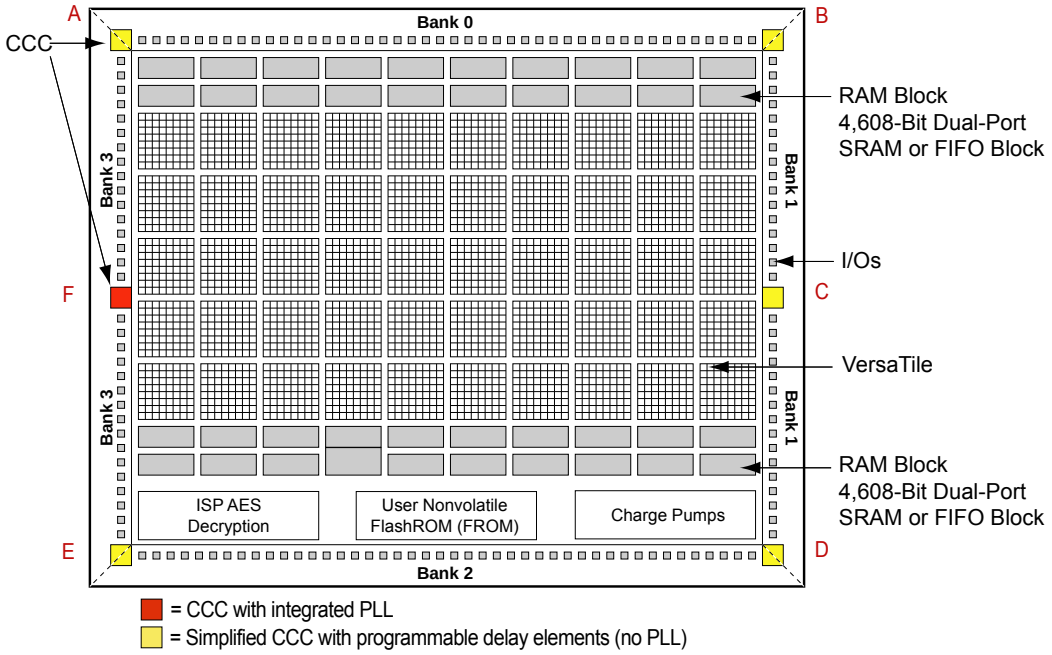


***Figure 3-14* • CCC Locations in IGLOO and ProASIC3 Family Devices (except 10 k through 30 k gate devices)**

Note: The number and architecture of the banks are different for some devices.

10 k through 30 k gate devices do not support PLL features. In these devices, there are two CCC-GLs at the lower corners (one at the lower right, and one at the lower left). These CCC-GLs do not have programmable delays.

IGLOOe and ProASIC3E CCC Locations

IGLOOe and ProASIC3E devices have six CCCs—one in each of the four corners and one each in the middle of the east and west sides of the device (Figure 3-15).

All six CCCs are integrated with PLLs, except in PQFP-208 package devices. PQFP-208 package devices also have six CCCs, of which two include PLLs and four are simplified CCCs. The CCCs with PLLs are implemented in the middle of the east and west sides of the device (middle right and middle left). The simplified CCCs without PLLs are located in the four corners of the device (Figure 3-16).

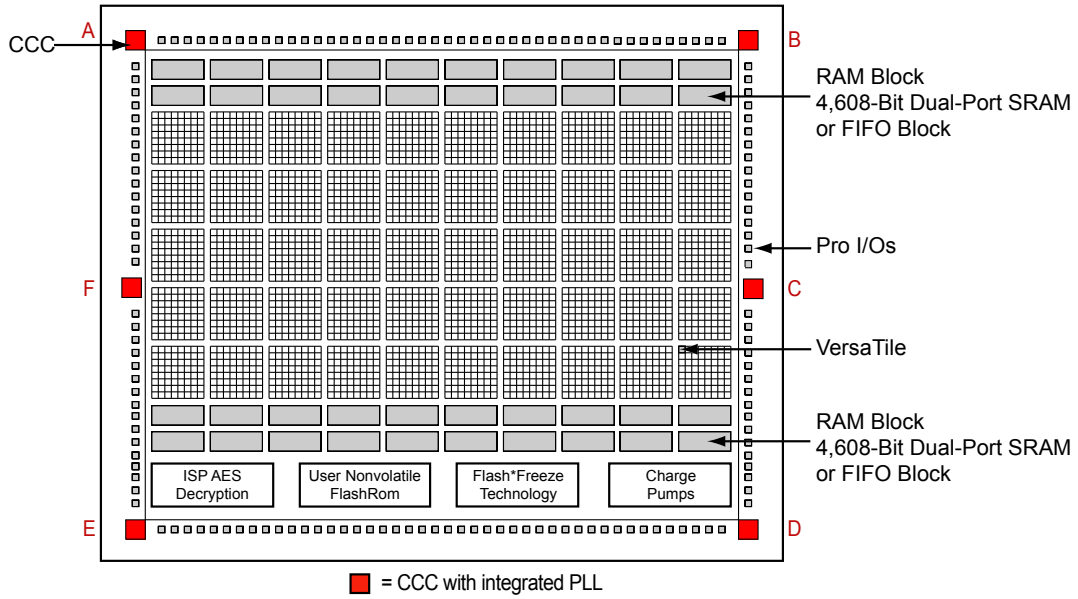


***Figure 3-15* • CCC Locations in IGLOOe and ProASIC3E Family Devices (except PQFP-208 package)**

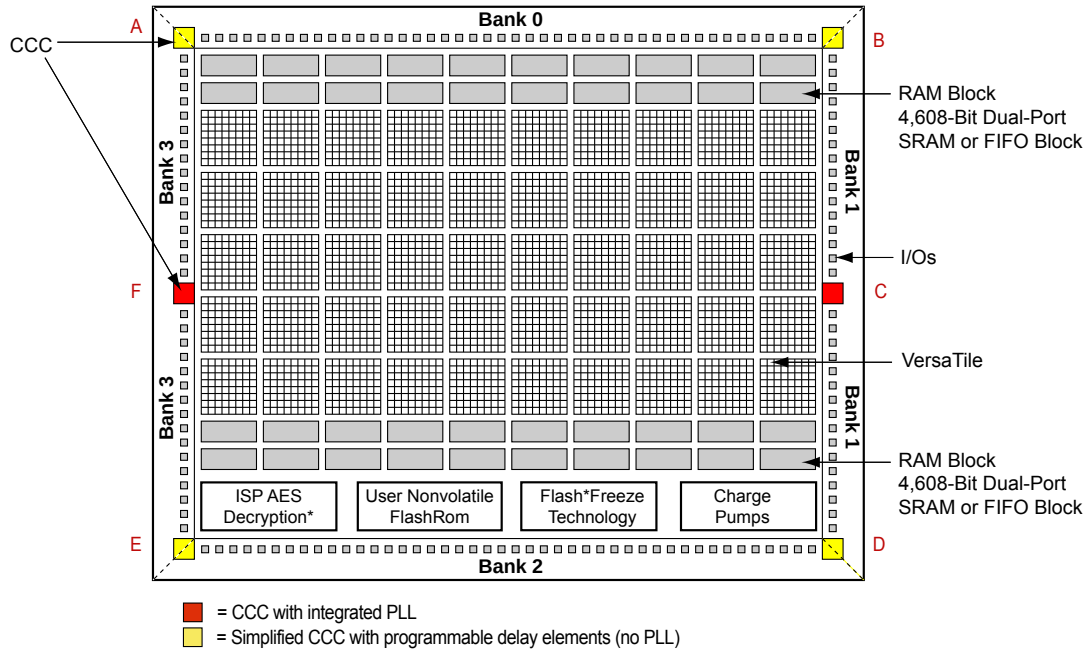


***Figure 3-16* • CCC Locations in ProASIC3E Family Devices (PQFP-208 package)**

Fusion CCC Locations

Fusion devices have six CCCs: one in each of the four corners and one each in the middle of the east and west sides of the device (Figure 3-17 and Figure 3-18). The device can have one integrated PLL in the middle of the west side of the device or two integrated PLLs in the middle of the east and west sides of the device (middle right and middle left).

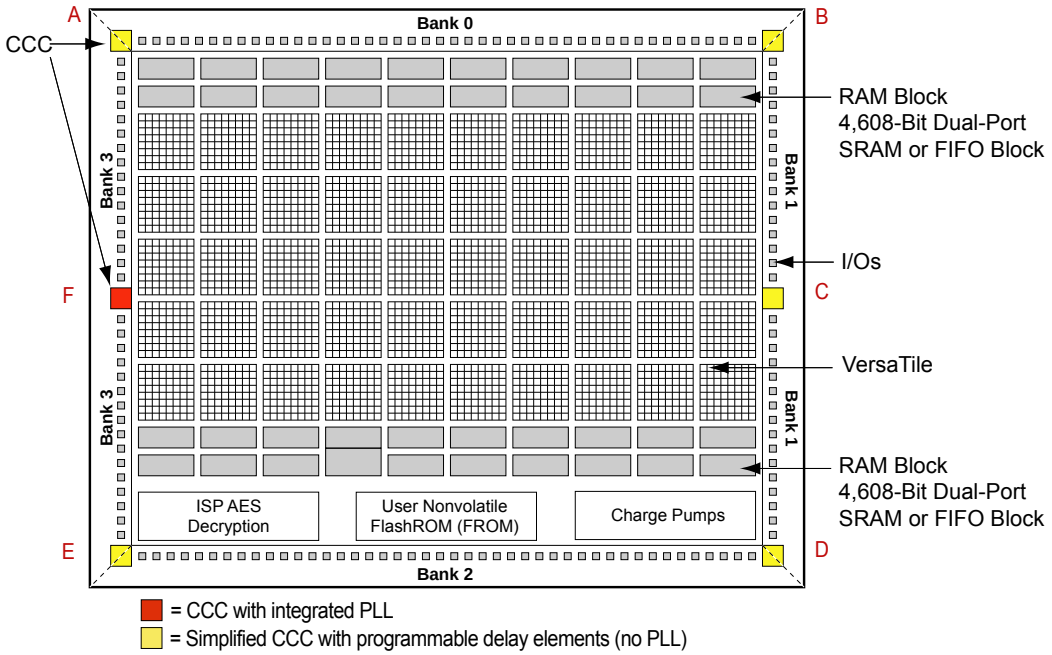


***Figure 3-17* • CCC Locations in Fusion Family Devices (AFS090, AFS250, M1AFS250)**

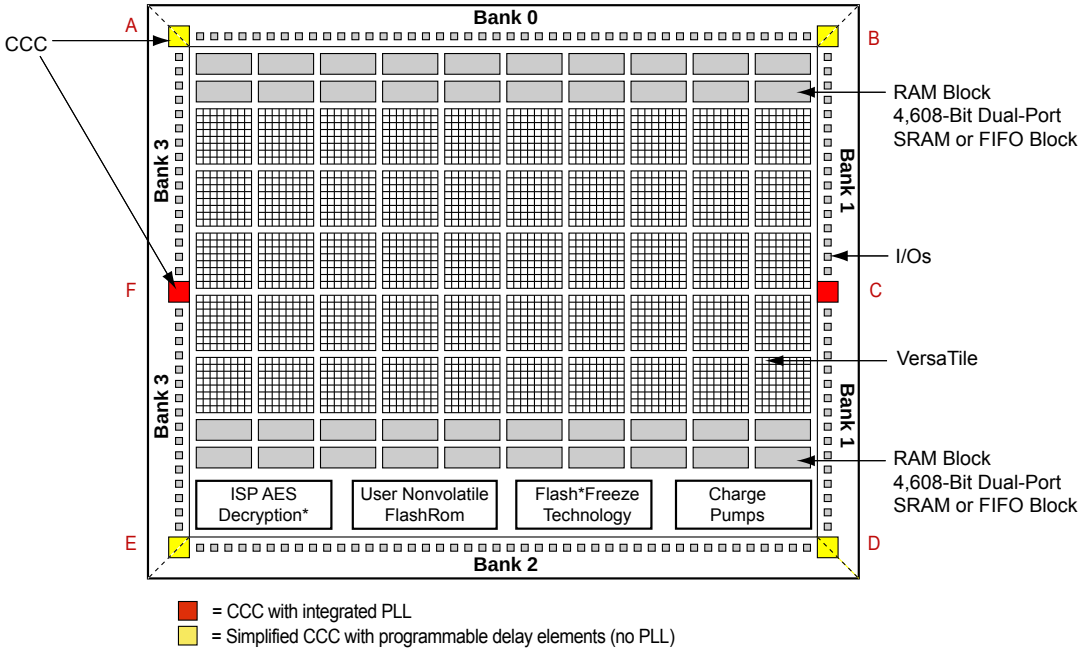


***Figure 3-18* • CCC Locations in Fusion Family Devices (except AFS090, AFS250, M1AFS250)**

PLL Core Specifications

PLL core specifications can be found in the DC and Switching Characteristics chapter of the appropriate family datasheet.

Loop Bandwidth

Common design practice for systems with a low-noise input clock is to have PLLs with small loop bandwidths to reduce the effects of noise sources at the output. Table 3-6 shows the PLL loop bandwidth, providing a measure of the PLL's ability to track the input clock and jitter.

***Table 3-6 •* –3 dB Frequency of the PLL**

	Minimum (T_a = +125°C, VCCA = 1.4 V)	Typical (T_a = +25°C, VCCA = 1.5 V)	Maximum (T_a = –55°C, VCCA = 1.6 V)
–3 dB Frequency	15 kHz	25 kHz	45 kHz

PLL Core Operating Principles

This section briefly describes the basic principles of PLL operation. The PLL core is composed of a phase detector (PD), a low-pass filter (LPF), and a four-phase voltage-controlled oscillator (VCO). Figure 3-19 illustrates a basic single-phase PLL core with a divider and delay in the feedback path.

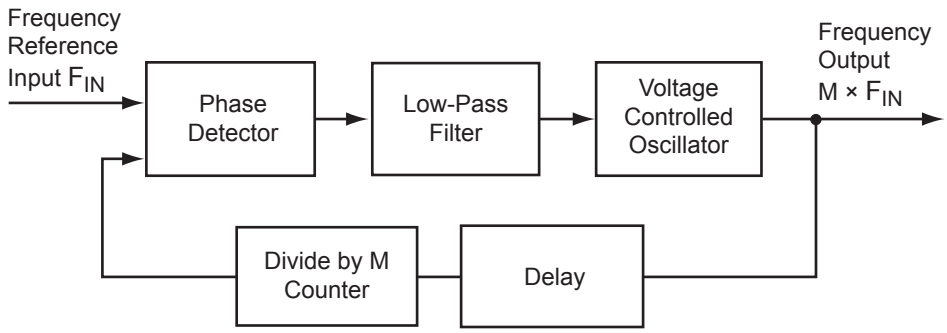


***Figure 3-19 •* Simplified PLL Core with Feedback Divider and Delay**

The PLL is an electronic servo loop that phase-aligns the PD feedback signal with the reference input. To achieve this, the PLL dynamically adjusts the VCO output signal according to the average phase difference between the input and feedback signals.

The first element is the PD, which produces a voltage proportional to the phase difference between its inputs. A simple example of a digital phase detector is an Exclusive-OR gate. The second element, the LPF, extracts the average voltage from the phase detector and applies it to the VCO. This applied voltage alters the resonant frequency of the VCO, thus adjusting its output frequency.

Consider Figure 3-19 with the feedback path bypassing the divider and delay elements. If the LPF steadily applies a voltage to the VCO such that the output frequency is identical to the input frequency, this steady-state condition is known as lock. Note that the input and output phases are also identical. The PLL core sets a LOCK output signal HIGH to indicate this condition.

Should the input frequency increase slightly, the PD detects the frequency/phase difference between its reference and feedback input signals. Since the PD output is proportional to the phase difference, the change causes the output from the LPF to increase. This voltage change increases the resonant frequency of the VCO and increases the feedback frequency as a result. The PLL dynamically adjusts in this manner until the PD senses two phase-identical signals and steady-state lock is achieved. The opposite (decreasing PD output signal) occurs when the input frequency decreases.

Now suppose the feedback divider is inserted in the feedback path. As the division factor M (shown in Figure 3-20 on page 77) is increased, the average phase difference increases. The average phase

difference will cause the VCO to increase its frequency until the output signal is phase-identical to the input after undergoing division. In other words, lock in both frequency and phase is achieved when the output frequency is M times the input. Thus, clock division in the feedback path results in multiplication at the output.

A similar argument can be made when the delay element is inserted into the feedback path. To achieve steady-state lock, the VCO output signal will be delayed by the input period *less* the feedback delay. For periodic signals, this is equivalent to time-advancing the output clock by the feedback delay.

Another key parameter of a PLL system is the acquisition time. Acquisition time is the amount of time it takes for the PLL to achieve lock (i.e., phase-align the feedback signal with the input reference clock). For example, suppose there is no voltage applied to the VCO, allowing it to operate at its free-running frequency. Should an input reference clock suddenly appear, a lock would be established within the maximum acquisition time.

Functional Description

This section provides detailed descriptions of PLL block functionality: clock dividers and multipliers, clock delay adjustment, phase adjustment, and dynamic PLL configuration.

Clock Dividers and Multipliers

The PLL block contains five programmable dividers. Figure 3-20 shows a simplified PLL block.

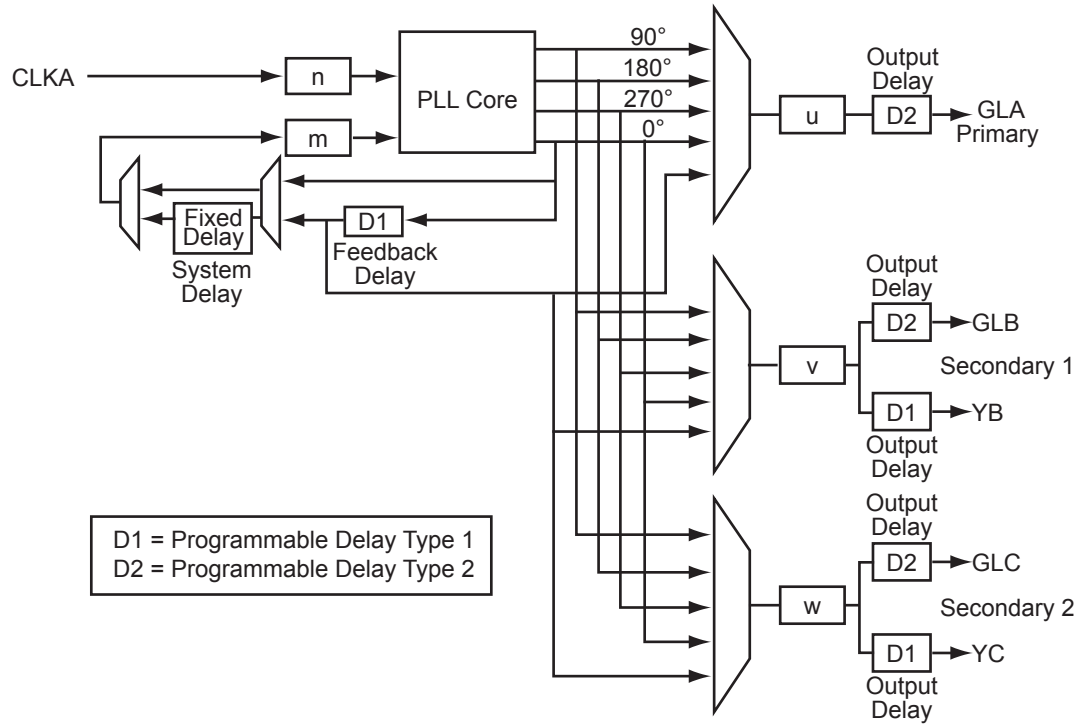


***Figure 3-20* • PLL Block Diagram**

Dividers *n* and *m* (the input divider and feedback divider, respectively) provide integer frequency division factors from 1 to 128. The output dividers *u*, *v*, and *w* provide integer division factors from 1 to 32. Frequency scaling of the reference clock CLKA is performed according to the following formulas:

$$f_{GLA} = f_{CLKA} \times m / (n \times u) \text{ – GLA Primary PLL Output Clock}$$

EQ 3-1

$$f_{GLB} = f_{YB} = f_{CLKA} \times m / (n \times v) \text{ – GLB Secondary 1 PLL Output Clock(s)}$$

EQ 3-2

$$f_{GLC} = f_{YC} = f_{CLKA} \times m / (n \times w) \text{ – GLC Secondary 2 PLL Output Clock(s)}$$

EQ 3-3

SmartGen provides a user-friendly method of generating the configured PLL netlist, which includes automatically setting the division factors to achieve the closest possible match to the requested frequencies. Since the five output clocks share the *n* and *m* dividers, the achievable output frequencies are interdependent and related according to the following formula:

$$f_{GLA} = f_{GLB} \times (v / u) = f_{GLC} \times (w / u)$$

EQ 3-4

Clock Delay Adjustment

There are a total of seven configurable delay elements implemented in the PLL architecture.

Two of the delays are located in the feedback path, entitled System Delay and Feedback Delay. System Delay provides a fixed delay of 2 ns (typical), and Feedback Delay provides selectable delay values from 0.6 ns to 5.56 ns in 160 ps increments (typical). For PLLs, delays in the feedback path will effectively advance the output signal from the PLL core with respect to the reference clock. Thus, the System and Feedback delays generate negative delay on the output clock. Additionally, each of these delays can be independently bypassed if necessary.

The remaining five delays perform traditional time delay and are located at each of the outputs of the PLL. Besides the fixed global driver delay of 0.755 ns for each of the global networks, the global multiplexer outputs (GLA, GLB, and GLC) each feature an additional selectable delay value, as given in Table 3-7.

***Table 3-7* • Delay Values in Libero SoC Software per Device Family**

Device	Typical	Starting Values	Increments	Ending Value
ProASIC3	200 ps	0 to 735 ps	200 ps	6.735 ns
IGLOO/ProASIC3L 1.5 V	360 ps	0 to 1.610 ns	360 ps	12.410 ns
IGLOO/ProASIC3L 1.2 V	580 ps	0 to 2.880 ns	580 ps	20.280 ns

The additional YB and YC signals have access to a selectable delay from 0.6 ns to 5.56 ns in 160 ps increments (typical). This is the same delay value as the CLKDLY macro. It is similar to CLKDLY, which bypasses the PLL core just to take advantage of the phase adjustment option with the delay value.

The following parameters must be taken into consideration to achieve minimum delay at the outputs (GLA, GLB, GLC, YB, and YC) relative to the reference clock: routing delays from the PLL core to CCC outputs, core outputs and global network output delays, and the feedback path delay. The feedback path delay acts as a time advance of the input clock and will offset any delays introduced beyond the PLL core output. The routing delays are determined from back-annotated simulation and are configuration-dependent.

Phase Adjustment

The four phases available (0, 90, 180, 270) are phases with respect to VCO (PLL output). The VCO is divided to achieve the user's CCC required output frequency (GLA, YB/GLB, YC/GLC). The division happens after the selection of the VCO phase. The effective phase shift is actually the VCO phase shift divided by the output divider. This is why the visual CCC shows both the actual achievable phase and more importantly the actual delay that is equivalent to the phase shift that can be achieved.

Dynamic PLL Configuration

The CCCs can be configured both statically and dynamically.

In addition to the ports available in the Static CCC, the Dynamic CCC has the dynamic shift register signals that enable dynamic reconfiguration of the CCC. With the Dynamic CCC, the ports CLKB and CLKC are also exposed. All three clocks (CLKA, CLKB, and CLKC) can be configured independently.

The CCC block is fully configurable. The following two sources can act as the CCC configuration bits.

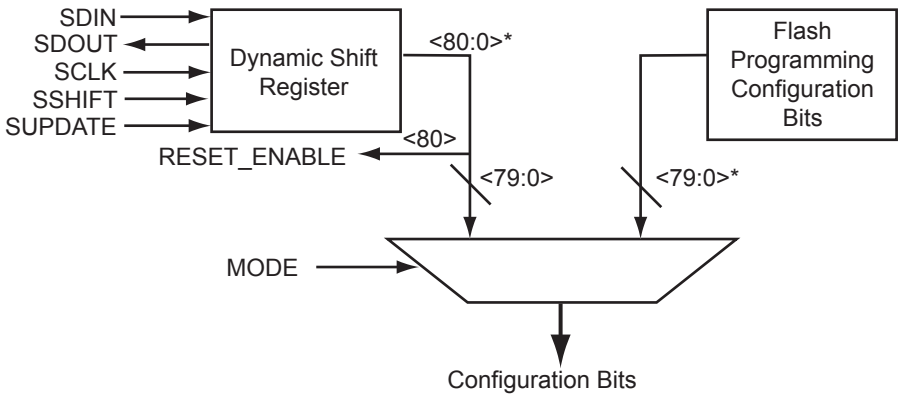
Flash Configuration Bits

The flash configuration bits are the configuration bits associated with programmed flash switches. These bits are used when the CCC is in static configuration mode. Once the device is programmed, these bits cannot be modified. They provide the default operating state of the CCC.

Dynamic Shift Register Outputs

This source does not require core reprogramming and allows core-driven dynamic CCC reconfiguration. When the dynamic register drives the configuration bits, the user-defined core circuit takes full control over SDIN, SDOUT, SCLK, SSHIFT, and SUPDATE. The configuration bits can consequently be dynamically changed through shift and update operations in the serial register interface. Access to the logic core is accomplished via the dynamic bits in the specific tiles assigned to the PLLs.

Figure 3-21 illustrates a simplified block diagram of the MUX architecture in the CCCs.



*Note: *For Fusion, bit <88:81> is also needed.*

***Figure 3-21* • The CCC Configuration MUX Architecture**

The selection between the flash configuration bits and the bits from the configuration register is made using the MODE signal shown in Figure 3-21. If the MODE signal is logic HIGH, the dynamic shift register configuration bits are selected. There are 81 control bits to configure the different functions of the CCC.

Each group of control bits is assigned a specific location in the configuration shift register. For a list of the 81 configuration bits (C[80:0]) in the CCC and a description of each, refer to "PLL Configuration Bits Description" on page 82. The configuration register can be serially loaded with the new configuration data and programmed into the CCC using the following ports:

- SDIN: The configuration bits are serially loaded into a shift register through this port. The LSB of the configuration data bits should be loaded first.
- SDOUT: The shift register contents can be shifted out (LSB first) through this port using the shift operation.
- SCLK: This port should be driven by the shift clock.
- SSHIFT: The active-high shift enable signal should drive this port. The configuration data will be shifted into the shift register if this signal is HIGH. Once SSHIFT goes LOW, the data shifting will be halted.
- SUPDATE: The SUPDATE signal is used to configure the CCC with the new configuration bits when shifting is complete.

To access the configuration ports of the shift register (SDIN, SDOUT, SSHIFT, etc.), the user should instantiate the CCC macro in his design with appropriate ports. Microsemi recommends that users choose SmartGen to generate the CCC macros with the required ports for dynamic reconfiguration.

Users must familiarize themselves with the architecture of the CCC core and its input, output, and configuration ports to implement the desired delay and output frequency in the CCC structure. Figure 3-22 shows a model of the CCC with configurable blocks and switches.

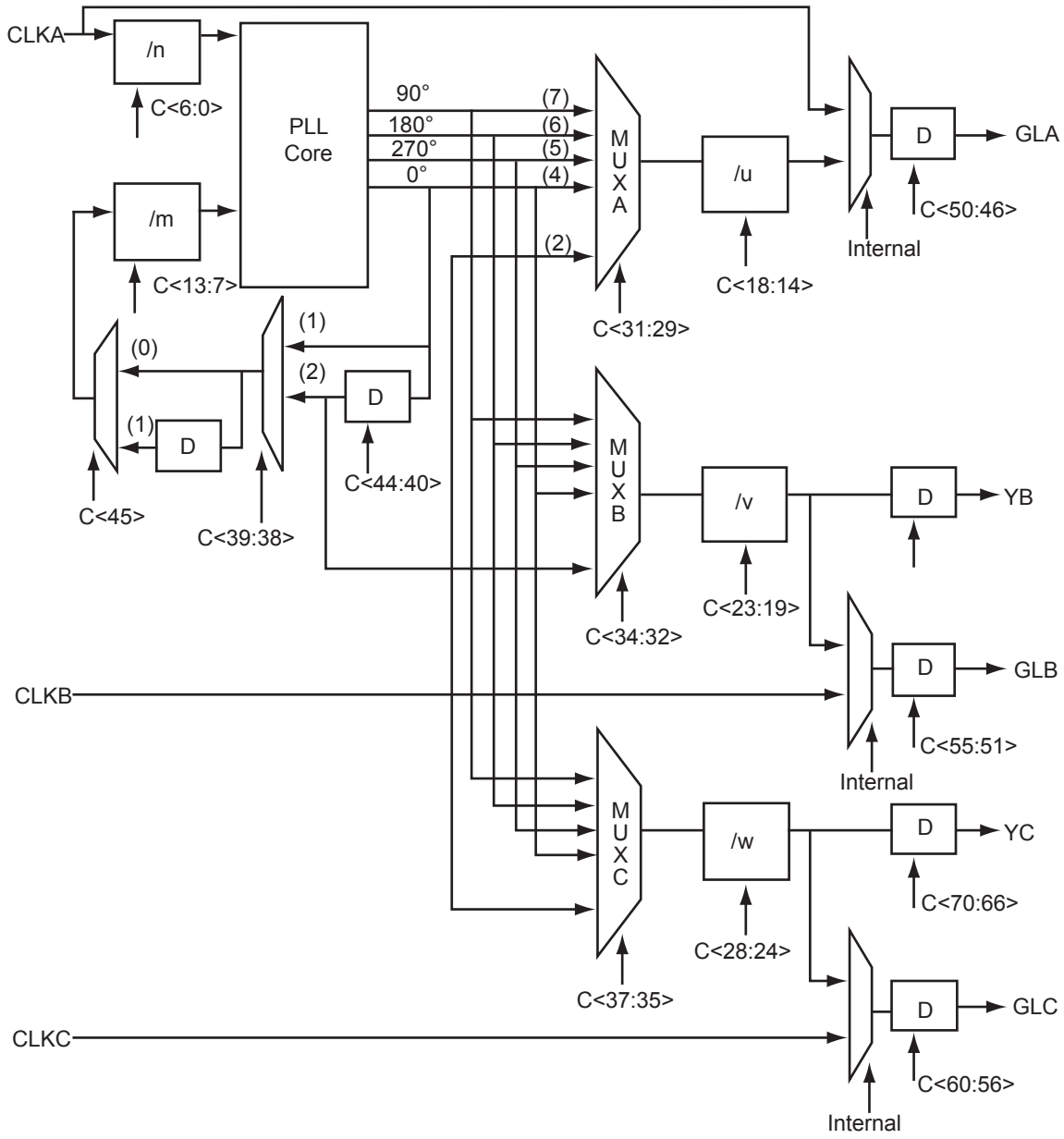


***Figure 3-22* • CCC Block Control Bits – Graphical Representation of Assignments**

Loading the Configuration Register

The most important part of CCC dynamic configuration is to load the shift register properly with the configuration bits. There are different ways to access and load the configuration shift register:

- JTAG interface
- Logic core
- Specific I/O tiles

JTAG Interface

The JTAG interface requires no additional I/O pins. The JTAG TAP controller is used to control the loading of the CCC configuration shift register.

Low power flash devices provide a user interface macro between the JTAG pins and the device core logic. This macro is called UJTAG. A user should instantiate the UJTAG macro in his design to access the configuration register ports via the JTAG pins.

For more information on CCC dynamic reconfiguration using UJTAG, refer to the "UJTAG Applications in Microsemi's Low Power Flash Devices" section on page 419.

Logic Core

If the logic core is employed, the user must design a module to provide the configuration data and control the shifting and updating of the CCC configuration shift register. In effect, this is a user-designed TAP controller, which requires additional chip resources.

Specific I/O Tiles

If specific I/O tiles are used for configuration, the user must provide the external equivalent of a TAP controller. This does not require additional core resources but does use pins.

Shifting the Configuration Data

To enter a new configuration, all 81 bits must shift in via SDIN. After all bits are shifted, SSHIFT must go LOW and SUPDATE HIGH to enable the new configuration. For simulation purposes, bits <71:73> and <77:80> are "don't care."

The SUPDATE signal must be LOW during any clock cycle where SSHIFT is active. After SUPDATE is asserted, it must go back to the LOW state until a new update is required.

PLL Configuration Bits Description

***Table 3-8* • Configuration Bit Descriptions for the CCC Blocks**

Config. Bits	Signal	Name	Description
<88:87>	GLMUXCFG [1:0][1]	NGMUX configuration	The configuration bits specify the input clocks to the NGMUX (refer to Table 3-17 on page 86).[2]
86	OCDIVHALF[1]	Division by half	When the PLL is bypassed, the 100 MHz RC oscillator can be divided by the divider factor in Table 3-18 on page 87.
85	OBDIVHALF[1]	Division by half	When the PLL is bypassed, the 100 MHz RC oscillator can be divided by a 0.5 factor (refer to Table 3-18 on page 87).
84	OADIVHALF[1]	Division by half	When the PLL is bypassed, the 100 MHz RC oscillator can be divided by certain 0.5 factor (refer to Table 3-16 on page 86).

Notes:

1. *The <88:81> configuration bits are only for the Fusion dynamic CCC.*
2. *This value depends on the input clock source, so Layout must complete before these bits can be set. After completing Layout in Designer, generate the "CCC_Configuration" report by choosing **Tools** > **Report** > **CCC_Configuration**. The report contains the appropriate settings for these bits.*

***Table 3-8* • Configuration Bit Descriptions for the CCC Blocks (continued)**

Config. Bits	Signal	Name	Description
83	RXCSEL[1]	CLKC input selection	Select the CLKC input clock source between RC oscillator and crystal oscillator (refer to Table 3-16 on page 86).[2]
82	RXBSEL[1]	CLKB input selection	Select the CLKB input clock source between RC oscillator and crystal oscillator (refer to Table 3-16 on page 86).[2]
81	RXASEL[1]	CLKA input selection	Select the CLKA input clock source between RC oscillator and crystal oscillator (refer to Table 3-16 on page 86).[2]
80	RESETEN	Reset Enable	Enables (active high) the synchronization of PLL output dividers after dynamic reconfiguration (SUPDATE). The Reset Enable signal is READ-ONLY.
79	DYNCSEL	Clock Input C Dynamic Select	Configures clock input C to be sent to GLC for dynamic control.[2]
78	DYNBSEL	Clock Input B Dynamic Select	Configures clock input B to be sent to GLB for dynamic control.[2]
77	DYNASEL	Clock Input A Dynamic Select	Configures clock input A for dynamic PLL configuration.[2]
<76:74>	VCOSEL[2:0]	VCO Gear Control	Three-bit VCO Gear Control for four frequency ranges (refer to Table 3-19 on page 87 and Table 3-20 on page 87).
73	STATCSEL	MUX Select on Input C	MUX selection for clock input C[2]
72	STATBSEL	MUX Select on Input B	MUX selection for clock input B[2]
71	STATASEL	MUX Select on Input A	MUX selection for clock input A[2]
<70:66>	DLYC[4:0]	YC Output Delay	Sets the output delay value for YC.
<65:61>	DLYB[4:0]	YB Output Delay	Sets the output delay value for YB.
<60:56>	DLYGLC[4:0]	GLC Output Delay	Sets the output delay value for GLC.
<55:51>	DLYGLB[4:0]	GLB Output Delay	Sets the output delay value for GLB.
<50:46>	DLYGLA[4:0]	Primary Output Delay	Primary GLA output delay
45	XDLYSEL	System Delay Select	When selected, inserts System Delay in the feedback path in Figure 3-20 on page 77.
<44:40>	FBDLY[4:0]	Feedback Delay	Sets the feedback delay value for the feedback element in Figure 3-20 on page 77.
<39:38>	FBSEL[1:0]	Primary Feedback Delay Select	Controls the feedback MUX: no delay, include programmable delay element, or use external feedback.
<37:35>	OCMUX[2:0]	Secondary 2 Output Select	Selects from the VCO's four phase outputs for GLC/YC.
<34:32>	OBMUX[2:0]	Secondary 1 Output Select	Selects from the VCO's four phase outputs for GLB/YB.

Notes:

1. *The <88:81> configuration bits are only for the Fusion dynamic CCC.*
2. *This value depends on the input clock source, so Layout must complete before these bits can be set. After completing Layout in Designer, generate the "CCC_Configuration" report by choosing* ***Tools*** *>* ***Report*** *>* ***CCC_Configuration****. The report contains the appropriate settings for these bits.*

***Table 3-8* • Configuration Bit Descriptions for the CCC Blocks (continued)**

Config. Bits	Signal	Name	Description
<31:29>	OAMUX[2:0]	GLA Output Select	Selects from the VCO's four phase outputs for GLA.
<28:24>	OCDIV[4:0]	Secondary 2 Output Divider	Sets the divider value for the GLC/YC outputs. Also known as divider *w* in Figure 3-20 on page 77. The divider value will be OCDIV[4:0] + 1.
<23:19>	OBDIV[4:0]	Secondary 1 Output Divider	Sets the divider value for the GLB/YB outputs. Also known as divider *v* in Figure 3-20 on page 77. The divider value will be OBDIV[4:0] + 1.
<18:14>	OADIV[4:0]	Primary Output Divider	Sets the divider value for the GLA output. Also known as divider *u* in Figure 3-20 on page 77. The divider value will be OADIV[4:0] + 1.
<13:7>	FBDIV[6:0]	Feedback Divider	Sets the divider value for the PLL core feedback. Also known as divider *m* in Figure 3-20 on page 77. The divider value will be FBDIV[6:0] + 1.
<6:0>	FINDIV[6:0]	Input Divider	Input Clock Divider (/n). Sets the divider value for the input delay on CLKA. The divider value will be FINDIV[6:0] + 1.

Notes:

1. *The <88:81> configuration bits are only for the Fusion dynamic CCC.*
2. *This value depends on the input clock source, so Layout must complete before these bits can be set. After completing Layout in Designer, generate the "CCC_Configuration" report by choosing* ***Tools*** *>* ***Report*** *>* ***CCC_Configuration****. The report contains the appropriate settings for these bits.*

Table 3-9 to Table 3-15 on page 86 provide descriptions of the configuration data for the configuration bits.

***Table 3-9* • Input Clock Divider, FINDIV[6:0] (/n)**

FINDIV<6:0> State	Divisor	New Frequency Factor
0	1	1.00000
1	2	0.50000
⋮	⋮	⋮
127	128	0.0078125

***Table 3-10* • Feedback Clock Divider, FBDIV[6:0] (/m)**

FBDIV<6:0> State	Divisor	New Frequency Factor
0	1	1
1	2	2
⋮	⋮	⋮
127	128	128

***Table 3-11* • Output Frequency Dividers**
***A* Output Divider, OADIV <4:0> (/u);**
***B* Output Divider, OBDIV <4:0> (/v);**
***C* Output Divider, OCDIV <4:0> (/w)**

OADIV<4:0>; OBDIV<4:0>; CDIV<4:0> State	Divisor	New Frequency Factor
0	1	1.00000
1	2	0.50000
⋮	⋮	⋮
31	32	0.03125

***Table 3-12* • MUXA, MUXB, MUXC**

OAMUX<2:0>; OBMUX<2:0>; OCMUX<2:0> State	MUX Input Selected
0	None. Six-input MUX and PLL are bypassed. Clock passes only through global MUX and goes directly into HC ribs.
1	Not available
2	PLL feedback delay line output
3	Not used
4	PLL VCO 0° phase shift
5	PLL VCO 270° phase shift
6	PLL VCO 180° phase shift
7	PLL VCO 90° phase shift

***Table 3-13* • 2-Bit Feedback MUX**

FBSEL<1:0> State	MUX Input Selected
0	Ground. Used for power-down mode in power-down logic block.
1	PLL VCO 0° phase shift
2	PLL delayed VCO 0° phase shift
3	N/A

***Table 3-14* • Programmable Delay Selection for Feedback Delay and Secondary Core Output Delays**

FBDLY<4:0>; DLYYB<4:0>; DLYYC<4:0> State	Delay Value
0	Typical delay = 600 ps
1	Typical delay = 760 ps
2	Typical delay = 920 ps
⋮	⋮
31	Typical delay = 5.56 ns

***Table 3-15* • Programmable Delay Selection for Global Clock Output Delays**

DLYGLA<4:0>; DLYGLB<4:0>; DLYGLC<4:0> State	Delay Value
0	Typical delay = 225 ps
1	Typical delay = 760 ps
2	Typical delay = 920 ps
⋮	⋮
31	Typical delay = 5.56 ns

***Table 3-16* • Fusion Dynamic CCC Clock Source Selection**

RXASEL	DYNASEL	Source of CLKA
1	0	RC Oscillator
1	1	Crystal Oscillator
RXBSEL	**DYNBSEL**	**Source of CLKB**
1	0	RC Oscillator
1	1	Crystal Oscillator
RXBSEL	**DYNCSEL**	**Source of CLKC**
1	0	RC Oscillator
1	1	Crystal Oscillator

***Table 3-17* • Fusion Dynamic CCC NGMUX Configuration**

GLMUXCFG<1:0>	NGMUX Select Signal	Supported Input Clocks to NGMUX
00	0	GLA
	1	GLC
01	0	GLA
	1	GLINT
10	0	GLC
	1	GLINT

***Table 3-18* • Fusion Dynamic CCC Division by Half Configuration**

OADIVHALF / OBDIVHALF / OCDIVHALF	OADIV<4:0> / OBDIV<4:0> / OCDIV<4:0> (in decimal)	Divider Factor	Input Clock Frequency	Output Clock Frequency (MHz)
1	2	1.5	100 MHz RC Oscillator	66.7
	4	2.5		40.0
	6	3.5		28.6
	8	4.5		22.2
	10	5.5		18.2
	12	6.5		15.4
	14	7.5		13.3
	16	8.5		11.8
	18	9.5		10.5
	20	10.5		9.5
	22	11.5		8.7
	24	12.5		8.0
	26	13.5		7.4
	28	14.5		6.9
0	0–31	1–32	Other Clock Sources	Depends on other divider settings

***Table 3-19* • Configuration Bit <76:75> / VCOSEL<2:1> Selection for All Families**

	VCOSEL[2:1]							
	00		01		10		11	
Voltage	Min. (MHz)	Max. (MHz)	Min. (MHz)	Max. (MHz)	Min. (MHz)	Max. (MHz)	Min. (MHz)	Max. (MHz)
IGLOO and IGLOO PLUS								
1.2 V ± 5%	24	35	30	70	60	140	135	160
1.5 V ± 5%	24	43.75	30	87.5	60	175	135	250
ProASIC3L, RT ProASIC3, and Military ProASIC3/L								
1.2 V ± 5%	24	35	30	70	60	140	135	250
1.5 V ± 5%	24	43.75	30	70	60	175	135	350
ProASIC3 and Fusion								
1.5 V ± 5%	24	43.75	33.75	87.5	67.5	175	135	350

***Table 3-20* • Configuration Bit <74> / VCOSEL<0> Selection for All Families**

VCOSEL[0]	Description
0	Fast PLL lock acquisition time with high tracking jitter. Refer to the corresponding datasheet for specific value and definition.
1	Slow PLL lock acquisition time with low tracking jitter. Refer to the corresponding datasheet for specific value and definition.

Software Configuration

SmartGen automatically generates the desired CCC functional block by configuring the control bits, and allows the user to select two CCC modes: Static PLL and Delayed Clock (CLKDLY).

Static PLL Configuration

The newly implemented Visual PLL Configuration Wizard feature provides the user a quick and easy way to configure the PLL with the desired settings (Figure 3-23). The user can invoke SmartGen to set the parameters and generate the netlist file with the appropriate flash configuration bits set for the CCCs. As mentioned in "PLL Macro Block Diagram" on page 61, the input reference clock CLKA can be configured to be driven by Hardwired I/O, External I/O, or Core Logic. The user enters the desired settings for all the parameters (output frequency, output selection, output phase adjustment, clock delay, feedback delay, and system delay). Notice that the actual values (divider values, output frequency, delay values, and phase) are shown to aid the user in reaching the desired design frequency in real time. These values are typical-case data. Best- and worst-case data can be observed through static timing analysis in SmartTime within Designer.

For dynamic configuration, the CCC parameters are defined using either the external JTAG port or an internally defined serial interface via the built-in dynamic shift register. This feature provides the ability to compensate for changes in the external environment.

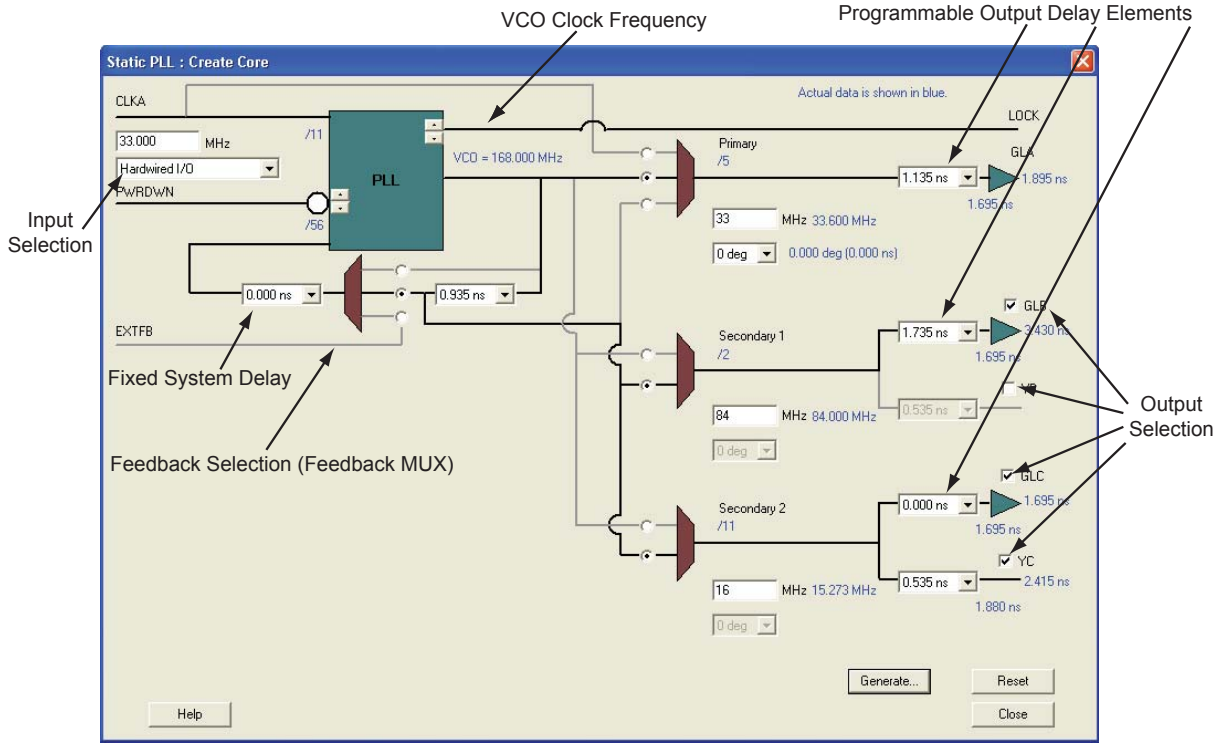


Figure 3-23 • **Visual PLL Configuration Wizard**

Feedback Configuration

The PLL provides both internal and external feedback delays. Depending on the configuration, various combinations of feedback delays can be achieved.

Internal Feedback Configuration

This configuration essentially sets the feedback multiplexer to route the VCO output of the PLL core as the input to the feedback of the PLL. The feedback signal can be processed with the fixed system and the adjustable feedback delay, as shown in Figure 3-24. The dividers are automatically configured by SmartGen based on the user input.

Indicated below is the System Delay pull-down menu. The System Delay can be bypassed by setting it to 0. When set, it adds a 2 ns delay to the feedback path (which results in delay advancement of the output clock by 2 ns).

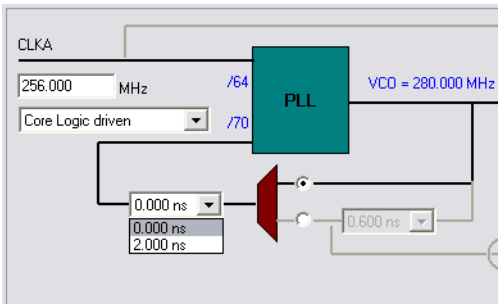


Figure 3-24 • **Internal Feedback with Selectable System Delay**

Figure 3-25 shows the controllable Feedback Delay. If set properly in conjunction with the fixed System Delay, the total output delay can be advanced significantly.

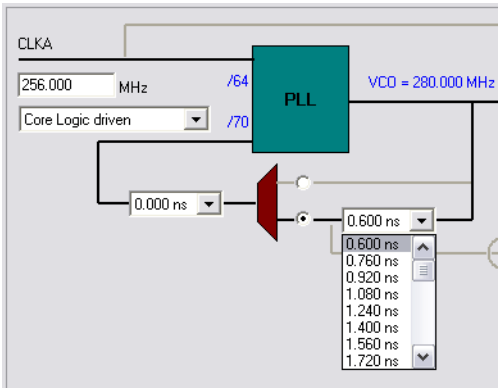


Figure 3-25 • **Internal Feedback with Selectable Feedback Delay**

External Feedback Configuration

For certain applications, such as those requiring generation of PCB clocks that must be matched with existing board delays, it is useful to implement an external feedback, EXTFB. The Phase Detector of the PLL core will receive CLKA and EXTFB as inputs. EXTFB may be processed by the fixed System Delay element as well as the *M* divider element. The EXTFB option is currently not supported.

After setting all the required parameters, users can generate one or more PLL configurations with HDL or EDIF descriptions by clicking the **Generate** button. SmartGen gives the option of saving session results and messages in a log file:

```
****************
Macro Parameters
****************

Name                            : test_pll
Family                          : ProASIC3E
Output Format                   : VHDL
Type                            : Static PLL
Input Freq(MHz)                 : 10.000
CLKA Source                     : Hardwired I/O
Feedback Delay Value Index      : 1
Feedback Mux Select             : 2
XDLY Mux Select                 : No
Primary Freq(MHz)               : 33.000
Primary PhaseShift              : 0
Primary Delay Value Index       : 1
Primary Mux Select              : 4
Secondary1 Freq(MHz)            : 66.000
Use GLB                         : YES
Use YB                          : YES
GLB Delay Value Index           : 1
YB Delay Value Index            : 1
Secondary1 PhaseShift           : 0
Secondary1 Mux Select           : 4
Secondary2 Freq(MHz)            : 101.000
Use GLC                         : YES
Use YC                          : NO
GLC Delay Value Index           : 1
YC Delay Value Index            : 1
Secondary2 PhaseShift           : 0
Secondary2 Mux Select           : 4

…
…
…

Primary Clock frequency 33.333
Primary Clock Phase Shift 0.000
Primary Clock Output Delay from CLKA 0.180

Secondary1 Clock frequency 66.667
Secondary1 Clock Phase Shift 0.000
Secondary1 Clock Global Output Delay from CLKA 0.180
Secondary1 Clock Core Output Delay from CLKA 0.625

Secondary2 Clock frequency 100.000
Secondary2 Clock Phase Shift 0.000
Secondary2 Clock Global Output Delay from CLKA 0.180
```

Below is an example Verilog HDL description of a legal PLL core configuration generated by SmartGen:

```
module test_pll(POWERDOWN,CLKA,LOCK,GLA);
input POWERDOWN, CLKA;
output  LOCK, GLA;
```

```
    wire VCC, GND;

    VCC VCC_1_net(.Y(VCC));
    GND GND_1_net(.Y(GND));
    PLL Core(.CLKA(CLKA), .EXTFB(GND), .POWERDOWN(POWERDOWN),
        .GLA(GLA), .LOCK(LOCK), .GLB(), .YB(), .GLC(), .YC(),
        .OADIV0(GND), .OADIV1(GND), .OADIV2(GND), .OADIV3(GND),
        .OADIV4(GND), .OAMUX0(GND), .OAMUX1(GND), .OAMUX2(VCC),
        .DLYGLA0(GND), .DLYGLA1(GND), .DLYGLA2(GND), .DLYGLA3(GND)
        , .DLYGLA4(GND), .OBDIV0(GND), .OBDIV1(GND), .OBDIV2(GND),
        .OBDIV3(GND), .OBDIV4(GND), .OBMUX0(GND), .OBMUX1(GND),
        .OBMUX2(GND), .DLYYB0(GND), .DLYYB1(GND), .DLYYB2(GND),
        .DLYYB3(GND), .DLYYB4(GND), .DLYGLB0(GND), .DLYGLB1(GND),
        .DLYGLB2(GND), .DLYGLB3(GND), .DLYGLB4(GND), .OCDIV0(GND),
        .OCDIV1(GND), .OCDIV2(GND), .OCDIV3(GND), .OCDIV4(GND),
        .OCMUX0(GND), .OCMUX1(GND), .OCMUX2(GND), .DLYYC0(GND),
        .DLYYC1(GND), .DLYYC2(GND), .DLYYC3(GND), .DLYYC4(GND),
        .DLYGLC0(GND), .DLYGLC1(GND), .DLYGLC2(GND), .DLYGLC3(GND)
        , .DLYGLC4(GND), .FINDIV0(VCC), .FINDIV1(GND), .FINDIV2(
        VCC), .FINDIV3(GND), .FINDIV4(GND), .FINDIV5(GND),
        .FINDIV6(GND), .FBDIV0(VCC), .FBDIV1(GND), .FBDIV2(VCC),
        .FBDIV3(GND), .FBDIV4(GND), .FBDIV5(GND), .FBDIV6(GND),
        .FBDLY0(GND), .FBDLY1(GND), .FBDLY2(GND), .FBDLY3(GND),
        .FBDLY4(GND), .FBSEL0(VCC), .FBSEL1(GND), .XDLYSEL(GND),
        .VCOSEL0(GND), .VCOSEL1(GND), .VCOSEL2(GND));
    defparam Core.VCOFREQUENCY = 33.000;
endmodule
```

The "PLL Configuration Bits Description" section on page 82 provides descriptions of the PLL configuration bits for completeness. The configuration bits are shown as busses only for purposes of illustration. They will actually be broken up into individual pins in compilation libraries and all simulation models. For example, the FBSEL[1:0] bus will actually appear as pins FBSEL1 and FBSEL0. The setting of these select lines for the static PLL configuration is performed by the software and is completely transparent to the user.

Dynamic PLL Configuration

To generate a dynamically reconfigurable CCC, the user should select **Dynamic CCC** in the configuration section of the SmartGen GUI (Figure 3-26). This will generate both the CCC core and the configuration shift register / control bit MUX.

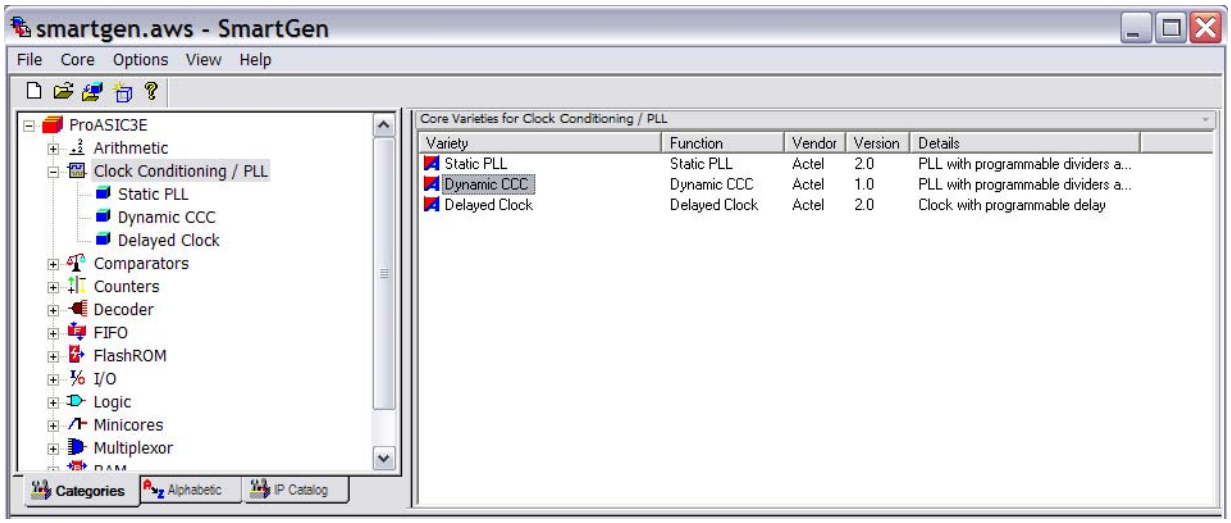


***Figure 3-26* • SmartGen GUI**

Even if dynamic configuration is selected in SmartGen, the user must still specify the static configuration data for the CCC (Figure 3-27). The specified static configuration is used whenever the MODE signal is set to LOW and the CCC is required to function in the static mode. The static configuration data can be used as the default behavior of the CCC where required.

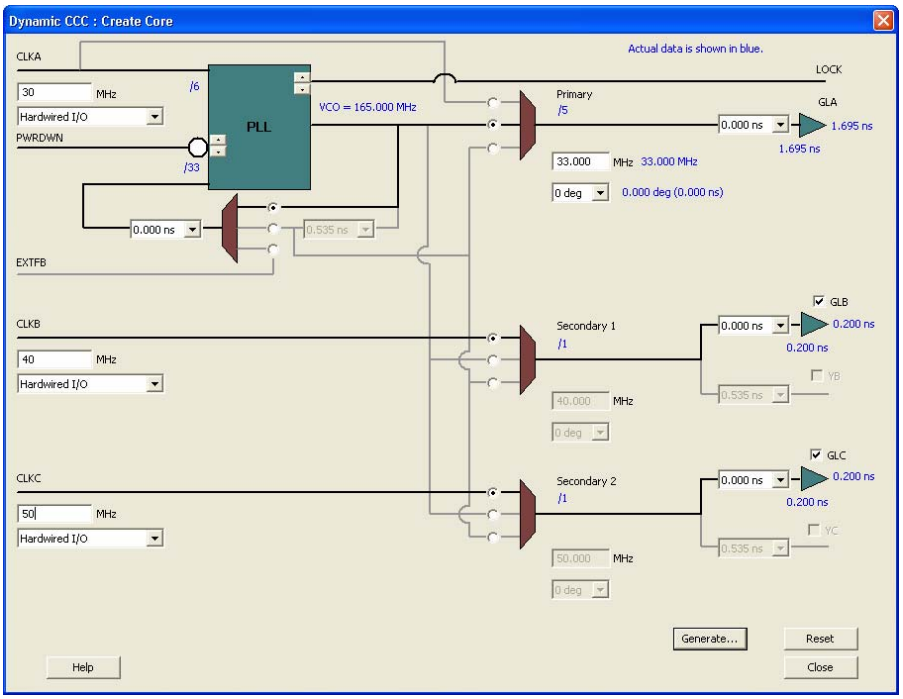


***Figure 3-27* • Dynamic CCC Configuration in SmartGen**

When SmartGen is used to define the configuration that will be shifted in via the serial interface, SmartGen prints out the values of the 81 configuration bits. For ease of use, several configuration bits are automatically inferred by SmartGen when the dynamic PLL core is generated; however, <71:73> (STATASEL, STATBSEL, STATCSEL) and <77:79> (DYNASEL, DYNBSEL, DYNCSEL) depend on the input clock source of the corresponding CCC. Users must first run Layout in Designer to determine the exact setting for these ports. After Layout is complete, generate the "CCC_Configuration" report by choosing **Tools** > **Reports** > **CCC_Configuration** in the Designer software. Refer to "PLL Configuration Bits Description" on page 82 for descriptions of the PLL configuration bits. For simulation purposes, bits <71:73> and <78:80> are "don't care." Therefore, it is strongly suggested that SmartGen be used to generate the correct configuration bit settings for the dynamic PLL core.

After setting all the required parameters, users can generate one or more PLL configurations with HDL or EDIF descriptions by clicking the **Generate** button. SmartGen gives the option of saving session results and messages in a log file:

```
****************
Macro Parameters
****************

Name                            : dyn_pll_hardio
Family                          : ProASIC3E
Output Format                   : VERILOG
Type                            : Dynamic CCC
Input Freq(MHz)                 : 30.000
CLKA Source                     : Hardwired I/O
Feedback Delay Value Index      : 1
Feedback Mux Select             : 1
XDLY Mux Select                 : No
Primary Freq(MHz)               : 33.000
Primary PhaseShift              : 0
Primary Delay Value Index       : 1
Primary Mux Select              : 4
Secondary1 Freq(MHz)            : 40.000
Use GLB                         : YES
Use YB                          : NO
GLB Delay Value Index           : 1
YB Delay Value Index            : 1
Secondary1 PhaseShift           : 0
Secondary1 Mux Select           : 0
Secondary1 Input Freq(MHz)      : 40.000
CLKB Source                     : Hardwired I/O
Secondary2 Freq(MHz)            : 50.000
Use GLC                         : YES
Use YC                          : NO
GLC Delay Value Index           : 1
YC Delay Value Index            : 1
Secondary2 PhaseShift           : 0
Secondary2 Mux Select           : 0
Secondary2 Input Freq(MHz)      : 50.000
CLKC Source                     : Hardwired I/O

Configuration Bits:
FINDIV[6:0]     0000101
FBDIV[6:0]      0100000
OADIV[4:0]      00100
OBDIV[4:0]      00000
OCDIV[4:0]      00000
OAMUX[2:0]      100
OBMUX[2:0]      000
OCMUX[2:0]      000
FBSEL[1:0]      01
FBDLY[4:0]      00000
XDLYSEL         0
DLYGLA[4:0]     00000
DLYGLB[4:0]     00000
```

```
DLYGLC[4:0]     00000
DLYYB[4:0]      00000
DLYYC[4:0]      00000
VCOSEL[2:0]     100

Primary Clock Frequency 33.000
Primary Clock Phase Shift 0.000
Primary Clock Output Delay from CLKA 1.695

Secondary1 Clock Frequency 40.000
Secondary1 Clock Phase Shift 0.000
Secondary1 Clock Global Output Delay from CLKB 0.200

Secondary2 Clock Frequency 50.000
Secondary2 Clock Phase Shift 0.000
Secondary2 Clock Global Output Delay from CLKC 0.200

#####################################
# Dynamic Stream Data
#####################################
--------------------------------------
|NAME    |SDIN     |VALUE   |TYPE     |
--------------------------------------
|FINDIV  |[6:0]    |0000101 |EDIT     |
|FBDIV   |[13:7]   |0100000 |EDIT     |
|OADIV   |[18:14]  |00100   |EDIT     |
|OBDIV   |[23:19]  |00000   |EDIT     |
|OCDIV   |[28:24]  |00000   |EDIT     |
|OAMUX   |[31:29]  |100     |EDIT     |
|OBMUX   |[34:32]  |000     |EDIT     |
|OCMUX   |[37:35]  |000     |EDIT     |
|FBSEL   |[39:38]  |01      |EDIT     |
|FBDLY   |[44:40]  |00000   |EDIT     |
|XDLYSEL |[45]     |0       |EDIT     |
|DLYGLA  |[50:46]  |00000   |EDIT     |
|DLYGLB  |[55:51]  |00000   |EDIT     |
|DLYGLC  |[60:56]  |00000   |EDIT     |
|DLYYB   |[65:61]  |00000   |EDIT     |
|DLYYC   |[70:66]  |00000   |EDIT     |
|STATASEL|[71]     |X       |MASKED   |
|STATBSEL|[72]     |X       |MASKED   |
|STATCSEL|[73]     |X       |MASKED   |
|VCOSEL  |[76:74]  |100     |EDIT     |
|DYNASEL |[77]     |X       |MASKED   |
|DYNBSEL |[78]     |X       |MASKED   |
|DYNCSEL |[79]     |X       |MASKED   |
|RESETEN |[80]     |1       |READONLY |
```

Below is the resultant Verilog HDL description of a legal dynamic PLL core configuration generated by SmartGen:

```
module dyn_pll_macro(POWERDOWN, CLKA, LOCK, GLA, GLB, GLC, SDIN, SCLK, SSHIFT, SUPDATE,
  MODE, SDOUT, CLKB, CLKC);

input POWERDOWN, CLKA;
output  LOCK, GLA, GLB, GLC;
input  SDIN, SCLK, SSHIFT, SUPDATE, MODE;
output  SDOUT;
input  CLKB, CLKC;

  wire VCC, GND;

  VCC VCC_1_net(.Y(VCC));
  GND GND_1_net(.Y(GND));
```

```
  DYNCCC Core(.CLKA(CLKA), .EXTFB(GND), .POWERDOWN(POWERDOWN), .GLA(GLA), .LOCK(LOCK),
    .CLKB(CLKB), .GLB(GLB), .YB(), .CLKC(CLKC), .GLC(GLC), .YC(), .SDIN(SDIN),
    .SCLK(SCLK), .SSHIFT(SSHIFT), .SUPDATE(SUPDATE), .MODE(MODE), .SDOUT(SDOUT),
    .OADIV0(GND), .OADIV1(GND), .OADIV2(VCC), .OADIV3(GND), .OADIV4(GND), .OAMUX0(GND),
    .OAMUX1(GND), .OAMUX2(VCC), .DLYGLA0(GND), .DLYGLA1(GND), .DLYGLA2(GND),
    .DLYGLA3(GND), .DLYGLA4(GND), .OBDIV0(GND), .OBDIV1(GND), .OBDIV2(GND),
    .OBDIV3(GND), .OBDIV4(GND), .OBMUX0(GND), .OBMUX1(GND), .OBMUX2(GND), .DLYYB0(GND),
    .DLYYB1(GND), .DLYYB2(GND), .DLYYB3(GND), .DLYYB4(GND), .DLYGLB0(GND),
    .DLYGLB1(GND), .DLYGLB2(GND), .DLYGLB3(GND), .DLYGLB4(GND), .OCDIV0(GND),
    .OCDIV1(GND), .OCDIV2(GND), .OCDIV3(GND), .OCDIV4(GND), .OCMUX0(GND), .OCMUX1(GND),
    .OCMUX2(GND), .DLYYC0(GND), .DLYYC1(GND), .DLYYC2(GND), .DLYYC3(GND), .DLYYC4(GND),
    .DLYGLC0(GND), .DLYGLC1(GND), .DLYGLC2(GND), .DLYGLC3(GND), .DLYGLC4(GND),
    .FINDIV0(VCC), .FINDIV1(GND), .FINDIV2(VCC), .FINDIV3(GND), .FINDIV4(GND),
    .FINDIV5(GND), .FINDIV6(GND), .FBDIV0(GND), .FBDIV1(GND), .FBDIV2(GND),
    .FBDIV3(GND), .FBDIV4(GND), .FBDIV5(VCC), .FBDIV6(GND), .FBDLY0(GND), .FBDLY1(GND),
    .FBDLY2(GND), .FBDLY3(GND), .FBDLY4(GND), .FBSEL0(VCC), .FBSEL1(GND),
    .XDLYSEL(GND), .VCOSEL0(GND), .VCOSEL1(GND), .VCOSEL2(VCC));
  defparam Core.VCOFREQUENCY = 165.000;

endmodule
```

Delayed Clock Configuration

The CLKDLY macro can be generated with the desired delay and input clock source (Hardwired I/O, External I/O, or Core Logic), as in Figure 3-28.

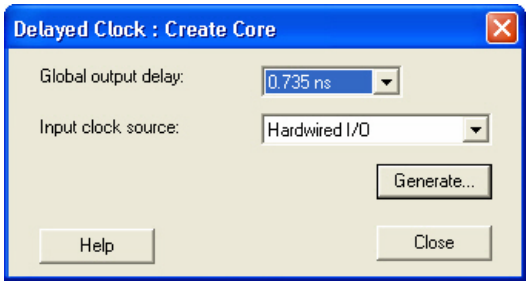


***Figure 3-28* • Delayed Clock Configuration Dialog Box**

After setting all the required parameters, users can generate one or more PLL configurations with HDL or EDIF descriptions by clicking the **Generate** button. SmartGen gives the option of saving session results and messages in a log file:

```
****************
Macro Parameters
****************

Name                            : delay_macro
Family                          : ProASIC3
Output Format                   : Verilog
Type                            : Delayed Clock
Delay Index                     : 2
CLKA Source                     : Hardwired I/O

Total Clock Delay = 0.935 ns.

The resultant CLKDLY macro Verilog netlist is as follows:

module delay_macro(GL,CLK);

output GL;
input  CLK;
```

```
  wire VCC, GND;

  VCC VCC_1_net(.Y(VCC));
  GND GND_1_net(.Y(GND));
  CLKDLY Inst1(.CLK(CLK), .GL(GL), .DLYGL0(VCC), .DLYGL1(GND), .DLYGL2(VCC),
    .DLYGL3(GND), .DLYGL4(GND));

endmodule
```

Detailed Usage Information

Clock Frequency Synthesis

Deriving clocks of various frequencies from a single reference clock is known as frequency synthesis. The PLL has an input frequency range from 1.5 to 350 MHz. This frequency is automatically divided down to a range between 1.5 MHz and 5.5 MHz by input dividers (not shown in Figure 3-19 on page 76) between PLL macro inputs and PLL phase detector inputs. The VCO output is capable of an output range from 24 to 350 MHz. With dividers before the input to the PLL core and following the VCO outputs, the VCO output frequency can be divided to provide the final frequency range from 0.75 to 350 MHz. Using SmartGen, the dividers are automatically set to achieve the closest possible matches to the specified output frequencies.

Users should be cautious when selecting the desired PLL input and output frequencies and the I/O buffer standard used to connect to the PLL input and output clocks. Depending on the I/O standards used for the PLL input and output clocks, the I/O frequencies have different maximum limits. Refer to the family datasheets for specifications of maximum I/O frequencies for supported I/O standards. Desired PLL input or output frequencies will not be achieved if the selected frequencies are higher than the maximum I/O frequencies allowed by the selected I/O standards. Users should be careful when selecting the I/O standards used for PLL input and output clocks. Performing post-layout simulation can help detect this type of error, which will be identified with pulse width violation errors. Users are strongly encouraged to perform post-layout simulation to ensure the I/O standard used can provide the desired PLL input or output frequencies. Users can also choose to cascade PLLs together to achieve the high frequencies needed for their applications. Details of cascading PLLs are discussed in the "Cascading CCCs" section on page 101.

In SmartGen, the actual generated frequency (under typical operating conditions) will be displayed beside the requested output frequency value. This provides the ability to determine the exact frequency that can be generated by SmartGen, in real time. The log file generated by SmartGen is a useful tool in determining how closely the requested clock frequencies match the user specifications. For example, assume a user specifies 101 MHz as one of the secondary output frequencies. If the best output frequency that could be achieved were 100 MHz, the log file generated by SmartGen would indicate the actual generated frequency.

Simulation Verification

The integration of the generated PLL and CLKDLY modules is similar to any VHDL component or Verilog module instantiation in a larger design; i.e., there is no special requirement that users need to take into account to successfully synthesize their designs.

For simulation purposes, users need to refer to the VITAL or Verilog library that includes the functional description and associated timing parameters. Refer to the Software Tools section of the Microsemi SoC Products Group website to obtain the family simulation libraries. If Designer is installed, these libraries are stored in the following locations:

<Designer_Installation_Directory>\lib\vtl\95\proasic3.vhd

<Designer_Installation_Directory>\lib\vtl\95\proasic3e.vhd

<Designer_Installation_Directory>\lib\vlog\proasic3.v

<Designer_Installation_Directory>\lib\vlog\proasic3e.v

For Libero users, there is no need to compile the simulation libraries, as they are conveniently pre-compiled in the Model*Sim*® Microsemi simulation tool.

The following is an example of a PLL configuration utilizing the clock frequency synthesis and clock delay adjustment features. The steps include generating the PLL core with SmartGen, performing simulation for verification with Model*Sim*, and performing static timing analysis with SmartTime in Designer.

Parameters of the example PLL configuration:

Input Frequency – 20 MHz

Primary Output Requirement – 20 MHz with clock advancement of 3.02 ns

Secondary 1 Output Requirement – 40 MHz with clock delay of 2.515 ns

Figure 3-29 shows the SmartGen settings. Notice that the overall delays are calculated automatically, allowing the user to adjust the delay elements appropriately to obtain the desired delays.

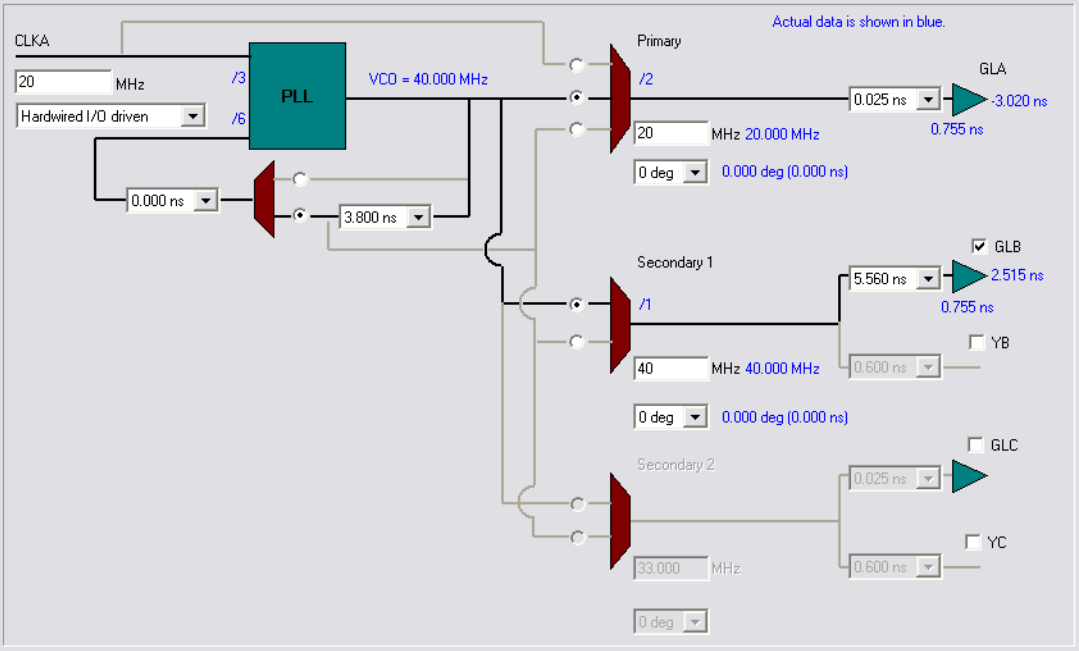


Figure 3-29 • **SmartGen Settings**

After confirming the correct settings, generate a structural netlist of the PLL and verify PLL core settings by checking the log file:

```
Name                              : test_pll_delays
Family                            : ProASIC3E
Output Format                     : VHDL
Type                              : Static PLL
Input Freq(MHz)                   : 20.000
CLKA Source                       : Hardwired I/O
Feedback Delay Value Index        : 21
Feedback Mux Select               : 2
XDLY Mux Select                   : No
Primary Freq(MHz)                 : 20.000
Primary PhaseShift                : 0
Primary Delay Value Index         : 1
Primary Mux Select                : 4
Secondary1 Freq(MHz)              : 40.000
Use GLB                           : YES
Use YB                            : NO
…
…
…
Primary Clock frequency 20.000
Primary Clock Phase Shift 0.000
```

```
Primary Clock Output Delay from CLKA -3.020

Secondary1 Clock frequency 40.000
Secondary1 Clock Phase Shift 0.000
Secondary1 Clock Global Output Delay from CLKA 2.515
```

Next, perform simulation in Model*Sim* to verify the correct delays. Figure 3-30 shows the simulation results. The delay values match those reported in the SmartGen PLL Wizard.

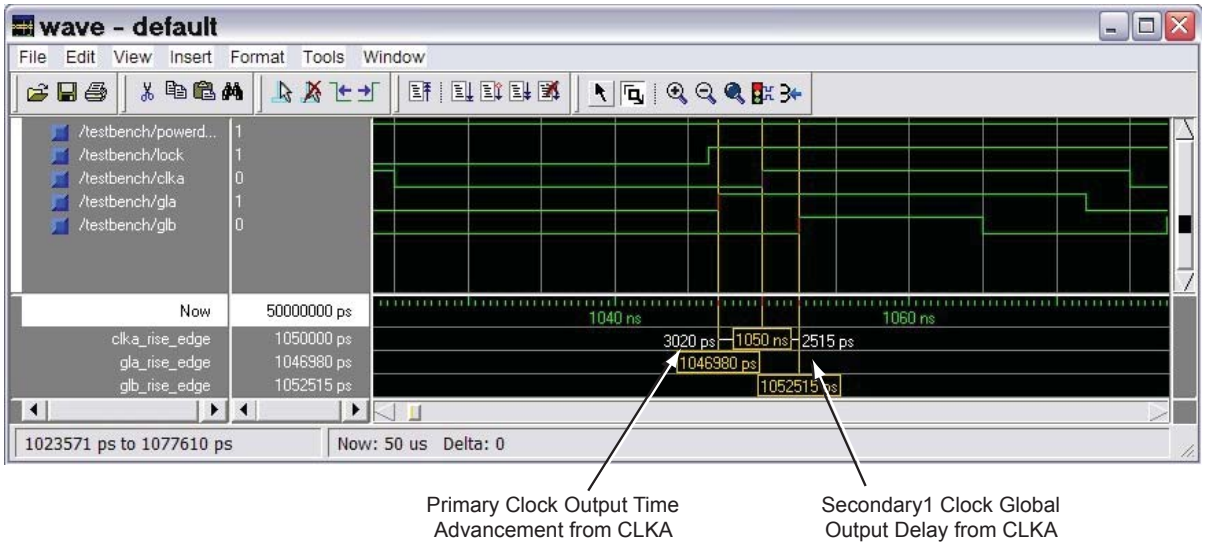
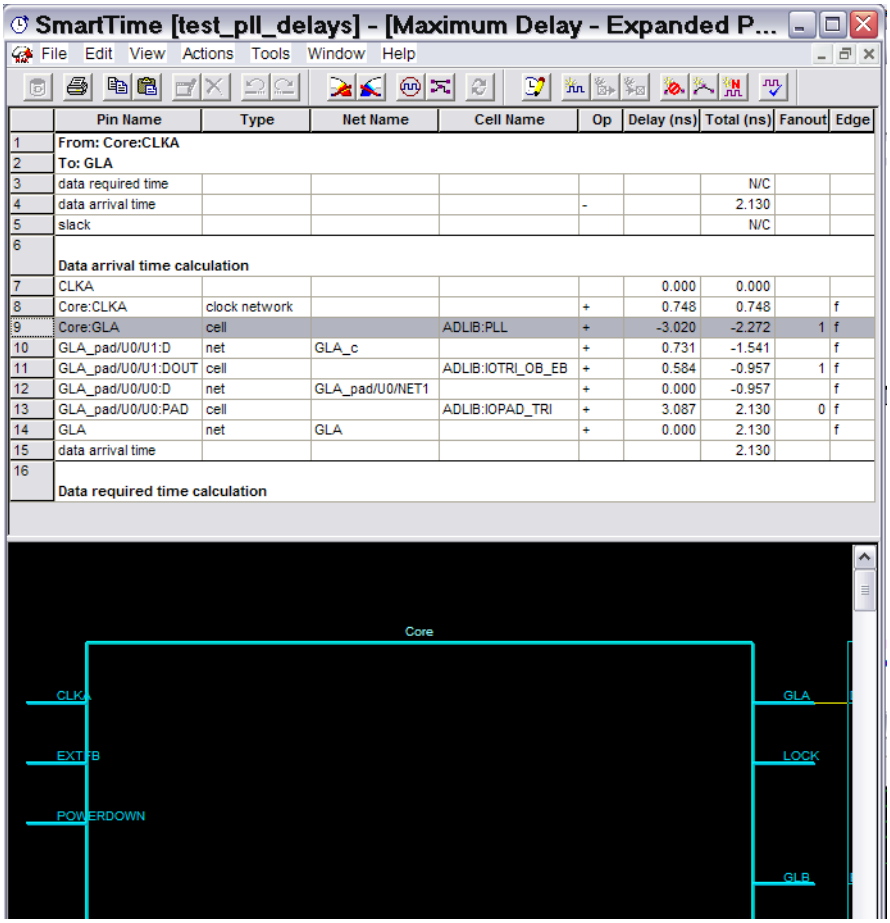


Figure 3-30 • **Model*Sim* Simulation Results**

The timing can also be analyzed using SmartTime in Designer. The user should import the synthesized netlist to Designer, perform Compile and Layout, and then invoke SmartTime. Go to **Tools** > **Options** and change the maximum delay operating conditions to **Typical Case**. Then expand the Clock-to-Out paths of GLA and GLB and the individual components of the path delays are shown. The path of GLA is shown in Figure 3-31 on page 99 displaying the same delay value.



	Pin Name	Type	Net Name	Cell Name	Op	Delay (ns)	Total (ns)	Fanout	Edge
1	From: Core:CLKA								
2	To: GLA								
3	data required time						N/C		
4	data arrival time				-		2.130		
5	slack						N/C		
6	Data arrival time calculation								
7	CLKA					0.000	0.000		
8	Core:CLKA	clock network			+	0.748	0.748		f
9	Core:GLA	cell		ADLIB:PLL	+	-3.020	-2.272	1	f
10	GLA_pad/U0/U1:D	net	GLA_c		+	0.731	-1.541		f
11	GLA_pad/U0/U1:DOUT	cell		ADLIB:IOTRI_OB_EB	+	0.584	-0.957	1	f
12	GLA_pad/U0/U0:D	net	GLA_pad/U0/NET1		+	0.000	-0.957		f
13	GLA_pad/U0/U0:PAD	cell		ADLIB:IOPAD_TRI	+	3.087	2.130	0	f
14	GLA	net	GLA		+	0.000	2.130		f
15	data arrival time						2.130		
16	Data required time calculation								

Figure 3-31 • **Static Timing Analysis Using SmartTime**

Place-and-Route Stage Considerations

Several considerations must be noted to properly place the CCC macros for layout.

For CCCs with clock inputs configured with the Hardwired I/O–Driven option:

- PLL macros must have the clock input pad coming from one of the GmA* locations.
- CLKDLY macros must have the clock input pad coming from one of the Global I/Os.

If a PLL with a Hardwired I/O input is used at a CCC location and a Hardwired I/O–Driven CLKDLY macro is used at the same CCC location, the clock input of the CLKDLY macro must be chosen from one of the GmB* or GmC* pin locations. If the PLL is not used or is an External I/O–Driven or Core Logic–Driven PLL, the clock input of the CLKDLY macro can be sourced from the GmA*, GmB*, or GmC* pin locations.

For CCCs with clock inputs configured with the External I/O–Driven option, the clock input pad can be assigned to any regular I/O location (IO******** pins). Note that since global I/O pins can also be used as regular I/Os, regardless of CCC function (CLKDLY or PLL), clock inputs can also be placed in any of these I/O locations.

By default, the Designer layout engine will place global nets in the design at one of the six chip globals. When the number of globals in the design is greater than six, the Designer layout engine will automatically assign additional globals to the quadrant global networks of the low power flash devices. If the user wishes to decide which global signals should be assigned to chip globals (six available) and which to the quadrant globals (three per quadrant for a total of 12 available), the assignment can be achieved with PinEditor, ChipPlanner, or by importing a placement constraint file. Layout will fail if the

global assignments are not allocated properly. See the "Physical Constraints for Quadrant Clocks" section for information on assigning global signals to the quadrant clock networks.

Promoted global signals will be instantiated with CLKINT macros to drive these signals onto the global network. This is automatically done by Designer when the Auto-Promotion option is selected. If the user wishes to assign the signals to the quadrant globals instead of the default chip globals, this can done by using ChipPlanner, by declaring a physical design constraint (PDC), or by importing a PDC file.

Physical Constraints for Quadrant Clocks

If it is necessary to promote global clocks (CLKBUF, CLKINT, PLL, CLKDLY) to quadrant clocks, the user can define PDCs to execute the promotion. PDCs can be created using PDC commands (pre-compile) or the MultiView Navigator (MVN) interface (post-compile). The advantage of using the PDC flow over the MVN flow is that the Compile stage is able to automatically promote any regular net to a global net before assigning it to a quadrant. There are three options to place a quadrant clock using PDC commands:

- Place a clock core (not hardwired to an I/O) into a quadrant clock location.
- Place a clock core (hardwired to an I/O) into an I/O location (set_io) or an I/O module location (set_location) that drives a quadrant clock location.
- Assign a net driven by a regular net or a clock net to a quadrant clock using the following command:

  ```
  assign_local_clock -net <net name> -type quadrant <quadrant clock region>
  ```

  where

  `<net name>` is the name of the net assigned to the local user clock region.

  `<quadrant clock region>` defines which quadrant the net should be assigned to. Quadrant clock regions are defined as UL (upper left), UR (upper right), LL (lower left), and LR (lower right).

Note: If the net is a regular net, the software inserts a CLKINT buffer on the net.

For example:

```
assign_local_clock -net localReset -type quadrant UR
```

Keep in mind the following when placing quadrant clocks using MultiView Navigator:

Hardwired I/O–Driven CCCs

- Find the associated clock input port under the Ports tab, and place the input port at one of the Gmn* locations using PinEditor or I/O Attribute Editor, as shown in Figure 3-32.

***Figure 3-32* • Port Assignment for a CCC with Hardwired I/O Clock Input**

- Use quadrant global region assignments by finding the clock net associated with the CCC macro under the Nets tab and creating a quadrant global region for the net, as shown in Figure 3-33.

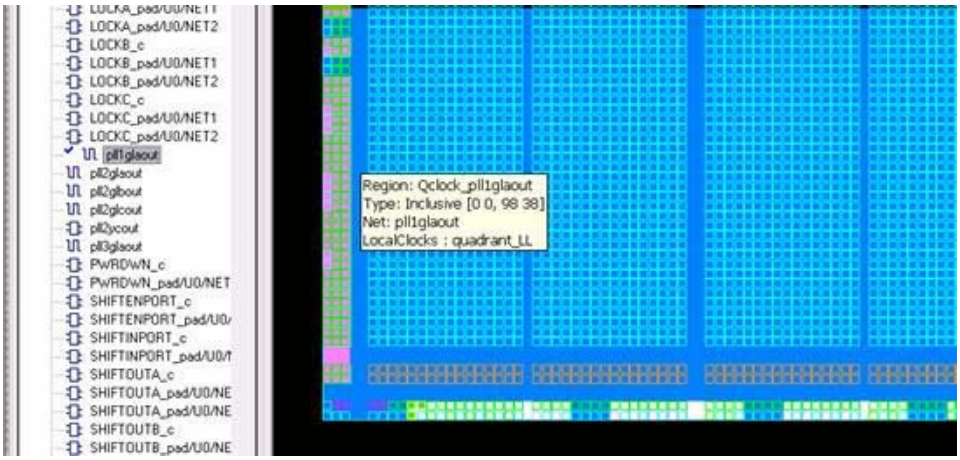


***Figure 3-33* • Quadrant Clock Assignment for a Global Net**

External I/O–Driven CCCs

The above-mentioned recommendation for proper layout techniques will ensure the correct assignment. It is possible that, especially with External I/O–Driven CCC macros, placement of the CCC macro in a desired location may not be achieved. For example, assigning an input port of an External I/O–Driven CCC near a particular CCC location does not guarantee global assignments to the desired location. This is because the clock inputs of External I/O–Driven CCCs can be assigned to any I/O location; therefore, it is possible that the CCC connected to the clock input will be routed to a location other than the one closest to the I/O location, depending on resource availability and placement constraints.

Clock Placer

The clock placer is a placement engine for low power flash devices that places global signals on the chip global and quadrant global networks. Based on the clock assignment constraints for the chip global and quadrant global clocks, it will try to satisfy all constraints, as well as creating quadrant clock regions when necessary. If the clock placer fails to create the quadrant clock regions for the global signals, it will report an error and stop Layout.

The user must ensure that the constraints set to promote clock signals to quadrant global networks are valid.

Cascading CCCs

The CCCs in low power flash devices can be cascaded. Cascading CCCs can help achieve more accurate PLL output frequency results than those achievable with a single CCC. In addition, this technique is useful when the user application requires the output clock of the PLL to be a multiple of the reference clock by an integer greater than the maximum feedback divider value of the PLL (divide by 128) to achieve the desired frequency.

For example, the user application may require a 280 MHz output clock using a 2 MHz input reference clock, as shown in Figure 3-34 on page 102.

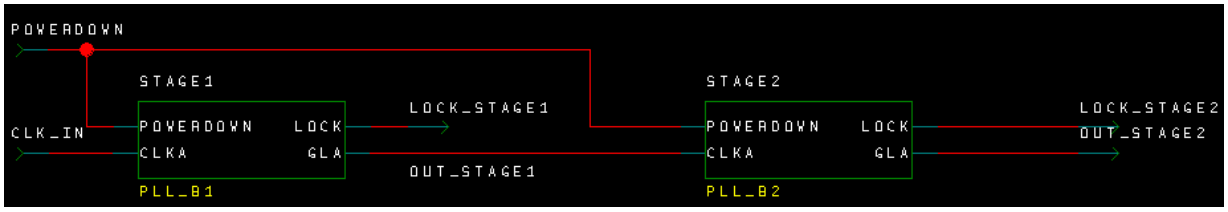


Figure 3-34 • **Cascade PLL Configuration**

Using internal feedback, we know from EQ 3-1 on page 78 that the maximum achievable output frequency from the primary output is

$$f_{GLA} = f_{CLKA} \times m / (n \times u) = 2\ MHz \times 128 / (1 \times 1) = 256\ MHz$$

EQ 3-5

Figure 3-35 shows the settings of the initial PLL. When configuring the initial PLL, specify the input to be either Hardwired I/O–Driven or External I/O–Driven. This generates a netlist with the initial PLL routed from an I/O. Do not specify the input to be Core Logic–Driven, as this prohibits the connection from the I/O pin to the input of the PLL.

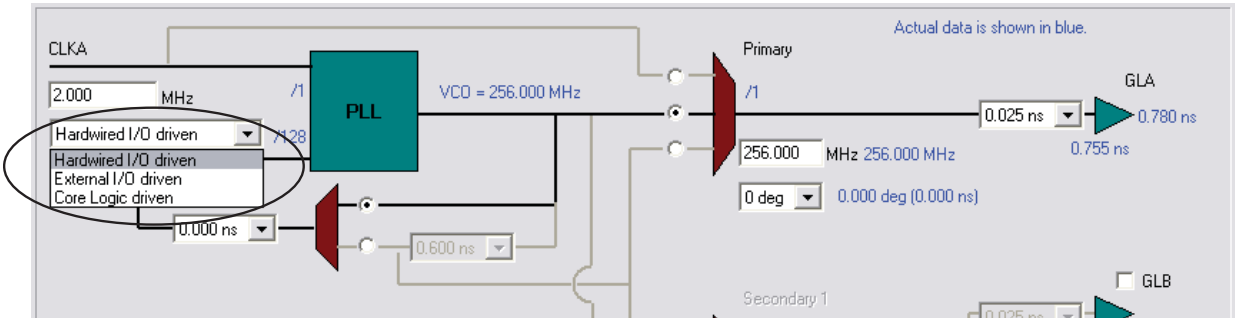


Figure 3-35 • **First-Stage PLL Showing Input of 2 MHz and Output of 256 MHz**

A second PLL can be connected serially to achieve the required frequency. EQ 3-1 on page 78 to EQ 3-3 on page 78 are extended as follows:

$$f_{GLA2} = f_{GLA} \times m_2 / (n_2 \times u_2) = f_{CLKA1} \times m_1 \times m_2 / (n_1 \times u_1 \times n_2 \times u_2)$$ – Primary PLL Output Clock

EQ 3-6

$$f_{GLB2} = f_{YB2} = f_{CLKA1} \times m_1 \times m_2 / (n_1 \times n_2 \times v_1 \times v_2)$$ – Secondary 1 PLL Output Clock(s)

EQ 3-7

$$f_{GLC2} = f_{YC2} = f_{CLKA1} \times m_1 \times m_2 / (n_1 \times n_2 \times w_1 \times w2)$$ – Secondary 2 PLL Output Clock(s)

EQ 3-8

In the example, the final output frequency (f_{output}) from the primary output of the second PLL will be as follows (EQ 3-9):

$$f_{output} = f_{GLA2} = f_{GLA} \times m_2 / (n_2 \times u_2) = 256\ MHz \times 70 / (64 \times 1) = 280\ MHz$$

EQ 3-9

Figure 3-36 on page 103 shows the settings of the second PLL. When configuring the second PLL (or any subsequent-stage PLLs), specify the input to be Core Logic–Driven. This generates a netlist with the second PLL routed internally from the core. Do not specify the input to be Hardwired I/O–Driven or External I/O–Driven, as these options prohibit the connection from the output of the first PLL to the input of the second PLL.

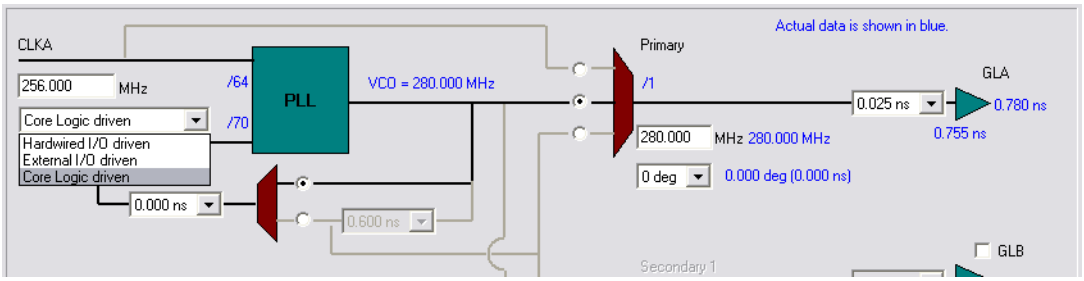


***Figure 3-36* • Second-Stage PLL Showing Input of 256 MHz from First Stage and Final Output of 280 MHz**

Figure 3-37 shows the simulation results, where the first PLL's output period is 3.9 ns (~256 MHz), and the stage 2 (final) output period is 3.56 ns (~280 MHz).

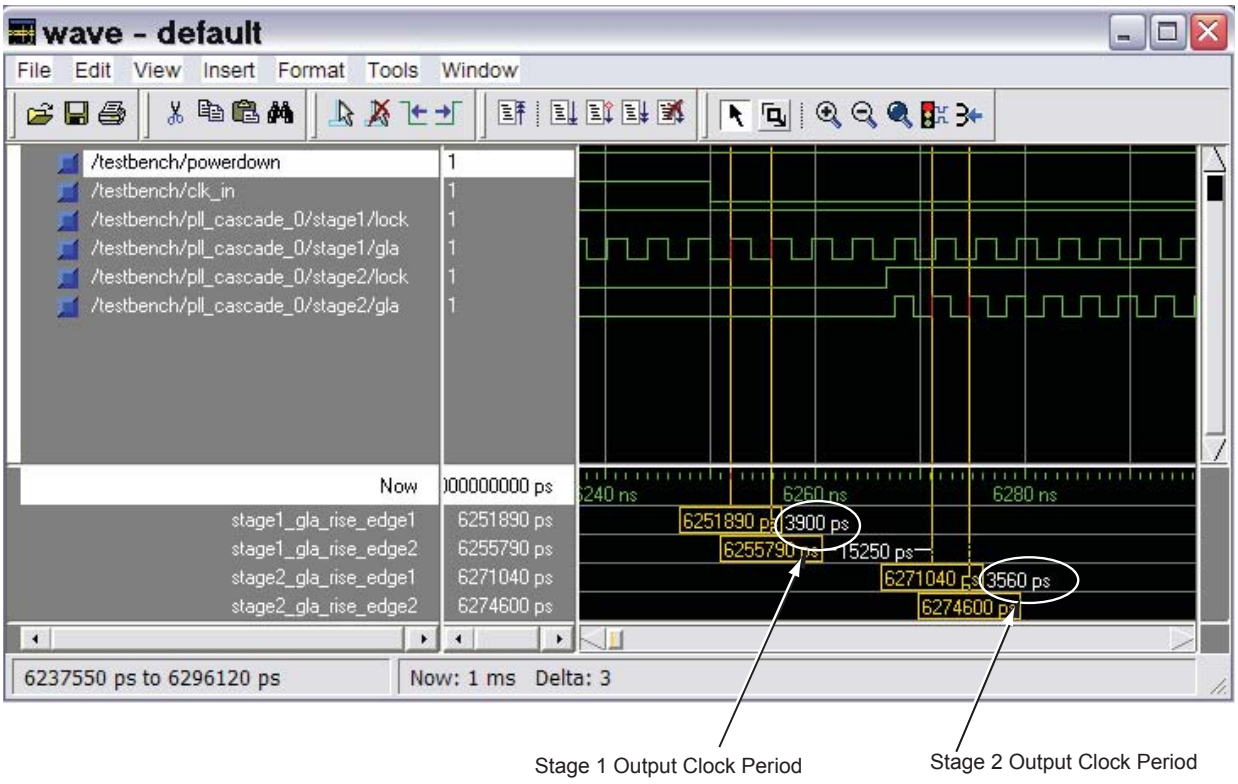


***Figure 3-37* • Model*Sim* Simulation Results**

Recommended Board-Level Considerations

The power to the PLL core is supplied by VCCPLA/B/C/D/E/F (VCCPLx), and the associated ground connections are supplied by VCOMPLA/B/C/D/E/F (VCOMPLx). When the PLLs are not used, the Designer place-and-route tool automatically disables the unused PLLs to lower power consumption. The user should tie unused VCCPLx and VCOMPLx pins to ground. Optionally, the PLL can be turned on/off during normal device operation via the POWERDOWN port (see Table 3-3 on page 60).

PLL Power Supply Decoupling Scheme

The PLL core is designed to tolerate noise levels on the PLL power supply as specified in the datasheets. When operated within the noise limits, the PLL will meet the output peak-to-peak jitter specifications specified in the datasheets. User applications should always ensure the PLL power supply is powered from a noise-free or low-noise power source.

However, in situations where the PLL power supply noise level is higher than the tolerable limits, various decoupling schemes can be designed to suppress noise to the PLL power supply. An example is provided in Figure 3-38. The VCCPLx and VCOMPLx pins correspond to the PLL analog power supply and ground.

Microsemi strongly recommends that two ceramic capacitors (10 nF in parallel with 100 nF) be placed close to the power pins (less than 1 inch away). A third generic 10 µF electrolytic capacitor is recommended for low-frequency noise and should be placed farther away due to its large physical size. Microsemi recommends that a 6.8 µH inductor be placed between the supply source and the capacitors to filter out any low-/medium- and high-frequency noise. In addition, the PCB layers should be controlled so the VCCPLx and VCOMPLx planes have the minimum separation possible, thus generating a good-quality RF capacitor.

For more recommendations, refer to the *Board-Level Considerations* application note.

Recommended 100 nF capacitor:

- Producer BC Components, type X7R, 100 nF, 16 V
- BC Components part number: 0603B104K160BT
- Digi-Key part number: BC1254CT-ND
- Digi-Key part number: BC1254TR-ND

Recommended 10 nF capacitor:

- Surface-mount ceramic capacitor
- Producer BC Components, type X7R, 10 nF, 50 V
- BC Components part number: 0603B103K500BT
- Digi-Key part number: BC1252CT-ND
- Digi-Key part number: BC1252TR-ND

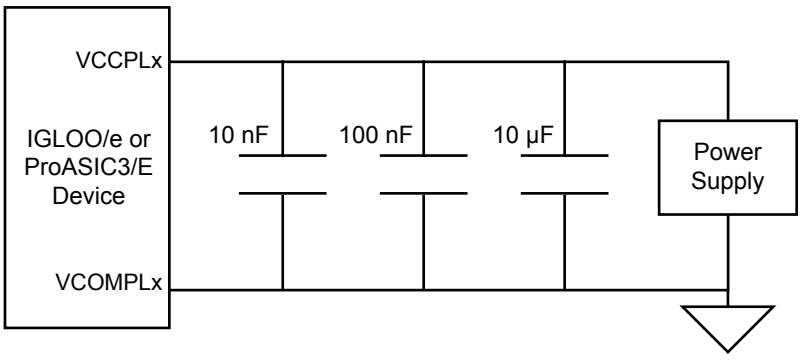


***Figure 3-38* • Decoupling Scheme for One PLL (should be replicated for each PLL used)**

Conclusion

The advanced CCCs of the IGLOO and ProASIC3 devices are ideal for applications requiring precise clock management. They integrate easily with the internal low-skew clock networks and provide flexible frequency synthesis, clock deskewing, and/or time-shifting operations.

Related Documents

Application Notes

Board-Level Considerations

http://www.microsemi.com/soc/documents/ALL_AC276_AN.pdf

Datasheets

Fusion Family of Mixed Signal FPGAs

http://www.microsemi.com/soc/documents/Fusion_DS.pdf

User's Guides

IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide

http://www.microsemi.com/soc/documents/pa3_libguide_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

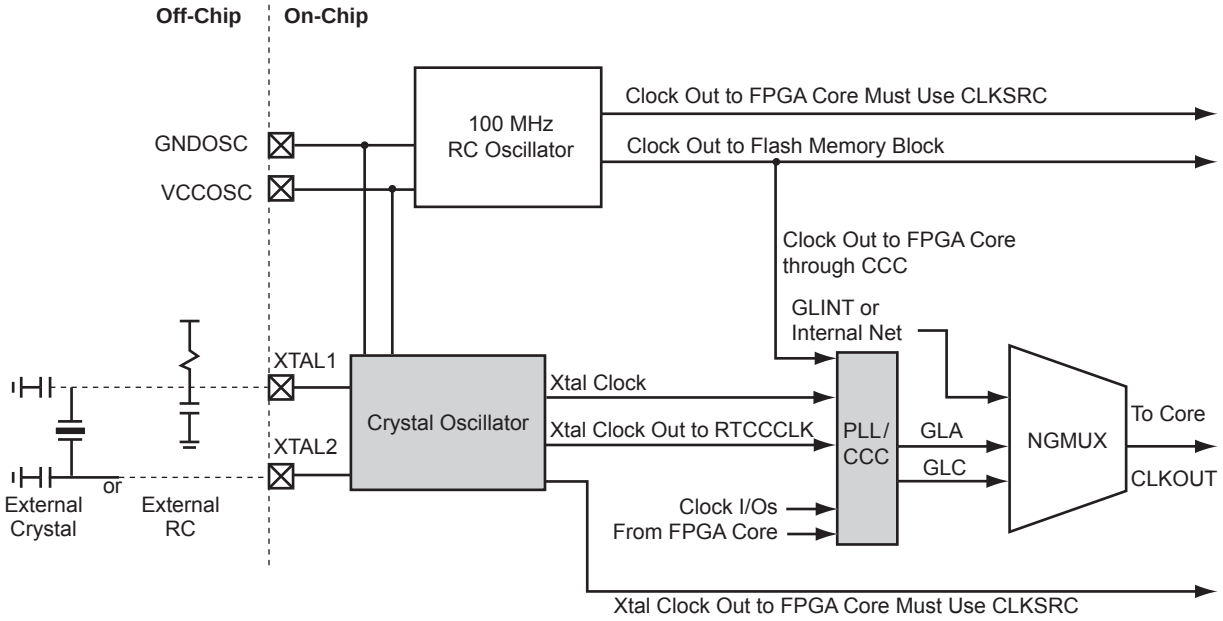
Date	Changes	Page
August 2012	The "Implementing EXTFB in ProASIC3/E Devices" section is new (SAR 36647).	62
	Table 3-7 • Delay Values in Libero SoC Software per Device Family was added to the "Clock Delay Adjustment" section (SAR 22709).	78
	The "Phase Adjustment" section was rewritten to explain better why the visual CCC shows both the actual phase and the actual delay that is equivalent to this phase shift (SAR 29647).	79
	The hyperlink for the *Board-Level Considerations* application note was corrected (SAR 36663)	104, 105
December 2011	Figure 3-20 • PLL Block Diagram, Figure 3-22 • CCC Block Control Bits – Graphical Representation of Assignments, and Table 3-12 • MUXA, MUXB, MUXC were revised to change the phase shift assignments for PLLs 4 through 7 (SAR 33791).	77, 81, 85
June 2011	The description for RESETEN in Table 3-8 • Configuration Bit Descriptions for the CCC Blocks was revised. The phrase "and should not be modified via dynamic configuration" was deleted because RESETEN is read only (SAR 25949).	82
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
	Notes were added where appropriate to point out that IGLOO nano and ProASIC3 nano devices do not support differential inputs (SAR 21449).	N/A

Date	Changes	Page
v1.4 (December 2008)	The"CCC Support in Microsemi's Flash Devices" section was updated to include IGLOO nano and ProASIC3 nano devices.	55
	Figure 3-2 • CCC Options: Global Buffers with No Programmable Delay was revised to add the CLKBIBUF macro.	56
	The description of the reference clock was revised in Table 3-2 • Input and Output Description of the CLKDLY Macro.	57
	Figure 3-7 • Clock Input Sources (30 k gates devices and below) is new. Figure 3-8 • Clock Input Sources Including CLKBUF, CLKBUF_LVDS/LVPECL, and CLKINT (60 k gates devices and above) applies to 60 k gate devices and above.	64
	The "IGLOO and ProASIC3" section was updated to include information for IGLOO nano devices.	65
	A note regarding Fusion CCCs was added to Figure 3-9 • Illustration of Hardwired I/O (global input pins) Usage for IGLOO and ProASIC3 devices 60 k Gates and Larger and the name of the figure was changed from Figure 4-8 • Illustration of Hardwired I/O (global input pins) Usage. Figure 3-10 • Illustration of Hardwired I/O (global input pins) Usage for IGLOO and ProASIC3 devices 30 k Gates and Smaller is new.	66
	Table 3-5 • Number of CCCs by Device Size and Package was updated to include IGLOO nano and ProASIC3 nano devices. Entries were added to note differences for the CS81, CS121, and CS201 packages.	70
	The "Clock Conditioning Circuits without Integrated PLLs" section was rewritten.	71
	The "IGLOO and ProASIC3 CCC Locations" section was updated for nano devices.	73
	Figure 4-13 • CCC Locations in the 15 k and 30 k Gate Devices was deleted.	4-20
v1.3 (October 2008)	This document was updated to include Fusion and RT ProASIC3 device information. Please review the document very carefully.	N/A
	The "CCC Support in Microsemi's Flash Devices" section was updated.	55
	In the "Global Buffer with Programmable Delay" section, the following sentence was changed from: "In this case, the I/O must be placed in one of the dedicated global I/O locations." To "In this case, the software will automatically place the dedicated global I/O in the appropriate locations."	56
	Figure 3-4 • CCC Options: Global Buffers with PLL was updated to include OADIVRST and OADIVHALF.	59
	In Figure 3-6 • CCC with PLL Block "fixed delay" was changed to "programmable delay".	59
	Table 3-3 • Input and Output Signals of the PLL Block was updated to include OADIVRST and OADIVHALF descriptions.	60
	Table 3-8 • Configuration Bit Descriptions for the CCC Blocks was updated to include configuration bits 88 to 81. Note 2 is new. In addition, the description for bit <76:74> was updated.	82
	Table 3-16 • Fusion Dynamic CCC Clock Source Selection and Table 3-17 • Fusion Dynamic CCC NGMUX Configuration are new.	86
	Table 3-18 • Fusion Dynamic CCC Division by Half Configuration and Table 3-19 • Configuration Bit <76:75> / VCOSEL<2:1> Selection for All Families are new.	87

Date	Changes	Page
v1.2 (June 2008)	The following changes were made to the family descriptions in Figure 3-1 • Overview of the CCCs Offered in Fusion, IGLOO, and ProASIC3: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	53
v1.1 (March 2008)	Table 3-1 • Flash-Based FPGAs and the associated text were updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	55
	The "Global Input Selections" section was updated to include 15 k gate devices as supported I/O types for globals, for CCC only.	63
	Table 3-5 • Number of CCCs by Device Size and Package was revised to include ProASIC3L, IGLOO PLUS, A3P015, AGL015, AGLP030, AGLP060, and AGLP125.	70
	The "IGLOO and ProASIC3 CCC Locations" section was revised to include 15 k gate devices in the exception statements, as they do not contain PLLs.	73
v1.0 (January 2008)	Information about unlocking the PLL was removed from the "Dynamic PLL Configuration" section.	79
	In the "Dynamic PLL Configuration" section, information was added about running Layout and determining the exact setting of the ports.	92
	In Table 3-8 • Configuration Bit Descriptions for the CCC Blocks, the following bits were updated to delete "transport to the user" and reference the footnote at the bottom of the table: 79 to 71.	82

4 – Fusion Clock Resources

The Fusion mixed signal FPGA family of devices has a wide variety of on-chip clocking peripherals, as shown in Figure 4-1. The on-chip resources enable the creation, manipulation, and distribution of clock signals both internally and externally. An integrated internal RC oscillator produces a 100 MHz clock without external components. For systems that require more precise clock signals, the Fusion mixed signal FPGA family also supports an on-chip crystal oscillator circuit. In conjunction with the crystal oscillator circuit, the on-chip Real-Time Counter (RTC) provides timed wake-up or power-up sequences and thus the ability to supply time and date stamps.



Note: This is a simplified block diagram of the clocking resources within the Fusion device. For details regarding the global networks and clocking resources, refer to the Fusion Mixed Signal FPGAs datasheet. For details regarding the PLL/CCC, refer to the "Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" section on page 53.

Figure 4-1 • **Fusion Mixed Signal FPGA Clocking System**

Like the ProASIC3/E family of flash-based FPGAs, the flash-based Fusion device integrates up to two phase-locked loop (PLL) cores in the embedded Clock Conditioning Circuits (CCCs). The PLL can be clocked from the internal RC oscillator, crystal oscillator, or any other internal signal. The integrated PLL provides the capability to alter the clock source by multiplying, dividing, synchronizing, advancing, or delaying the signal.

The Fusion mixed signal FPGA also integrates one No-Glitch MUX (NGMUX) for each PLL. The NGMUX enables designers to switch between multiple clock sources without introducing glitches into the clock network.

This chapter includes the following sections:

- "Internal RC Oscillator" on page 110
- "Crystal Oscillator (XTLOSC)" on page 114
- "No-Glitch Multiplexer (NGMUX)" on page 123
- "Real-Time Counter (RTC)" on page 131

For information on using the CCC and PLL, refer to the "Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" section on page 53.

Internal RC Oscillator

The internal RC oscillator is an on-chip free-running clock source capable of generating a 100 MHz source without external components. Using the internal RC oscillator in conjunction with the integrated PLL enables designers to generate clocks of varying frequency and phase, clocking both on- and off-chip resources.

The *Fusion Mixed Signal FPGAs* datasheet contains both timing and accuracy characteristics for the internal RC oscillator peripheral.

RC Oscillator Usage

The internal RC oscillator is capable of driving any of the clock macros (i.e., a static or dynamic PLL) directly after instantiation. To drive a macro in the FPGA core, the RC oscillator must first be routed through the CLKSRC macro. See the examples below on manually instantiating the RCOSC and CLKSRC macros. SmartGen can also be used to implement these macros. For more information on using SmartGen, refer to the *SmartGen, FlashROM, ASB, and Flash Memory System Builder User's Guide*.

Example: RC Oscillator Driving Clock Macros

The following example manually instantiates the internal RC oscillator using the RCOSC macro, and connects the 100 MHz clock output to the input pin of a SmartGen-generated PLL. Since the 100 MHz clock output does not connect to FPGA core logic, the CLKSRC macro is not needed.

Verilog

```
module myClocks (
  NSYSRESET,
  CLK25MHZ
);

  input NSYSRESET;
  output CLK25MHZ;

  wire CLK100;

  RCOSC uRCOSC (
    .CLKOUT (CLK100)
  );

  myPLL myPLL1 (
    .POWERDOWN (1'b1),
    .CLKA (CLK100),
    .LOCK (),
    .GLA  (CLK25MHZ),
    .OADIVRST (NSYSRESET)
  );

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity myClocks is
  port (
    NSYSRESET: in std_logic;
    CLK25MHZ: out std_logic
  );
end entity myClocks;

architecture myClocks is
  signal CLK100 : std_logic;

  component RCOSC
  port (
    CLKOUT: out std_logic
  );

  component myPLL
  port (
    POWERDOWN: in std_logic;
    CLKA: out std_logic;
    LOCK: out std_logic;
    GLA: out std_logic;
    OADIVRST: in std_logic
  );

begin
  uRCOSC : RCOSC
  port map (
    CLKOUT => CLK100
  );

  myPLL1 : myPLL
  port map (
    POWERDOWN => '1',
    CLKA => CLK100,
    LOCK => open,
    GLA => CLK25MHZ,
    OADIVRST => NSYSRESET
  );

end architecture myClocks;
```

Example: RC Oscillator Driving FPGA Core Logic

The following example manually instantiates the internal RC oscillator and connects the 100 MHz clock output to the FPGA core logic, which in turn generates a 25 MHz clock. Both the RCOSC and CLKSRC macros are used, RCOSC to instantiate the internal RC oscillator and CLKSRC to connect the RCOSC output to FPGA core logic.

Verilog

```
module myClock (
  NSYSRESET,
  CLK25MHZ
);

  input NSYSRESET;
  output CLK25MHZ;

  wire CLK100, SYSCLK;

  reg [1:0] iCOUNT;

  RCOSC uRCOSC (
    .CLKOUT (CLK100)
  );

  CLKSRC uCLKSRC (
    .A  (CLK100),
    .Y  (SYSCLK)
  );

  always @ (negedge NSYSRESET or posedge SYSCLK)
  begin
    if (NSYSRESET == 1'b0)
      iCOUNT = 2'b0;
    else iCOUNT = iCOUNT + 1'b1;
  end

  assign CLK25MHZ = iCOUNT[1];

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity myClock is
  port (
    NSYSRESET: in std_logic;
    CLK25MHZ: out std_logic
  );
end entity myClock;

architecture myClock is
  signal CLK100 : std_logic;
  signal SYSCLK : std_logic;

  component RCOSC
  port (
    CLKOUT: out std_logic
  );

  component CLKSRC
  port (
    A: in std_logic;
    Y: out std_logic
  );

begin

  uRCOSC : RCOSC
  port map (
    CLKOUT => CLK100
  );

  uCLKSRC : CLKSRC
  port map (
    A => CLK100,
    Y => SYSCLK
  );

  process (NSYSRESET, SYSCLK)
  variable iCOUNT: std_logic_vector(1 downto 0);
  begin
  if (NSYSRESET = '0')
    iCOUNT := (others => '0');
  elsif (SYSCLK'event and SYSCLK = '1')
    iCOUNT := iCOUNT + '1';
  end

  CLK25MHZ <= iCOUNT(1);

end architecture myClock;
```

RC Oscillator Tips and Package Connections

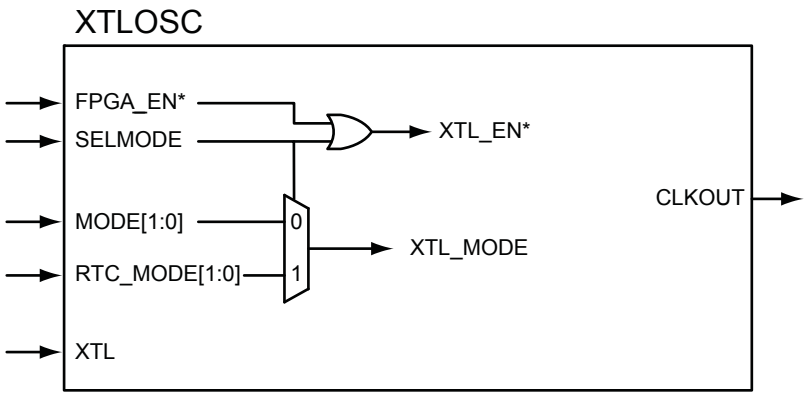
Although the internal RC oscillator is only a one-port macro, the GNDOSC and VCCOSC package pins must be connected externally to provide a power and ground source for the resource and the crystal oscillator. If neither the RC nor the crystal oscillator is used, refer to the *Fusion Mixed Signal FPGAs* datasheet for accurate termination guidelines.

Crystal Oscillator (XTLOSC)

The crystal oscillator (XTLOSC) generates the clock from an external crystal. The output of the XTLOSC CLKOUT signal can be selected as an input to the PLL. Refer to the "Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" section on page 53 for more details. The XTLOSC can operate in normal operation and standby mode (RTC is running and 1.5 V is not present).

In normal operation, the internal FPGA_EN signal is 1 as long as 1.5 V is present for $V_{CC.}$. The internal enable signal for the crystal oscillator, XTL_EN, is enabled since FPGA_EN is asserted. The XTL_MODE signal can use MODE or RTC_MODE, depending on SELMODE.

During standby, 1.5 V is not available. FPGA_EN is 0 and SELMODE must be asserted in order for XTL_EN to be enabled. Hence XTL_MODE relies on RTC_MODE. SELMODE and RTC_MODE must be connected to RTCXTLSEL and RTCXTLMODE from the AB, respectively, for correct operation during standby. Refer to the "Real-Time Counter (RTC)" section on page 131 for a detailed description.



*Note: *Internal signal; does not exist in macro.*

Figure 4-2 • **XTLOSC Macro**

***Table 4-1* • XTLOSC Signals Description**

Signal Name	Width	Direction	Function
XTL_EN*	1		Enables the crystal. Active high.
XTL_MODE*	2		Settings for the crystal clock for different frequencies: **Value** **Modes** **Frequency Range** b'00 RC network 32 kHz to 4 MHz b'01 Low gain 32 to 200 kHz b'10 Medium gain 0.20 to 2.0 MHz b'11 High gain 2.0 to 20.0 MHz
SELMODE	1	IN	Selects the source of XTL_MODE and also enables the XTL_EN. Connect from RTCXTLSEL from AB. 0: For normal operation or sleep mode of operation XTL_EN depends on FPGA_EN, XTL_MODE depends on MODE 1: For Standby mode of operation XTL_EN is enabled, XTL_MODE depends on RTC_MODE
RTC_MODE[1:0]	2	IN	Settings for the crystal clock for different frequency ranges. XTL_MODE uses RTC_MODE when SELMODE is 1.
MODE[1:0]	2	IN	Settings for the crystal clock for different frequency ranges. XTL_MODE uses MODE when SELMODE is 0. In standby, MODE inputs will be 0s.
FPGA_EN*	1	IN	0 when 1.5 V is not present for V_{CC} 1 when 1.5 V is present for V_{CC}
XTL	1	IN	Crystal clock source
CLKOUT	1	OUT	Crystal clock output

*Note: *Internal signal and does not exist in macro*

The crystal oscillator can be configured in one of the four modes:

- RC network, 32 kHz to 4 MHz
- Low gain, 32 to 200 kHz
- Medium gain, 0.20 to 2.0 MHz
- High gain, 2.0 to 20.0 MHz

In RC network mode, the XTAL1 pin is connected to an RC circuit, as shown in Figure 4-1 on page 109. The XTAL2 pin should be left floating. The RC value can be chosen based on Figure 4-3 for any desired frequency between 32 kHz and 4 MHz. The RC network mode can also accommodate an external clock source on XTAL1 instead of an RC circuit.

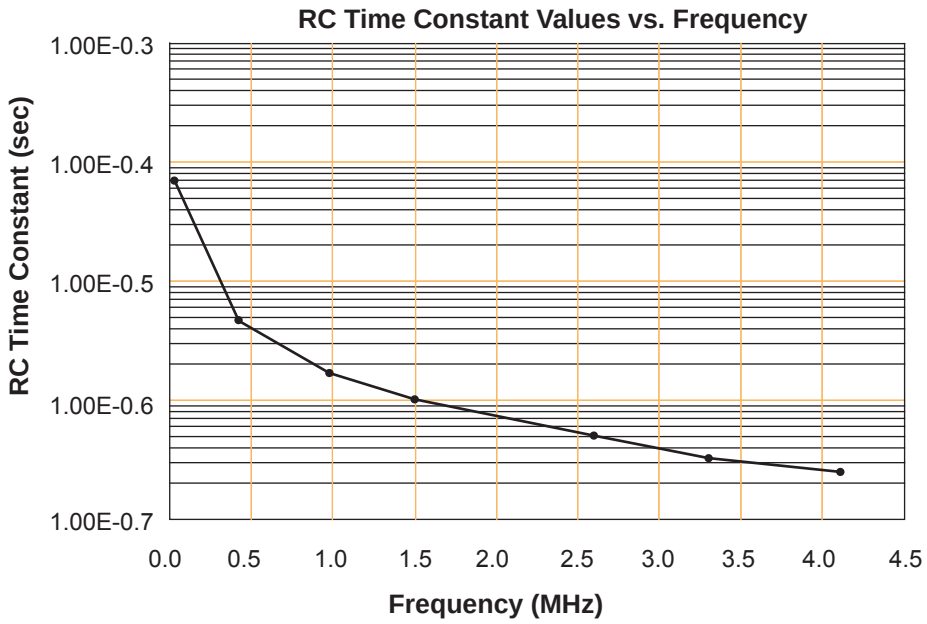


Figure 4-3 • **Crystal Oscillator: RC Time Constant Values vs. Frequency (typical)**

In low gain, medium gain, and high gain, an external crystal component or ceramic resonator can be added onto XTAL1 and XTAL2, as shown in Figure 4-1 on page 109.

Example: Crystal Oscillator Driving the Real-Time Counter

The following example manually instantiates the crystal oscillator using the XTLOSC macro and connects the external 32.768 kHz crystal output to the RTC in the Analog Block (AB). Since the 32.768 kHz clock output does not connect to FPGA core logic, the CLKSRC macro is not needed. The examples below assumes that the Analog Configuration MUX (ACM) has been previously configured and is controlling the functionality of the RTC. For more information on the ACM, refer to the "Designing the Fusion Analog System" section on page 233.

Verilog

```
module myRTC (
  CLK10MHZ,
  CLK32kHz
);

  input CLK10MHZ;
  input CLK32kHz;

  wire iRTCCLK, iRTCSELMODE;
  wire [1:0] iRTCMODE;

  wire iACMCLK, iACMWEN, iACMRESET;
```

```
  wire [7:0] iACMADDR, iACMRDATA, iACMWDATA;

  XTLOSC uXTLOSC (
    .XTL  (CLK32kHz),
    .CLKOUT (iRTCCLK),
    .SELMODE (iRTCSELMODE),
    .MODE (2'b0),
    .RTCMODE (iRTCMODE)
  );

  AB uAB (
    // Note: Several of the Analog Block signals
    //  have been omitted from this
    //  example, only the critical signals
    //  are present.

    .SYSCLK (CLK10MHZ),

    .ACMCLK (iACMCLK),
    .ACMWEN (iACMWEN),
    .ACMRESET (iACMRESET),
    .ACMWDATA (iACMWDATA),
    .ACMADDR (iACMADDR),
    .ACMRDATA (iACMRDATA),

    .RTCCLK (iRTCCLK),
    .RTCXTLSEL (iRTCSELMODE),
    .RTCXTLMODE (iRTCMODE),
    .RTCMATCH (),
    .RTCPSMMATCH ()
  );

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity myRTC is
  port (
    CLK10MHZ: in std_logic;
    CLK32kHz: in std_logic
  );
end entity myRTC;

architecture myRTC is
  signal iRTCCLK : std_logic;
  signal iRTCSELMODE : std_logic;
  signal iRTCMODE : std_logic_vector(1 downto 0);

  signal iACMCLK : std_logic;
  signal iACMRESET : std_logic;
  signal iACMWEN : std_logic;
  signal iACMADDR : std_logic_vector(7 downto 0;
  signal iACMRDATA : std_logic_vector(7 downto 0;
  signal iACMWDATA : std_logic_vector(7 downto 0;

  component XTLOSC
  port (
    XTL: in std_logic;
    CLKOUT: out std_logic;
    SELMODE: in std_logic_vector(1 downto 0);
    RTCMODE: in std_logic_vector(1 downto 0);
    MODE: in std_logic_vector(1 downto 0)
  );
```

```
AB
  -- Note: Several of the Analog Block signals
  --  have been omitted from this
  --  example, only the critical signals
  --  are present.
  port (
    SYSCLK: in std_logic;

    ACMCLK: in std_logic;
    ACMWEN: in std_logic;
    ACMRESET: in std_logic;
    ACMADDR: in std_logic_vector(7 downto 0);
    ACMWDATA: in std_logic_vector(7 downto 0);
    ACMRDATA: out std_logic_vector(7 downto 0);

    RTCCLK: in std_logic;
    RTCXTLSEL: out std_logic;
    RTCXTLMODE: out std_logic_vector(1 downto 0);
    RTCMATCH: out std_logic;
    RTCPSMATCH: out std_logic
  );

begin

  uXTLOSC : XTLOSC
  port map (
    XTL => CLK32kHz,
    CLKOUT => iRTCCLK,
    SELMODE => iRTCSELMODE,
    MODE => “00”,
    RTCMODE => iRTCMODE
  );

  uAB : AB
  -- Note: Several of the Analog Block signals
  --  have been omitted from this
  --  example, only the critical signals
  --  are present.
  port map (
    SYSCLK => CLK10MHZ,

    ACMCLK => iACMCLK,
    ACMWEN => iACMWEN,
    ACMRESET => iACMRESET,
    ACMWDATA => iACMWDATA,
    ACMADDR => iACMADDR,
    ACMRDATA => iACMRDATA,

    RTCCLK => iRTCCLK,
    RTCXTLSEL => iRTCSELMODE,
    RTCXTLMODE => iRTCMODE,
    RTCMATCH => open,
    RTCPSMMATCH => open
  );

end architecture myRTC;
```

Example: Crystal Oscillator Driving Clock Macros

The following example manually instantiates the crystal oscillator using the XTLOSC macro and connects the external crystal to the input pin of a SmartGen-generated PLL. Since the external crystal does not connect to FPGA core logic, the CLKSRC macro is not needed.

Verilog

```
module myXTAL (
  CLK50MHZ,
  NSYSRESET,
  XTAL10MHZ
);

  input XTAL10MHZ;
  input NSYSRESET;
  output CLK50MHZ;

  wire iXTLCLK;

  XTLOSC uXTLOSC (
    .XTL  (XTAL10MHZ),
    .CLKOUT (iXTLCLK),
    .SELMODE (1'b0),
    .MODE (2'b11),
    .RTCMODE (2'b0)
  );

  myPLL myPLL1 (
    .POWERDOWN (1'b1),
    .CLKA (iXTLCLK),
    .LOCK (),
    .GLA  (CLK50MHZ),
    .OADIVRST (NSYSRESET)
  );

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity myXTAL is
  port (
    CLK50MHZ: out std_logic;
    NSYSRESET: in std_logic;
    XTAL10MHZ: in std_logic
  );
end entity myXTAL;

architecture myXTAL is
  signal iXTLCLK : std_logic;

  component XTLOSC
  port (
    XTL: in std_logic;
    CLKOUT: out std_logic;
    SELMODE: in std_logic_vector(1 downto 0);
    RTCMODE: in std_logic_vector(1 downto 0);
    MODE: in std_logic_vector(1 downto 0)
  );

  component myPLL
  port (
    POWERDOWN: in std_logic;
    CLKA: out std_logic;
    LOCK: out std_logic;
    GLA: out std_logic;
    OADIVRST: in std_logic
  );

begin
  uXTLOSC : XTLOSC
  port map (
    XTL => XTAL10MHZ,
    CLKOUT => iXTLCLK,
    SELMODE => ‘0’,
    MODE => “11”,
    RTCMODE => “00”
  );

  myPLL1 : myPLL
  port map (
    POWERDOWN => ‘1’,
    CLKA => iXTLCLK,
    LOCK => open,
    GLA => CLK50MHZ,
    OADIVRST => NSYSRESET
  );

end architecture myXTAL;
```

Example: Crystal Oscillator Driving FPGA Core Logic

The following example manually instantiates the crystal oscillator and connects the external crystal output to the FPGA core logic, which in turn generates a clock divided by four. Both the XTLOSC and CLKSRC macros are used, XTLOSC to instantiate the crystal oscillator and CLKSRC to connect the XTLOSC output to FPGA core logic.

Verilog

```
module myCLKDIV (
  CLKDIV4,
  NSYSRESET,
  XTAL10MHZ
);

  input XTAL10MHZ;
  input NSYSRESET;
  output CLKDIV4;

  wire iXTLCLK;
  wire SYSCLK;

  reg [1:0] iCOUNT;

  XTLOSC uXTLOSC (
    .XTL  (XTAL10MHZ),
    .CLKOUT (iXTLCLK),
    .SELMODE (1'b0),
    .MODE (2'b11),
    .RTCMODE (2'b0)
  );

  CLKSRC uCLKSRC (
    .A  (iXTLCLK),
    .Y  (SYSCLK)
  );

  always @ (negedge NSYSRESET or posedge SYSCLK)
  begin
    if (NSYSRESET == 1'b0)
      iCOUNT = 2'b0;
    else iCOUNT = iCOUNT + 1'b1;
  end

  assign CLKDIV4 = iCOUNT[1];

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity myCLKDIV is
  port (
    CLKDIV4: out std_logic;
    NSYSRESET: in std_logic;
    XTAL10MHZ: in std_logic
  );
end entity myCLKDIV;

architecture myCLKDIV is
  signal iXTLCLK : std_logic;
  signal SYSCLK : std_logic;

  component XTLOSC
  port (
    XTL: in std_logic;
    CLKOUT: out std_logic;
    SELMODE: in std_logic_vector(1 downto 0);
    RTCMODE: in std_logic_vector(1 downto 0);
    MODE: in std_logic_vector(1 downto 0)
  );

  component CLKSRC
  port (
    A: in std_logic;
    Y: out std_logic
  );

begin

  uXTLOSC : XTLOSC
  port map (
    XTL => XTAL10MHZ,
    CLKOUT => iXTLCLK,
    SELMODE => '0',
    MODE => "11",
    RTCMODE => "00"
  );

  uCLKSRC : CLKSRC
  port map (
    A => iXTLCLK,
    Y => SYSCLK
  );

  process (NSYSRESET, SYSCLK)
  variable iCOUNT: std_logic_vector(1 downto 0);
  begin
    if (NSYSRESET = '0')
      iCOUNT := (others => '0');
    elsif (SYSCLK'event and SYSCLK = '1')
      iCOUNT := iCOUNT + '1';
  end

  CLKDIV4 <= iCOUNT(1);

end architecture myCLKDIV;
```

Crystal Oscillator Tips and Package Connections

When using the crystal oscillator, the GNDOSC and VCCOSC external package pins must be connected to provide power and ground sources for this resource and the internal RC oscillator (Table 4-2). If neither the RC nor the crystal oscillator is used, refer to the *Fusion Mixed Signal FPGAs* datasheet for accurate termination guidelines.

In addition, the XTL pin of the XTLOSC macro is connected to the XTAL1 package pin, the crystal oscillator circuit input. When using an external RC network the XTAL2 package pin must be left unconnected.

***Table 4-2* • Crystal Oscillator Tips and Package Connections**

Signal Name	Direction	Description
XTAL1	Input	Input to crystal oscillator circuit. This pin is used to connect the external crystal, ceramic resonator, RC network, or external clock input. When using an external crystal or ceramic oscillator, Microsemi recommends using external capacitors (refer to the *Fusion Mixed Signal FPGAs* datasheet for the recommended capacitor values). If using an external RC network or clock input, use XTAL1 and leave XTAL2 unconnected.
XTAL2	Input	Input to crystal oscillator circuit. This pin is used to connect the external crystal, ceramic resonator, RC network, or external clock input. When using an external crystal or ceramic oscillator, Microsemi recommends using external capacitors (refer to the *Fusion Mixed Signal FPGAs* datasheet for the recommended capacitor values). If using an external RC network or clock input, use XTAL1 and leave XTAL2 unconnected.
VCCOSC	Input	External power supply (3.3 V) for both the integrated RC and crystal oscillator circuits
GNDOSC	Input	External ground supply for both the integrated RC and crystal oscillator circuits

No-Glitch Multiplexer (NGMUX)

Up to two No-Glitch Multiplexers, positioned downstream from the PLL/CCC blocks, are integrated into the Fusion device, as shown in Figure 4-4. The NGMUX provides a special switching sequence between two asynchronous clock domains, which avoids generating any unwanted narrow glitch pulses. It switches between two different clock sources and the output goes to the global network, as shown in Figure 4-5 on page 124.

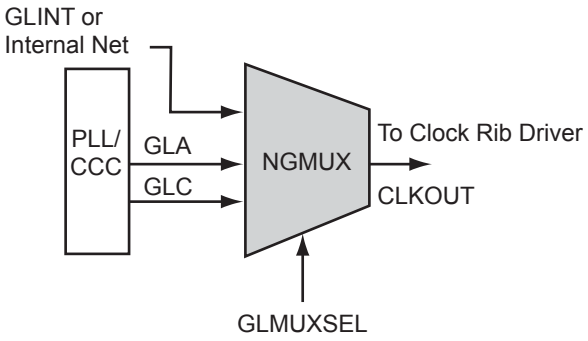


***Figure 4-4* • No-Glitch MUX**

Figure 4-5 • **No-Glitch MUX Macro**

The NGMUX allows the following inputs: PLL outputs GLA and GLC, and GLINT or any internal net. However, there are some restrictions on these signals, which are described in "NGMUX Usage" on page 126.

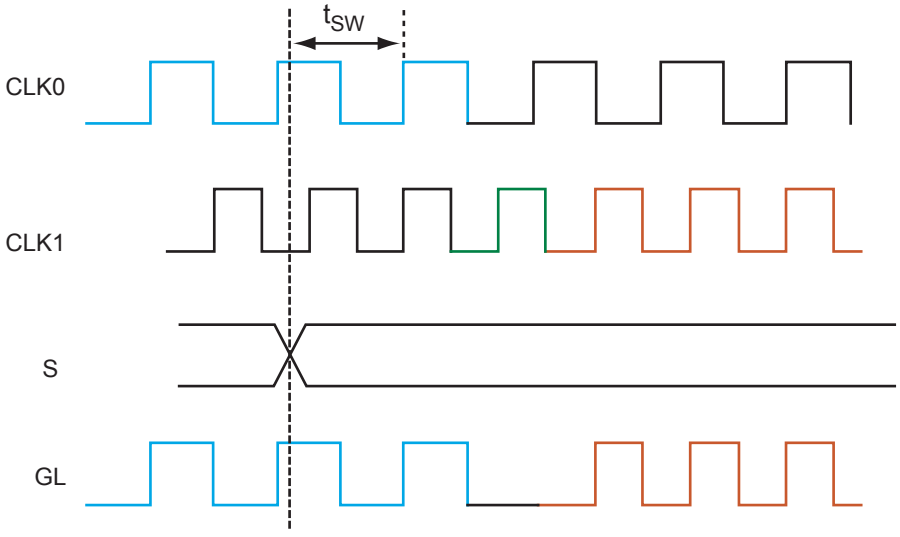
NGMUX Modes of Operation

The signals driving NGMUX have the same specifications as the output clock of the PLL/CCC. The following examples show various scenarios for the switching sequence between two asynchronous clock domains CLK0 and CLK1.

Case 1: Both CLK0 and CLK1 Active

When both the CLK0 and CLK1 inputs to the NGMUX are active, the switching sequence between the two clock sources (from CLK0 to CLK1) is as below. An example is shown in Figure 4-6.

1. A transition on S initiates the clock source switch.
2. GL drives one last complete CLK0 positive pulse (i.e., one rising edge followed by one falling edge).
3. GL stays LOW until the second rising edge of CLK1 occurs.
4. At the second CLK1 rising edge, GL continuously delivers CLK1.



Note: Minimum t_{SW} = 0.05 ns at 35°C (typical conditions)

Figure 4-6 • **Active CLK0/CLK1 Inputs to NGMUX**

Case 2: CLK0 Stopped or at Very Low Frequency

If CLK0 stops or runs at a very low frequency after S transition, the timeout circuitry inside the FPGA is used. The sequence of switching between the two clock sources (from CLK0 to CLK1) is described and illustrated below.

Case 2A: No Rising CLK0 Edge

If CLK0 does not have a rising edge before the seventh CLK1 rising edge, the switching sequence between the two clock sources (from CLK0 to CLK1) is as shown in Figure 4-7.

1. At the seventh CLK1 rising edge, GL will go LOW until the ninth CLK1 rising edge.
2. At the ninth CLK1 rising edge, GL will continuously deliver the CLK1 signal.

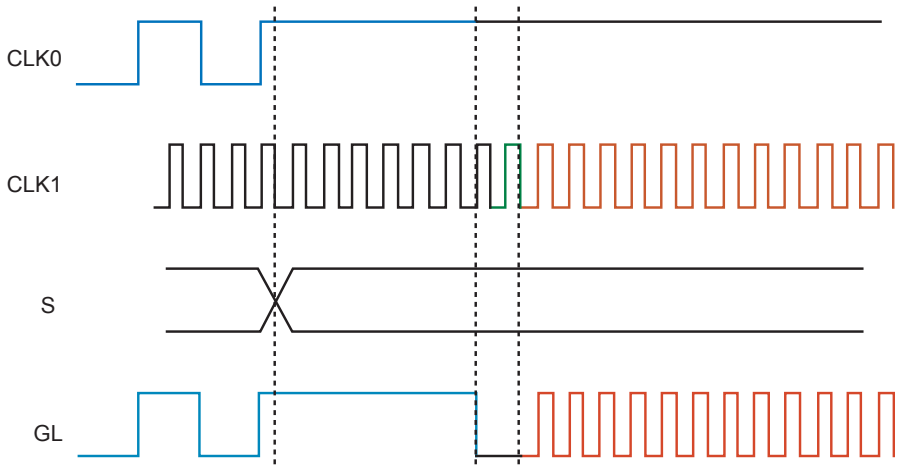


Figure 4-7 • **Low-Frequency CLK0 after S Transition, No Rising CLK0 Edge before Seventh Rising CLK1 Edge**

Case 2B: No Falling CLK0 Edge

If a CLK0 rising edge occurs before the seventh CLK1 rising edge but a CLK0 falling edge does not occur before the fifteenth CLK1 rising edge, the sequence of switching between the two clock sources (from CLK0 to CLK1) is as shown in Figure 4-8.

1. At the fifteenth CLK1 rising edge, GL will go LOW until the seventeenth CLK1 rising edge.
2. At the seventeenth CLK1 rising edge, GL will continuously deliver the CLK1 signal.

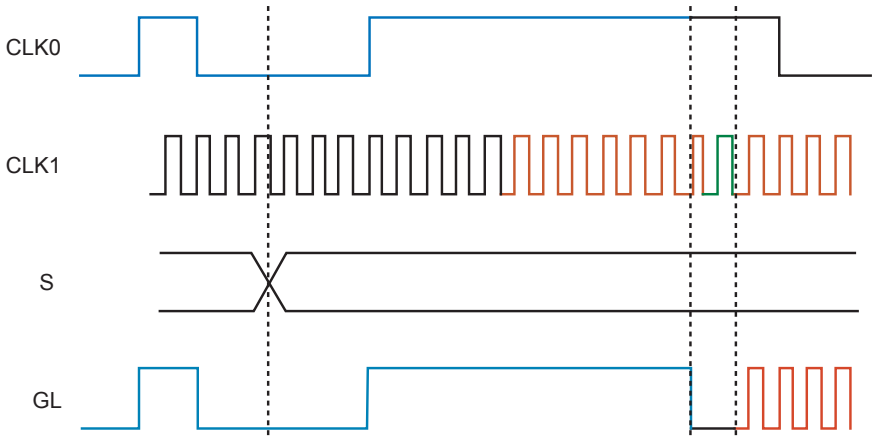


Figure 4-8 • **Low-Frequency CLK0 after S Transition, Rising CLK0 Edge before Seventh Rising CLK1 Edge**

NGMUX Usage

The software implementation of the NGMUX has been simplified to a 2:1 multiplexer, as shown in Figure 4-5 on page 124. The two clock input ports are CLK0 and CLK1, and the output clock port is GL. The allowable inputs to the NGMUX are as follows:

- The GLA and GLC outputs of a PLL
- The GLA output of a PLL and GLINT (the fanout of GLINT must be 1)
- The GLA output of a PLL and an internal net

SmartGen can also be used to implement these macros. For more information on using SmartGen, refer to the *SmartGen, FlashROM, Analog System Builder, and Flash Memory System Builder User's Guide*.

The following example manually instantiates the No-Glitch Multiplexer using the NGMUX macro and connects the CLK0 and CLK1 ports to the output ports of a SmartGen-generated PLL.

Verilog

```
module myCLKMUX (
  SYSCLK,
  CLK75MHZ,
  CLKSEL
);

  input CLKSEL;
  input CLK75MHZ;
  output SYSCLK;

  wire iGLA, iGLC;

  NGMUX uNGMUX (
    .CLK0 (iGLA),
    .CLK1 (iGLC),
    .S  (CLKSEL),
    .GL  (SYSCLK)
  );

  myPLL myPLL1 (
    .POWERDOWN (1'b1),
    .CLKA ( CLK75MHZ),
    .LOCK (),
    .GLA  ( iGLA),
    .GLC  (iGLC),
    .OADIVRST (NSYSRESET)
  );

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity my CLKMUX is
  port (
    SYSCLK: out std_logic;
    CLK75MHZ: in std_logic;
    CLKSEL: in std_logic
  );
end entity my CLKMUX;

architecture my CLKMUX is
  signal iGLA : std_logic;
  signal iGLC : std_logic;

  component NGMUX
  port (
    CLK0: in std_logic;
    CLK1: in std_logic;
    S: in std_logic;
    GL : out std_logic
  );

  component myPLL
  port (
    POWERDOWN: in std_logic;
    CLKA: out std_logic;
    LOCK: out std_logic;
    GLA: out std_logic;
    GL C: out std_logic;
    OADIVRST: in std_logic
  );

begin
  uNGMUX : NGMUX
  port map (
    CLK0 => iGLA,
    CLK1 => iGLC,
    S => CLKSEL,
    GL => SYSCLK
  );

  myPLL1 : myPLL
  port map (
    POWERDOWN => '1',
    CLKA => CLK75MHZ,
    LOCK => open,
    GLA => iGLA,
    GL C => iGLC,
    OADIVRST => NSYSRESET
  );

end architecture my CLKMUX;
```

NGMUX Tips

The following design considerations are recommended when using the NGMUX:

- The NGMUX has a fixed design location and is intended to be placed downstream from the PLL.
- Hardwire the NGMUX CLK0 input to PLL output GLA, as GLA must drive the NGMUX CLK0 input. GLA can only have a fanout of one, as the GLA global driver is used for the NGMUX output.
- If the two inputs to the NGMUX are PLL outputs GLA and GLC, you may lose the global network driver for GLC because it is consumed by the PLL output. Since the global network in Fusion is segmented, local clock networks can be used even though the whole global network is not available.
- Since the NGMUX macro has a fixed location (downstream from the PLL), routing delay can occur when the input to NGMUX comes from a regular net.
- The NGMUX GL output uses the GLA global network.

NGMUX Timing Analysis

Timing analysis verifies the functionality of the design with timing information. To check the design functionality for NGMUX, designers should check both the static and dynamic timing analyses as follows.

NGMUX Static Timing Analysis

Static timing analysis on both clock inputs is performed separately using the SmartTime tool in Designer (Figure 4-9). Run setup and hold checks on the source pins of CLK0 and CLK1.

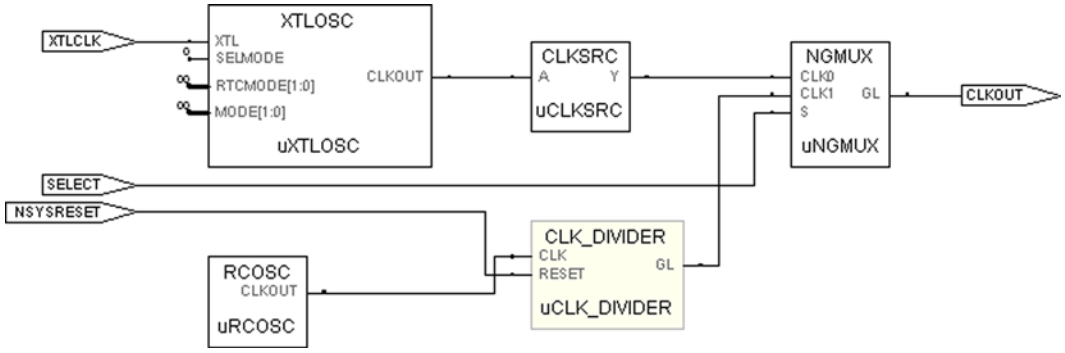


Figure 4-9 • **Static Timing Analysis Example**

Figure 4-9 on page 128 shows how the two inputs of the NGMUX are connected from the uCLK_DIVIDER and uCLKSRC components. In this instance, setup and hold time checks in SmartTime are done for both clocks, uCLK_DIVIDER/Inst1:GL and uCLKSRC:Y, as shown in Figure 4-10.

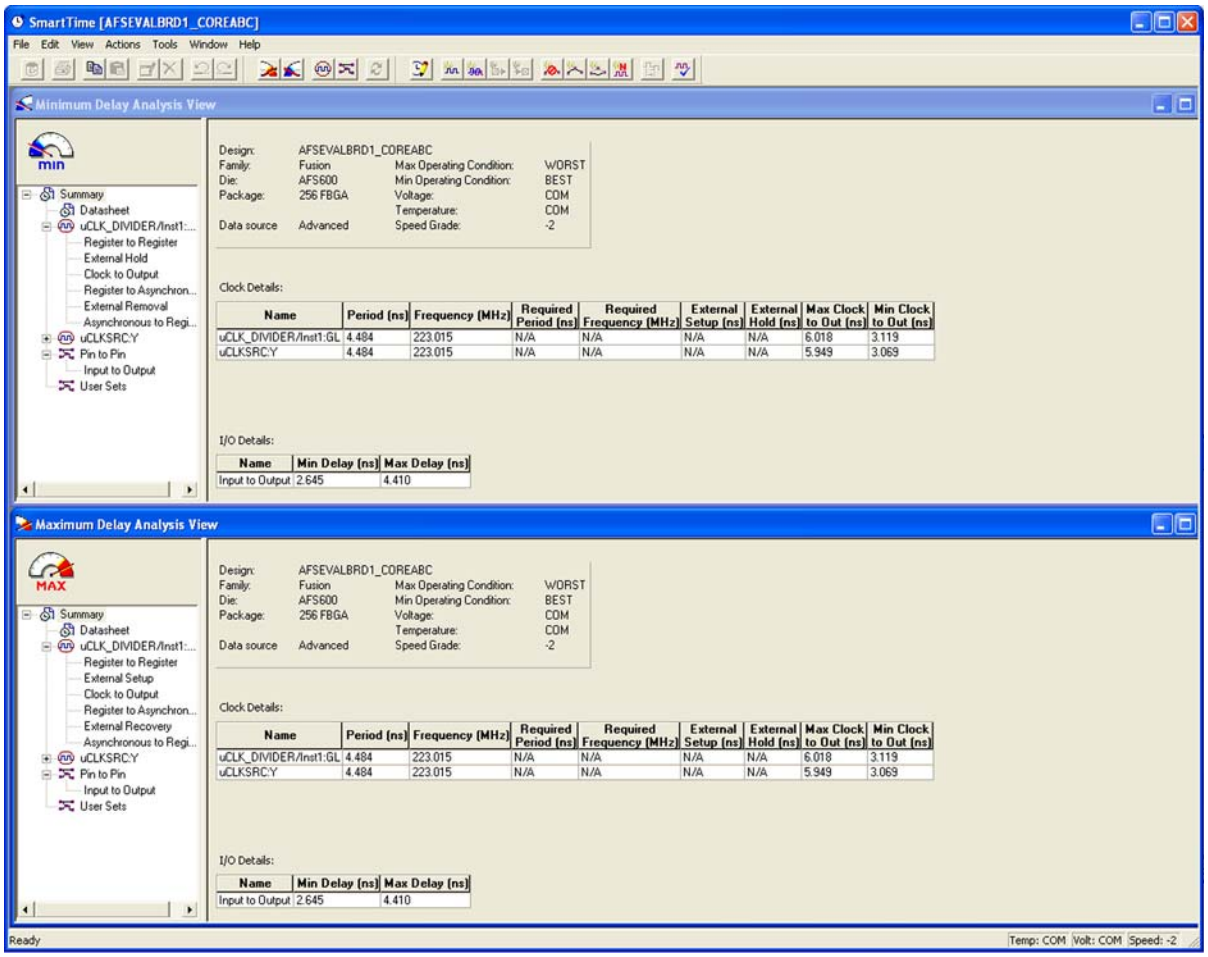


Figure 4-10 • **Checking Setup and Hold Times for Clocks**

NGMUX Dynamic Timing Analysis

For dynamic timing analysis, run a back-annotated timing simulation using the ModelSim® tool, and check the NGMUX signals, as shown in Figure 4-11.

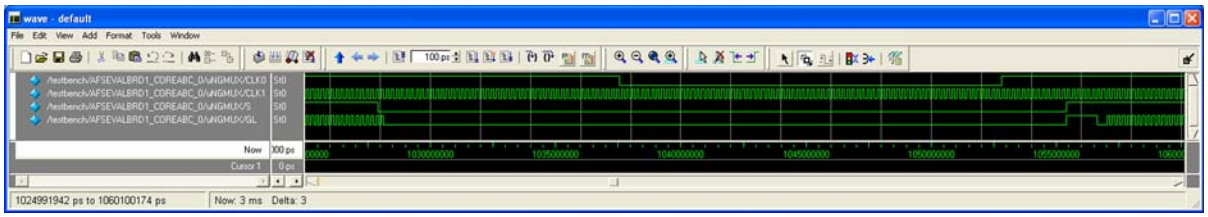


Figure 4-11 • **Dynamic Timing Analysis for NGMUX Signals**

A transition of S from High to Low initiates a switch to CLK0, and from Low to High initiates a switch to CLK1. The output of the NGMUX is undefined if S switches again before the previous switch operation has completed.

NGMUX Connections

The NGMUX has two input ports that are intended to be connected to clock signals. Connect the CLK0 input to a net driven by a clock signal. The fanout of the net connecting CCC:GLA to NGMUX:CLK0 should always be one. The driver of this clock signal will automatically be placed in the GLA tile for the CCC location in which the NGMUX is placed.

There are two possible connections for the NGMUX:CLK1 input:

1. The CLK1 port can be driven by a clock signal. The fanout of the net connecting CCC:GLC to NGMUX:CLK1 should always be one. The driver will automatically be placed in the CCC:GLC tile for the location in which the NGMUX is placed.
2. The CLK1 port can also be driven by a routed net; in this case, there is no restriction on the placement or fanout of the logic/net driving NGMUX:CLK1.

The integrated Fusion oscillators cannot drive the NGMUX directly, as they do not produce the clock signals. You must connect them to the NGMUX inputs through a valid clock macro (i.e., CLKSRC), refer to "Internal RC Oscillator" on page 110 and "Crystal Oscillator (XTLOSC)" on page 114 for more information.

NGMUX Placement

The NGMUX macros are placed automatically during layout. The NGMUX macros can be manually placed by doing the following:

1. **Placing NGMUX.** One of the two available locations for NGMUX must be chosen: TILE5 of the central CCC locations (shown in yellow in Figure 4-12).
2. **Placing the driver for CLK0**. The driver for CLK0 has to be a CCC macro that can be placed in the CCC:GLA tile (shown in green in Figure 4-12). The CCC macros that can be placed here are as follows:
   - CLKBUF (only non-V_{REF} versions)
   - CLKBIBUF
   - CLKSRC
   - CLKDLY
   - CLKDIVDLY
   - PLL
3. **Placing the driver for CLK1**. If the driver for CLK1 is a CCC macro, it infers a hardwired connection. This macro must be placed in the CCC:GLC tile (shown in pink in Figure 4-12) and has the same CCC macro restrictions as CLK0.

   When the CLK1 port drives a PLL:GLC instance, the PLL:GLA instance of the same PLL must become the driver for CLK0.

   When the CCC macro is driven from the hardwired I/O, placing the I/O controls the placement of the CCC macro.

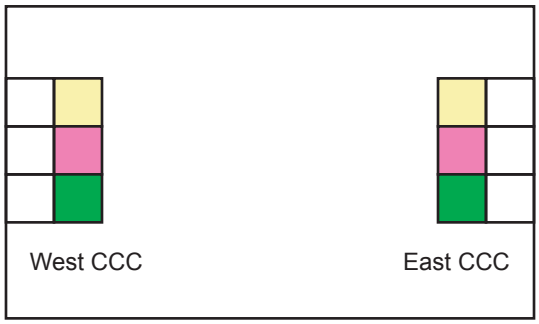


***Figure 4-12* • NGMUX Interconnects**

Real-Time Counter (RTC)

The addition of the real-time counter enables the Fusion mixed signal FPGA to support both standby and sleep modes of operation, greatly reducing power consumption in many applications. The RTC also provides implementation of a time and date calendar, enabling embedded systems to log data with time and date stamps.

RTC Usage

The RTC resides within the Fusion Analog Block and has the following features and requirements:

- The RTC must be driven by the crystal oscillator circuit, and the crystal oscillator must be configured to operate in RTC mode.
- The MATCH signal on the output of the RTC system asserts when the value in the counter matches the value specified in the match register.
- There is an optional output RTCPSMMATCH that is triggered on a match. The RTCPSMMATCH signal can be used to signal the internal voltage regulator to power up/down and must be connected to the RTCPSM macro so the voltage regulator actives when the MATCH signal is asserted.

The MATCH signal asserts when the counter is equal to the value contained in MATCHREG. MATCHREG is a 40-bit register located in the ACM.

The RTC count register (COUNTER) can be preloaded with a zero or non-zero start value. The default value is zero. This is also a 40-bit register located in the ACM.

The control/status register (CTRL_STAT) is an 8-bit register located within the ACM that defines the operation of the RTC. The control register can reset the RTC, enabling operation to begin with all zeroes in the counter. The RTC can be configured to clear upon a match with the match register, or it can continue to count while the match signal is still asserted. Designers can also enable the Fusion device to power on at a specific time or at periodic intervals. For more information on the CTRL_STAT register, refer to the *Fusion Mixed Signal FPGAs* datasheet.

SmartGen can also be used to implement these macros. For more information on using SmartGen, refer to the *SmartGen, FlashROM, Analog System Builder, and Flash Memory System Builder User's Guide*.

The following example manually instantiates the crystal oscillator using the XTLOSC macro and connects the external 32.768 KHz crystal output to the RTC in the AB. Since the 32.768 KHz clock output is not connected to FPGA core logic, the CLKSRC macro is not needed. The example below assumes that the ACM has been previously configured and is controlling the functionality of the RTC. For more information on the ACM refer to the "Designing the Fusion Analog System" section on page 233.

Verilog

```
module myRTC (
  CLK10MHZ,
  CLK32kHz
);

  input CLK10MHZ;
  input CLK32kHz;

  wire iRTCCLK, iRTCSELMODE;
  wire [1:0] iRTCMODE;

  wire iACMCLK, iACMWEN, iACMRESET;
  wire [7:0] iACMADDR, iACMRDATA, iACMWDATA;

  XTLOSC uXTLOSC (
    .XTL  (CLK32kHz),
    .CLKOUT (iRTCCLK),
    .SELMODE (iRTCSELMODE),
    .MODE (2'b0),
    .RTCMODE (iRTCMODE)
  );

  AB uAB (
    // Note: Several of the Analog Block signals
    //  have been omitted from this
    //  example, only the critical signals
    //  are present.

    .SYSCLK (CLK10MHZ),

    .ACMCLK (iACMCLK),
    .ACMWEN (iACMWEN),
    .ACMRESET (iACMRESET),
    .ACMWDATA (iACMWDATA),
    .ACMADDR (iACMADDR),
    (iACMRDATA),

    .RTCCLK (iRTCCLK),
    .RTCXTLSEL (iRTCSELMODE),
    .RTCXTLMODE (iRTCMODE),
    .RTCMATCH (),
    .RTCPSMMATCH ()
  );

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;

entity myRTC is
  port (
    CLK10MHZ: in std_logic;
    CLK32kHz: in std_logic
  );
end entity my RTC;

architecture my RTC is
  signal iRTCCLK : std_logic;
  signal iRTCSELMODE : std_logic;
  signal iRTCMODE : std_logic_vector(1 downto 0);

  signal iACMCLK : std_logic;
  signal iACMRESET : std_logic;
  signal iACMWEN : std_logic;
  signal iACMADDR : std_logic_vector(7 downto 0;
  signal iACMRDATA : std_logic_vector(7 downto 0;
  signal iACMWDATA : std_logic_vector(7 downto 0;

  component XTLOSC
  port (
    XTL: in std_logic;
    CLKOUT: out std_logic;
    SELMODE: in std_logic_vector(1 downto 0);
    RTCMODE: in std_logic_vector(1 downto 0);
    MODE: in std_logic_vector(1 downto 0)
  );

  component AB
  -- Note: Several of the Analog Block signals
  --  have been omitted from this
  --  example, only the critical signals
  --  are present.
  port (
    SYSCLK: in std_logic;

    ACMCLK: in std_logic;
    ACMWEN: in std_logic;
    ACMRESET: in std_logic;
    ACMADDR: in std_logic_vector(7 downto 0);
    ACMWDATA: in std_logic_vector(7 downto 0);
    ACMRDATA: out std_logic_vector(7 downto 0);

    RTCCLK: in std_logic;
    RTCXTLSEL: out std_logic;
    RTCXTLMODE: out std_logic_vector(1 downto 0);
    RTCMATCH: out std_logic;
    RTCPSMATCH: out std_logic
  );

begin

  uXTLOSC : XTLOSC
  port map (
    XTL => CLK32kHz,
    CLKOUT => iRTCCLK,
    SELMODE => iRTCSELMODE,
    MODE => “00”,
    RTCMODE => iRTCMODE
  );
```

```
  uAB : AB
  -- Note: Several of the Analog Block signals
  --  have been omitted from this
  --  example, only the critical signals
  --  are present.
  port map (
    SYSCLK => CLK10MHZ,

    ACMCLK => iACMCLK,
    ACMWEN => iACMWEN,
    ACMRESET => iACMRESET,
    ACMWDATA => iACMWDATA,
    ACMADDR => iACMADDR,
    ACMRDATA => iACMRDATA,

    RTCCLK => iRTCCLK,
    RTCXTLSEL => iRTCSELMODE,
    RTCXTLMODE => iRTCMODE,
    RTCMATCH => open,
    RTCPSMMATCH => open
  );

end architecture myRTC;
```

RTC Tips

The following design considerations are advised when using the RTC:

- When the RTC is not configured to reset the counter when a match occurs, the time interval between active RTCMATCH occurrences is equal to the total cumulative time count of the 40-bit RTC. In other words, the counter must overflow and reach the MATCHREG value again to create an active RTCMATCH output. The time required for the counter to overflow would not be practical for most applications; Microsemi recommends that the counter be reset upon a match condition if the RTCMATCH signal is needed.
- Each bit of the 40-bit COUNTER is compared to each bit of the 40-bit MATCHREG via XNOR gates, and the result is stored in the MATCHBITS register, enabling the designer to check whether an individual bit match has occurred.
- The location of the RTC registers within the ACM is shown in Table 4-3.

***Table 4-3* • Location of the RTC within the Analog Configuration MUX**

ACM_ADDR[7:0]	Register Name	Description
0x40	COUNTER0	Counter Bits [7:0]
0x41	COUNTER1	Counter Bits [15:8]
0x42	COUNTER2	Counter Bits [23:16]
0x43	COUNTER3	Counter Bits [31:24]
0x44	COUNTER4	Counter Bits [39:32]
0x48	MATCHREG0	Match Register Bits [7:0]
0x49	MATCHREG1	Match Register Bits [15:8]
0x4A	MATCHREG2	Match Register Bits [23:16]
0x4B	MATCHREG3	Match Register Bits [31:24]
0x4C	MATCHREG4	Match Register Bits [39:32]
0x50	MATCHBITS0	Individual Match Bits [7:0]
0x51	MATCHBITS1	Individual Match Bits [15:8]
0x52	MATCHBITS2	Individual Match Bits [23:16]
0x53	MATCHBITS3	Individual Match Bits [31:24]
0x54	MATCHBITS4	Individual Match Bits [39:32]
0x58	CTRL_STAT	Control / Status Register Bits
0x59	TEST_REG	Test Register

RTC Interconnection

Figure 4-13 shows the interconnection between the RTC and the various components in the Fusion device. If any hardwired input is not used, connect it to GND, and leave unused outputs floating (see the "Verilog" section on page 132 and the "VHDL" section on page 133 for an example of unused outputs left floating). For all hardwired connections, the fanout of the net connecting the two hardwired pins must be one.

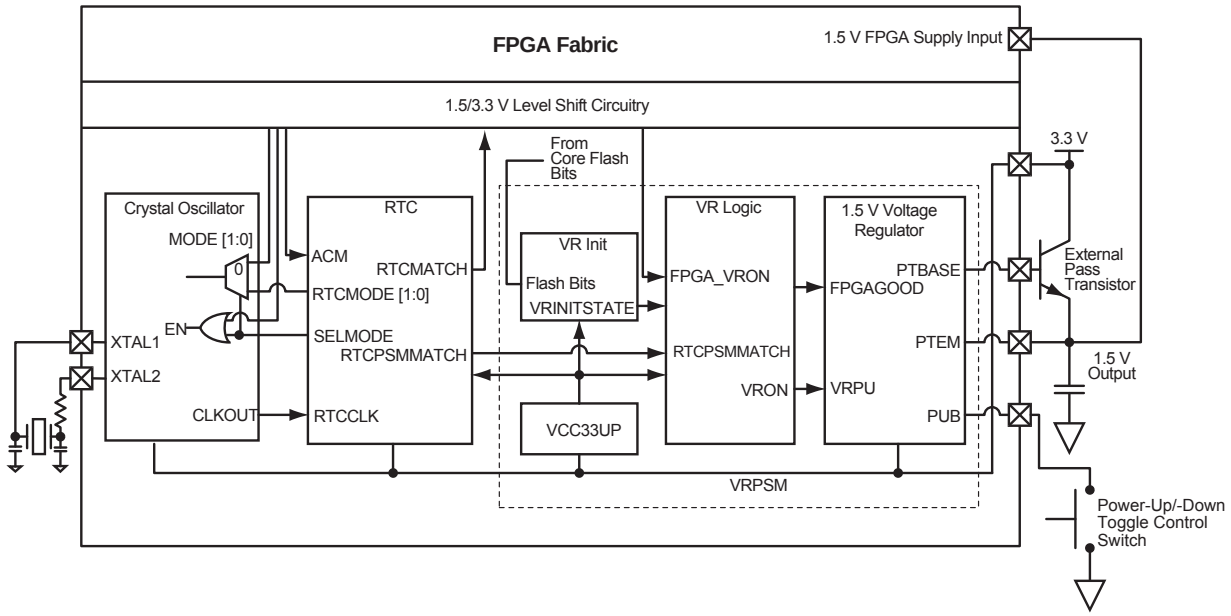


***Figure 4-13* • RTC Interconnection Diagram**

Related Documents

Datasheet

Fusion Mixed Signal FPGAs datasheet

http://www.microsemi.com/soc/documents/Fusion_DS.pdf

User's Guides

SmartGen, FlashROM, ASB, and Flash Memory System Builder User's Guide

http://www.microsemi.com/documents/genguide_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
v1.1 (August 2009)	The "Crystal Oscillator" section was revised to remove Table 4-1 · RC Oscillator Tips and Package Connections and Table 4-2 · Crystal Oscillator Mode Settings. The text of the "Crystal Oscillator Usage" section was replaced with new text and Figure 4-2 • XTLOSC Macro was replaced.	114
	Figure 4-3 • Crystal Oscillator: RC Time Constant Values vs. Frequency (typical) is new.	116

5 – Fusion Embedded Flash Memory Blocks

Microsemi has not only utilized flash memory technology to configure the Fusion FPGA core tiles, but also offers up to four embedded flash blocks (FBs), each 2 Mbits in density, for system initialization and general data storage. The embedded flash memory blocks are accessible by both on-chip and off-chip resources. Except for AFS090, the read and write bus width for all devices is 32. The read and write bus width for AFS090 is 16. Internally, the FPGA core fabric can directly access the FB's data bus. Externally, the embedded flash memory is accessible via the JTAG port or through interfaces implemented within the FPGA fabric: AMBA AHB (Advanced Microcontroller Bus Architecture Advanced High-Performance Bus), CoreCFI (Common Flash Interface), or a proprietary interface. It can be partitioned along page boundaries, giving designers better control over memory space usage, and the ability to individually protect each memory space against page loss, page overwrite, and external access. It also offers priority action control of the memory operations during simultaneous access requests.

The embedded flash memory can be configured using the Flash Memory System Builder in Microsemi Libero System-on-Chip (SoC) software, using the SmartDesign IP Deployment Platform (IDP) for microprocessor usage, and manually through RTL. The following sections discuss design usage details for each method of configuration.

Using the Embedded Flash Memory for Initialization

The Flash Memory System Builder in Libero SoC enables designers to easily configure the embedded flash memory via a simple GUI interface. The GUI interface provides FPGA designers the ability to quickly partition and configure the embedded flash memory for initialization or general data storage purposes. This section discusses the uses of the initialization IP offered in Libero SoC as a block within the embedded Flash Memory System.

The Flash Memory System Builder offers the ability to support five types of clients. A client is a design block whose functionality is dependent on or configured by the data stored in the embedded flash memory. These clients also utilize the custom IP functionality added to the flash memory control logic, only available through the Flash Memory Block System Builder. Two of the client types are used to add general-purpose data storage capability to the Flash Memory System. These clients are described in the "Using the Embedded Flash Memory for General Data Storage" section on page 158. The other three clients are described in the following subsections. As clients are added to the Flash Memory System Builder, the Flash Memory System is partitioned and configured accordingly. Once each client's memory partition is configured, the HDL code is generated for the Flash Memory System and is ready to be interfaced with the design project. Figure 5-1 on page 138 portrays the embedded flash with initialization IP system interconnects to its clients.

The Flash Memory Block System Builder supports the following initialization clients:

- The Analog System Client is used to configure the flash memory for the storage of the Fusion Analog System initialization and configuration parameters, which are stored in the spare pages of the flash memory.
- The RAM Initialization Client is used to create and configure a flash memory partition to store RAM initialization data to be reloaded at power-up for context-saving applications.
- The Standalone Initialization Client is used to create and configure a general-purpose flash memory partition to store initialization data interfaced to, for example, a RAM/FIFO or ROM emulation. The initialization interface must be custom-designed.

Each client spans a minimum of one page (128 bytes) and can span up to 2,048 pages, depending on the number of free pages available. The Analog System Client itself does not take any of the regular pages; it is stored entirely in the reserved (spare) pages.

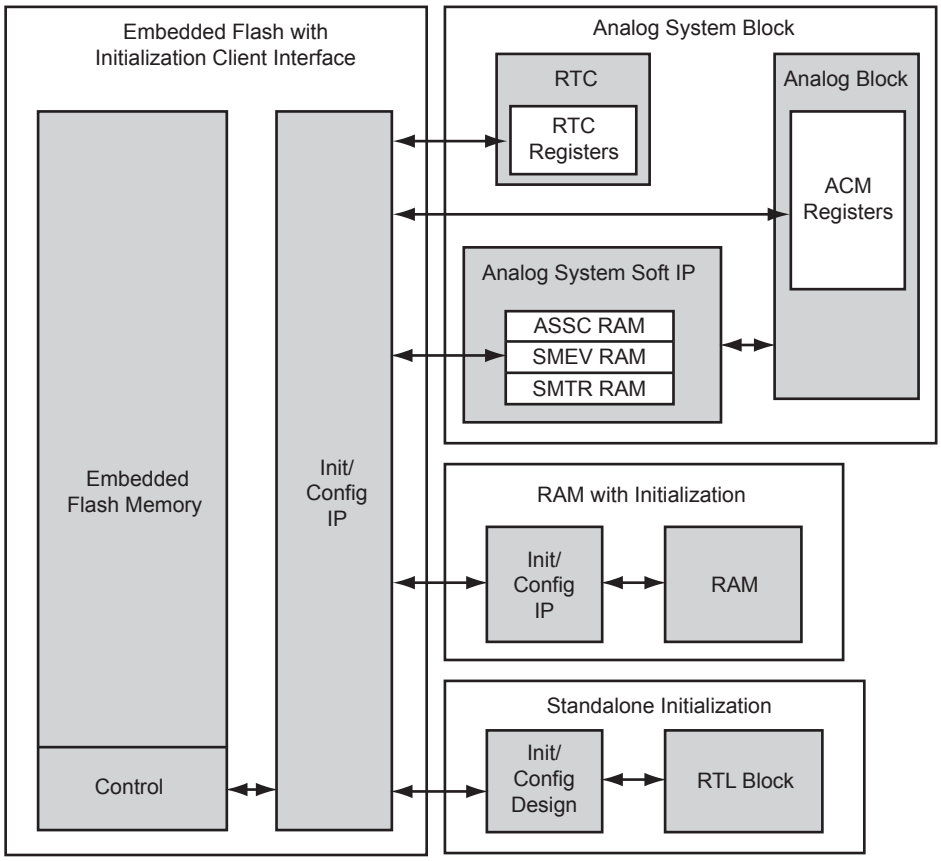


***Figure 5-1* • Embedded Flash with Initialization Client System Interconnects**

Embedded Flash Initialization IP Interface

The Flash Memory System Builder has the capability to generate the embedded flash with an integrated initialization IP circuit. The initialization circuit was designed to read data from embedded flash and store its contents in usable volatile registers or RAM for quick and easy accesses by the on-chip systems. The initialization circuit includes a common interface between the flash memory block and its supported clients. Specific write enable or data control signals are included in the circuit to control the write and read accesses between the clients and embedded flash.

The Analog System Client and RAM Initialization Client are the two key clients that interface directly to the flash initialization circuit. A user design block can also be interfaced to the embedded flash for initialization with the use of the Standalone Initialization Client. However, the initialization interface bridge must be designed by the user and added to the user block design.

Once the initialization clients and flash memory have been configured, the HDL is generated per the specifications entered in the Libero SoC GUI. The embedded flash memory module includes a common initialization interface between all clients and the control signals specific to the client. Table 5-1 on page 139 includes a list of all the common initialization interface ports and their descriptions.

***Table 5-1* • Flash Initialization Interface Common Client Ports**

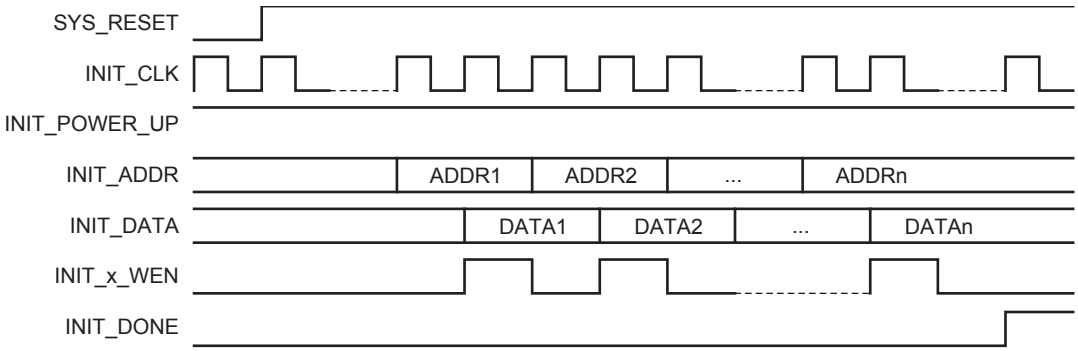
Flash Port	Direction	Description
SYS_RESET	Input	Asynchronous active-low system reset signal that will hold the flash memory in an initial state until released. This input is often common to the FPGA design's main reset.
INIT_CLK	Input	Initialization clock input whose maximum operating frequency is 10 MHz; it is rising-edge-active. For synchronous operation between the flash and its clients, its source clock should be common.
INIT_POWER_UP	Input	Active-high input to the flash, used to activate the initialization IP circuitry at power-up. INIT_POWER_UP must be HIGH at least until INIT_DONE transitions from LOW to HIGH. It can typically be tied HIGH unless performing on-demand updates to the system volatile registers as described in the "Managing the Initialization Process for Power Management" section on page 142. If being controlled, its transitions must be synchronous to INIT_CLK.
INIT_DONE	Output	Active-high initialization-done signal. Upon flash SYS_RESET, INIT_DONE defaults to LOW and remains LOW during the initialization activity. It transitions HIGH synchronously to INIT_CLK once the initialization process completes.
INIT_ADDR	Output	This 9-bit initialization address bus is common to the entire flash initialization system. During initialization and the save-to-flash activity, INIT_ADDR synchronously transitions with the rising edge of INIT_CLK. After flash SYS_RESET, INIT_ADDR defaults to 0x000. For each initialization client, INIT_ADDR begins at 0x000 and sequentially increments through all addresses in the flash page(s) assigned to the client's memory partition.
INIT_DATA	Output	The 9-bit initialization data bus is common to the entire flash initialization system. During the initialization activity, INIT_DATA synchronously transitions with the rising edge of INIT_CLK. After flash SYS_RESET, INIT_DATA defaults to 0x000. Depending on the client being initialized, either all nine bits or only the eight least significant bits are utilized as data.
INIT_x_WEN	Output	Active-high write enable (chip select) control signals from flash to the Analog System's volatile registers. INIT_x_WEN is a generic description for multiple signals to the Analog Block (AB). Refer to the "Analog System Client" section on page 143 for name specifics. Each signal is a 1-bit port from the Flash Block. After flash SYS_RESET, all signals default to LOW. During initialization activity, each signal synchronously transitions with the rising edge of INIT_CLK. Only one signal is activated at a time. Once activated, it will remain active until all addresses in the defined memory space are accessed.

***Table 5-1* • Flash Initialization Interface Common Client Ports (continued)**

Flash Port	Direction	Description
<RAM_CLIENT>_x_DAT_VAL	Output	Active-high data valid control signal (chip select) from the flash to the RAM block(s). <RAM_CLIENT>_x_DAT_VAL is a generic description for multiple signals to the RAM block. Refer to the "RAM Initialization Client" section on page 146 for name specifics. Each signal is a 1-bit port from the Flash Block. After flash SYS_RESET, all signals default to LOW. During initialization or save-to-flash activity, each signal synchronously transitions with the rising edge of INIT_CLK. Only one signal is activated at a time. Once activated, it will remain active until all addresses in the client's defined memory space are accessed.
<CLIENT_NAME>_<CHIP_SELECT>	Output	Active-high chip select for the initialization client from the flash to the defined block. <CLIENT_NAME>_<CHIP_SELECT> is a generic description for multiple signals to the block to be initialized. Refer to the "Standalone Initialization Client" section on page 150 for name specifics. Each signal is a 1-bit port from the Flash Block. After flash SYS_RESET, all signals default to LOW. During initialization or save-to-flash activity, each signal synchronously transitions with the rising edge of INIT_CLK. Only one signal is activated at a time. Once activated, it will remain active until all addresses in the client's defined memory space are accessed.

At power-up, after the flash reset is inactive, the initialization circuit reads the partitions from flash and writes the data to volatile memory. The INIT_POWER_UP signal must be HIGH, and INIT_DONE LOW, to trigger this process. If all clients are to initialize only at power-up, the INIT_POWER_UP signal can be tied HIGH. Once the initialization process begins, the INIT_DONE output flag is asserted LOW. During its LOW state, the main system design can use the INIT_DONE signal as a mask to prevent the system design from accessing the Analog System or the other clients being initialized during this process. None of the initialization clients should be used, including the on-chip ADC, until the initialization process completes.

The initialization circuit first initializes the Analog System block, followed by the RAM and other clients. For each client being initialized, a chip select or data valid control signal is produced by the Flash Memory System. As shown in Figure 5-2 on page 141, the INIT_x_WEN signal pulses HIGH when the write action to the volatile memory can be synchronously executed. Once the initialization process completes, the INIT_DONE signal transitions from LOW to HIGH, indicating that the process has completed and normal device operation can proceed. Figure 5-3 on page 141 shows a simplified simulation of the initialization process at power-up. The specific write enable or data valid signals are shown pulsing, activating the initialization process for each particular client in the system.



Note: INIT_x_WEN is a generic description of the write enable (or chip select) output signal behavior from flash to the volatile memory. Refer to the individual client sections for the exact signal names.

***Figure 5-2* • Basic Initialization Timing Diagram**

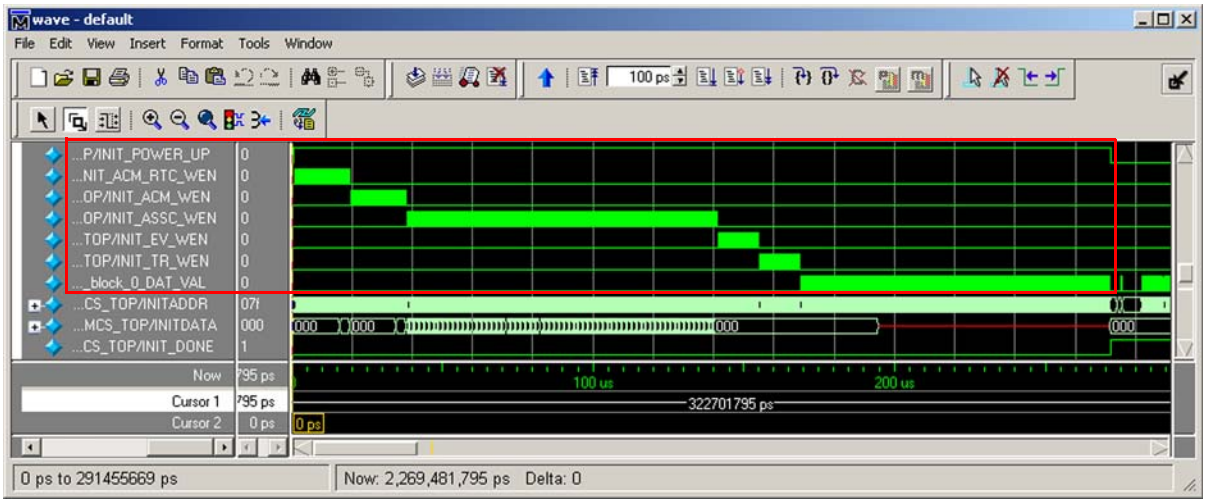


***Figure 5-3* • Complete Initialization Process at Power-Up**

Clock Configuration Options

The initialization process must operate at a frequency of 10 MHz or less. For a synchronous design, the flash module INIT_CLK input should also be connected to the flash initialization client's clock input pins. The Fusion No-Glitch MUX (NGMUX) macro can be used to switch between the slower frequency for the initialization process (INIT_CLK network) and a higher frequency that will utilize the full performance of the embedded Flash Block. Figure 5-4 shows an example of NGMUX usage in a system design.

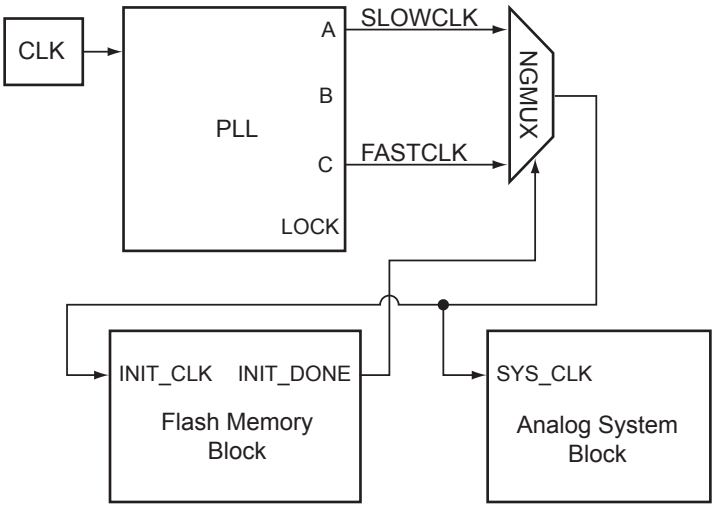


***Figure 5-4* • Analog System and Flash Memory Block NGMUX Example**

Managing the Initialization Process for Power Management

The initialization process can be managed to perform on-demand context (data) save and reload for power management purposes so critical data will not be lost in sleep or standby low-power modes. When entering a low-power mode, the main system design can perform a context save, for example, from RAM to flash. Once the context save is complete, the FPGA can be powered off or placed in low-power state without losing its critical data. Once the FPGA is to resume its normal operation, the initialization process begins performing the context reload operation with the preserved data from the last context save. For a complete Fusion context save and reload reference design, refer to the *Context Save and Reload* application note.

The Flash Initialization IP circuitry has the built-in capability to perform the context save and reload operations via the RAM Initialization Client and Standalone Initialization Client. The CLIENT_UPDATE input to the flash module is used to control the context save operation, and the INIT_POWER_UP signal is used to control the context reload. The "RAM Initialization Client" section on page 146 provides details on performing the context save operation.

After FPGA power-up, if the INIT_POWER_UP signal is HIGH, the initialization process will occur, performing the context reload. However, if an on-demand context reload operation is needed in the application, the initialization circuitry can be manipulated to perform addition context reload operations without powering the device down. The key is to clear the INIT_DONE signal so the HIGH INIT_POWER_UP can trigger a new context reload. This can be achieved by creating a controlled flash SYS_RESET signal (FLASH_RESET).

FLASH_RESET should be a registered active-low signal with a reset that is the main system reset for the design. If Fusion's internal Voltage Regulator Power Supply Monitor (VRPSM) is being used, a power-on reset pulse can be generated by connecting a simple external RC circuit to the 3.3 V power supply. For an internal power-on reset, if the PLL is being used in the design, the PLL lock signal can also serve as the FPGA system reset. After a system reset, the FLASH_RESET should default to HIGH. Once a command is received by the control logic to perform a context reload, the FLASH_RESET signal should be synchronously pulsed LOW, clearing the INIT_DONE signal and triggering a new initialization process, which performs the context reload. The initialization process will, however, initialize all its clients connected to the initialization interface. The write enable or data valid control signals to those clients that

should not be updated must be masked to LOW, preventing the writes from occurring. This is especially recommended for the Analog System module. Refer to the *Context Save and Reload* application note for complete implementation details. The INIT_DONE signal should also be masked to HIGH for the Analog System.

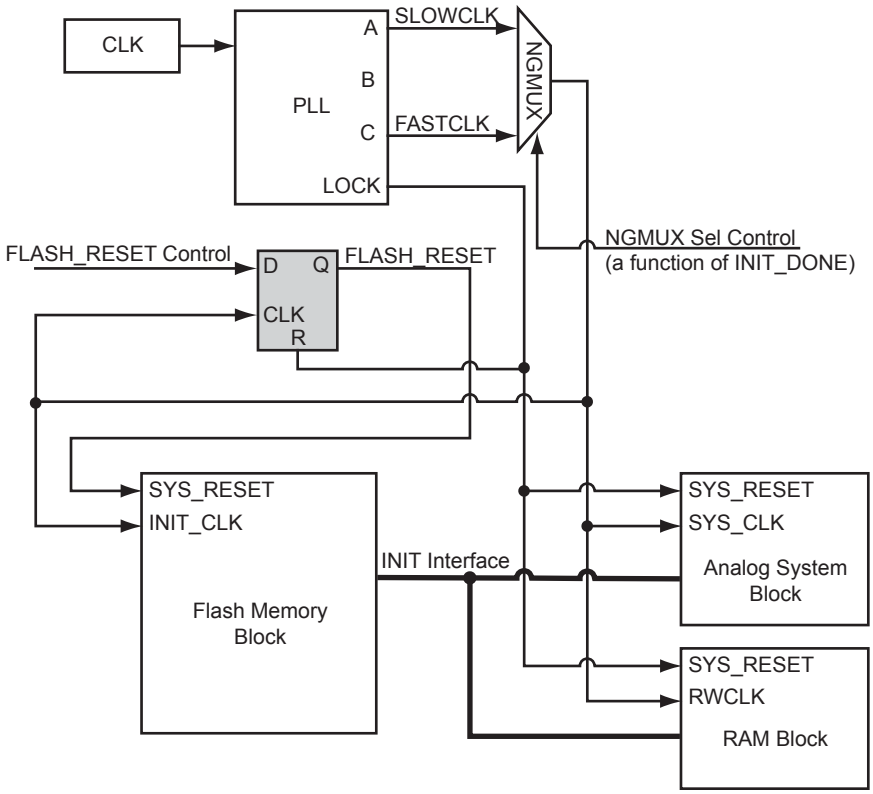


***Figure 5-5* • Reset and Clock Connection Diagram for On-Demand Context Reload**

Analog System Client

When creating the Analog System in Libero SoC using the Analog System Builder, a configuration file is generated and its data stored in the spare pages within the embedded flash memory during FPGA programming. The Flash Memory Analog System Client is used to create the memory partitions to store this configuration data.

The Analog System uses the embedded flash memory to hold the nonvolatile configuration data for the analog subsystem. After power-up and during the initialization process, the flash memory is read and the data stored in the Analog System's volatile register or RAM blocks within the analog subsystem.

The analog subsystem functions initialized during the initialization process are as follows (if selected during the Analog System configuration):

- Analog Configuration MUX (ACM)
- Programmable Real-Time Counter (RTC)
- ADC Sample Sequence Controller (ASSC)
- System Monitor Evaluation Phase State Machine (SMEV)
- System Monitor Transition Phase State Machine (SMTR)

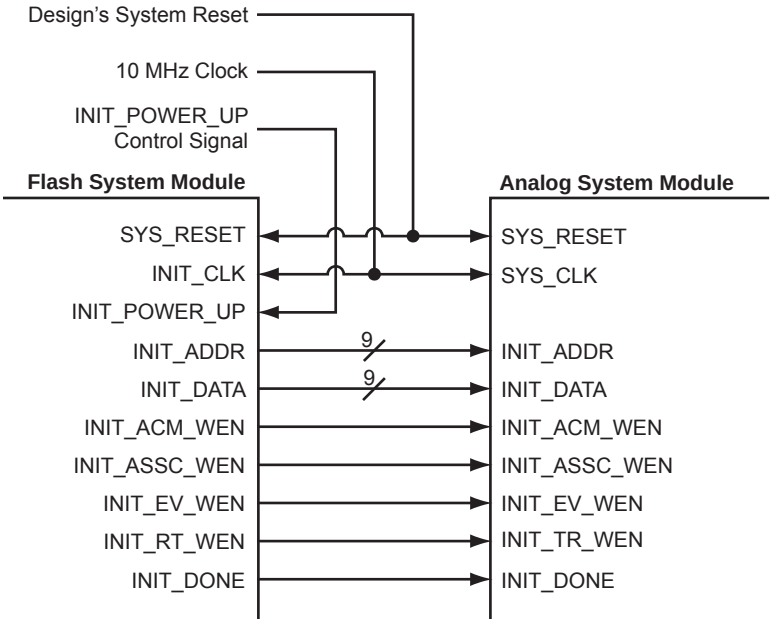
Analog System and Flash Memory Interconnects

The Flash Initialization IP circuitry includes a common interface connected to all clients, as described in Table 5-1 on page 139. However, each client includes interconnects that serve purposes specific to the client, such as the write enable signals to the Analog System register and RAM blocks. The write enable signals are active only during the initialization process triggered by the HIGH state of INIT_POWER_UP and the LOW state of INIT_DONE. Table 5-2 describes all Analog System Client–specific signals from the Flash Initialization IP circuit to the Analog System block.

Table 5-2 • **Analog System Client-Specific Flash Ports**

Flash Port	Direction	Description
INIT_ACM_RTC_WEN	Output	Active-high RTC peripheral chip select and write enable control signals from flash to the Analog System's RTC volatile registers. After flash SYS_RESET, INIT_ACM_RTC_WEN defaults to LOW. During the initialization activity, INIT_ACM_RTC_WEN synchronously transitions with the rising edge of INIT_CLK. Once activated, it will remain active until all addresses in the RTC memory space are accessed. No other chip select signals will be activated until the INIT_ACM_RTC_WEN activity completes.
INIT_ACM_WEN	Output	Active-high ACM registers' chip select and write enable control signals from flash to the Analog System's ACM volatile registers. After flash SYS_RESET, INIT_ACM_WEN defaults to LOW. During the initialization activity, INIT_ACM_WEN synchronously transitions with the rising edge of INIT_CLK. Once activated, it will remain active until all addresses in the ACM memory space are accessed. No other chip select signals will be activated until the INIT_ACM_WEN activity completes.
INIT_ASSC_WEN	Output	Active-high ASSC memory chip select and write enable control signals from flash to the Analog System's ACM RAM block. After flash SYS_RESET, INIT_ASSC_WEN defaults to LOW. During the initialization activity, INIT_ASSC_WEN synchronously transitions with the rising edge of INIT_CLK. Once activated, it will remain active until all addresses in the ACM memory space are accessed. No other chip select signals will be activated until the INIT_ASSC_WEN activity completes.
INIT_EV_WEN	Output	Active-high SMEV memory chip select and write enable control signals from flash to the Analog System's SMEV RAM block. After flash SYS_RESET, INIT_EV_WEN defaults to LOW. During the initialization activity, INIT_EV_WEN synchronously transitions with the rising edge of INIT_CLK. Once activated, it will remain active until all addresses in the SMEV memory space are accessed. No other chip select signals will be activated until the INIT_EV_WEN activity completes.
INIT_TR_WEN	Output	Active-high SMTR memory chip select and write enable control signals from flash to the Analog System's SMTR RAM block. After flash SYS_RESET, INIT_TR_WEN defaults to LOW. During the initialization activity, INIT_TR_WEN synchronously transitions with the rising edge of INIT_CLK. Once activated, it will remain active until all addresses in the SMTR memory space are accessed. No other chip select signals will be activated until the INIT_TR_WEN activity completes.

Figure 5-6 is a connection diagram showing the interconnects between the flash memory and the Analog System. If other flash memory clients exist in the system, they will also be connected to the common initialization interface ports.



Note: The above names represent the actual port names for the flash and Analog System HDL modules.

Figure 5-6 • **Analog System and Flash Memory Initialization Interface Connection Diagram**

Figure 5-7 is a timing diagram showing the write activity from flash memory to the Analog System's ASSC RAM block. At the positive edge of clock, the initialization circuit synchronously generates the address of the data to be written. At the following positive edge of the clock, the data is produced and INIT_ASSC_WEN is pulsed HIGH. The INIT_ASSC_WEN signal to the Analog System serves as an ASSC RAM chip select and write enable. The data is then synchronously written to the ASSC RAM memory on the next positive edge of the clock.

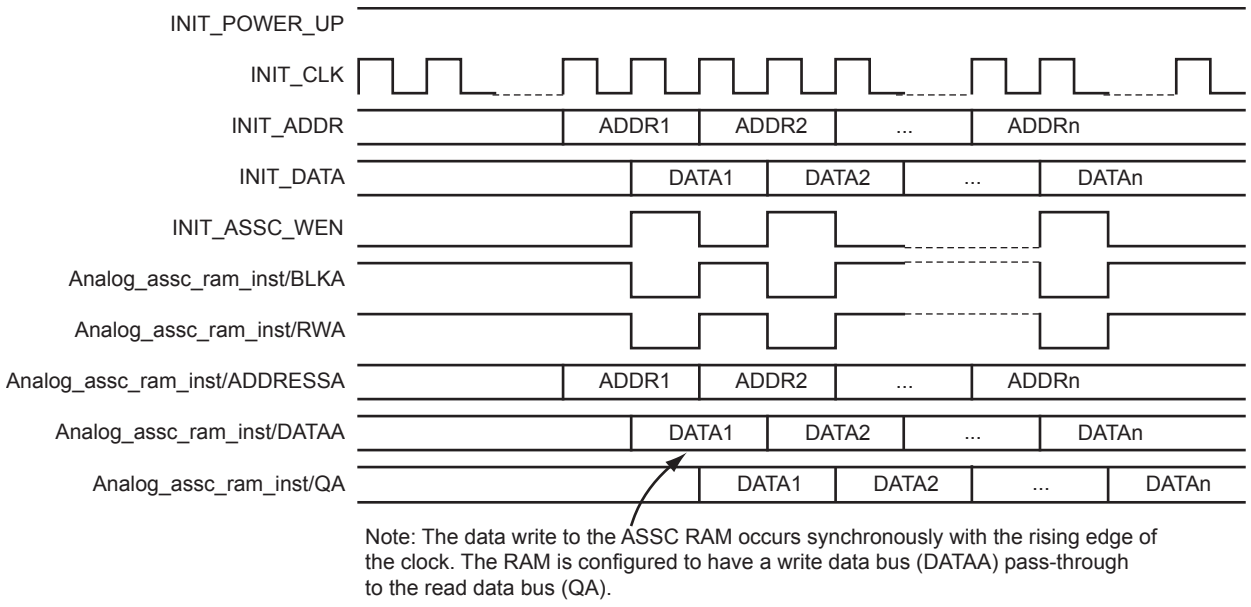


Figure 5-7 • **Initialization Interface Writing to ASSC RAM**

RAM Initialization Client

The RAM Initialization Client allows the user to create a flash memory partition to permanently store the RAM initialization data. After power-up, during the initialization process, the data stored in the RAM initialization flash memory partition is read, and its data is written to the RAM blocks. The Flash Memory System Builder allows for multiple RAM initialization flash memory partitions, giving the designer better flexibility over the system design's critical data. A JTAG read and/or write protection option is also available for each RAM Initialization Client.

When creating the RAM in Libero SoC, the **RAM with Initialization** option must be selected before generating the RAM so the initialization IP is included in the RAM block. The RAM initialization core's configuration GUI allows for the configuration of both two-port and dual-port RAMs with combined or separate read and write clocks. It also provides the ability to save the RAM contents back into flash via the **Enable On-Demand Save to Flash Memory** option in the RAM with Initialization configuration GUI. The initialization interface for the RAM shares the RAM's clocks; therefore, during the initialization or save-to-flash process, the RAM clock frequency must be no greater than 10 MHz.

RAM and Flash Memory Initialization Interconnects

The Flash Initialization IP circuitry includes a common interface connected to all clients, as described in Table 5-1 on page 139. However, each client includes interconnects that serve purposes specific to the client, such as the save-to-flash interface signals of the RAM with Initialization core IP. These signals are active only during the initialization process triggered by the HIGH state of INIT_POWER_UP while INIT_DONE is LOW, and during a save-to-flash process triggered by a HIGH state of CLIENT_UPDATE. Table 5-3 describes all signals from the Flash Initialization IP circuit specific to RAM with Initialization.

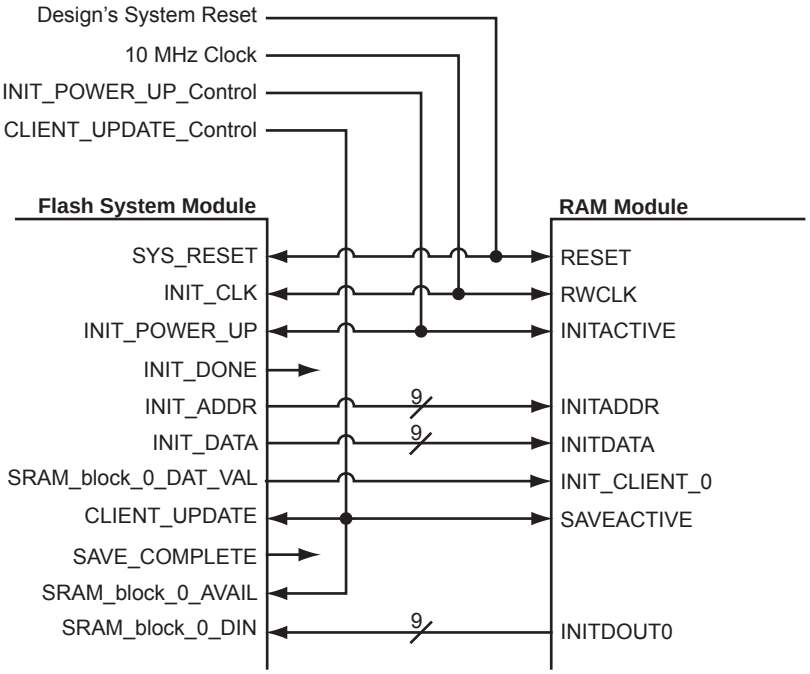
Table 5-3 • **RAM with Initialization Client-Specific Flash Ports**

Flash Port	Direction	Description
<RAM_CLIENT>_x_DAT_VAL	Output	Active-high data valid signal that behaves as a chip select for the RAM client. The number of data valid signals (labeled 0 to *x*) depends on the number of RAM blocks used to satisfy the RAM configuration selected in Libero SoC. Each signal is a 1-bit output port of the Flash Block. After flash SYS_RESET, the data valid signal(s) default to LOW. <RAM_CLIENT>_x_DAT_VAL synchronously transitions with the rising edge of INIT_CLK during initialization and save-to-flash activity. Once activated, it will remain active until all addresses in the RAM memory space are accessed. No other chip select signals will be activated until all <RAM_CLIENT>_x_DAT_VAL activity completes.
CLIENT_UPDATE	Input	Active-high control input signal used to trigger the save-to-flash process, which reads data from RAM and stores it to flash for context saving. CLIENT_UPDATE is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. CLIENT_UPDATE must be LOW at power-up and held LOW until the initialization process completes (INIT_DONE transitions from LOW to HIGH with INIT_POWER_UP held HIGH). To trigger the save-to-flash activity, CLIENT_UPDATE must be transitioned from LOW to HIGH synchronously with INIT_CLK. It must be held HIGH until the save-to-flash activity completes (SAVE_COMPLETE pulses HIGH). Once complete, CLIENT_UPDATE must then be transitioned LOW synchronously with INIT_CLK.

***Table 5-3* • RAM with Initialization Client-Specific Flash Ports (continued)**

Flash Port	Direction	Description
SAVE_COMPLETE	Output	Active-high output flag used to identify when the on-demand save-to-flash memory operation completes. SAVE_COMPLETE is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. After flash SYS_RESET, SAVE_COMPLETE defaults to LOW. The save-to-flash activity is triggered by the HIGH state of CLIENT_UPDATE. Once complete, SAVE_COMPLETE will transition HIGH and back LOW synchronously with INIT_CLK.
<RAM_CLIENT>_x_AVAIL	Input	Active-high data available input that behaves as a chip-select to the RAM-Initialization flash memory partition. The number is data valid signals (labeled '0' to 'x') depend on the number of RAM blocks used to satisfy the RAM configuration selected in Libero SoC. Each signal is a 1-bit input port of the Flash Block. <RAM_CLIENT>_x_AVAIL is only available if the "Enable On-Demand save to Flash Memory" feature was selected in the Libero SoC GUI. These input(s) can be connected directly to the CLIENT_UPDATE control input port. <RAM_CLIENT>_x_AVAIL must be Low at power-up and held Low the save to flash activity is triggered by the High state of CLIENT_UPDATE. <RAM_CLIENT>_x_AVAIL must transition High and held High until the save to flash activity completes. Once complete, <RAM_CLIENT>_x_AVAIL must be transitioned Low synchronously with INIT_CLK.
<RAM_CLIENT>_block_x_DIN	Input	The 9-bit save-to-flash data bus that supplies the data read from RAM to be written to flash. <RAM_CLIENT>_block_x_DIN is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. The number of data busses (labeled 0 to *x*) depends on the number of RAM blocks used to satisfy the RAM configuration selected in Libero SoC. Each data bus is a 9-bit input port of the Flash Block. During save-to-flash activity, <RAM_CLIENT>_block_x_DIN synchronously transitions with the rising edge of the RAM block's RWCLK, RCLK, or port-B clock (depending on the RAM configuration). After flash SYS_RESET, <RAM_CLIENT>_block_x_DIN defaults to 0x000. Depending on the client configuration, either all nine bits or only the eight least significant bits are utilized as data.

Figure 5-8 is a connection diagram showing the interconnects between the flash memory and RAM initialization ports. If other flash memory clients exist in the system, they will also be connected to the common initialization interface ports.



Note: The above names represent the actual port names for the flash and RAM HDL modules.

Figure 5-8 • **Two-Port RAM and Flash Memory Initialization Interface Connection Diagram**

Figure 5-9 is a timing diagram showing the write activity from flash memory to the RAM block through the initialization interface. At the positive edge of the clock, the initialization circuit synchronously generates the address of the data to be written. At the following positive edge of the clock, the data is produced and the SRAM_block_0_DAT_VAL signal is pulsed HIGH. The SRAM_block_0_DAT_VAL signal to the RAM serves as a chip select and write enable. The data is then synchronously written to the RAM on the next positive edge of the clock.

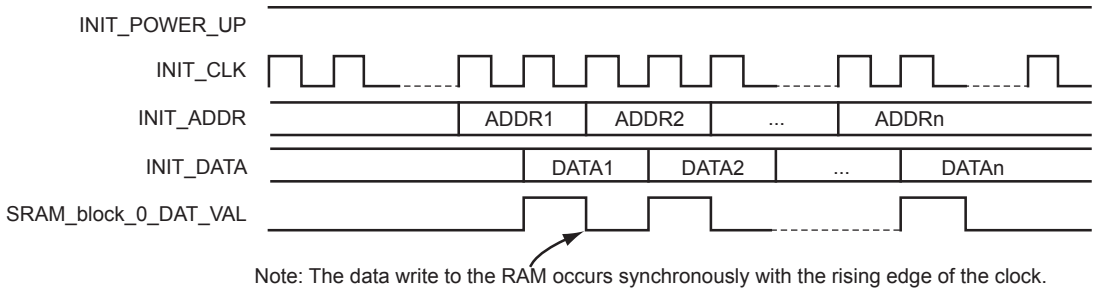


Figure 5-9 • **RAM Initialization Process Timing Detail**

Figure 5-10 on page 149 is a simulation example showing the complete save-to-flash process. To start the process, at the positive edge of the clock, CLIENT_UPDATE is transitioned HIGH. The flash initialization circuit provides the read address from RAM on the positive edge of the clock, as shown in Figure 5-11 on page 149. At the following positive edge of the clock, the data is read and put out from INITDOUT0. The flash initialization circuit then stores the data in the page buffer at the next positive edge of the clock. Once all data has been read from RAM and stored in flash, the SAVE_COMPLETE signal is

synchronously pulsed HIGH at the positive edge of the clock, holding a HIGH state for several clock cycles.

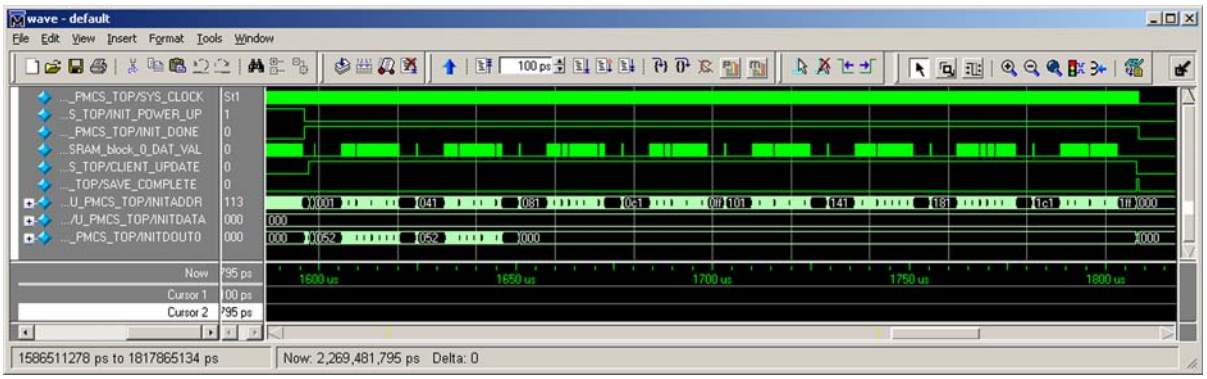


Figure 5-10 • **Complete Save-to-Flash Process**

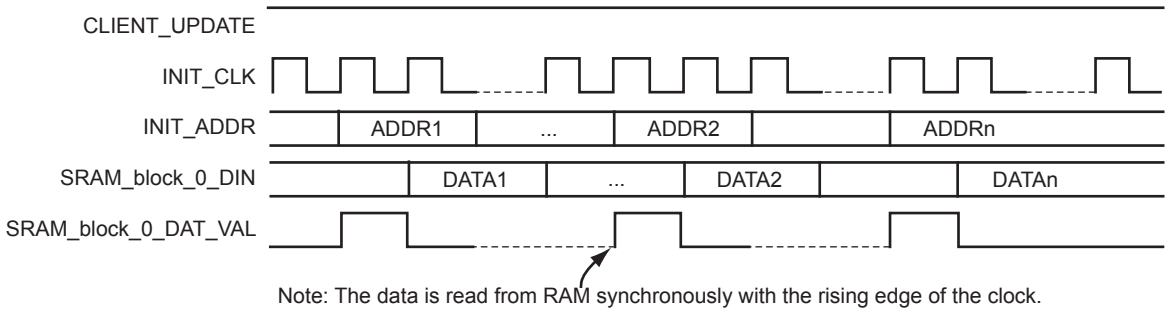


Figure 5-11 • **Save-to-Flash Timing Details**

Two-Port vs. Dual-Port RAM with Initialization Clock Configurations

The RAM with Initialization core IP can be configured with a single clock for both read and write operations, or with separate read and write clocks. Since the RAM initialization IP blocks share the standard RAM clock ports, the 10 MHz maximum frequency of the initialization interface must be considered when defining the system clocks.

For both the two-port and dual-port RAM with Initialization core configurations, if a single read and write clock is selected, the NGMUX macro can be used to switch between the slower frequency for the initialization or save-to-flash process and the faster frequency used to perform normal RAM accesses. Figure 5-5 on page 143 gives a possible clock configuration using the NGMUX as described.

For the two-port RAM with Initialization core configuration, if separate read and write clock ports are selected, the write clock port is used by the RAM during the initialization process. If the on-demand save-to-flash feature is enabled, the read clock port is used by the RAM during the save-to-flash process, and the read data bus is configured as non-pipelined.

For the dual-port RAM with Initialization core configuration, if the separate port-A and port-B clock option is selected, the port-A clock is used to perform the RAM write actions during the initialization process. If the on-demand save-to-flash feature is enabled, the port-B clock is used to perform the RAM read actions during the save-to-flash process, and the output data bus on port-B is configured as non-pipelined. If on-demand save-to-flash is not enabled, port-B is free to run at any desired frequency within the RAM operating range.

Both the initialization and save-to-flash processes must operate at a frequency no greater than 10 MHz. The NGMUX can be used to switch between a slower and a faster clock. For a synchronous design,

Microsemi recommends that a common clock source be used for all initialization modules, as shown in Figure 5-12.

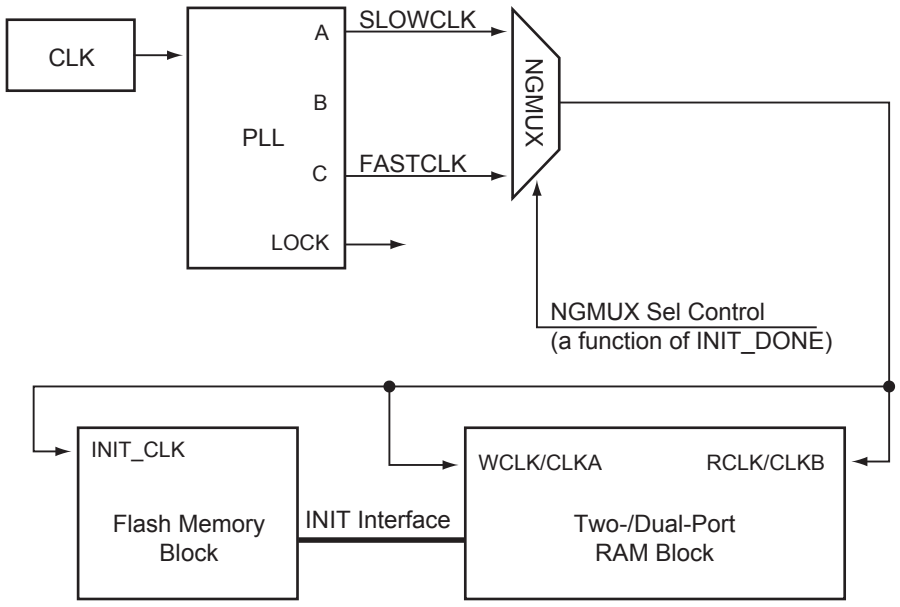


***Figure 5-12* • RAM Initialization Two-Clock Configuration Diagram**

Standalone Initialization Client

The standalone Flash Memory System initialization IP initializes all its clients with the data stored in the associated flash memory partition during the initialization process at power-up. The initialization client provides the ability to create a flash memory partition to initialize a desired block's volatile memory or registers. Once the partition has been created, access is given to the flash memory's initialization IP interface ports specific to the client's needs.

The Flash Memory System Builder's Initialization Client Libero SoC GUI allows the user to specify the client's name, starting address, word size, and memory-contents file. The memory-contents file supplies the initialization data values to be stored in the client's flash memory partition during FPGA programming. The user can also select the **Enable On-Demand Save to Flash Memory** feature, and protect the flash memory partition from JTAG access. And the user can name the client's chip select and save request ports.

Initialization Client Flash Memory Ports

The Flash Initialization IP circuitry includes a common interface connected to all clients, as described in Table 5-1 on page 139. However, each client includes interconnects that serve purposes specific to the client, such as the save-to-flash interface signals. These signals are active only during the initialization process triggered by the HIGH state of INIT_POWER_UP while INIT_DONE is LOW, and during a save-to-flash process triggered by a HIGH state of CLIENT_UPDATE. Table 5-4 on page 151 describes all Standalone Initialization Client–specific signals from the Flash Initialization IP circuit.

***Table 5-4* • Initialization Client-Specific Flash Ports**

Flash Port	Direction	Description
<CLIENT_NAME>_<CHIP_SELECT>	Output	Active-high chip select for the Standalone Initialization Client. After flash SYS_RESET, <CLIENT_NAME>_<CHIP_SELECT> defaults to LOW. During initialization and save-to-flash activity, <CLIENT_NAME>_<CHIP_SELECT> synchronously transitions with the rising edge of INIT_CLK. Once activated, it will remain active until all addresses in the client's memory space are accessed. No other chip select signals will be activated until the <CLIENT_NAME>_<CHIP_SELECT> activity completes.
CLIENT_UPDATE	Input	Active-high control input signal used to trigger the save-to-flash process, which reads data from client and stores it to flash for context saving. CLIENT_UPDATE is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. CLIENT_UPDATE must be LOW at power-up and held LOW until the initialization process completes (INIT_DONE transitions from LOW to HIGH with INIT_POWER_UP held HIGH). To trigger the save-to-flash activity, CLIENT_UPDATE must be transitioned from LOW to HIGH synchronously with INIT_CLK. It must be held HIGH until the save-to-flash activity completes (SAVE_COMPLETE pulses HIGH). Once complete, CLIENT_UPDATE must then be transitioned LOW synchronously with INIT_CLK.
SAVE_COMPLETE	Output	Active-high output flag used to identify when the on-demand save-to-flash memory operation completes. SAVE_COMPLETE is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. After flash SYS_RESET, SAVE_COMPLETE defaults to LOW. Save-to-flash activity is triggered by the HIGH state of CLIENT_UPDATE. Once complete, SAVE_COMPLETE will transition HIGH and back LOW synchronously with INIT_CLK.

***Table 5-4* • Initialization Client-Specific Flash Ports (continued)**

Flash Port	Direction	Description
<CLIENT_NAME>_<SAVE_REQUEST>	Input	Active-high save request input that behaves as a chip select to the initialization client's flash memory partition. The <CLIENT_NAME>_<SAVE_REQUEST> signal is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. These input(s) can be connected directly to the CLIENT_UPDATE control input port. The <CLIENT_NAME>_<SAVE_REQUEST> signal must be LOW at power-up and held LOW. Save-to-flash activity is triggered by the HIGH state of the CLIENT_UPDATE signal. The <CLIENT_NAME>_<SAVE_REQUEST> signal must transition HIGH and be held HIGH until the save-to-flash activity completes. <CLIENT_NAME>_<SAVE_REQUEST> must be transitioned LOW synchronously with INIT_CLK once complete.
<CLIENT_NAME>_DIN	Input	The 9-bit save-to-flash data bus that supplies the data read from initialization client to be written to flash. <CLIENT_NAME>_DIN is only available if the **Enable On-Demand Save to Flash Memory** feature was selected in the Libero SoC GUI. During the save-to-flash activity, <CLIENT_NAME>_DIN synchronously transitions with the rising edge of INIT_CLK. After flash SYS_RESET, <CLIENT_NAME>_DIN should default to 0x000. Depending on the client configuration, either all nine bits or only the eight least significant bits are utilized as data.

Initialization Client Usages

The Standalone Initialization Client provides designers the utmost flexibility, providing access to the Flash Initialization IP interface to set initial values—for example, for the Fusion synchronous FIFO, for ROM emulation, and for CoreABC (AMBA Bus Controller) IP instruction memory space. An initialization wrapper for the initialization clients must be generated by the designer (except when used with the CoreABC IP). An example of the initialization wrapper design for the Fusion synchronous FIFO is described in the "Fusion FIFO with Initialization Example" section on page 153.

To perform ROM emulation, the flash memory's initialization client and RAM blocks are required. A wrapper must be created providing the Flash Initialization IP interface write access to the RAM. The user interface should only include the read access ports of the RAM (RADDR, RD, RCLK, and REN). Depending on the data bus width, proper bus width handling must be taken into consideration in the wrapper design, since the initialization interface writes nine bits of data per chip select. Within the wrapper, the RAM write access ports may be set to a default state tying the WEN port HIGH. The WCLK port should be connected to the 10 MHz initialization clock network.

In some cases, the RAM with Initialization core IP may be the simplest approach for ROM emulation. The initialization interface IP is already included in the RAM core module. When generating the RAM with Initialization block, the two-port RAM with separate read and write clock ports is a recommended configuration. When creating the flash memory RAM Initialization Client partition, the **Enable On-Demand Save to Flash Memory** option should be cleared. All user RAM write access ports may be set to a default state, tying off the WEN port to HIGH. The WCLK port should be connected to the 10 MHz initialization clock network.

CoreABC is a simple, low-gate-count controller that uses the Flash Memory System's initialization client to initialize at power-up the RAM used for instruction code execution. A complete design example can be found in the *Design Example* section of the "Designing the Fusion Analog System" section on page 233.

Fusion FIFO with Initialization Example

When creating the FIFO with Initialization design, the flash memory initialization client and synchronous FIFO blocks are required. A wrapper must be created providing the Flash Initialization IP interface write access to the FIFO, and the save-to-flash interface read access. Depending on the data bus width, proper bus width handling must be taken into consideration in the wrapper design, since the initialization interface writes nine bits of data per chip select. Figure 5-13 shows the selected synchronous FIFO configuration used in this example.

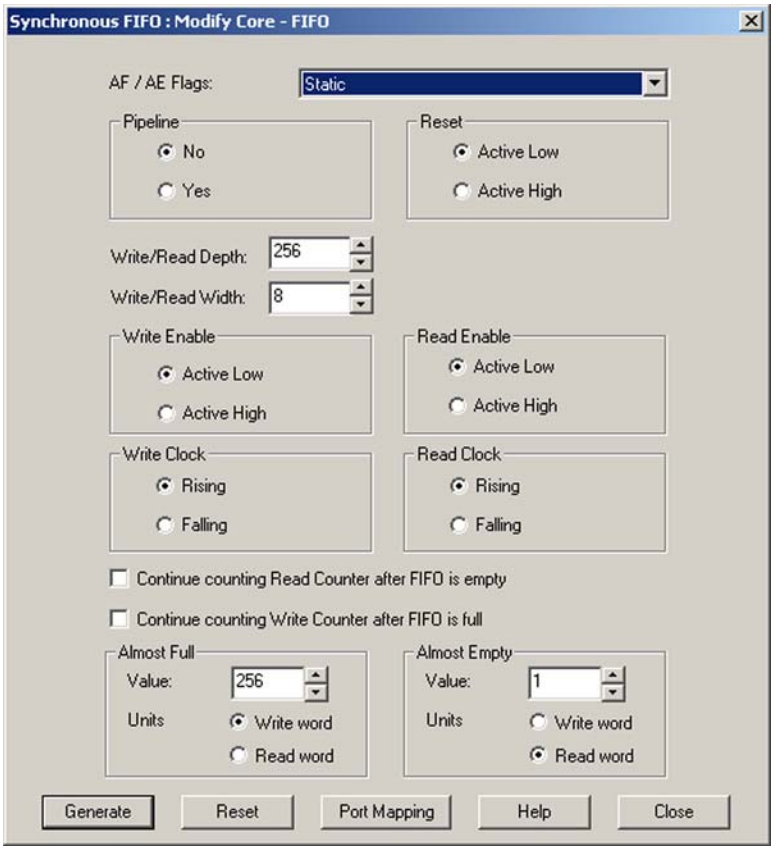


***Figure 5-13* • Synchronous FIFO Configuration**

The Flash Memory System Builder's initialization client is used to generate the flash memory partition for the FIFO. Figure 5-14 shows the required initialization client configuration based on the FIFO configuration shown in Figure 5-13 on page 153.

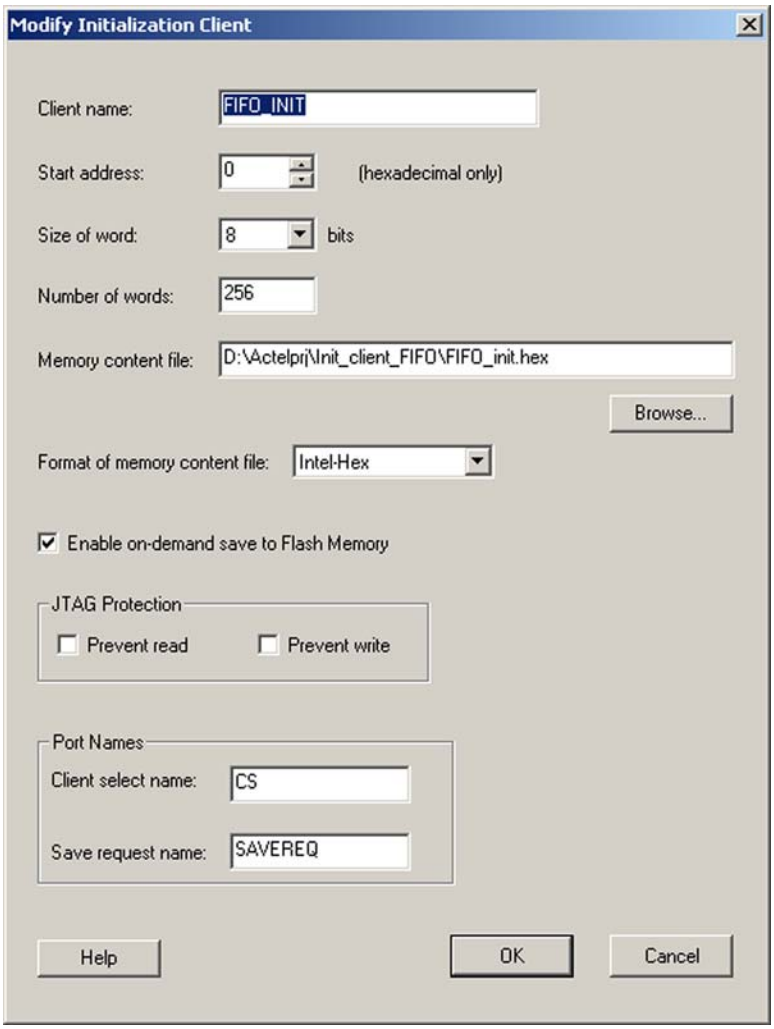
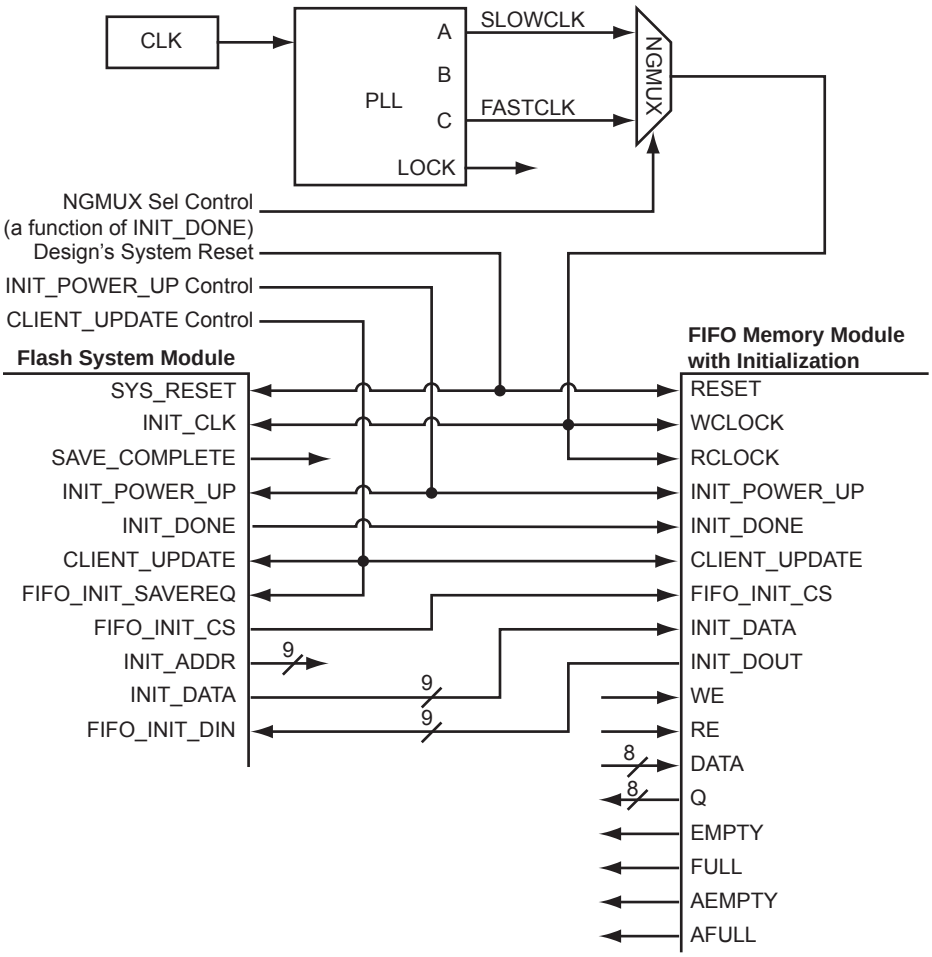


Figure 5-14 • **Initialization Client Configuration for the FIFO**

Once the flash memory block with the FIFO initialization partition has been generated, the following initialization ports, given in Verilog, are added to the Flash Block:

```
input         INIT_CLK;
input         SYS_RESET;
input         INIT_POWER_UP;
output        INIT_DONE;
output [8:0]  INIT_DATA;
output [7:0]  INIT_ADDR;
output        FIFO_INIT_CS;
input         FIFO_INIT_SAVEREQ;
input  [8:0]  FIFO_INIT_DIN;
output        SAVE_COMPLETE;
input         CLIENT_UPDATE;
```

The FIFO with Initialization interface wrapper must connect to the above ports. The INIT_ADDR port is not used in this design, since the Flash Block is being interfaced to a FIFO. Figure 5-15 shows the connections between the Flash Block and the FIFO with Initialization wrapper.



Note: The above names represent the actual port names for the flash and FIFO HDL modules.

Figure 5-15 • **FIFO with Initialization Interface Connection Diagram**

The FIFO with Initialization wrapper design should multiplex the control signals from the flash memory's initialization circuit with the FIFO's user access ports for a dual read and write access FIFO configuration. The following, given in Verilog, describes the main wrapper design:

```
assign FIFO_DATA = (INIT_POWER_UP & !INIT_DONE) ? INIT_DATA[7:0] : DATA;
assign FIFO_WEN = (INIT_POWER_UP & !INIT_DONE) ? !FIFO_INIT_CS : WE;
assign FIFO_REN = CLIENT_UPDATE ? !FIFO_INIT_CS : RE;
assign INIT_DOUT = {1'b0,Q};
FIFO U_FIFO(
  .DATA(FIFO_DATA),
  .Q(Q),
  .WE(FIFO_WEN),
  .RE(FIFO_REN),
  .WCLOCK(WCLOCK),
  .RCLOCK(RCLOCK),
  .FULL(FULL),
  .EMPTY(EMPTY),
  .RESET(RESET),
  .AEMPTY(AEMPTY),
  .AFULL(AFULL));
```

Figure 5-16 is a simulation example showing the FIFO's complete initialization process. A HIGH state on INIT_POWER_UP while INIT_DONE is LOW will trigger the initialization process. During this process, the FIFO should be in an empty state. The FIFO_INIT_CS signal is pulsed with the positive edge of the clock when a FIFO write operation should occur, as shown in Figure 5-17. With every HIGH state of FIFO_INIT_CS, INIT_DATA has valid data to be written into the FIFO. At the positive edge of the clock while FIFO_INIT_CS is HIGH, the data is synchronously written. Once all data values have been written into the FIFO, the FIFO's AFULL flag and the flash memory's INIT_DONE signals synchronously transition HIGH. The user must take care not to access the FIFO via the user access ports or the save-to-flash interface unless the INIT_DONE signal is HIGH.

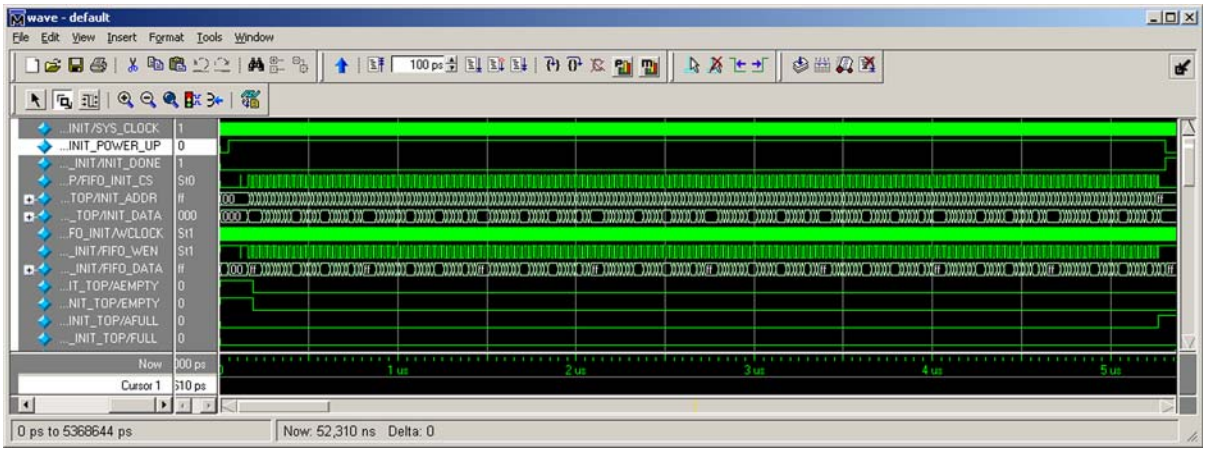


***Figure 5-16* • Complete FIFO Initialization Process**

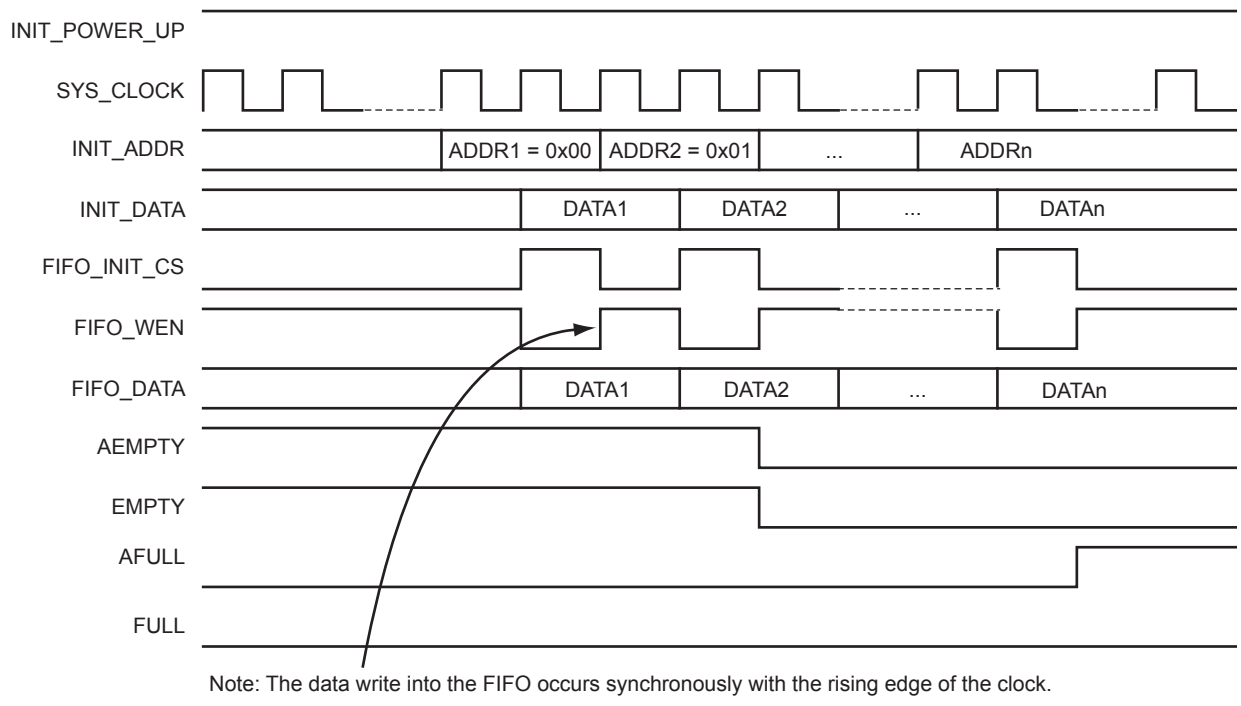


***Figure 5-17* • FIFO Initialization Write Operation Timing Details**

Figure 5-18 is a simulation example showing the FIFO's complete save-to-flash process. To start the process, CLIENT_UPDATE is transitioned HIGH at the positive edge of the clock. However, the user must take care not to activate CLIENT_UPDATE while INIT_DONE is LOW or if the FIFO is not full. The AFULL flag should be used to monitor the filled state of the FIFO, since the FULL flag will not transition HIGH unless the entire physical FIFO memory has been filled. In this example, although the FIFO is configured as 256×8 and all locations are written, the FIFO4K18 macro is instantiated for this configuration. The FIFO_INIT_CS signal is pulsed with the positive edge of clock when a FIFO read operation should occur, as shown in Figure 5-19. Once all data values have been read from the FIFO and stored in flash, the FIFO EMPTY signal and the flash memory SAVE_COMPLETE signal are synchronously transitioned HIGH at the positive edge of the clock. The SAVE_COMPLETE signal will hold a HIGH state for several clock cycles before transitioning LOW.

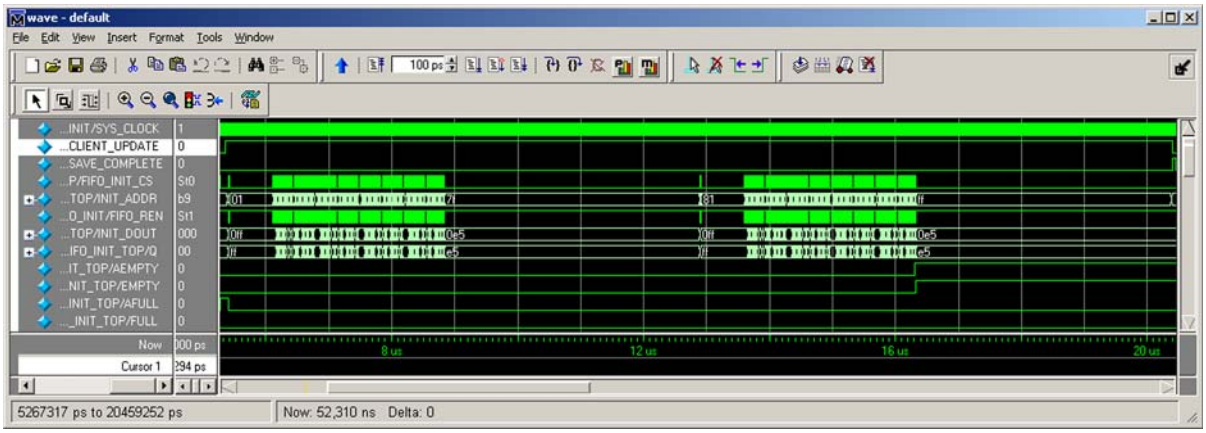


***Figure 5-18* • Complete FIFO Save-to-Flash Process**

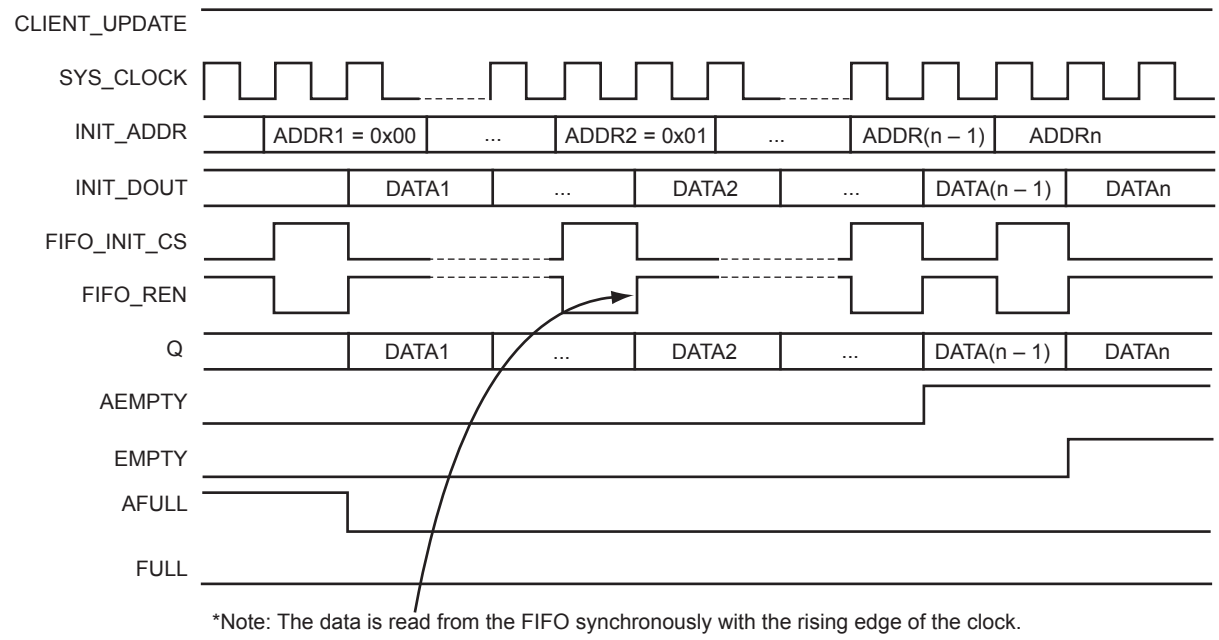


***Figure 5-19* • FIFO Save-to-Flash Read Operation Timing Details**

Using the Embedded Flash Memory for General Data Storage

A key feature of the embedded flash memory is its ability to be used as general data storage by the FPGA fabric. The nonvolatile nature of the flash allows for permanent storage of the design system's key parameters and variables. Fusion's embedded flash memory blocks can be instantiated in a design for general data storage via the following methods:

- The embedded flash memory macro can be instantiated directly into the RTL design in text or through SmartDesign.
- The Flash Memory System Builder's Data Storage Client in Libero SoC can be used to configure the embedded flash memory to be used for general-purpose data storage.
- SmartDesign IDP can be used to interface the embedded flash memory with the Common Flash Interface via the CoreCFI IP. The Flash Memory System Builder in Libero SoC is then used to configure the embedded flash memory to be paired with CoreCFI.

The following sections discuss the basic flash memory operations available to Fusion's embedded flash memory, and its three general data storage usages. Details related to the silicon implementation of the embedded flash memory block, including timing characteristics, can be found in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet.

Flash Memory Macro and Interface

Fusion contains up to four embedded flash memory blocks, each 2 Mbits in density. In an RTL design, a single embedded flash memory block corresponds to a single instance of the embedded flash memory block (NVM) macro. The embedded flash memory block is referenced as "FB" in this section and in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet. However, the macro used to instantiate an embedded flash memory block in a design is named "NVM" and will be referenced as such throughout this section.

The NVM macro can be instantiated directly into an RTL design without the intervention of Libero SoC Flash Memory System Builder. With the exception of a RESET operation, all operations on an NVM instance are synchronous to the rising edge of the clock. Table 5-5 contains a complete list of the input and output ports for the NVM macro.

***Table 5-5* • Flash Block Macro Port Descriptions**

Flash Port	Direction	Description
RESET	Input	Asynchronous active-low reset input signal. Holds the Flash Block's control logic in an initial state until released.
CLK	Input	Flash memory input clock whose maximum clock period is dictated by $t_{MPWCLKNVM}$. All memory operations and status are synchronous to the rising edge of this clock.
ADDR[17:0]	Input	The 18-bit byte-based Flash Block input address bus. ADDR must transition synchronously with the rising edge of CLK. The minimum addressable data size is 8 bits. For a data width of 16 bits, ADDR[0] is ignored and ADDR[1] becomes the lowest-order address; for a data width of 32 bits, ADDR[1:0] is ignored and ADDR[2] becomes the lowest-order address.
WEN	Input	Active-high write enable control input signal used for writing data into the flash memory Page Buffer. WEN must transition synchronously with the rising edge of CLK.
PROGRAM	Input	Active-high program operation control input signal used for writing the contents of the Page Buffer into the flash memory array page addressed. PROGRAM must transition synchronously with the rising edge of CLK.

***Table 5-5* • Flash Block Macro Port Descriptions (continued)**

Flash Port	Direction	Description
REN	Input	Active-high read enable control input signal used for read data from the flash memory array, Page Buffer, block buffer, or status registers. REN must transition synchronously with the rising edge of CLK.
READNEXT	Input	Active-high read-next operation control input signal. This feature loads the next block relative to that stored in the block buffer from the flash memory array while reads from the block buffer are being performed. READNEXT must be asserted along with REN to initiate the read-next operation, and must transition synchronously with the rising edge of CLK.
RD[31:0]	Output	The 32-bit output read data bus. The data on RD transitions synchronously with the rising edge of CLK. After REN has been asserted, issuing a data read operation, all data must be sampled from RD when BUSY is not asserted (when LOW). Data put out on RD are LSB-oriented. Upon a RESET, RD is initialized to zero.
WD[31:0]	Input	The 32-bit input write data bus. Data put in on WD must be LSB-oriented; any unused pins must be grounded or driven LOW. The data on WD must transition synchronously with the rising edge of CLK.
DATAWIDTH[1:0]	Input	The 2-bit RD and WD data bus width control input signal. DATAWIDTH must transition synchronously with the rising edge of CLK. • If DATAWIDTH is '00', the data busses contain one byte of data forming an 8-bit word (RD/WD[7:0]). • If DATAWIDTH is '01', the data busses contain two bytes of data forming a 16-bit word (RD/WD[15:0]). • If DATAWIDTH is '1x', the data busses contain four bytes of data forming a 32-bit word (RD/WD[31:0]).
PIPE	Input	Active-high pipeline stage control input signal. When asserted, a pipeline stage is added to the read data output. Read operations complete in five cycles instead of the typical four. The addition of the pipeline stage is recommended to be used for CLK frequencies greater than 50 MHz. PIPE must transition synchronously with the rising edge of CLK and be asserted along with REN.
PAGESTATUS	Input	Active-high page status control input signal. When asserted, a page status read operation is initiated. PAGESTATUS must transition synchronously with the rising edge of CLK and be asserted along with REN.
ERASEPAGE	Input	Active-high page erase control input signal asserted when the addressed page is to be programmed with all zeros. ERASEPAGE must transition synchronously with the rising edge of CLK.
DISCARDPAGE	Input	Active-high discard page control input signal asserted when the contents of the Page Buffer are to be discarded so that a new page write can be started. DISCARDPAGE must transition synchronously with the rising edge of CLK.

***Table 5-5* • Flash Block Macro Port Descriptions (continued)**

Flash Port	Direction	Description
AUXBLOCK	Input	Active-high auxiliary block select input signal asserted when the page address is used to access the auxiliary block within the page addressed. AUXBLOCK must transition synchronously with the rising edge of CLK.
SPAREPAGE	Input	Active-high spare page select input signal asserted when the sector addressed is used to access the spare page within that sector. SPAREPAGE must transition synchronously with the rising edge of CLK.
UNPROTECTPAGE	Input	Active-high unprotect page control input signal used to clear the protection of the addressed page. UNPROTECTPAGE must transition synchronously with the rising edge of CLK.
OVERWRITEPAGE	Input	Active-high overwrite page control input signal asserted when the page addressed is to be overwritten, if writable, with the contents of the Page Buffer. OVERWRITEPAGE must transition synchronously with the rising edge of CLK and must be asserted along with PROGRAM.
OVERWRITEPROTECT	Input	Active-high overwrite protect control input signal used to change the protection bit of the addressed page. OVERWRITEPROTECT must transition synchronously with the rising edge of CLK and must be asserted along with PROGRAM or ERASEPAGE.
PAGELOSSPROTECT	Input	Active-high page-loss protect control input signal used to prevent writes to any other page except the current addressed page in the Page Buffer, until the page is either discarded or programmed. PAGELOSSPROTECT must transition synchronously with the rising edge of CLK and be asserted along with PROGRAM or ERASEPAGE.
LOCKREQUEST	Input	Active-high lock request control input signal asserted when user access (including JTAG) to the flash memory array is to be prevented. LOCKREQUEST must transition synchronously with the rising edge of CLK.

***Table 5-5* • Flash Block Macro Port Descriptions (continued)**

Flash Port	Direction	Description
BUSY	Output	Active-high busy control output signal used to indicate when the flash memory is performing an operation. BUSY transitions synchronously with the rising edge of CLK. Upon a RESET, BUSY pulses HIGH for several cycles before settling to a LOW state.
STATUS[1:0]	Output	The 2-bit flash memory operation status output signals used to indicate the status of the last completed operation. STATUS transitions synchronously with the rising edge of CLK. Upon a RESET, STATUS is initialized to 0x00. • When STATUS is '00', it indicates that the last operation completed successfully. • When STATUS is '01' after a read operation, it indicates that a single error was detected during the last completed operation and was corrected. • When STATUS is '01' after a write operation, it indicates that the last completed operation addressed a write-protected page. • When STATUS is '01' after an erase-page/program operation, it indicates that the Page Buffer was unmodified during the last completed operation. • When STATUS is '10' after a read operation, it indicates that two or more errors were detected during the last completed operation. • When STATUS is '10' after an erase-page/program operation, it indicates that the compare operation failed during the last completed operation. • When STATUS is '11' after a write operation, it indicates that the attempt to write to another page before programming the current page was made during the last completed operation.

Below are the Verilog and VHDL representations of an NVM macro instantiation. For simulation purposes, the NVM macro may reference an Microsemi Memory File used to preload the memory with user-defined data. This is achieved by overriding the MEMORYFILE parameter in the simulation model during the NVM instantiation. The following is the NVM macro instantiation in Verilog and VHDL with the MEMORYFILE parameter override.

Verilog NVM Macro Instance

```
NVM U_NVM (
  .CLK (CLK),
  .RESET (RESET),
  .ADDR (ADDR),
  .REN (REN),
  .WEN (WEN),
  .READNEXT (READNEXT),
  .ERASEPAGE (ERASEPAGE),
  .PROGRAM (PROGRAM),
  .DATAWIDTH (DATAWIDTH),
  .RD (RD),
  .WD (WD),
  .BUSY (BUSY),
  .STATUS (STATUS),
  .SPAREPAGE (SPAREPAGE),
  .AUXBLOCK (AUXBLOCK),
  .UNPROTECTPAGE (UNPROTECTPAGE),
  .DISCARDPAGE (DISCARDPAGE),
  .OVERWRITEPROTECT (OVERWRITEPROTECT),
  .PAGELOSSPROTECT (PAGELOSSPROTECT),
  .PAGESTATUS (PAGESTATUS),
  .OVERWRITEPAGE (OVERWRITEPAGE),
  .PIPE (PIPE),
  .LOCKREQUEST (LOCKREQUEST)
);
```

Verilog NVM Macro Instance with Memory File

```
NVM #( .MEMORYFILE("<FLASH_INIT_FILE_NAME>.mem") U_NVM (
  ...
);
```

VHDL NVM Macro Instance with Memory File

```
architecture DEF_ARCH of <FLASH_NAME> is

component NVM

  generic (MEMORYFILE:string := "");

  port(
    ...
  );
end component;

begin

NVM_INST : NVM
  generic map(MEMORYFILE => "<FLASH_INIT_FILE_NAME>.mem")

  port map(
    ...
  );

end DEF_ARCH;
```

The memory array declared in the simulation model stores data that is one block wide. It is 64k×140 bits. The addressing scheme for accessing this array consists of 16 bits, as shown in Figure 5-20.

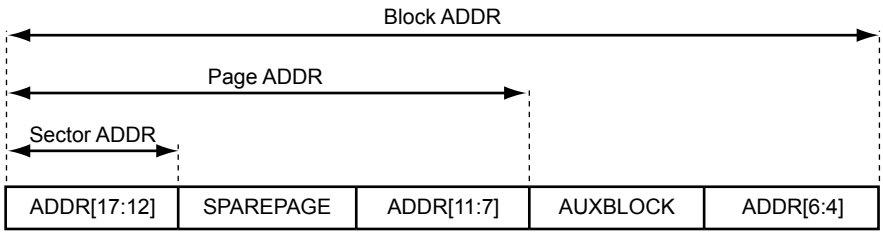


Figure 5-20 • **Addressing Scheme for Accessing the Flash Memory Array**

ADDR[17:0] is the embedded flash memory interface address; SPAREPAGE and AUXBLOCK are input signals. The memory file for preloading the Flash Array consists of strings of address and data in hexadecimal notation with address delimiters ('@'), and must conform to the rules given below:

1. Each line must contain a string of fixed length (35 characters) and start with an '@' if it corresponds to an address.
2. Each line following the address line corresponds to a block of data starting at the block address specified in the address line. This applies until the next line with an address specifier ('@') is encountered.
3. Each data block consists of 35 hex characters. Hex[31:0] are the data characters corresponding to 16 bytes of user data, where Hex[1:0] corresponds to Byte0 and Hex[31:30] corresponds to Byte15. Hex[34:32] are ECC-related bits and must be addressed manually.

Based on these rules, the format looks like the following:

```
@Block_Address_0
Block_Data_0 ( required )
Block_Data_1 ( optional )
Block_Data_2 ( optional )
...
...
Block_Data_8 ( Aux block data for this page, optional )
@Block_Address_n
Block_Data_n ( required )
Block_Data_n+1 ( optional )
Block_Data_n+2 ( optional )
...
...
...
Block_Data_n+8 ( Aux block data for this page, optional )
```

A typical memory file looks like the following:

```
@000...0000     // Beginning with @, start address in hex format. 0s to be padded
                // between @ and hex address to get a string of length 35.
ab101fd01...    // 35 hexadecimal characters corresponding to each block of flash memory
                // block cell
eab9c4......
@0004030....    // Start address for next data stream
c805489e....    // 35 hexadecimal characters corresponding to each block of flash memory
                // block cell
96986391
```

Data Storage Client Interface

The Flash Memory System Builder's Data Storage Client in Libero SoC can be used to configure the embedded flash memory to be used as general-purpose data storage. The Data Storage Client allows the user to create a flash memory partition, configure the address and data busses, and select an initialization file containing the flash array's initial values. JTAG read and/or write protections can also be added to prevent external access to the embedded flash memory array contents. Figure 5-21 shows the Data Storage Configuration window in Libero SoC.

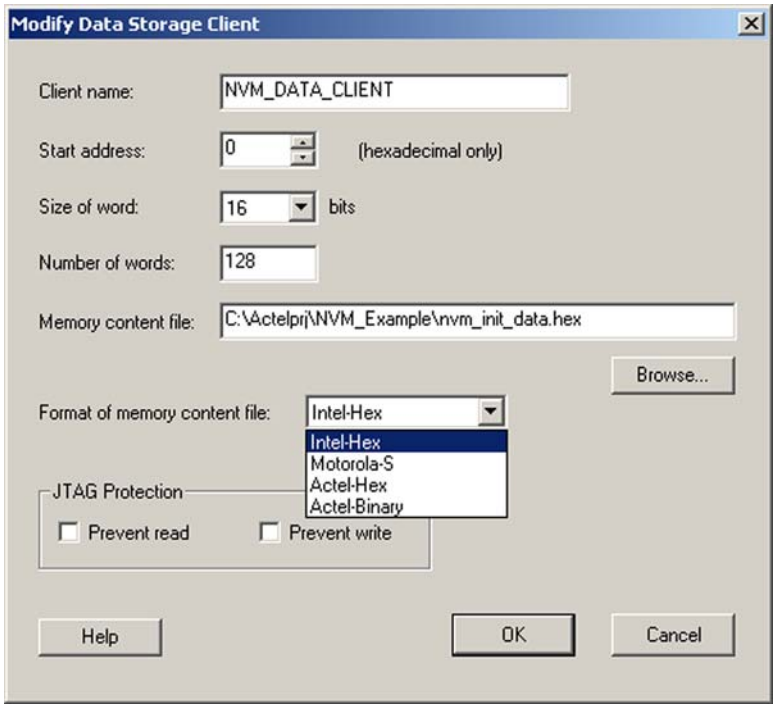


Figure 5-21 • **Data Storage Client Configuration Window**

The Data Storage Client spans a minimum of one page (128 bytes) and can go up to 2,048 pages, depending on the number of free pages available. The starting address for the Data Storage Client must be set to a value along the page boundaries—e.g., 0x00, 0x80, 0x100, etc. The word size for the read and write data busses can be either 8, 16, or 32 bits. The total number of words can be anywhere from one to 262,144 for 8-bit words, one to 131,072 for 16-bit words, and one to 65,536 for 32-bit words.

Once the Data Storage Client configuration is complete and the Flash Memory System Builder's IP is generated, the user can then instantiate the embedded flash memory system in the design. The overall ports list for the embedded flash memory system module may vary depending on the total number of clients added to the Flash Memory System. Refer to the "Using the Embedded Flash Memory for Initialization" section on page 137 and the "Common Flash Interface Data Client" section on page 169 for additional details regarding the other clients available in the Flash Memory System Builder. Table 5-6 on page 165 describes the Data Storage Client–specific ports.

***Table 5-6* • Data Storage Client–Specific Port Descriptions**

Flash Port	Direction	Description
USER_RESET	Input	Asynchronous active-low reset input signal. Holds the Flash Block's control logic in an initial state until released.
USER_CLK	Input	Flash memory input clock whose maximum clock period is dictated by $t_{MPWCLKNVM}$. All memory operations and status are synchronous to the rising edge of this clock.
USER_ADD[17:0]	Input	The 18-bit byte-based Flash Block input address bus. USER_ADD must transition synchronously with the rising edge of USER_CLK. For a data width of 16 bits, USER_ADD[0] is ignored and USER_ADD[1] becomes the lowest-order address; for a data width of 32 bits, USER_ADD[1:0] is ignored and USER_ADD[2] becomes the lowest-order address.
USER_WRITE	Input	Active-high write enable control input signal used for writing data into the flash memory Page Buffer. USER_WRITE must transition synchronously with the rising edge of USER_CLK.
USER_PROGRAM	Input	Active-high program operation control input signal used for writing the contents of the Page Buffer into the flash memory array page addressed. USER_PROGRAM must transition synchronously with the rising edge of USER_CLK.
USER_READ	Input	Active-high read enable control input signal used for read data from the flash memory array, Page Buffer, block buffer, or status registers. USER_READ must transition synchronously with the rising edge of USER_CLK.
USER_READ_NEXT	Input	Active-high read-next operation control input signal. This feature loads the next block relative to that stored in the block buffer from the flash memory array while reads from the block buffer are being performed. USER_READ_NEXT must be asserted along with USER_READ to initiate the read-next operation, and must transition synchronously with the rising edge of USER_CLK.
USER_DOUT[7:0], [15:0], or [31:0]	Output	The 8-, 16-, or 32-bit output read data bus. The data on USER_DOUT transitions synchronously with the rising edge of USER_CLK. After USER_READ has been asserted issuing a data read operation, all data must be sampled from USER_DOUT when USER_NVM_BUSY is not asserted (when LOW). Data put out on USER_DOUT are LSB-oriented. Upon a RESET, RD is initialized to zero. Upon a USER_RESET, RD is initialized to zero.
USER_DATA[7:0], [15:0], or [31:0]	Input	The 8-, 16-, or 32-bit input write data bus. Data put in on USER_DATA must be LSB-oriented; any unused pins must be grounded or driven LOW. The data on USER_DATA must transition synchronously with the rising edge of USER_CLK.

***Table 5-6* • Data Storage Client–Specific Port Descriptions (continued)**

Flash Port	Direction	Description
USER_WIDTH or USER_WIDTH[1:0]	Input	The 1- or 2-bit USER_DOUT and USER_DATA data bus width control input signal. DATAWIDTH must transition synchronously with the rising edge of USER_CLK. • If USER_WIDTH is '00', the data busses contain one byte of data forming an 8-bit word (USER_DOUT/USER_DATA[7:0]). • If USER_WIDTH is '01', the data busses contain two bytes of data forming a 16-bit word (USER_DOUT/USER_DATA[15:0]). • If USER_WIDTH is '1x', the data busses contain four bytes of data forming a 32-bit word (USER_DOUT/USER_DATA[31:0]). For a 1-bit USER_WIDTH: If USER_WIDTH is 1, the data busses are 16 bits wide; otherwise, they are 8 bits wide.
USER_PAGE_STATUS	Input	Active-high page status control input signal. When asserted, a page status read operation is initiated. USER_PAGE_STATUS must transition synchronously with the rising edge of USER_CLK and must be asserted along with USER_READ.
USER_ERASE_PAGE	Input	Active-high page erase control input signal asserted when the addressed page is to be programmed with all zeros. USER_ERASE_PAGE must transition synchronously with the rising edge of USER_CLK.
USER_DISCARD_PAGE	Input	Active-high discard page control input signal asserted when the contents of the Page Buffer are to be discarded so a new page write can be started. USER_DISCARD_PAGE must transition synchronously with the rising edge of USER_CLK.
USER_OVERWRITE_PAGE	Input	Active-high overwrite page control input signal asserted when the page addressed is to be overwritten, if writable, with the contents of the Page Buffer. USER_OVERWRITE_PAGE must transition synchronously with the rising edge of USER_CLK and must be asserted along with USER_PROGRAM.
USER_AUX_BLOCK	Input	Active-high auxiliary block select input signal asserted when the page address is used to access the auxiliary block within the page addressed. USER_AUX_BLOCK must transition synchronously with the rising edge of USER_CLK.
USER_SPARE_PAGE	Input	Active-high spare page select input signal asserted when the sector addressed is used to access the spare page within that sector. USER_SPARE_PAGE must transition synchronously with the rising edge of USER_CLK.
USER_UNPROT_PAGE	Input	Active-high unprotect page control input signal used to clear the protection of the addressed page. USER_UNPROT_PAGE must transition synchronously with the rising edge of USER_CLK.

***Table 5-6* • Data Storage Client–Specific Port Descriptions (continued)**

Flash Port	Direction	Description
USER_OVERWRITE_PROT	Input	Active-high overwrite protect control input signal used to change the protection bit of the addressed page. USER_OVERWRITE_PROT must transition synchronously with the rising edge of USER_CLK and must be asserted along with USER_PROGRAM or USER_ERASE_PAGE.
USER_PAGELOSS_PROT	Input	Active-high page-loss protect control input signal used to prevent writes to any other page than the current addressed page in the Page Buffer, until the page is either discarded or programmed. USER_PAGELOSS_PROT must transition synchronously with the rising edge of USER_CLK and must be asserted along with USER_PROGRAM or USER_ERASE_PAGE.
USER_LOCK	Input	Active-high lock request control input signal asserted when user access (including JTAG) to the flash memory array is to be prevented. USER_LOCK must transition synchronously with the rising edge of USER_CLK.

***Table 5-6* • Data Storage Client–Specific Port Descriptions (continued)**

Flash Port	Direction	Description
USER_NVM_BUSY	Output	Active-high busy control output signal used to indicate when the flash memory is performing an operation. USER_NVM_BUSY transitions synchronously with the rising edge of USER_CLK. Upon a USER_RESET, USER_NVM_BUSY pulses HIGH for several cycles before settling to a LOW state.
USER_NVM_STATUS[1:0]	Output	The 2-bit flash memory operation status output signals used to indicate the status of the last completed operation. USER_NVM_STATUS transitions synchronously with the rising edge of USER_CLK. Upon a USER_RESET, USER_NVM_STATUS is initialized to 0x00. • When USER_NVM_STATUS is '00', it indicates that the last operation completed successfully. • When USER_NVM_STATUS is '01' after a read operation, it indicates that a single error was detected during the last completed operation and was corrected. • When USER_NVM_STATUS is '01' after a write operation, it indicates that the last completed operation addressed a write-protected page. • When USER_NVM_STATUS is '01' after an erase-page/program operation, it indicates that the Page Buffer was unmodified during the last completed operation. • When USER_NVM_STATUS is '10' after a read operation, it indicates that two or more errors were detected during the last completed operation. • When USER_NVM_STATUS is '10' after an erase-page/program operation, it indicates that the compare operation failed during the last completed operation. • When USER_NVM_STATUS is '11' after a write operation, it indicates that an attempt to write to another page before programming the current page was made during the last completed operation.

Common Flash Interface Data Client

The CoreCFI IP is available to designers through Microsemi's SmartDesign IDP. CoreCFI provides an industry-standard interface to the embedded flash memory blocks within the Fusion family of FPGAs. The CoreCFI IP module is targeted to provide a functional subset of the Common Flash Interface standards with a design emphasis given to minimizing design size.

Using SmartDesign IDP to generate the CoreCFI IP HDL code, users must perform the following steps:

1. Open SmartDesign IDP through Libero SoC and add CoreCFI to the SmartDesign project.
2. Configure the CoreCFI IP. Figure 5-22 shows the CoreCFI IP Configuration GUI window in SmartDesign. The **FPGA Family** selected must be Fusion. The **Size (address width)** can be 6 to 18 bits. The number of address bits selected indicates the amount of the embedded flash memory accessible through CoreCFI—e.g., 10 bits = 1 kB, 12 bits = 4 kB, 18 bits = 256 kB.
3. Using the **Auto-Stitch to Top Level** feature of SmartDesign, bring all ports to the top on the block so that CoreCFI can be interfaced to the embedded flash memory module and other controlling circuits.

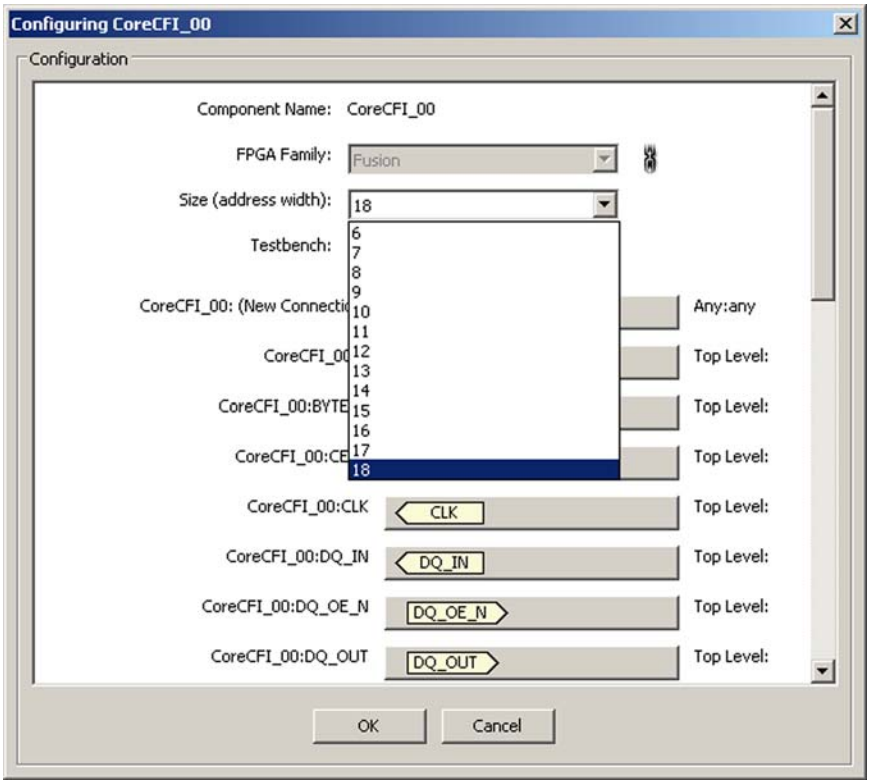


Figure 5-22 • **CoreCFI IP Configuration Window**

Once the CoreCFI IP is configured, use the **Save and Generate** function to deploy the HDL code and other related files, including the query data memory file. The query data memory file contains the Microsemi-specific 16-bit Command Set and Control Interface ID code as well as the embedded flash memory parameters and interface configuration data. Refer to the *SmartDesign User's Guide* and *CoreCFI Handbook* for additional configuration details.

The Flash Memory System Builder's CFI Data Client in Libero SoC is used to store the query data for the CoreCFI IP into the reserved (spare) pages of the embedded flash memory. This client does not take up any of the 2,048 pages available to designers for initialization or general data storage use.

Figure 5-23 on page 170 shows the CFI Data Client Configuration window in Libero SoC. Using the dialog box, the location of the query data memory file generated by SmartDesign during CoreCFI IP

deployment must be specified. The memory file format is Microsemi Binary and should not be modified by the user. SmartDesign deploys the query data memory file to the following directory location:

C:\Actelprj\<Libero_Project>\smartdesign\common\CORECFI\rtl\vlog\test\user\corecfi_query.mem

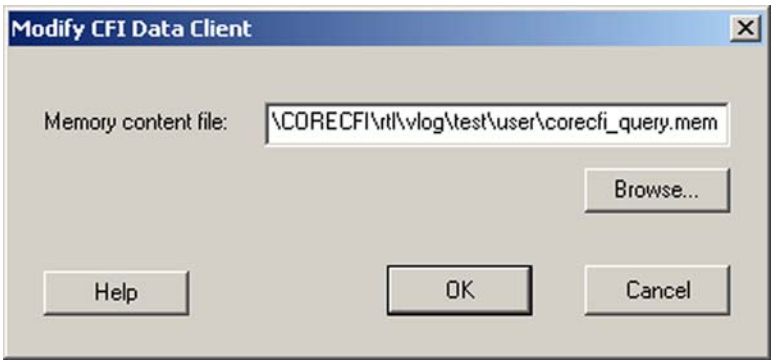


Figure 5-23 • **CFI Data Client Configuration Window**

During CoreCFI IP deployment, SmartDesign generates a sample HDL top file, named *corecfi_chip*, that instantiates both CoreCFI and the embedded flash memory system. Figure 5-24 illustrates an example CoreCFI IP system, complete with all interconnects between the main system blocks. The interface between CoreCFI and the embedded flash memory is through the Data Storage Client interface ports, as listed in Table 5-6 on page 165. Table 5-7 on page 171 describes the CoreCFI-specific ports.

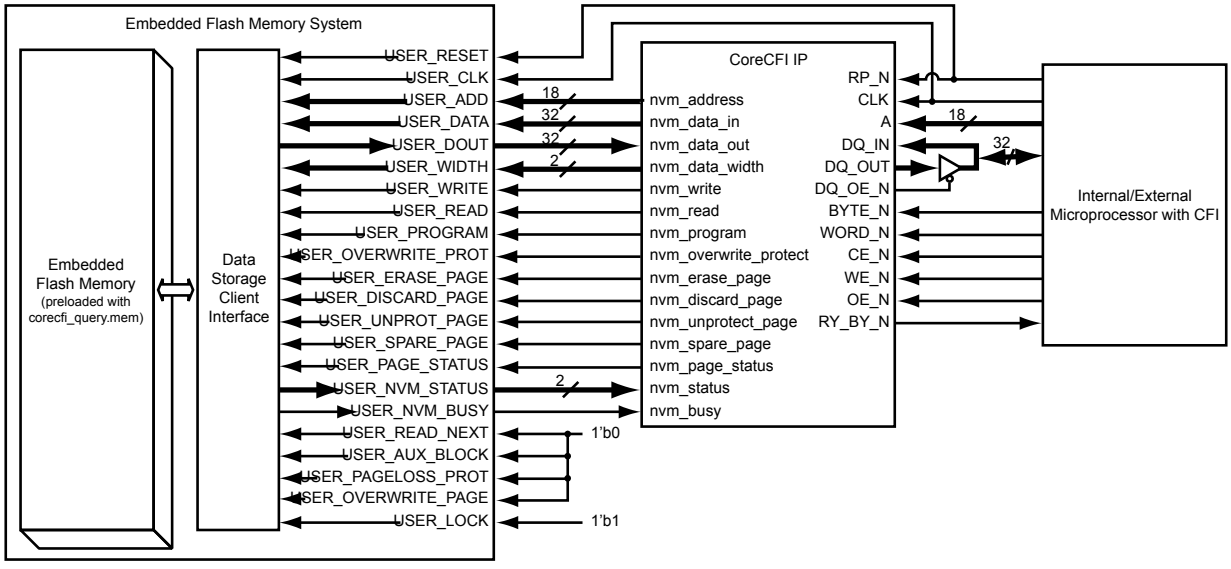


Figure 5-24 • **CoreCFI IP System Diagram**

Table 5-7 • **CoreCFI-Specific Port Descriptions**

CoreCFI Port	Direction	Description
RP_N	Input	Asynchronous active low reset input signal that holds CoreCFI in an initial state until released. It is recommended that RP_N not be asserted while RY_BY_N is asserted; otherwise, the embedded flash memory may be damaged. When RP_N is asserted, CoreCFI is placed in Read Array mode, the status register is set to 0x80 (ready), and the data bus ports are tristated.
CLK	Input	Flash memory input clock whose maximum clock period is dictated by $t_{MPWCLKNVM}$. All operations and status are synchronous to the rising edge of this clock.
CE_N	Input	Active low chip enable control input signal. When asserted, CoreCFI is enabled. CE_N must transition synchronously with the rising edge of CLK and must be asserted along with WE_N.
WE_N	Input	Active low write enable control input signal used to initiate a write operation. WE_N must transition synchronously with the rising edge of CLK and must be asserted along with CE_N. If CE_N and WE_N are not both asserted, the write command will be ignored. A write command takes one clock cycle to execute; both CE_N and WE_N must be deasserted upon completion.
OE_N	Input	Active low output enable control input signal used to control the direction of the CFI bidirectional data bus, and asserted during a read command. However, the bidirectional CFI data bus as listed in the CFI specification is split into an input data bus (DQ_IN) and an output data bus (DQ_OUT) in CoreCFI. The bidirectional CFI data bus must be formed outside of CoreCFI, where the direction control signal is an output of CoreCFI (DQ_OE_N), as shown in Figure 5-24 on page 170, and is a function of OE_N. OE_N must transition synchronously with the rising edge of CLK and must be asserted along with CE_N. If CE_N and OE_N are not both asserted, the read command will be ignored.
A[17:0]	Input	Active high 18-bit input address bus used to address a location in the Flash Array during a command execution. For a data width of 16 bits, A[0] is ignored and A[1] becomes the lowest-order address; for a data width of 32 bits, A[1:0] is ignored and A[2] becomes the lowest-order address.
WORD_N	Input	The 2-bit {WORD_N, BYTE_N} data bus width control input signal. Both WORD_N and BYTE_N must transition synchronously with the rising edge of CLK and must be asserted together. • If 'x0', the data bus contains one byte of data forming an 8-bit word (DQ_IN/OUT [7:0]). • If '01', the data bus contains two bytes of data forming a 16-bit word (DQ_IN/OUT [15:0]). • If '11', the data bus contains four bytes of data forming a 32-bit word (DQ_IN/OUT [31:0]).
BYTE_N	Input	
DQ_OE_N	Output	Active-low output enable control output signal used to control the direction of the CFI bidirectional data bus formed external to the CoreCFI IP, as shown in Figure 5-24 on page 170. DQ_OE_N transitions synchronously with the rising edge of CLK and is asserted when both CE_N and OE_N are asserted. Upon an RP_N assertion, DQ_OE_N is initialized to zero.

***Table 5-7* • CoreCFI-Specific Port Descriptions (continued)**

CoreCFI Port	Direction	Description
DQ_IN[31:0]	Input	The 32-bit input data bus used during a write command. The data on DQ_IN must transition synchronously with the rising edge of CLK. Data put in on DQ_IN must be LSB-oriented. Configuration depends on the state of {WORD_N, BYTE_N}: if in the 8-bit word mode, the DQ_IN[31:8] ports are ignored; if in the 16-bit word mode, the DQ_IN[31:16] ports are ignored. As shown in Figure 5-24 on page 170, DQ_IN should be connected to the CFI bidirectional bus.
DQ_OUT[31:0]	Output	The 32-bit output data bus used during a read command. The data on DQ_OUT transitions synchronously with the rising edge of CLK. Data put out on DQ_OUT is LSB-oriented. Configuration depends on the state of {WORD_N, BYTE_N}: if in the 8-bit word mode, the DQ_OUT[31:8] ports are ignored; if in the 16-bit word mode, the DQ_OUT[31:16] ports are ignored. As shown in Figure 5-24 on page 170, DQ_OUT should be connected to the CFI bidirectional bus. Upon an RP_N assertion, DQ_OUT is initialized to zero.

CoreCFI supports the Read Query, Read, Automatic Erase, Automatic Write, Lock, and Status CFI operations. The command descriptions are summarized in Table 5-8 and Table 5-9 on page 173. Refer to the *CoreCFI Handbook* (available on the Microsemi SoC Products Group website or in SmartDesign) for the CFI command details.

***Table 5-8* • Supported CFI Command Descriptions**

Command	Description
Read Query	The Read Query command causes CoreCFI to load the query database from a spare page of the embedded flash memory. Query data is always supplied on the least significant 8 bits of DQ_OUT. The address of the query data starts at 10h in 32-bit, 20h in 16-bit, or 40h in 8-bit mode.
Read ID Codes	The Read ID Codes command causes CoreCFI to load either the manufacturer code, die size code, or page lock status from the embedded flash memory onto DQ_OUT. The identifier codes returned are either values stored in the query data spare page or the lock status of a page in the Flash Array.
Read Array	The Read Array command causes CoreCFI to be placed in read array mode, where the content of the addressed location of the Flash Array is loaded onto DQ_OUT. Upon a reset, CoreCFI is initialized to the read array mode state.
Read Status	The Read Status command causes CoreCFI to load the status register onto DQ_OUT. The status register provides the status of the last write, erase, or lock command execution.
Clear Status	The Clear Status command causes CoreCFI to clear registered status register bits.
Erase Page	The Erase Page command causes CoreCFI to erase the addressed page of the Flash Array. A page erase activity fills the contents of a page with zeros.
Single Write	The Single Write command causes CoreCFI to write the data placed on DQ_IN to the addressed location of the Flash Array. This command reads the entire addressed page, modifies the address location, and programs the page into the Flash Array. If more than one location is to be modified, the Multiple Write command should be used.
Multiple Write	The Multiple Write command causes CoreCFI to be placed in the page write mode, where the contents of DQ_IN are written to the Page Buffer of the embedded flash memory. If the write activity exceeds the page boundary, the data written will wrap to the top of the page. Once all values are written into the page, the page is written into the Flash Array.
Page Lock	The Page Lock command causes CoreCFI to lock the addressed page, preventing any erase or write commands to the page from executing.
Page Unlock	The Page Unlock command causes CoreCFI to unlock the addressed page, allowing all erase or write commands to the page to execute.

***Table 5-9* • CFI Command Algorithm Summary**

Command	No. of Bus Cycles	First Bus Cycle			Second Bus Cycle		
		Operation	Address	Data	Operation	Address	Data
Read Query	2	Write	X	0x98	Read	Query Address	Query Data
Read ID Codes	2	Write	X	0x90	Read	Identifier Address	Identifier Data
Read Array	1 or 2	Write	X	0xFF	Read	Array Address	Array Data
Read Status	2	Write	X	0x70	Read	X	Status Data
Clear Status	1	Write	X	0x50	–	–	–
Erase Page	2	Write	Page Address	0x20	Write	Page Address	D0h
Single-Write	2	Write	Page Address	0x40	Write	Array Address	Array Data
Multi-Write	2	Write	Page Address	0xE8	Write	Page Address	N = Num. of elements – 1
Page Lock	2	Write	X	0x60	Write	Page Address	0x01
Page Unlock	2	Write	X	0x60	Write	Page Address	0xD0

Flash Operation Priority

The embedded flash memory has a built-in priority for operations when multiple actions are requested simultaneously. Table 5-10 shows the operation priority order—priority 0 is the highest. Access to the embedded flash memory is controlled by the BUSY (USER_BUSY in the Data Storage Client interface) signal. The BUSY output is synchronous to the CLK (USER_CLK in the Data Storage Client interface) signal. The embedded flash memory operations are only accepted in cycles where BUSY is not asserted (LOW).

***Table 5-10* • Flash Memory Operation Priority**

Operation	Priority
System Initialization	0 (highest)
Flash Memory Reset	1
Read	2
Write	3
Erase Page	4
Program	5
Unprotect Page	6
Discard Page	7

The system initialization operation is the highest in the priority order. The system initialization occurs upon a system reset of a Fusion FPGA. All FPGA operations should be halted during the system initialization process. Refer to the "Using the Embedded Flash Memory for Initialization" section on page 137 for additional information.

If read and write operations are performed simultaneously, for example, the read operation takes precedence over the write operation. The write data supplied during this operation is ignored, and the data remains unchanged. Also, if an erase page and a write operation are performed simultaneously, the write operation takes precedence over the erase page operation. The write data supplied during this operation executes. All other priority order situations behave similarly.

Flash Busy Signal Handling

The BUSY (USER_BUSY in the Data Storage Client interface) signal is one of the most important signals in the embedded flash memory; all operations on the embedded flash memory should be designed based on the BUSY signal. The BUSY signal is asserted HIGH whenever a flash operation is in progress, then deasserted after the operation is complete. To shorten simulation run time, the run time of the different operations (Write/Program/Erase/Read) is shortened in the simulation model. Therefore, the system design should be based on the BUSY signal assertion and deassertion status instead of counting the operation cycles for each operation. All flash memory inputs are ignored while BUSY is asserted. Inputs can be updated for the next operation at the rising edge of the clock with no hold time requirement.

During an embedded flash memory reset, the contents of the flash memory control logic block, such as the contents of the Block Buffer and Page Buffer, are cleared. Once RESET (USER_RESET in the Data Storage Client interface) is asserted LOW, the BUSY signal is asserted HIGH. After RESET is deasserted, the BUSY signal is deasserted approximately 25 µs later. Therefore, the system design should accommodate this BUSY period by monitoring the BUSY signal status. All operations can be executed only after the BUSY signal is deasserted.

For continuous operations, like the continuous reads for microprocessor instruction executions, the flash clock may be adjusted to compensate for the flash memory BUSY period—for example, during the loading of a new page from the Flash Array into the page buffer. Typically, the flash clock is sourced by the microprocessor system clock in an application. SmartTime, Microsemi's gate-level static timing analysis tool available in Microsemi's Designer software, will provide the maximum frequency for the system design. This maximum frequency will be adjusted to compensate for these BUSY periods.

Write Operations and Page Programming

The embedded flash memory offers a write operations class of commands. These commands include the page buffer write, discard page, program page, erase page, and overwrite page operations. All write commands are page-based operations. The embedded flash memory write operations modify the contents of both the Block and Page Buffers. As shown in Figure 5-25, the Block and Page Buffers are sub-blocks of the embedded flash memory and consist of volatile registers.

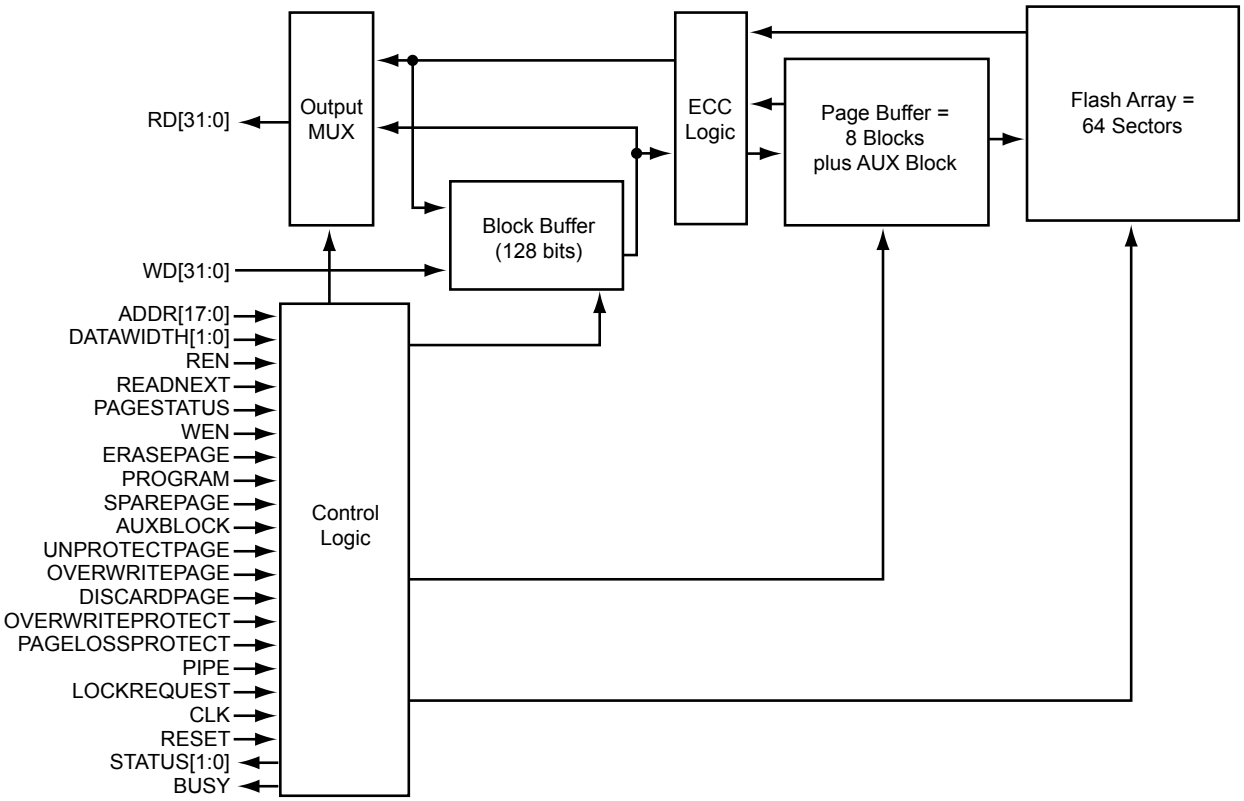


Figure 5-25 • **Flash Memory Block Diagram**

A write operation to a location in a page that is not already in the Page Buffer will cause the page to be read from the Flash Array and stored in the Page Buffer. The number of BUSY cycles required to complete the page buffer load is variable. The block that was addressed during the write operation will be loaded into the Block Buffer, and the data written by WD (USER_DATA in the Data Storage Client interface) will overwrite the data in the Block Buffer. After the data is written to the Block Buffer, the Block Buffer is then written to the Page Buffer to keep both buffers in sync. Subsequent writes to the same block will overwrite both the Block and Page Buffers without incurring BUSY cycles. A write operation to another block in the page will cause the addressed block to be loaded from the Page Buffer and into the Block Buffer, and the write will be performed as previously described. The Block Buffer load will incur four BUSY cycles (five cycles with PIPE asserted). The contents of the Page Buffer will be stored into the Flash Array only when a program page operation is executed. During the program page operation execution, the BUSY (USER_BUSY in the Data Storage Client interface) signal will be asserted HIGH for ~8 ms until the page programming completes. Figure 5-26 on page 176 is the timing diagram for a program page operation.

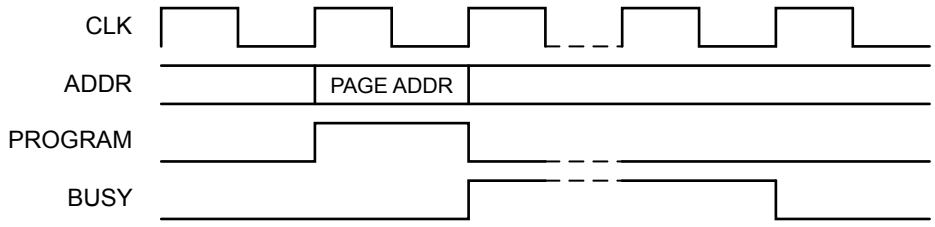


Figure 5-26 • **Program Page Operation Timing Diagram**

A page write operation is initiated by asserting the WEN (USER_WRITE in the Data Storage Client interface) signal HIGH. The page write operation automatically triggers a Block or Page Buffer load operation when a change in the block or page address is detected. During a Block Buffer load operation, the embedded flash memory logic loads the contents of the addressed block from the Page Buffer into the Block Buffer volatile registers. During a Page Buffer load operation, the embedded flash memory logic loads the contents of the addressed page from the Flash Array into the Page Buffer volatile registers. The BUSY signal is asserted HIGH during both loading processes. Figure 5-27 is the timing diagram for a page write operation. Any page being written using a page write or program page operation that is overwrite-protected will result in the STATUS signals being set to '01'; the page data stored in the Page Buffer or Flash Array are left unchanged. During a page write operation, the protected page is detected during both the page and block buffer loading processes.

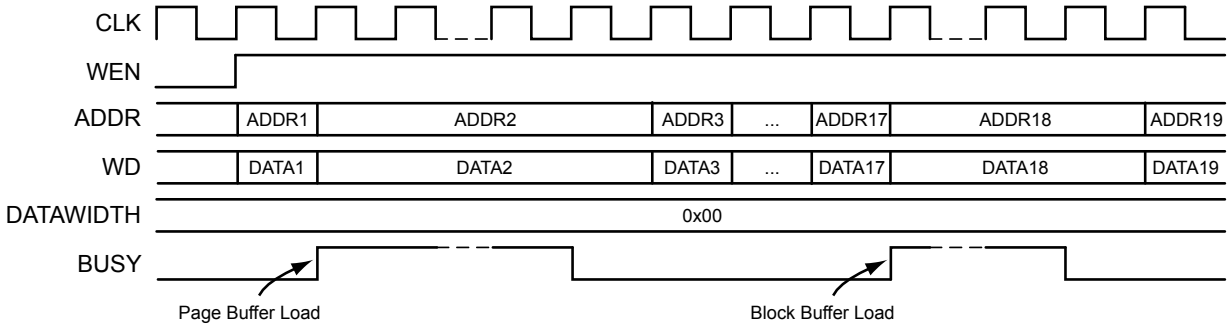


Figure 5-27 • **Page Write Operation Timing Diagram**

Writing to more than one page without executing a program page operation before changing the page address will result in the loss of the Page Buffer data. The page-loss protection option can be enabled to protect against the loss of the Page Buffer data by asserting the PAGELOSSPROTECT (USER_PAGELOSS_PROT in the Data Storage Client interface) signal HIGH during a program or erase page operation. Any page that is page-loss-protected will result in the STATUS (USER_NVM_STATUS in the Data Storage Client interface) signals being set to '11' when an attempt is made to write to a new page leaving the Page Buffer unchanged. The PAGELOSSPROTECT signal can be tied permanently HIGH; it is only sampled when the PROGRAM or ERASEPAGE signals are asserted HIGH. Microsemi recommends always enabling the page-loss protection option. Figure 5-28 on page 177 is the timing diagram showing the page-loss protection fault STATUS update.

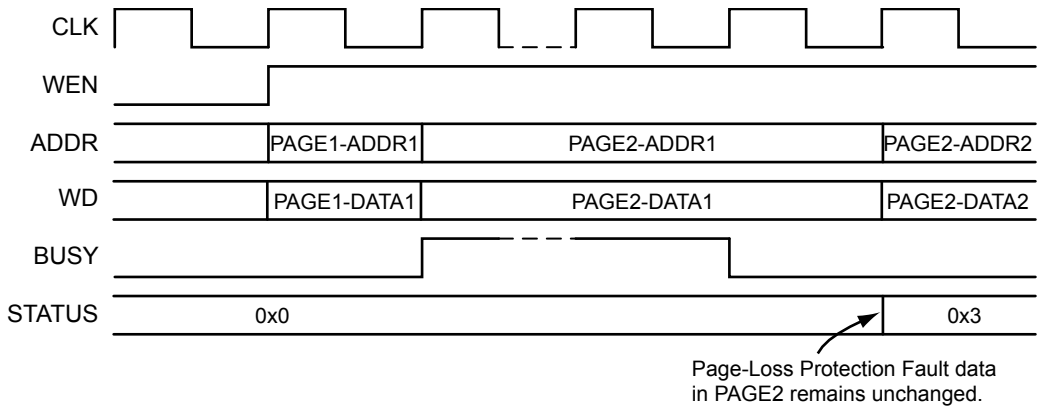


Figure 5-28 • **Page-Loss Protection Fault Timing Diagram**

To discard the contents of the modified Page Buffer, the discard page operation can be initiated by asserting the DISCARDPAGE (USER_DISCARD_PAGE in the Data Storage Client interface) signal HIGH for one clock cycle. This command will result in the Page Buffer being marked as unmodified. The BUSY signal will remain asserted until the discard page operation completes.

The erase page operation erases the addressed Flash Array page by filling the Page Buffer volatile registers with all zeros and issuing a program page operation. It is initiated by asserting the ERASEPAGE (USER_ERASE_PAGE in the Data Storage Client interface) signal HIGH while addressing the page to be erased. During the erase page operation execution, the BUSY signal will be asserted HIGH until the operation completes. Both the erase page and page write operations require fewer cycles when executing on the same page rather than a new page. Any page that is overwrite-protected will result in the STATUS signals being set to '01' when an attempt is made to erase the page, and the addressed page's data will be left unchanged.

The overwrite page operation can be used to overwrite any addressed page in the Flash Array with the contents of the Page Buffer. The operation can be initiated by asserting the OVERWRITEPAGE (USER_OVERWRITE_PAGE in the Data Storage Client interface) signal HIGH during a program page operation. Any page that is overwrite-protected will result in the STATUS signals being set to '01' when an attempt is made to program a page with OVERWRITEPAGE asserted, and the addressed page's data will be left unchanged. Figure 5-29 is the timing diagram showing the overwrite protection fault STATUS update.

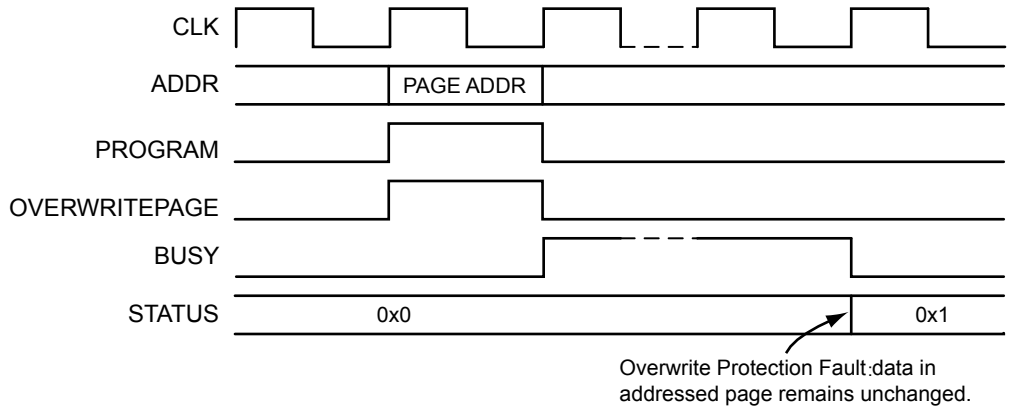


Figure 5-29 • **Overwrite Protection Fault Timing Diagram**

The overwrite protect mechanism is used to protect the contents of the selected Flash Array's pages from being overwritten. Asserting the OVERWRITEPROTECT (USER_OVERWRITE_PROT in the Data Storage Client interface) signal HIGH when a program page operation is undertaken will set the overwrite

protection option for the addressed page. OVERWRITEPROTECT can be held HIGH if multiple pages are to be overwritten; it is only sampled when the PROGRAM or ERASEPAGE signals are asserted HIGH. OVERWRITEPROTECT is ignored in all other operations. Any page that is overwrite-protected will result in the STATUS signals being set to '01' when an attempt is made to either write, program, or erase the protected page, as shown in Figure 5-30.

To clear the overwrite protect option for a given page, the unprotect page operation must be performed, and the page must be programmed with the OVERWRITEPROTECT pin cleared to save the new protection settings. An unprotect page operation is initiated by asserting the UNPROTECTPAGE (USER_UNPROT_PAGE in the Data Storage Client interface) signal HIGH while addressing the page. If the addressed page is not in the Page Buffer, the unprotect page operation will trigger a Page Buffer load operation. The load operation occurs only if the current page in the Page Buffer was programmed into the Flash Array or is not page-loss-protected. During the unprotect page operation execution, the BUSY signal will be asserted HIGH until the operation completes. If either the OVERWRITEPROTECT or UNPROTECTPAGE signal is asserted, the other must be deasserted. The unprotect page operation may result in the STATUS signals being set to '01' when the page has a single-bit correctable error, '10' for a double-bit uncorrectable error, or '11' when the Page Buffer has encountered a page-loss protection situation, during the operation execution. Figure 5-30 is the timing diagram showing the page-loss protection fault STATUS update during an unprotect page operation.

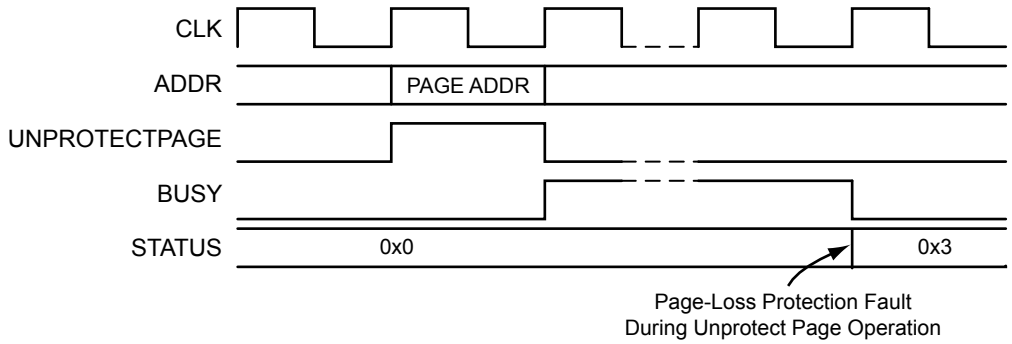


***Figure 5-30* • Unprotect Page Operation Page-Loss Protection Fault Timing Diagram**

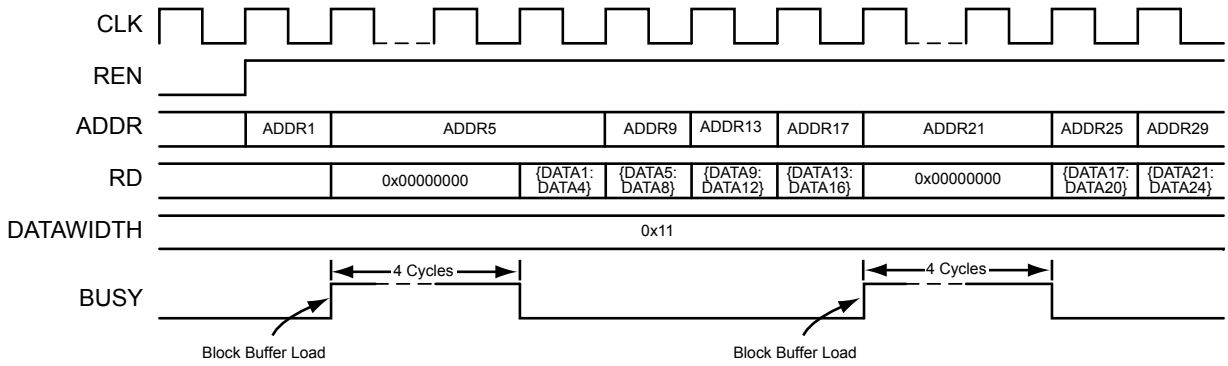
Read Operations

Read operations are designed to read data from the Flash Array and page status registers. The read operations support read operations with and without read-next or pipeline stage enabled. All read commands are page-based operations. The embedded flash memory read operation reads the contents of the Block Buffers, which are loaded from either the Page Buffer or the Flash Array. The Block and Page Buffers are sub-blocks of the embedded flash memory, consisting of volatile registers. Refer to Figure 5-25 on page 175 for the flash memory block diagram.

A read operation to a location in a page that is not already stored in the Page Buffer will cause the data from the Flash Array to be read and stored directly into the Block Buffer. The BUSY (USER_BUSY in the Data Storage Client interface) signal is asserted HIGH during the Block Buffer loading process for approximately four or five clock cycles. Any subsequent blocks addressed within the same page will be filled with data from the Flash Array with the same BUSY period consequence. However, a read operation to a location already stored in the Page Buffer is loaded into the Block Buffer without a BUSY period.

A read operation is initiated by asserting the REN (USER_READ in the Data Storage Client interface) signal HIGH. If the Block Buffer load is from the Flash Array, the BUSY signal is asserted HIGH for approximately four or five clock cycles during the Block Buffer loading process. The contents of the block that was addressed during the read operation will be placed on the RD (USER_DOUT in the Data Storage Client interface) output data bus. For frequencies greater that 50 MHz, a pipeline stage before the data read is placed on the RD data bus may be added by asserting the PIPE signal HIGH along with REN. If the pipeline stage is enabled, the BUSY signal is asserted for five clock cycles during each Block

Buffer load process; otherwise, it is asserted for four cycles. Figure 5-31 is the timing diagram for a page read operation.



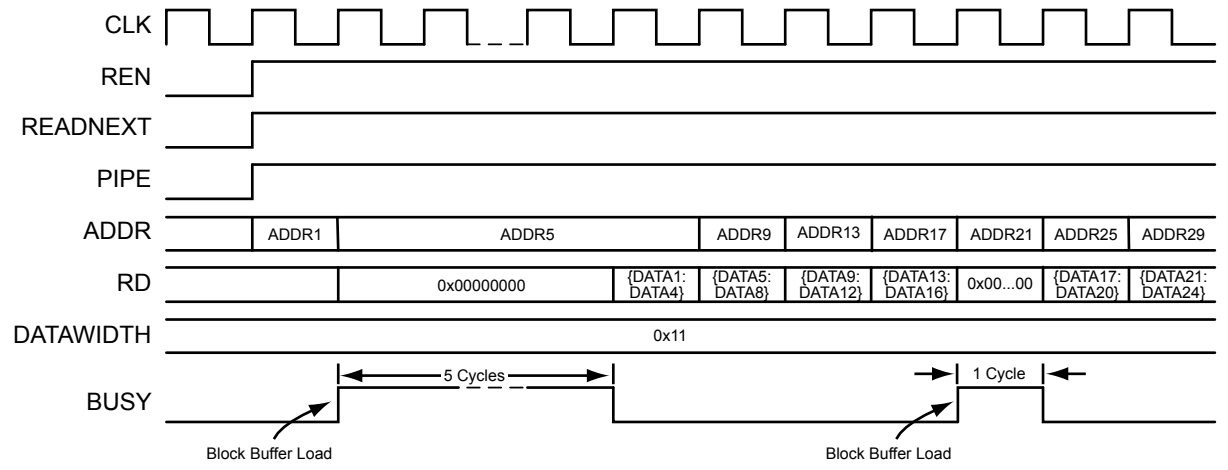
Note: With 16- and 32-bit RD data bus widths, bytes are placed on the bus in big-endian byte order.

***Figure 5-31* • Page Read Operation Timing Diagram**

The read-next operation is a feature by which the next block to the current block in the Block Buffer is read from the Flash Array while performing reads from the Block Buffer, to minimize BUSY wait states during a sequential read operation. It is enabled by asserting the READNEXT (USER_READ_NEXT in the Data Storage Client interface) signal HIGH along with REN. Since the read-next operation executes look-ahead reads, it is performed in a predetermined manner, as follows:

- When reading within a page, the next block fetched will be the next in linear address.
- When reading the last data block of a page, it will fetch the first block of the next page.
- When reading spare pages, it will read the first block of the next sector's spare page.
- When reading the last sector, it will wrap around to sector 0.
- When reading the auxiliary blocks, it will read the next linear page's auxiliary block.

When an address on the ADDR (USER_ADD in the Data Storage Client interface) bus does not agree with the predetermined look-ahead address, there is a time penalty for this access. The embedded flash memory must complete the current look-ahead read before starting the next. The worst-case BUSY period is a total of nine cycles before data is delivered. Figure 5-32 is the timing diagram for a pipeline-staged read-next page read operation.



Note: With 16- and 32-bit RD data bus widths, bytes are placed on the bus in big-endian byte order.

***Figure 5-32* • Pipeline-Staged Read-Next Page Read Operation Timing Diagram**

The status registers of each page of the embedded flash memory can be read by asserting the PAGESTATUS (USER_PAGE_STATUS in the Data Storage Client interface) signal along with REN. The contents of the addressed page's status register will be driven onto the RD data bus. The format of the data returned by a page status read is shown in Table 5-11. Figure 5-33 on page 180 is the timing diagram for a page status register read operation.

***Table 5-11* • Page Status Register Bit Definitions**

Bit(s)	Name	Description
[31:8]	Write Count	The number of times the page addressed has been programmed or erased. *Note: Max Write Count is set to 16,384 and the initial count is set to 2,288, which will flag when you write more than 14,096 cycles.*
[7:4]	Reserved	Reads as 0.
[3]	Over-Threshold	Over-threshold indicator. See the "Program Operation" section of the *Fusion Family of Mixed Signal Flash FPGAs* datasheet for details.
[2]	Read Protect	The read protect bit for the page set via the JTAG interface. If 1, the page is read-protected.
[1]	Write Protect	The write protect bit for the page set via the JTAG interface. If 1, the page is write-protected.
[0]	Overwrite Protect	The overwrite protect bit used to protect the page from being inadvertently overwritten. The bit must be set by asserting the OVERWRITEPROTECT signal during a program operation. The page cannot be overwritten without first performing an unprotect page operation. Refer to the "Write Operations and Page Programming" section on page 175 for additional information.

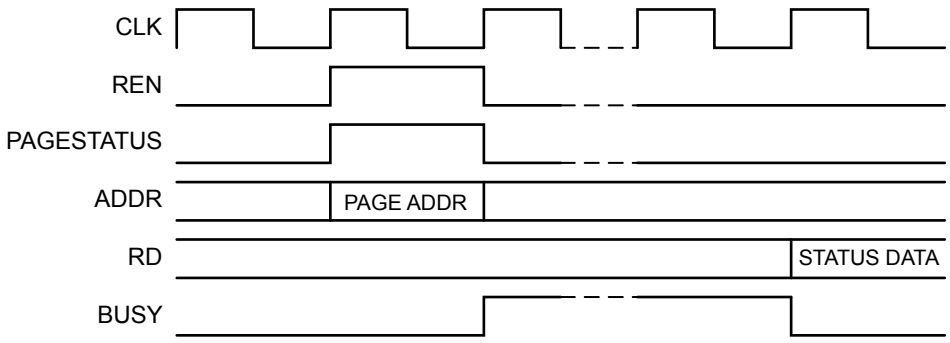


***Figure 5-33* • Page Status Register Read Operation**

Note on Updating the Contents of Flash

The possibility of data corruption due to a programming interruption is common to flash technology, and precautions must be taken when updating the data contents of any flash memory, including Fusion's embedded flash memory block. An interruption may occur, for example, as a result of a loss of power to the Fusion FPGA or an unexpected reset operation of the Flash Block control logic. Therefore, it is recommended that the appropriate measure in the application be taken to prevent such interruptions from occurring by adding power-down ramp control circuitry, low voltage detection circuitry, etc., to allow for the 8 ms program and erase page operations to complete their execution.

If an interruption of the write or read operation occurs, the immediate data stored in the Block or Page Buffer is lost, but the Flash Array's data (a sub-block of Flash Block holding the nonvolatile data contents, as shown in Figure 5-25 on page 175) remains valid. However, if an interruption of a program or erase operation occurs, the page addressed during the interruption may be left in a locked state. Although it may be that no physical damage to the Flash Array will have occurred, data corruption of the page is likely to have occurred, resulting in the possible corruption of the page's auxiliary block control data. Only

the page addressed at the time of the interruption may incur data corruption; no other page should be affected.

A Flash Block's page contains eight blocks of user data and one auxiliary block. The auxiliary block is mostly used for storing control data. The auxiliary block control bits of concern are the write protect, read protect, overwrite protect, and write count bits. If the protection bits are corrupted, in most cases, reprogramming the embedded flash memory contents with the programming (STAPL) file will restore the protection bits to their predefined states; however, the page's write count will be lost. If the overwrite protection bit is the only bit of concern, the application can set/clear this bit by using the OVERWRITEPROTECT or UNPROTECTPAGE input signals to the NVM macro, as described in the "Write Operations and Page Programming" section on page 175.

Microprocessor/Microcontroller Interface

The embedded flash memory can be interfaced to a microprocessor or microcontroller for use as its nonvolatile instruction execution memory space and data storage memory space. To enable the embedded flash memory to communicate with the microprocessor's or microcontroller's bus architecture, an interface bridge must be developed and added to the design. Microsemi offers, as a DirectCore IP through SmartDesign IDP, the CoreAhbNvm IP, which contains a hardware bridge between the embedded flash memory block and the industry-standard Advanced Microcontroller Bus Architecture's high-performance system backbone bus (AHB). Microprocessors can then access data stored in the flash memory via the software-driven CFI command protocol. The following sections describe CoreAhbNvm's design configuration and CFI commands.

CoreAhbNvm IP Configuration

SmartDesign IDP is used to develop a microcontroller design for Microsemi FPGAs, utilizing the ARM7,™ Cortex-M1, and CoreABC microprocessor or microcontroller IP offerings, together with the various AMBA-AHB and AMBA-APB (Advanced Peripheral Bus) IP peripherals. For Fusion FPGA designs, CoreAhbNvm can be paired with either the ARM7 (CoreMP7 IP) or Cortex-M1 microprocessors. The microprocessors must communicate with CoreAhbNvm through an AHB master, also available through SmartDesign as CoreAHB and CoreAHBLite.

When using CoreAhbNvm's embedded flash memory as the microprocessor's instruction execution memory, CoreAhbNvm must be connected as CoreAHB or CoreAHBLite's slave-0 (AHBmslave0) port. After system reset, CoreAhbNvm defaults to the CFI's read-array mode for the continuous reading of the instruction code stored in memory. Since data is read from the embedded flash memory's page buffer, when changing the instruction address to a new page in flash, the microprocessor must allow for the time required to load a new flash memory page into its page buffer before capturing the data. Therefore, the microprocessor and AHB-master system clock must be reduced appropriately, such that continuous reads can be performed without needing to pause for page buffer loading. When performing static timing analysis using SmartTime, the maximum operating frequency of the microprocessor's system clock is typically reduced to, for example, 15 MHz for the ARM7.

When using CoreAhbNvm's embedded flash memory as the microprocessor's nonvolatile data storage, CoreAhbNvm must be connected to any slave port other than slave-0 (AHBmslave1–AHBmslave15), reserving slave-0 for the instruction execution memory. Note that the Remap input to CoreAHB and CoreAHBLite is used to swap between the slave-0 (AHBmslave0) and slave-1 (AHBmslave1) ports. The Remap feature of the AHB architecture is typically used to swap boot memory spaces (from flash to RAM and vice versa). Therefore, be sure to plan the AHB system carefully taking all AHB slave configurations into consideration. For additional details, refer to the *CoreAHB*, *CoreAHBLite*, and *CoreRemap* datasheets.

Once the entire microcontroller design is complete, SmartDesign's "Save & Generate" operation will produce the RTL code and testbench for the design and save it in a project directory, which then is imported into Libero SoC. Libero SoC can then be used to complete the Fusion FPGA design flow. Refer to the *SmartDesign User's Guide* and the *Libero SoC User's Guide* for additional usage information. Figure 5-34 on page 182 is an example of a simple Fusion CoreMP7 microcontroller SmartDesign project, with CoreAhbNvm used as its instruction execution memory.

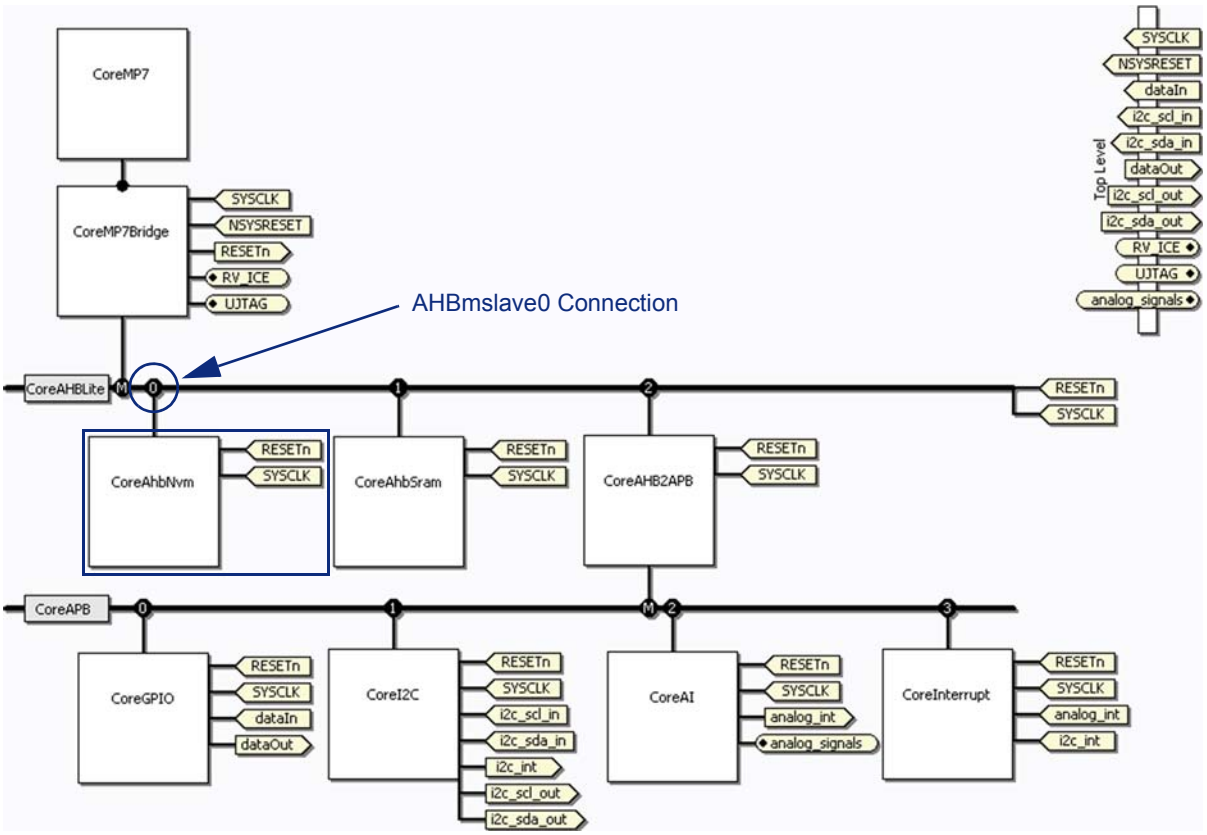


Figure 5-34 • **Simple Fusion SmartDesign Project with CoreAhbNvm Used as the Instruction Execution Memory**

SmartDesign allows designers to easily configure the IP though a simple GUI. Figure 5-35 shows the CoreAhbNvm IP SmartDesign configuration GUI window.

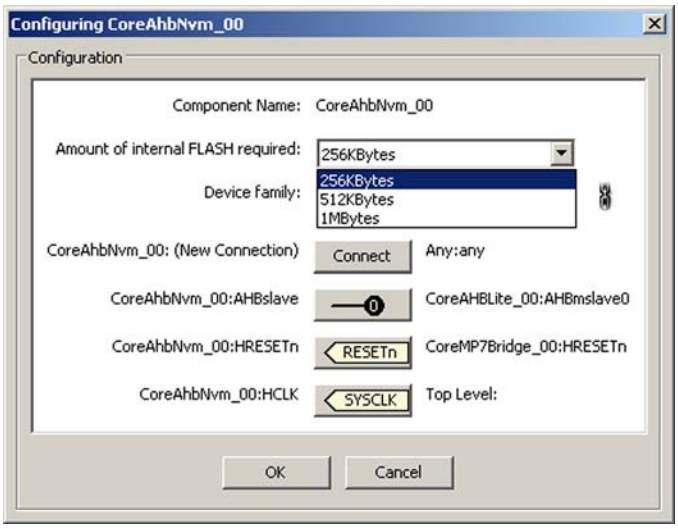


Figure 5-35 • **CoreAhbNvm SmartDesign Configuration GUI**

CoreAhbNvm can be configured to include the embedded flash as a 256-kbyte, 512-kbyte, or 1-Mbyte flash memory data space. By default, CoreAhbNvm is configured to include a 256-kbyte flash memory. Up to four CoreAhbNvm blocks can be instantiated for a single Fusion design, depending on the targeted Fusion FPGA device. Be sure to size the flash memory for each CoreAhbNvm block appropriately, such that the total number of Flash Block instances does not exceed the total number available on the targeted Fusion FPGA device. Table 5-12 lists the required number of Flash Block instances for each flash memory space size configuration option. Cross-reference the required number of Flash Block instances with the number of blocks available in the targeted Fusion device, as found in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet.

Table 5-12 • **Flash Memory Space Size vs. Required Number of Memory Block Instances**

Flash Memory Size Option	Number of Flash Block Instances
256 kbytes	1
512 kbytes	2
1 Mbytes	4

The CoreAhbNvm port signals are described in Table 5-13. If CoreAhbNvm is paired with either the on-chip ARM7 or Cortex-M1, SmartDesign's Auto Stitch feature can be used to make all the required interconnects between CoreAhbNvm and CoreAHB/CoreAHBLite. CoreAhbNvm, however, can also be used as an extension of an external microprocessor's nonvolatile memory space by placing all its AHB interface signals at the top of the chip design. CoreAhbNvm can then be connected and controlled by an external AHB master.

***Table 5-13* • CoreAhbNvm Port Signal Descriptions**

Signal	Direction	Description
HCLK	Input Bus Clock	Rising-edge-active AHB clock, which times all bus transfers and all signal timings
HRESETn	Input Reset	Active-low asynchronous bus reset signal used to reset the system and the bus. This is the only active-low AHB signal.
HTRANS[1:0]	Input	Transfer control input signals that indicate the type of the current transfer: 00 – Idle 01 – Busy 10 – Non-Sequential 11 – Sequential HTRANS transitions synchronously with the rising edge of HCLK.
HADDR[19:0]	Input	The 32-bit AHB system address bus from the AHB master. HADDR transitions synchronously with the rising edge of HCLK.
HWRITE	Input	Transfer direction control signal. When HIGH, this signal indicates a write transfer; when LOW, a read transfer. HWRITE transitions synchronously with the rising edge of HCLK.
HSIZE[2:0]	Input	Indicates the size of the transfer, which can be byte (8-bit), halfword (16-bit), or word (32-bit). HSIZE transitions synchronously with the rising edge of HCLK.
HWDATA[31:0]	Input	32-bit write input data bus from the AHB master. HWDATA transitions synchronously with the rising edge of HCLK.
HREADYIN	Input	Active-high ready signal input from all other AHB slaves. HREADYIN transitions synchronously with the rising edge of HCLK.
HSEL	Input	Active-high slave select signal input, which is a combinatorial decode of HADDR. Indicates that this slave is currently being selected. HSEL transitions synchronously with the rising edge of HCLK.
HRDATA[31:0]	Output	32-bit read output data bus written back to the AHB master. HRDATA transitions synchronously with the rising edge of HCLK. Upon an HRESETn reset, the HRDATA output is zero.

***Table 5-13* • CoreAhbNvm Port Signal Descriptions (continued)**

Signal	Direction	Description
HREADY	Output	Transfer-done control output signal. When HIGH, the HREADY signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer. HREADY transitions synchronously with the rising edge of HCLK. Upon an HRESETn reset, the HREADY output is HIGH.
HRESP[1:0]	Output	Transfer response output signals, which have the following meanings: 00 – Okay 01 – Error 10 – Retry 11 – Split HRESP transitions synchronously with the rising edge of HCLK. Upon an HRESETn reset, the HRESP output is in an Okay state (0x00).

Supported CFI Commands

The data stored within CoreAhbNvm's flash memory blocks for instruction execution or data storage can be accessed by the microprocessor or microcontroller via the software-driven Common Flash Interface. The CFI is a command-driven interface standard used to standardize the low-level flash software algorithms. The CoreAhbNvm IP supports a subset of the CFI commands. The supported commands are summarized in Table 5-14.

***Table 5-14* • Supported CFI Command Descriptions**

Command	No. of Bus Cycles	First Bus Cycle			Second Bus Cycle		
		Operation	Address	Data	Operation	Address	Data
Read Status	2	Write	X	0x70	Read	X	Status Data
Clear Status	1	Write	X	0x50	–	–	–
Read Array	1 or 2	Write	X	0xFF	Read	Array Address	Array Data
Erase Page	2	Write	Page Address	0x20	Write	Page Address	D0h
Single Write	2	Write	Page Address	0x40	Write	Array Address	Array Data
Multi-Write	2	Write	Page Address	0xE8	Write	Page Address	No. of elements – 1

Upon a system reset (LOW on HRESETn), CoreAhbNvm defaults to the read-array operation mode to support the microprocessor instruction execution usage. No write command is needed to place the CoreAhbNvm IP in the read-array mode as instructed in Table 5-14.

As part of the CFI, a status register is used to provide feedback to the software regarding command execution. During each command, the status register should be read to ensure that the flash memory is not busy executing a command (bit 7 set to 1) and that the command executed appropriately. All new commands are ignored while the flash memory is busy (bit 7 set to 0). The status register also includes the Program/Erase, Write, and Read/Protection Error flags (bits 5, 4, and 1, respectively). All three error flags are registered; therefore, once an error is detected, the status register must be cleared using the Clear Status command. Once a CFI command is issued and bit 7 (the Ready flag) is set to 1, the error flags should be checked to ensure that the completed operation did not incur an error. Table 5-15 on page 186 includes a bit description of the status register flags.

***Table 5-15* • Status Register Bit Descriptions**

Bit(s)	Flag	Description
7	Ready	Active-high Ready flag used to indicate when the flash memory is ready to receive new commands or is busy processing a command's operation. If Ready is set to 1, the flash memory is ready to receive new commands, and the previous command's operation is complete. If Ready is set to 0, the flash memory is busy processing the command's operation. No new commands should be issued until the Ready flag is at 1. Upon an HRESETn reset, the Ready flag defaults to 1.
6	–	–
5	Program or Erase Error	Active-high Program or Erase Error flag used to indicate whether an error occurred during a program or erase operation. If the Program or Erase Error flag is set to 1, the flash memory incurred an error during a program or erase operation. A locked page access will cause a program or erase operation to fail, triggering the error flag to transition to 1. If the flag transitions to 1, it will remain there until cleared by the Clear Status command operation. Upon an HRESETn reset, the Program or Erase Error flag defaults to 0.
4	Write Error	Active-high Write Error flag used to indicate whether an error occurred during a write operation. If the Write Error flag is set to 1, an error occurred during the page buffer write operation. If the Write Error flag is set to 0, the page buffer write operation was successful. If the flag transitions to 1, it will remain there until cleared by the Clear Status command operation. Upon an HRESETn reset, the Write Error flag defaults to 0.
3–2	–	–
1	Read or Protection Error	The Read or Protection Error flag is used to indicate whether an error occurred during a read, program, or erase operation. If a read operation completed and the Read Error flag is set to 1, an error occurred during the read operation. If a program/erase operation completed and the Protection Error flag is set to 1, the page program or erase operation failed because the page being accessed is protected. If the flag is set to 0, the read, program, or erase operation completed successfully. If the flag transitions to 1, it will remain there until cleared by the Clear Status command operation. Upon an HRESETn reset, the Read or Protection Error flag defaults to 0.
0	–	–

Read Status Command

The status register contains flags used to inform the user when the flash is ready for the next operation or when an error occurred with the last operation performed. Prior to issuing any new array write or read commands to the CFI control logic, the Ready flag (bit 7 of the status register) should be checked. If Ready is set to 1, the flash memory is ready to receive a new command. If Ready is set to 0, the flash memory is busy performing an operation and all new commands issued will be ignored. Once the Ready flag is set to 1, the Program/Erase, Write, and Read/Protection Error flags (bits 5, 4, and 1, respectively) should be checked to ensure that the completed operation did not incur an error. All three error flags are registered; therefore, once an error is detected, the status register must be cleared using the Clear Status command.

Read Status command execution is performed as follows:

1. Issue the Read Status command by writing the command value 0x70 to any location in the flash array.

   ```
   [Any Array Address] = 0x70
   ```

2. Once the command has been issued, read and store the contents of the status register to a temporary variable. To read the contents of the status register, issue a read operation at any address in the flash array. The value read will be the contents of the status register.

   ```
   Status_Register_Contents = [Any Array Address]
   ```

Clear Status Command

The Clear Status command clears the Program/Erase, Write, and Read/Protection Error flags (bits 5, 4, and 1, respectively) of the status register. Clear Status command execution is performed in a single step:

Issue the Clear Status command by writing the command value 0x50 to any location in the flash array.

```
[Any Array Address] = 0x50
```

Read Array Command

The Read Array command is used to place CoreAhbNvm in read-array mode. Once CoreAhbNvm is in read-array mode, it will remain in this mode until a new CFI command is issued. Upon an HRESETn reset, CoreAhbNvm defaults to read-array mode to support microprocessor instruction execution usage. When performing a continuous read and a page boundary has been crossed, the embedded flash memory must reload the page buffer with the new flash page being accessed. In this situation, either the Ready flag of the status register must be monitored or the system clock frequency must be adjusted to allow for the loading of the page buffer. If monitoring the Ready flag, once it is at 1, the Read Error flag should be checked to ensure that the completed operation did not incur an error. The Read Error flag is registered; therefore, once an error is detected, the status register must be cleared using the Clear Status command.

Read Array command execution is performed as follows:

1. Issue the Read Array command by writing the command value 0xFF to any location in the Flash Array.

   ```
   [Any Array Address] = 0xFF
   ```

2. Once the command has been issued, the contents of the array can be read by issuing a read operation at the selected array address in flash.

   ```
   Array_Contents = [Array Address]
   ```

If monitoring the Ready flag, perform the following:

3. The Ready flag (bit 7) of the status register must be monitored to determine when the read operation completes. Using the Read Status command, the status register should be polled until Ready flag is set to 1, signaling that the write operation has completed.

   ```
   Ready = Status_Register_Contents & 0x80
   ```

4. The Read Error flag (bit 1) of the status register should also be checked to determine if an error occurred during the read operation. Once the Ready flag is set to 1, the status of the flag can be read.

   ```
   Read_Error = Status_Register_Contents & 0x02
   ```

Erase Page Command

The Erase Page command is used to erase an entire page by writing 0x00 to all locations of the selected page in the embedded flash memory array. Once the Erase Page command has completed, the Protection and Erase Error flags (bits 1 and 5 of the status register) should be checked to determine whether an erase error occurred and whether the page was protected.

Erase Page command execution is performed as follows:

1. Issue the Erase Page command by writing the command value 0x20 to any location within the flash array page to be erased.

   ```
   [Page Address] = 0x20
   ```

2. Issue the Erase Confirm command by writing the command value 0xD0, addressing any location within the flash array page to be erased. Any other command issued will abort the Erase Page command.

   ```
   [Page Address] = 0xD0
   ```

3. The Ready flag (bit 7) of the status register must be monitored to determine when the erase operation completes. Using the Read Status command, the status register should be polled until the Ready flag is at 1, signaling that the write operation has completed.

   ```
   Ready = Status_Register_Contents & 0x80
   ```

4. The Protection and Erase Error flags (bits 1 and 5) of the status register should also be checked to determine whether an error occurred during the write operation. Once the Ready flag is at 1, the status of both flags can be read.

   ```
   Protection_Error = Status_Register_Contents & 0x02
   Erase_Error = Status_Register_Contents & 0x20
   ```

Single Write Command

The Single Write command is used to write a single byte, halfword, or word to the embedded flash memory. When performing a Single Write command operation, the entire page buffer will be written to the embedded flash memory. Therefore, users should avoid a Single Write command operation when more than one location in a page must be written; a Multi-Write command operation should be used wherever possible. Once the write command has completed, the Write and Program Error flags (bits 4 and 5 of the status register) should be check to determine whether an error occurred.

Single Write command execution is performed as follows:

1. Issue the Single Write command by writing the command value 0x40 to any location within the flash array page. It may be simplest to write the command value to the array address of the data to be written.

   ```
   [Page Address] = 0x40
   ```

2. Once the command has been issued, a second write operation must be issued. The write operation consists of writing the array data contents to the array address.

   ```
   [Array Address] = New_Array_Data
   ```

3. The Ready flag (bit 7) of the status register must be monitored to determine when the write operation completes. Using the Read Status command, the status register should be polled until the Ready flag is at 1, signaling that the write operation has completed.

   ```
   Ready = Status_Register_Contents & 0x80
   ```

4. The Write and Program Error flags (bits 4 and 5) of the status register should also be checked to determine whether an error occurred during the write operation. Once the Ready flag is at 1, the status of both flags can be captured.

   ```
   Write_Error = Status_Register_Contents & 0x10
   Program_Error = Status_Register_Contents & 0x20
   ```

Multi-Write Command

The Multi-Write command is used to write multiple consecutive bytes, half-words, or words to a single page of the embedded flash memory. Initially, the flash memory page is read from the array and loaded into the flash memory's page buffer. *N*, the number of data elements (bytes, words, or double words) to be written to the array, minus one, is then written to CoreAhbNvm and serves as the maximum number of data elements used by the internal counter. The expected N count ranges are N = 00h to N = 7Fh (e.g., 1 to 128 bytes) in 8-bit mode, N = 00h to N = 3Fh in 16-bit mode, and N = 00h to N = 1Fh in 32-bit mode. All data is written into the page buffer sequentially, starting from the first address of the page data is written to. If N exceeds the starting address plus N addresses of the selected page, the data writes will wrap around onto the top of the current page stored in the page buffer. Once all data values are written into the page buffer, the Program Buffer Confirm command (0xD0) is expected to be issued at the next write cycle after the last data element is written; any other command issued at this point in the sequence will prevent the programming of the page buffer into the array (the Multi-Write command will be aborted). Once the program operation has completed, the Write and Program Error flags (bits 4 and 5 of the status register) should be checked to determine whether an error occurred.

Multi-Write command execution is performed as follows:

1. Issue the Multi-Write command by writing the command value 0xE8, addressing the flash page to be written. It may be simplest to write the command value to the starting address of the sequence of data values to be written.

   ```
   [Page Address] = 0xE8
   ```

2. The Ready flag (bit 7) of the status register must be monitored to determine when the page buffer fill operation completes. Using the Read Status command, the status register should be polled until the Ready flag is at 1, signaling that the page buffer fill operation has completed.

   ```
   Ready = Status_Register_Contents & 0x80
   ```

3. Issue a write operation, writing the number of elements to be written to the flash array, minus one (N), to the page address. Again, it may be simplest to write N to the starting address.

   ```
   [Page Address] = N       // (N = number of elements - 1)
   ```

4. Once the command and number of elements has been issued, a sequence of write operations must be issued. The write operation consists of writing the array data contents to the array address. All data is written sequentially, beginning from the starting address, set by the address of the first data value written. Repeat this step until all N elements have been written into the page buffer.

   ```
   [Array Address] = New_Array_Data
   ```

5. Issue the Buffer Program Confirm command by writing the command value 0xD0, addressing the flash page to be written. Any other command issued will abort the programming of the page buffer.

   ```
   [Page Address] = 0xD0
   ```

6. The Ready flag (bit 7) of the status register must be monitored to determine when the page buffer write operation completes. Using the Read Status command, the status register should be polled until the Ready flag is at 1, signaling that the write operation has completed.

   ```
   Ready = Status_Register_Contents & 0x80
   ```

7. The Write and Program Error flags (bits 4 and 5) of the status register should also be checked to determine whether an error occurred during the write operation. Once the Ready flag is at 1, the status of both flags can be captured.

   ```
   Write_Error = Status_Register_Contents & 0x10
   Program_Error = Status_Register_Contents & 0x20
   ```

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
December 2011	Occurrences of CoreConsole were replaced with SmartDesign (SAR 24945).	N/A
	The following sentence was added to the introductory section of this chapter: "Except for AFS090, the read and write bus width for all devices is 32. The read and write bus width for AFS090 is 16. Internally, the FPGA core fabric can directly access the FB's data bus" (SARs 25234, 30296).	137
	The following note was added to Table 5-11 • Page Status Register Bit Definitions, for the Write Count register bit definition: "Max Write Count is set to 16,384 and the initial count is set to 2,288, which will flag when you write more than 14,096 cycles" (SAR 24408).	180

6 – FlashROM in Microsemi's Low Power Flash Devices

Introduction

The Fusion, IGLOO, and ProASIC3 families of low power flash-based devices have a dedicated nonvolatile FlashROM memory of 1,024 bits, which provides a unique feature in the FPGA market. The FlashROM can be read, modified, and written using the JTAG (or UJTAG) interface. It can be read but not modified from the FPGA core. Only low power flash devices contain on-chip user nonvolatile memory (NVM).

Architecture of User Nonvolatile FlashROM

Low power flash devices have 1 kbit of user-accessible nonvolatile flash memory on-chip that can be read from the FPGA core fabric. The FlashROM is arranged in eight banks of 128 bits (16 bytes) during programming. The 128 bits in each bank are addressable as 16 bytes during the read-back of the FlashROM from the FPGA core. Figure 6-1 shows the FlashROM logical structure.

The FlashROM can only be programmed via the IEEE 1532 JTAG port. It cannot be programmed directly from the FPGA core. When programming, each of the eight 128-bit banks can be selectively reprogrammed. The FlashROM can only be reprogrammed on a bank boundary. Programming involves an automatic, on-chip bank erase prior to reprogramming the bank. The FlashROM supports synchronous read. The address is latched on the rising edge of the clock, and the new output data is stable after the falling edge of the same clock cycle. For more information, refer to the timing diagrams in the DC and Switching Characteristics chapter of the appropriate datasheet. The FlashROM can be read on byte boundaries. The upper three bits of the FlashROM address from the FPGA core define the bank being accessed. The lower four bits of the FlashROM address from the FPGA core define which of the 16 bytes in the bank is being accessed.

		Byte Number in Bank 4 LSB of ADDR (READ)															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bank Number 3 MSB of ADDR (READ)	7																
	6																
	5																
	4																
	3																
	2																
	1																
	0																

***Figure 6-1* • FlashROM Architecture**

FlashROM Support in Flash-Based Devices

The flash FPGAs listed in Table 6-1 support the FlashROM feature and the functions described in this document.

***Table 6-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 6-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 6-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

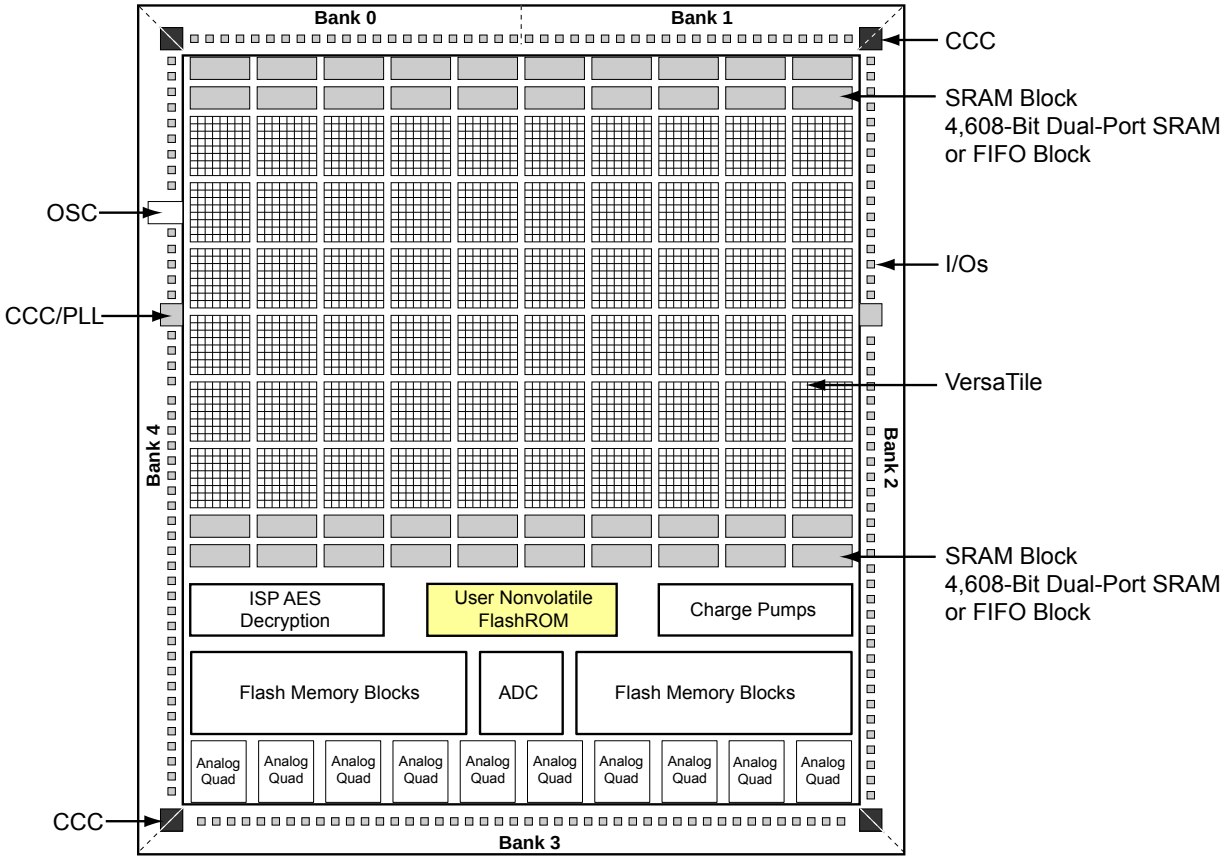


***Figure 6-2* • Fusion Device Architecture Overview (AFS600)**

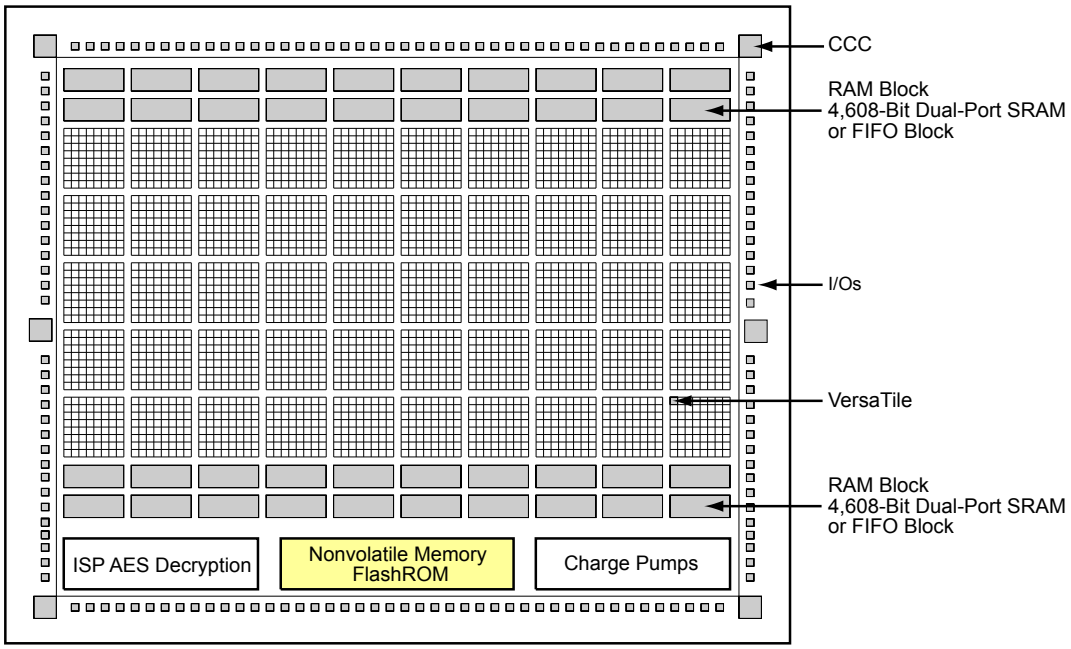


***Figure 6-3* • ProASIC3 and IGLOO Device Architecture**

FlashROM Applications

The SmartGen core generator is used to configure FlashROM content. You can configure each page independently. SmartGen enables you to create and modify regions within a page; these regions can be 1 to 16 bytes long (Figure 6-4).

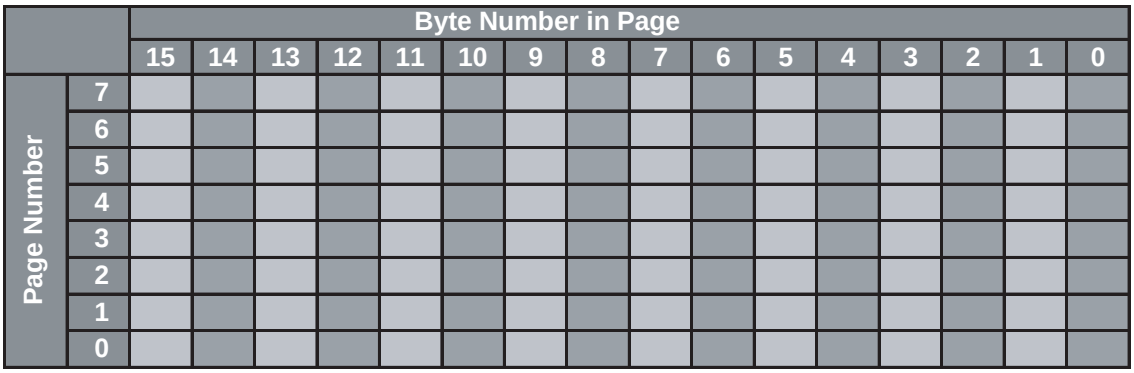


***Figure 6-4* • FlashROM Configuration**

The FlashROM content can be changed independently of the FPGA core content. It can be easily accessed and programmed via JTAG, depending on the security settings of the device. The SmartGen core generator enables each region to be independently updated (described in the "Programming and Accessing FlashROM" section on page 196). This enables you to change the FlashROM content on a per-part basis while keeping some regions "constant" for all parts. These features allow the FlashROM to be used in diverse system applications. Consider the following possible uses of FlashROM:

- Internet protocol (IP) addressing (wireless or fixed)
- System calibration settings
- Restoring configuration after unpredictable system power-down
- Device serialization and/or inventory control
- Subscription-based business models (e.g., set-top boxes)
- Secure key storage
- Asset management tracking
- Date stamping
- Version management

FlashROM Security

Low power flash devices have an on-chip Advanced Encryption Standard (AES) decryption core, combined with an enhanced version of the Microsemi flash-based lock technology (FlashLock®). Together, they provide unmatched levels of security in a programmable logic device. This security applies to both the FPGA core and FlashROM content. These devices use the 128-bit AES (Rijndael) algorithm to encrypt programming files for secure transmission to the on-chip AES decryption core. The same algorithm is then used to decrypt the programming file. This key size provides approximately 3.4×10^{38} possible 128-bit keys. A computing system that could find a DES key in a second would take approximately 149 trillion years to crack a 128-bit AES key. The 128-bit FlashLock feature in low power flash devices works via a FlashLock security Pass Key mechanism, where the user locks or unlocks the device with a user-defined key. Refer to the "Security in Low Power Flash Devices" section on page 365.

If the device is locked with certain security settings, functions such as device read, write, and erase are disabled. This unique feature helps to protect against invasive and noninvasive attacks. Without the correct Pass Key, access to the FPGA is denied. To gain access to the FPGA, the device first must be unlocked using the correct Pass Key. During programming of the FlashROM or the FPGA core, you can generate the security header programming file, which is used to program the AES key and/or FlashLock Pass Key. The security header programming file can also be generated independently of the FlashROM and FPGA core content. The FlashLock Pass Key is not stored in the FlashROM.

Low power flash devices with AES-based security allow for secure remote field updates over public networks such as the Internet, and ensure that valuable intellectual property (IP) remains out of the hands of IP thieves. Figure 6-5 shows this flow diagram.

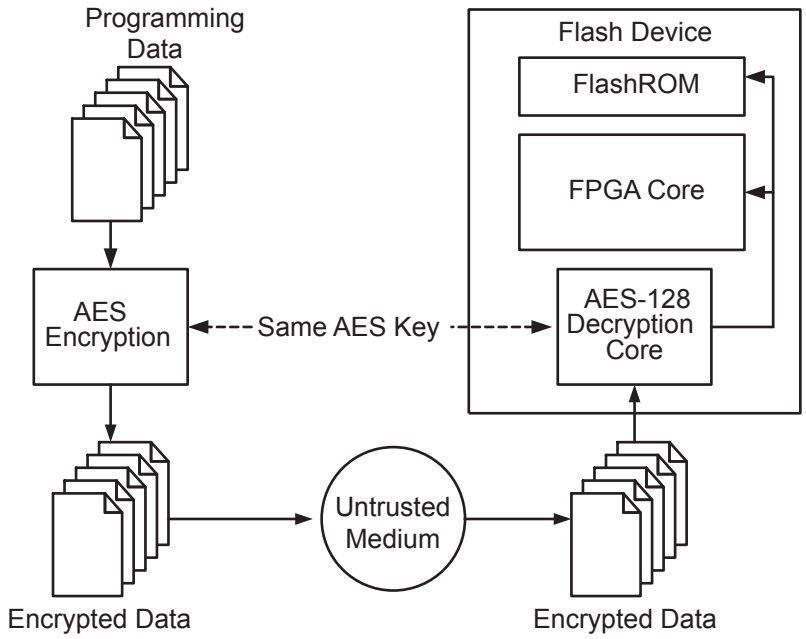


***Figure 6-5* • Programming FlashROM Using AES**

Programming and Accessing FlashROM

The FlashROM content can only be programmed via JTAG, but it can be read back selectively through the JTAG programming interface, the UJTAG interface, or via direct FPGA core addressing. The pages of the FlashROM can be made secure to prevent read-back via JTAG. In that case, read-back on these secured pages is only possible by the FPGA core fabric or via UJTAG.

A 7-bit address from the FPGA core defines which of the eight pages (three MSBs) is being read, and which of the 16 bytes within the selected page (four LSBs) are being read. The FlashROM content can be read on a random basis; the access time is 10 ns for a device supporting commercial specifications. The FPGA core will be powered down during writing of the FlashROM content. FPGA power-down during FlashROM programming is managed on-chip, and FPGA core functionality is not available during programming of the FlashROM. Table 6-2 summarizes various FlashROM access scenarios.

***Table 6-2* • FlashROM Read/Write Capabilities by Access Mode**

Access Mode	FlashROM Read	FlashROM Write
JTAG	Yes	Yes
UJTAG	Yes	No
FPGA core	Yes	No

Figure 6-6 shows the accessing of the FlashROM using the UJTAG macro. This is similar to FPGA core access, where the 7-bit address defines which of the eight pages (three MSBs) is being read and which of the 16 bytes within the selected page (four LSBs) are being read. Refer to the "UJTAG Applications in Microsemi's Low Power Flash Devices" section on page 419 for details on using the UJTAG macro to read the FlashROM.

Figure 6-7 on page 197 and Figure 6-8 on page 197 show the FlashROM access from the JTAG port. The FlashROM content can be read on a random basis. The three-bit address defines which page is being read or updated.

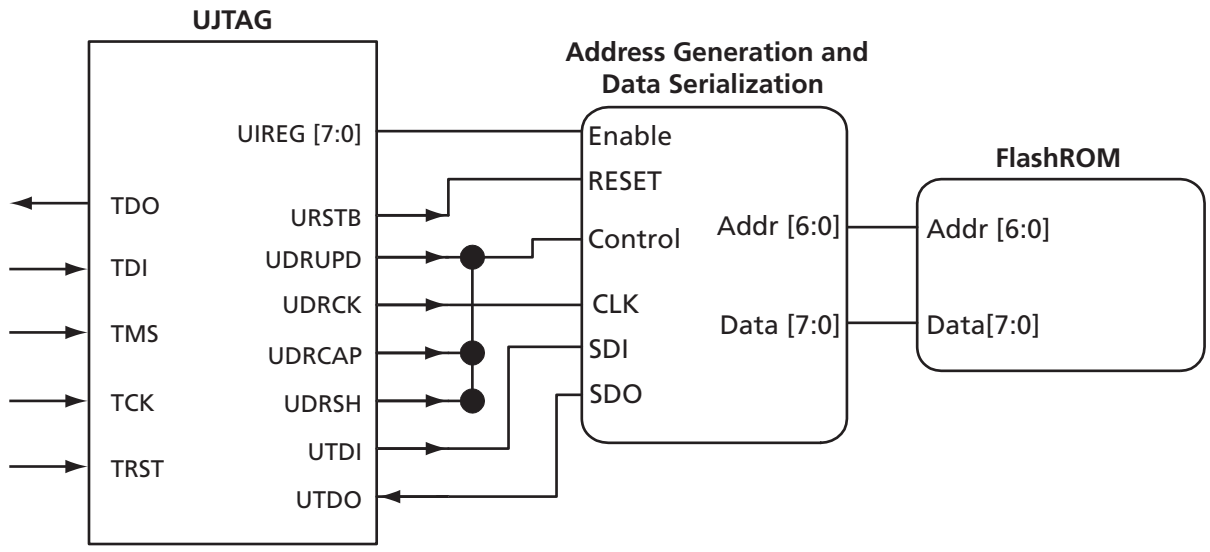


***Figure 6-6* • Block Diagram of Using UJTAG to Read FlashROM Contents**

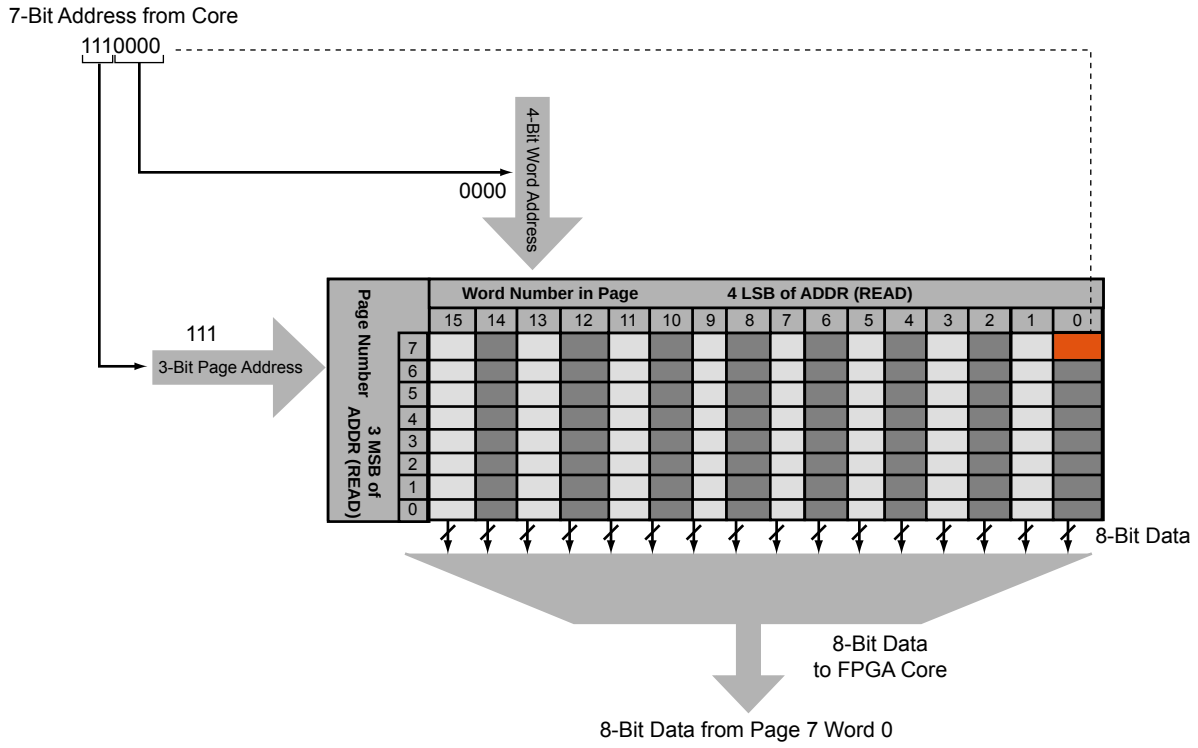


***Figure 6-7* • Accessing FlashROM Using FPGA Core**

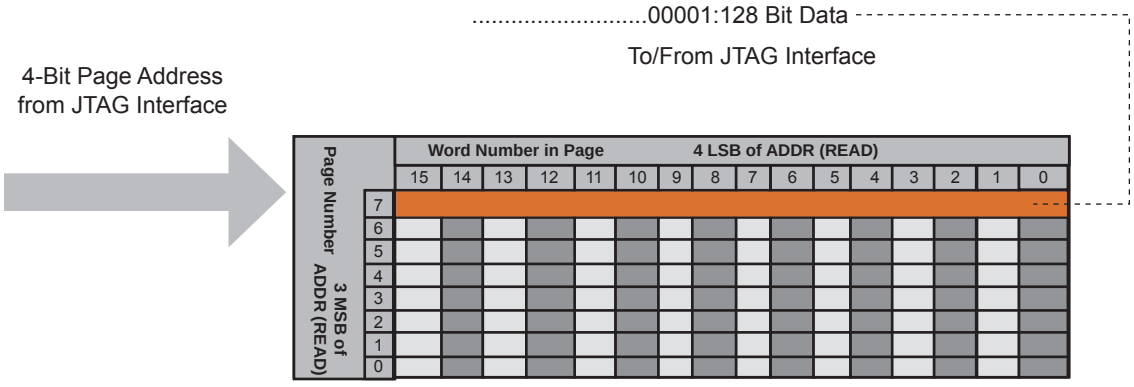


***Figure 6-8* • Accessing FlashROM Using JTAG Port**

FlashROM Design Flow

The Microsemi Libero System-on-Chip (SoC) software has extensive FlashROM support, including FlashROM generation, instantiation, simulation, and programming. Figure 6-9 shows the user flow diagram. In the design flow, there are three main steps:

1. FlashROM generation and instantiation in the design
2. Simulation of FlashROM design
3. Programming file generation for FlashROM design

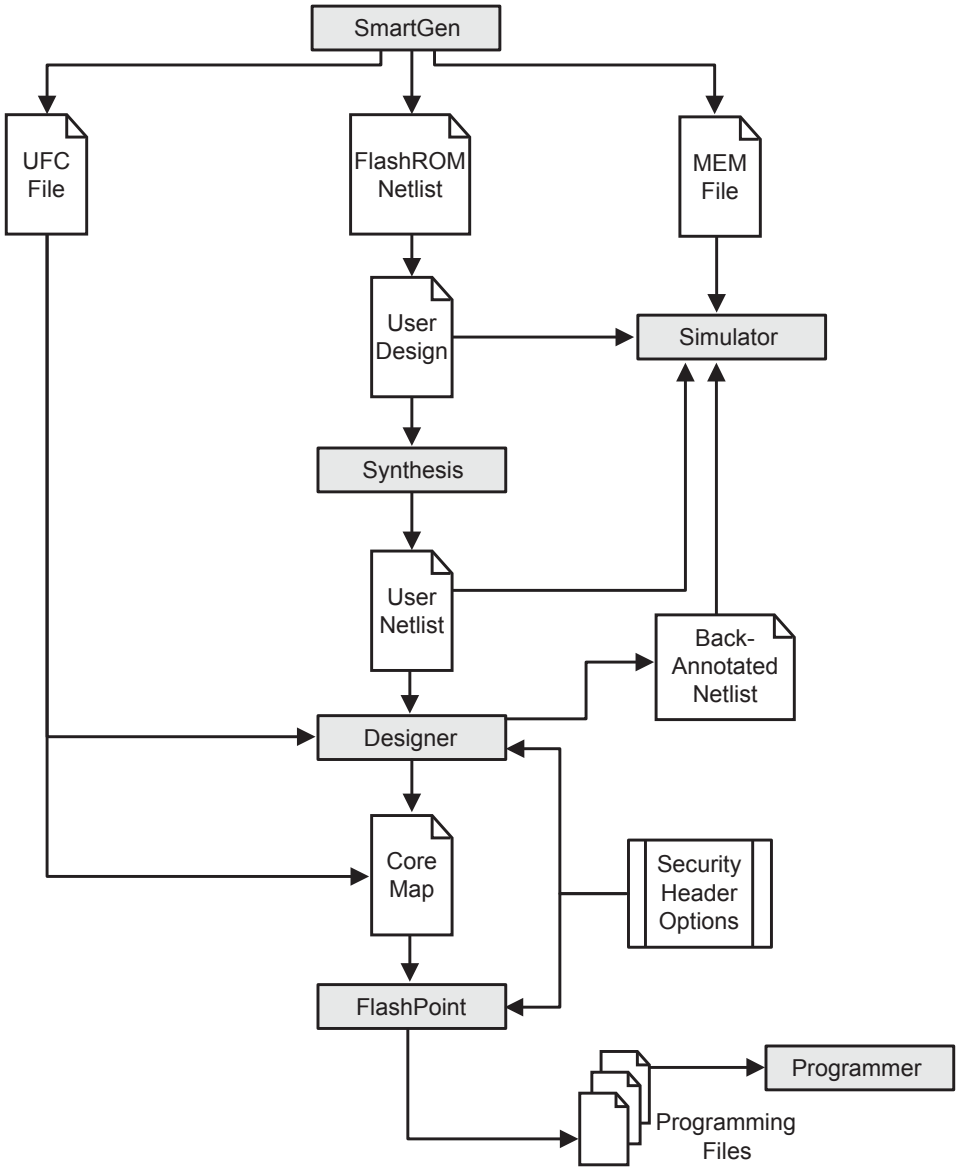


***Figure 6-9* • FlashROM Design Flow**

FlashROM Generation and Instantiation in the Design

The SmartGen core generator, available in Libero SoC and Designer, is the only tool that can be used to generate the FlashROM content. SmartGen has several user-friendly features to help generate the FlashROM contents. Instead of selecting each byte and assigning values, you can create a region within a page, modify the region, and assign properties to that region. The FlashROM user interface, shown in Figure 6-10, includes the configuration grid, existing regions list, and properties field. The properties field specifies the region-specific information and defines the data used for that region. You can assign values to the following properties:

1. Static Fixed Data—Enables you to fix the data so it cannot be changed during programming time. This option is useful when you have fixed data stored in this region, which is required for the operation of the design in the FPGA. Key storage is one example.
2. Static Modifiable Data—Select this option when the data in a particular region is expected to be static data (such as a version number, which remains the same for a long duration but could conceivably change in the future). This option enables you to avoid changing the value every time you enter new data.
3. Read from File—This provides the full flexibility of FlashROM usage to the customer. If you have a customized algorithm for generating the FlashROM data, you can specify this setting. You can then generate a text file with data for as many devices as you wish to program, and load that into the FlashPoint programming file generation software to get programming files that include all the data. SmartGen will optionally pass the location of the file where the data is stored if the file is specified in SmartGen. Each text file has only one type of data format (binary, decimal, hex, or ASCII text). The length of each data file must be shorter than or equal to the selected region length. If the data is shorter than the selected region length, the most significant bits will be padded with 0s. For multiple text files for multiple regions, the first lines are for the first device. In SmartGen, **Load Sim. Value From File** allows you to load the first device data in the MEM file for simulation.
4. Auto Increment/Decrement—This scenario is useful when you specify the contents of FlashROM for a large number of devices in a series. You can specify the step value for the serial number and a maximum value for inventory control. During programming file generation, the actual number of devices to be programmed is specified and a start value is fed to the software.

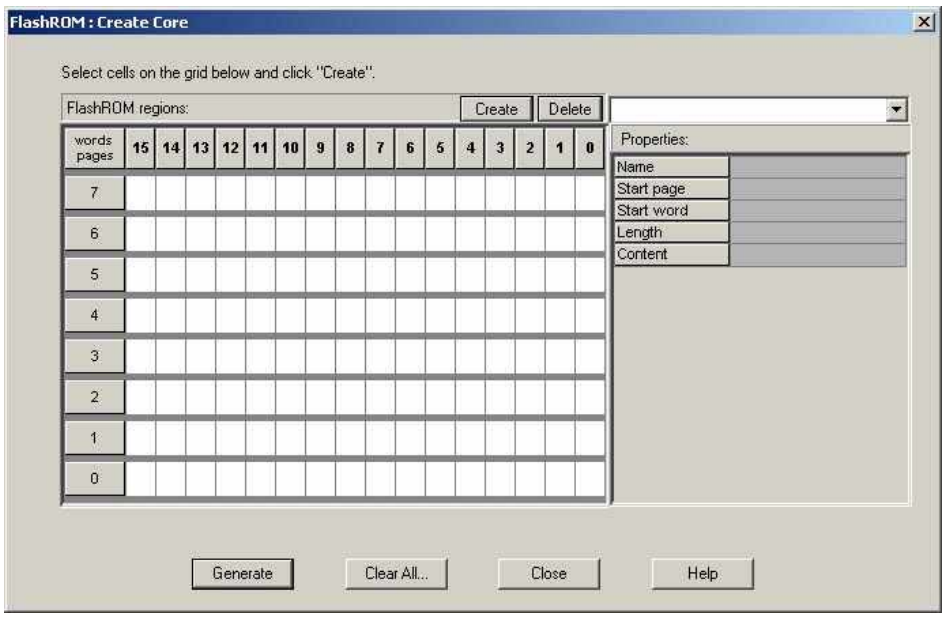


Figure 6-10 • **SmartGen GUI of the FlashROM**

SmartGen allows you to generate the FlashROM netlist in VHDL, Verilog, or EDIF format. After the FlashROM netlist is generated, the core can be instantiated in the main design like other SmartGen cores. Note that the macro library name for FlashROM is UFROM. The following is a sample FlashROM VHDL netlist that can be instantiated in the main design:

```
library ieee;
use ieee.std_logic_1164.all;
library fusion;

entity FROM_a is
  port( ADDR : in std_logic_vector(6 downto 0); DOUT : out std_logic_vector(7 downto 0));
end FROM_a;

architecture DEF_ARCH of  FROM_a is

  component UFROM
    generic (MEMORYFILE:string);
    port(DO0, DO1, DO2, DO3, DO4, DO5, DO6, DO7 : out std_logic;
      ADDR0, ADDR1, ADDR2, ADDR3, ADDR4, ADDR5, ADDR6 : in std_logic := 'U') ;
  end component;

  component GND
    port( Y : out std_logic);
  end component;

signal U_7_PIN2 : std_logic ;

begin

  GND_1_net : GND port map(Y => U_7_PIN2);
  UFROM0 : UFROM
  generic map(MEMORYFILE => "FROM_a.mem")
  port map(DO0 => DOUT(0), DO1 => DOUT(1), DO2 => DOUT(2), DO3 => DOUT(3), DO4 => DOUT(4),
    DO5 => DOUT(5), DO6 => DOUT(6), DO7 => DOUT(7), ADDR0 => ADDR(0), ADDR1 => ADDR(1),
    ADDR2 => ADDR(2), ADDR3 => ADDR(3), ADDR4 => ADDR(4), ADDR5 => ADDR(5),
    ADDR6 => ADDR(6));

end DEF_ARCH;
```

SmartGen generates the following files along with the netlist. These are located in the SmartGen folder for the Libero SoC project.

1. MEM (Memory Initialization) file
2. UFC (User Flash Configuration) file
3. Log file

The MEM file is used for simulation, as explained in the "Simulation of FlashROM Design" section on page 201. The UFC file, generated by SmartGen, has the FlashROM configuration for single or multiple devices and is used during STAPL generation. It contains the region properties and simulation values. Note that any changes in the MEM file will not be reflected in the UFC file. Do not modify the UFC to change FlashROM content. Instead, use the SmartGen GUI to modify the FlashROM content. See the "Programming File Generation for FlashROM Design" section on page 201 for a description of how the UFC file is used during the programming file generation. The log file has information regarding the file type and file location.

Simulation of FlashROM Design

The MEM file has 128 rows of 8 bits, each representing the contents of the FlashROM used for simulation. For example, the first row represents page 0, byte 0; the next row is page 0, byte 1; and so the pattern continues. Note that the three MSBs of the address define the page number, and the four LSBs define the byte number. So, if you send address 0000100 to FlashROM, this corresponds to the page 0 and byte 4 location, which is the fifth row in the MEM file. SmartGen defaults to 0s for any unspecified location of the FlashROM. Besides using the MEM file generated by SmartGen, you can create a binary file with 128 rows of 8 bits each and use this as a MEM file. Microsemi recommends that you use different file names if you plan to generate multiple MEM files. During simulation, Libero SoC passes the MEM file used as the generic file in the netlist, along with the design files and testbench. If you want to use different MEM files during simulation, you need to modify the generic file reference in the netlist.

```
……….
UFROM0: UFROM
--generic map(MEMORYFILE => "F:\Appsnotes\FROM\test_designs\testa\smartgen\FROM_a.mem")
--generic map(MEMORYFILE => "F:\Appsnotes\FROM\test_designs\testa\smartgen\FROM_b.mem")
……….
```

The VITAL and Verilog simulation models accept the generics passed by the netlist, read the MEM file, and perform simulation with the data in the file.

Programming File Generation for FlashROM Design

FlashPoint is the programming software used to generate the programming files for flash devices. Depending on the applications, you can use the FlashPoint software to generate a STAPL file with different FlashROM contents. In each case, optional AES decryption is available. To generate a STAPL file that contains the same FPGA core content and different FlashROM contents, the FlashPoint software needs an Array Map file for the core and UFC file(s) for the FlashROM. This final STAPL file represents the combination of the logic of the FPGA core and FlashROM content.

FlashPoint generates the STAPL files you can use to program the desired FlashROM page and/or FPGA core of the FPGA device contents. FlashPoint supports the encryption of the FlashROM content and/or FPGA Array configuration data. In the case of using the FlashROM for device serialization, a sequence of unique FlashROM contents will be generated. When generating a programming file with multiple unique FlashROM contents, you can specify in FlashPoint whether to include all FlashROM content in a single STAPL file or generate a different STAPL file for each FlashROM (Figure 6-11). The programming software (FlashPro) handles the single STAPL file that contains the FlashROM content from multiple devices. It enables you to program the FlashROM content into a series of devices sequentially (Figure 6-11). See the *FlashPro User's Guide* for information on serial programming.

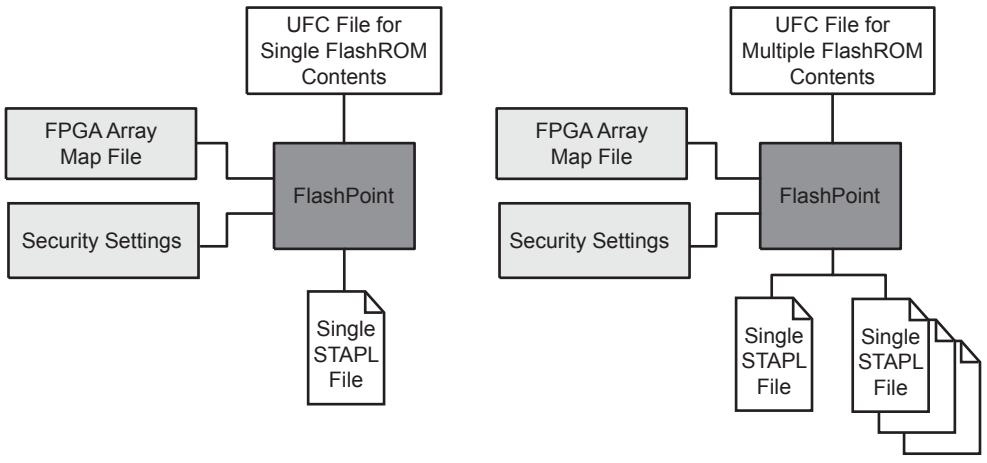


***Figure 6-11* • Single or Multiple Programming File Generation**

Figure 6-12 shows the programming file generator, which enables different STAPL file generation methods. When you select **Program FlashROM** and choose the UFC file, the FlashROM Settings window appears, as shown in Figure 6-13. In this window, you can select the FlashROM page you want to program and the data value for the configured regions. This enables you to use a different page for different programming files.

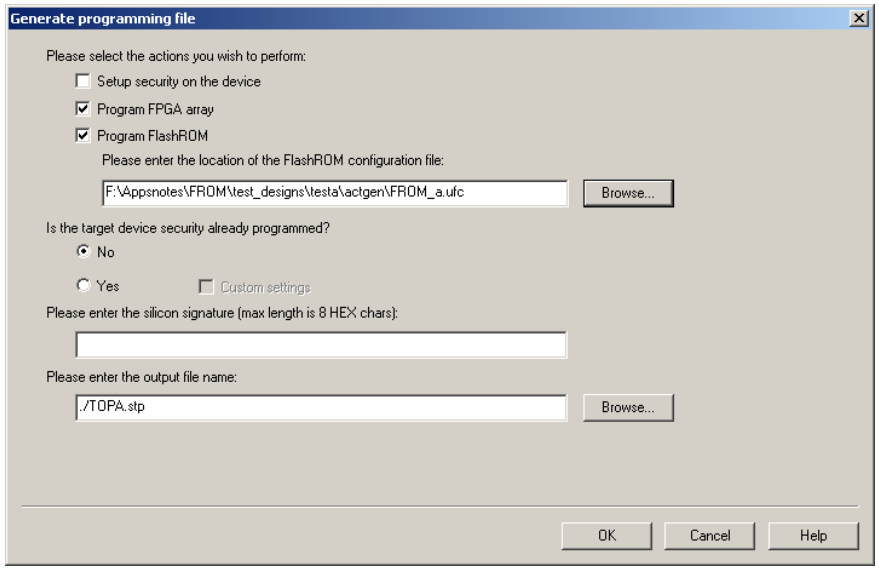


***Figure 6-12* • Programming File Generator**

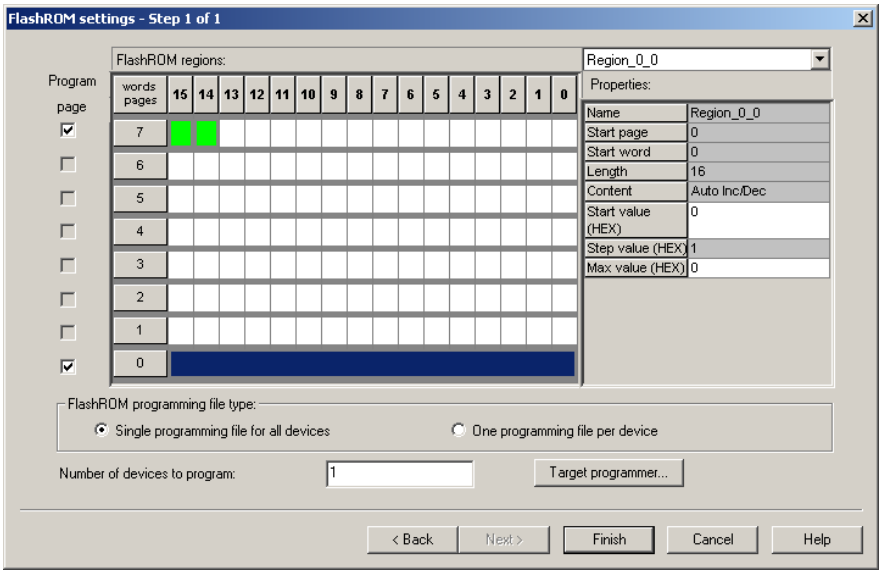


***Figure 6-13* • Setting FlashROM during Programming File Generation**

The programming hardware and software can load the FlashROM with the appropriate STAPL file. Programming software handles the single STAPL file that contains multiple FlashROM contents for multiple devices, and programs the FlashROM in sequential order (e.g., for device serialization). This feature is supported in the programming software. After programming with the STAPL file, you can run DEVICE_INFO to check the FlashROM content.

DEVICE_INFO displays the FlashROM content, serial number, Design Name, and checksum, as shown below:

```
EXPORT IDCODE[32] = 123261CF
EXPORT SILSIG[32] = 00000000
User information :
CHECKSUM: 61A0
Design Name:        TOP
Programming Method: STAPL
Algorithm Version: 1
Programmer: UNKNOWN
=======================================
FlashROM Information :
EXPORT Region_7_0[128] = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
=======================================
Security Setting :
Encrypted FlashROM Programming Enabled.
Encrypted FPGA Array Programming Enabled.
=======================================
```

The Libero SoC file manager recognizes the UFC and MEM files and displays them in the appropriate view. Libero SoC also recognizes the multiple programming files if you choose the option to generate multiple files for multiple FlashROM contents in Designer. These features enable a user-friendly flow for the FlashROM generation and programming in Libero SoC.

Custom Serialization Using FlashROM

You can use FlashROM for device serialization or inventory control by using the Auto Inc region or Read From File region. FlashPoint will automatically generate the serial number sequence for the Auto Inc region with the **Start Value**, **Max Value**, and **Step Value** provided. If you have a unique serial number generation scheme that you prefer, the Read From File region allows you to import the file with your serial number scheme programmed into the region. See the *FlashPro User's Guide* for custom serialization file format information.

The following steps describe how to perform device serialization or inventory control using FlashROM:

1. Generate FlashROM using SmartGen. From the Properties section in the FlashROM Settings dialog box, select the **Auto Inc** or **Read From File** region. For the Auto Inc region, specify the desired step value. You will not be able to modify this value in the FlashPoint software.
2. Go through the regular design flow and finish place-and-route.
3. Select **Programming File in Designer** and open **Generate Programming File** (Figure 6-12 on page 202).
4. Click **Program FlashROM**, browse to the UFC file, and click **Next**. The FlashROM Settings window appears, as shown in Figure 6-13 on page 202.
5. Select the FlashROM page you want to program and the data value for the configured regions. The STAPL file generated will contain only the data that targets the selected FlashROM page.
6. Modify properties for the serialization.
   – For the Auto Inc region, specify the **Start** and **Max** values.
   – For the Read From File region, select the file name of the custom serialization file.
7. Select the FlashROM programming file type you want to generate from the two options below:
   – Single programming file for all devices: generates one programming file with all FlashROM values.
   – One programming file per device: generates a separate programming file for each FlashROM value.
8. Enter the number of devices you want to program and generate the required programming file.
9. Open the programming software and load the programming file. The programming software, FlashPro3 and Silicon Sculptor II, supports the device serialization feature. If, for some reason, the device fails to program a part during serialization, the software allows you to reuse or skip the serial data. Refer to the *FlashPro User's Guide* for details.

Conclusion

The Fusion, IGLOO, and ProASIC3 families are the only FPGAs that offer on-chip FlashROM support. This document presents information on the FlashROM architecture, possible applications, programming, access through the JTAG and UJTAG interface, and integration into your design. In addition, the Libero tool set enables easy creation and modification of the FlashROM content.

The nonvolatile FlashROM block in the FPGA can be customized, enabling multiple applications.

Additionally, the security offered by the low power flash devices keeps both the contents of FlashROM and the FPGA design safe from system over-builders, system cloners, and IP thieves.

Related Documents

User's Guides

FlashPro User's Guide

http://www.microsemi.com/documents/FlashPro_UG.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 6-1 • Flash-Based FPGAs.	192
v1.3 (October 2008)	The "FlashROM Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	192
	Figure 6-2 • Fusion Device Architecture Overview (AFS600) was replaced. Figure 6-5 • Programming FlashROM Using AES was revised to change "Fusion" to "Flash Device."	193, 195
	The *FlashPoint User's Guide* was removed from the "User's Guides" section, as its content is now part of the *FlashPro User's Guide*.	204
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 6-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	192
v1.1 (March 2008)	The chapter was updated to include the IGLOO PLUS family and information regarding 15 k gate devices. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	N/A

7 – SRAM and FIFO Memories in Microsemi's Low Power Flash Devices

Introduction

As design complexity grows, greater demands are placed upon an FPGA's embedded memory. Fusion, IGLOO, and ProASIC3 devices provide the flexibility of true dual-port and two-port SRAM blocks. The embedded memory, along with built-in, dedicated FIFO control logic, can be used to create cascading RAM blocks and FIFOs without using additional logic gates.

IGLOO, IGLOO PLUS, and ProASIC3L FPGAs contain an additional feature that allows the device to be put in a low power mode called Flash*Freeze. In this mode, the core draws minimal power (on the order of 2 to 127 µW) and still retains values on the embedded SRAM/FIFO and registers. Flash*Freeze technology allows the user to switch to Active mode on demand, thus simplifying power management and the use of SRAM/FIFOs.

Device Architecture

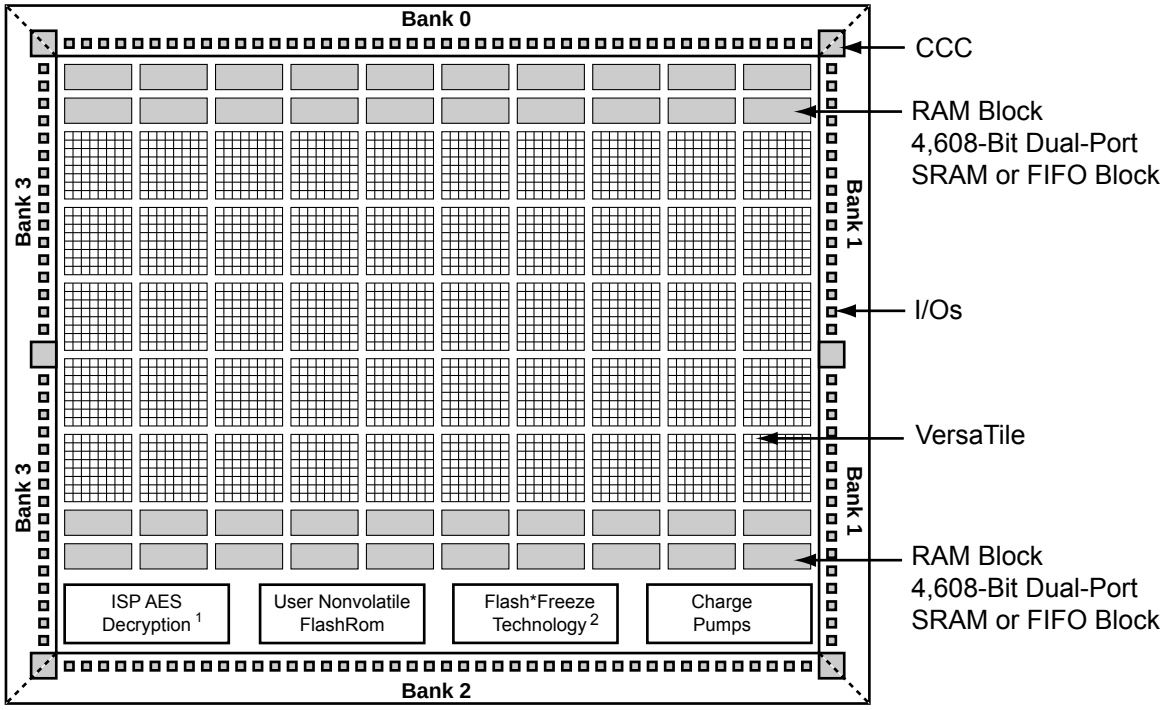
The low power flash devices feature up to 504 kbits of RAM in 4,608-bit blocks (Figure 7-1 on page 206 and Figure 7-2 on page 207). The total embedded SRAM for each device can be found in the datasheets. These memory blocks are arranged along the top and bottom of the device to allow better access from the core and I/O (in some devices, they are only available on the north side of the device). Every RAM block has a flexible, hardwired, embedded FIFO controller, enabling the user to implement efficient FIFOs without sacrificing user gates.

In the IGLOO and ProASIC3 families of devices, the following memories are supported:

- 30 k gate devices and smaller do not support SRAM and FIFO.
- 60 k and 125 k gate devices support memories on the north side of the device only.
- 250 k devices and larger support memories on the north and south sides of the device.

In Fusion devices, the following memories are supported:

- AFS090 and AFS250 support memories on the north side of the device only.
- AFS600 and AFS1500 support memories on the north and south sides of the device.



Notes:

1. *AES decryption not supported in 30 k gate devices and smaller.*
2. *Flash*Freeze is supported in all IGLOO devices and the ProASIC3L devices.*

***Figure 7-1* • IGLOO and ProASIC3 Device Architecture Overview**

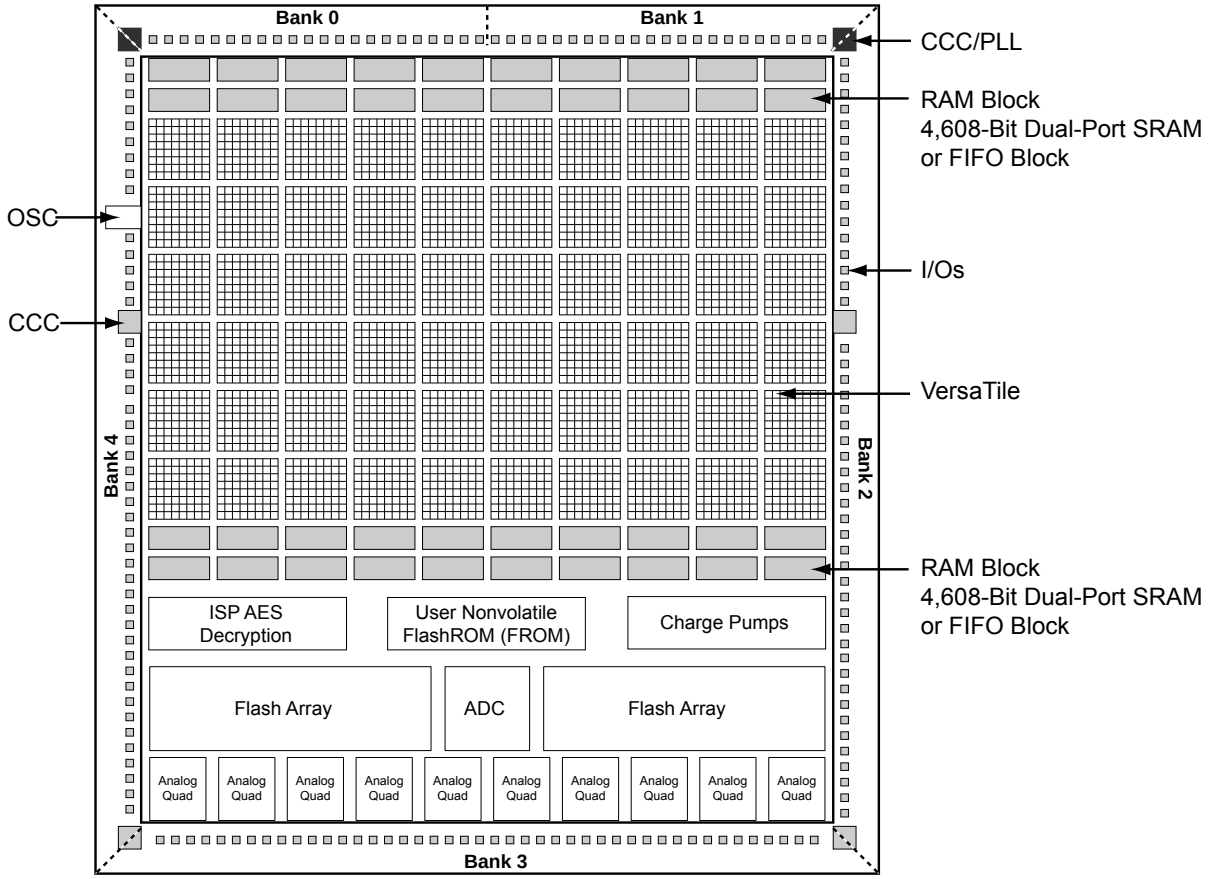


***Figure 7-2* • Fusion Device Architecture Overview (AFS600)**

SRAM/FIFO Support in Flash-Based Devices

The flash FPGAs listed in Table 7-1 support SRAM and FIFO blocks and the functions described in this document.

***Table 7-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 7-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 7-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

SRAM and FIFO Architecture

To meet the needs of high-performance designs, the memory blocks operate strictly in synchronous mode for both read and write operations. The read and write clocks are completely independent, and each can operate at any desired frequency up to 250 MHz.

- 4k×1, 2k×2, 1k×4, 512×9 (dual-port RAM—2 read / 2 write or 1 read / 1 write)
- 512×9, 256×18 (2-port RAM—1 read / 1 write)
- Sync write, sync pipelined / nonpipelined read

Automotive ProASIC3 devices support single-port SRAM capabilities or dual-port SRAM only under specific conditions. Dual-port mode is supported if the clocks to the two SRAM ports are the same and 180° out of phase (i.e., the port A clock is the inverse of the port B clock). The Libero SoC software macro libraries support a dual-port macro only. For use of this macro as a single-port SRAM, the inputs and clock of one port should be tied off (grounded) to prevent errors during design compile. For use in dual-port mode, the same clock with an inversion between the two clock pins of the macro should be used in the design to prevent errors during compile.

The memory block includes dedicated FIFO control logic to generate internal addresses and external flag logic (FULL, EMPTY, AFULL, AEMPTY).

Simultaneous dual-port read/write and write/write operations at the same address are allowed when certain timing requirements are met.

During RAM operation, addresses are sourced by the user logic, and the FIFO controller is ignored. In FIFO mode, the internal addresses are generated by the FIFO controller and routed to the RAM array by internal MUXes.

The low power flash device architecture enables the read and write sizes of RAMs to be organized independently, allowing for bus conversion. For example, the write size can be set to 256×18 and the read size to 512×9.

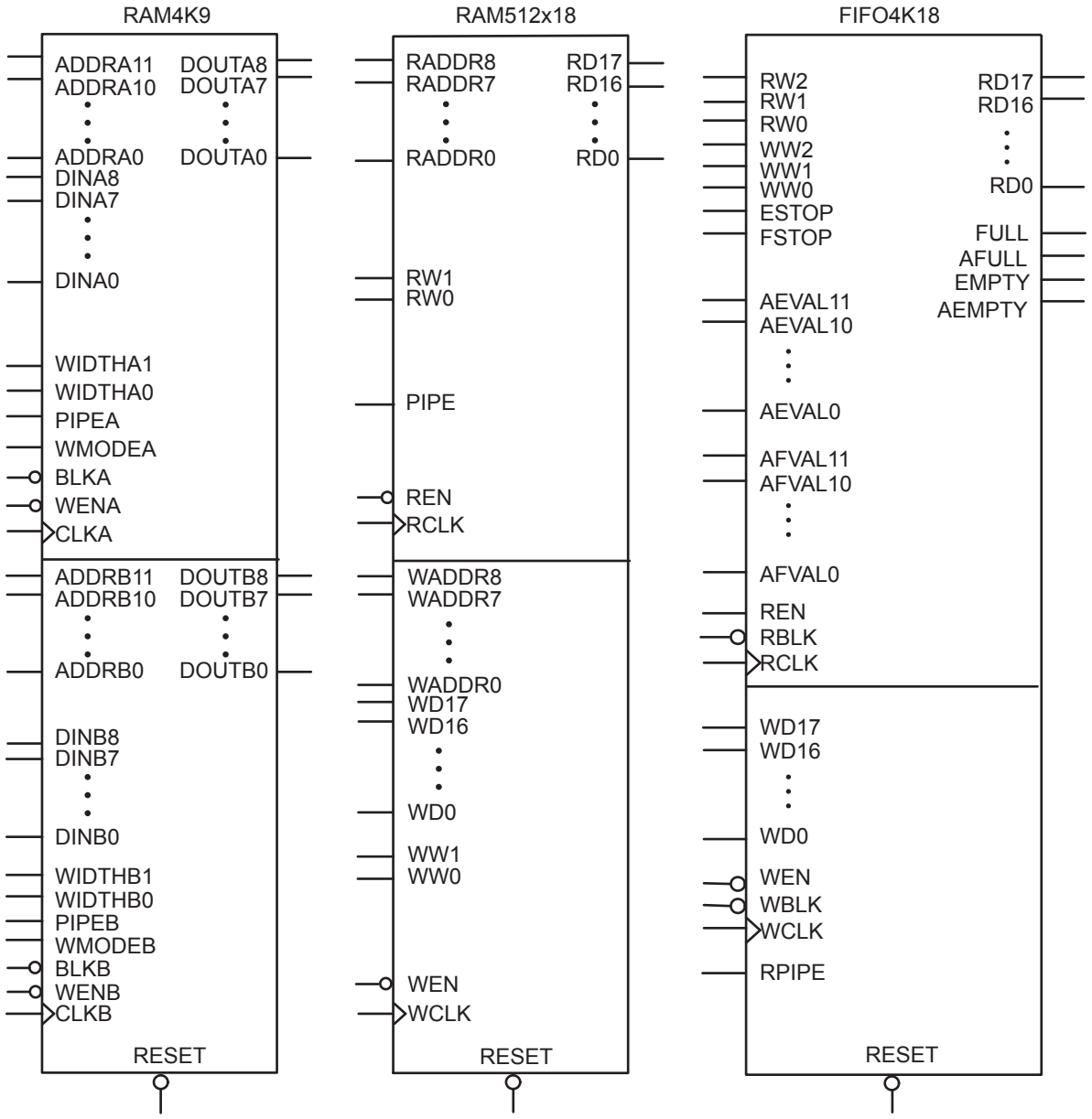
Both the write width and read width for the RAM blocks can be specified independently with the WW (write width) and RW (read width) pins. The different D×W configurations are 256×18, 512×9, 1k×4, 2k×2, and 4k×1. When widths of one, two, or four are selected, the ninth bit is unused. For example, when writing nine-bit values and reading four-bit values, only the first four bits and the second four bits of each nine-bit value are addressable for read operations. The ninth bit is not accessible.

Conversely, when writing four-bit values and reading nine-bit values, the ninth bit of a read operation will be undefined. The RAM blocks employ little-endian byte order for read and write operations.

Memory Blocks and Macros

Memory blocks can be configured with many different aspect ratios, but are generically supported in the macro libraries as one of two memory elements: RAM4K9 or RAM512X18. The RAM4K9 is configured as a true dual-port memory block, and the RAM512X18 is configured as a two-port memory block. Dual-port memory allows the RAM to both read from and write to either port independently. Two-port memory allows the RAM to read from one port and write to the other using a common clock or independent read and write clocks. If needed, the RAM4K9 blocks can be configured as two-port memory blocks. The memory block can be configured as a FIFO by combining the basic memory block with dedicated FIFO controller logic. The FIFO macro is named FIFO4KX18 (Figure 7-3 on page 210).

Clocks for the RAM blocks can be driven by the VersaNet (global resources) or by regular nets. When using local clock segments, the clock segment region that encompasses the RAM blocks can drive the RAMs. In the dual-port configuration (RAM4K9), each memory block port can be driven by either rising-edge or falling-edge clocks. Each port can be driven by clocks with different edges. Though only a rising-edge clock can drive the physical block itself, the Microsemi Designer software will automatically bubble-push the inversion to properly implement the falling-edge trigger for the RAM block.



Notes:

1. *Automotive ProASIC3 devices restrict RAM4K9 to a single port or to dual ports with the same clock 180° out of phase (inverted) between clock pins. In single-port mode, inputs to port B should be tied to ground to prevent errors during compile. This warning applies only to automotive ProASIC3 parts of certain revisions and earlier. Contact Technical Support at soc_tech@microsemi.com for information on the revision number for a particular lot and date code.*
2. *For FIFO4K18, the same clock 180° out of phase (inverted) between clock pins should be used.*

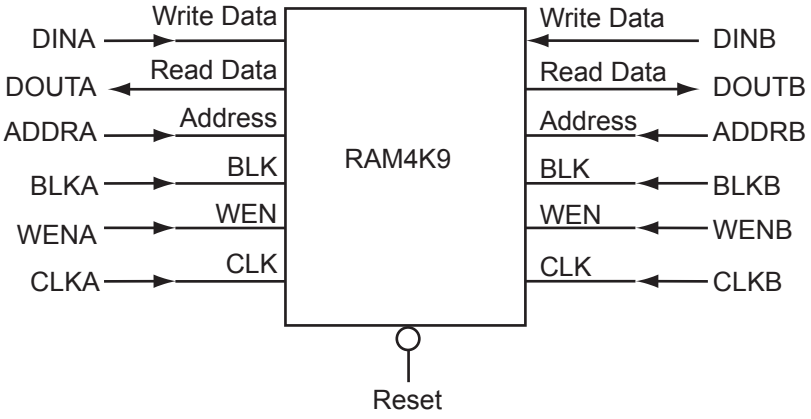
***Figure 7-3* • Supported Basic RAM Macros**

SRAM Features

RAM4K9 Macro

RAM4K9 is the dual-port configuration of the RAM block (Figure 7-4). The RAM4K9 nomenclature refers to both the deepest possible configuration and the widest possible configuration the dual-port RAM block can assume, and does not denote a possible memory aspect ratio. The RAM block can be configured to the following aspect ratios: 4,096×1, 2,048×2, 1,024×4, and 512×9. RAM4K9 is fully synchronous and has the following features:

- Two ports that allow fully independent reads and writes at different frequencies
- Selectable pipelined or nonpipelined read
- Active-low block enables for each port
- Toggle control between read and write mode for each port
- Active-low asynchronous reset
- Pass-through write data or hold existing data on output. In pass-through mode, the data written to the write port will immediately appear on the read port.
- Designer software will automatically facilitate falling-edge clocks by bubble-pushing the inversion to previous stages.



Note: For timing diagrams of the RAM signals, refer to the appropriate family datasheet.

***Figure 7-4* • RAM4K9 Simplified Configuration**

Signal Descriptions for RAM4K9

Note: Automotive ProASIC3 devices support single-port SRAM capabilities, or dual-port SRAM only under specific conditions. Dual-port mode is supported if the clocks to the two SRAM ports are the same and 180° out of phase (i.e., the port A clock is the inverse of the port B clock). Since Libero SoC macro libraries support a dual-port macro only, certain modifications must be made. These are detailed below.

The following signals are used to configure the RAM4K9 memory element:

WIDTHA and WIDTHB

These signals enable the RAM to be configured in one of four allowable aspect ratios (Table 7-2 on page 212).

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, WIDTHB should be tied to ground.

***Table 7-2* • Allowable Aspect Ratio Settings for WIDTHA[1:0]**

WIDTHA[1:0]	WIDTHB[1:0]	D×W
00	00	4k×1
01	01	2k×2
10	10	1k×4
11	11	512×9

Note: The aspect ratio settings are constant and cannot be changed on the fly.

BLKA and BLKB

These signals are active-low and will enable the respective ports when asserted. When a BLKx signal is deasserted, that port's outputs hold the previous value.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, BLKB should be tied to ground.

WENA and WENB

These signals switch the RAM between read and write modes for the respective ports. A LOW on these signals indicates a write operation, and a HIGH indicates a read.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, WENB should be tied to ground.

CLKA and CLKB

These are the clock signals for the synchronous read and write operations. These can be driven independently or with the same driver.

Note: For Automotive ProASIC3 devices, dual-port mode is supported if the clocks to the two SRAM ports are the same and 180° out of phase (i.e., the port A clock is the inverse of the port B clock). For use of this macro as a single-port SRAM, the inputs and clock of one port should be tied off (grounded) to prevent errors during design compile.

PIPEA and PIPEB

These signals are used to specify pipelined read on the output. A LOW on PIPEA or PIPEB indicates a nonpipelined read, and the data appears on the corresponding output in the same clock cycle. A HIGH indicates a pipelined read, and data appears on the corresponding output in the next clock cycle.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, PIPEB should be tied to ground. For use in dual-port mode, the same clock with an inversion between the two clock pins of the macro should be used in the design to prevent errors during compile.

WMODEA and WMODEB

These signals are used to configure the behavior of the output when the RAM is in write mode. A LOW on these signals makes the output retain data from the previous read. A HIGH indicates pass-through behavior, wherein the data being written will appear immediately on the output. This signal is overridden when the RAM is being read.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, WMODEB should be tied to ground.

RESET

This active-low signal resets the control logic, forces the output hold state registers to zero, disables reads and writes from the SRAM block, and clears the data hold registers when asserted. It does not reset the contents of the memory array.

While the RESET signal is active, read and write operations are disabled. As with any asynchronous reset signal, care must be taken not to assert it too close to the edges of active read and write clocks.

ADDRA and ADDRB

These are used as read or write addresses, and they are 12 bits wide. When a depth of less than 4 k is specified, the unused high-order bits must be grounded (Table 7-3 on page 213).

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, ADDRB should be tied to ground.

***Table 7-3* • Address Pins Unused/Used for Various Supported Bus Widths**

D×W	ADDRx	
	Unused	Used
4k×1	None	[11:0]
2k×2	[11]	[10:0]
1k×4	[11:10]	[9:0]
512×9	[11:9]	[8:0]

Note: The "x" in ADDRx implies A or B.

DINA and DINB

These are the input data signals, and they are nine bits wide. Not all nine bits are valid in all configurations. When a data width less than nine is specified, unused high-order signals must be grounded (Table 7-4).

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, DINB should be tied to ground.

DOUTA and DOUTB

These are the nine-bit output data signals. Not all nine bits are valid in all configurations. As with DINA and DINB, high-order bits may not be used (Table 7-4). The output data on unused pins is undefined.

***Table 7-4* • Unused/Used Input and Output Data Pins for Various Supported Bus Widths**

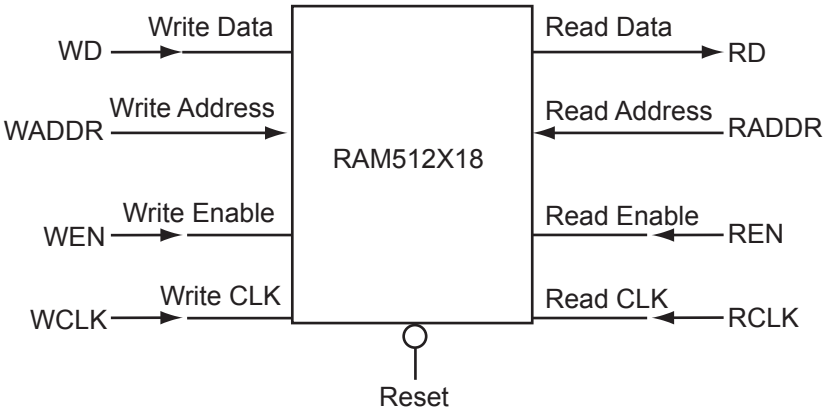
D×W	DINx/DOUTx	
	Unused	Used
4k×1	[8:1]	[0]
2k×2	[8:2]	[1:0]
1k×4	[8:4]	[3:0]
512×9	None	[8:0]

Note: The "x" in DINx or DOUTx implies A or B.

RAM512X18 Macro

RAM512X18 is the two-port configuration of the same RAM block (Figure 7-5 on page 214). Like the RAM4K9 nomenclature, the RAM512X18 nomenclature refers to both the deepest possible configuration and the widest possible configuration the two-port RAM block can assume. In two-port mode, the RAM block can be configured to either the 512×9 aspect ratio or the 256×18 aspect ratio. RAM512X18 is also fully synchronous and has the following features:

- Dedicated read and write ports
- Active-low read and write enables
- Selectable pipelined or nonpipelined read
- Active-low asynchronous reset
- Designer software will automatically facilitate falling-edge clocks by bubble-pushing the inversion to previous stages.



Note: For timing diagrams of the RAM signals, refer to the appropriate family datasheet.

Figure 7-5 • **512X18 Two-Port RAM Block Diagram**

Signal Descriptions for RAM512X18

RAM512X18 has slightly different behavior from RAM4K9, as it has dedicated read and write ports.

WW and RW

These signals enable the RAM to be configured in one of the two allowable aspect ratios (Table 7-5).

Table 7-5 • **Aspect Ratio Settings for WW[1:0]**

WW[1:0]	RW[1:0]	D×W
01	01	512×9
10	10	256×18
00, 11	00, 11	Reserved

WD and RD

These are the input and output data signals, and they are 18 bits wide. When a 512×9 aspect ratio is used for write, WD[17:9] are unused and must be grounded. If this aspect ratio is used for read, RD[17:9] are undefined.

WADDR and RADDR

These are read and write addresses, and they are nine bits wide. When the 256×18 aspect ratio is used for write or read, WADDR[8] and RADDR[8] are unused and must be grounded.

WCLK and RCLK

These signals are the write and read clocks, respectively. They can be clocked on the rising or falling edge of WCLK and RCLK.

WEN and REN

These signals are the write and read enables, respectively. They are both active-low by default. These signals can be configured as active-high.

RESET

This active-low signal resets the control logic, forces the output hold state registers to zero, disables reads and writes from the SRAM block, and clears the data hold registers when asserted. It does not reset the contents of the memory array.

While the RESET signal is active, read and write operations are disabled. As with any asynchronous reset signal, care must be taken not to assert it too close to the edges of active read and write clocks.

PIPE

This signal is used to specify pipelined read on the output. A LOW on PIPE indicates a nonpipelined read, and the data appears on the output in the same clock cycle. A HIGH indicates a pipelined read, and data appears on the output in the next clock cycle.

SRAM Usage

The following descriptions refer to the usage of both RAM4K9 and RAM512X18.

Clocking

The dual-port SRAM blocks are only clocked on the rising edge. SmartGen allows falling-edge-triggered clocks by adding inverters to the netlist, hence achieving dual-port SRAM blocks that are clocked on either edge (rising or falling). For dual-port SRAM, each port can be clocked on either edge and by separate clocks by port. Note that for Automotive ProASIC3, the same clock, with an inversion between the two clock pins of the macro, should be used in design to prevent errors during compile.

Low power flash devices support inversion (bubble-pushing) throughout the FPGA architecture, including the clock input to the SRAM modules. Inversions added to the SRAM clock pin on the design schematic or in the HDL code will be automatically accounted for during design compile without incurring additional delay in the clock path.

The two-port SRAM can be clocked on the rising or falling edge of WCLK and RCLK.

If negative-edge RAM and FIFO clocking is selected for memory macros, clock edge inversion management (bubble-pushing) is automatically used within the development tools, without performance penalty.

Modes of Operation

There are two read modes and one write mode:

- Read Nonpipelined (synchronous—1 clock edge): In the standard read mode, new data is driven onto the RD bus in the same clock cycle following RA and REN valid. The read address is registered on the read port clock active edge, and data appears at RD after the RAM access time. Setting PIPE to OFF enables this mode.
- Read Pipelined (synchronous—2 clock edges): The pipelined mode incurs an additional clock delay from address to data but enables operation at a much higher frequency. The read address is registered on the read port active clock edge, and the read data is registered and appears at RD after the second read clock edge. Setting PIPE to ON enables this mode.
- Write (synchronous—1 clock edge): On the write clock active edge, the write data is written into the SRAM at the write address when WEN is HIGH. The setup times of the write address, write enables, and write data are minimal with respect to the write clock.

RAM Initialization

Each SRAM block can be individually initialized on power-up by means of the JTAG port using the UJTAG mechanism. The shift register for a target block can be selected and loaded with the proper bit configuration to enable serial loading. The 4,608 bits of data can be loaded in a single operation.

FIFO Features

The FIFO4KX18 macro is created by merging the RAM block with dedicated FIFO logic (Figure 7-6 on page 216). Since the FIFO logic can only be used in conjunction with the memory block, there is no separate FIFO controller macro. As with the RAM blocks, the FIFO4KX18 nomenclature does not refer to a possible aspect ratio, but rather to the deepest possible data depth and the widest possible data width. FIFO4KX18 can be configured into the following aspect ratios: 4,096×1, 2,048×2, 1,024×4, 512×9, and 256×18. In addition to being fully synchronous, the FIFO4KX18 also has the following features:

- Four FIFO flags: Empty, Full, Almost-Empty, and Almost-Full
- Empty flag is synchronized to the read clock
- Full flag is synchronized to the write clock
- Both Almost-Empty and Almost-Full flags have programmable thresholds
- Active-low asynchronous reset
- Active-low block enable
- Active-low write enable
- Active-high read enable
- Ability to configure the FIFO to either stop counting after the empty or full states are reached or to allow the FIFO counters to continue

- Designer software will automatically facilitate falling-edge clocks by bubble-pushing the inversion to previous stages.

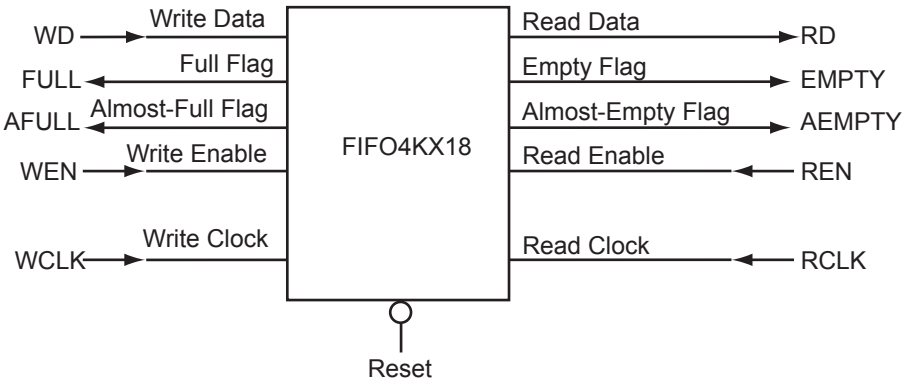


***Figure 7-6* • FIFO4KX18 Block Diagram**

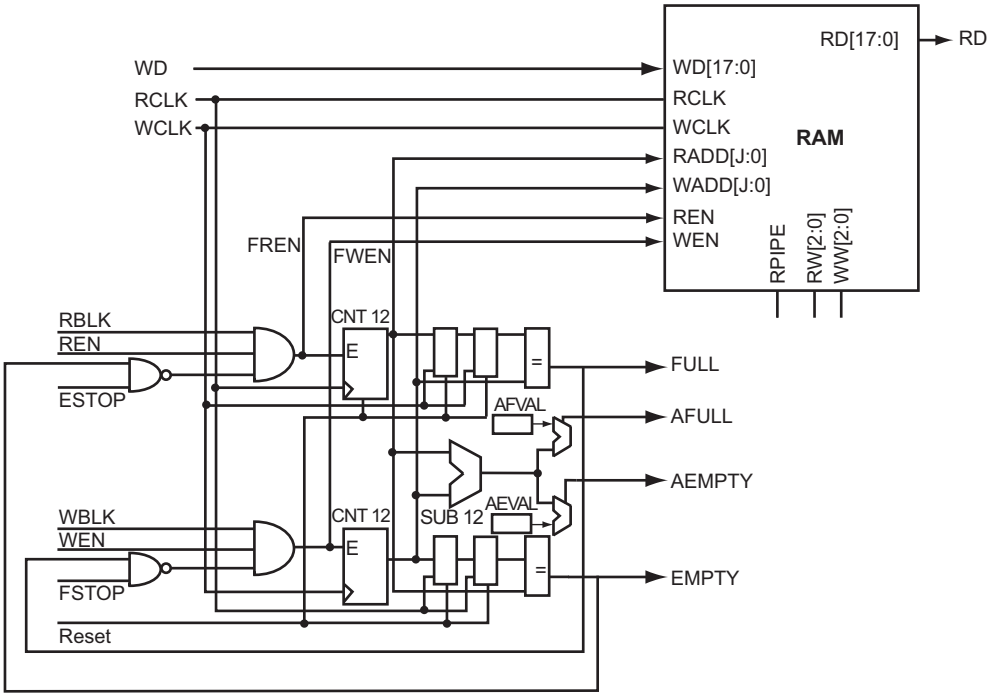


***Figure 7-7* • RAM Block with Embedded FIFO Controller**

The FIFOs maintain a separate read and write address. Whenever the difference between the write address and the read address is greater than or equal to the almost-full value (AFVAL), the Almost-Full flag is asserted. Similarly, the Almost-Empty flag is asserted whenever the difference between the write address and read address is less than or equal to the almost-empty value (AEVAL).

Due to synchronization between the read and write clocks, the Empty flag will deassert after the second read clock edge from the point that the write enable asserts. However, since the Empty flag is synchronized to the read clock, it will assert after the read clock reads the last data in the FIFO. Also, since the Full flag is dependent on the actual hardware configuration, it will assert when the actual physical implementation of the FIFO is full.

For example, when a user configures a 128×18 FIFO, the actual physical implementation will be a 256×18 FIFO element. Since the actual implementation is 256×18, the Full flag will not trigger until the

256×18 FIFO is full, even though a 128×18 FIFO was requested. For this example, the Almost-Full flag can be used instead of the Full flag to signal when the 128th data word is reached.

To accommodate different aspect ratios, the almost-full and almost-empty values are expressed in terms of data bits instead of data words. SmartGen translates the user's input, expressed in data words, into data bits internally. SmartGen allows the user to select the thresholds for the Almost-Empty and Almost-Full flags in terms of either the read data words or the write data words, and makes the appropriate conversions for each flag.

After the empty or full states are reached, the FIFO can be configured so the FIFO counters either stop or continue counting. For timing numbers, refer to the appropriate family datasheet.

Signal Descriptions for FIFO4K18

The following signals are used to configure the FIFO4K18 memory element:

WW and RW

These signals enable the FIFO to be configured in one of the five allowable aspect ratios (Table 7-6).

***Table 7-6* • Aspect Ratio Settings for WW[2:0]**

WW[2:0]	RW[2:0]	D×W
000	000	4k×1
001	001	2k×2
010	010	1k×4
011	011	512×9
100	100	256×18
101, 110, 111	101, 110, 111	Reserved

WBLK and RBLK

These signals are active-low and will enable the respective ports when LOW. When the RBLK signal is HIGH, that port's outputs hold the previous value.

WEN and REN

Read and write enables. WEN is active-low and REN is active-high by default. These signals can be configured as active-high or -low.

WCLK and RCLK

These are the clock signals for the synchronous read and write operations. These can be driven independently or with the same driver.

Note: For the Automotive ProASIC3 FIFO4K18, for the same clock, 180° out of phase (inverted) between clock pins should be used.

RPIPE

This signal is used to specify pipelined read on the output. A LOW on RPIPE indicates a nonpipelined read, and the data appears on the output in the same clock cycle. A HIGH indicates a pipelined read, and data appears on the output in the next clock cycle.

RESET

This active-low signal resets the control logic and forces the output hold state registers to zero when asserted. It does not reset the contents of the memory array (Table 7-7 on page 218).

While the RESET signal is active, read and write operations are disabled. As with any asynchronous RESET signal, care must be taken not to assert it too close to the edges of active read and write clocks.

WD

This is the input data bus and is 18 bits wide. Not all 18 bits are valid in all configurations. When a data width less than 18 is specified, unused higher-order signals must be grounded (Table 7-7 on page 218).

RD

This is the output data bus and is 18 bits wide. Not all 18 bits are valid in all configurations. Like the WD bus, high-order bits become unusable if the data width is less than 18. The output data on unused pins is undefined (Table 7-7).

***Table 7-7* • Input Data Signal Usage for Different Aspect Ratios**

D×W	WD/RD Unused
4k×1	WD[17:1], RD[17:1]
2k×2	WD[17:2], RD[17:2]
1k×4	WD[17:4], RD[17:4]
512×9	WD[17:9], RD[17:9]
256×18	–

ESTOP, FSTOP

ESTOP is used to stop the FIFO read counter from further counting once the FIFO is empty (i.e., the EMPTY flag goes HIGH). A HIGH on this signal inhibits the counting.

FSTOP is used to stop the FIFO write counter from further counting once the FIFO is full (i.e., the FULL flag goes HIGH). A HIGH on this signal inhibits the counting.

For more information on these signals, refer to the "ESTOP and FSTOP Usage" section.

FULL, EMPTY

When the FIFO is full and no more data can be written, the FULL flag asserts HIGH. The FULL flag is synchronous to WCLK to inhibit writing immediately upon detection of a full condition and to prevent overflows. Since the write address is compared to a resynchronized (and thus time-delayed) version of the read address, the FULL flag will remain asserted until two WCLK active edges after a read operation eliminates the full condition.

When the FIFO is empty and no more data can be read, the EMPTY flag asserts HIGH. The EMPTY flag is synchronous to RCLK to inhibit reading immediately upon detection of an empty condition and to prevent underflows. Since the read address is compared to a resynchronized (and thus time-delayed) version of the write address, the EMPTY flag will remain asserted until two RCLK active edges after a write operation removes the empty condition.

For more information on these signals, refer to the "FIFO Flag Usage Considerations" section on page 219.

AFULL, AEMPTY

These are programmable flags and will be asserted on the threshold specified by AFVAL and AEVAL, respectively.

When the number of words stored in the FIFO reaches the amount specified by AEVAL while reading, the AEMPTY output will go HIGH. Likewise, when the number of words stored in the FIFO reaches the amount specified by AFVAL while writing, the AFULL output will go HIGH.

AFVAL, AEVAL

The AEVAL and AFVAL pins are used to specify the almost-empty and almost-full threshold values. They are 12-bit signals. For more information on these signals, refer to the "FIFO Flag Usage Considerations" section on page 219.

FIFO Usage

ESTOP and FSTOP Usage

The ESTOP pin is used to stop the read counter from counting any further once the FIFO is empty (i.e., the EMPTY flag goes HIGH). Likewise, the FSTOP pin is used to stop the write counter from counting any further once the FIFO is full (i.e., the FULL flag goes HIGH).

The FIFO counters in the device start the count at zero, reach the maximum depth for the configuration (e.g., 511 for a 512×9 configuration), and then restart at zero. An example application for ESTOP, where the read counter keeps counting, would be writing to the FIFO once and reading the same content over and over without doing another write.

FIFO Flag Usage Considerations

The AEVAL and AFVAL pins are used to specify the 12-bit AEMPTY and AFULL threshold values. The FIFO contains separate 12-bit write address (WADDR) and read address (RADDR) counters. WADDR is incremented every time a write operation is performed, and RADDR is incremented every time a read operation is performed. Whenever the difference between WADDR and RADDR is greater than or equal to AFVAL, the AFULL output is asserted. Likewise, whenever the difference between WADDR and RADDR is less than or equal to AEVAL, the AEMPTY output is asserted. To handle different read and write aspect ratios, AFVAL and AEVAL are expressed in terms of total data bits instead of total data words. When users specify AFVAL and AEVAL in terms of read or write words, the SmartGen tool translates them into bit addresses and configures these signals automatically. SmartGen configures the AFULL flag to assert when the write address exceeds the read address by at least a predefined value. In a 2k×8 FIFO, for example, a value of 1,500 for AFVAL means that the AFULL flag will be asserted after a write when the difference between the write address and the read address reaches 1,500 (there have been at least 1,500 more writes than reads). It will stay asserted until the difference between the write and read addresses drops below 1,500.

The AEMPTY flag is asserted when the difference between the write address and the read address is less than a predefined value. In the example above, a value of 200 for AEVAL means that the AEMPTY flag will be asserted when a read causes the difference between the write address and the read address to drop to 200. It will stay asserted until that difference rises above 200. Note that the FIFO can be configured with different read and write widths; in this case, the AFVAL setting is based on the number of write data entries, and the AEVAL setting is based on the number of read data entries. For aspect ratios of 512×9 and 256×18, only 4,096 bits can be addressed by the 12 bits of AFVAL and AEVAL. The number of words must be multiplied by 8 and 16 instead of 9 and 18. The SmartGen tool automatically uses the proper values. To avoid halfwords being written or read, which could happen if different read and write aspect ratios were specified, the FIFO will assert FULL or EMPTY as soon as at least one word cannot be written or read. For example, if a two-bit word is written and a four-bit word is being read, the FIFO will remain in the empty state when the first word is written. This occurs even if the FIFO is not completely empty, because in this case, a complete word cannot be read. The same is applicable in the full state. If a four-bit word is written and a two-bit word is read, the FIFO is full and one word is read. The FULL flag will remain asserted because a complete word cannot be written at this point.

Variable Aspect Ratio and Cascading

Variable aspect ratio and cascading allow users to configure the memory in the width and depth required. The memory block can be configured as a FIFO by combining the basic memory block with dedicated FIFO controller logic. The FIFO macro is named FIFO4KX18. Low power flash device RAM can be configured as 1, 2, 4, 9, or 18 bits wide. By cascading the memory blocks, any multiple of those widths can be created. The RAM blocks can be from 256 to 4,096 bits deep, depending on the aspect ratio, and the blocks can also be cascaded to create deeper areas. Refer to the aspect ratios available for each macro cell in the "SRAM Features" section on page 211. The largest continuous configurable memory area is equal to half the total memory available on the device, because the RAM is separated into two groups, one on each side of the device.

The SmartGen core generator will automatically configure and cascade both RAM and FIFO blocks. Cascading is accomplished using dedicated memory logic and does not consume user gates for depths up to 4,096 bits deep and widths up to 18, depending on the configuration. Deeper memory will utilize some user gates to multiplex the outputs.

Generated RAM and FIFO macros can be created as either structural VHDL or Verilog for easy instantiation into the design. Users of Libero SoC can create a symbol for the macro and incorporate it into a design schematic.

Table 7-10 on page 221 shows the number of memory blocks required for each of the supported depth and width memory configurations, and for each depth and width combination. For example, a 256-bit deep by 32-bit wide two-port RAM would consist of two 256×18 RAM blocks. The first 18 bits would be stored in the first RAM block, and the remaining 14 bits would be implemented in the other 256×18 RAM block. This second RAM block would have four bits of unused storage. Similarly, a dual-port memory block that is 8,192 bits deep and 8 bits wide would be implemented using 16 memory blocks. The dual-port memory would be configured in a 4,096×1 aspect ratio. These blocks would then be cascaded two deep to achieve 8,192 bits of depth, and eight wide to achieve the eight bits of width.

Table 7-8 and Table 7-9 show the maximum potential width and depth configuration for each device. Note that 15 k and 30 k gate devices do not support RAM or FIFO.

***Table 7-8* • Memory Availability per IGLOO and ProASIC3 Device**

Device		RAM Blocks	Maximum Potential Width[1]		Maximum Potential Depth[2]	
IGLOO IGLOO nano IGLOO PLUS	ProASIC3 ProASIC3 nano ProASIC3L		Depth	Width	Depth	Width
AGL060 AGLN060 AGLP060	A3P060 A3PN060	4	256	72 (4×18)	16,384 (4,096×4)	1
AGL125 AGLN125 AGLP125	A3P125 A3PN125	8	256	144 (8×18)	32,768 (4,094×8)	1
AGL250 AGLN250	A3P250/L A3PN250	8	256	144 (8×18)	32,768 (4,096×8)	1
AGL400	A3P400	12	256	216 (12×18)	49,152 (4,096×12)	1
AGL600	A3P600/L	24	256	432 (24×18)	98,304 (4,096×24)	1
AGL1000	A3P1000/L	32	256	576 (32×18)	131,072 (4,096×32)	1
AGLE600	A3PE600	24	256	432 (24×18)	98,304 (4,096×24)	1
	A3PE1500	60	256	1,080 (60×18)	245,760 (4,096×60)	1
AGLE3000	A3PE3000/L	112	256	2,016 (112×18)	458,752 (4,096×112)	1

Notes:

1. *Maximum potential width uses the two-port configuration.*
2. *Maximum potential depth uses the dual-port configuration.*

***Table 7-9* • Memory Availability per Fusion Device**

Device	RAM Blocks	Maximum Potential Width[1]		Maximum Potential Depth[2]	
		Depth	Width	Depth	Width
AFS090	6	256	108 (6×18)	24,576 (4,094×6)	1
AFS250	8	256	144 (8×18)	32,768 (4,094×8)	1
AFS600	24	256	432 (24×18)	98,304 (4,096×24)	1
AFS1500	60	256	1,080 (60×18)	245,760 (4,096×60)	1

Notes:

1. *Maximum potential width uses the two-port configuration.*
2. *Maximum potential depth uses the dual-port configuration.*

Table 7-10 • **RAM and FIFO Memory Block Consumption**

			Depth									
			256		512	1,024	2,048	4,096	8,192	16,384	32,768	65,536
			Two-Port	Dual-Port	Dual-Port	Dual-Port	Dual-Port	Dual-Port	Dual-Port	Dual-Port	Dual-Port	Dual-Port
Width	1	Number Block	1	1	1	1	1	1	2	4	8	16 × 1
		Configuration	Any	Any	Any	1,024 × 4	2,048 × 2	4,096 × 1	2 × (4,096 × 1) Cascade Deep	4 × (4,096 × 1) Cascade Deep	8 × (4,096 × 1) Cascade Deep	16 × (4,096 × 1) Cascade Deep
	2	Number Block	1	1	1	1	1	2	4	8	16	32
		Configuration	Any	Any	Any	1,024×4	2,048 × 2	2 × (4,096 × 1) Cascaded Wide	4 × (4,096 × 1) Cascaded 2 Deep and 2 Wide	8 × (4,096 × 1) Cascaded 4 Deep and 2 Wide	16 × (4,096 × 1) Cascaded 8 Deep and 2 Wide	32 × (4,096 × 1) Cascaded 16 Deep and 2 Wide
	4	Number Block	1	1	1	1	2	4	8	16	32	64
		Configuration	Any	Any	Any	1,024 × 4	2 × (2,048 × 2) Cascaded Wide	4 × (4,096 × 1) Cascaded Wide	4 × (4,096 × 1) Cascaded 2 Deep and 4 Wide	16 × (4,096 × 1) Cascaded 4 Deep and 4 Wide	32 × (4,096 × 1) Cascaded 8 Deep and 4 Wide	64 × (4,096 × 1) Cascaded 16 Deep and 4 Wide
	8	Number Block	1	1	1	2	4	8	16	32	64	
		Configuration	Any	Any	Any	2 × (1,024 × 4) Cascaded Wide	4 × (2,048 × 2) Cascaded Wide	8 × (4,096 × 1) Cascaded Wide	16 × (4,096 × 1) Cascaded 2 Deep and 8 Wide	32 × (4,096 × 1) Cascaded 4 Deep and 8 Wide	64 × (4,096 × 1) Cascaded 8 Deep and 8 Wide	
	9	Number Block	1	1	1	2	4	8	16	32		
		Configuration	Any	Any	Any	2 × (512 × 9) Cascaded Deep	4 × (512 × 9) Cascaded Deep	8 × (512 × 9) Cascaded Deep	16 × (512 × 9) Cascaded Deep	32 × (512 × 9) Cascaded Deep		
	16	Number Block	1	1	1	4	8	16	32	64		
		Configuration	256 × 18	256 × 18	256 × 18	4 × (1,024 × 4) Cascaded Wide	8 × (2,048 × 2) Cascaded Wide	16 × (4,096 × 1) Cascaded Wide	32 × (4,096 × 1) Cascaded 2 Deep and 16 Wide	32 × (4,096 × 1) Cascaded 4 Deep and 16 Wide		
	18	Number Block	1	2	2	4	8	18	32			
		Configuration	256 × 8	2 × (512 × 9) Cascaded Wide	2 × (512 × 9) Cascaded Wide	4 × (512 × 9) Cascaded 2 Deep and 2 Wide	8 × (512 × 9) Cascaded 4 Deep and 2 Wide	16 × (512 × 9) Cascaded 8 Deep and 2 Wide	16 × (512 × 9) Cascaded 16 Deep and 2 Wide			
	32	Number Block	2	4	4	8	16	32	64			
		Configuration	2 × (256 × 18) Cascaded Wide	4 × (512 × 9) Cascaded Wide	4 × (512 × 9) Cascaded Wide	8 × (1,024 × 4) Cascaded Wide	16 × (2,048 × 2) Cascaded Wide	32 × (4,096 × 1) Cascaded Wide	64 × (4,096 × 1) Cascaded 2 Deep and 32 Wide			
	36	Number Block	2	4	4	8	16	32				
		Configuration	2 × (256 × 18) Cascaded Wide	4 × (512 × 9) Cascaded Wide	4 × (512 × 9) Cascaded Wide	4 × (512 × 9) Cascaded 2 Deep and 4 Wide	16 × (512 × 9) Cascaded 4 Deep and 4 Wide	16 × (512 × 9) Cascaded 8 Deep and 4 Wide				
	64	Number Block	4	8	8	16	32	64				
		Configuration	4 × (256 × 18) Cascaded Wide	8 × (512 × 9) Cascaded Wide	8 × (512 × 9) Cascaded Wide	16 × (1,024 × 4) Cascaded Wide	32 × (2,048 × 2) Cascaded Wide	64 × (4,096 × 1) Cascaded Wide				
	72	Number Block	4	8	8	16	32					
		Configuration	4 × (256 × 18) Cascaded Wide	8 × (512 × 9) Cascaded Wide	8 × (512 × 9) Cascaded Wide	16 × (512 × 9) Cascaded Wide	16 × (512 × 9) Cascaded 4 Deep and 8 Wide					

Note: Memory configurations represented by grayed cells are not supported.

Initializing the RAM/FIFO

The SRAM blocks can be initialized with data to use as a lookup table (LUT). Data initialization can be accomplished either by loading the data through the design logic or through the UJTAG interface. The UJTAG macro is used to allow access from the JTAG port to the internal logic in the device. By sending the appropriate initialization string to the JTAG Test Access Port (TAP) Controller, the designer can put the JTAG circuitry into a mode that allows the user to shift data into the array logic through the JTAG port using the UJTAG macro. For a more detailed explanation of the UJTAG macro, refer to the "FlashROM in Microsemi's Low Power Flash Devices" section on page 191.

A user interface is required to receive the user command, initialization data, and clock from the UJTAG macro. The interface must synchronize and load the data into the correct RAM block of the design. The main outputs of the user interface block are the following:

- Memory block chip select: Selects a memory block for initialization. The chip selects signals for each memory block that can be generated from different user-defined pockets or simple logic, such as a ring counter (see below).
- Memory block write address: Identifies the address of the memory cell that needs to be initialized.
- Memory block write data: The interface block receives the data serially from the UTDI port of the UJTAG macro and loads it in parallel into the write data ports of the memory blocks.
- Memory block write clock: Drives the WCLK of the memory block and synchronizes the write data, write address, and chip select signals.

Figure 7-8 shows the user interface between UJTAG and the memory blocks.

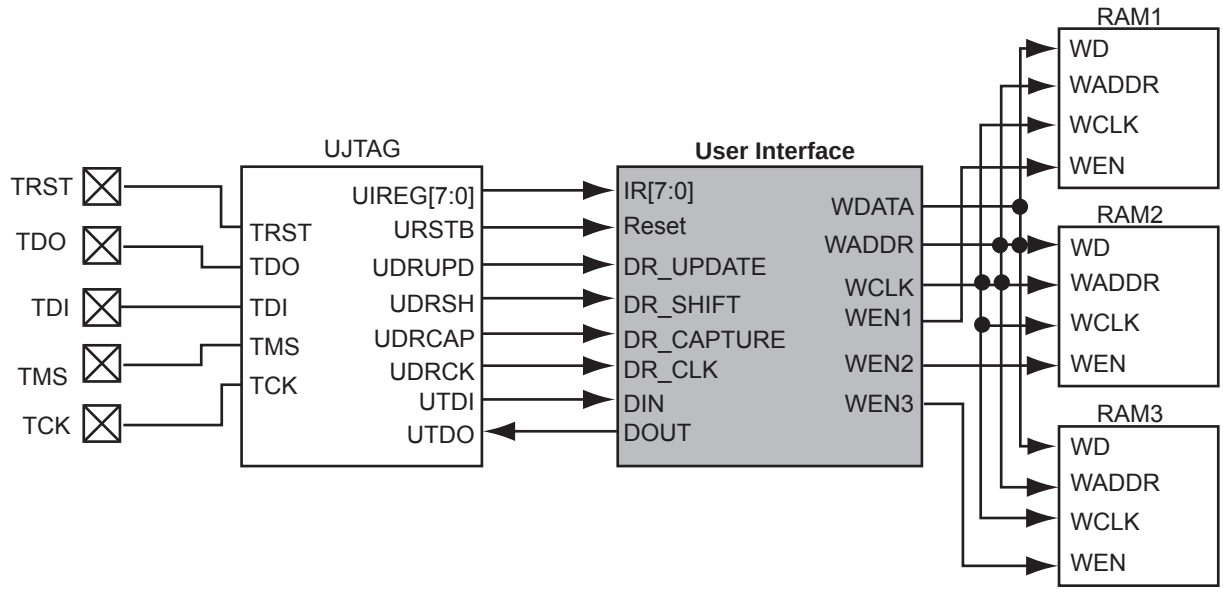


***Figure 7-8* • Interfacing TAP Ports and SRAM Blocks**

An important component of the interface between the UJTAG macro and the RAM blocks is a serial-in/parallel-out shift register. The width of the shift register should equal the data width of the RAM blocks. The RAM data arrives serially from the UTDI output of the UJTAG macro. The data must be shifted into a shift register clocked by the JTAG clock (provided at the UDRCK output of the UJTAG macro).

Then, after the shift register is fully loaded, the data must be transferred to the write data port of the RAM block. To synchronize the loading of the write data with the write address and write clock, the output of the shift register can be pipelined before driving the RAM block.

The write address can be generated in different ways. It can be imported through the TAP using a different instruction opcode and another shift register, or generated internally using a simple counter. Using a counter to generate the address bits and sweep through the address range of the RAM blocks is

recommended, since it reduces the complexity of the user interface block and the board-level JTAG driver.

Moreover, using an internal counter for address generation speeds up the initialization procedure, since the user only needs to import the data through the JTAG port.

The designer may use different methods to select among the multiple RAM blocks. Using counters along with demultiplexers is one approach to set the write enable signals. Basically, the number of RAM blocks needing initialization determines the most efficient approach. For example, if all the blocks are initialized with the same data, one enable signal is enough to activate the write procedure for all of them at the same time. Another alternative is to use different opcodes to initialize each memory block. For a small number of RAM blocks, using counters is an optimal choice. For example, a ring counter can be used to select from multiple RAM blocks. The clock driver of this counter needs to be controlled by the address generation process.

Once the addressing of one block is finished, a clock pulse is sent to the (ring) counter to select the next memory block.

Figure 7-9 illustrates a simple block diagram of an interface block between UJTAG and RAM blocks.

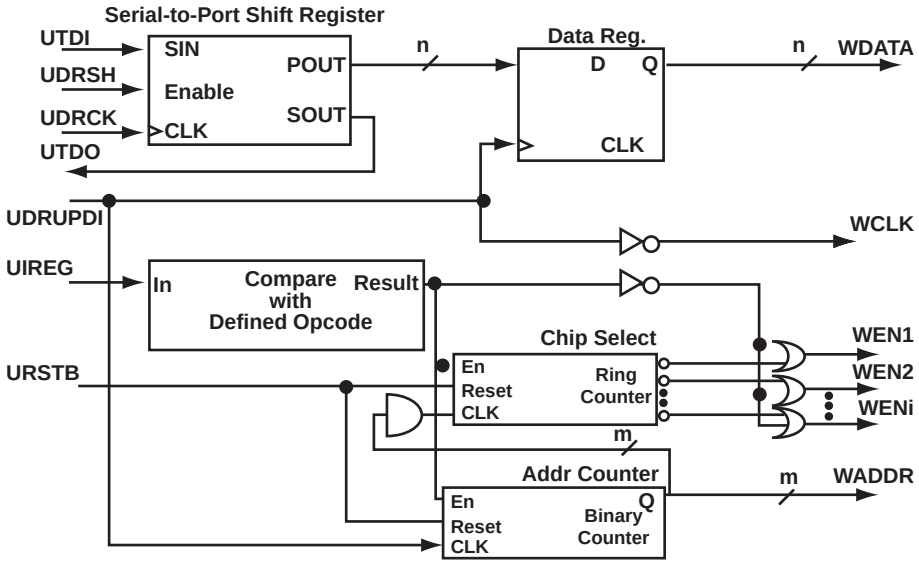


Figure 7-9 • **Block Diagram of a Sample User Interface**

In the circuit shown in Figure 7-9, the shift register is enabled by the UDRSH output of the UJTAG macro. The counters and chip select outputs are controlled by the value of the TAP Instruction Register. The comparison block compares the UIREG value with the "start initialization" opcode value (defined by the user). If the result is true, the counters start to generate addresses and activate the WEN inputs of appropriate RAM blocks.

The UDRUPD output of the UJTAG macro, also shown in Figure 7-9, is used for generating the write clock (WCLK) and synchronizing the data register and address counter with WCLK. UDRUPD is HIGH when the TAP Controller is in the Data Register Update state, which is an indication of completing the loading of one data word. Once the TAP Controller goes into the Data Register Update state, the UDRUPD output of the UJTAG macro goes HIGH. Therefore, the pipeline register and the address counter place the proper data and address on the outputs of the interface block. Meanwhile, WCLK is defined as the inverted UDRUPD. This will provide enough time (equal to the UDRUPD HIGH time) for the data and address to be placed at the proper ports of the RAM block before the rising edge of WCLK. The inverter is not required if the RAM blocks are clocked at the falling edge of the write clock. An example of this is described in the "Example of RAM Initialization" section on page 224.

Example of RAM Initialization

This section of the document presents a sample design in which a 4×4 RAM block is being initialized through the JTAG port. A test feature has been implemented in the design to read back the contents of the RAM after initialization to verify the procedure.

The interface block of this example performs two major functions: initialization of the RAM block and running a test procedure to read back the contents. The clock output of the interface is either the write clock (for initialization) or the read clock (for reading back the contents). The Verilog code for the interface block is included in the "Sample Verilog Code" section on page 225.

For simulation purposes, users can declare the input ports of the UJTAG macro for easier assignment in the testbench. However, the UJTAG input ports should not be declared on the top level during synthesis. If the input ports of the UJTAG are declared during synthesis, the synthesis tool will instantiate input buffers on these ports. The input buffers on the ports will cause Compile to fail in Designer.

Figure 7-10 shows the simulation results for the initialization step of the example design.

The CLK_OUT signal, which is the clock output of the interface block, is the inverted DR_UPDATE output of the UJTAG macro. It is clear that it gives sufficient time (while the TAP Controller is in the Data Register Update state) for the write address and data to become stable before loading them into the RAM block.

Figure 7-11 presents the test procedure of the example. The data read back from the memory block matches the written data, thus verifying the design functionality.

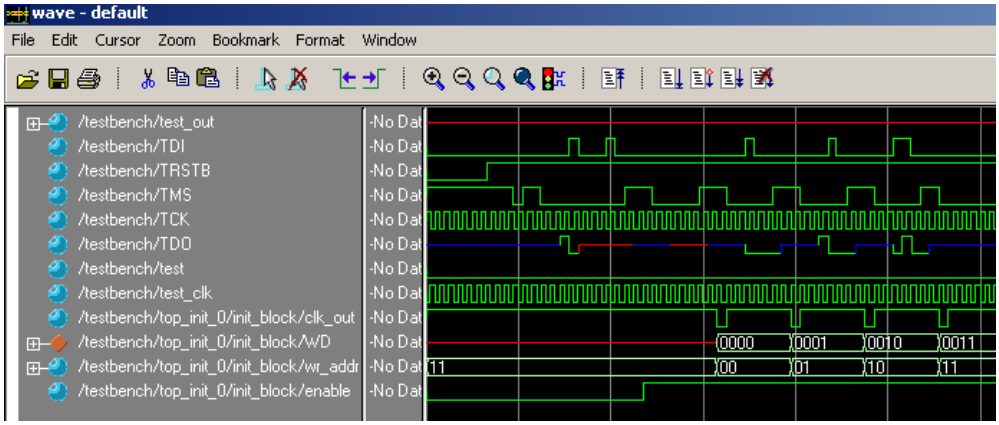


***Figure 7-10* • Simulation of Initialization Step**

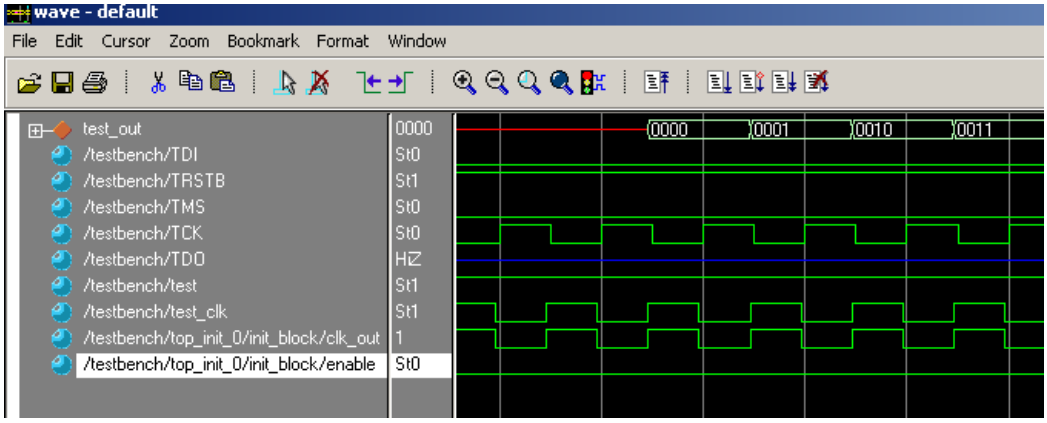


***Figure 7-11* • Simulation of the Test Procedure of the Example**

The ROM emulation application is based on RAM block initialization. If the user's main design has access only to the read ports of the RAM block (RADDR, RD, RCLK, and REN), and the contents of the RAM are already initialized through the TAP, then the memory blocks will emulate ROM functionality for the core design. In this case, the write ports of the RAM blocks are accessed only by the user interface block, and the interface is activated only by the TAP Instruction Register contents.

Users should note that the contents of the RAM blocks are lost in the absence of applied power. However, the 1 kbit of flash memory, FlashROM, in low power flash devices can be used to retain data after power is removed from the device. Refer to the "SRAM and FIFO Memories in Microsemi's Low Power Flash Devices" section on page 205 for more information.

Sample Verilog Code

Interface Block

```
`define Initialize_start 8'h22 //INITIALIZATION START COMMAND VALUE
`define Initialize_stop 8'h23 //INITIALIZATION START COMMAND VALUE

module interface(IR, rst_n, data_shift, clk_in, data_update, din_ser, dout_ser, test,
  test_out,test_clk,clk_out,wr_en,rd_en,write_word,read_word,rd_addr, wr_addr);

input [7:0] IR;
input [3:0] read_word; //RAM DATA READ BACK
input rst_n, data_shift, clk_in, data_update, din_ser; //INITIALIZATION SIGNALS
input test, test_clk; //TEST PROCEDURE CLOCK AND COMMAND INPUT
output [3:0] test_out; //READ DATA
output [3:0] write_word; //WRITE DATA
output [1:0] rd_addr; //READ ADDRESS
output [1:0] wr_addr; //WRITE ADDRESS
output dout_ser; //TDO DRIVER
output clk_out, wr_en, rd_en;

wire [3:0] write_word;
wire [1:0] rd_addr;
wire [1:0] wr_addr;
wire [3:0] Q_out;
wire enable, test_active;

reg clk_out;

//SELECT CLOCK FOR INITIALIZATION OR READBACK TEST
always @(enable or test_clk or data_update)
begin
  case ({test_active})
    1 : clk_out = test_clk ;
    0 : clk_out = !data_update;
    default : clk_out = 1'b1;
  endcase
end

assign test_active = test && (IR == 8'h23);
assign enable = (IR == 8'h22);
assign wr_en = !enable;
assign rd_en = !test_active;
assign test_out = read_word;
assign dout_ser = Q_out[3];

//4-bit SIN/POUT SHIFT REGISTER
shift_reg data_shift_reg (.Shiften(data_shift), .Shiftin(din_ser), .Clock(clk_in),
  .Q(Q_out));

//4-bit PIPELINE REGISTER
D_pipeline pipeline_reg (.Data(Q_out), .Clock(data_update), .Q(write_word));
```

```
//
addr_counter counter_1 (.Clock(data_update), .Q(wr_addr), .Aset(rst_n),
  .Enable(enable));
addr_counter counter_2 (.Clock(test_clk), .Q(rd_addr), .Aset(rst_n),
  .Enable( test_active));

endmodule
```

Interface Block / UJTAG Wrapper

This example is a sample wrapper, which connects the interface block to the UJTAG and the memory blocks.

```
// WRAPPER
module top_init (TDI, TRSTB, TMS, TCK, TDO, test, test_clk, test_ out);

input TDI, TRSTB, TMS, TCK;
output TDO;
input test, test_clk;
output [3:0] test_out;

wire [7:0] IR;
wire reset, DR_shift, DR_cap, init_clk, DR_update, data_in, data_out;
wire clk_out, wen, ren;
wire [3:0] word_in, word_out;
wire [1:0] write_addr, read_addr;

UJTAG UJTAG_U1 (.UIREG0(IR[0]), .UIREG1(IR[1]), .UIREG2(IR[2]), .UIREG3(IR[3]),
  .UIREG4(IR[4]), .UIREG5(IR[5]), .UIREG6(IR[6]), .UIREG7(IR[7]), .URSTB(reset),
  .UDRSH(DR_shift), .UDRCAP(DR_cap), .UDRCK(init_clk), .UDRUPD(DR_update),
  .UT-DI(data_in), .TDI(TDI), .TMS(TMS), .TCK(TCK), .TRSTB(TRSTB), .TDO(TDO),
  .UT-DO(data_out));
mem_block RAM_block (.DO(word_out), .RCLOCK(clk_out), .WCLOCK(clk_out), .DI(word_in),
  .WRB(wen), .RDB(ren), .WAD-DR(write_addr), .RADDR(read_addr));
interface init_block (.IR(IR), .rst_n(reset), .data_shift(DR_shift), .clk_in(init_clk),
  .data_update(DR_update), .din_ser(data_in), .dout_ser(data_out), .test(test),
  .test_out(test_out), .test_clk(test_clk), .clk_out(clk_out), .wr_en(wen),
  .rd_en(ren), .write_word(word_in), .read_word(word_out), .rd_addr(read_addr),
  .wr_addr(write_addr));

endmodule
```

Address Counter

```
module addr_counter (Clock, Q, Aset, Enable);

input Clock;
output [1:0] Q;
input Aset;
input Enable;

reg [1:0] Qaux;

always @(posedge Clock or negedge Aset)
begin
  if (!Aset) Qaux <= 2'b11;
  else if (Enable) Qaux <= Qaux + 1;
end

assign Q = Qaux;

endmodule
```

Pipeline Register

```
module D_pipeline (Data, Clock, Q);

input [3:0] Data;
input Clock;
output [3:0] Q;

reg [3:0] Q;

always @ (posedge Clock) Q <= Data;

endmodule
```

4x4 RAM Block (created by SmartGen Core Generator)

```
module mem_block(DI,DO,WADDR,RADDR,WRB,RDB,WCLOCK,RCLOCK);

input [3:0] DI;
output [3:0] DO;
input [1:0] WADDR, RADDR;
input WRB, RDB, WCLOCK, RCLOCK;

wire WEBP, WEAP, VCC, GND;

VCC VCC_1_net(.Y(VCC));
GND GND_1_net(.Y(GND));
INV WEBUBBLEB(.A(WRB), .Y(WEBP));
RAM4K9 RAMBLOCK0(.ADDRA11(GND), .ADDRA10(GND), .ADDRA9(GND), .ADDRA8(GND),
  .ADDRA7(GND), .ADDRA6(GND), .ADDRA5(GND), .ADDRA4(GND), .ADDRA3(GND), .ADDRA2(GND),
  .ADDRA1(RADDR[1]), .ADDRA0(RADDR[0]), .ADDRB11(GND), .ADDRB10(GND), .ADDRB9(GND),
  .ADDRB8(GND), .ADDRB7(GND), .ADDRB6(GND), .ADDRB5(GND), .ADDRB4(GND), .ADDRB3(GND),
  .ADDRB2(GND), .ADDRB1(WADDR[1]), .ADDRB0(WADDR[0]), .DINA8(GND), .DINA7(GND),
  .DINA6(GND), .DINA5(GND), .DINA4(GND), .DINA3(GND), .DINA2(GND), .DINA1(GND),
  .DINA0(GND), .DINB8(GND), .DINB7(GND), .DINB6(GND), .DINB5(GND), .DINB4(GND),
  .DINB3(DI[3]), .DINB2(DI[2]), .DINB1(DI[1]), .DINB0(DI[0]), .WIDTHA0(GND),
  .WIDTHA1(VCC), .WIDTHB0(GND), .WIDTHB1(VCC), .PIPEA(GND), .PIPEB(GND),
  .WMODEA(GND), .WMODEB(GND), .BLKA(WEAP), .BLKB(WEBP), .WENA(VCC), .WENB(GND),
  .CLKA(RCLOCK), .CLKB(WCLOCK), .RESET(VCC), .DOUTA8(), .DOUTA7(), .DOUTA6(),
  .DOUTA5(), .DOUTA4(), .DOUTA3(DO[3]), .DOUTA2(DO[2]), .DOUTA1(DO[1]),
  .DOUTA0(DO[0]), .DOUTB8(), .DOUTB7(), .DOUTB6(), .DOUTB5(), .DOUTB4(), .DOUTB3(),
  .DOUTB2(), .DOUTB1(), .DOUTB0());
INV WEBUBBLEA(.A(RDB), .Y(WEAP));

endmodule
```

Software Support

The SmartGen core generator is the easiest way to select and configure the memory blocks (Figure 7-12). SmartGen automatically selects the proper memory block type and aspect ratio, and cascades the memory blocks based on the user's selection. SmartGen also configures any additional signals that may require tie-off.

SmartGen will attempt to use the minimum number of blocks required to implement the desired memory. When cascading, SmartGen will configure the memory for width before configuring for depth. For example, if the user requests a 256×8 FIFO, SmartGen will use a 512×9 FIFO configuration, not 256×18.

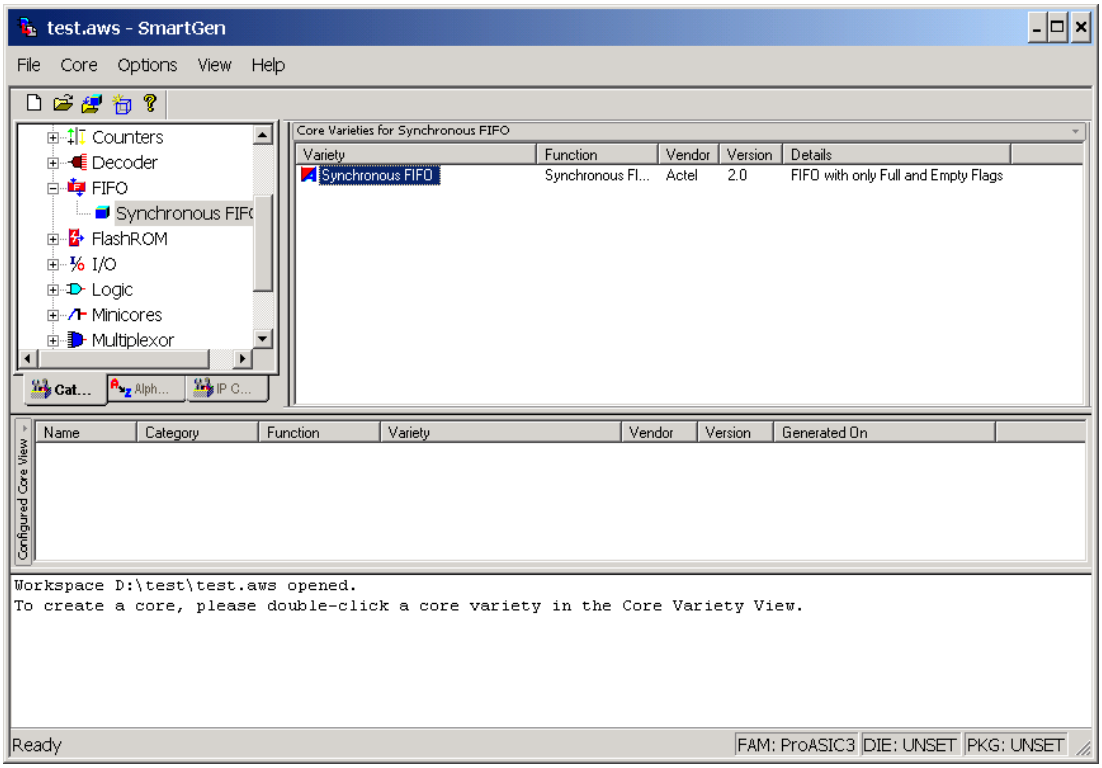


Figure 7-12 • **SmartGen Core Generator Interface**

SmartGen enables the user to configure the desired RAM element to use either a single clock for read and write, or two independent clocks for read and write. The user can select the type of RAM as well as the width/depth and several other parameters (Figure 7-13).

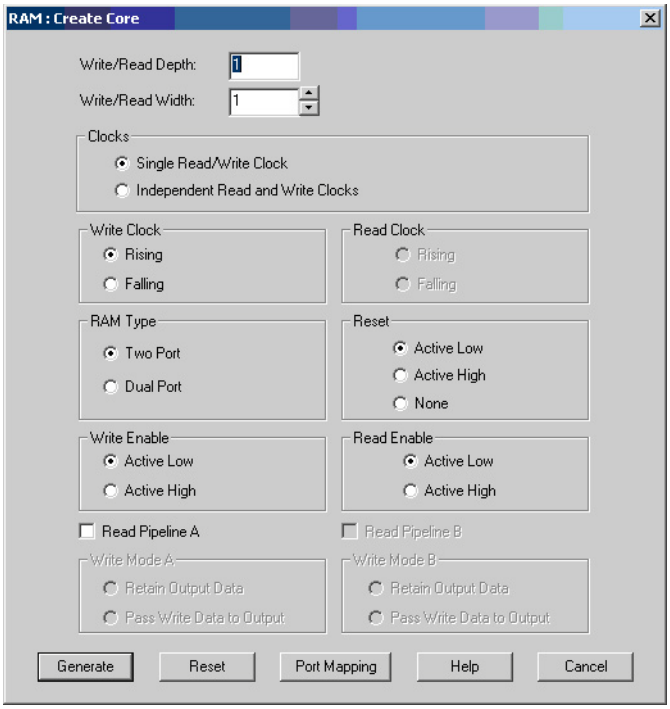


***Figure 7-13* • SmartGen Memory Configuration Interface**

SmartGen also has a Port Mapping option that allows the user to specify the names of the ports generated in the memory block (Figure 7-14).

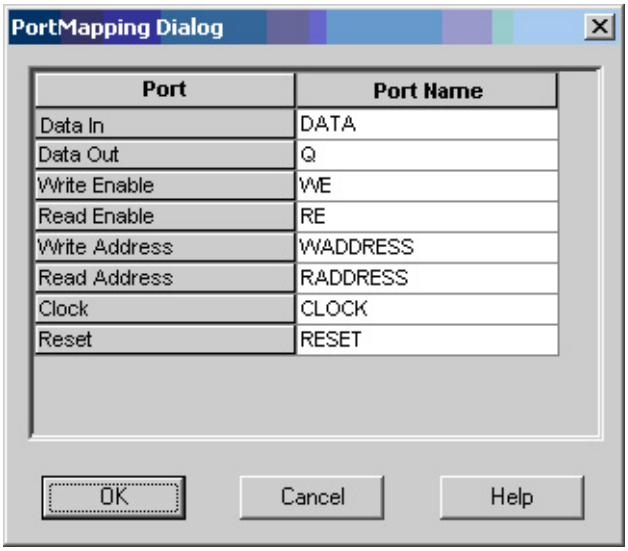


***Figure 7-14* • Port Mapping Interface for SmartGen-Generated Memory**

SmartGen also configures the FIFO according to user specifications. Users can select no flags, static flags, or dynamic flags. Static flag settings are configured using configuration flash and cannot be altered

without reprogramming the device. Dynamic flag settings are determined by register values and can be altered without reprogramming the device by reloading the register values either from the design or through the UJTAG interface described in the "Initializing the RAM/FIFO" section on page 222.

SmartGen can also configure the FIFO to continue counting after the FIFO is full. In this configuration, the FIFO write counter will wrap after the counter is full and continue to write data. With the FIFO configured to continue to read after the FIFO is empty, the read counter will also wrap and re-read data that was previously read. This mode can be used to continually read back repeating data patterns stored in the FIFO (Figure 7-15).

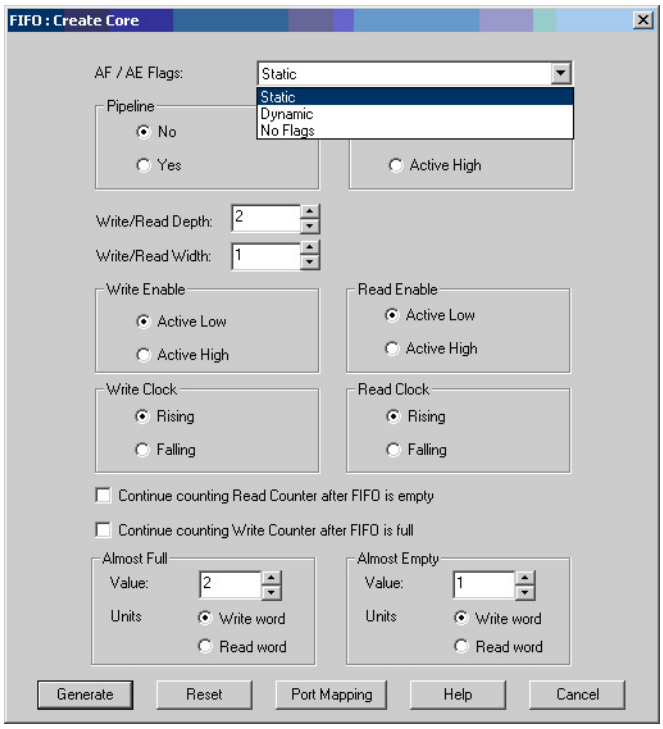


***Figure 7-15* • SmartGen FIFO Configuration Interface**

FIFOs configured using SmartGen can also make use of the port mapping feature to configure the names of the ports.

Limitations

Users should be aware of the following limitations when configuring SRAM blocks for low power flash devices:

- SmartGen does not track the target device in a family, so it cannot determine if a configured memory block will fit in the target device.
- Dual-port RAMs with different read and write aspect ratios are not supported.
- Cascaded memory blocks can only use a maximum of 64 blocks of RAM.
- The Full flag of the FIFO is sensitive to the maximum depth of the actual physical FIFO block, not the depth requested in the SmartGen interface.

Conclusion

Fusion, IGLOO, and ProASIC3 devices provide users with extremely flexible SRAM blocks for most design needs, with the ability to choose between an easy-to-use dual-port memory or a wide-word two-port memory. Used with the built-in FIFO controllers, these memory blocks also serve as highly efficient FIFOs that do not consume user gates when implemented. The SmartGen core generator provides a fast and easy way to configure these memory elements for use in designs.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
August 2012	The note connected with Figure 7-3 • Supported Basic RAM Macros, regarding RAM4K9, was revised to explain that it applies only to part numbers of certain revisions and earlier (SAR 29574).	210
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.5 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 7-1 • Flash-Based FPGAs.	208
	IGLOO nano and ProASIC3 nano devices were added to Figure 7-8 • Interfacing TAP Ports and SRAM Blocks.	222
v1.4 (October 2008)	The "SRAM/FIFO Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	208
	The "SRAM and FIFO Architecture" section was modified to remove "IGLOO and ProASIC3E" from the description of what the memory block includes, as this statement applies to all memory blocks.	209
	Wording in the "Clocking" section was revised to change "IGLOO and ProASIC3 devices support inversion" to "Low power flash devices support inversion." The reference to IGLOO and ProASIC3 development tools in the last paragraph of the section was changed to refer to development tools in general.	215
	The "ESTOP and FSTOP Usage" section was updated to refer to FIFO counters in devices in general rather than only IGLOO and ProASIC3E devices.	218
v1.3 (August 2008)	The note was removed from Figure 7-7 • RAM Block with Embedded FIFO Controller and placed in the WCLK and RCLK description.	216
	The "WCLK and RCLK" description was revised.	217
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 7-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	208
v1.1 (March 2008)	The "Introduction" section was updated to include the IGLOO PLUS family.	205
	The "Device Architecture" section was updated to state that 15 k gate devices do not support SRAM and FIFO.	205
	The first note in Figure 7-1 • IGLOO and ProASIC3 Device Architecture Overview was updated to include mention of 15 k gate devices, and IGLOO PLUS was added to the second note.	207

Date	Changes	Page
v1.1 (continued)	Table 7-1 • Flash-Based FPGAs and associated text were updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	208
	The text introducing Table 7-8 • Memory Availability per IGLOO and ProASIC3 Device was updated to replace "A3P030 and AGL030" with "15 k and 30 k gate devices." Table 7-8 • Memory Availability per IGLOO and ProASIC3 Device was updated to remove AGL400 and AGLE1500 and include IGLOO PLUS and ProASIC3L devices.	220

8 – Designing the Fusion Analog System

Introduction

Fusion devices offer robust and flexible analog mixed signal capability in addition to the high-performance flash FPGA fabric and flash memory block. The many built-in analog peripherals include a configurable 32:1 analog MUX, up to 10 independent MOSFET gate driver outputs, and a configurable analog-to-digital converter (ADC). Fusion also introduces the Analog Quad I/O structure; each Analog Quad consists of three analog inputs and one gate driver. Each Quad can be configured in various built-in circuit combinations, such as prescaler circuits, three-digital-input circuits, a current monitor circuit, or a temperature monitor circuit. Each prescaler has multiple scaling factors programmed by FPGA signals to support a large range of analog inputs with positive or negative polarity. When the current monitor circuit is selected, two adjacent analog inputs measure the voltage drop across a small external sense resistor. Built-in operational amplifiers amplify small voltage signals for accurate current measurement. One analog input in each Quad can be connected to an external temperature monitor diode. These components are used as the building blocks in designing an analog system.

The Analog Quad I/O configuration, ADC resolution, channel sampling sequence, and sampling rate are programmable and implemented in the FPGA logic using Designer and Microsemi Libero SoC software tool support. An overview of different design methodologies is covered in the "Fusion Design Solutions and Methodologies" section on page 247. This chapter gives a detailed description of the Analog Quads, ADC, and Analog Configuration MUX (ACM). It also covers the details of the analog system with the explanation and sample calculations of accuracy, sample rate, sample sequencing, acquisition time, ADC clocking, and prescaler selection.

Analog-to-Digital Converter Background

An analog-to-digital converter is used to capture discrete samples of a continuous analog voltage and provide a discrete binary representation of the signal.

Analog-to-digital converters are generally characterized in three ways:

- Input voltage range
- Resolution
- Bandwidth or conversion rate

The input voltage range of an ADC is determined by its reference voltage (VREF). Fusion devices include an internal 2.56 V reference, or the user can supply an external reference of up to 3.3 V. The following examples use the internal 2.56 V reference, so the full-scale input range of the ADC is 0 to 2.56 V. For input signal ranges less than or greater than V_{REF}, an analog scaling function such as that built into the Fusion Analog Quad Prescaler can be used to amplify or attenuate the input signal, thus matching the input voltage range of the ADC.

The resolution (LSB) of the ADC is a function of the number of binary bits in the converter. The ADC approximates the value of the input voltage using 2^n “steps,” where *n* is the number of bits in the converter. Each step therefore represents VREF / 2^n volts. In the case of the Fusion ADC configured for 12-bit operation, the LSB is 2.56 V / 4096 = 0.625 mV.

Finally, bandwidth is an indication of the maximum number of conversions the ADC can perform each second. The bandwidth of an ADC is constrained by its architecture and several key performance characteristics. In addition, the bandwidth is limited by Fusion system considerations. See the "Sample Rate and Sample Sequence Calculation" section on page 240.

There are several popular ADC architectures, each with its own advantages and limitations. The analog-to-digital converter in Fusion devices is a switched-capacitor Successive Approximation Register (SAR) ADC. It supports 8-, 10-, and 12-bit modes of operation with a cumulative sample rate up to 600 k samples per second (ksps). Built-in bandgap circuitry offers 1% internal voltage reference accuracy, or an external reference voltage can be used.

As shown in Figure 8-1, a SAR ADC contains N capacitors with binary-weighted values.

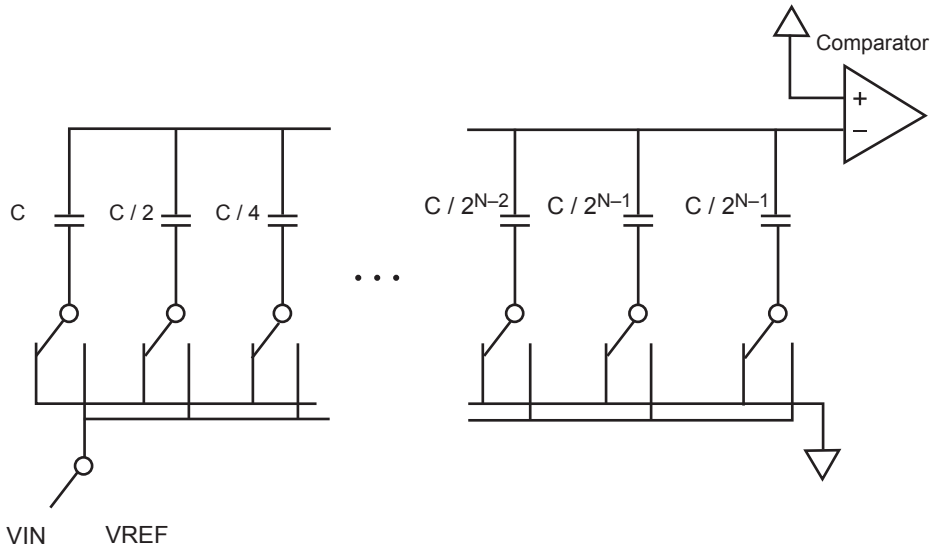


***Figure 8-1* • Example SAR ADC Architecture**

To begin a conversion, all of the capacitors are quickly discharged. Then V_{IN} is applied to all the capacitors for a period of time (acquisition time) during which the capacitors are charged to a value very close to VIN. Then all of the capacitors are switched to ground, and thus $–V_{IN}$ is applied across the comparator.

Now the conversion process begins. First, C is switched to V_{REF}. Because of the binary weighting of the capacitors, the voltage at the input of the comparator is then –VIN + VREF / 2. If V_{IN} is greater than VREF / 2, the output of the comparator is 1; otherwise, the comparator output is 0. A register is clocked to retain this value as the MSB of the result.

Next, if the MSB is 0, C is switched back to ground; otherwise, it remains connected to V_{REF}, and C / 2 is connected to VREF. The result at the comparator input is now either –VIN + VREF / 4 or –VIN + 3 VREF / 4 (depending on the state of the MSB), and the comparator output now indicates the value of the next most significant bit. This bit is likewise registered, and the process continues for each subsequent bit until a conversion is completed. The conversion process requires some acquisition time plus N + 1 ADC clock cycles to complete.

This process results in a binary approximation of V_{IN}. Generally, there is a fixed interval T, the sampling period, between the samples. The inverse of the sampling period is often referred to as the sampling frequency f_s = 1 / T. The combined effect is illustrated in Figure 8-2.

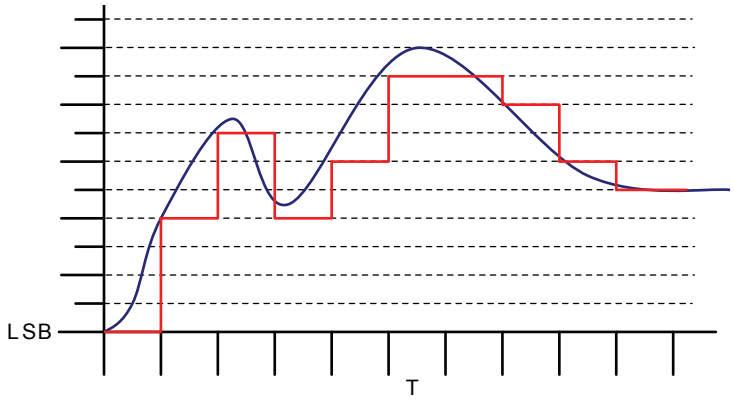


***Figure 8-2* • Analog-to-Digital Conversion Example**

Figure 8-2 demonstrates that if the signal changes faster than the sampling rate can accommodate, or if the actual value of V_{IN} falls between "counts" in the result, this information is lost during the conversion. There are several techniques that can be used to address these issues.

First, the sampling rate must be chosen to provide enough samples to adequately represent the input signal. Based on the Nyquist-Shannon Sampling Theorem, the minimum sampling rate must be at least twice the frequency of the highest frequency component in the target signal (Nyquist Frequency). For example, to re-create the frequency content of an audio signal with up to 22 kHz bandwidth, the user must sample it at a minimum of 44 ksps. However, as shown in Figure 8-2, significant post-processing of the data is required to interpolate the value of the waveform during the time between each sample.

Similarly, to re-create the amplitude variation of a signal, the signal must be sampled with adequate resolution. Continuing with the audio example, the dynamic range of the human ear (the ratio of the amplitude of the threshold of hearing to the threshold of pain) is generally accepted to be 135 dB, and the dynamic range of a typical symphony orchestra performance is around 85 dB. Most commercial recording media provide about 96 dB of dynamic range using 16-bit sample resolution. But 16-bit fidelity does not necessarily mean that you need a 16-bit ADC. As long as the input is sampled at or above the Nyquist Frequency, post-processing techniques can be used to interpolate intermediate values and reconstruct the original input signal to within desired tolerances.

If sophisticated digital signal processing (DSP) capabilities are available, the best results are obtained by implementing a reconstruction filter, which is used to interpolate many intermediate values with higher resolution than the original data. Interpolating many intermediate values, increases the effective number of samples, and higher resolution increases the effective number of bits in the sample. In many cases, however, it is not cost-effective or necessary to implement such a sophisticated reconstruction algorithm. For applications that do not require extremely fine reproduction of the input signal, alternative methods can enhance digital sampling results with relatively simple post-processing. The details of such techniques are out of the scope of this chapter; refer to the *Improving ADC Results Through Oversampling and Post-Processing of Data* white paper for more information.

ADC Clock

When the Fusion ADC is used, the ADC clock determines the sampling throughput, and the system clock determines the operating speed of the SmartGen IP and/or the user's design logic. This section examines the relationship between these two clocks and how the sampling rate is related to the accuracy. A simplified block diagram of the ADC is given in Figure 8-3.

The ADC clock has a maximum frequency of 10 MHz and can be derived from the system clock. To generate the ADC clock, the system clock is divided by a multiple of four. The exact multiple of four used is determined by the 8-bit user-configurable TVC[7:0] register (EQ 1).

ADC Clock Frequency (MHz) = System Clock (MHz) / (4 × (TVC_reg + 1))

EQ 1

where TVC_reg is the TVC register value, from 0 to 255.

The TVC register setting is used to ensure that the ADC clock frequency does not exceed 10 MHz. Note that the 10 MHz maximum frequency for the ADC clock implies that a higher system clock frequency does not always result in a higher ADC clock frequency. For example, a 40 MHz system clock frequency enables a maximum ADC clock frequency of 10 MHz (TVC register value of 0), whereas a 50 MHz system frequency results in a slower maximum ADC clock frequency, 6.25 MHz, because a TVC register value of 0 would give an ADC clock frequency of 12.5 MHz—above the 10 MHz limit. Setting the TVC register value to 1 in this case gives a maximum ADC clock frequency of 6.25 MHz.

In general, the performance of the ADC is the rate at which the ADC can acquire or sample the analog input and convert it into a digital value. Datasheet specifications define this in terms of samples per second (S/sec) or hertz (Hz). The inverse of the conversion time is the sampling rate for the channel. However, the sampling rate reported by SmartGen includes the ADC Sample Sequence Controller (ASSC) overhead time. This time, the turnaround time, defines how fast an ADC client can process data and give another start conversion signal. With no wait states, the ASSC turnaround time is 10 system clock cycles.

$$\text{Sample Rate} = \frac{1}{\text{Conversion Time} + \text{Turnaround Time}} (\text{S/sec})$$

EQ 2

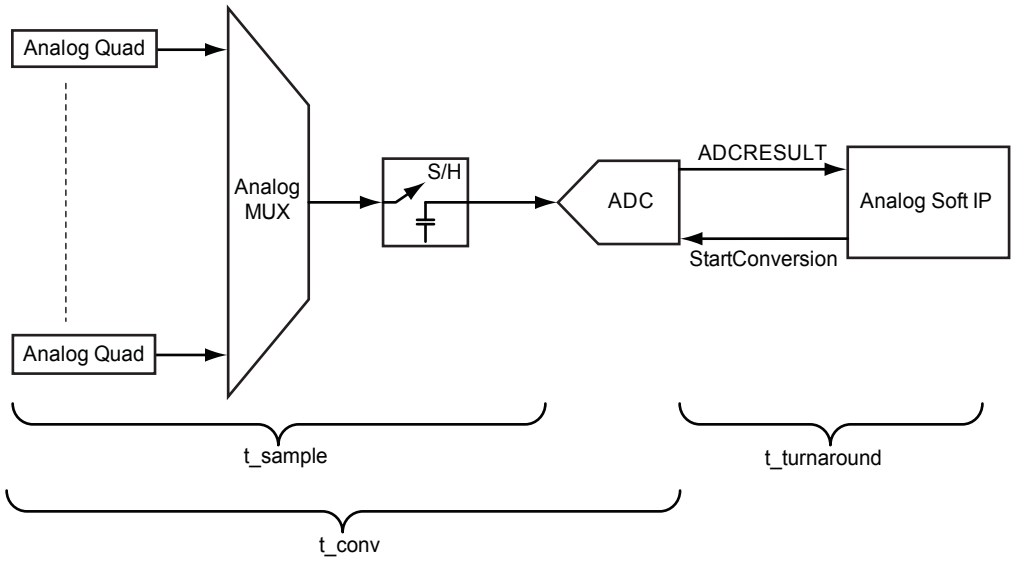


***Figure 8-3* • Simplified ADC Diagram**

The conversion time (t_conv) is the total time required to convert an analog input signal into a digital output (EQ 8-3).

t_conv = t_sync_read + t_sample + t_distrib + t_post_cal + t_sync_write

EQ 3

The components of EQ 3 are defined in Table 8-1.

***Table 8-1* • ADC Conversion Time Formula Elements**

Equation	Description
t_sync_read = sys_clk_period	• Time to latch the input data • sys_clk_period is the ADC interface clock (10 ns to 250 ns).
t_sample = (2 + STC) × adc_clock_period	• STC is the Sample Time Control in the SmartGen GUI. This changes the acquisition time t_sample (sample-and-hold time). STC[7:0] ranges from 0 to 255. • adc_clock_period is the ADC internal clock period (100 ns to 2 µs).
t_distrib = resolution × adc_clock_period	• Time of charge redistribution • adc_clock_period is the ADC internal clock period (100 ns to 2 µs). • Selectable 8-/10-/12-bit resolution mode
t_post_cal = 2 × adc_clock_period	• Time for post-calibration • adc_clock_period is the ADC internal clock period.
t_sync_write = sys_clk_period	• Time for latching the output data • sys_clk_period is the ADC interface clock period (10 ns to 250 ns).

Example 1

Given that only one channel is used without prescaler, the maximum sample rate of a channel can be calculated as follows:

System Clock Period = 1 / (40 MHz) = 25 ns

ADC Clock Period = 1 / (10 MHz) = 100 ns

Acquisition Time = t_sample = 0.4 µs

Resolution = 8

t_conv = t_sync_read + t_sample + t_distrib + t_post_cal + t_sync_write

t_conv = 25 ns + 400 ns + 8 × 100 ns + 2 × 100 ns + 25 ns = 1.45 µs

t_turnaround = 10 × sys_clk = 250 ns

$$\text{Sample Rate} = \frac{1}{1.45\ \mu s + 0.25\ \mu s}\text{ns} = 588\ \text{ksps}$$

Note: To avoid using the prescaler, the maximum voltage value must be set between 2.01 V and 2.56 V in the SmartGen GUI to configure the peripheral as a direct input.

Example 2

Given that only one channel is used with the prescaler, the maximum sample rate of a channel can be calculated as follows:

System Clock Period = 1 / (80 MHz) = 12.5 ns

ADC Clock Period = 1 / (10 MHz) = 100 ns

Acquisition Time = Settling time = 10 µs (max.)

Resolution = 8

t_conv = t_sync_read + t_sample + t_distrib + t_post_cal + t_sync_write

t_conv = 12.5 ns + 10000 ns + 8 × 100 ns + 2 × 100 ns + 12.5 ns = 11.025 µs

Add turnaround time: 11.025 µs + 0.125 µs = 11.15 µs

$$\text{Sample Rate} = \frac{1}{11.025\ \mu s + 0.125\ \mu s}\text{ns} = 89.68\ \text{ksps}$$

Example 3

Given that only one channel is used with the prescaler, the maximum sample rate of a channel can be calculated as follows:

System Clock Period = 1 / (40 MHz) = 25 ns

ADC Clock = 1 / (10 MHz) = 100 ns

Acquisition Time = Settling time = 10 µs (max.)

Resolution = 8

t_conv = t_sync_read + t_sample + t_distrib + t_post_cal + t_sync_write

t_conv = 25 ns + 10000 ns + 8 × 100 ns + 2 × 100 ns + 25 ns = 11.05 µs

Add turnaround time: 11.05 µs + 0.25 µs = 11.3 µs

$$\text{Sample Rate} = \frac{1}{11.05\ \mu s + 0.25\ \mu s}\text{ns} = 88.49\ \text{ksps}$$

Note that when the prescaler is used, a 10 µs settling/acquisition time is recommended for increased accuracy. SmartGen automatically computes values for the STC, clock divider setting (TVC), and ADC clock period. The goal of SmartGen is to meet the minimum sample time requirement with the highest possible ADC clock frequency, which implies a low TVC value and high STC value.

Sample Sequencing Overview

As described in the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet and illustrated in Figure 8-4, there is one ADC in the Analog Block (AB) and up to ten Analog Quads, with three analog inputs each: AV, AC, and AT. The analog input to ADC is selected through a MUX architecture controlled by the CHNUMBER select input. FPGA fabric access to the CHNUMBER input of the AB provides users the flexibility to define custom sample sequencing among the Analog Quads.

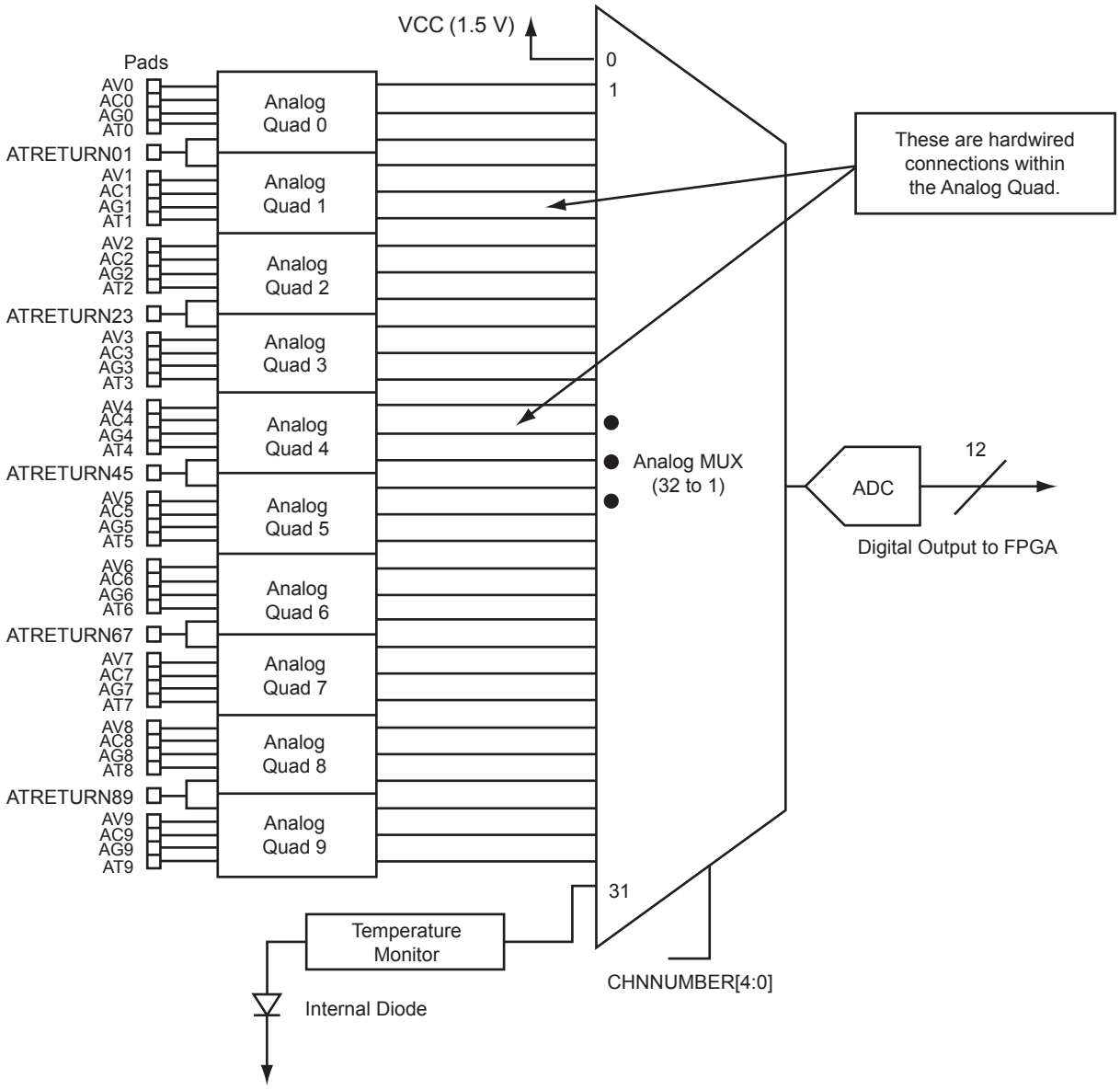


***Figure 8-4* • Analog Block ADC and MUX Architecture**

The flexibility of sample sequencing in the Fusion AB architecture enables conditional sequences and control of the sampling rate of each channel. For example, the designer can sample critical inputs (requiring a higher sampling rate) more often than non-critical inputs. It is also feasible for the design to change sampling sequence and/or rate of analog inputs during operation whenever required.

There is no automatic internal sequencing in the architecture shown in Figure 8-4. Therefore, the CHNUMBER input of the MUX must be controlled and defined by the user's design at all points

throughout its implementation (e.g., SmartGen IP, CoreAI (Analog Interface) register space, and custom logic).

Sample Rate and Sample Sequence Calculation

As the Fusion ADC can be shared among different channels (32 channels in all), the sample rate can be calculated based on the system sampling rate or per-channel sampling rate (EQ 4 and EQ 5).

$$\text{System Sampling Rate} = \frac{\text{Total \# of Samples}}{\text{Total (conversion + turnaround) Time of All Samples}} \quad \textit{EQ 4}$$

$$\text{Channel Sampling Rate} = \frac{\text{Total \# of Samples for a Channel}}{\text{Total \# All Samples}} \times \text{System Sampling Rate} \quad \textit{EQ 5}$$

Example: Equal Weight and Equal Conversion Time

Each channel has a conversion time of 2 µs, as shown in Figure 8-5.

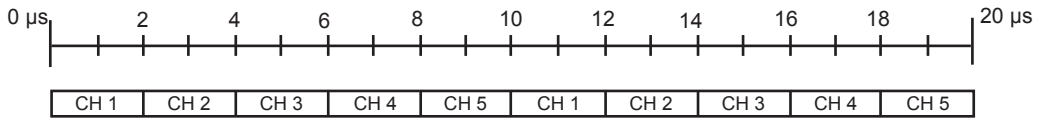


***Figure 8-5* • Equal Weight and Equal Conversion Time**

In this case, there are 10 samples, which take a total of 20 µs. Thus, the total system sampling rate is 10 / (20 µs) = 500 ksps.

Channel 1 sampling rate: (2 / 10) × 500 ksps = 100 ksps

Channel 2 sampling rate: (2 / 10) × 500 ksps = 100 ksps

Channel 3 sampling rate: (2 / 10) × 500 ksps = 100 ksps

Channel 4 sampling rate: (2 / 10) × 500 ksps = 100 ksps

Channel 5 sampling rate: (2 / 10) × 500 ksps = 100 ksps

Example: Unequal Weight and Equal Conversion Time

In this example, the channels are not equally weighted in the sampling sequences shown in Figure 8-6.

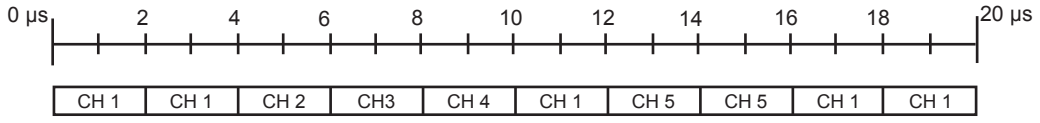


***Figure 8-6* • Unequal Weight and Equal Conversion Time**

In this case, there are 10 samples that take a total of 20 µs, giving a total system sampling rate of 500 ksps (as above). However, the individual channel sampling rates are different.

Channel 1 sampling rate: (5 / 10) × 500 ksps = 250 ksps

Channel 2 sampling rate: (1 / 10) × 500 ksps = 50 ksps

Channel 3 sampling rate: (1 / 10) × 500 ksps = 50 ksps

Channel 4 sampling rate: (1 / 10) × 500 ksps = 50 ksps

Channel 5 sampling rate: (2 / 10) × 500 ksps = 100 ksps

Example: Unequal Weight and Unequal Conversion Time

In this example, channels have different conversion times and are not equally weighted in the sampling sequence, as shown in Figure 8-7.

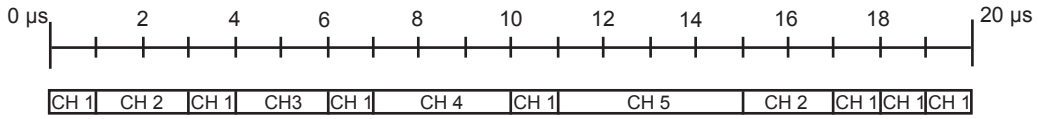


***Figure 8-7* • Unequal Weight and Unequal Conversion Time**

In this case, there are 12 samples in 20 µs, giving a total system sampling rate of 600 ksps.

Channel 1 sampling rate: (7 / 12) × 600 ksps = 349 ksps

Channel 2 sampling rate: (2 / 12) × 600 ksps = 99.6 ksps

Channel 3 sampling rate: (1 / 12) × 600 ksps = 49.8 ksps

Channel 4 sampling rate: (1 / 12) × 600 ksps = 49.8 ksps

Channel 5 sampling rate: (1 / 12) × 600 ksps = 49.8 ksps

Acquisition Time Calculation

Acquisition time (t_sample) specifies how long an analog input signal has to charge the internal capacitor array. Figure 8-8 shows a simplified internal input sampling mechanism of a SAR ADC. The internal impedance (Z_{INAD}), external source resistance (R_{source}), and sample capacitor (C_{INAD}) form a simple RC network. As a result, the accuracy of the ADC can be affected if the ADC is given insufficient time to charge the capacitor. To resolve this problem, the user can either reduce the source resistance or increase the sampling time by changing the acquisition time in the design or in the SmartGen GUI.

Using the ADC with Direct Input

When the Fusion ADC is driven by a direct input (the prescaler is not used), Microsemi recommends driving the analog input pin with low source impedance (R_{source}) for fast acquisition time. High source impedance (R_{source}) is acceptable, but the acquisition time will be increased.

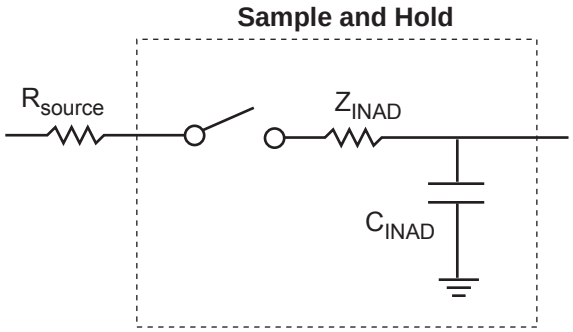


***Figure 8-8* • Simplified Sample and Hold Circuitry**

EQ 6 can be used to approximate the acquisition time that can be entered in SmartGen. In this equation, 5ô is used as an example to approximate the acquisition time, but it is application-dependent. Based on the acquisition time, SmartGen will provide the sample rate calculation using EQ 4 on page 240 and EQ 5 on page 240. If the actual acquisition time is higher than the software setting, the settling time error can affect the accuracy of the ADC, because the sampling capacitor is only partially charged within the given sampling cycle, referred to as the acquisition period (t_sample).

$$\text{Acquisition Time (t_sample)} \sim 5 \times (R_{source} + Z_{INAD}) \times C_{INAD}$$

EQ 6

Users can calculate the minimum actual acquisition time by using EQ 7:

$$V_{OUT} = V_{IN}(1 - e^{-t/RC})$$

EQ 7

For 0.5 LSB gain error, V_{out} should be replaced with (V_{in} – 0.5 × LSB Value):

$$(\text{VIN} - 0.5 \times \text{LSB Value}) = \text{VIN}(1 - e^{-t/RC})$$

EQ 8

where V_{IN} is the ADC reference voltage (V_{REFADC}).

Solving EQ 8,

$$t = RC \times \ln\left(\frac{V_{IN}}{0.5 \times \text{LSB Value}}\right)$$

EQ 9

where R = Z_{INAD} + R_{source} and C = C_{INAD}.

Examples are given in Table 8-2 and Table 8-3.

***Table 8-2* • ADC Parameters – V_{IN} = 2.56 V, R = 4000 Ω (R_{source} ~ 0 Ω), and C = 18 pF**

Resolution (bits)	LSB Value (mV)	Min. Sample/Hold Time for 0.5 LSB (µs)
8	10	0.449
10	2.5	0.549
12	0.625	0.649

***Table 8-3* • ADC Parameters – V_{IN} = 3.3 V, R = 4000 Ω (R_{source} ~ 0 Ω), and C = 18 pF**

Resolution (bits)	LSB Value (mV)	Min. Sample/Hold Time for 0.5 LSB (µs)
8	12.891	0.449
10	3.223	0.549
12	0.806	0.649

Using the ADC with Built-In Prescaler

When using the prescaler, the user can achieve the highest sampling rate by providing a recommended 10 µs acquisition time, which is the maximum settling time when prescaler is used with the ADC. Using SmartGen, users can set the acquisition time in the software GUI so the corresponding sampling rate will be calculated automatically. Users with other design flows must ensure the control logic is allocating sufficient time for the ADC to perform a conversion that satisfies the accuracy requirement.

Prescaler Selection

As mentioned in the "Analog-to-Digital Converter Background" section on page 233, the analog input signals to the ADC must be mapped to the ADC reference voltage range. If the maximum value of an analog input voltage is greater than or less than the ADC reference voltage, the embedded prescaler feature can be used to amplify or attenuate the input voltage signal to match the input voltage range of the ADC. In SmartGen design flows, designers can enter the expected voltage range, and the software will configure the appropriate factors. If a design flow other than SmartGen is used, refer to the "Prescaler Control Truth Table" in the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet to select the appropriate scaling factor.

Analog Configuration MUX (ACM)

The FPGA core uses the Analog Configuration MUX to interface with the Analog Quad configuration settings and Real-Time Counter (RTC) system. To use the Analog Quads, appropriate configuration settings (scaling factor, polarity, prescaler usage, etc.) must be set before using these features. Each Analog Quad has one byte of register space to store its configuration settings, and users can access these registers via the ACM, which is part of the Analog Block Macro. Similarly, the RTC counter register and match register can be accessed by the FPGA core via the ACM. Table 8-4 shows the ACM ports for the FPGA core to interface with the Analog Quads and RTC system.

In a SmartGen design flow, the configuration setting in the GUI is translated into the corresponding configuration bits for the Analog Quads and RTC registers. The configuration settings data is stored in the embedded flash memory, and when the device is powered up or reset, the SmartGen IP loads the configuration settings from the embedded flash memory into the corresponding registers. The Analog Quads and RTC systems are functional after this initialization stage; INIT_DONE (active high) from the SmartGen IP indicates the completion of the initialization stage.

With a different design flow, the Analog Quads and RTC registers must be configured. The configuration data can be stored in the embedded FlashROM, flash memory, or core logic tiles.

Table 8-4 • **ACM Interface**

Port Name	Type	Description
ACMWDATA[7:0]	Input	Writing data from the FPGA core to the analog system
ACMRDATA[7:0]	Output	Reading the data from the analog system to the FPGA core
ACMADDRESS[7:0]	Input	Address
ACMWEN	Input	0 for reading 1 for writing
ACMCLK	Input	Clock input from the FPGA (maximum frequency = 10 MHz)
ACMRESET	Input	Active-low asynchronous reset

Figure 8-9 and Figure 8-10 show the ACM Read and Write operations. Refer to Table 2-44, “Analog Configuration Multiplexer (ACM) Timing,” in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet for the corresponding timing parameters.

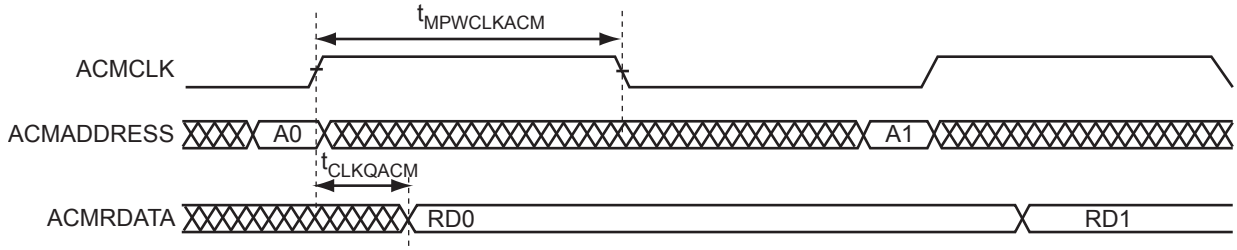


Figure 8-9 • **ACM Read Waveforms**

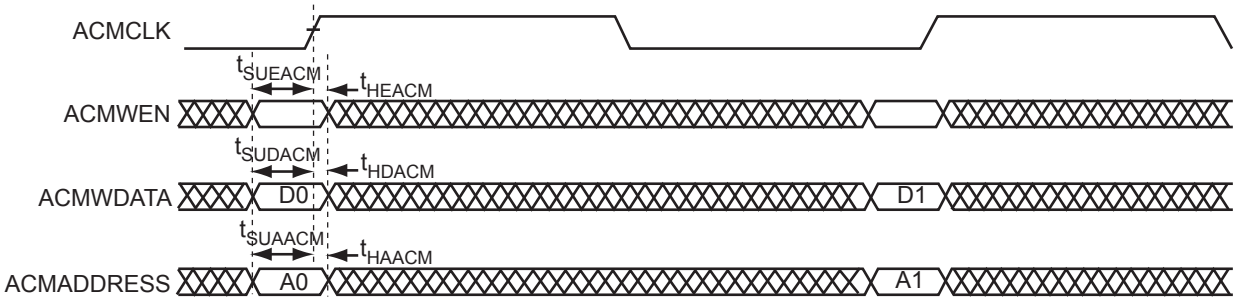
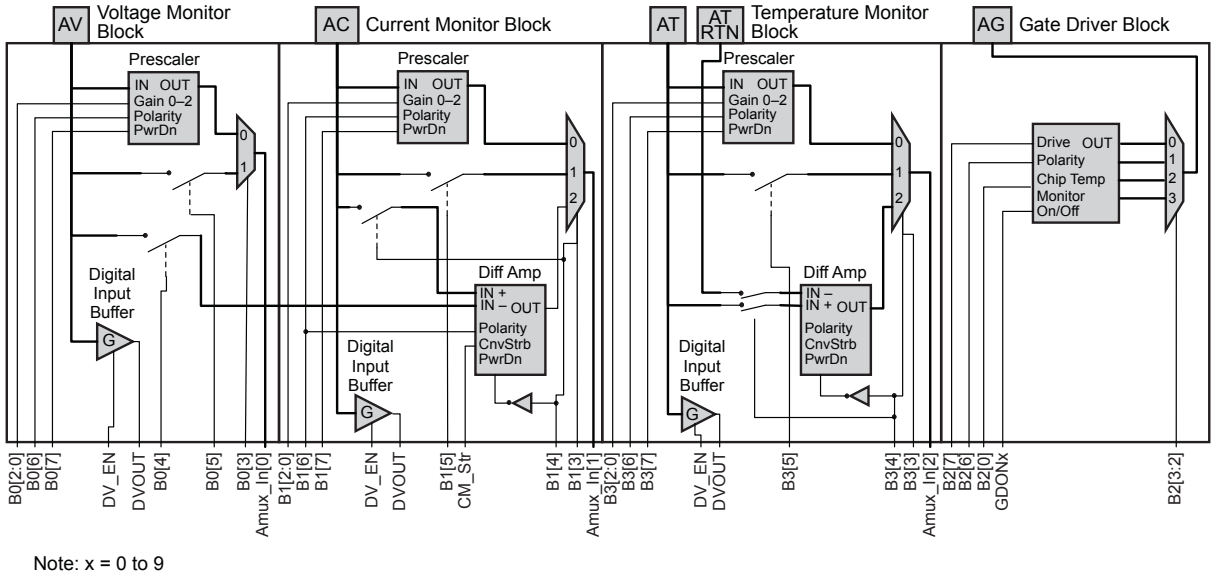


Figure 8-10 • **ACM Write Waveforms**

Figure 8-11 shows the block diagram of the Analog Block. The figure shows how the configuration byte (B0–4) is connected to each block and how each block is interfaced with the user-accessible Analog Block macro and the ADC. Note that the soft IP interface to the AB manages the configuration, making connectivity transparent to the user.



Note: x = 0 to 9

Figure 8-11 • **Analog Block**

Refer to the *Analog Quad ACM Byte Assignment* table in the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet for the explanation of each bit setting and its corresponding configurations.

Refer to the *ACM Address Decode Table for Analog Quad* table in the *Fusion Family of Mixed-Signal Flash FPGAs* datasheet for the corresponding address for each Analog Quad and RTC register.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
December 2011	Some duplicate text was removed from the "ADC Clock" section (SAR 20525).	N/A

9 – Fusion Design Solutions and Methodologies

With a rich mixture of analog and digital features, coupled with the ability to build mixed signal designs either in HDL or with a processor, the Fusion mixed signal FPGA offers designers the ultimate in flexibility. However, with product flexibility comes design complexity. Microsemi offers several design solutions that make the design process simple for all users.

HDL Design with Analog System Soft IP

The Analog System Soft IP Design Flow is an IP-based design method for HDL designs, which establishes a backbone to interconnect the Fusion FPGA fabric, the Analog System, the embedded flash memory block, and other peripherals. Figure 9-1 on page 248 provides an overview of the design flow.

The Analog System Soft IP Design Flow offers a number of advantages to users. All the required soft IP cores are free. Sample sequence control, averaging/filtering and threshold response functions are built-in and specified by an intuitive GUI in Libero SoC software. IP configuration and connectivity are tightly integrated into SmartDesign and Libero SoC, enabling users to rapidly and seamlessly implement the complete analog and peripheral interface. Users do not have to write up their own code to control the analog and flash memory systems.

Users have several options for integrating the Analog System into their design, but at the heart of it all are the Analog System Builder and Flash Memory System Builder from Libero SoC. The Analog System Builder creates VHDL or Verilog source code and the configuration file to be stored in the embedded flash memory. Users can invoke the Analog System Builder and Flash Memory System Builder, along with the generators for other Fusion and IP cores, from the Cores Catalog window in Libero SoC, or modify an existing configuration from within SmartDesign. Other Fusion-specific core generators in Libero SoC include the No-Glitch MUX (NGMUX), RC Oscillator, Crystal Oscillator, SRAM, FIFO, FlashROM, I/Os, and Voltage Regulator Power Supply Monitor (VRPSM).

SmartDesign is a unique block-diagram-based design entry tool introduced in Libero SoC v8.0 that gives users the capability to visually create block-level system designs and automatically abstract the result into synthesis-ready source code. Designers can build their entire design in SmartDesign. For Fusion designs, SmartDesign identifies required connections between blocks in the Analog System and automatically stitches them together. SmartDesign also provides a connectivity grid, which enables users to graphically perform port mapping across all the blocks in the design. If the Analog System is used in SmartDesign, SmartDesign will audit the connections and any updates made to the Analog System. For example, if the Analog System is regenerated, the tool will remind the user that the NVM client must be regenerated.

If users do not use SmartDesign, they can generate the Analog System and required embedded flash memory clients from the Libero Cores Catalog window. They must then manually stitch the Fusion blocks together with the user HDL code.

Note: Any time the Analog System is regenerated, the Analog System client for the embedded flash memory must also be regenerated from the Flash Memory System Builder in Libero SoC.

After building the HDL design, users should take their design through the rest of the FPGA design flow—synthesis, post-synthesis simulation, and Designer functions including compile, place-and-route, package pin assignment, and static timing analysis—and then perform back-annotated simulation. Once the design is finalized and timing has been verified, the design is ready to be programmed into the FPGA using the FlashPro programmer and software. The "Design State Management in SmartDesign" section on page 249 and the "Changing Memory Content" section on page 249 discuss how to handle design iterations for Fusion within Libero SoC.

FlashPoint provides the interface to generate programming files for Fusion devices. The FlashROM, Fusion embedded flash memory, and security settings can be reset and reprogrammed using FlashPoint, which is integrated into both Designer and the FlashPro software.

With FlashPoint integrated with FlashPro, users can modify the contents of the embedded flash memory or the FlashROM without using Designer or Libero SoC. By default, Designer's program file generation

from FlashPoint generates a programming database file (PDB) file instead of a STAPL file. A PDB file is required to enable users to make modifications using the FlashPro software. When users need to change the programming settings in FlashPro, they can simply click the **Configure PDB** button. All modifications are stored in the PDB file, and FlashPro uses the information to program the device with the appropriate settings. Libero SoC / Designer does not have to be open to make these modifications.

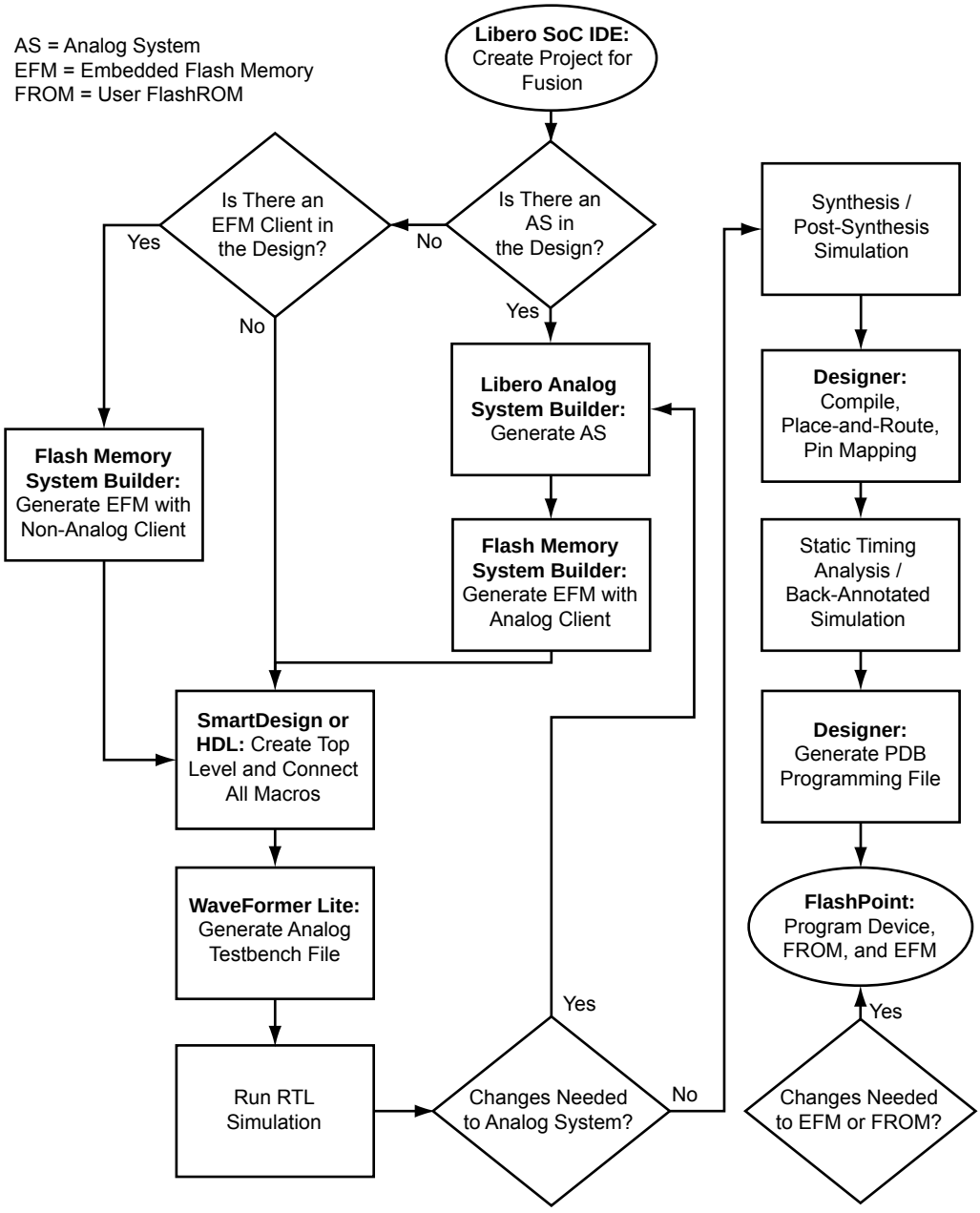


Figure 9-1 **• Analog System Soft IP Design Flow**

Design State Management in SmartDesign

When any component with instances in a SmartDesign design is changed, all instances of that component detect the change. If the change only affects the memory content, user changes do not affect the component's behavior or port interface, and the user's SmartDesign design does not need to be updated. If the change affects the behavior of the instantiated component but the change does not affect the component's port interface, the design must be resynthesized, but the SmartDesign design does not need to be updated.

If the port interface of the instantiated component is changed, the user must reconcile the new definition for all instances of the component and resolve any mismatches. If a port is deleted, SmartDesign will remove that port and clear all the connections to that port when the user reconciles all instances. If a new port is added to the component, instances of that component will contain the new port when the user reconciles all instances. The affected instances are identified in the SmartDesign design in the Connectivity Grid and the Canvas with an exclamation point. Right-click an instance and choose **Update With Latest Component**.

Note: For HDL modules instantiated in a SmartDesign design, if the modification causes syntax errors, SmartDesign does not detect the port changes. The changes will be recognized when the syntax errors are resolved.

Changing Memory Content

For certain cores such as Analog System Builder and Flash Memory, it is possible to change the configuration such that only the memory content used for programming is altered. In this case, Libero SoC only invalidates the programming file, but synthesis, compile, and place-and-route results remain valid.

When the user modifies the memory content of a core—such as Analog System Builder or RAM with Initialization—that is used by a Flash Memory core, the Flash Memory core indicates that one of its dependent components has changed and that it needs to be regenerated. This indication is shown in the Design Hierarchy or Files tab. In these cases, Libero SoC indicates that the programming file is out of date, but synthesis and place-and-route remain valid. The user only needs to regenerate the programming file in FlashPoint.

If any core is regenerated when the HDL file is not modified, the Libero Project Manager design state will not invalidate synthesis or place-and-route results. For these scenarios, the new embedded NVM data file (EFC file) will be used to update the programming file within Libero SoC, or can be imported into FlashPro. Some specific cores are listed below:

- RAM with Initialization core – The memory content can be modified without invalidating synthesis.
- Analog System Builder core – The following can be modified without invalidating synthesis:
  - Existing flag settings: threshold levels, assertion/deassertion counts, OVER/UNDER type
  - Modifying sequence order or adding sequence operations
  - Changing acquisition times
  - Resistor value for the current monitor
  - RTC time settings
  - Gate driver source current (when destination is a negative voltage rail--the only source of current from the gate driver is the ground plane)
- Flash Memory System Builder core – The following can be modified without invalidating synthesis:
  - Modifying memory file or memory content for clients
  - JTAG protection for initialization clients

Microprocessor/Microcontroller Design

Microsemi offers several microprocessor and microcontroller solutions for customers, all of which are tightly integrated with Libero SoC, optimized for Microsemi FPGA architecture, and supplied with a complete toolset for code compile and debug. Here is a summary of Microsemi's available solutions:

- **Cortex-M1**: The first ARM® processor developed specifically for implementation in FPGAs. Cortex-M1 is available without license fees or royalties for use in M1 ProASIC3/E and Fusion devices.
- **CoreMP7**: CoreMP7 is a soft IP version of the ARM7TDMI-S™ that is optimized for use in M7 Fusion and ProASIC3/E flash-based FPGAs. CoreMP7 is available with no license fees or royalties—bringing ARM7™ to the masses.
- **Core8051(s)**: Both Core8051 and Core8051s are code-compatible with the industry-standard 8051 architecture, allowing designers to utilize existing code while shortening design time. Further, both are single-cycle execution architectures, executing one instruction per clock cycle. Core8051 has the traditional SFR memory space and includes the standard 8051 peripherals, and Core8051s replaces the traditional SFR interface with an Advanced Peripheral Bus (APB) interface, allowing the customization of the 8051 peripheral set.
- **CoreABC**: CoreABC (AMBA [Advanced Microcontroller Bus Architecture] Bus Controller) is the smallest and first RTL-programmable soft microcontroller available for FPGAs. The free controller resides on the APB, can be implemented in as few as 241 tiles, and can be used in the smallest Microsemi devices.
- **LEON3**: LEON3 is a 32-bit processor based on the SPARC V8 architecture, optimized for use in Microsemi FPGAs. A fault-tolerant version of the LEON3 processor is available for system-critical applications.
- **AMBA**: Microsemi supplies a full range of subsystem IP cores: AMBA bus interfaces, memory controllers, timers, and others. The subsystem IP connects to the processor via the AMBA bus and is available for free in SmartDesign.

For the above ARM-based processors and CoreABC, Microsemi offers CoreAI (Analog Interface) to interface with the Analog System. CoreAI allows for simple control of the analog peripherals within Fusion. Control can be implemented with an internal or external microprocessor or microcontroller, or with user-created custom logic within the FPGA fabric. The AMBA APB slave interface is used as the primary control mechanism within CoreAI, as shown in Figure 9-2. CoreAI instantiates the Analog Block (AB) macro, which includes the Analog Configuration MUX (ACM) interface, Analog Quads, and Real-Time Counter (RTC). Several aspects of CoreAI can be configured using top-level parameters (Verilog) or generics (VHDL). For a detailed description of the parameters/generics, refer to the *CoreAI Handbook*.

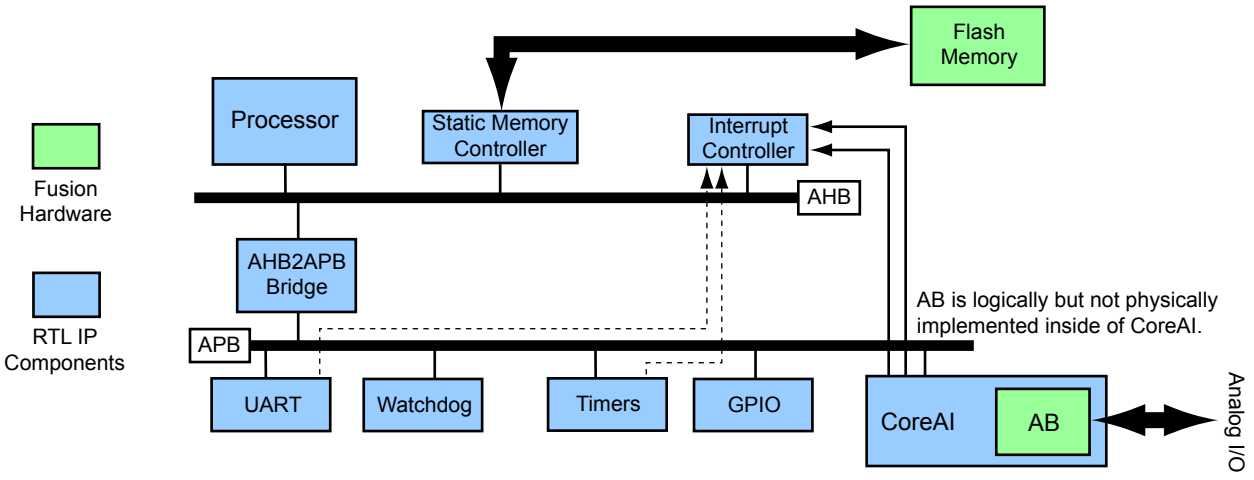


***Figure 9-2* • Processor System Using CoreAI**

Tools Overview

Microsemi offers FPGA development tools for microprocessor and microcontrollers, and a complete development and debug environment for Microsemi's microprocessor solutions (Figure 9-3). With Microsemi solutions, users can shorten development time using SmartDesign IP Deployment Platform (IDP), which includes a graphical user interface and a block stitcher to simplify the assembly of IP cores for embedded applications in FPGAs. This tool integrates with Libero SoC, which includes Microsemi Designer software for place-and-route. To enable Cortex-M1 and CoreMP7 users to debug the programs they write for their processors, there is optional hardware within the core that implements JTAG debug features, such as breakpoints. Various third-party tools, like the ARM RealView® Developer Kit and Microsemi's own free SoftConsole, provide tools for building, debugging, and managing software development projects that run on the processor. The toolkit, available from Microsemi, contains an optimized C compiler, debugger, assembler, and instruction set simulator. For an overview of the processor design flow, refer to the *Actel Processor Design Flow webcast*. With a processor built into the Fusion FPGA fabric, the Fusion embedded flash memory can be used for program storage, which can in turn be executed out of internal or external memory. Implement the appropriate IP in SmartDesign for an internal or external RAM interface. Use the Data Storage Client from the Flash Memory System Builder in Libero SoC to create a partition in the flash memory and FlashPoint to load program code during device programming.

For CoreABC-based designs, users can choose either a soft or hard implementation of the core. In the soft implementation, CoreABC operates on assembly instruction code residing in flash memory and executed either directly out of flash or from local SRAM. In the hard implementation, CoreABC becomes a VHDL- or Verilog-coded state machine derived from assembly code. For either implementation, Microsemi recommends that users configure and generate the core using SmartDesign. With SmartDesign, users can easily connect the required peripherals and build the subsystem including CoreAI on the APB. Once the SmartDesign component including CoreABC and CoreAI is generated, users should follow a standard HDL FPGA design flow. Note that if CoreABC soft mode is used, users must create an initialization client for the RAM from the Embedded Flash Memory, using the Flash Memory System Builder in Libero SoC. The initialization client can be created by loading the memory contents file that is automatically created by SmartDesign during generation of CoreABC. For more information on building a design with CoreABC, refer to the *Fusion Starter Kit User's Guide and Tutorial*.

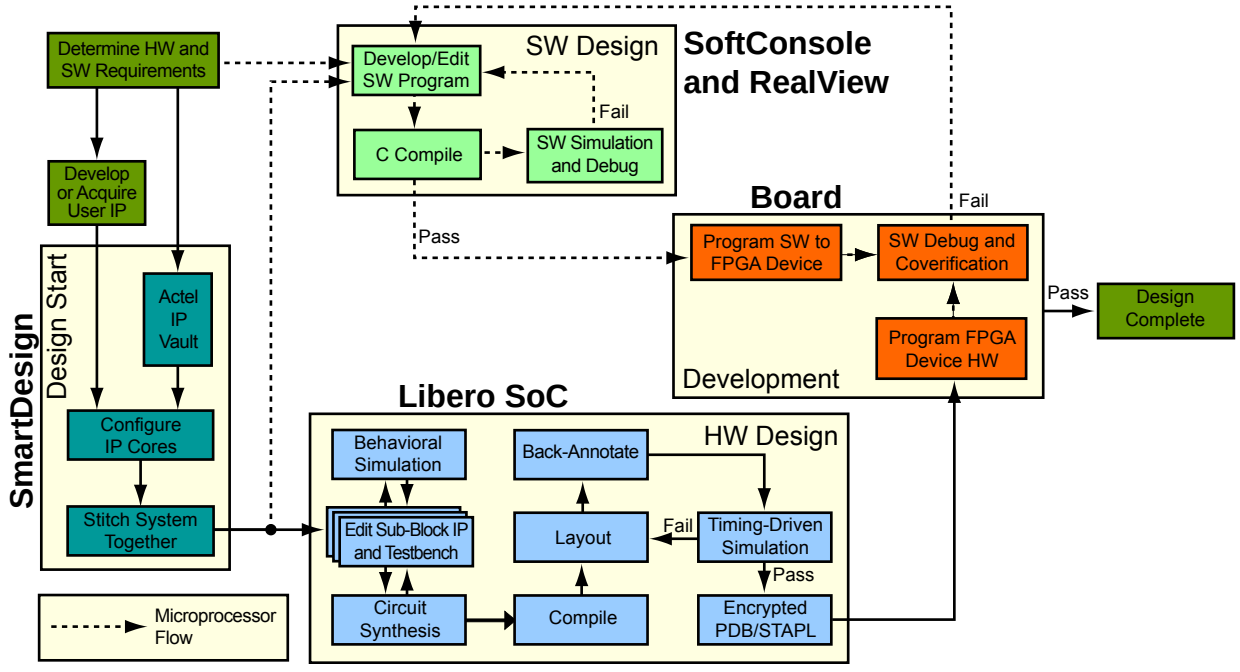


Figure 9-3 • **Fusion Design Flow**

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
December 2011	Occurrences of CoreConsole were replaced with SmartDesign (SAR 24945).	N/A
	The "Changing Memory Content" section was revised to add the following explanation for the gate driver source current: "when destination is a negative voltage rail—the only source of current from the gate driver is the ground plane" (SAR 31653).	249

10 – Interfacing with the Fusion Analog System: Processor/Microcontroller Interface

Objective

This chapter describes the applications in which a microprocessor or microcontroller is the core of the design that controls the Fusion Analog Block (AB). The design's microprocessor/microcontroller interacts with CoreAI (Analog Interface) as the Analog Block interface and does not access the AB macro directly.

The design in this chapter uses CoreAI within Microsemi SmartDesign IP Deployment Platform (IDP). However, the contents and usage of this chapter are not limited to SmartDesign users.

CoreAI

Introduction

CoreAI is Microsemi's Analog Interface core designed to facilitate access to the Fusion Analog Block (AB) by a microprocessor/microcontroller. CoreAI provides register/address space that can be written to or read from by a microprocessor/microcontroller to configure, control, and interact with the Analog Block. For more information on CoreAI specifications and usage, refer to the *CoreAI Handbook*. This section describes how the microprocessor/microcontroller accesses and configures CoreAI to implement voltage, current, and temperature monitoring, as well as gate-driving applications.

CoreAI Settings in SmartDesign

CoreAI can be configured by writing the desired values into all required CoreAI address spaces. CoreAI's parameters and generics, used by the core's source code, can be set within SmartDesign. Refer to the CoreAI Parameter/Generic Descriptions table in the *CoreAI Handbook* for a complete list of parameters.

Analog Configuration MUX (ACM) Clocking, Interrupt, and Internal Temperature Monitor Configuration

The first step of CoreAI configuration in SmartDesign is the ACM clock divider setting, shown in Figure 10-1. The ACM clock (ACMCLK) maximum frequency is limited to 10 MHz, so the user must select a setting that will ensure that ACMCLK is not greater than 10 MHz.

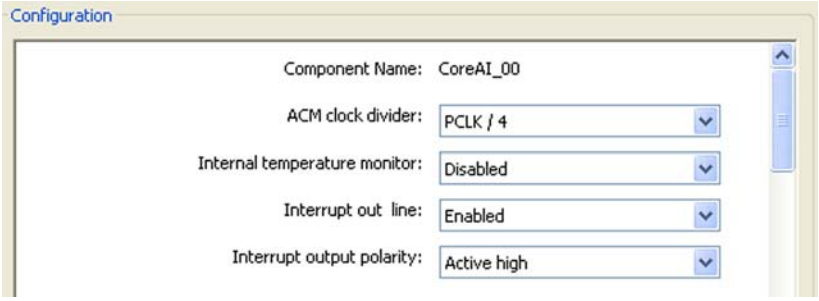


***Figure 10-1* • ACM Clock Configuration in SmartDesign**

In the ACM clock divider setting, select the dividing factor to be used based on EQ 1:

$$F_{ACMCLK} = F_{PCLK} / n$$

EQ 1

where n = 2, 4, 8, or 16; F_{ACMCLK} is the frequency of the ACMCLK; and F_{PCLK} is the frequency of PCLK, the peripheral bus clock usually connected to the design main system clock.

The PCLK frequency is essentially the speed of the Advanced Peripheral Bus (APB) and the clock speed of the design's microprocessor/microcontroller. For example, if the system clock speed is designed to be more than 20 MHz, the ACM clock divider factor cannot be set to two, since it correlates to an ACM clock frequency of more than 10 MHz.

Analog Quad Configuration

The main part of CoreAI configuration in SmartDesign is dedicated to the Analog Quad settings (Figure 10-2). Each quad consists of four settings—AV, AC, AT, and AG—and represents the Fusion device architecture.

***Figure 10-2* • Analog Quad Configuration in SmartDesign**

The SmartDesign GUI settings facilitate the configuration of the core's internal register space (initialization). The settings in SmartDesign set the parameters and generics of the CoreAI source code and create information to be used in the Analog Block's initialization. SmartDesign will export all files used for initialization to the software export folder. *Acm_defines.h* contains definitions for the values used to configure the ACM (defined per user settings in SmartDesign) in the Analog Block of a Fusion device.

Note: For CoreABC (AMBA [Advanced Microcontroller Bus Architecture] Bus Controller) users, if the APBWRT ACM command is enabled, a Verilog or VHDL file will be generated. This file contains a lookup table that will configure the ACM with user-specified settings. Initialization files and their usage are discussed in further detail in "Analog Configuration MUX Initialization" on page 258. The user can also configure the ACM bus without the SmartDesign-generated initialization files.

The following describes the Analog Quad software GUI settings and their effects on CoreAI parameters/generics:

- AVn input: The AV configuration drop-down menu lists all supported analog input voltage ranges and polarities for ACM initialization purposes. The main categories for AV configuration are as follows:
  - **Analog voltage input enabled/disabled**: If disabled, the specified analog AV input of the Analog Block will be tied LOW internally and will not be listed as an accessible CoreAI port in the code.
  - **AV input used as digital input**: If used as a digital input, the corresponding DAVOUT output port will be listed as a top-level port of the core, to be connected to the digital input of your design.
- ACn input: The AC configuration drop-down menu supports all analog input voltage ranges and polarities. The basic settings are as follows:
  - **AC pin disabled**: If disabled, the specified analog AC input of the Analog Block will be tied LOW internally and will not be listed as an accessible CoreAI port in the code.
  - **AC pin used as current monitor**: If the AC pin is set to be used as a current monitor or voltage monitor, the CFG_ACx bits will be set as described in the *CoreAI Handbook*. In current monitoring applications, sampling the current from an Analog Quad (configured as current monitor) is controlled by the corresponding CMSTB input. SmartDesign gives the option to configure the CMSTB input of a current monitoring quad to be either register-driven

and controlled by the software, or hardware-driven. If configured to be hardware-driven, an HD_CMSTB port is added to the CoreAI input pins and should be controlled by the user logic in the FPGA fabric.

  - **AC pin used as voltage monitor**: If the AC pin is set to be used as a voltage monitor, the CFG_ACx bits will be set as described in the *CoreAI Handbook*.
  - **AC input used as digital input**: If used as a digital input, the corresponding DACOUT output port will be listed as a top-level port of the core, to be connected to the digital input of your design.

- ATn input: In the AT configuration drop-down menu, the following parameters can be set:
  - **AT input enabled as temperature monitor or completely disabled**: If disabled, the specified analog AT input of the Analog Block will be tied LOW internally and will not be listed as an accessible CoreAI port in the code.
  - **AT input used as digital input**: If used as a digital input, the corresponding DATOUT output port will be listed as a top-level port of CoreAI, to be connected the digital input of your design.

  In temperature monitoring applications, sampling temperature from the AT input pin (configured as temperature monitor) is controlled by the corresponding TMSTB input. SmartDesign gives the option to configure the TMSTB input of a temperature monitoring quad to be either a software-driven or a hardware-driven register space. If configured to be hardware-driven, an HD_TMSTB port is added to the CoreAI input pins. The HD_TMSTB port must be controlled by the user's logic in the FPGA fabric. A similar implementation is used in the CoreAI module for internal temperature monitoring.
- AGn output: In the AG configuration drop-down menu, the following parameters can be set:
  - **AG output disabled**: If disabled, the specified AG output will be unused and omitted from the top-level CoreAI ports. The corresponding GDON input of the Analog Block will be tied LOW internal to the core.
  - **AG output enabled and driven by software-controlled register (software-driven)**: If the AG output is enabled and driven by software, the AG output pin will turn on or off (consequently turning the external PMOS or NMOS on and off) when 1 or 0 is written to the desired location of the analog-to-digital converter (ADC) control registers.
  - **AG output enabled and driven by FPGA core logic (hardware-driven)**: If the AG output is enabled and driven by hardware, the corresponding HD_GDONx input of CoreAI will be added to the top-level ports of the core, and the AG output pin will follow the logic that drives that HD_GDON input of CoreAI. In this case, the appropriate GDON bit in ADC control register 5 will be set to 1 (enabled) during initialization of the CoreAI.

ADC Settings

The main ADC settings can be configured in SmartDesign:

- Mode
- TVC
- STC (Sample Time Control—used to define sampling time of ADC: {2 to 257} × ADC clock period)
- ADCRESET
- ADCSTART
- Power Down
- VAREF Selection
- ADC Channel Number Control

The purpose and specific usage of each setting are described in the *Analog-to-Digital Converter* section of the "Designing the Fusion Analog System" section on page 233 and in the *CoreAI Handbook*. These settings can be controlled in two manners: software- or hardware-driven. When software-driven, these settings are controlled by an APB register/address space in CoreAI according to the address mapping described in the *CoreAI Handbook*. When hardware driven, these settings are controlled by direct inputs (e.g., HD_TVC) to the core and need to be managed accordingly by the user design in the FPGA fabric or tied to specific values.

When the Analog Block (ADCSTART) and ADC (ADCRESET) are configured as software-driven, the ADCSTART and ADCRESET registers are self-clearing, and ADCSTART or ADCRESET can be initiated by writing 1 to the specific register address. These registers will clear themselves and be ready for the next ADCSTART or ADCRESET request with no need for the user to set these registers back to 0 prior to the next request. The self-clearing functionality does not exist when ADCSTART or ADCRESET are hardware-driven, and these inputs must be reset to 0 before issuing the next request.

Clocking Scheme

There are four main clock domains in a basic mixed signal Fusion design: the system clock (SYSCLK), ACMCLK, ADCCLK, and the initialization clock.

Any basic, functional mixed signal Fusion design that uses Analog Quads needs to interface with the ACM and ADC. Besides the frequency requirements of ACMCLK and ADCCLK, there will be areas in which data is transferred from one clock domain to another between SYSCLK and ACMCLK or ADCCLK.

SYSCLK

The system clock (SYSCLK) is the microprocessor clock. When CoreABC (or any other processor/controller) and CoreAI are used together, SYSCLK and PCLK (the APB interface clock) are normally connected together as the overall system clock.

ACMCLK

ACMCLK is the clock input to the ACM used for configuration/initialization of the Analog Quads. The ACMCLK frequency is limited to 10 MHz. CoreAI uses an internal clock divider that divides the input SYSCLK by a user-defined factor and drives it to the ACMCLK input of the Analog Block. When CoreAI is used as an APB peripheral to the microprocessor, ACMCLK will be seamless if the PCLK / n factor is set appropriately in the CoreAI settings to limit the ACMCLK frequency to below 10 MHz.

ADCCLK

ADCCLK is the clock input to the Analog Block used by the ADC. ADCCLK is used for internal ADC operations and serves as a reference for determining the conversion time of the ADC (through the STC setting). The ADCCLK maximum frequency is 10 MHz. An internal clock divider inside the Analog Block is used to divide the input system clock (SYSCLK) and generate the ADCCLK input to the ADC. The internal divider value is configurable through the 8-bit TVC register, where TVC can be set from 0 to 255 (EQ 2):

$$ADCCLK = SYSCLK / (4 \times (1 + TVC)) \quad \textit{EQ 2}$$

Setting TVC to 0 sets the ADCCLK frequency to the SYSCLK frequency divided by four, and the ADCCLK-to-SYSCLK division ratio increases in steps of four as the TVC value increases. This is important if the design requires ADCCLK to run at specific speed to maintain a certain sampling rate. For example, if the ADC is required to run at 10 MHz, the SYSCLK input to the Analog Block should be 40 MHz (TVC = 0), 80 MHz (TVC = 1), 120 MHz (TVC = 2), etc. If a required SYSCLK frequency does not result in the determined ADCCLK frequency (governed by EQ 2), SYSCLK can be fed to the Fusion CCC to generate an auxiliary SYSCLK signal with the desired frequency. This auxiliary SYSCLK signal can then drive the Analog Block. Use EQ 2 and appropriate TVC settings to determine the appropriate frequency of the auxiliary SYSCLK that will generate the desired ADCCLK frequency.

Initialization Clock

Typically in microprocessor-based applications, the processor's program code is stored in a nonvolatile memory, used during power-up boots. If the microprocessor program code is stored in the Fusion embedded flash memory, clocking the embedded flash memory is an important part of the design's clocking scheme. If the embedded flash memory is used to store the processor's program code, there are two general options for the processor to access the code:

- Accessing the embedded flash memory directly. The performance of SYSCLK is limited by the speed of the embedded flash memory. In this scheme the embedded flash memory clock and SYSCLK are driven by the same signal.
- Initializing embedded SRAM blocks with the contents of the embedded flash memory and running the processor's program from SRAM. The contents of the SRAM should be initialized by the embedded flash memory (power-up initialization). The memory initialization clients can be created using Libero SoC software. The dual-port SRAM has one port connected to the embedded flash memory and one port connected to the microprocessor. The embedded flash memory (the initialization part of the design) is clocked separately from the operational part of the design. The initialization clock (INIT_CLK) input of the initialization client is limited to 10 MHz maximum frequency. Therefore, if the SYSCLK frequency is more than 10 MHz, INIT_CLK should be driven separately. The Fusion CCC can be used to generate INIT_CLK with a frequency less than 10 MHz.

  When CoreABC is the system's microcontroller, the instruction program SRAMs are instantiated within the CoreABC module exported from SmartDesign, and the clocking scheme, shown in Figure 10-3, can be used to ensure that the initialization of the SRAM from flash memory is performed by a clock of less than 10 MHz. Once the initialization is completed, INIT_DONE will assert (active high), and the design runs from the high-speed system clock.

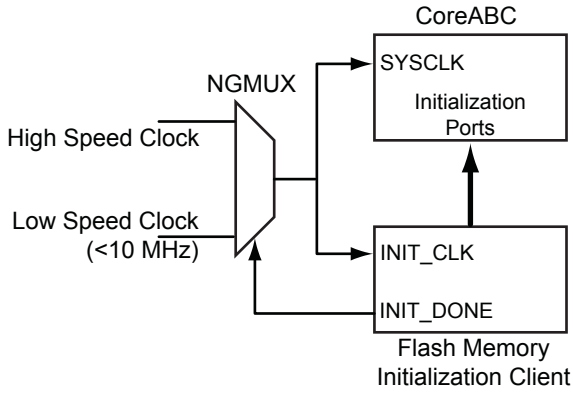


***Figure 10-3* • Clocking Scheme when Initializing SRAM with CoreABC Instructions**

The clocking architecture shown in Figure 10-3 uses an NGMUX block as the multiplexer to switch between the high- and low-speed clocks. The usage of the NGMUX macro is to prevent glitches on the clock output of the MUX when switching between the two clock inputs. Refer to the *No-Glitch Multiplexer (NGMUX)* section of the "Fusion Clock Resources" section on page 109 for more information about the NGMUX.

Analog Configuration MUX Initialization

The ACM is a register space in the Analog Block used to configure the Analog Quads and the Real-Time Counter (RTC) architecture with the user's specification. Since the configuration scheme is stored in registers, the ACM needs to be initialized after each power-up. When using CoreAI as the interface to the Analog Block, the initialization of the ACM should be done in two steps:

1. Reset the ACM register space to put unused Analog Quads into a known mode.
2. Configure used Analog Quads (and RTC if used in the design) to desired specification.

Note: In addition to ACM initialization, the ADC needs to be calibrated after power-up for correct functionality; see "ADC Configuration and Calibration" on page 261.

ACM Reset

The ACM registers can be reset by activating the active-high ACMRESET bit of the CoreAI register space.[1] When configuring CoreAI, this input is controlled by bit 0 of the ACM control/status register. The ACM can be reset by writing 1 to this bit. Note that bit 0 of the ACM control/status register is self-clearing: when written with 1, it will clear itself; it does not need to be written with 0 to clear it. The ACM control/status register is designed to be located at address 0x00 of the CoreAI internal register address map.

ACM Initialization

Before a design enters the operational phase, the ACM must be initialized with the desired configuration for the Analog Quads and/or RTC registers. When SmartDesign generates all the necessary files for a microprocessor-based design, it also generates the files to be used for ACM initialization.

There are two files, found in the *SoftwareExport* folder of the SmartDesign directory, that can be used for ACM initialization: *acm_defines.h* and *quads_acm_cfg.h*. These two files contain information on initialization values for different ACM address spaces per user entries in the CoreAI settings within SmartDesign. These files can be referenced by the general microprocessor program design to be used for initialization prior to entry into operational sections.

The use of the above-mentioned files is optional. The user can manually write to the desired ACM address space (from a microprocessor/microcontroller through the APB into CoreAI) with values that will initialize the targeted Analog Quads and/or RTC registers to the desired configuration.

Note: When writing to ACM registers, the design should check the ACM status register to ensure that the previous ACM actions (write, read, or reset) are completed and that the ACM is ready to be accessed again.

1. *Note that the ACMRESET input to the Analog Block is active low. To reset the ACM in the CoreAI register space, the ACM RESET bit should be set to 1.*

ACM Initialization Specific to CoreABC

In addition to the initialization files and methodology described above, if the APBWRT ACM instruction is activated in the CoreABC settings in SmartDesign, SmartDesign will export an HDL file named *acmtable.vhd*. This file can be found in the *CoreABC/RTL* folder within the Libero project. Below is an example of an *amctable.vhd* file generated by SmartDesign:

```
-- ******************************************************************/
-- Copyright 2007 Actel Corporation. All rights reserved.
-- IP Solutions Group
--
-- ANY USE OR REDISTRIBUTION IN PART OR IN WHOLE MUST BE HANDLED IN
-- ACCORDANCE WITH THE ACTEL LICENSE AGREEMENT AND MUST BE APPROVED
-- IN ADVANCE IN WRITING.
--
-- File: INSTRUCTIONS.vhd
--
-- Description: Simple APB Bus Controller
--       ACM Lookup table
--
-- Rev: 2.3  01Mar07 IPB : Production Release
--
-- Notes:
--
-- ******************************************************************/

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

use work.support.all;

entity ACMTABLE is
  generic ( ID   : integer range 0 to 9;
            TM   : integer range 0 to 99
          );
  port   ( ACMADDR : in std_logic_vector(7 downto 0);
             ACMDATA : out std_logic_vector(7 downto 0);
             ACMDO  : out std_logic
           );
end ACMTABLE;

architecture RTL of ACMTABLE is

begin

-- This is dummy data used for testing

process(ACMADDR)
variable ADDRINT : integer range 0 to 255;
begin
  ADDRINT := conv_integer(ACMADDR);
  ACMDO <= '1';

  if TM>0 then
    case ADDRINT is
      when 0  to 99 => ACMDATA <= not ACMADDR;
      when 101 to 255 => ACMDATA <= not ACMADDR;
      when others   => ACMDATA <= (others =>'-'); ACMDO <= '0';
    end case;
  end if;

  if TM=0 then
  -- CCDirective Insert code
```

```
  --ACM lookup table for CoreABC_00(ID=0) with CoreAI_00
    if ID=0 then
      case ADDRINT is
        when 1 => ACMDATA <= conv_std_logic_vector(16#83#, 8);
        when 2 => ACMDATA <= conv_std_logic_vector(16#82#, 8);
        when 5 => ACMDATA <= conv_std_logic_vector(16#82#, 8);
        when 7 => ACMDATA <= conv_std_logic_vector(16#80#, 8);
        when 9 => ACMDATA <= conv_std_logic_vector(16#83#, 8);
        when 11 => ACMDATA <= conv_std_logic_vector(16#80#, 8);
        when 13 => ACMDATA <= conv_std_logic_vector(16#82#, 8);
        when 15 => ACMDATA <= conv_std_logic_vector(16#80#, 8);
        when 17 => ACMDATA <= conv_std_logic_vector(16#82#, 8);
        when 19 => ACMDATA <= conv_std_logic_vector(16#80#, 8);
        when others => ACMDATA <= (others => '-'); ACMDO <= '0';
      end case;
    end if;
  end if;

end process;

end RTL;
```

The last portion of the *acmtable.vhd* file defines the ACM address spaces and the corresponding values used to initialize certain Analog Quads to user specifications.

If the user decides to perform all the initialization within the CoreABC program code without utilizing any logic tiles from the FPGA fabric, this last portion of the *acmtable.vhd* file can be used to determine the constant values in the ACM write commands. The following example shows a sample of a CoreABC program in which location 1 of the ACM address space is initialized with 83h, as defined in the above *acmtable.vhd*:

```
// The following example assumes that CoreAI is in slot 0 of the APB and CoreABC is the
// bus master. Refer to the table "CoreAI Internal Register Address Map" in the CoreAI
// Handbook for ACM address mapping. Write 1 (as ACM ADDRESS) to address 0x04 of the
// CoreAI register space.
APBWRT DAT 0 0x04 1
// Write 0x83 into ACM DATA of CoreAI register space. This will result in writing 0x83
// into address 1 of ACM address space.
APBWRT DAT 0 0x08 0x83
CALL $WAIT ACM_WRITE
$WAIT_ACM_WRITE
  // Read ACM STATUS register of CoreAI
  APBREAD 0 0x00
  // Check to see if bit 4 of ACM status is cleared (write busy)
  BITTST 4
  // Remain in the loop if bit 4 is not cleared yet
  JUMP IFNOT ZERO $WAIT_ACM_WRITE
  RETURN
```

In the above example, the WAIT_ACM_WRITE function ensures that the write into ACM register space is completed before proceeding to the next ACM write instruction.

The *acmtable.vhd* file represents a MUX architecture where the output value is determined by the corresponding ACM address, which acts as the select lines. If the user intends to use the APBWRT ACM command in the design, this MUX can be implemented in the FPGA fabric (using tiles) and can be controlled by the CoreABC program to initialize ACM registers.

The following example shows the usage of the APBWRT ACM instruction in CoreABC to initialize the ACM using ACM table files:

```
// The following example assumes that CoreAI is in slot 0 of the APB and CoreABC is the
// bus master. Refer to the table "CoreAI Internal Register Address Map" in the CoreAI
// Handbook for ACM address mapping.
// Only Analog Quads are being initialized in this example (no RTC). According to the
// "ACM Address Map for Configuring Analog Quads and RTC" table in the CoreAI
// Handbook, the maximum ACM size for configuration is assumed to be 28h.
```

```
$WaitRegProg
CALL $WaitACMReady
// Write accumulator value in the ACM Address register of CoreAI
APBWRT ACC 0 0x04
// Write the value from acmtable file into ACM Data register and
// start an ACM write
APBWRT ACM 0 0x08
// Increment accumulator
INC
// Compare accumulator to 0x28
CMP 0x28
JUMP IFNOT ZERO $WaitRegProg

$WaitACMReady
  PUSH
$WaitACMReady1
  APBREAD 0 0x00
  AND 0x001C
  JUMP IFNOT ZERO $WaitACMReady1
  POP
  RETURN
```

In the above example, the CoreABC accumulator sweeps from address 0x00 to 0x28 of the ACM address space, and for each of these addresses, the APBWRT ACM command writes the appropriate values to the ACM data register space for initialization of all Analog Quads.

ADC Configuration and Calibration

ACM initialization is only used to configure the Analog Quads and/or RTC block. The ADC configuration (Mode, TVC, STC, etc.) is done through several inputs to the Analog Block. When configuring CoreAI in SmartDesign, the ADC settings can be set to be hardwired or register-controlled. When an ADC setting is selected to be hardwired, there is no need to initialize that setting before the operational phase of the design. However, when the ADC setting is selected to be software register-controlled, the settings are driven by a set of registers generally labeled as ADC control registers in CoreAI.

The most important configuration settings are in ADC control registers 1 and 2. The only configuration setting in ADC control register 2 is the STC value. Since this register also controls the ADCSTART and CHNUMBER inputs to the ADC, it will be written to during the operational phase. Therefore, there is no need for an initialization operation on STC (when selected to be register-controlled) before entering the operational phase of the design. The desired settings need to be written to ADC control register 1 to configure (TVC, PWRDOWN, VAREFSEL, and MODE). This only needs to happen once after each power-up, similar to ACM initialization.

ADC will self-calibrate after device power-up and after deactivation of ADCRESET, if ADCRESET is applied.

When the ADC is in calibration, bit 15 of the ADC status register will be set to 1 to flag that the ADC is busy calibrating itself. When the calibration is completed, this bit will be reset to 0.

Note: Both ADC calibration and ACM initialization (see "ACM Initialization" on page 258) should be completed after power-up before the Analog Block can properly function in the user's design.

It is common design practice to issue an ADCRESET request and check the status of bit 15 of ADC status register before entering the operation phase of the design. When this bit is cleared, the design enters the operation phase.

In the following example, CoreABC issues an ADCRESET request and then checks the status of the ADC for completion of calibration:

```
// In this example, ADC_STATUS and ADC_CTRL1 represent the address of the ADC status
// register and ADC control register 1, respectively, in the CoreAI address mapping.
// It is also assumed that CoreAI is in slot 0 of the APB.
// Write 0x0040 into ADC_CTRL1 register space of CoreAI
APBWRT 0 ADC_CTRL1 0x0040
$WaitCalibrate
    // Read ADC_STATUS register space of CoreAI
    APBREAD 0 ADC_STATUS
    AND 0x8000
    JUMP IFNOT ZERO $WaitCalibrate
```

Implementing Voltage Monitoring Applications

After completion of both ACM initialization and ADC calibration, the design enters the functional stage. This section describes how to implement a voltage monitoring application and provides design techniques to enhance sampling rate. The voltage monitoring operations are as follows:

- Selecting the analog voltage input pin to be monitored
- Sampling the voltage on the selected pin
- Translating the ADC output results into application-specific data
- Implementing a digital low-pass filter (or averaging) for voltage monitoring (optional)
- Implementing a sampling sequence

Channel Selection, ADC Sample, and Conversion Request

Voltage monitoring channel selection, requesting the ADC to start a sample, and conversion are all implemented through ADC control register 2 in the CoreAI register space. This register also controls the STC value input to the ADC. If, during the configuration of CoreAI in SmartDesign, the STC input to the ADC is set to be hardwired, bits 7–0 of ADC control register 2 will be considered as “don’t care.” A request to the ADC to sample and convert a specified channel is performed by activating ADCSTART. Since ADCSTART and CHNUMBER are both controlled by ADC control register 2, these two signals will be fed to ADC at the same time. Note that the ADCSTART bit is self-clearing and will be cleared to 0 after the user writes 1 to it to request an ADC start. The following example shows a CoreABC instruction in which the ADC is requested to sample and convert channel 2, with STC set to 4:

```
// In this example, ADC_CTRL2 represents the address of ADC control register 2 in the
// CoreAI address mapping. It is also assumed that CoreAI is in slot 0 of the APB.
APBWRT 0 ADC_CTRL2 0x2204
```

Obtaining Results from the ADC Output

The ADC status register in CoreAI contains status bits for the ADC and the ADC output results. Immediately after issuing an ADCSTART request, the ADC starts sampling the selected channel. During this period, the SAMPLE bit (bit 14) of the ADC status register will be set HIGH. When sampling is completed, the ADC enters conversion mode. In this mode, the SAMPLE bit will be cleared and the BUSY bit (bit 13) will be set HIGH. When the conversion is completed, the BUSY bit will be cleared, the ADC output will be placed on the RESULTS bits (bits 11 to 0), and the DATAVALID bit (bit 12) will be set HIGH. In a typical voltage monitoring application, the only status bit that needs to be monitored after issuing an ADCSTART request is the DATAVALID bit. Once DATAVALID is set HIGH, the RESULTS bits are ready to be used by the microprocessor/microcontroller. Depending on whether the ADC is configured to operate in 12-, 10-, or 8-bit mode, the ADC output will be stored in RESULTS[11:0], RESULTS[11:2], or RESULTS[11:4], respectively.

The following example shows a program routine in CoreABC instructions that continuously checks the status of the DATAVALID pin after issuing an ADCSTART request. Once DATAVALID is HIGH, it will clear all bits of the read value except the ADC RESULTS bits.

```
// In this example, ADC_STATUS represents the address of ADC status register in the
// CoreAI address mapping. It is also assumed that CoreAI is in slot 0 of the APB and
// that ADC is configured in 8-bit mode.
$ADC_Wait
  APBREAD 0 ADC_STATUS
  // Check to see of bit 12 of ADC_STATUS register is set or not
  BITTST 12
  // Remain in the loop if bit 12 is not set to 1
  JUMP IF ZERO $ADC_Wait
  AND 0x0FF0
  RETURN
```

Instead of continuous reading of the ADC_STATUS register to check DATAVALID, ADC_STATUS can be fed to the microprocessor as an interrupt and set HIGH. Then the microprocessor can read from the ADC_STATUS register to obtain RESULTS. For more information, refer to the "ADC Configuration and Calibration" section on page 261.

Sample Sequencing

When the microprocessor receives the latest results from the ADC status register, the microprocessor can issue another ADCSTART request on the same channel (to achieve more sampling on a particular channel) or a different channel. The desired sampling sequence can be achieved by writing the appropriate value to the CHNUMBER bit of ADC control register 2 when issuing an ADCSTART request. Refer to the *Sample Sequencing Overview* and *Sample Rate and Sample Sequence Calculation* sections of the "Designing the Fusion Analog System" section on page 233 for background information on sampling theory.

Sample Averaging

In applications where the measured voltage line is expected to be noisy or where voltage variations are too fast for the application to respond to, it is recommended to take multiple samples from a voltage and use the computed average value of the measured samples as representative of the sampled voltage. The user can choose the filtering factor by deciding how many samples need to be taken and averaged to acquire a single data point for processing. The higher the filtering factor, the lower the effective sampling rate will be.

The following example shows a CoreABC program in which a low-pass filter on the voltage measured from channel 2 is implemented with a filtering factor of four. In other words, four samples are taken from the single channel (v1, v2, v3, and v4) and averaged: (v1 + v2 + v3 + v4) / 4. The ultimate result to be used in the rest of the program is stored in address 0 of the internal memory.

```
// In this example, it is assumed that CoreAI is in slot 0 of the APB. Also,
// ADC_CTRL2 represents the ADC control register 2 address in the CoreAI register
// space. Also, this example calls the $ADC_WAIT function as described in
// "Obtaining Results from the ADC Output" on page 263. In the following example,
// the 8-bit output of the ADC is averaged using four samples, and the final average
// value is stored in internal RAM address 0 when the function is completed.
// Load Z register (used as loop counter) with value 4
LOADZ DAT 4
// Write 0 to internal RAM address 0
RAMWRT 0 DAT 0
$SAMPLE_FILTER
  // Issue ADC sample and Conversion Request
  APBWRT 0 ADC_CTRL2 0x2204
  CALL $ADC_WAIT
  // Divide by 4 and add to previous values
  SHR0
  SHR0
  // Add content of internal RAM address 0 to accumulator
  ADD RAM 0
  // Write accumulator value to internal RAM address 0
  RAMWRT 0 ACC
  // Decrement the Z register value (loop counter)
  DECZ
  // Check to see of Z register (loop counter) is 0 or not
  JUMP IFNOT ZZERO $SAMPLE_FILTER
$DONE
HALT
```

Once the four samples are averaged, the sampled values are discarded. Depending on the application and sampling sequence, the user can only discard the oldest sampled value and keep the most recent values to be used for averaging with the next sampled data.

Techniques to Enhance Design Performance/Throughput

After each ADCSTART request to the Analog Block, there is a period of time (duration of ADC sample and conversion) during which the design's microprocessor continuously checks the ADC status register to indicate when the ADC is done and data is ready to be fetched from RESULTS bits. The processing throughput of the design can improve significantly if the microprocessor/microcontroller can perform other necessary tasks while waiting for the ADC to complete its cycle. There are two general methods to do this, depending on the application's requirements:

- After activating the ADCSTART input to the Analog Block and requesting an ADC sample and conversion to start, the design's microprocessor can go on performing other tasks that do not need the results of the ADC conversion. When these tasks are completed (or periodically), the microprocessor can return and check the ADC status register for DATAVALID assertion. The drawback of this method is that the ADC maximum sampling rate capability may be compromised. In other words, the ADC might complete conversion and sit idle prior to the microprocessor's checking the ADC status register.

- The required flag in the ADC status register (DATAVALID) can be used as an interrupt input to the microprocessor. In this case, it can be assured that the microprocessor will attend to the ADC, read the data output, issue another ADCSTART request, and continue with the rest of the tasks until the next interrupt from the DATAVALID bit. The time the ADC is idle is reduced to a minimal level, enhancing the sampling rate of the ADC at the given operating frequency.

The maximum frequency of ADCCLK is 10 MHz, and the relationship between ADCCLK and SYSCLK (the system clock) is governed by the TVC setting. Therefore, maximum ADCCLK frequency is achieved at certain frequencies. For example, with a SYSCLK frequency of 40 MHz and TVC set to 0, the user can achieve 10 MHz on the ADCCLK. On the other hand, if SYSCLK is at 50 MHz, the maximum achievable ADCCLK frequency is 6.25 MHz (TVC = 1). Higher SYSCLK frequencies do not necessary translate into higher ADCCLK frequencies. This is due to the fact that ADCCLK is bounded by a 10 MHz limit and is also governed by the TVC value.

Implementing Current Monitor Applications

Regardless of the type of the signal being measured (i.e., voltage, current, or temperature), the ACM needs to be initialized at power-up to configure the Analog Quads. The ADC may need to be configured and calibrated on device power-up, a global reset event, or at the user's discretion. "Analog Configuration MUX Initialization" on page 258 and "ADC Configuration and Calibration" on page 261 still apply when a current monitoring application is being implemented.

The current monitoring procedure is very similar to the voltage monitoring steps described in "Implementing Voltage Monitoring Applications" on page 262. The only additional step required to perform sampling on a current monitor channel is the accurate stimulation of the CMSTB inputs to the Analog Block. For current sampling on a desired channel, CMSTB should be kept LOW for more than the TMPWC value (refer to the values in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet) to discharge previous measurements, and then HIGH for at least the TMPWC value prior to the ADCSTART request. Figure 10-4 illustrates the assertion/deassertion of the CMSTB input of a specific channel prior to starting sampling.

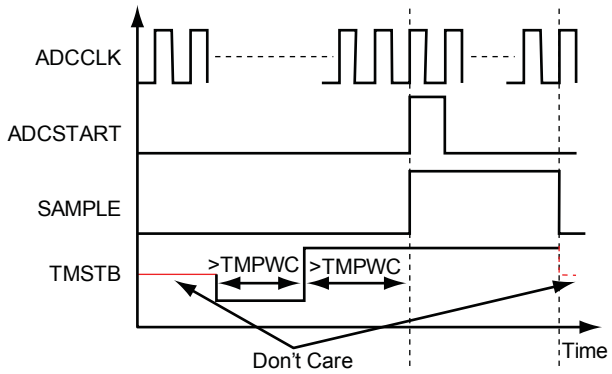


***Figure 10-4* • Assertion/Deassertion of CMSTB Input**

The user should not assert another ADCSTART prior to the completion of current ADC conversion. When sampling a current, ADCSTART cannot be asserted for at least for 2 × TMPWC after the previous ADCSTART. If the next ADCSTART is also a current monitor (or temperature monitor), there should be at least another TMPWC waiting period during which CMSTB of the desired channel is kept LOW before assertion of ADCSTART. The selected channel's CMSTB should be deasserted after the ADC has completed sampling the channel (as shown in Figure 10-4). CMSTB can be kept HIGH (after assertion of ADCSTART) until DATAVALID is asserted.

If CoreAI is configured in SmartDesign to have a hardwired CMSTB input for the desired channels (HD_CMSTBn input), the user will need to stimulate CMSTB to fulfill the requirements discussed in this section.

In summary, the following are the necessary operations to implement current monitoring:

- Selecting the analog current input pin to be monitored
- Ensuring the corresponding CMSTB input pin is LOW for more than the required TMPWC value
- Asserting CMSTB HIGH for longer than TMPWC
- Issuing an ADCSTART request on the desired CHNUMBER
- Ensuring that CMSTB remains HIGH until sampling is completed or until assertion of DATAVALID
- Translating the ADC output results into application-specific data once DATAVALID is asserted
- Implementing a digital low-pass filter for voltage monitoring (optional)
- Implementing a sampling sequence (selecting the next channel)

Translating the ADC output results to the actual measured current is slightly different from the same operation in voltage monitoring. As shown in Figure 2-56 of the *Fusion Family of Mixed Signal Flash FPGAs* datasheet, when configured as current monitor, a fixed 10× prescaler is used to buffer the differential voltage measured across the external resistor into the ADC. Therefore, the maximum differential voltage that can be measured with the ADC (before overflow) is limited by the V_{REF} value (EQ 3):

Max. Differential Voltage across Resistor = I(max) × R = VREF / 10

EQ 3

where R is the external resistor value in ohms. Therefore, the measured current value, based on the ADC mode, can be calculated as shown in Table 10-1.

***Table 10-1* • ADC Output Translation in Current Monitoring Applications**

LSB for 8-Bit ADC (mA)	LSB for 10-Bit ADC (mA)	LSB for 12-Bit ADC (mA)
VREF / (10 × R × 0.255)	VREF / (10 × R × 1.023)	VREF / (10 × R × 4.095)

Implementing Temperature Monitor Applications

Regardless of the type of signal being measured (i.e., voltage, current, or temperature), the ACM needs to be initialized at power-up to configure the Analog Quads. The ADC may need to be configured and calibrated as well at device power-up, a global reset event, or the user's discretion. Therefore, the "Analog Configuration MUX Initialization" section on page 258 and the "ADC Configuration and Calibration" section on page 261 still apply when a current monitoring application is being implemented.

The temperature monitoring procedure is very similar to the current monitoring steps described in the "Implementing Current Monitor Applications" section on page 265. The only difference is the stimulation of the TMSTB input for the temperature monitoring channels instead of CMSTB in current monitoring. For accurate temperature sampling on a desired channel, TMSTB should be kept LOW for longer than the TMPWT value to discharge previous measurements, and then HIGH for longer than the TMPWT value prior to the ADCSTART request. The TMSTB input of the selected temperature monitoring input should remain HIGH at least until the completion of sampling. Figure 10-5 on page 267 illustrates the assertion/deassertion of the TMSTB input of a specific channel prior to starting sampling. TMSTB can be kept HIGH (after assertion of ADCSTART) until DATAVALID is asserted.

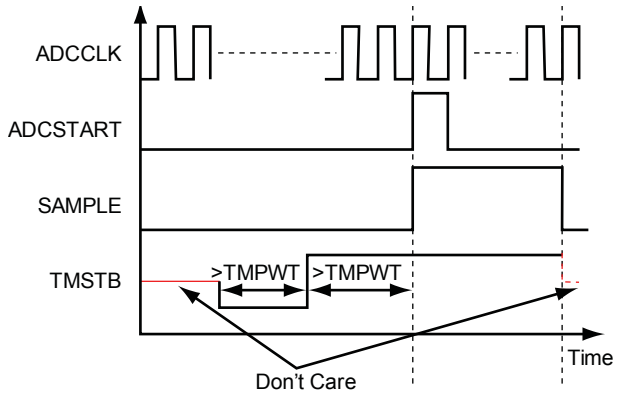


***Figure 10-5* • Assertion/Deassertion of TMSTB Input**

Since, in CoreAI, the TMSTB inputs and ADCSTART are in two different register spaces, they cannot be asserted at the same time (see Figure 10-5). Assert TMSTB HIGH first and then issue ADCSTART. If CoreAI is configured in SmartDesign to have a hardwired TMSTB input for desired channels (HD_TMSTBn input), stimulate TMSTB accordingly.

In summary, the following are the necessary operations to implement temperature monitoring:

- Selecting the analog temperature input pin to be monitored
- Ensuring the corresponding TMSTB input pin is Low for more than the minimum value of TMPWT
- Asserting TMSTB HIGH for more than the minimum value of TMPWT
- Issuing an ADCSTART request on the desired CHNUMBER
- Ensuring that TMSTB remains HIGH until completion of sampling or assertion of DATAVALID
- Translating the ADC output results into application-specific data once DATAVALID is asserted
- Implementing a digital low-pass filter for voltage monitoring (optional)
- Implementing a sampling sequence (selecting the next channel)

Translating the ADC output voltage in temperature monitoring applications depends on the V_{REF} value of the ADC, the ADC mode, and the characteristics of the external (temperature sensor) diode. The relationship between the measured voltage and the external temperature is described in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet.

Implementing Gate Driver Applications

Implementing gate driver applications is different from implementing voltage, temperature, and current monitoring because gate drivers are outputs, and driving them does not require interaction with the ADC. However, the ACM initialization procedure, described in the "Analog Configuration MUX Initialization" section on page 258, is still necessary to configure the selected Analog Quads as gate drivers with specific parameters, such as drive strength and polarity. Gate driver applications can be implemented as software-/register-driven or hardware-driven.

Software-Controlled Gate Drivers

When a specific gate driver is configured in CoreAI to be software-controlled, the corresponding AG output pad (analog gate driver) follows the contents of the corresponding bit in CoreAI ADC control register 5. In this approach, the microprocessor can turn the external MOSFET (connected to the AG output pad) off or on by writing 1 or 0 to the corresponding bit of ADC control register 5. A typical use model for a software-driven gate is in power sequencing applications, where the system management processor turns the voltage rails on/off in the desired sequence. The following example shows a sample CoreABC program in which three gate drivers are turned on sequentially.

```
// In this example, it is assumed that CoreAI is in slot 0 of the APB. Also, ADC_CTRL5
// represents the ADC control register 5 address in the CoreAI register space.
```

```
APBWRT DAT 0 ADC_CTRL5 0x0001
APBWRT DAT 0 ADC_CTRL5 0x0002
APBWRT DAT 0 ADC_CTRL5 0x0003
```

Hardware-Controlled Gate Drivers

When a specific gate driver is configured in CoreAI to be hardware-controlled, the corresponding AG output pad (analog gate driver) follows the corresponding HDGDON input of CoreAI. When designing with SmartDesign, the selected HDGDON inputs can be added to the top-level ports to be driven by the FPGA fabric.

In a typical use model for hardware-controlled gate drivers, the AG pad drives the external MOSFET gate with a pulse width modulation (PWM) signal.

Design Example

This section includes a practical example of a design using CoreABC and CoreAI to implement the following applications:

- Voltage, current, and temperature monitors
- Gate drivers

Functionality

Figure 10-6 illustrates the top-level functionality of the example design.

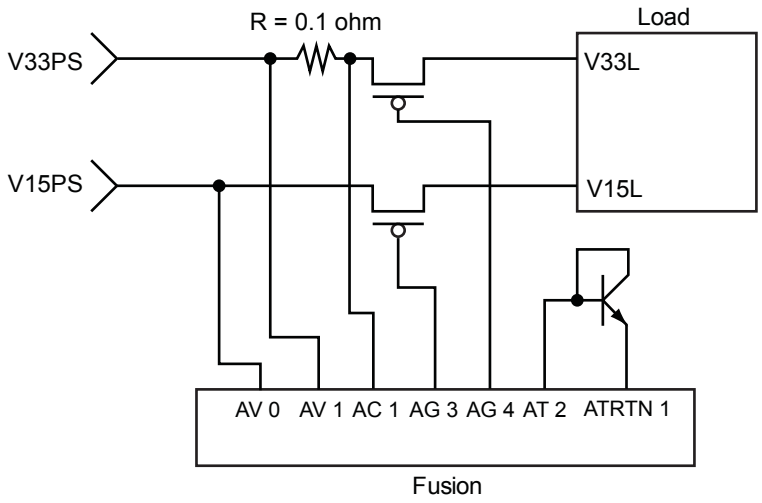


***Figure 10-6* • Top-Level Functionality of the Example Design**

Figure 10-6 shows two voltage supplies in the system: V15PS supplying 1.5 V and V33PS supplying 3.3 V. Two voltage rails (V15L and V33L) drive the power inputs of a load. These voltage rails are powered up through two MOSFETs that are controlled by the AG3 and AG4 gate drivers of the Fusion device. The current on the 3.3 V supply rail is measured across a 0.1 Ω resistor.

Figure 10-7 shows the flow chart of the example design, implemented in Fusion using CoreABC and CoreAI.

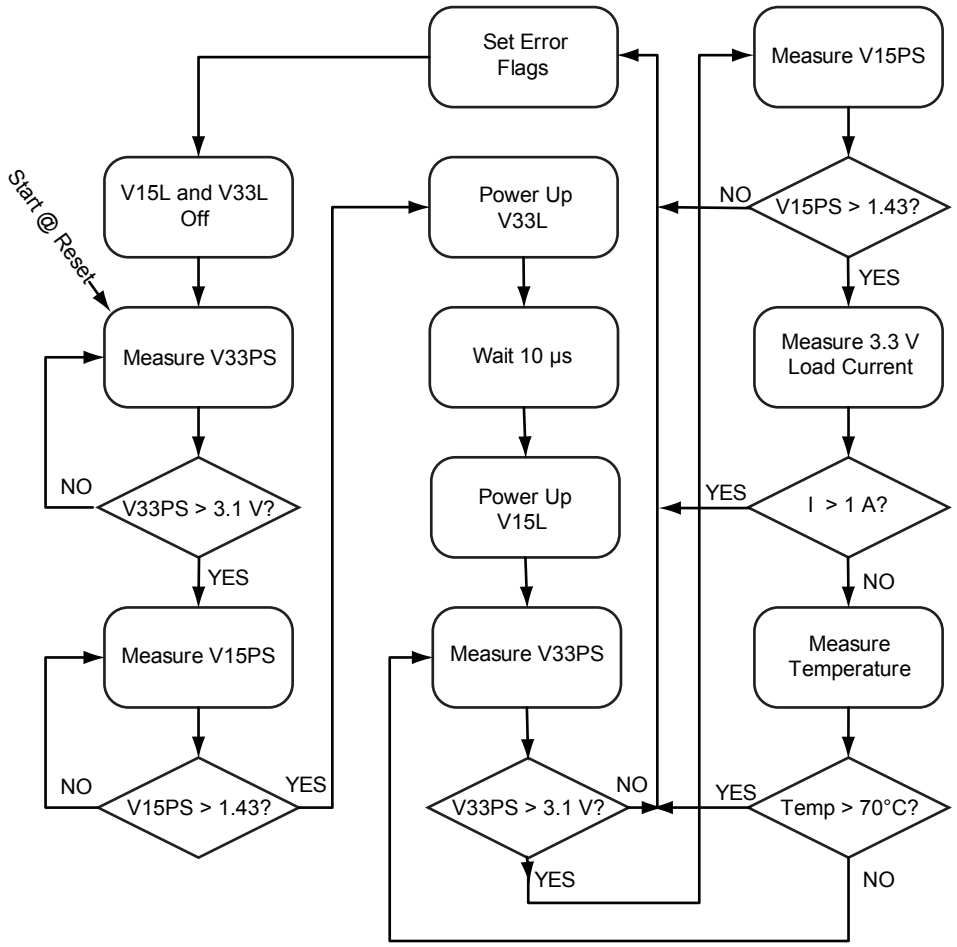


***Figure 10-7* • Flow Chart of Example Design**

The operational phase of the design starts with measuring the power supplies and ensuring that they satisfy the minimum requirements of the load voltage rails. Once the power supplies are up and running, the V33L and V15L rails are powered up sequentially (3.3 V first and 1.5 V second after 10 µs). After powering the rails, the design continuously monitors the 3.3 V supply, 1.5 V supply, 3.3 V rail current, and operating temperature (defined sampling sequence). If no abnormal condition occurs, the design remains in this loop monitoring the operating conditions until one of the supplies is turned off externally. In case of an abnormal situation (e.g., current being more than 1 A), the design sets error flags, turns off both V33L and V15L, and checks for supply lines, and if they are still up and running, it attempts to power up the load again.

Implementation in a Fusion Device

This design example is implemented using CoreABC as the core microcontroller and CoreAI as the interface to the Fusion Analog Block. The steps below describe the CoreABC and CoreAI settings and connections and the CoreABC program that implements the flow chart illustrated in Figure 10-7 on page 269.

Building the Example Design System in SmartDesign

CoreABC and CoreAI in this example are connected together using an APB, as shown in Figure 10-8.

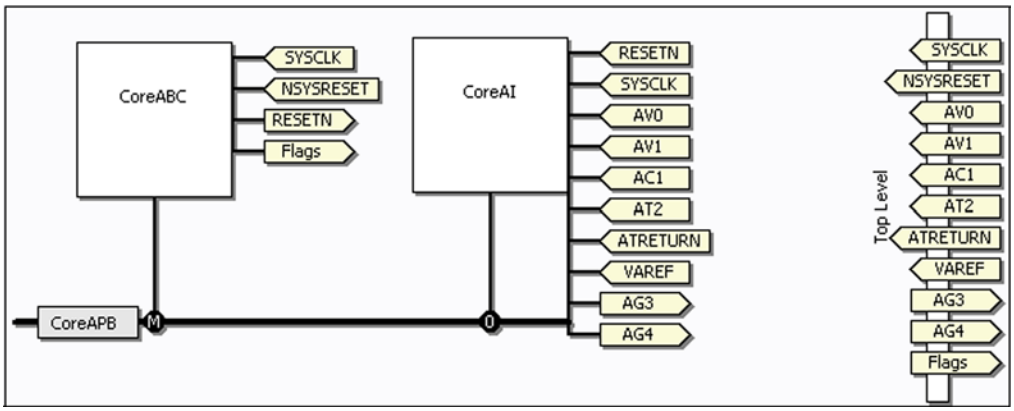


***Figure 10-8* • Implementation of Example Design in SmartDesign**

The following are the major settings of the CoreABC and CoreAI configuration in SmartDesign:

CoreABC Settings:

- APB address bus width: 8
- APB data bus width: 16
- Number of CoreABC outputs: 1
- Instruction store: soft

CoreAI Settings:

- ACM clock divider: 4
- AV0: 0–2 V analog input
- AV1: 0–4 V analog input
- AC1: current monitor
- AT2: temperature monitor
- AG3: software-driven
- AG4: software-driven
- VAREF: internal 2.56 V
- ADC mode: fixed 12-bit mode
- TVC: fixed to 0
- STC: register-controlled
- APB interface width: 16 bits

The above settings result in the memory address map shown in Table 10-2 on page 271.

The memory of the design in SmartDesign can be obtained from the following location:

<SmartDesign Project Directory>/SoftwareExport/<projectname>/memorymap.html

***Table 10-2* • Memory Map of the Example Design**

Address	Type	Width	Reset Value	Name	Description
base address + 0x00	Read/write	16	0x0	ACM_CTRL_STATUS	ACM Control Status Register
base address + 0x04	Read/write	16	0x0	ACM_ADDR	ACM Address Register
base address + 0x08	Read/write	8	0x0	ACM_DATA	ACM Data Register
base address + 0x0C	Read/write	16	0x0	ADC_CTRL_1	ACM Control Register 1
base address + 0x10	Read/write	16	0x0	ADC_CTRL_2	ACM Control Register 2
base address + 0x14	Read/write	16	0x0	ADC_CTRL_3	ACM Control Register 3
base address + 0x18	Read/write	16	0x0	ADC_CTRL_4	ACM Control Register 4
base address + 0x1C	Read/write	16	0x0	ADC_CTRL_5	ACM Control Register 5
base address + 0x20	Read-only	16	0x0	ADC_STATUS	ADC Status Register
base address + 0x24	Read-only	16	0x0	READ_FIFO_DATA_OUTPUT	Read FIFO Data Output
base address + 0x28	Read-only	8	0x0	READ_FIFO_STATUS	Read FIFO Status
base address + 0x2C	Read/write	16	0x0	IRQ_ENABLE	Interrupt Enable Register
base address + 0x30	Read-only	16	0x0	IRQ_STATUS	Interrupt Status Register

The initialization values for the ACM register space used to configure the Analog Quads can be calculated as discussed in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet or obtained from the *acmtable.vhd* (or *acmtable.v*) file. In this example, the ACM initialization is performed by using the APBWRT instruction to write desired values to the corresponding ACM register space. As discussed in "ACM Initialization Specific to CoreABC" on page 259, designers can take advantage of the APBWRT ACM command and write to the ACM registers with the values defined in the *acmtable.vhd* (or **.v*) file.

CoreABC Instruction Program

The CoreABC program in the Fusion FPGA Fabric User's Guide design files executes the functionality depicted in Figure 10-7 on page 269. The SmartDesign project of this example can also be found in the Fusion FPGA Fabric User's Guide design files. $Wait_10us is called whenever needed to ensure that the elapsed timing for TMSTB and CMSTB activation/deactivation is more than the minimum requirement. Inside the function, a loop counter is loaded with a count value and then decremented until it reaches zero. The count value should be chosen such that the elapsed time after exiting the function is 10 µs. Each of the instructions used in the $Wait_10us routine takes three clock cycles to execute. Assuming SYSCLK runs at 40 MHz, the count value can be obtained from EQ 4:

$$\text{Delay} = (3 + 3 + \text{count_value} \times (3 + 3) + 3) / f$$

EQ 4

where f = 40 MHz.

Instructions in this example are based on CoreABC v2.1. Some instructions may vary (e.g., loop instructions used in the $Wait_10us routine) if a different version of CoreABC is used.

In a real application design, the $Wait_10us routine can be replaced by a set of instructions that perform part of the design functionality during the required TMPWC or TMPWT period. This increases the processing performance of the design.

HDL Implementation

After building the core of the design in SmartDesign, the design has be to be completed in HDL where the microprocessor and the CoreAI system (along with any other peripherals in the system) are imported into Libero SoC in HDL format and connected to the rest of the design, to go through the rest of the

FPGA design flow. The HDL code in the Fusion FPGA Fabric User's Guide design files is the top-level wrapper used in the example design.

In this example, the CoreABC instructions are stored in soft mode. Therefore, the instructions can be stored in the Fusion embedded flash memory or in external memory. The instructions are loaded into SRAM from the flash memory at power-up (initialization), and the microprocessor runs off the SRAM; CoreABC offers a feature to run directly off the flash memory as well. In the HDL code, Init_Block represents the initialization client of the Fusion embedded flash memory. An embedded CCC is used to generate two clock signals: a 10 MHz clock to drive the embedded flash memory and the initialization process to load the SRAM with CoreABC instructions, and a 40 MHz clock used as the system clock for the operational phase of the design.

Designing with the RTC

CoreAI enables you to interface with the Fusion RTC. When configuring CoreAI in SmartDesign, you need to specify the RTC and its parameters. CoreAI provides the necessary I/Os, as described in the *CoreAI Handbook*. ACM address space 0x40 to 0x58 is used to read or write specific RTC parameters, such as count or match values. Reading from and writing to these registers is no different from other ACM read/writes described in the "Analog Configuration MUX Initialization" section on page 258. The only difference is the targeted address of the ACM register space.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
December 2011	Occurrences of CoreConsole were replaced with SmartDesign (SAR 24945).	N/A
July 2010	This chapter is no longer published separately with its own part number and version but is now part of the Fusion FPGA Fabric User's Guide.	N/A
v1.0 (December 2007)	Corrected the operations given in the "Implementing Temperature Monitor Applications" section.	266

11 – Interfacing with the Fusion Analog System: IP Interface

Fusion Analog System Soft IP Design

The Analog System Soft IP Design Flow is an IP-based design method for HDL designs that establishes a backbone to interconnect the Fusion FPGA fabric, the Analog System, the embedded flash memory block, and other peripherals. The Analog System soft IP includes an Analog-to-Digital Converter Sample Sequence Controller (ASSC) that sets up the ADC sample sequence, a System Monitor Evaluation Phase State Machine (SMEV) that compares the ADC results to user-defined threshold values, and a System Monitor Transition Phase State Machine (SMTR) that asserts threshold flags accordingly. Using the IP configuration catalog in Microsemi Libero SoC software, the user creates and configures the VHDL or Verilog Analog System soft IP along with the Flash Memory Analog System Client. The user can configure over/under threshold flags, acquisition time, filtering factor, and assert/deassert samples for analog inputs. More details are addressed in the "Basic Analog Block Settings" section on page 283. Once the IP is configured, the user instantiates the Analog System into HDL or builds the system in a SmartDesign project. Standard HDL design flow is used to complete the design, as described in the "Fusion Design Solutions and Methodologies" section on page 247.

When creating the Analog System in Libero SoC using the Analog System Builder (ASB), a configuration file is generated, and its data is stored in the spare pages within the embedded flash memory during FPGA programming. The Flash Memory Analog System Client is used to create the memory partitions to store this configuration data.

The Analog System uses the embedded flash memory to hold the nonvolatile configuration data for the analog subsystem. After power-up and during the initialization process, the flash memory is read and the data is stored in the Analog System's volatile register or RAM blocks within the analog subsystem. More information about the embedded flash memory system and clients is available in the "Fusion Embedded Flash Memory Blocks" section on page 137.

Note: Any time the Analog System is regenerated, the Analog System Client must also be regenerated from the Flash Memory System Builder in Libero SoC.

The Analog System Soft IP Design Flow offers a number of advantages to users. All the necessary soft IP cores are free. Sample sequence control, averaging/filtering, and threshold response functions are built-in and specified by an intuitive GUI in Libero SoC. IP configuration and connectivity are tightly integrated into SmartDesign and Libero SoC, enabling users to rapidly and seamlessly implement the complete analog and peripheral interface. Users do not have to write up their own code to control the analog and flash memory systems.

For processor- or microcontroller-based designs, a more efficient implementation can be realized through CoreAI (Analog Interface) and the methods discussed in the "Interfacing with the Fusion Analog System: Processor/Microcontroller Interface" section on page 253.

System Overview – Interface Components

Figure 11-1 gives an overview of the interface between the Analog System soft IP, the ADC, the clock circuitry, the device RAM, and the embedded flash memory. As shown in Figure 11-1, there are three Analog Interface soft IP blocks and several blocks that interact directly or indirectly with the Analog Interface soft IP blocks. The Analog Interface soft IP components are listed in Table 11-1 on page 275, and the components that interact with these soft IP components are listed in Table 11-2 on page 275. Detailed descriptions for each of the components listed in Table 11-1 on page 275 are included in the "SmartGen Soft IP Blocks" section on page 277.

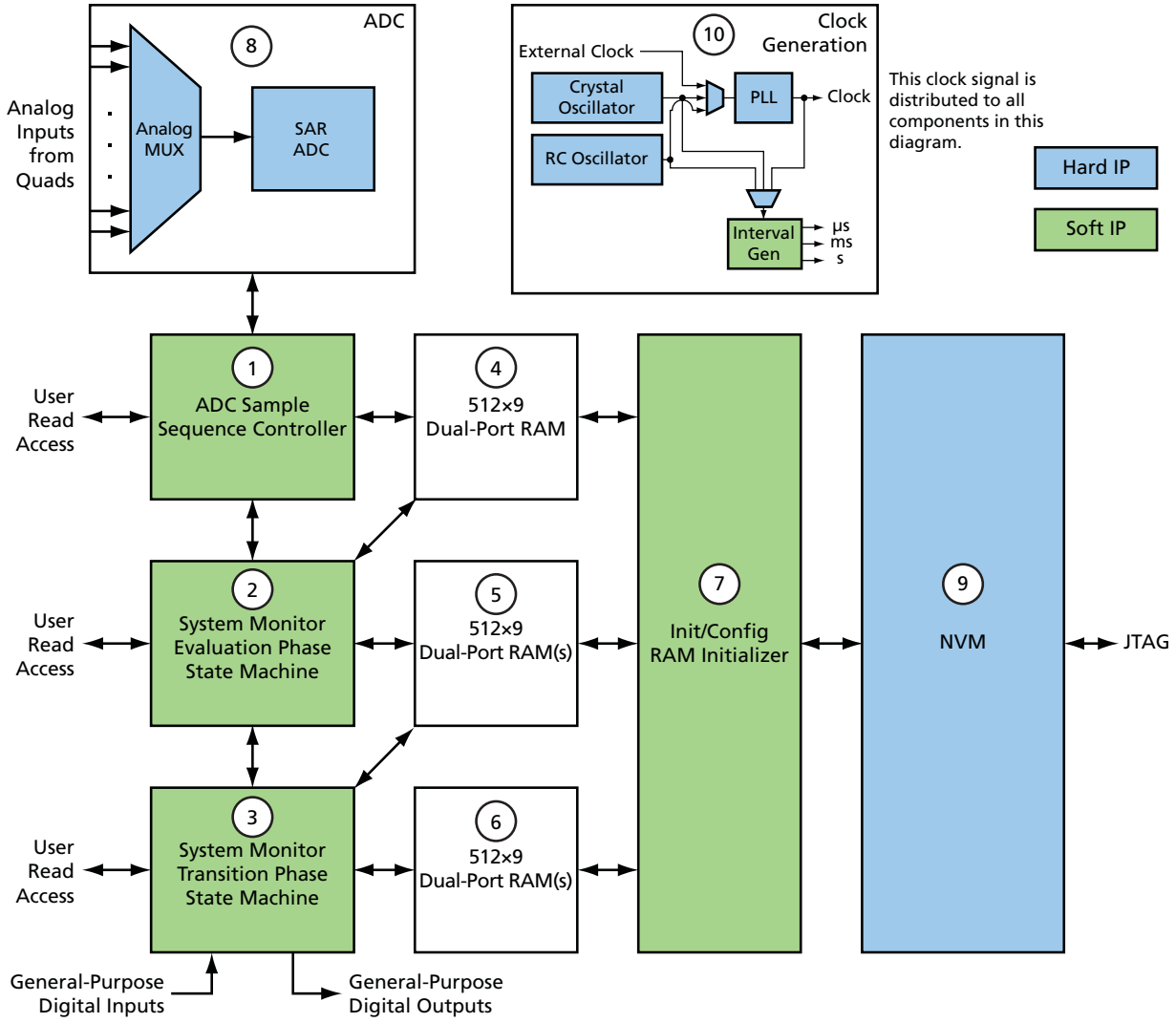


***Figure 11-1* • Fusion Interface Components (relative to Analog Interface soft IP)**

***Table 11-1* • Fusion Analog Interface Soft IP Components**

Number	Description
1	Analog-to-Digital Converter Sample Sequence Controller: The ASSC is a configurable sequencer that sets up the order of samples from the ADC, controls various measurement parameters of the ADC samples, and sends control commands to the ADC.
2	System Monitor Evaluation Phase State Machine: The SMEV reads ADC samples from the result locations in the ASSC RAM and compares the results to user-defined threshold values. Comparison results are stored in SMEV RAM.
3	System Monitor Transition Phase State Machine: The SMTR reads from SMEV RAM and, for each enabled channel, checks comparison results (previously calculated by the SMEV block) and generates the threshold flags defined by the user in the Analog System Builder.

***Table 11-2* • Fusion Components that Interact with Analog Interface Soft IP Components**

Number	Description
4, 5, 6	512×9 dual-port RAM blocks: These RAM blocks are used to store "program" sequences for the SMTR and SMEV. They are also used to store data samples calculated by the ADC and data values that control the operation of the ASSC. They are initialized by the Init/Config soft IP block, which transfers data from nonvolatile memory (NVM) after a system reset. Note that these RAM blocks can also be modified while the system is live to allow the user to perform real-time system debugging before committing resources to programming the NVM. This debugging feature is addressed by Synplicity® Identify software. For more information, refer to the Identify user's guide.
7	Init/Config RAM Initializer: The sole purpose of this soft IP block is to initialize the system RAM blocks after a system reset, by reading data from the NVM.
8	ADC: This analog-to-digital converter hard IP component is the main interface between the external analog voltage, current, and temperature sources, and the internal digital FPGA user logic (soft IP). It is selectable for 8-, 10-, or 12-bit operation.
9	NVM: The nonvolatile memory (flash) will be used, at minimum, to store program sequences for the ASSC, SMEV, and SMTR.
10	Clock Generation: Internal or external clock generation for digital system time base reference. The internal PLL, internal RC oscillator circuit, or external crystal oscillator circuit can be used in this process.

System Operation

The Analog Block (AB) System contains the Analog Block hard IP and the Analog Interface soft IP, which includes ASSC, SMEV, SMTR, and their corresponding SRAM blocks. The Flash Memory System contains the embedded flash memory hard IP block and the interface soft IP, or the Init/Config IP block. Figure 11-2 shows the generic connections between the Analog Block System and the Flash Memory System in the SmartGen soft IP design flow.

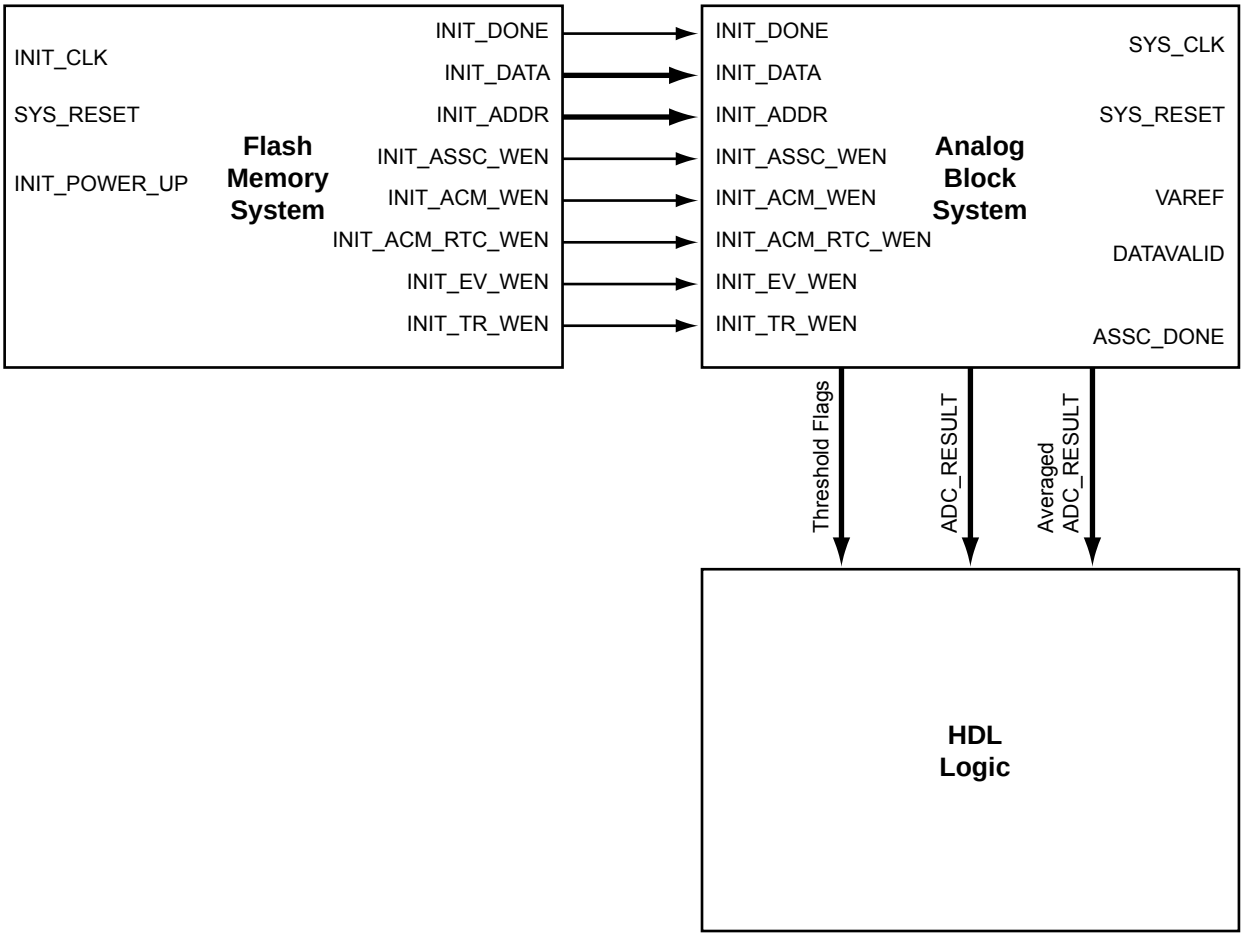


***Figure 11-2* • SmartGen Soft IP Design Flow – Generic Connections**

Initialization

The Init/Config soft IP is used to accomplish the initialization of the Fusion Analog Block. All user-defined Analog Block parameters are preprogrammed into the embedded flash memory and are loaded into the corresponding Analog System soft IP RAM blocks and Analog Configuration MUX (ACM) registers by the Init/Config IP during device power-up. For more information about Analog Client Initialization, refer to the "Fusion Embedded Flash Memory Blocks" section on page 137.

Sample and Convert

Once the initialization and calibration is done, or INIT_DONE and calibrate_o are asserted HIGH, the Analog System soft IP starts functioning. The ASSC IP controls the sample sequence, the SMEV applies a moving average to the ADC conversion output and compares the outputs with the preset threshold values, and the SMTR checks the comparison results and acts on them based on predefined behavior.

The user can also probe and process the ADC output directly and set up corresponding reactions. The two modes are described below.

Threshold Flags Operation Mode

The threshold values are preset during Analog System soft IP configuration in Libero SoC. The values are programmed into the embedded flash through the Analog System Client, then loaded into the SRAM during initialization. SMEV soft IP does the comparison between the average ADC results and the threshold values, and saves the comparison results to the SMEV SRAM. The SMTR soft IP reads the results from the SMEV SRAM and asserts or deasserts user-defined threshold flags, which are general-purpose outputs (GPOs) of the Analog System Block. The GPOs can be used by internal logic in the FPGA array, or they can trigger external I/Os directly.

ADC Result Direct Access Mode

Instead of reacting on the threshold value comparison result from the SMEV, users can directly access the ADC results and convert them to meaningful voltage, current, or temperature values and process these values for different purposes. To accomplish this, the user first needs to expose the necessary ports from the ASSC or SMEV IP through the advanced options in the Analog System Builder. The corresponding ports are listed in the ASSC and SMEV sections. Second, the user needs to build an interface to read out the valid ADC results from the ASSC or SMEV RAM. The basic logic of the interface is discussed in the CoreAI section of the "Interfacing with the Fusion Analog System: Processor/Microcontroller Interface" section on page 253. Sample code for fetching the ADC result is located in the "Sample Code" section on page 291. Once the ADC result has been fetched, it can be translated back to a voltage, current, or temperature value. Sample code for this is also provided in the "Sample Code" section on page 291.

SmartGen Soft IP Blocks

ADC Sample Sequence Controller (ASSC)

Function

The ASSC is a configurable sequencer that sets up the order of samples from the ADC, controls various measurement parameters of the ADC samples, and sends control commands to the ADC. It also contains multiplexer logic for reading from and writing to a 512×9 dual-port RAM block. In addition to controlling sequencing of ADC samples, the ASSC block performs digital post-scaling of the ADC samples, and ADC saturation detection, functions that were previously handled with additional soft IP blocks.

Note: The ASSC does not control the various prescaler and other signals within each Analog Quad (except for the current monitor and temperature monitor strobe connections). These are defined during soft IP configuration in Libero SoC and initialized via the Init/Config soft IP block into the ACM.

Interfaces

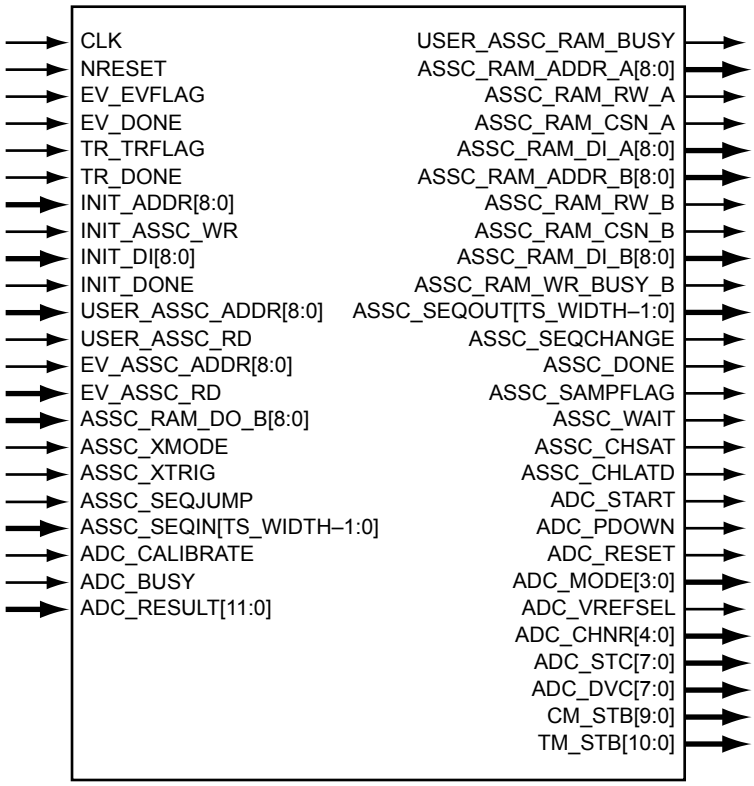


***Figure 11-3* • ASSC I/O Signal Diagram**

Note: All signals are active high (logic 1) unless otherwise noted. All port width specifications are in Verilog notation. Any alphabetic text within port width brackets indicates that the specified port width is controlled via a generic (VHDL) or parameter (Verilog) described in Table 11-3. All I/O signals are synchronous to the rising edge of the CLK signal unless otherwise noted.

The signals in Table 11-3 are the typical signals users should monitor in the simulation for the regular ADC conversion process.

***Table 11-3* • ASSC I/O Signal Descriptions**

Name	Type	Description
CLK	Input	System Clock: Reference clock for all internal logic (100 MHz maximum). This signal is connected to the top level as SYS_CLK.
NRESET	Input	Active-low asynchronous reset. This signal is connected to the top level as SYS_RESET.
INIT_ADDR[8:0]	Input	Init/Config RAM Address: These address signals come from the Init/Config soft IP block for writing to the 512×9 ASSC RAM.
INIT_DONE	Input	Init/Config Done: This static signal indicates that the Init/Config soft IP block has completed loading all of its clients (including the ASSC RAM block) from data stored in the internal Fusion NVM block(s).

***Table 11-3* • ASSC I/O Signal Descriptions (continued)**

Name	Type	Description
ASSC_DONE	Output	ASSC Done: This output indicates that the ASSC block has completed the current function and is either waiting for further action (e.g., the power-down or stop function) or is about to transition to the next sequence (e.g., the sample or calibration function). If the ASSC_DONE signal is active at the same time that the ASSC_SAMPFLAG signal is active, this indicates that a sample function has just completed and will cause the SMEV block to commence with evaluation sequences followed by transition sequences within the SMTR block; otherwise, if the ASSC_DONE signal is active and the ASSC_SAMPFLAG signal is inactive (logic 0), the SMEV block will not have evaluation sequences for the current sequence (timeslot). This output is connected to the SMEV block and may optionally be used external to the Analog Interface soft IP blocks by the user.
ADC_CALIBRATE	Input	ADC Calibration: This signal from the ADC indicates that internal calibration is currently in effect. This input connects to the calibrate_o output from the ADC.
ADC_RESULT[11:0]	Input	ADC Result: These signals comprise the conversion result from the ADC. In 12-bit mode, the ADC uses all bits; in 10-bit mode, it uses bits 11:2; and in 8-bit mode, it uses bits 11:4. All unused ADC bits are set to logic 0 when in 10-bit or 8-bit mode. These inputs connect to the result_o[11:0] outputs from the ADC.
The following signals are used for jump sequence control.		
ASSC_XMODE	Input	External Trigger Mode: If this input is logic 1, the ASSC uses the ASSC_XTRIG signal to transition to and complete the current sequence timeslot. If this input is logic 0 (default operation for automated sequencing), the internal timeslot counter is used to automatically advance to the next sequence number. This input can come from the SMTR (from one of the GPO signals) or user logic external to the Analog Interface soft IP blocks, or can be statically tied off to logic 0 or logic 1.
ASSC_XTRIG	Input	External Trigger: If the ASSC_XMODE input is logic 1 and this input is held at logic 1 for exactly one clock cycle, the ASSC block will transition to and complete the current sequence. If the ASSC_XMODE input is logic 0 (default operation for automated sequencing), this input is ignored. This input can come from the SMTR (from one of the GPO signals) or user logic external to the Analog Interface soft IP blocks, or can be statically tied off to logic 0. If this signal is used to control external triggering, the user should monitor the ASSC_DONE signal to know after which point the ASSC_XTRIG will again have effect.

***Table 11-3* • ASSC I/O Signal Descriptions (continued)**

Name	Type	Description
ASSC_SEQJUMP	Input	Sequence Jump Enable: Setting this signal to logic 1 jumps to the sequence number indicated on the ASSC_SEQIN[TS_WIDTH–1:0] input pins after the current sequence timeslot has completed. This input can come from the SMTR (from one of the GPO signals) or user logic external to the Analog Interface soft IP blocks, or can be statically tied off to logic 0.
ASSC_SEQIN[TS_WIDTH–1:0]	Input	Sequence Number In: These inputs are used in conjunction with the ASSC_SEQJUMP signal to jump to a particular sequence number from the current sequence after the current sequence timeslot has completed. The SMTR sets these signals. These inputs can come from the SMTR (from several of the GPO signals) or user logic external to the Analog Interface soft IP blocks, or can be statically tied off to any combination of logic 0 and logic 1 values.
ASSC_SEQOUT[TS_WIDTH–1:0]	Output	Sequence Number Out: These outputs denote the current sequence timeslot. The SMEV block uses these signals. These outputs are connected to the SMEV block and may optionally be used external to the Analog Interface soft IP blocks by the user.
ASSC_SEQCHANGE	Output	Sequence Change: This output indicates that the outputs ASSC_SEQOUT[TS_WIDTH–1:0] will change after the very next rising edge of CLK. This output is connected to the SMEV block and may optionally be used external to the Analog Interface soft IP blocks by the user.
Users should monitor the following signals if they want to access the ADC conversion results from the ASSC RAM.		
USER_ASSC_RAM_BUSY	Output	ASSC RAM Busy: This output signal indicates that either the Init/Config soft IP block or the SMEV soft IP block is busy accessing the A-port of the ASSC RAM. This signal can optionally be used by user logic external to the Analog Interface soft IP blocks, or can be left unconnected if unused.
USER_ASSC_ADDR[8:0]	Input	User RAM Address: These address signals can be controlled by the user to allow read access from the A-port of the 512×9 ASSC RAM. If unused, these signals should be tied off to logic 0 or logic 1.
USER_ASSC_RD	Input	User RAM Read Enable: This control signal can be controlled by the user to allow read access from the A-port of the 512×9 ASSC dual-port RAM (the user will need to connect to the ASSC_RAM_DO_A[8:0] port for read data). If unused, this signal should be tied off to logic 0. **The user must ensure that the ASSC_RAM_BUSY signal is inactive at logic 0 while this signal is activated; otherwise, the data read from the A-port of the ASSC RAM will not be from the USER_ASSC_ADDR[8:0] address.**
ASSC_RAM_DI_A[8:0]	Output	ASSC RAM Write Data: These signals are connected to the A-port data inputs (write data) of the 512×9 ASSC RAM.

System Monitor Evaluation Phase State Machine (SMEV)

Function

The SMEV reads ADC samples from the result[11:0] locations in the ASSC RAM after each channel sequence has been processed by the ASSC block, and performs evaluation processing on the ADC samples. The SMEV block performs digital low-pass filtering of these samples, compares the low-pass-filtered samples against compare thresholds, and writes the results back into one or more 512×9 dual-port RAM tiles (the number of 512×9 RAM tiles required will depend upon how many program sequences are required for each application). Although the SMEV block deals with samples from the ADC, there is no direct link between it and the ADC; the ASSC block writes all raw ADC samples into the ASSC 512×9 dual-port RAM, which the SMEV reads during the evaluation phase.

Interfaces

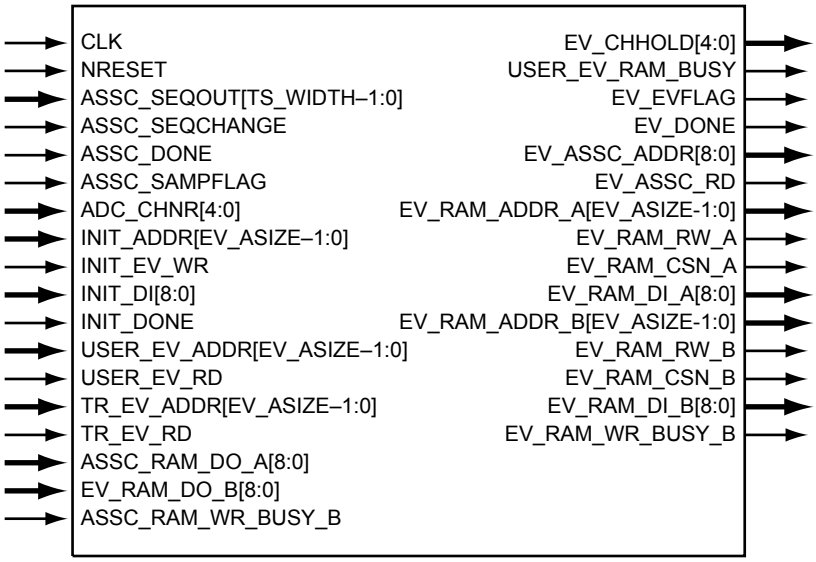


***Figure 11-4* • SMEV I/O Signal Diagram**

Note: All signals are active high (logic 1) unless otherwise noted. All port width specifications are in Verilog notation. Any alphabetic text within port width brackets indicates that the specified port width is controlled via a generic (VHDL) or parameter (Verilog) described in Table 11-4 on page 282.

To access the ADC conversion results from the SMEV RAM, users should monitor the signals in Figure 11-4.

***Table 11-4* • SMEV I/O Signal Descriptions**

Name	Type	Description
USER_EV_ADDR[EV_ASIZE–1:0]	Input	User RAM Address: These address signals can be controlled by the user to allow read access from the 512×9 SMEV RAM(s). If unused, these signals should be tied off to logic 0 or logic 1.
USER_EV_RD	Input	User RAM Read Enable: This control signal can be controlled by the user to allow read access from the A-port of the 512×9 SMEV dual-port RAM(s) (the user will need to connect to the EV_RAM_DO_A[8:0] port for read data). If unused, this signal should be tied off to logic 0. **The user must ensure that the USER_EV_RAM_BUSY signal is inactive at logic 0 while this signal is activated; otherwise, the data read from the A-port of the SMEV RAM(s) will not be from the USER_EV_ADDR[EV_ASIZE–1:0] address.**
USER_EV_RAM_BUSY	Output	SMEV RAM Busy: This output signal indicates that either the Init/Config Soft IP block or the SMTR block is busy accessing the A-port of the SMEV RAM(s). This signal can optionally be used by user logic external to the Analog Interface soft IP blocks or can be left unconnected if unused.
ASSC_RAM_WR_BUSY_B	Input	ASSC Busy Writing: This active-high signal indicates that the ASSC block is busy writing to the B-port of its dual-port RAM (this input is only used for non-Fusion/-ProASIC®3 technology implementation, such as ProASIC PLUS®, which has no true dual-port RAM).
EV_RAM_WR_BUSY_B	Output	SMEV Busy Writing: This active-high signal is for user status monitoring and indicates that the SMEV block is busy writing to the B-port of its dual-port RAM. It must be connected to the SMTR block for non-Fusion/-ProASIC3 technology implementation, such as ProASIC PLUS; otherwise, it can be left unconnected.
EV_RAM_DI_A[8:0]	Output	SMEV RAM Write Data: These signals are connected to the A-port data inputs (write data) of the 512×9 SMEV RAM(s).

System Monitor Transition Phase State Machine (SMTR)

Function

For each enabled channel, the SMTR checks comparison results previously calculated by the SMEV block and stored in the SMEV RAM, and asserts or deasserts different user-defined threshold flags, which are general purpose outputs (GPOs) of the Analog System Block. The GPOs can be used by internal logic in the FPGA array, or they can trigger external I/Os directly.

Interfaces

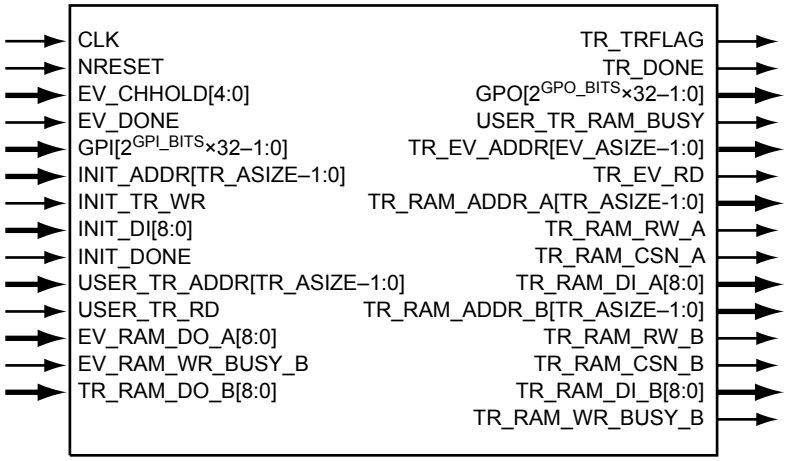


***Figure 11-5* • SMTR I/O Signal Diagram**

Note: All signals are active high (logic 1) unless otherwise noted.

Basic Analog Block Settings

In the Analog System Builder main window, the user can enter system clock frequency and ADC resolution. The system clock is used to drive the ASSC, SMEV, and SMTR soft IP blocks. Also, ADC_CLK is derived from the system clock, and has to be equal to or less than 10 MHz. The ADC block has a built-in divider (4×, minimum divider = 4) to automatically divide the system clock into the appropriate ADC_CLK range. Users can achieve maximum rate for ADC_CLK (10 MHz) by selecting a system clock frequency of 40 MHz or 80 MHz. For more information about the ADC sample rate and accuracy, refer to the *Analog-to-Digital Converter Background* section in the "Designing the Fusion Analog System" section on page 233.

Refer to the *Fusion Starter Kit User's Guide and Tutorial* for a sample design implementing the following settings.

AV Parameter Settings

To configure a voltage monitor, the user can choose to use either direct analog input or prescaled input.

Direct analog input limits the input voltage to less than the VAREF voltage. Usually, it is 2.56 V if the internal reference voltage is chosen, or it can be 0 to 3.3 V if an external reference voltage is used.

For example, if internal VAREF (2.56 V) is selected, choosing a direct analog input to sample a signal that swings between 0 to 2.56 V can avoid the gain and offset error that could be introduced by the prescaler.

However, if the input voltage is too small (0 V – 0.2 V) compared to 2.56 V, and if direct input is used, resolution will be degraded. At this time, a prescaler should be used to amplify the signal for better resolution.

If the input signal is greater than VAREF, the prescaler must be used to scale down the input range before the ADC can sample and convert it.

Once the ADC finishes converting the analog signal to a digital value, it filters (averages) the resulting digital output. Digital filtering is a single-pole low-pass filter built in soft gates, that can be used to improve the signal-to-noise ratio. If the ADC input data is very erratic, the filtering will smooth out the input and reduce the noise.

The filtered value is calculated using EQ 1:

$$Filtering_result_n = filtering_result_{n-1} + (ADC_Result_n / filtering_factor) - (filtering_result_{n-1} / filtering_factor)$$

EQ 1

If the digital filtering factor is set to 1, it is ignored.

In some cases where the inputs have very low frequency and the electrical environment is not very noisy, it may be possible to proceed without any special filtering of input analog signals. However, in most applications it is desirable to at least implement a simple post-conversion digital filter inside the FPGA by oversampling and averaging several results to reduce the effects of random noise in the conversion signal path and improve overall accuracy. This simple averaging is automatically handled in the software by setting the digital filtering factor in the Analog System Builder to specify how many samples are averaged (when the factor = *N*, 2^N samples are averaged together).

For situations where greater accuracy is required, an external analog filter may be needed to eliminate non-random and out-of-band noise sources. If an analog filter is not used to restrict the input signal content to the band of interest, any out-of-band signals or noise will be aliased into the conversion result as random in-band noise.

Some applications—for example, those that require frequency detection—may need both external analog filtering to limit out-of-band effects, and more sophisticated digital processing such as a multi-tap Finite Impulse Response (FIR) filter. A wide variety of digital filtering methods are available through the FPGA gates available in a Fusion device.

Once a digital filter factor is selected, the Initial Value option is activated. This initial value is used for simulation purposes. The user can preset an initial value to imitate a real situation. For example, let the input signal be a 3.3 V power supply that fluctuates around 3.3 V with a range of 50 mV. The user can set 3.3 V as the initial value for simulation mode.

Acquisition time defines how much time the user gives the ADC to conduct the sampling and conversion. If the acquisition time is too short, the input signal may not even be settled yet, and the ADC will just sample some invalid signals. The recommendation is at least 0.2 µs for direct analog input, and 10 µs for the prescaled input. Refer to the *Analog-to-Digital Converter Background* section in the "Designing the Fusion Analog System" section on page 233 for more info on acquisition time.

Maximum voltage defines the expected maximum input voltage on this particular channel.

Users can set threshold flags for SMEV IP to compare against the input voltage, and SMTR IP will trigger the corresponding flags based on the SMEV comparison results.

The Assert Samples and Deassert Samples parameters define after how many consecutive events a flag should be asserted or deasserted.

AC Parameter Settings

Besides all the parameters discussed in the voltage monitor configuration, a current monitor acquisition time should be at least 5 µs. Users should also define the signal polarity and make sure that the potential on the adjacent AV pad MUST be greater than the AC pad.

A realistic sense resistor value (0.005 – 100 Ω) should be entered in the AC peripheral configuration window. The adjacent AV channel in the same Analog Quad can still be used as a voltage monitor.

AT Parameter Settings

Similar to the current monitor, a 5 µs acquisition time and digital filter factor value of greater than 512 are recommended for better conversion accuracy.

Sample Sequence Setting

Users can choose to sample some or all of the analog channels. To manually adjust the sample order, select **Allow manual modification of operating sequence** in the sample sequence configuration window.

The last operation should always jump back to the main procedure or jump to another procedure.

Package Pin Assignment

Users can assign the package pin number for the AV/AC/AT or gate driver peripherals in the Analog System Builder window, and this assignment will be honored in the Designer software.

Soft IP Implementation Options

Default Implementation (ASSC, SMEV, and SMTR)

The default implementation for the Analog System soft IP includes the ASSC, SMEV, and SMTR IP blocks for the Fusion Analog System and the Init/Config IP block for the Fusion flash memory system. This section focuses on the Analog Block soft IP. The Init/Config IP is discussed in the *Using the Embedded Flash Memory for Initialization* section in the "Fusion Embedded Flash Memory Blocks" section on page 137.

Figure 11-6 is a view of the IP and the RAM blocks in the Analog System. The datapath of the system is shown in the figure (MUXes are not shown for clarity).

Dual-port RAMs are used in the soft IP. All IP blocks access their corresponding RAMs through PORTB. All IP blocks access each other's RAM through PORTA. All user access is through PORTA.

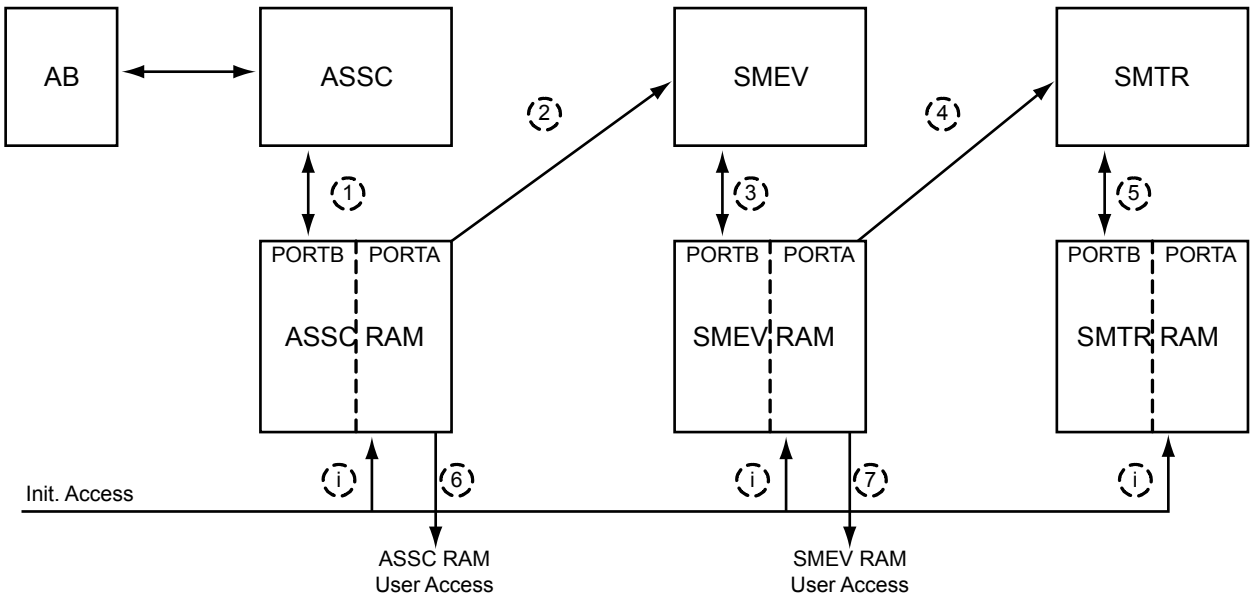


Figure 11-6 • **Analog System – IP and RAM Blocks**

The sequence of events for processing ADC data is as follows:

1. ASSC reads its opcode from the ASSC RAM for slot N processing.
2. ASSC processes ADC for slot N.
3. ASSC completes slot N processing and writes ADC result value to ASSC RAM.
4. ASSC signals DONE.

5. SMEV wakes up and begins reading SMEV RAM for opcodes.
6. ASSC reads its opcode from the ASSC RAM for slot N + 1 processing.
7. SMEV reads ASSC RAM for ADC result value of slot N processing.
8. The SMEV and SMTR state machines do not execute in parallel. The SMEV state machine finishes its processing and then signals the SMTR to begin.

Figure 11-7 is the simulation result illustrating the soft IP events:

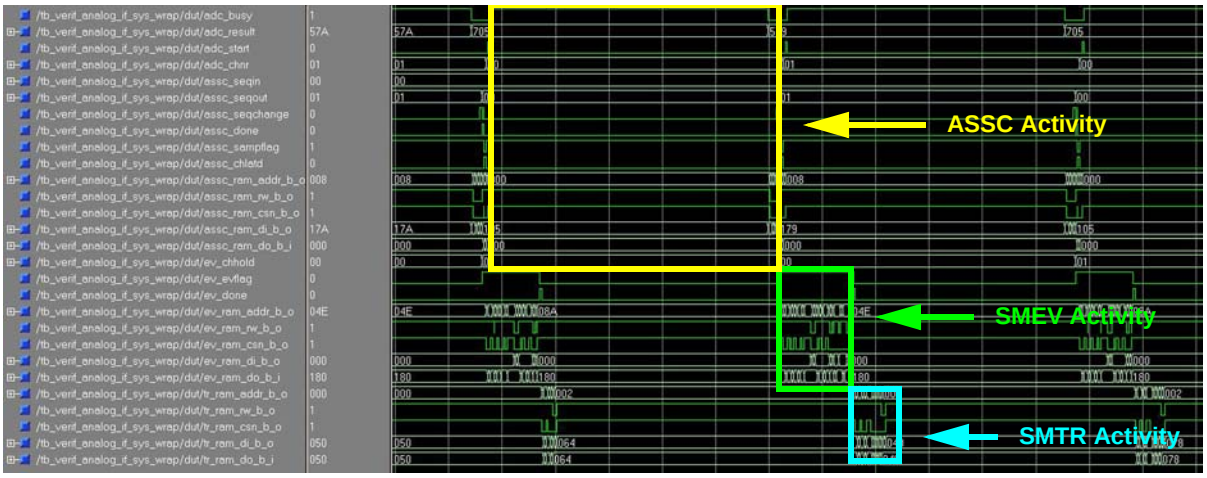


***Figure 11-7* • Simulation Result Illustrating Soft IP Events**

Figure 11-8 shows the soft IP process for multiple channels in a pipeline mode:

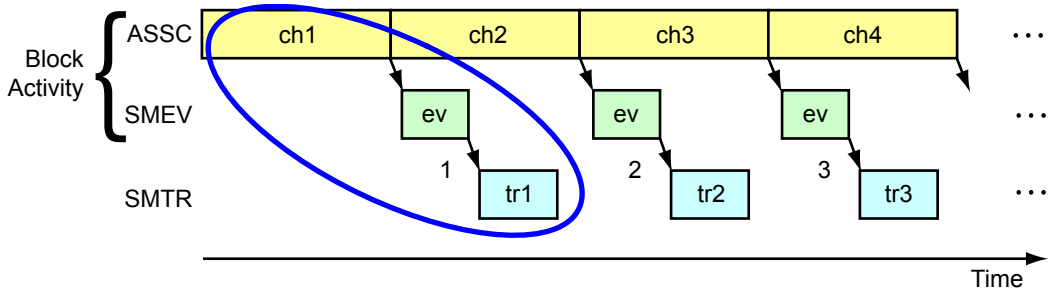


***Figure 11-8* • Soft IP Process for Multiple Channels in Pipeline Mode**

Use Default Implementation and Expose ADC Result (ASSC I/Os, SMEV I/Os, and ACM I/Os)

When users want to directly access the raw ADC results from the ADC or from the ASSC RAM, or access the averaged ADC result from the SMEV RAM, they can select the corresponding ports in the Advanced Options window. They will need to develop corresponding HDL code to access those interfaces. Users can also export and access the ACM bus, as described in the *Analog Configuration MUX (ACM)* section in the "Interfacing with the Fusion Analog System: Processor/Microcontroller Interface" section on page 253. For sample HDL code, refer to the *Fusion Starter Kit User's Guide and Tutorial*.

User Accessing ADC_RESULT Directly

To read ADC_RESULT from the ADC directly, the following signals should be monitored closely:

ASSC_DONE (active-high)

DATAVALID (active-high)

ADC_CHNUMBER

ADC_RESULT

Figure 11-9 shows the timing relationship among the four signals listed above.

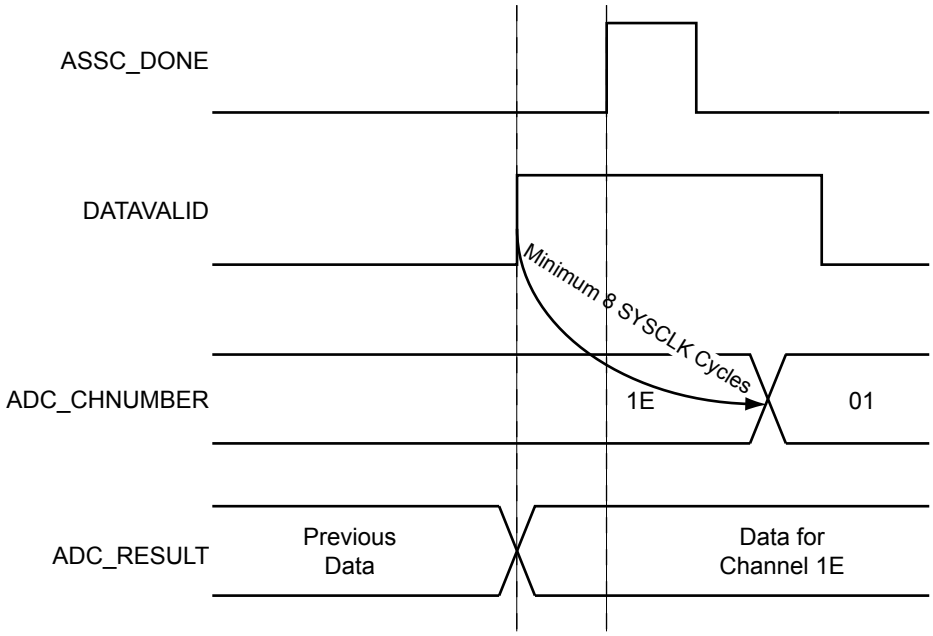


***Figure 11-9* • How to Read Valid ADC_RESULT**

ASSC_DONE assertion determines which channel data is available on the ADC_RESULT bus.

DATAVALID assertion indicates that the new ADC_RESULT is ready.

When both ASSC_DONE and DATAVALID are asserted, the user should record which channel number is active and then read ADC_RESULT for that channel.

The time elapsed from the rising edge of DATAVALID to the next channel number change is at least eight SYSCLK cycles. If there is concern that ADC_RESULT may not be read out and processed as fast as the channel changes, ADC_RESULT and ADC_CHNUMBER should be latched—this is one of the functions of the ASSC RAM.

ASSC and User Accessing ASSC RAM

A signal is exported to indicate that the ASSC is reading or writing the SRAM (USER_ASSC_RAM_BUSY). The user should not access this interface while that is occurring.

SMEV and User Accessing ASSC RAM

The USER_ASSC_RAM_BUSY signal indicates that the SMEV is accessing the RAM, and as stated in the documentation, the user should not access this interface while that is occurring.

SMEV and User Accessing SMEV RAM

A signal is exported to indicate that the SMEV is reading or writing the SRAM (USER_EV_RAM_BUSY). This indicates to the user to stop reading.

SMTR and User Accessing SMEV RAM

The USER_EV_RAM_BUSY signal indicates that the SMTR is accessing the RAM, and as stated in the documentation, the user should not access this interface while that is occurring.

In general, the USER_ASSC_RAM_BUSY or USER_EV_RAM_BUSY signal indicates that other IP is accessing the corresponding RAM, and the user should not access the interface while that is occurring. In other words, users can only access the RAM content while the BUSY signal is deasserted.

For a given channel, the ADC result is saved as two parts in two consecutive address locations inside the RAM. For example:

```
*******************************************************************************
            ASSC Memory Content Report
*******************************************************************************
Slot   Channel   Address  Bits       Value
-------------------------------------------------------------------------------
0      AV0
                  3|       [08:00]|   Raw ADC Result [08:00]
                  4|       [02:00]|   Raw ADC Result [11:09]
-------------------------------------------------------------------------------


*******************************************************************************
            SMEV Memory Content Report
*******************************************************************************
       Channel   Address  Bits       Value
-------------------------------------------------------------------------------
       AV0
                  75|      [08:00]|   Averaged ADC Result [08:00]
                  76|      [02:00]|   Averaged ADC Result [11:09]
-------------------------------------------------------------------------------
```

When the BUSY signal is deasserted, the user must execute two User Reads (RD1 and RD2) to read out the whole ADC result. If the BUSY signal is asserted after RD1 but before RD2, the user must re-execute RD1 to read the lower address data once the BUSY signal is deasserted again, followed by RD2 to read the higher address data. Meanwhile, the user must control the address increment appropriately. For sample HDL coding on this topic, refer to the "Sample Code" section on page 291. This code is also used in the Fusion Starter Kit tutorial design example.

The user interfaces to the ASSC RAM or the SMEV RAM are the same. The glue logic to the interface should keep monitoring the BUSY signal and sending the right RAM address at the appropriate time, then execute the READ action. For more details, refer to the timing diagrams in Figure 11-10 on page 289 and Figure 11-11 on page 289.

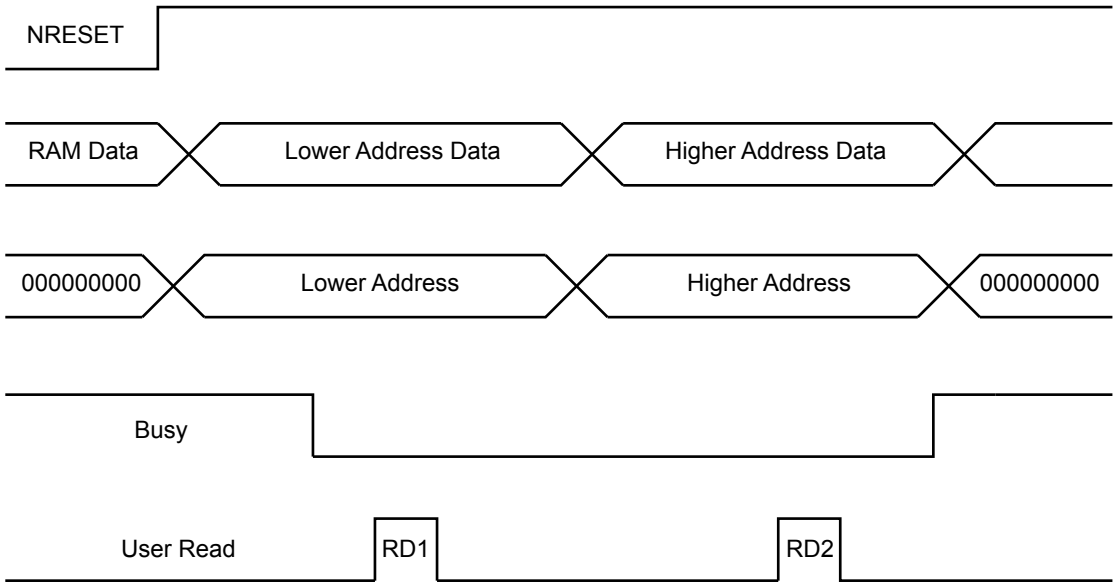


***Figure 11-10* • User Read Soft IP Block RAMs**

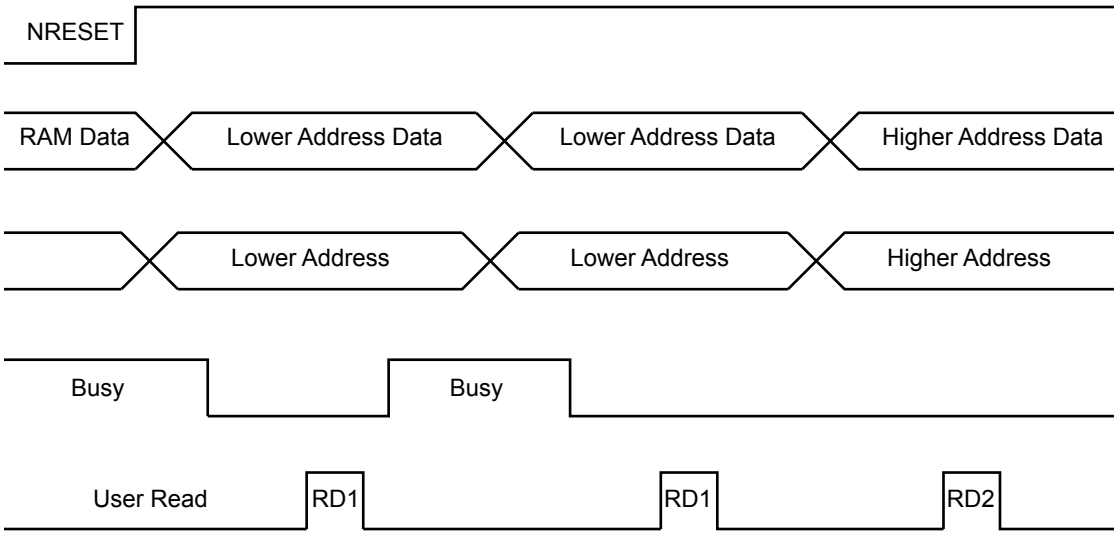


***Figure 11-11* • User Read Soft IP Block RAMs (continued)**

Use IP Cores for ADC Sequence Control Only

When users do not need the SMEV (averaging ADC results) and SMTR (triggering threshold flags) functions, but only need the ADC sequence control function (ASSC IP block), they can select the **IP cores for ADC sequence control only** option in the Advanced Options window. Additionally, if they want to directly access the raw ADC result from the ADC or from the ASSC RAM, or access the ACM bus, they can select the corresponding ports in the Advanced Options window.

VAREF Capacitor Value Selection

The Fusion device can be configured to generate a 2.56 V internal reference voltage (VAREF) that can be used by the ADC. When VAREF is internally generated by the Fusion device, a by-pass capacitor must be connected from this pin to ADCGNDREF. For more information, refer to the “Pin Description” section of the *Fusion Family of Mixed Signal Flash FPGAs* datasheet.

In the SmartGen Analog System Builder, under Advanced Options, users can select the capacitor value based on the system level requirements. Depending on the capacitor value, a delay circuitry will be automatically added to ensure the SmartGen IP will not perform an ADC conversion until the voltage level on the VAREF is stable. Table 11-5 shows the corresponding settling time based on the selectable capacitor value in the software. If the capacitor value is different than the selectable values in the software, the user should pick the next higher capacitor value to ensure sufficient settling time is added. The additional delay will significantly increase the simulation time, but the user may consider changing the resolution of the simulator to speed up the simulation process.

***Table 11-5* • Settling Time for Using the Internal VAREF Reference Voltage**

Capacitor Value (µF)	Settling Time (ms)
3.3	116.45
10.0	324.73
22.0	750.24

Note: Users who are using the standalone AB macro and build their own control interface need to be aware that there is no hold time check in SmartTime for the following signals: VAREFSEL, TVC, STC, MODE, CHNUMBER. Users need to ensure these signals remain stable for at least one clock cycle after the assertion of ADCSTART (see Figure 11-12) and the simulation model provides a check to ensure this requirement is met.

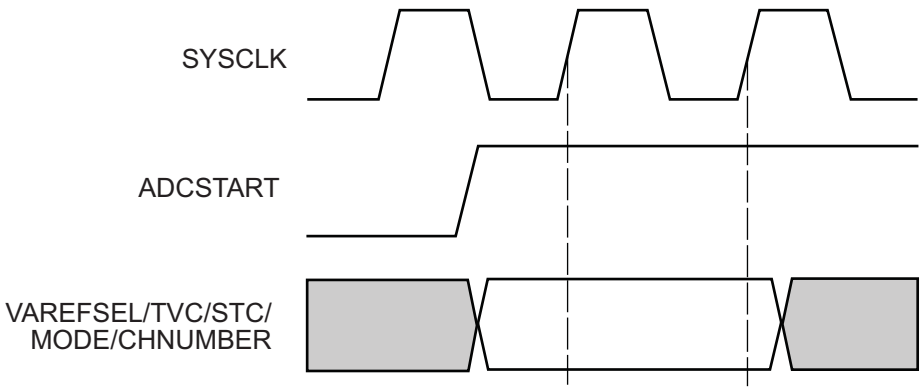


***Figure 11-12* • ADCSTART Timing Diagram**

Analog Configuration MUX (ACM)

The ACM is an interface between FPGA/JTAG test registers and Analog Quad configuration latches, and the Real-Time Counter (RTC). The Analog Block consists of four 8-bit latches per Analog Quad, which get initialized through the ACM. These latches act as configuration bits for Analog Quads. The Analog System soft IP generated from Libero SoC configures the ACM latches automatically. If the user does not use the soft IP, the ACM can be configured manually. For more information, refer to the *Analog Configuration MUX (ACM) Initialization* in the "Interfacing with the Fusion Analog System: Processor/Microcontroller Interface" section on page 253. The ACM block runs off the core voltage supply (1.5 V).

Sample Code

The following sample VHDL code is used to read out the ADC conversion results from either the ASSC RAM or the SMEV RAM.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ram_reader is
  port (
    CLK         :  in std_logic; -- clk
    NRESET       :  in std_logic; -- active low reset

    ASSCDONE      :  in std_logic;
    EVDATA       :  in std_logic_vector( 8 downto 0 );
    EVBUSY       :  in std_logic;

    EVADDR       :  out std_logic_vector( 8 downto 0 );
    EVRD        :  out std_logic;

    AVGDATA       :  out std_logic_vector( 11 downto 0 )
  );
end ram_reader;

architecture ctrl of ram_reader is

-- constants for ACM addresses
constant EV_AVG_DATA1 : std_logic_vector( 8 downto 0 ) := "001001011"; --75d
constant EV_AVG_DATA2 : std_logic_vector( 8 downto 0 ) := "001001100"; --76d

-- state machine
type fsm_type is ( IDLE,
          RDWAIT,
          RD1,
          RD2
         );

signal state        : fsm_type;

-- internal registers to hold match values
signal avgdata_reg      : std_logic_vector( 11 downto 0 );




-- internal signals
signal evrd_i      : std_logic;
signal evaddr_i     : std_logic_vector( 8 downto 0 );
signal ev_d1en, ev_d2en : std_logic;

begin

-- toplevel port maps
EVADDR  <= evaddr_i;
EVRD   <= evrd_i;
AVGDATA <= avgdata_reg;

--------------------------------------------------------------------------------
-- EV AVG DATA REGISTER
--------------------------------------------------------------------------------
process(CLK, NRESET)
begin
  if NRESET = '0' then
    avgdata_reg <= (others=>'0') ;
```

```
  elsif rising_edge(CLK) then
    if ( ev_d1en = '1' ) then
      avgdata_reg( 8 downto 0 ) <= EVDATA( 8 downto 0 );
    else
      avgdata_reg( 8 downto 0 ) <= avgdata_reg( 8 downto 0);
    end if;

    if ( ev_d2en = '1' ) then
      avgdata_reg( 11 downto 9 ) <= EVDATA( 2 downto 0 );
    else
      avgdata_reg( 11 downto 9 ) <= avgdata_reg( 11 downto 9 );
    end if;

  end if;
end process;

-------------------------------------------------------------------------------
-- MAIN STATE MACHINE
-------------------------------------------------------------------------------
process(CLK, NRESET)
begin
  if NRESET = '0' then
    evaddr_i <= "000000000";
    evrd_i <= '0';
    ev_d1en <= '0';
    ev_d2en <= '0';
    state <= IDLE ;
  elsif rising_edge(CLK) then
    case state is
      when IDLE =>
        ev_d2en <= '0';
        if ( ASSCDONE = '1' ) then
          state <= RDWAIT;
        else
          state <= IDLE;
        end if;
      when RDWAIT =>
        if ( EVBUSY = '0' ) then
          evaddr_i <= EV_AVG_DATA1;
          evrd_i <= '1';
          state <= RD1;
         end if;
      when RD1 =>
        if ( EVBUSY = '1' ) then
          state <= RDWAIT;
        else
          evaddr_i <= EV_AVG_DATA2;
          ev_d1en <= '1';
          evrd_i <= '1';
          state <= RD2;
        end if;
      when RD2 =>
        if ( EVBUSY = '1' ) then
          state <= RD1;
        else
          evaddr_i <= "000000000";
          evrd_i <= '0';
          ev_d1en <= '0';
          ev_d2en <= '1';
          state <= IDLE;
        end if;
      when others =>
        state <= IDLE;
    end case;
  end if;
```

```
end process;


end ctrl;
```

The following VHDL code is used to translate the ADC result back to a voltage, current, or temperature value.

```
scale: process (reset_n, clock)
begin
if reset_n = '0' then
  scaled_input <= (others => '0');
elsif clock'event and clock = '1' then
  case format_select is
    when "VOLTAGE" =>
      -- 8V full scale, display volts
      -- need to multiply ADC counts x2
      scaled_input <= "000" & counts_in & '0';
    when "TEMPERATURE" =>
      -- drop two LSBs to read in deg K
      scaled_input <= "000000" & counts_in_int_int(11 downto 2);
    when others =>
      null;
  end case;
end if;

end process;
```

Note: For the current conversion, the ADC result is the differential voltage value across the current sense resistor. To get the final current value, divide the differential voltage value by the sense resistor value.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
August 2012	In the "VAREF Capacitor Value Selection" section, the sentence, "When VAREF is internally generated by the Fusion device, a by-pass capacitor must be connected from this pin to ground," was updated to replace the word "ground" with "ADCGNDREF" (SAR 36725).	290
July 2010	This chapter is no longer published separately with its own part number and version but is now part of the Fusion FPGA Fabric User's Guide.	N/A
v1.2 (December 2008)	EQ 1 was revised to correctly subscript portions of variable names.	284
v1.1 (October 2008)	Modified ADC_RESULT[11:0] description.	279
51700092-007-1	Added "VAREF Capacitor Value Selection" section.	290

12 – Temperature, Voltage, and Current Calibration in Fusion FPGAs

Introduction

Fusion mixed signal FPGAs integrate configurable analog features, including I/Os, prescalers, low-pass filters, and an analog-to-digital converter (ADC), enabling customers to perform temperature, voltage, and current measurements in their applications. Analog components have a specific accuracy for a given set of conditions. The accuracy can have a broad range of definitions and is affected by many parameters in the system. For example, in a temperature measurement application, the accuracy of the measured temperature is influenced by the accuracy of on-chip elements (temperature sensor, op amps, and ADC), use model (sample rate, ADC resolution setting, post-processing, etc.), and board-level considerations. For the purpose of this document, accuracy can be defined as the difference/error between the actual value and the measured value. For example, in a temperature measurement application, an accuracy of ±2°C means that the measured value may be up to ±2°C different from the actual value.

If the difference between the measured value and the actual value is too great, you can use calibration to bring the measured value closer to the actual value. Calibration assumes a profile for the relationship between the actual value and the measured value. This profile depends on the characteristics of the components used in the measurement. There are two calibration profiles: one corrects for offset error only, and the second accounts for both offset and gain errors. Figure 12-1 illustrates these typical profiles that define the calibration implementation methodology.

To completely calibrate a system, users can calibrate individual components, or they can calibrate the entire system, taking into account the error of all the individual components working together. Many users may decide to perform both levels of calibration. This document provides a description of the factory calibration methodology for voltage inputs on Fusion devices, and also provides recommendations on system calibration methods for voltage, temperature, and current using Fusion. Using Microsemi's device calibration solution, the Fusion ADC sampling accuracy for voltage prescalar inputs can be improved to 1%, enabling Fusion to better meet customers' design requirements.

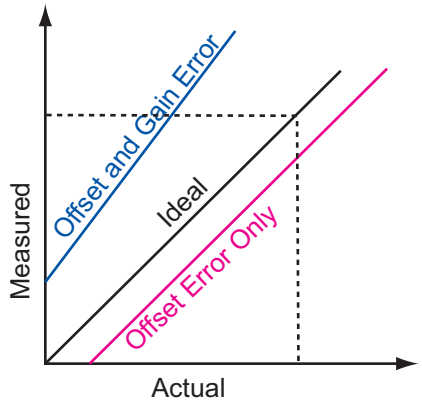


***Figure 12-1* • Example Profiles of Measured Value and Actual Value Variations**

General Calibration Concept

Calibration Methods

Based on the measured-versus-actual profile and the required accuracy, customers can define the most efficient method for translating the measured value into an actual value. In most analog components, including the Fusion FPGA, the relationship between measured and actual values follows the profiles illustrated in Figure 12-1 on page 295. This document describes only calibration methodologies associated with offset-only and offset-and-gain corrections.

Offset-Only Calibration

Offset-only calibration (sometimes known as one-point calibration) is based on the relationship between the measured and actual values given in EQ 12-1:

$$y = x + c$$

EQ 12-1

where

y = actual value
x = measured value
c = offset compensation between measured and actual values

As shown by EQ 12-1, offset-only calibration accounts for the offset between the actual and measured values.

Offset-and-Gain Calibration

If the correlation between the actual and measured values is defined primarily by an offset, as shown in the offset error line in Figure 12-1 on page 295, or if the actual measured value is naturally constrained to a specific region, offset-only calibration may be sufficient to achieve a high degree of accuracy.

However, in some cases, especially if the range of the measurement varies widely, the difference between the actual and measured values not only includes an offset but is also governed by a gain variation, as shown in Figure 12-1 on page 295. In such cases, offset-only calibration may not provide sufficient correction to achieve the accuracy required by the customer's application. In this case, offset-and-gain calibration (also known as two-point calibration) can be implemented to achieve a higher level of accuracy.

In two-point calibration, the relationship between the measured and actual values is governed by EQ 12-2:

$$y = mx + c$$

EQ 12-2

where

y = actual value
x = measured value
c = offset compensation between measured and actual values
m = gain compensation between measured and actual values

Choosing between one-point calibration and two-point calibration depends on many parameters, some of which include the following:

- Customer application's required accuracy
- Measurement gain and offset error of electrical components, such as the Fusion FPGA
- Application's operating range

Customer Application's Required Accuracy

Given the required accuracy of an application within its operating range, designers can use the specified gain and offset error of Fusion FPGAs to determine the suitable calibration method to achieve the desired accuracy. Refer to the *Fusion Family of Mixed Signal Flash FPGAs* datasheet for more information.

Calibration Measurements

To calculate m or c in EQ 12-1 and EQ 12-2 on page 296, measurements must be taken in a known environment so measured data can be compared against actual values. The number of required data points depends on the method of calibration. One data point would suffice for one-point calibration (determining offset), whereas for two-point calibration, two data points are needed to define gain and offset. In calibration measurements, a known actual value (temperature, current, or voltage) is supplied to the system, and the measured value is recorded.

Offset-Only Calibration Measurement

In offset-only (or one-point) calibration measurement, an actual value of P_{a1} (e.g., temperature) is applied to the system, and its value is measured as P_{m1} by the system. Then, the offset value c in Figure 12-1 on page 296 can be calculated as shown in EQ 12-3.

$$c = y1 – x1$$

EQ 12-3

Offset-and-Gain (two-point) Calibration Measurement

As shown in Figure 12-2, to calculate m and c in EQ 12-2 on page 296, two data points are needed for two-point calibration.

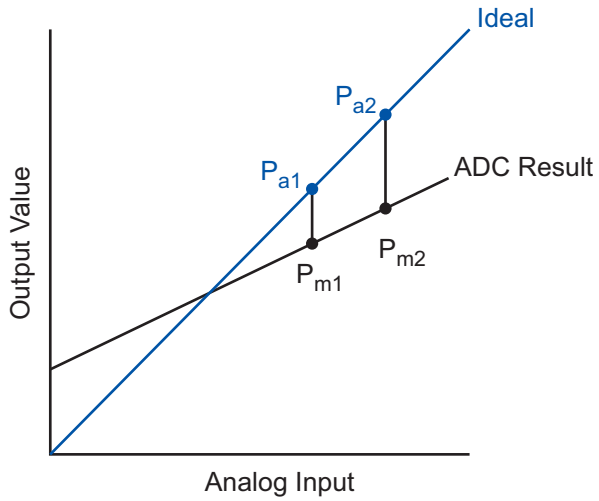


***Figure 12-2* • Two-Point Calibration for Offset-and-Gain Error**

Therefore, two known actual values (y_1 and y_2) must be applied to the system, and two measured points (x_1 and x_2) must be recorded. The m and c values in EQ 12-2 on page 296 are calculated as shown in EQ 12-4 and EQ 12-5.

$$m = (y_2 – y_1) / (x_2 – x_1)$$

EQ 12-4

$$c = (y_1 \times x_2 – y_2 \times x_1) / (x_2 – x_1)$$

EQ 12-5

Choosing Calibration Data Points

In one-point calibration, from a practical point of view, Microsemi recommends that the applied actual value be in the middle of the operating range as defined by the application. For example, if the system measures a voltage that operates from 0 V to 5 V, taking the calibration measurement at ~2.5 V will typically give the best results.

For two-point calibration measurement, Microsemi recommends that the two data points be taken at 20% and 80% of the operation range. For example, if a temperature measurement application is used in a system that operates from 0°C to 50°C, Microsemi recommends that the calibration measurements be taken at 10°C and 40°C.

In many voltage or current monitoring applications where the operating range includes 0 V or 0 A, customers tend to choose 0 V or 0 A as one of their calibration measurement data points. This is mainly driven by the simplicity of setup for measurements at the ground level. However, the ground level of the system is typically noisy due to the operation of the system and other noise factors. In such situations, the calibration measurement may not be sufficient to achieve the accuracy level required by the overall design. Therefore, Microsemi recommends that zero-level measurements be avoided for voltage and current calibration data collection.

Fusion FPGAs offer up to 32 analog channels for temperature, current, or voltage measurements. Many applications, such as system management, use more than one analog channel in their design. Though all these channels use a single ADC inside Fusion, each prescaler circuit within the analog I/O structure has a unique set of op amp circuits. Therefore, it is necessary to calibrate each channel that requires the increased level of accuracy independently. In this case, each channel has its own calibration coefficient based on the method used for calibration (one-point or two-point).

Furthermore, in an analog (voltage/current/temperature) measurement, application designers exploit other components besides the Fusion FPGA to sense, transport, or amplify the measured parameters.

Customers can use two general approaches in calibrating these systems:

1. Calibrate each device used in the measurement individually.
2. Calibrate all the utilized devices when operating together in the system.

In the first approach, the customer calibrates each device individually in a controlled setting. In this case, the methods and recommendations explained in this document are applicable for the Fusion device. For other components, the customer should follow the recommendations and techniques provided by the vendor of each component.

In the second approach, all the system components in the application are used to take the calibration data points. In this case, the total measurement error can be adjusted by calibrating the measured values after the ADC. If this method is used, all the recommendations and techniques in this document can be applied.

Calibration Solution

Microsemi's device-level calibration solution offers significant improvement in ADC accuracy for voltage monitor applications. There are two ways of exercising the Fusion ADC for voltage monitoring: sampling prescaled analog input or sampling direct analog input. If a customer design requires better accuracy than the default Fusion ADC performance, then calibration is needed. Since direct analog input sampling accuracy is well within 1%, the Microsemi calibration solution does not offer any additional benefit, so it is only available for prescaled inputs.

Temperature and current monitor calibration are not supported. Although calibration is not done at production, the offset value for the temperature monitor is stored in eNVM along with other calibrated values. If CalibIP is used, the offset error is removed from the temperature monitor reading.

The calibration solution is a two-point offset-and-gain calibration scheme. The implementation is performed through the following two steps:

1. During production test and screening flows, m and c compensation values are determined for each analog voltage channel and stored in the flash memory block of each Fusion device.
2. In Microsemi Libero® software v8.2 SP1 and later, an RTL calibration IP block is built into the SmartGen Fusion Analog System Builder core. This calibration block reads the m and c values stored in the flash memory and uses them to calibrate data for each analog voltage channel.

Coefficient Measurement and Programming

During the Fusion production test flow, in a controlled environment, Microsemi measures the calibration coefficients, m and c, of every prescaler level of all 30 channels of each device. Measurements are done with the Fusion ADC VAREF set to 2.56 V. In other words, the coefficients do not apply to any customer designs that use a VAREF other than 2.56 V. Microsemi calibration implementation is disabled in software when VAREF is set to another value.

Then coefficients are programmed into the dedicated spare page of Fusion flash memory block (FB) 0 (AFS600 has blocks 0 and 1), from page 50 to 62.

Customers should avoid overwriting these spare pages. An old design generated prior to Libero software v8.2 SP1 utilizes these spare pages for Analog System configuration data. Programming an old design to a calibrated device could overwrite these spare pages and corrupt the coefficients. Calibration coefficients of that device would then no longer be available.

On the other hand, programming a design with a calibration block generated from Libero software v8.2 SP1 or newer to an uncalibrated device will result in erroneous data from the ADC. To avoid this, customers can pre-program the device with the POPULATION.stp file provided in the Libero software v8.2 SP1 release. This programming action populates the dedicated flash memory area for calibration with m = 1 and c = 0. Then customers can program the device with the design STAPL file.

Calibration IP Deployment

To implement Microsemi's calibration solution, customers must generate a new Analog System and Flash Memory System using Libero software v8.2 SP1 or newer. Microsemi's calibration IP solution is not available for processor systems that use the CoreAI to interface to the analog block.

Analog System Builder Update

Through the Analog System Builder, customers have an option to deploy a calibration IP block named CalibIP into Fusion designs. The CalibIP block is seamlessly inserted into the original Analog System macro, as shown in Figure 12-3 on page 12-300. Figure 12-3 on page 300 shows only the insertion into a

full Analog System; the same concept applies to the Sequence Only (without SMEV and SMTR stages) and ADC Only (without ASSC, SMEV, and SMTR stages) flows.

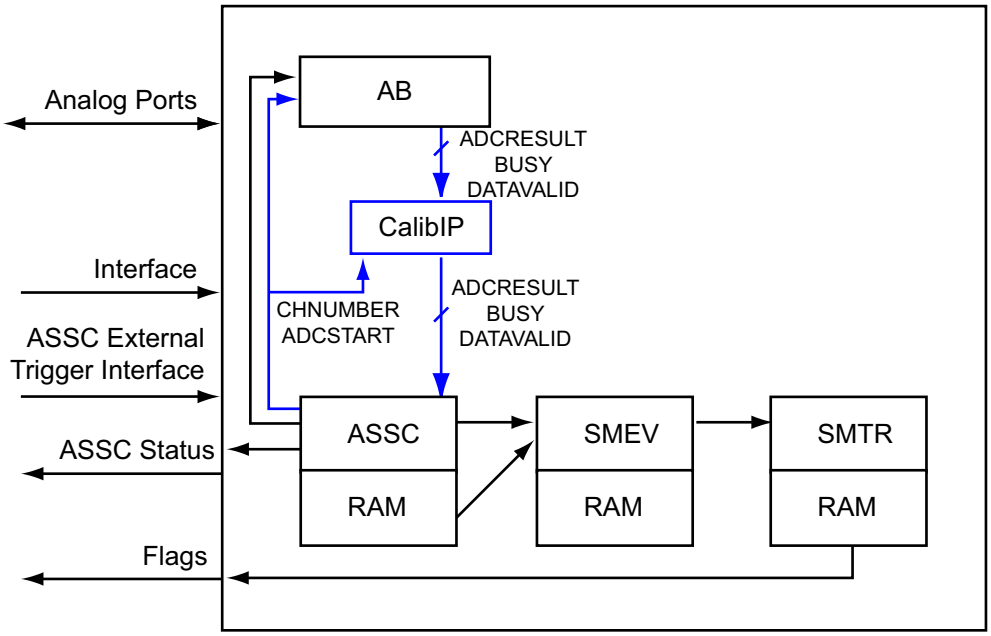


***Figure 12-3* • Full Analog System Macro with CalibIP**

During power-up, the initialization state machine, INIT/CONFIG IP, loads the coefficients from the flash memory block into a dedicated SRAM block for the CalibIP core. CalibIP reads the coefficients from the SRAM block and applies the m and c values to the raw ADCRESULT, following EQ 12-2 on page 296 to generate the calibrated ADCRESULT. The calibrated ADCRESULT then goes through the rest of the process as in the original processing flow.

There are two new ports created at the Analog Block top level to support calibration initialization from the flash memory block:

- INIT_CALIBROM_WEN – Write enable to ROM region, single-bit, active-high
- INIT_CALIBCOEFF_WEN – Write enable to coefficient region, single-bit, active-high

These are write enables for the INIT/CONFIG interface. Connect them to corresponding ports of the flash memory block top level, as shown in Figure 12-4 on page 12-301.

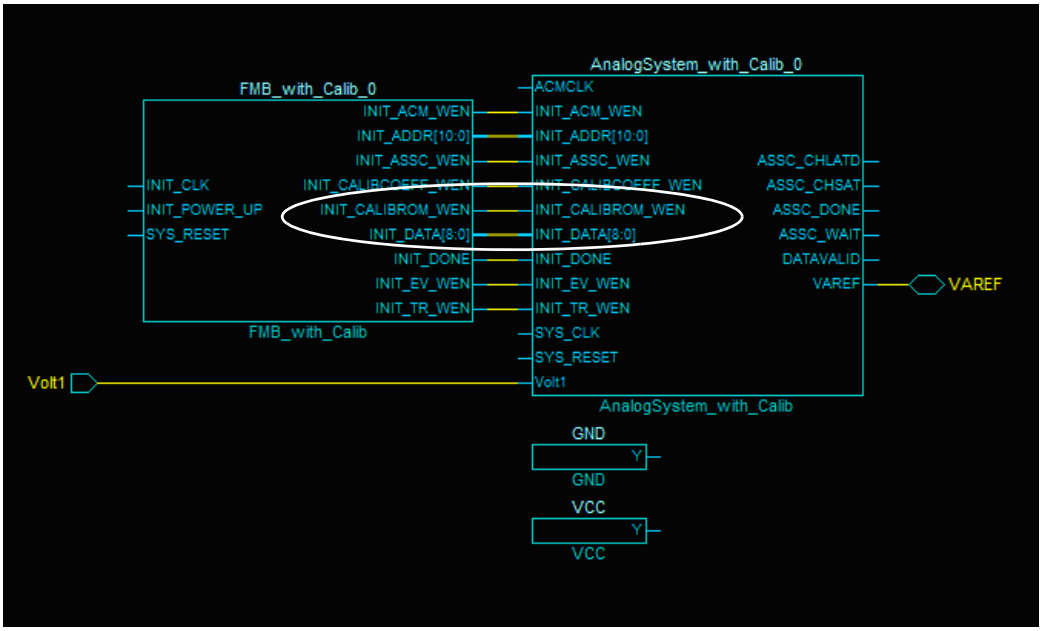


Figure 12-4 • **Connectivity between Analog System Block and Flash Memory Block**

Flash Memory System Builder Update

Inside the Flash Memory System Builder (FMSB), customers can generate an analog client to properly initialize the Analog System macro with CalibIP deployed. In addition to the regular Analog System configuration data partition, FMSB also creates two other partitions for the analog client: one for the coefficients' storage (CALIBCOEFFICIENT), from spare page 50 to 62, and one for a lookup table (CALIBROM) that records which channel and prescaler level need to be calibrated, from spare page 43 to 48.

Because of the new calibration coefficients' storage partition, the SMTR configuration data is assigned to flash memory block sector 63, from page 2,016 through 2,047 (or as addresses: 0x3F000 through 0x3FF80). SMTR uses up to 32 pages. The actual number of pages used in this sector depends on whether (and how many) flags are used in the Analog System design. The Flash Memory System Builder prevents customers from assigning any other clients to these pages.

The flash memory maps prior to and after the Libero software v8.2 SP1 release are shown in Figure 12-5 and Figure 12-6 on page 303.

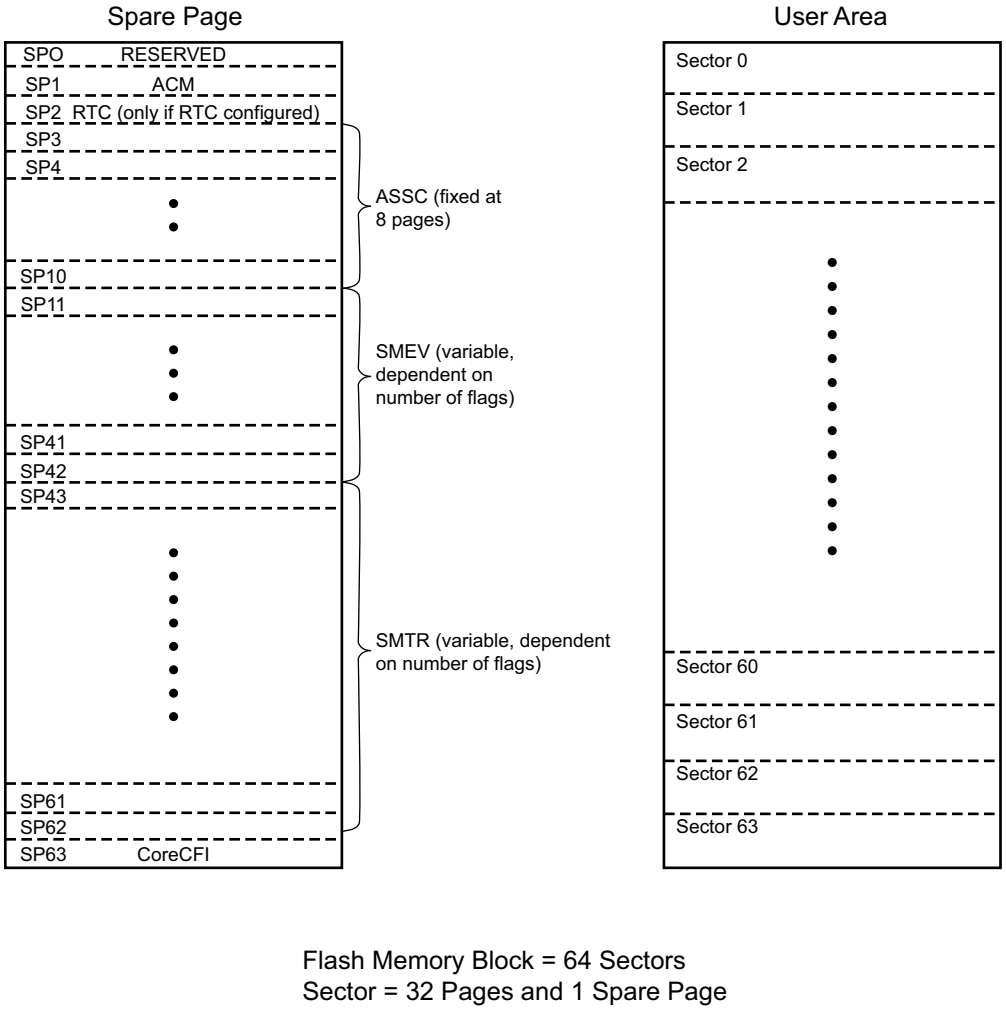


Figure 12-5 • **Flash Memory Map Prior to Libero Software v8.2 SP1**

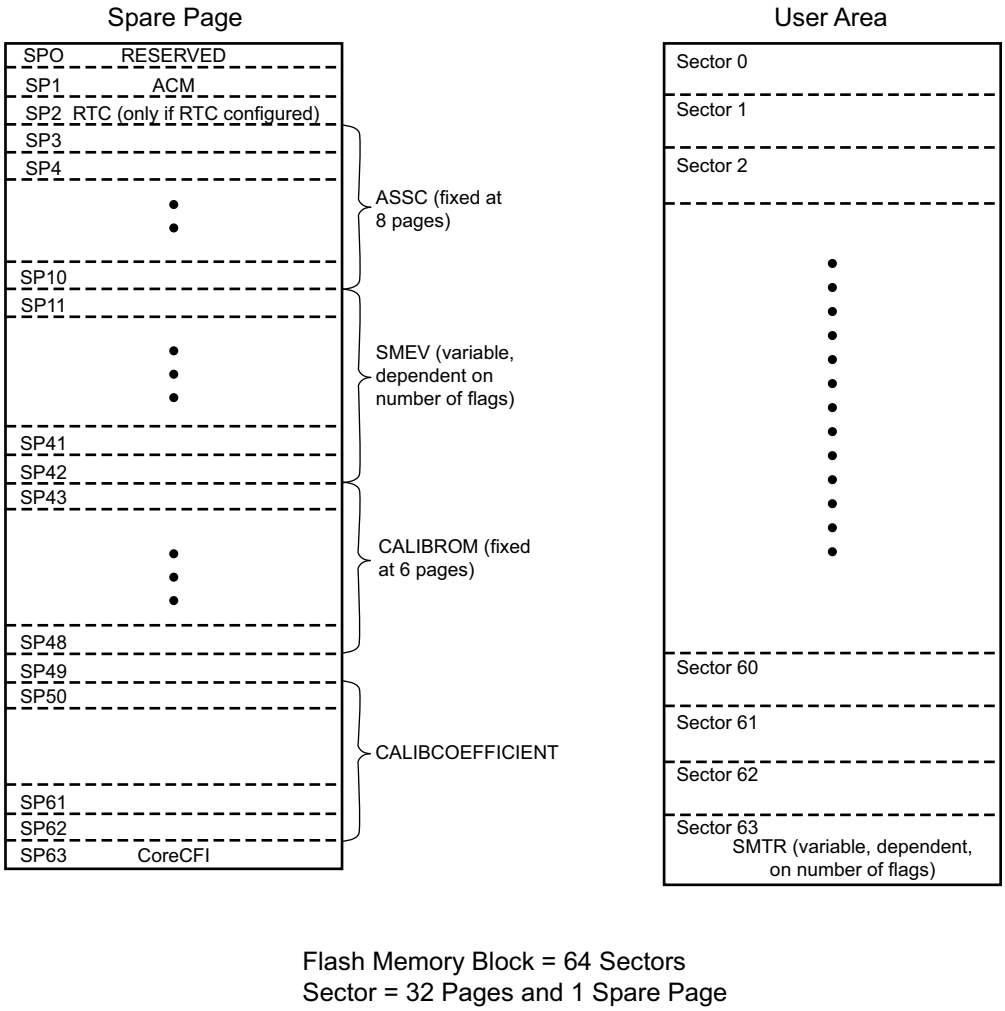


***Figure 12-6* • Flash Memory Map after Libero Software v8.2 SP1**

There are two new ports created at the flash memory block top level, corresponding to those created for the Analog Block:

- INIT_CALIBROM_WEN
- INIT_CALIBCOEFF_WEN

Connect these ports to the Analog Block top level.

Design Flow and Tips

For calibrated Fusion devices, there are several implementation scenarios.

Scenario 1: Brand New Design

Follow the regular design flow to implement the Microsemi calibration solution in a new design:

By default, the calibration IP is enabled in the Analog System macro, and the flash memory block initializes the IP. The Designer software places the corresponding analog client in flash memory block 0.

The content of the memory file (*.mem) is different from that of the embedded flash configuration file (*.efc). The *.mem file produced by the Flash Memory Builder for simulation purposes is populated with m = 1.0 and c = 0 for all channels and all prescaler combinations. CalibIP can function appropriately during simulation. The *.efc file for programming file generation does not include the m and c content because the coefficients are pre-programmed into the device during production test.

To disable the calibration IP, uncheck **Include calibration IP** in the Advanced Options of the Analog System Builder, as shown in Figure 12-7.

For a calibration-enabled design, when **Bypass calibration on saturated ADC input** is selected, the saturated ADC result is passed to the next level of computation without calibration. If unchecked, the saturated ADC result is calibrated before it is passed to the next level.

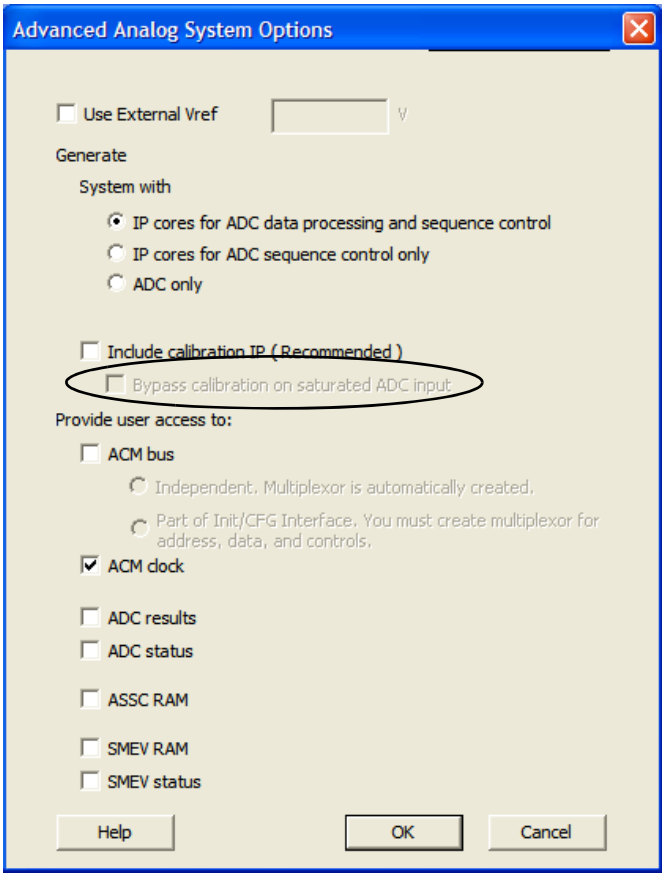


***Figure 12-7* • Disable CalibIP from Advanced Options (Libero Software v8.2 SP1)**

Scenario 2: Update Existing Design to Implement Calibration Solution

To update existing Fusion designs and utilize the Microsemi calibration solution, take the following steps to regenerate the design:

1. Open the design in Libero software v8.2 SP1 or newer.
2. Regenerate the Analog System macro in Analog System Builder.

   Open **Advanced Options, select the Include calibration IP** option, then regenerate the macro.
3. Regenerate the flash memory block.
4. Make sure the additional ports (INIT_CALIBROM_WEN and INIT_CALIBCOEFF_WEN) are properly connected, either through SmartDesign (Figure 12-8) or HDL coding.

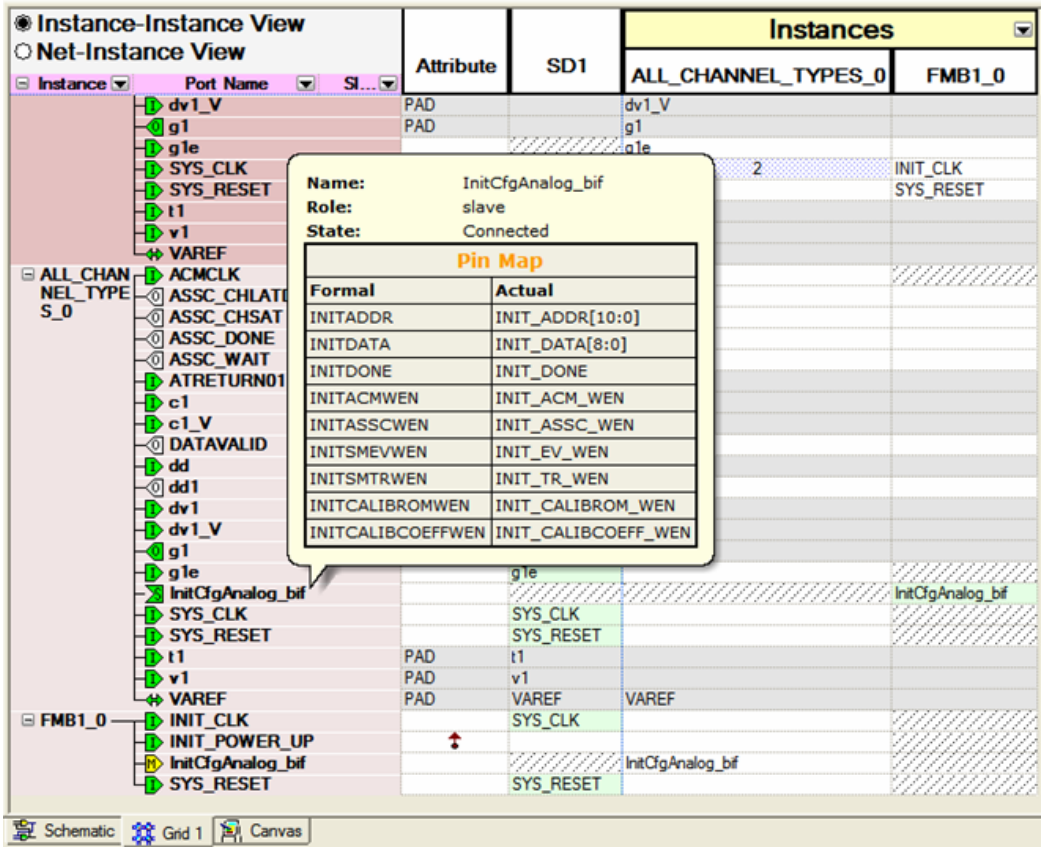


***Figure 12-8* • SmartDesign Connectivity Grid in Libero Software v8.2 SP1**

Go through the rest of the regular design flow (synthesis, compile, and layout with proper simulation and timing analysis).

Scenario 3: Existing Design for Firmware Image Update Only

To update the firmware image in the existing design without using the calibration IP, regenerate the Flash Memory Block. Then go through the rest of design flow (synthesis, compile, and layout with proper simulation and timing analysis).

Scenario 4: Maintain Existing Design in New Software Release without Using Calibration Solution

Microsemi recommends that all customers regenerate the Analog System Block and the flash memory block in Libero software v8.2 SP1 or newer software releases, unless the Analog System Block utilizes less than 20 channels and less than four flags per channel.

Programming a calibrated device with designs generated prior to Libero software v8.2 SP1 can overwrite and corrupt the pre-programmed coefficient data in the dedicated flash memory partition. The FlashPro software released with Libero software v8.2 SP1 detects whether there is a memory map overlap. If there is a memory overlap, FlashPro cancels the programming action and asks the user to regenerate the Analog System.

To program a design with calibration implemented to a targeted device that is uncalibrated, pre-program the device with the POPULATION.stp file provided by Microsemi. This programming action populates the dedicated flash memory area for calibration with m = 1 and c = 0. Then program the device with the design STAPL file.

Utilization and Performance

The total RAM block and core tile utilization to implement CalibIP and the corresponding initialization process is listed in Table 12-1. CalibIP and other IPs infer registers with enable. When these registers have a SET or RESET signal, assign the SET or RESET signal to a global resource to make sure that the register remains a one-tile implementation (NOT split into two tiles).

***Table 12-1* • Calibration Implementation Utilization Report**

RAM Block	Tile Count for CalibIP	
	Optimized for Area	Optimized for Speed
1	363	453

The CalibIP performance is listed in Table 12-2.

***Table 12-2* • CalibIP Performance Report**

Speed Grade	CalibIP Performance (MHz)	
	Optimized for Area	Optimized for Speed
Std.	45	57
–2	75	96

The performance of CalibIP is only limited by the latency introduced by the Compute Block. The Compute Block adds a latency of 14 clock cycles. For example, 14 clock cycles of a 40 MHz system clock is 0.35 microseconds. With calibration implemented, you can expect the ADC result 0.35 microseconds later than in a design without calibration implementation. The sampling rate is degraded by 2%.

Improvement from Microsemi Calibration Solution

Table 12-3 shows typical error using Microsemi's calibration solution.

***Table 12-3* • Fusion Analog System Typical Error with CalibIP Deployed**

Input Voltage (V)	Calibrated Typical Error per Positive Prescaler Setting[1] (%)							Direct ADC[2, 3] (%)
	16 V (AT)	16 V (12 V) (AV/AC)	8 V (AV/AC)	4 V (AT)	4 V (AV/AC)	2 V (AV/AC)	1 V (AV/AC)	VAREF = 2.56 V
15	1							
14	1							
12	1	1						
5	2	2	1					
3.3	2	2	1	1	1			
2.5	3	2	1	1	1			1
1.8	4	4	1	1	1	1		1
1.5	5	5	2	2	2	1		1
1.2	7	6	2	2	2	1		1
0.9	9	9	4	4	3	1	1	1

Notes:

1. *Requires enabling Analog Calibration in the Microsemi tool flow.*
2. *Direct ADC mode using an external VAREF of 2.56V±4.6mV, without Analog Calibration macro.*
3. *For input greater than 2.56 V, the ADC output will saturate. A higher VAREF or prescaler usage is recommended.*

Microprocessor-Based Design Flow

In a microprocessor-based design flow, designers can use CoreAI (Analog Interface) to interface and control the analog peripherals within the Fusion device family. Designers using Core8051, CoreMP7, or Cortex™-M1 with CoreAI can take advantage of the CoreAI Driver (provided in C code) to support the calibration features.

The CoreAI driver provides a set of Application Program Interface (API) functions to support different calibration modes and automatically calculate the final calibrated value, based on the m and c coefficient stored in the spare page of the Fusion flash memory block.

Currently, the calibration scheme only supports voltage monitor applications (AVx pins) and is only needed for prescaled voltage inputs. If direct analog input sampling accuracy is well within 1%, the Microsemi calibration solution does not offer any additional benefits. CoreAI driver must be used with CoreAI version 2.1 (or higher) and CoreAhbNvm version 1.3.135 (or higher). Refer to the *CoreAI Driver User's Guide* for more information.

Performing System-Level Calibration Using Fusion

Previous sections of this document describe the general approach to calibration using the offset-only and offset-plus-gain approaches, and provided a detailed explanation of Microsemi's calibration solution for Fusion voltage input signals. In addition to this solution, users may desire calibration of temperature and current inputs, as well as calibration of the entire system working together. A recommended approach to accomplishing these tasks is provided below. This methodology may be added in the user's design on top of the device-level calibration solution.

The "Calibration Measurements" section on page 297 explains how the calibration coefficients, as described in EQ 12-1 and EQ 12-2 on page 296, are calculated. EQ 12-1 and EQ 12-2 on page 296 (depending on the calibration method used) are implemented using an adder (one-point calibration) or a combination of an adder and a multiplier (two-point calibration), as shown in Figure 12-9.

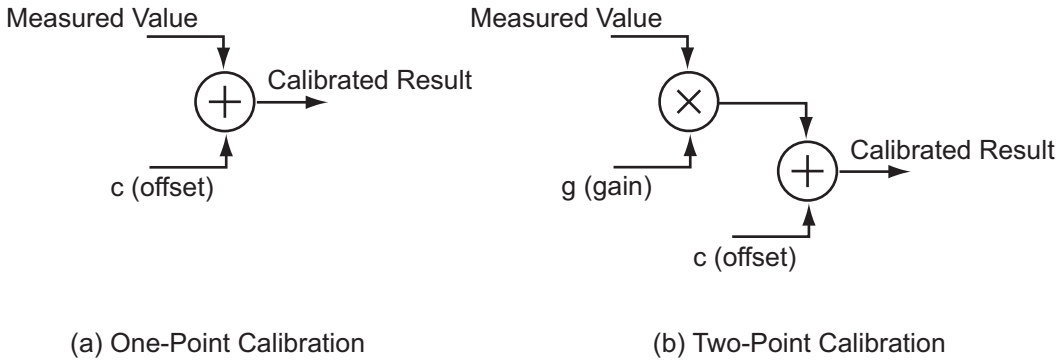


***Figure 12-9* • Implementation of One-Point and Two-Point Calibration**

The calibration coefficients (m and c in Figure 12-9) can be stored in the nonvolatile flash memory of each Fusion device. Therefore, inside the flash memory architecture, different memory addresses can contain the calibration coefficients for each channel in the design. Depending on which channel in the design is performing the measurement at the time, appropriate calibration coefficients can be fetched and fed into the adder and/or multiplier. For details on writing and reading to the Fusion flash memory block, refer to the "Fusion Embedded Flash Memory Blocks" section on page 137.

Where and how the adder and multiplier are implemented depends highly on the application and the user's design. Generally, there are three ways designers can implement the arithmetic functions in Figure 12-9 to produce the calibrated results of measurement. The following explains each of these implementations and their pros and cons:

Implementing a Dedicated Adder and Multiplier Using FPGA Core Gates

Pros

- High speed
- Efficient for one-point calibration

Cons

- Multiplier implementation consumes large number of gates

Using the Design's Microprocessor/Microcontroller ALU to Perform Calibration Calculations

Pros

- Saves FPGA gate resources compared to implementing dedicated multiplier

Cons

- May slow down microprocessor's performance depending on the overall sampling rate

Implementing the Numerical Calculations in Application Software

Pros

- Saves FPGA gates

Cons

- Depending on the application, may take up a lot of bandwidth from the host processor
- Only suitable for applications where there is software communicating with hardware

Conclusion

Designers use calibration to increase the accuracy achievable in applications involving analog components. Microsemi Fusion mixed signal FPGAs offer the capability of measuring analog voltage, current, and temperature. If customers require more accuracy than is inherent in Fusion FPGAs, calibration techniques can be used to achieve these requirements. Fusion FPGAs offer the advantage of having the calibration design and coefficients programmed into the FPGA itself without a need for any external components. This document discusses the typical calibration techniques and the implementation of Microsemi calibration solutions for Fusion FPGAs. The result shows that with the Microsemi calibration implementation, customers can achieve 1% ADC sampling accuracy for all Fusion devices and all channels.

Related Documents

Datasheets

Fusion Family of Mixed Signal Flash FPGAs

http://www.microsemi.com/soc/documents/Fusion_DS.pdf

List of Changes

The following table lists critical changes that were made in each revision of the document.

Date	Changes	Page
August 2012	The "Calibration Solution" section was modified to add the following information (SAR 30394): Although calibration is not done at production, the offset value for the temperature monitor is stored in eNVM along with other calibrated values. If CalibIP is used, the offset error is removed from the temperature monitor reading.	299
v1.0 (October 2008)	The "Microprocessor-Based Design Flow" section was added.	307
51900161-3/6.08 (June 2008)	This statement was added to the "Calibration IP Deployment" section: Microsemi's calibration IP solution is not available for processor systems that use the CoreAI to interface to the analog block.	299
	In the "Performing System-Level Calibration Using Fusion" section, a reference to the Fusion Handbook was added.	307
51900161-1/2.08 (February 2008)	Please read the document very carefully. Much information was added to the document.	N/A
	The "Introduction" section was updated.	295
	The heading title "Calibration Measurements" section is new and all subsections were significantly updated. Please note the variables in all equations were changed. In addition, the variable g was changed to m throughout the document.	297
	The "Calibration Solution" section and all subsections are new.	299
	The heading title "Implementing Calibration Design" was changed to "Performing System-Level Calibration Using Fusion" section. In Figure 12-9 • Implementation of One-Point and Two-Point Calibration, the m was changed to g.	307

13 – I/O Software Control in Low Power Flash Devices

Fusion, IGLOO, and ProASIC3 I/Os provide more design flexibility, allowing the user to control specific features by enabling certain I/O standards. Some features are selectable only for certain I/O standards, whereas others are available for all I/O standards. For example, slew control is not supported by differential I/O standards. Conversely, I/O register combining is supported by all I/O standards. For detailed information about which I/O standards and features are available on each device and each I/O type, refer to the I/O Structures section of the handbook for the device you are using.

Figure 13-1 shows the various points in the software design flow where a user can provide input or control of the I/O selection and parameters. A detailed description is provided throughout this document.

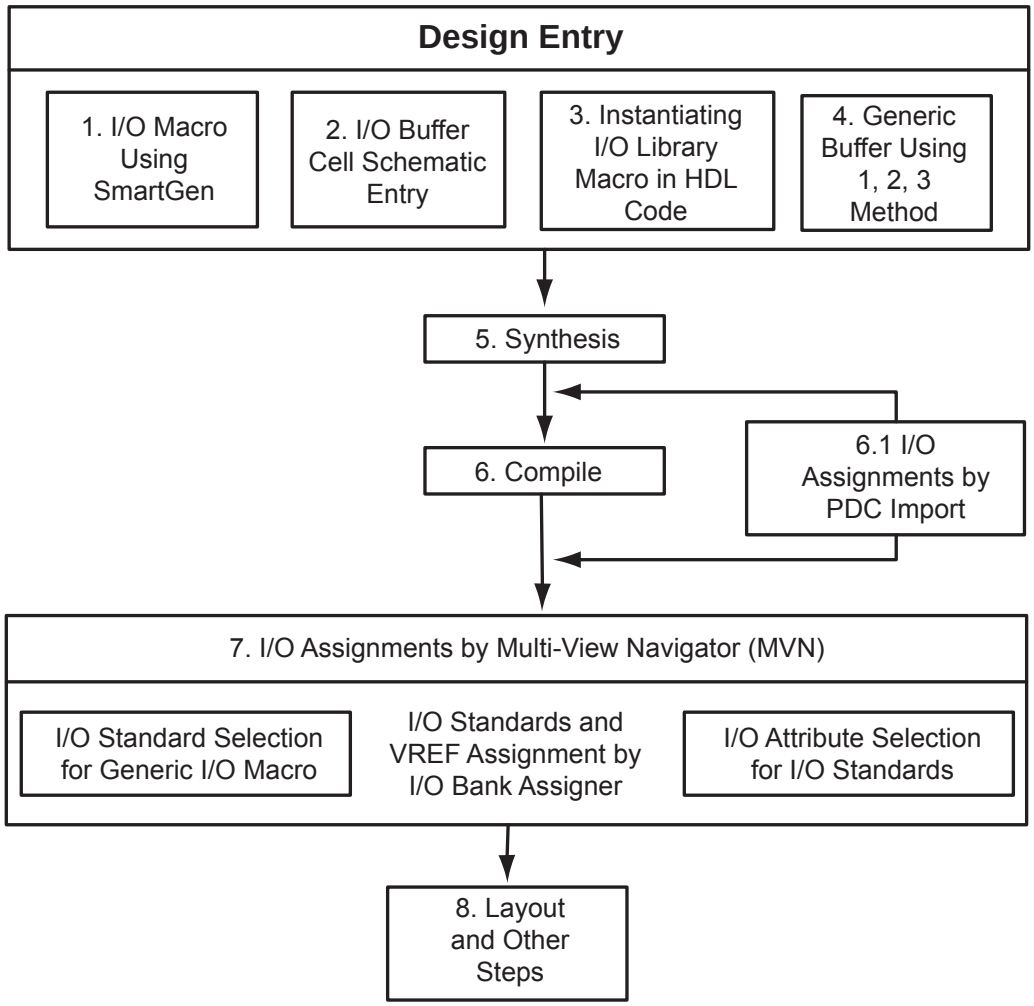


***Figure 13-1* • User I/O Assignment Flow Chart**

Flash FPGAs I/O Support

The flash FPGAs listed in Table 13-1 support I/Os and the functions described in this document.

***Table 13-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 13-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 13-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Software-Controlled I/O Attributes

Users may modify these programmable I/O attributes using the I/O Attribute Editor. Modifying an I/O attribute may result in a change of state in Designer. Table 13-2 details which steps have to be re-run as a function of modified I/O attribute.

***Table 13-2* • Designer State (resulting from I/O attribute modification)**

	Designer States[1]				
I/O Attribute	**Compile**	**Layout**	**Fuse**	**Timing**	**Power**
Slew Control[2]	No	No	Yes	Yes	Yes
Output Drive (mA)	No	No	Yes	Yes	Yes
Skew Control	No	No	Yes	Yes	Yes
Resistor Pull	No	No	Yes	Yes	Yes
Input Delay	No	No	Yes	Yes	Yes
Schmitt Trigger	No	No	Yes	Yes	Yes
OUT_LOAD	No	No	No	Yes	Yes
COMBINE_REGISTER	Yes	Yes	N/A	N/A	N/A

Notes:

1. *No = Remains the same, Yes = Re-run the step, N/A = Not applicable*
2. *Skew control does not apply to IGLOO nano, IGLOO PLUS, and ProASIC3 nano devices.*
3. *Programmable input delay is applicable only for ProASIC3E, ProASIC3EL, RT ProASIC3, and IGLOOe devices.*

Implementing I/Os in Microsemi Software

Microsemi Libero SoC software is integrated with design entry tools such as the SmartGen macro builder, the ViewDraw schematic entry tool, and an HDL editor. It is also integrated with the synthesis and Designer tools. In this section, all necessary steps to implement the I/Os are discussed.

Design Entry

There are three ways to implement I/Os in a design:

1. Use the SmartGen macro builder to configure I/Os by generating specific I/O library macros and then instantiating them in top-level code. This is especially useful when creating I/O bus structures.
2. Use an I/O buffer cell in a schematic design.
3. Manually instantiate specific I/O macros in the top-level code.

If technology-specific macros, such as INBUF_LVCMOS33 and OUTBUF_PCI, are used in the HDL code or schematic, the user will not be able to change the I/O standard later on in Designer. If generic I/O macros are used, such as INBUF, OUTBUF, TRIBUF, CLKBUF, and BIBUF, the user can change the I/O standard using the Designer I/O Attribute Editor tool.

Using SmartGen for I/O Configuration

The SmartGen tool in Libero SoC provides a GUI-based method of configuring the I/O attributes. The user can select certain I/O attributes while configuring the I/O macro in SmartGen. The steps to configure an I/O macro with specific I/O attributes are as follows:

1. Open Libero SoC.
2. On the left-hand side of the Catalog View, select **I/O**, as shown in Figure 13-2.

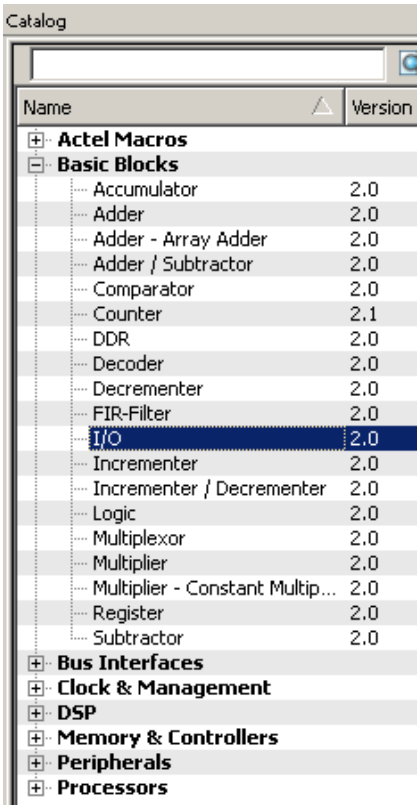


Figure 13-2 • **SmartGen Catalog**

3. Double-click I/O to open the Create Core window, which is shown in Figure 13-3).

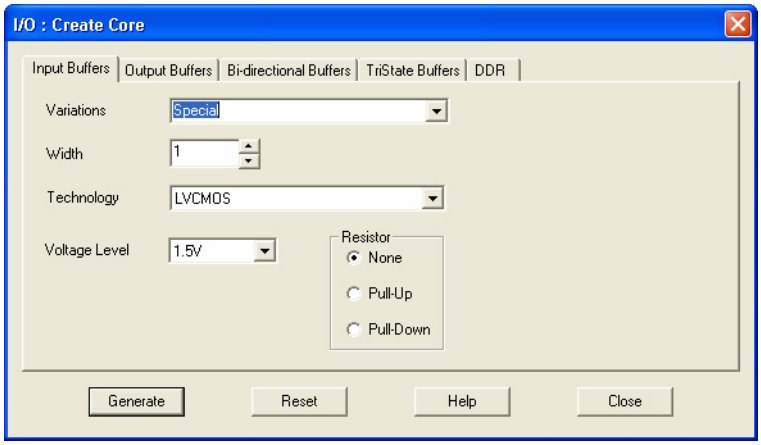


Figure 13-3 • **I/O Create Core Window**

As seen in Figure 13-3, there are five tabs to configure the I/O macro: Input Buffers, Output Buffers, Bidirectional Buffers, Tristate Buffers, and DDR.

Input Buffers

There are two variations: Regular and Special.

If the **Regular** variation is selected, only the Width (1 to 128) needs to be entered. The default value for Width is 1.

The **Special** variation has Width, Technology, Voltage Level, and Resistor Pull-Up/-Down options (see Figure 13-3). All the I/O standards and supply voltages (V_{CCI}) supported for the device family are available for selection.

Output Buffers

There are two variations: Regular and Special.

If the **Regular** variation is selected, only the Width (1 to 128) needs to be entered. The default value for Width is 1.

The **Special** variation has Width, Technology, Output Drive, and Slew Rate options.

Bidirectional Buffers

There are two variations: Regular and Special.

The **Regular** variation has Enable Polarity (Active High, Active Low) in addition to the Width option.

The **Special** variation has Width, Technology, Output Drive, Slew Rate, and Resistor Pull-Up/-Down options.

Tristate Buffers

Same as Bidirectional Buffers.

DDR

There are eight variations: DDR with Regular Input Buffers, Special Input Buffers, Regular Output Buffers, Special Output Buffers, Regular Tristate Buffers, Special Tristate Buffers, Regular Bidirectional Buffers, and Special Bidirectional Buffers.

These variations resemble the options of the previous I/O macro. For example, the Special Input Buffers variation has Width, Technology, Voltage Level, and Resistor Pull-Up/-Down options. DDR is not available on IGLOO PLUS devices.

4. Once the desired configuration is selected, click the **Generate** button. The Generate Core window opens (Figure 13-4).
5. Enter a name for the macro. Click **OK**. The core will be generated and saved to the appropriate location within the project files (Figure 13-5 on page 317).

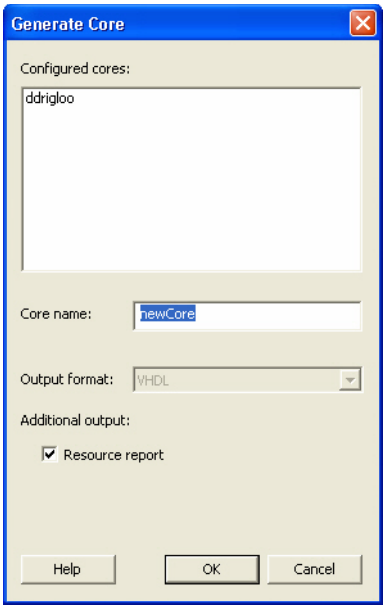


Figure 13-4 • **Generate Core Window**

6. Instantiate the I/O macro in the top-level code.

   The user must instantiate the DDR_REG or DDR_OUT macro in the design. Use SmartGen to generate both these macros and then instantiate them in your top level. To combine the DDR macros with the I/O, the following rules must be met:

Rules for the DDR I/O Function

- The fanout between an I/O pin (D or Y) and a DDR (DDR_REG or DDR_OUT) macro must be equal to one for the combining to happen on that pin.
- If a DDR_REG macro and a DDR_OUT macro are combined on the same bidirectional I/O, they must share the same clear signal.
- Registers will not be combined in an I/O in the presence of DDR combining on the same I/O.

Using the I/O Buffer Schematic Cell

Libero SoC software includes the ViewDraw schematic entry tool. Using ViewDraw, the user can insert any supported I/O buffer cell in the top-level schematic. Figure 13-5 shows a top-level schematic with different I/O buffer cells. When synthesized, the netlist will contain the same I/O macro.

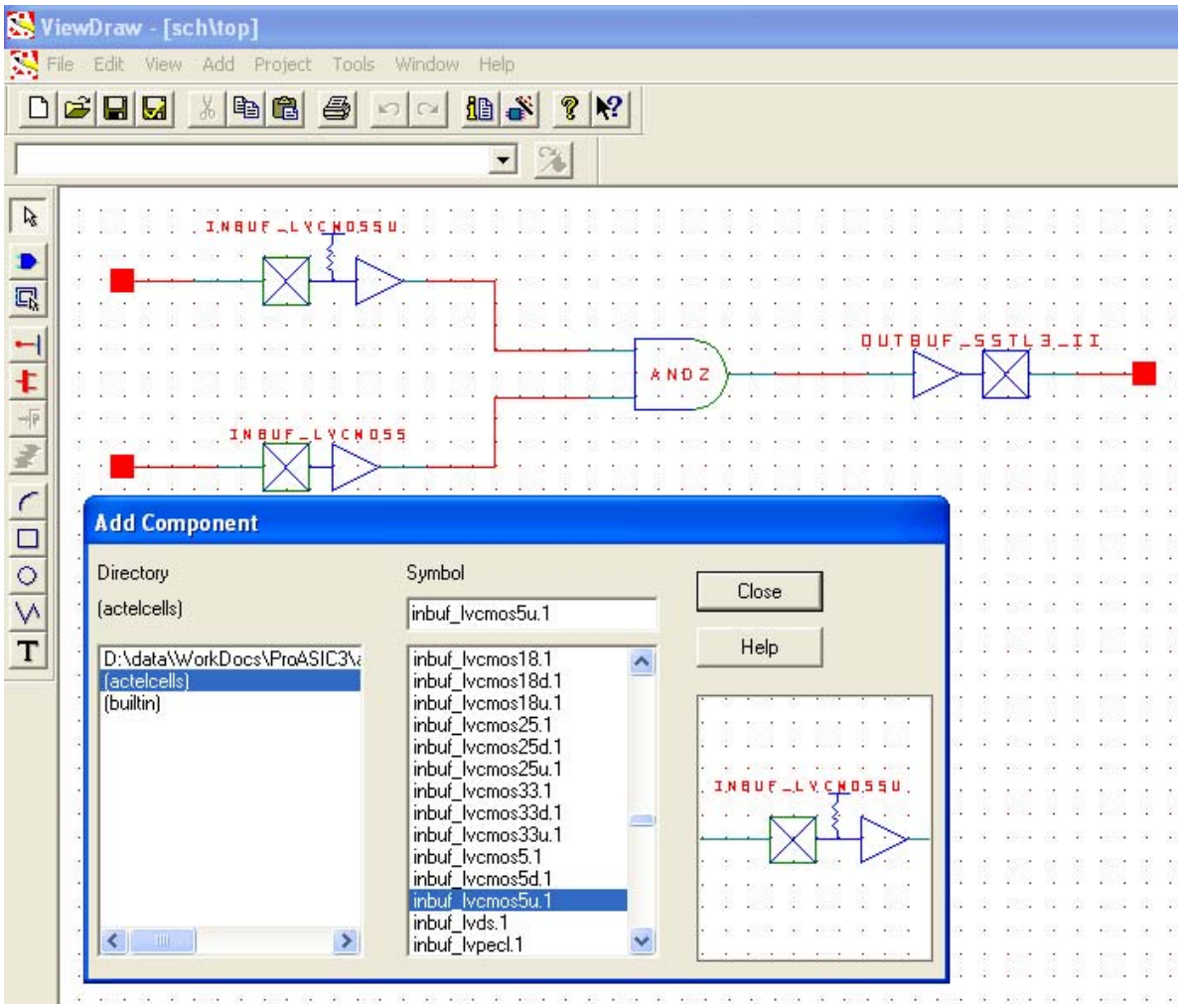


Figure 13-5 • **I/O Buffer Schematic Cell Usage**

Instantiating in HDL code

All the supported I/O macros can be instantiated in the top-level HDL code (refer to the *IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide* for a detailed list of all I/O macros). The following is an example:

```
library ieee;
use ieee.std_logic_1164.all;
library proasic3e;

entity TOP is
  port(IN2, IN1 : in std_logic; OUT1 : out std_logic);
end TOP;

architecture DEF_ARCH of TOP is

  component INBUF_LVCMOS5U
    port(PAD : in std_logic := 'U'; Y : out std_logic);
  end component;

  component INBUF_LVCMOS5
    port(PAD : in std_logic := 'U'; Y : out std_logic);
  end component;

  component OUTBUF_SSTL3_II
    port(D : in std_logic := 'U'; PAD : out std_logic);
  end component;

  Other component …..

signal x, y, z…….other signals : std_logic;

begin

  I1 : INBUF_LVCMOS5U
    port map(PAD => IN1, Y =>x);
  I2 : INBUF_LVCMOS5
    port map(PAD => IN2, Y => y);
  I3 : OUTBUF_SSTL3_II
    port map(D => z, PAD => OUT1);

  other port mapping…

end DEF_ARCH;
```

Synthesizing the Design

Libero SoC integrates with the Synplify® synthesis tool. Other synthesis tools can also be used with Libero SoC. Refer to the *Libero SoC User's Guide* or Libero online help for details on how to set up the Libero tool profile with synthesis tools from other vendors.

During synthesis, the following rules apply:

- Generic macros:
  - Users can instantiate generic INBUF, OUTBUF, TRIBUF, and BIBUF macros.
  - Synthesis will automatically infer generic I/O macros.
  - The default I/O technology for these macros is LVTTL.
  - Users will need to use the I/O Attribute Editor in Designer to change the default I/O standard if needed (see Figure 13-6 on page 319).
- Technology-specific I/O macros:
  - Technology-specific I/O macros, such as INBUF_LVCMO25 and OUTBUF_GTL25, can be instantiated in the design. Synthesis will infer these I/O macros in the netlist.

– The I/O standard of technology-specific I/O macros cannot be changed in the I/O Attribute Editor (see Figure 13-6).
– The user MUST instantiate differential I/O macros (LVDS/LVPECL) in the design. This is the only way to use these standards in the design (IGLOO nano and ProASIC3 nano devices do not support differential inputs).
– To implement the DDR I/O function, the user must instantiate a DDR_REG or DDR_OUT macro. This is the only way to use a DDR macro in the design.

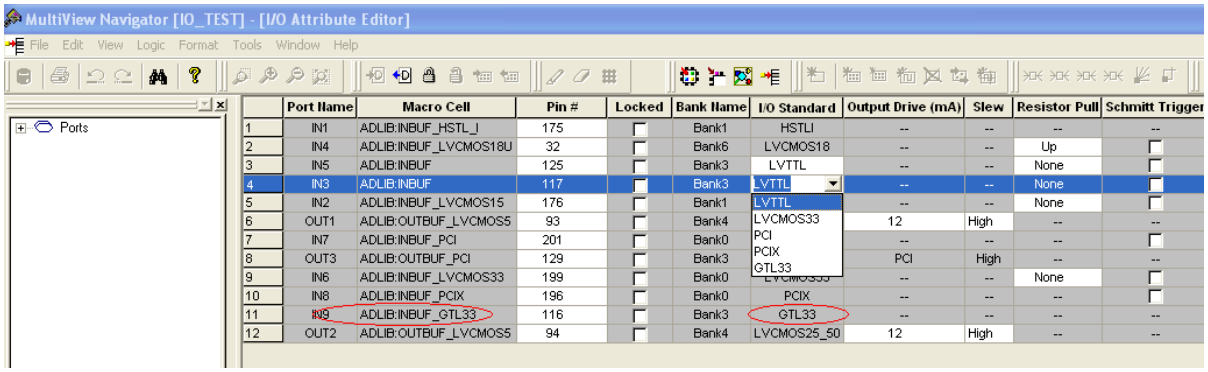


Figure 13-6 • **Assigning a Different I/O Standard to the Generic I/O Macro**

Performing Place-and-Route on the Design

The netlist created by the synthesis tool should now be imported into Designer and compiled. During Compile, the user can specify the I/O placement and attributes by importing the PDC file. The user can also specify the I/O placement and attributes using ChipPlanner and the I/O Attribute Editor under MVN.

Defining I/O Assignments in the PDC File

A PDC file is a Tcl script file specifying physical constraints. This file can be imported to and exported from Designer.

Table 13-3 shows I/O assignment constraints supported in the PDC file.

Table 13-3 • **PDC I/O Constraints**

Command	Action	Example	Comment
I/O Banks Setting Constraints			

set_iobank	Sets the I/O supply voltage, V_{CCI}, and the input reference voltage, V_{REF}, for the specified I/O bank.	`set_iobank bankname [-vcci vcci_voltage] [-vref vref_voltage]`
		
		`set_iobank Bank7 -vcci 1.50 -vref 0.75`
			Must use in case of mixed I/O voltage (V_{CCI}) design

set_vref	Assigns a V_{REF} pin to a bank.	`set_vref -bank [bankname] [pinnum]`
		
		`set_vref -bank Bank0 685 704 723 742 761`
			Must use if voltage-referenced I/Os are used

set_vref_defaults	Sets the default V_{REF} pins for the specified bank. This command is ignored if the bank does not need a V_{REF} pin.	`set_vref_defaults bankname`
		
		`set_vref_defaults bank2`	

Note: Refer to the Libero SoC User's Guide for detailed rules on PDC naming and syntax conventions.

***Table 13-3* • PDC I/O Constraints (continued)**

Command	Action	Example	Comment
I/O Attribute Constraint			

set_io	Sets the attributes of an I/O	`set_io portname`
		`[-pinname value]`
		`[-fixed value]`
		`[-iostd value]`
		`[-out_drive value]`
		`[-slew value]`
		`[-res_pull value]`
		`[-schmitt_trigger value]`
		`[-in_delay value]`
		`[-skew value]`
		`[-out_load value]`
		`[-register value]`
		
		`set_io IN2 -pinname 28`
		`-fixed yes -iostd LVCMOS15`
		`-out_drive 12 -slew high`
		`-RES_PULL None`
		`-SCHMITT_TRIGGER Off`
		`-IN_DELAY Off –skew off`
		`-REGISTER No`
			If the I/O macro is generic (e.g., INBUF) or technology-specific (INBUF_LVCMOS25), then all I/O attributes can be assigned using this constraint.
			If the netlist has an I/O macro that specifies one of its attributes, that attribute cannot be changed using this constraint, though other attributes can be changed.
			Example: OUTBUF_S_24 (low slew, output drive 24 mA)
			Slew and output drive cannot be changed.

I/O Region Placement Constraints			

define_region	Defines either a rectangular region or a rectilinear region	`define_region`
		`-name [region_name]`
		`-type [region_type] x1 y1 x2 y2`
		
		`define_region -name test`
		`-type inclusive 0 15 2 29`
			If any number of I/Os must be assigned to a particular I/O region, such a region can be created with this constraint.

assign_region	Assigns a set of macros to a specified region	`assign_region [region name]`
		`[macro_name...]`
		
		`assign_region test U12`
			This constraint assigns I/O macros to the I/O regions. When assigning an I/O macro, PDC naming conventions must be followed if the macro name contains special characters; e.g., if the macro name is \\$1I19\\, the correct use of escape characters is \\\\\$1I19\\\\.

Note: Refer to the Libero SoC User's Guide *for detailed rules on PDC naming and syntax conventions.*

Compiling the Design

During Compile, a PDC I/O constraint file can be imported along with the netlist file. If only the netlist file is compiled, certain I/O assignments need to be completed before proceeding to Layout. All constraints that can be entered in PDC can also be entered using ChipPlanner, I/O Attribute Editor, and PinEditor.

There are certain rules that must be followed in implementing I/O register combining and the I/O DDR macro (refer to the I/O Registers section of the handbook for the device that you are using and the "DDR" section on page 316 for details). Provided these rules are met, the user can enable or disable I/O register combining by using the PDC command `set_io portname –register yes|no` in the I/O Attribute Editor or selecting a check box in the Compile Options dialog box (see Figure 13-7). The Compile Options dialog box appears when the design is compiled for the first time. It can also be accessed by choosing **Options** > **Compile** during successive runs. I/O register combining is off by default. The PDC command overrides the setting in the Compile Options dialog box.

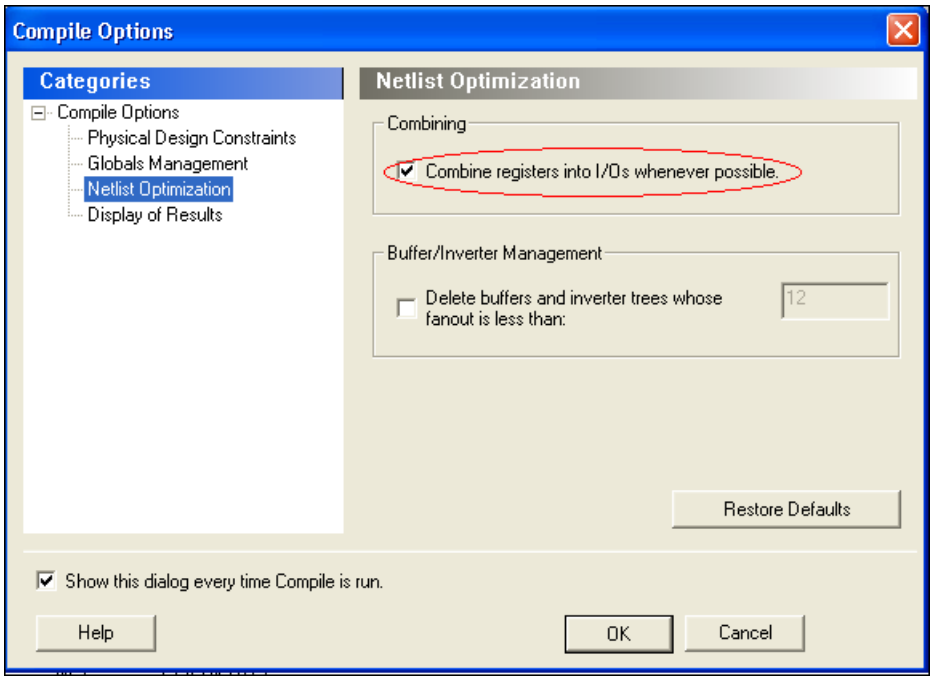


Figure 13-7 • **Setting Register Combining During Compile**

Understanding the Compile Report

The I/O bank report is generated during Compile and displayed in the log window. This report lists the I/O assignments necessary before Layout can proceed.

When Designer is started, the I/O Bank Assigner tool is run automatically if the Layout command is executed. The I/O Bank Assigner takes care of the necessary I/O assignments. However, these assignments can also be made manually with MVN or by importing the PDC file. Refer to the "Assigning Technologies and VREF to I/O Banks" section on page 324 for further description.

The I/O bank report can also be extracted from Designer by choosing **Tools** > **Report** and setting the Report Type to **IOBank**.

This report has the following tables: I/O Function, I/O Technology, I/O Bank Resource Usage, and I/O Voltage Usage. This report is useful if the user wants to do I/O assignments manually.

I/O Function

Figure 13-8 shows an example of the I/O Function table included in the I/O bank report:

I/O Function:

Type	w/o register	w/ register	w/ DDR register
Input I/O	7	0	1
Output I/O	1	1	0
Bidirectional I/O	0	0	0
Differential Input I/O Pairs	0	0	0
Differential Output I/O Pairs	0	0	1

Figure 13-8 • **I/O Function Table**

This table lists the number of input I/Os, output I/Os, bidirectional I/Os, and differential input and output I/O pairs that use I/O and DDR registers.

Note: IGLOO nano and ProASIC3 nano devices do not support differential inputs.

Certain rules must be met to implement registered and DDR I/O functions (refer to the I/O Structures section of the handbook for the device you are using and the "DDR" section on page 316).

I/O Technology

The I/O Technology table (shown in Figure 13-9) gives the values of VCCI and VREF (reference voltage) for all the I/O standards used in the design. The user should assign these voltages appropriately.

I/O Technology:

	Voltages		I/Os		
I/O Standard(s)	Vcci	Vref	Input	Output	Bidirectional
LVTTL	3.30v	N/A	1	1	0
LVCMOS33	3.30v	N/A	1	0	0
LVCMOS25_50	2.50v	N/A	1	1	0
LVCMOS18	1.80v	N/A	1	0	0
LVCMOS15	1.50v	N/A	1	0	0
PCIX	3.30v	N/A	1	0	0
LVDS	2.50v	N/A	0	2	0
SSTL3I (Input/Bidirectional)	3.30v	1.50v	1	0	0
GTLP33 (Input/Bidirectional)	3.30v	1.00v	1	0	0

Figure 13-9 • **I/O Technology Table**

I/O Bank Resource Usage

This is an important portion of the report. The user must meet the requirements stated in this table. Figure 13-10 shows the I/O Bank Resource Usage table included in the I/O bank report:

```
I/O Bank Resource Usage:

          |   Voltages    | Single I/Os  | Diff I/O Pairs |        Vref I/Os
          |-------|-------|------|-------|-------|--------|------|-------|----------
          | Vcci  | Vref  | Used | Total | Used  | Total  | Used | Total | Vref Pins
    ------|-------|-------|------|-------|-------|--------|------|-------|----------
    Bank0 | N/A   | N/A   | 0    | 25    | 0     | 12     | N/A  | N/A   | N/A
    Bank1 | N/A   | N/A   | 0    | 15    | 0     | 7      | N/A  | N/A   | N/A
    Bank2 | N/A   | N/A   | 0    | 17    | 0     | 6      | N/A  | N/A   | N/A
    Bank3 | N/A   | N/A   | 0    | 16    | 0     | 7      | N/A  | N/A   | N/A
    Bank4 | N/A   | N/A   | 0    | 15    | 0     | 7      | N/A  | N/A   | N/A
    Bank5 | N/A   | N/A   | 0    | 22    | 0     | 10     | N/A  | N/A   | N/A
    Bank6 | N/A   | N/A   | 0    | 19    | 0     | 9      | N/A  | N/A   | N/A
    Bank7 | N/A   | N/A   | 0    | 18    | 0     | 7      | N/A  | N/A   | N/A

Warning: IOPRL1: 8 I/O Bank(s) have not been assigned any voltages.
         The I/O modules located in these banks cannot be assigned any I/O macro.
```

Figure 13-10 • **I/O Bank Resource Usage Table**

The example in Figure 13-10 shows that none of the I/O macros is assigned to the bank because more than one VCCI is detected.

I/O Voltage Usage

The I/O Voltage Usage table provides the number of VREF (E devices only) and V_{CCI} assignments required in the design. If the user decides to make I/O assignments manually (PDC or MVN), the issues listed in this table must be resolved before proceeding to Layout. As stated earlier, VREF assignments must be made if there are any voltage-referenced I/Os.

Figure 13-11 shows the I/O Voltage Usage table included in the I/O bank report.

```
I/O Voltage Usage:

      Voltages    |     I/Os
    -------|-------|------|-------
     Vcci  | Vref  | Used | Total
    -------|-------|------|-------
     1.50v | N/A   | 1*   | 0
     1.80v | N/A   | 1*   | 0
     2.50v | N/A   | 4*   | 0
     3.30v | N/A   | 6*   | 0
     3.30v | 1.00v | 1*   | 0
     3.30v | 1.50v | 1*   | 0

Warning: IOPRL3: This design has infeasible I/O voltage requirement(s),
         which are indicated with a '*' in the I/O Voltage Usage table.
         Please consider importing a Physical Design Constraint (PDC) file or
         use the MultiView Navigator (MVN) to resolve the design's voltage requirements
         before running Layout.
```

Figure 13-11 • **I/O Voltage Usage Table**

The table in Figure 13-11 indicates that there are two voltage-referenced I/Os used in the design. Even though both of the voltage-referenced I/O technologies have the same VCCI voltage, their VREF voltages are different. As a result, two I/O banks are needed to assign the VCCI and VREF voltages.

In addition, there are six single-ended I/Os used that have the **same VCCI voltage. Since two banks are already assigned with the same VCCI voltage and there are enough u**nused bonded I/Os in

those banks, the user does not need to assign the same VCCI voltage to another bank. The user needs to assign the other three VCCI voltages to three more banks.

Assigning Technologies and VREF to I/O Banks

Low power flash devices offer a wide variety of I/O standards, including voltage-referenced standards. Before proceeding to Layout, each bank must have the required VCCI voltage assigned for the corresponding I/O technologies used for that bank. The voltage-referenced standards require the use of a reference voltage (VREF). This assignment can be done manually or automatically. The following sections describe this in detail.

Manually Assigning Technologies to I/O Banks

The user can import the PDC at this point and resolve this requirement. The PDC command is

```
set_iobank [bank name] –vcci [vcci value]
```

Another method is to use the I/O Bank Settings dialog box (**MVN** > **Edit** > **I/O Bank Settings**) to set up the V_{CCI} voltage for the bank (Figure 13-12).

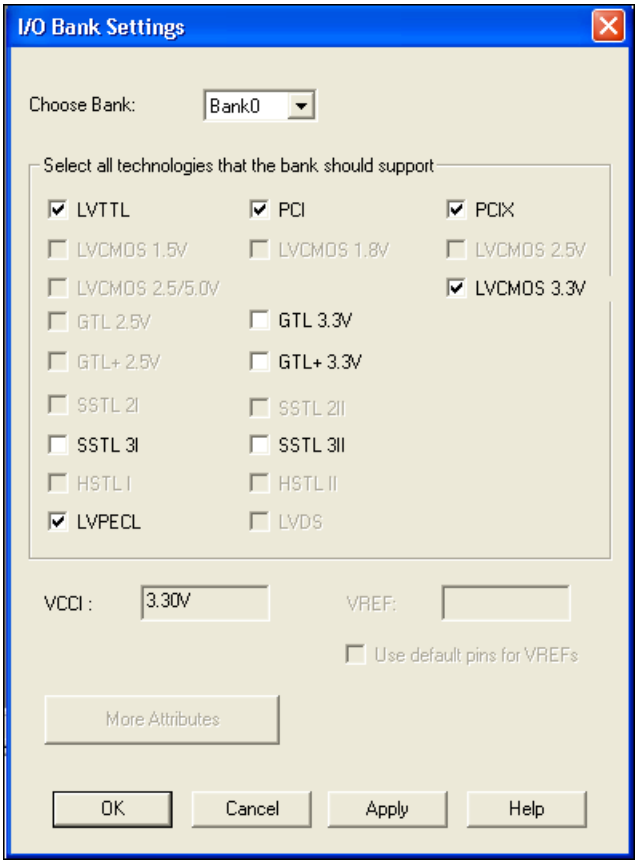


Figure 13-12 • **Setting VCCI for a Bank**

The procedure is as follows:

1. Select the bank to which you want VCCI to be assigned from the **Choose Bank** list.
2. Select the I/O standards for that bank. If you select any standard, the tool will automatically show all compatible standards that have a common VCCI voltage requirement.
3. Click **Apply**.
4. Repeat steps 1–3 to assign VCCI voltages to other banks. Refer to Figure 13-11 on page 323 to find out how many I/O banks are needed for VCCI bank assignment.

Manually Assigning VREF Pins

Voltage-referenced inputs require an input reference voltage (VREF). The user must assign VREF pins before running Layout. Before assigning a VREF pin, the user must set a VREF technology for the bank to which the pin belongs.

VREF Rules for the Implementation of Voltage-Referenced I/O Standards

The VREF rules are as follows:

1. Any I/O (except JTAG I/Os) can be used as a V_{REF} pin.
2. One V_{REF} pin can support up to 15 I/Os. It is recommended, but not required, that eight of them be on one side and seven on the other side (in other words, all 15 can still be on one side of VREF).
3. SSTL3 (I) and (II): Up to 40 I/Os per north or south bank in any position
4. LVPECL / GTL+ 3.3 V / GTL 3.3 V: Up to 48 I/Os per north or south bank in any position (not applicable for IGLOO nano and ProASIC3 nano devices)
5. SSTL2 (I) and (II) / GTL+ 2.5 V / GTL 2.5 V: Up to 72 I/Os per north or south bank in any position
6. VREF minibanks partition rule: Each I/O bank is physically partitioned into VREF minibanks. The VREF pins within a VREF minibank are interconnected internally, and consequently, only one VREF voltage can be used within each VREF minibank. If a bank does not require a VREF signal, the VREF pins of that bank are available as user I/Os.
7. The first VREF minibank includes all I/Os starting from one end of the bank to the first power triple and eight more I/Os after the power triple. Therefore, the first VREF minibank may contain (0 + 8), (2 + 8), (4 + 8), (6 + 8), or (8 + 8) I/Os.

   The second VREF minibank is adjacent to the first VREF minibank and contains eight I/Os, a power triple, and eight more I/Os after the triple. An analogous rule applies to all other VREF minibanks but the last.

   The last VREF minibank is adjacent to the previous one but contains eight I/Os, a power triple, and all I/Os left at the end of the bank. This bank may also contain (8 + 0), (8 + 2), (8 + 4), (8 + 6), or (8 + 8) available I/Os.

   Example:

   4 I/Os → Triple → 8 I/Os, 8 I/Os → Triple → 8 I/Os, 8 I/Os → Triple → 2 I/Os

   That is, minibank A = (4 + 8) I/Os, minibank B = (8 + 8) I/Os, minibank C = (8 + 2) I/Os.
8. Only minibanks that contain input or bidirectional I/Os require a VREF. A VREF is not needed for minibanks composed of output or tristated I/Os.

Assigning the VREF Voltage to a Bank

When importing the PDC file, the VREF voltage can be assigned to the I/O bank. The PDC command is as follows:

```
set_iobank –vref [value]
```

Another method for assigning VREF is by using **MVN** > **Edit** > **I/O Bank Settings** (Figure 13-13 on page 326).

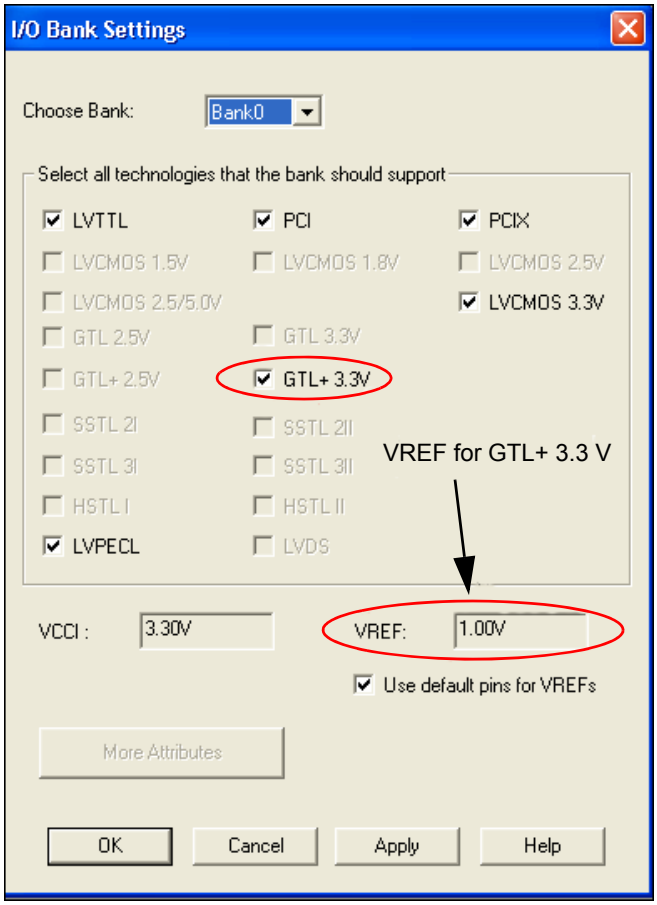


***Figure 13-13* • Selecting VREF Voltage for the I/O Bank**

Assigning VREF Pins for a Bank

The user can use default pins for VREF. In this case, select the **Use default pins for VREFs** check box (Figure 13-13). This option guarantees full VREF coverage of the bank. The equivalent PDC command is as follows:

```
set_vref_default [bank name]
```

To be able to choose VREF pins, adequate VREF pins must be created to allow legal placement of the compatible voltage-referenced I/Os.

To assign VREF pins manually, the PDC command is as follows:

```
set_vref –bank [bank name] [package pin numbers]
```

For ChipPlanner/PinEditor to show the range of a VREF pin, perform the following steps:

1. Assign VCCI to a bank using **MVN** > **Edit** > **I/O Bank Settings**.
2. Open **ChipPlanner**. Zoom in on an I/O package pin in that bank.
3. Highlight the pin and then right-click. Choose **Use Pin for VREF**.

4. Right-click and then choose **Highlight VREF range**. All the pins covered by that VREF pin will be highlighted (Figure 13-14).

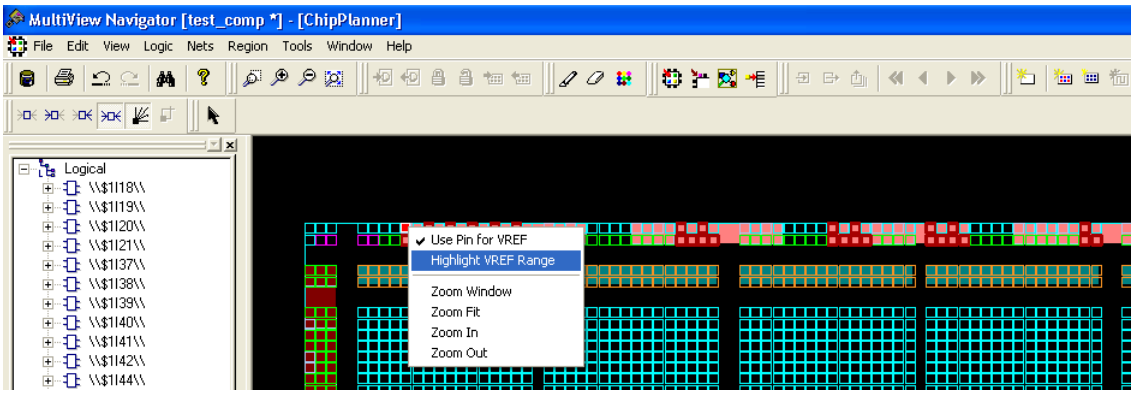


Figure 13-14 • **VREF Range**

Using PinEditor or ChipPlanner, VREF pins can also be assigned (Figure 13-15).

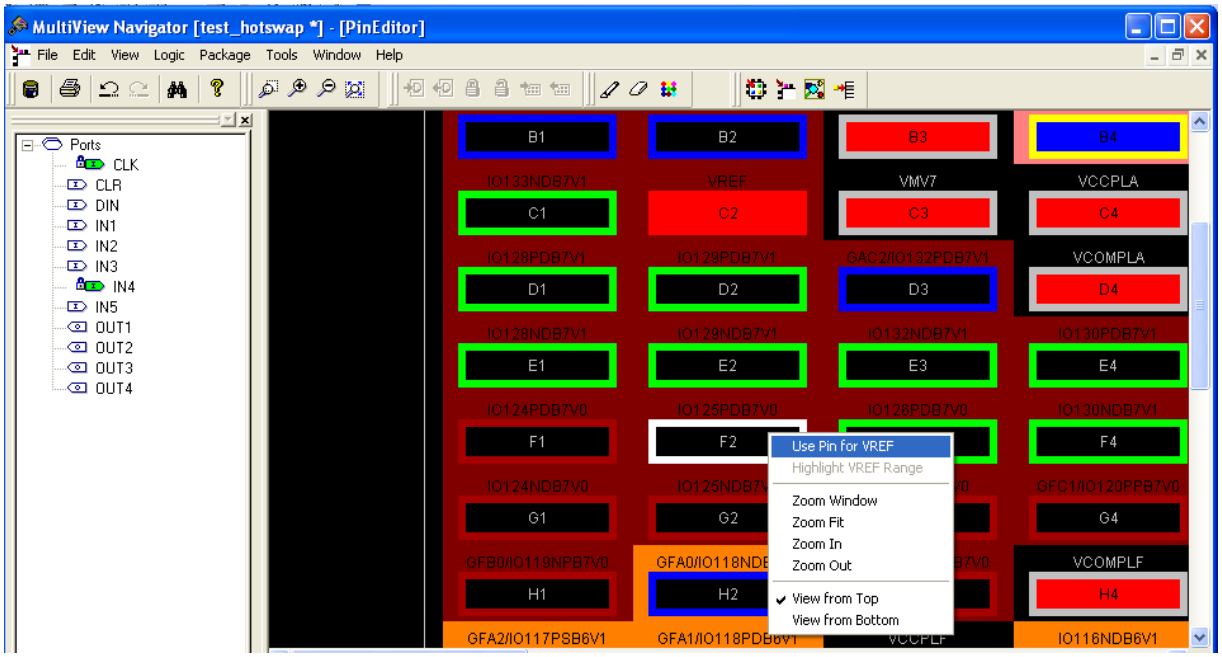


Figure 13-15 • **Assigning VREF from PinEditor**

To unassign a VREF pin:

1. Select the pin to unassign.
2. Right-click and choose **Use Pin for VREF.** The check mark next to the command disappears. The VREF pin is now a regular pin.

Resetting the pin may result in unassigning I/O cores, even if they are locked. In this case, a warning message appears so you can cancel the operation.

After you assign the VREF pins, right-click a VREF pin and choose **Highlight VREF Range** to see how many I/Os are covered by that pin. To unhighlight the range, choose **Unhighlight All** from the **Edit** menu.

Automatically Assigning Technologies to I/O Banks

The I/O Bank Assigner (IOBA) tool runs automatically when you run Layout. You can also use this tool from within the MultiView Navigator (Figure 13-17). The IOBA tool automatically assigns technologies and VREF pins (if required) to every I/O bank that does not currently have any technologies assigned to it. This tool is available when at least one I/O bank is unassigned.

To automatically assign technologies to I/O banks, choose I/O Bank Assigner from the **Tools** menu (or click the I/O Bank Assigner's toolbar button, shown in Figure 13-16).

***Figure 13-16* • I/O Bank Assigner's Toolbar Button**

Messages will appear in the Output window informing you when the automatic I/O bank assignment begins and ends. If the assignment is successful, the message "I/O Bank Assigner completed successfully" appears in the Output window, as shown in Figure 13-17.

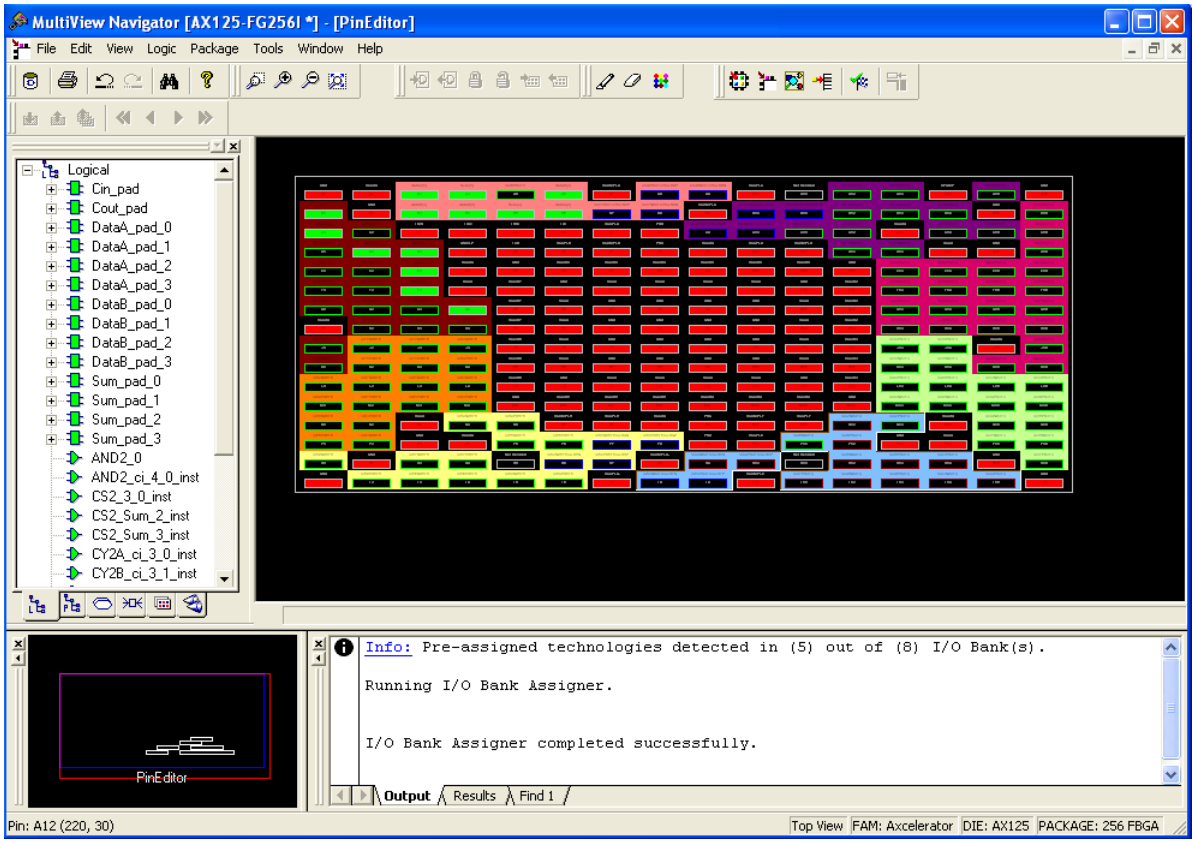


***Figure 13-17* • I/O Bank Assigner Displays Messages in Output Window**

If the assignment is not successful, an error message appears in the Output window.

To undo the I/O bank assignments, choose **Undo** from the **Edit** menu. Undo removes the I/O technologies assigned by the IOBA. It does not remove the I/O technologies previously assigned.

To redo the changes undone by the Undo command, choose **Redo** from the **Edit** menu.

To clear I/O bank assignments made before using the Undo command, manually unassign or reassign I/O technologies to banks. To do so, choose **I/O Bank Settings** from the **Edit** menu to display the I/O Bank Settings dialog box.

Conclusion

Fusion, IGLOO, and ProASIC3 support for multiple I/O standards minimizes board-level components and makes possible a wide variety of applications. The Microsemi Designer software, integrated with Libero SoC, presents a clear visual display of I/O assignments, allowing users to verify I/O and board-level design requirements before programming the device. The device I/O features and functionalities ensure board designers can produce low-cost and low power FPGA applications fulfilling the complexities of contemporary design needs.

Related Documents

User's Guides

Libero SoC User's Guide

http://www.microsemi.com/soc/documents/libero_ug.pdf

IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide

http://www.microsemi.com/soc/documents/pa3_libguide_ug.pdf

SmartGen Core Reference Guide

http://www.microsemi.com/soc/documents/genguide_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the document.

Date	Changes	Page
August 2012	The notes in Table 13-2 • Designer State (resulting from I/O attribute modification) were revised to clarify which device families support programmable input delay (SAR 39666).	313
June 2011	Figure 13-2 • SmartGen Catalog was updated (SAR 24310). Figure 8-3 • Expanded I/O Section and the step associated with it were deleted to reflect changes in the software.	314
	The following rule was added to the "VREF Rules for the Implementation of Voltage-Referenced I/O Standards" section: Only minibanks that contain input or bidirectional I/Os require a VREF. A VREF is not needed for minibanks composed of output or tristated I/Os (SAR 24310).	325
July 2010	Notes were added where appropriate to point out that IGLOO nano and ProASIC3 nano devices do not support differential inputs (SAR 21449).	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 13-1 • Flash-Based FPGAs.	312
	The notes for Table 13-2 • Designer State (resulting from I/O attribute modification) were revised to indicate that skew control and input delay do not apply to nano devices.	313
v1.3 (October 2008)	The "Flash FPGAs I/O Support" section was revised to include new families and make the information more concise.	312
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 13-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	312
v1.1 (March 2008)	This document was previously part of the *I/O Structures in IGLOO and ProASIC3 Devices* document. The content was separated and made into a new document.	N/A
	Table 13-2 • Designer State (resulting from I/O attribute modification) was updated to include note 2 for IGLOO PLUS.	313

14 – DDR for Microsemi's Low Power Flash Devices

Introduction

The I/Os in Fusion, IGLOO, and ProASIC3 devices support Double Data Rate (DDR) mode. In this mode, new data is present on every transition (or clock edge) of the clock signal. This mode doubles the data transfer rate compared with Single Data Rate (SDR) mode, where new data is present on one transition (or clock edge) of the clock signal. Low power flash devices have DDR circuitry built into the I/O tiles. I/Os are configured to be DDR receivers or transmitters by instantiating the appropriate special macros (examples shown in Figure 14-4 on page 336 and Figure 14-5 on page 337) and buffers (DDR_OUT or DDR_REG) in the RTL design. This document discusses the options the user can choose to configure the I/Os in this mode and how to instantiate them in the design.

Double Data Rate (DDR) Architecture

Low power flash devices support 350 MHz DDR inputs and outputs. In DDR mode, new data is present on every transition of the clock signal. Clock and data lines have identical bandwidths and signal integrity requirements, making them very efficient for implementing very high-speed systems. High-speed DDR interfaces can be implemented using LVDS (not applicable for IGLOO nano and ProASIC3 nano devices). In IGLOOe, ProASIC3E, AFS600, and AFS1500 devices, DDR interfaces can also be implemented using the HSTL, SSTL, and LVPECL I/O standards. The DDR feature is primarily implemented in the FPGA core periphery and is not tied to a specific I/O technology or limited to any I/O standard.

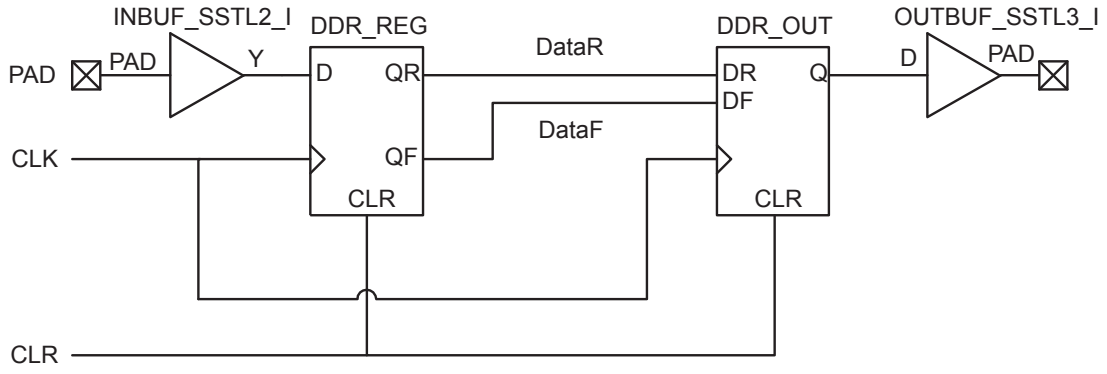


***Figure 14-1* • DDR Support in Low Power Flash Devices**

DDR Support in Flash-Based Devices

The flash FPGAs listed in Table 14-1 support the DDR feature and the functions described in this document.

***Table 14-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 14-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 14-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

I/O Cell Architecture

Low power flash devices support DDR in the I/O cells in four different modes: Input, Output, Tristate, and Bidirectional pins. For each mode, different I/O standards are supported, with most I/O standards having special sub-options. For the ProASIC3 nano and IGLOO nano devices, DDR is supported only in the 60 k, 125 k, and 250 k logic densities. Refer to Table 14-2 for a sample of the available I/O options. Additional I/O options can be found in the relevant family datasheet.

***Table 14-2* • DDR I/O Options**

DDR Register Type	I/O Type	I/O Standard	Sub-Options	Comments
Receive Register	Input	Normal	None	3.3 V TTL (default)
		LVCMOS	Voltage	1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)
			Pull-Up	None (default)
		PCI/PCI-X	None	
		GTL/GTL+	Voltage	2.5 V, 3.3 V (3.3 V default)
		HSTL	Class	I / II (I default)
		SSTL2/SSTL3	Class	I / II (I default)
		LVPECL	None	
		LVDS	None	
Transmit Register	Output	Normal	None	3.3 V TTL (default)
		LVTTL	Output Drive	2, 4, 6, 8, 12, 16, 24, 36 mA (8 mA default)
			Slew Rate	Low/high (high default)
		LVCMOS	Voltage	1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)
		PCI/PCI-X	None	
		GTL/GTL+	Voltage	1.8 V, 2.5 V, 3.3 V (3.3 V default)
		HSTL	Class	I / II (I default)
		SSTL2/SSTL3	Class	I / II (I default)
		LVPECL*	None	
		LVDS*	None	

*Note: *IGLOO nano and ProASIC3 nano devices do not support differential inputs.*

***Table 14-2* • DDR I/O Options (continued)**

DDR Register Type	I/O Type	I/O Standard	Sub-Options	Comments
Transmit Register (continued)	Tristate Buffer	Normal	Enable Polarity	Low/high (low default)
		LVTTL	Output Drive	2, 4, 6, 8, 12,16, 24, 36 mA (8 mA default)
			Slew Rate	Low/high (high default)
			Enable Polarity	Low/high (low default)
			Pull-Up/-Down	None (default)
		LVCMOS	Voltage	1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)
			Output Drive	2, 4, 6, 8, 12, 16, 24, 36 mA (8 mA default)
			Slew Rate	Low/high (high default)
			Enable Polarity	Low/high (low default)
			Pull-Up/-Down	None (default)
		PCI/PCI-X	Enable Polarity	Low/high (low default)
		GTL/GTL+	Voltage	1.8 V, 2.5 V, 3.3 V (3.3 V default)
			Enable Polarity	Low/high (low default)
		HSTL	Class	I / II (I default)
			Enable Polarity	Low/high (low default)
		SSTL2/SSTL3	Class	I / II (I default)
			Enable Polarity	Low/high (low default)
	Bidirectional Buffer	Normal	Enable Polarity	Low/high (low default)
		LVTTL	Output Drive	2, 4, 6, 8, 12, 16, 24, 36 mA (8 mA default)
			Slew Rate	Low/high (high default)
			Enable Polarity	Low/high (low default)
			Pull-Up/-Down	None (default)
		LVCMOS	Voltage	1.5 V, 1.8 V, 2.5 V, 5 V (1.5 V default)
			Enable Polarity	Low/high (low default)
			Pull-Up	None (default)
		PCI/PCI-X	None	
			Enable Polarity	Low/high (low default)
		GTL/GTL+	Voltage	1.8 V, 2.5 V, 3.3 V (3.3 V default)
			Enable Polarity	Low/high (low default)
		HSTL	Class	I / II (I default)
			Enable Polarity	Low/high (low default)
		SSTL2/SSTL3	Class	I / II (I default)
			Enable Polarity	Low/high (low default)

*Note: *IGLOO nano and ProASIC3 nano devices do not support differential inputs.*

Input Support for DDR

The basic structure to support a DDR input is shown in Figure 14-2. Three input registers are used to capture incoming data, which is presented to the core on each rising edge of the I/O register clock. Each I/O tile supports DDR inputs.

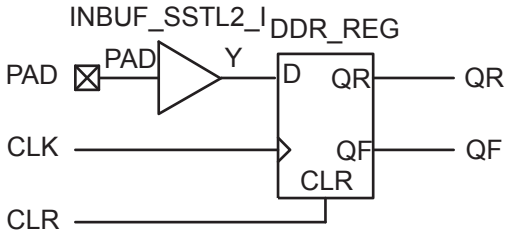


***Figure 14-2* • DDR Input Register Support in Low Power Flash Devices**

Output Support for DDR

The basic DDR output structure is shown in Figure 14-1 on page 331. New data is presented to the output every half clock cycle.

Note: DDR macros and I/O registers do not require additional routing. The combiner automatically recognizes the DDR macro and pushes its registers to the I/O register area at the edge of the chip. The routing delay from the I/O registers to the I/O buffers is already taken into account in the DDR macro.

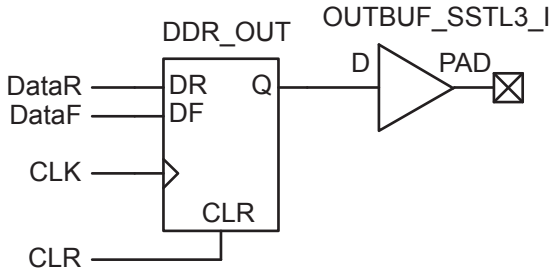


***Figure 14-3* • DDR Output Register (SSTL3 Class I)**

Instantiating DDR Registers

Using SmartGen is the simplest way to generate the appropriate RTL files for use in the design. Figure 14-4 shows an example of using SmartGen to generate a DDR SSTL2 Class I input register. SmartGen provides the capability to generate all of the DDR I/O cells as described. The user, through the graphical user interface, can select from among the many supported I/O standards. The output formats supported are Verilog, VHDL, and EDIF.

Figure 14-5 on page 337 through Figure 14-8 on page 340 show the I/O cell configured for DDR using SSTL2 Class I technology. For each I/O standard, the I/O pad is buffered by a special primitive that indicates the I/O standard type.

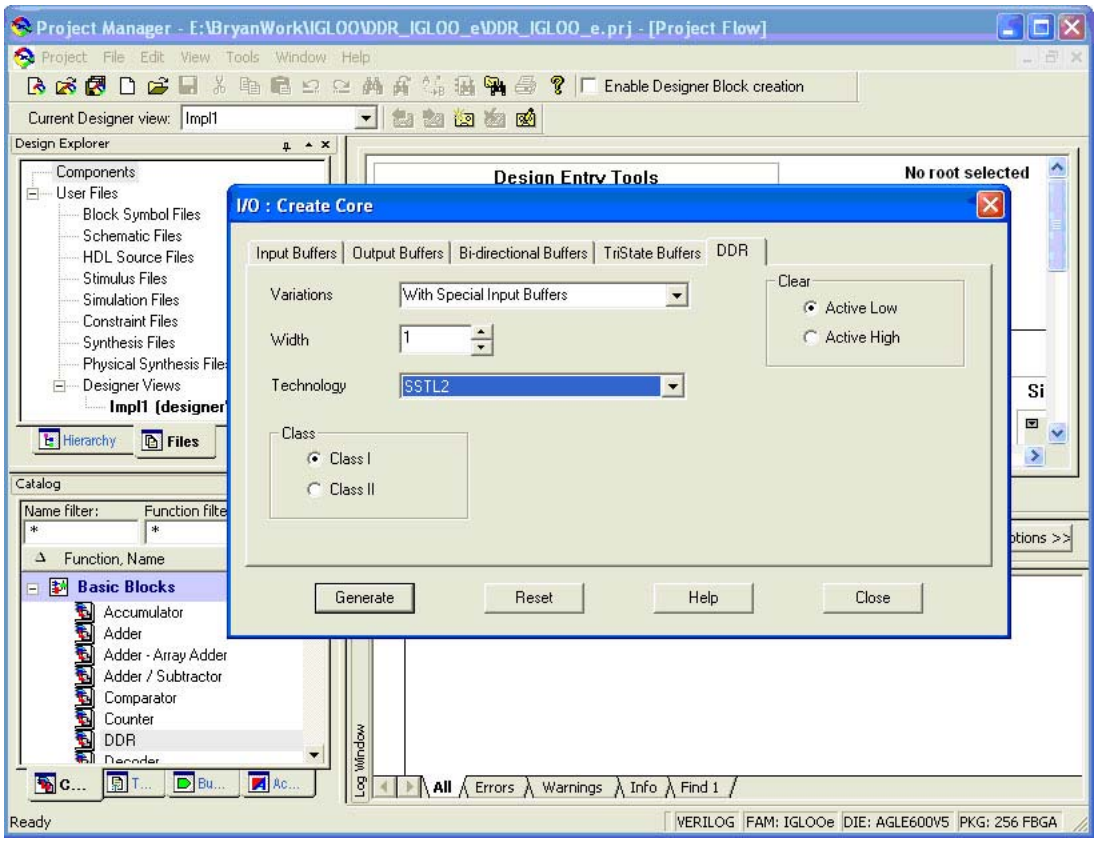


***Figure 14-4* • Example of Using SmartGen to Generate a DDR SSTL2 Class I Input Register**

DDR Input Register

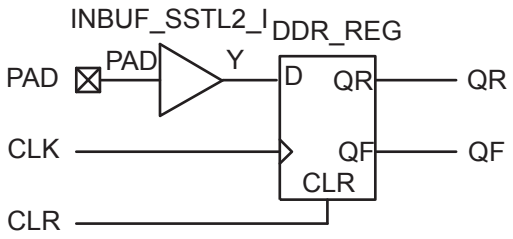


***Figure 14-5* • DDR Input Register (SSTL2 Class I)**

The corresponding structural representations, as generated by SmartGen, are shown below:

Verilog

```
module DDR_InBuf_SSTL2_I(PAD,CLR,CLK,QR,QF);

input   PAD, CLR, CLK;
output  QR, QF;

wire Y;

  INBUF_SSTL2_I INBUF_SSTL2_I_0_inst(.PAD(PAD),.Y(Y));
  DDR_REG DDR_REG_0_inst(.D(Y),.CLK(CLK),.CLR(CLR),.QR(QR),.QF(QF));

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
--The correct library will be inserted automatically by SmartGen
library proasic3; use proasic3.all;
--library fusion; use fusion.all;
--library igloo; use igloo.all;

entity DDR_InBuf_SSTL2_I is
  port(PAD, CLR, CLK : in std_logic;  QR, QF : out std_logic) ;
end DDR_InBuf_SSTL2_I;

architecture DEF_ARCH of  DDR_InBuf_SSTL2_I is

  component INBUF_SSTL2_I
    port(PAD : in std_logic := 'U'; Y : out std_logic) ;
  end component;

  component DDR_REG
    port(D, CLK, CLR : in std_logic := 'U'; QR, QF : out std_logic) ;
  end component;

signal Y : std_logic ;

begin

  INBUF_SSTL2_I_0_inst : INBUF_SSTL2_I
  port map(PAD => PAD, Y => Y);
  DDR_REG_0_inst : DDR_REG
  port map(D => Y, CLK => CLK, CLR => CLR, QR => QR, QF => QF);

end DEF_ARCH;
```

DDR Output Register

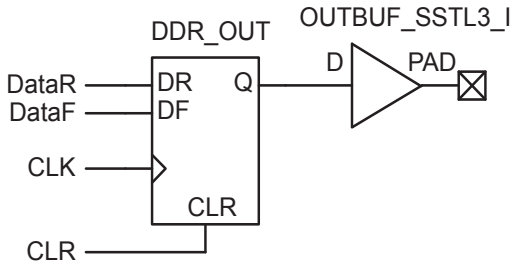


***Figure 14-6* • DDR Output Register (SSTL3 Class I)**

Verilog

```
module DDR_OutBuf_SSTL3_I(DataR,DataF,CLR,CLK,PAD);

input   DataR, DataF, CLR, CLK;
output  PAD;

wire Q, VCC;

  VCC VCC_1_net(.Y(VCC));
  DDR_OUT DDR_OUT_0_inst(.DR(DataR),.DF(DataF),.CLK(CLK),.CLR(CLR),.Q(Q));
  OUTBUF_SSTL3_I OUTBUF_SSTL3_I_0_inst(.D(Q),.PAD(PAD));

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
library proasic3; use proasic3.all;

entity DDR_OutBuf_SSTL3_I is
  port(DataR, DataF, CLR, CLK : in std_logic;  PAD : out std_logic) ;
end DDR_OutBuf_SSTL3_I;

architecture DEF_ARCH of  DDR_OutBuf_SSTL3_I is

  component DDR_OUT
    port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
  end component;

  component OUTBUF_SSTL3_I
    port(D : in std_logic := 'U'; PAD : out std_logic) ;
  end component;

  component VCC
    port( Y : out std_logic);
  end component;

signal Q, VCC_1_net : std_logic ;

begin

  VCC_2_net : VCC port map(Y => VCC_1_net);
  DDR_OUT_0_inst : DDR_OUT
  port map(DR => DataR, DF => DataF, CLK => CLK, CLR => CLR, Q => Q);
  OUTBUF_SSTL3_I_0_inst : OUTBUF_SSTL3_I
  port map(D => Q, PAD => PAD);

end DEF_ARCH;
```

DDR Tristate Output Register

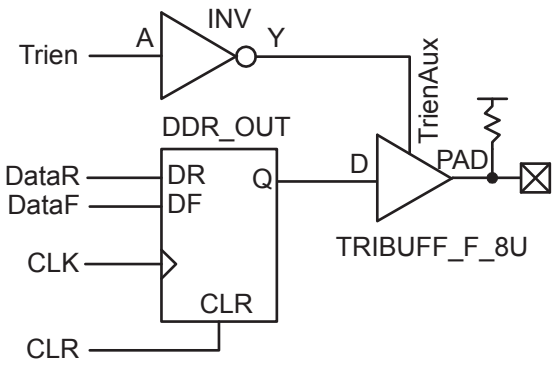


Figure 14-7 • **DDR Tristate Output Register, LOW Enable, 8 mA, Pull-Up (LVTTL)**

Verilog

```
module DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp(DataR, DataF, CLR, CLK, Trien,
  PAD);

input   DataR, DataF, CLR, CLK, Trien;
output  PAD;

wire TrienAux, Q;

  INV Inv_Tri(.A(Trien),.Y(TrienAux));
  DDR_OUT DDR_OUT_0_inst(.DR(DataR),.DF(DataF),.CLK(CLK),.CLR(CLR),.Q(Q));
  TRIBUFF_F_8U TRIBUFF_F_8U_0_inst(.D(Q),.E(TrienAux),.PAD(PAD));

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
library proasic3; use proasic3.all;

entity DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp is
  port(DataR, DataF, CLR, CLK, Trien : in std_logic;  PAD : out std_logic) ;
end DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp;

architecture DEF_ARCH of DDR_TriStateBuf_LVTTL_8mA_HighSlew_LowEnb_PullUp is

  component INV
    port(A : in std_logic := 'U'; Y : out std_logic) ;
  end component;

  component DDR_OUT
    port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
  end component;

  component TRIBUFF_F_8U
    port(D, E : in std_logic := 'U'; PAD : out std_logic) ;
  end component;

signal TrienAux, Q : std_logic ;

begin

  Inv_Tri : INV
  port map(A => Trien, Y => TrienAux);
```

```
  DDR_OUT_0_inst : DDR_OUT
  port map(DR => DataR, DF => DataF, CLK => CLK, CLR => CLR, Q => Q);
  TRIBUFF_F_8U_0_inst : TRIBUFF_F_8U
  port map(D => Q, E => TrienAux, PAD => PAD);

end DEF_ARCH;
```

DDR Bidirectional Buffer

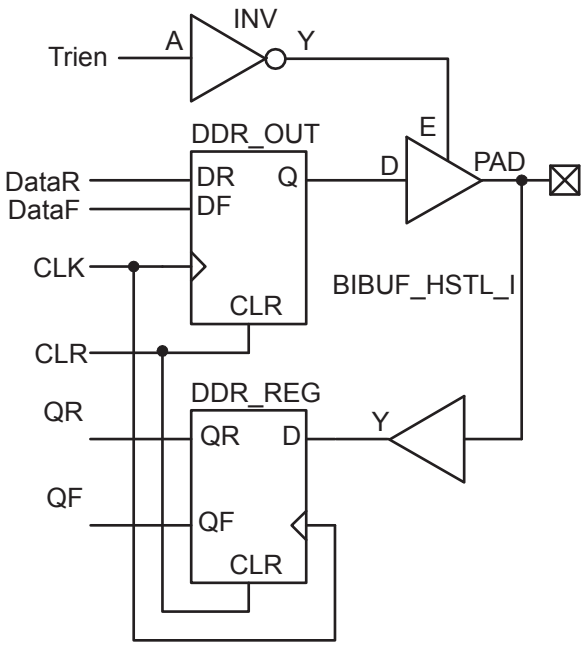


***Figure 14-8* • DDR Bidirectional Buffer, LOW Output Enable (HSTL Class II)**

Verilog

```
module DDR_BiDir_HSTL_I_LowEnb(DataR,DataF,CLR,CLK,Trien,QR,QF,PAD);

input   DataR, DataF, CLR, CLK, Trien;
output  QR, QF;
inout   PAD;

wire TrienAux, D, Q;

  INV Inv_Tri(.A(Trien), .Y(TrienAux));
  DDR_OUT DDR_OUT_0_inst(.DR(DataR),.DF(DataF),.CLK(CLK),.CLR(CLR),.Q(Q));
  DDR_REG DDR_REG_0_inst(.D(D),.CLK(CLK),.CLR(CLR),.QR(QR),.QF(QF));
  BIBUF_HSTL_I BIBUF_HSTL_I_0_inst(.PAD(PAD),.D(Q),.E(TrienAux),.Y(D));

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
library proasic3; use proasic3.all;

entity DDR_BiDir_HSTL_I_LowEnb is
  port(DataR, DataF, CLR, CLK, Trien : in std_logic; QR, QF : out std_logic;
    PAD : inout std_logic) ;
end DDR_BiDir_HSTL_I_LowEnb;

architecture DEF_ARCH of  DDR_BiDir_HSTL_I_LowEnb is

  component INV
    port(A : in std_logic := 'U'; Y : out std_logic) ;
  end component;

  component DDR_OUT
    port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
  end component;

  component DDR_REG
    port(D, CLK, CLR : in std_logic := 'U'; QR, QF : out std_logic) ;
  end component;

  component BIBUF_HSTL_I
    port(PAD : inout std_logic := 'U'; D, E : in std_logic := 'U'; Y : out std_logic) ;
  end component;

signal TrienAux, D, Q : std_logic ;

begin

  Inv_Tri : INV
  port map(A => Trien, Y => TrienAux);
  DDR_OUT_0_inst : DDR_OUT
  port map(DR => DataR, DF => DataF, CLK => CLK, CLR => CLR, Q => Q);
  DDR_REG_0_inst : DDR_REG
  port map(D => D, CLK => CLK, CLR => CLR, QR => QR, QF => QF);
  BIBUF_HSTL_I_0_inst : BIBUF_HSTL_I
  port map(PAD => PAD, D => Q, E => TrienAux, Y => D);

end DEF_ARCH;
```

Design Example

Figure 14-9 shows a simple example of a design using both DDR input and DDR output registers. The user can copy the HDL code in Libero SoC software and go through the design flow. Figure 14-10 and Figure 14-11 on page 343 show the netlist and ChipPlanner views of the ddr_test design. Diagrams may vary slightly for different families.

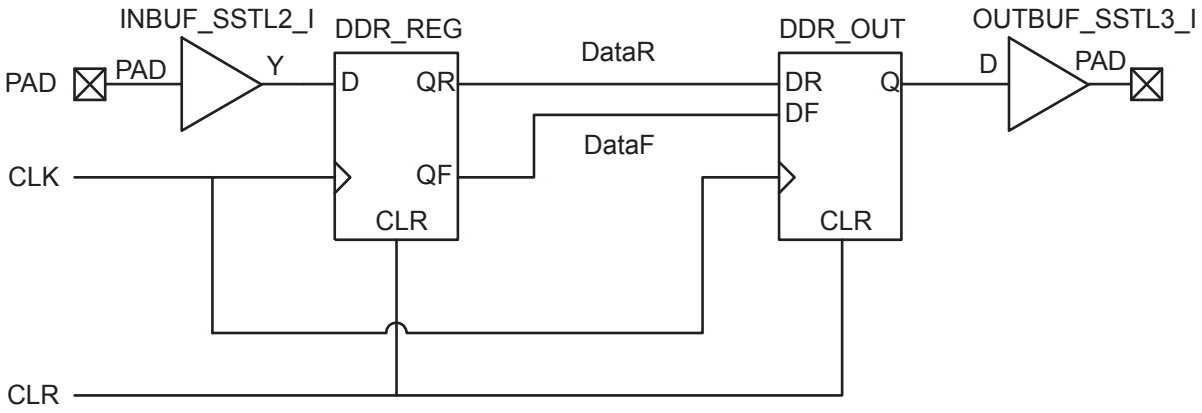


***Figure 14-9* • Design Example**

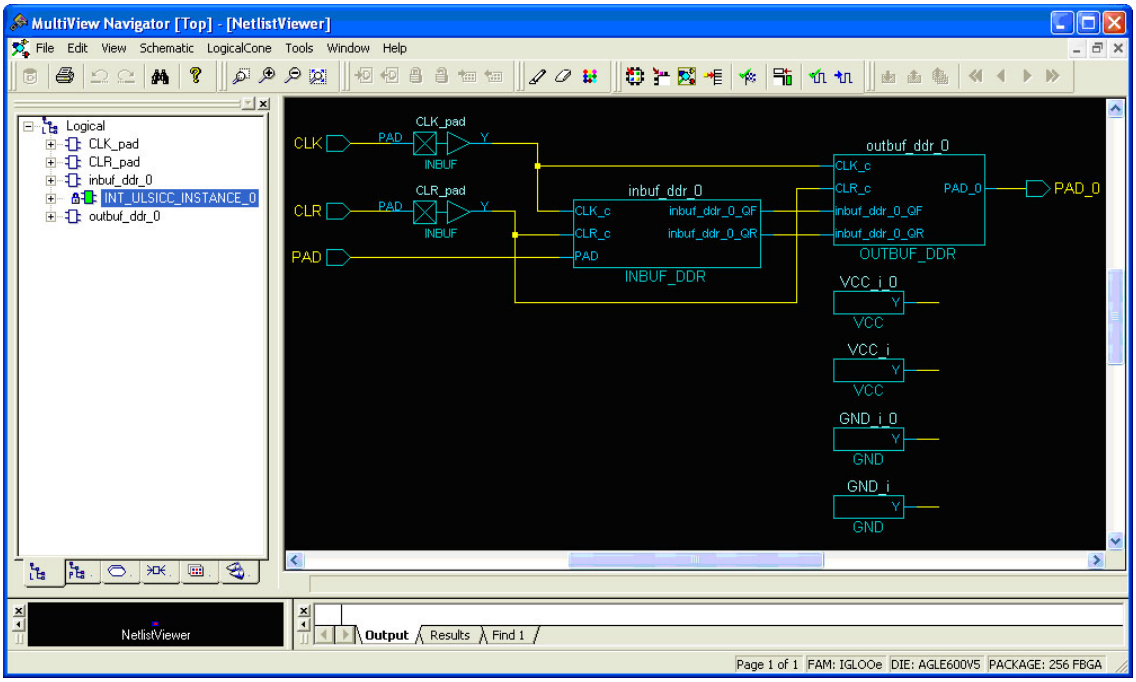


***Figure 14-10* • DDR Test Design as Seen by NetlistViewer for IGLOO/e Devices**

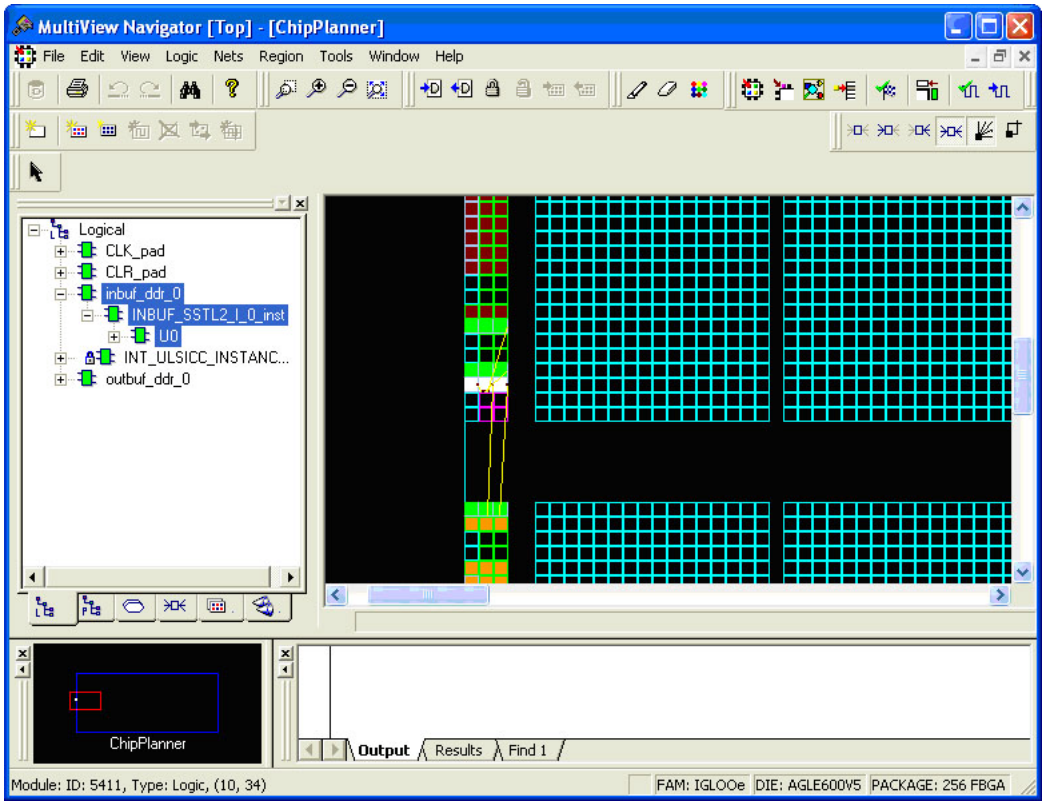


Figure 14-11 • **DDR Input/Output Cells as Seen by ChipPlanner for IGLOO/e Devices**

Verilog

```
module Inbuf_ddr(PAD,CLR,CLK,QR,QF);

input PAD, CLR, CLK;
output  QR, QF;

wire Y;

  DDR_REG DDR_REG_0_inst(.D(Y), .CLK(CLK), .CLR(CLR), .QR(QR), .QF(QF));
  INBUF INBUF_0_inst(.PAD(PAD), .Y(Y));

endmodule

module Outbuf_ddr(DataR,DataF,CLR,CLK,PAD);

input DataR, DataF, CLR, CLK;
output  PAD;

wire Q, VCC;

  VCC VCC_1_net(.Y(VCC));
  DDR_OUT DDR_OUT_0_inst(.DR(DataR), .DF(DataF), .CLK(CLK), .CLR(CLR), .Q(Q));
  OUTBUF OUTBUF_0_inst(.D(Q), .PAD(PAD));

endmodule
```

```
module ddr_test(DIN, CLK, CLR, DOUT);

input  DIN, CLK, CLR;
output DOUT;

  Inbuf_ddr Inbuf_ddr (.PAD(DIN), .CLR(clr), .CLK(clk), .QR(qr), .QF(qf));
  Outbuf_ddr Outbuf_ddr (.DataR(qr),.DataF(qf), .CLR(clr), .CLK(clk),.PAD(DOUT));

  INBUF INBUF_CLR (.PAD(CLR), .Y(clr));
  INBUF INBUF_CLK (.PAD(CLK), .Y(clk));

endmodule
```

Simulation Consideration

Microsemi DDR simulation models use inertial delay modeling by default (versus transport delay modeling). As such, pulses that are shorter than the actual gate delays should be avoided, as they will not be seen by the simulator and may be an issue in post-routed simulations. The user must be aware of the default delay modeling and must set the correct delay model in the simulator as needed.

Conclusion

Fusion, IGLOO, and ProASIC3 devices support a wide range of DDR applications with different I/O standards and include built-in DDR macros. The powerful capabilities provided by SmartGen and its GUI can simplify the process of including DDR macros in designs and minimize design errors. Additional considerations should be taken into account by the designer in design floorplanning and placement of I/O flip-flops to minimize datapath skew and to help improve system timing margins. Other system-related issues to consider include PLL and clock partitioning.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
	Notes were added where appropriate to point out that IGLOO nano and ProASIC3 nano devices do not support differential inputs (SAR 21449).	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 14-1 • Flash-Based FPGAs.	332
	The "I/O Cell Architecture" section was updated with information applicable to nano devices.	333
	The output buffer (OUTBUF_SSTL3_I) input was changed to D, instead of Q, in Figure 14-1 • DDR Support in Low Power Flash Devices, Figure 14-3 • DDR Output Register (SSTL3 Class I), Figure 14-6 • DDR Output Register (SSTL3 Class I), Figure 14-7 • DDR Tristate Output Register, LOW Enable, 8 mA, Pull-Up (LVTTL), and the output from the DDR_OUT macro was connected to the input of the TRIBUFF macro in Figure 14-7 • DDR Tristate Output Register, LOW Enable, 8 mA, Pull-Up (LVTTL).	331, 335, 338, 339
v1.3 (October 2008)	The "Double Data Rate (DDR) Architecture" section was updated to include mention of the AFS600 and AFS1500 devices.	331
	The "DDR Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	332
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 14-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	332
v1.1 (March 2008)	The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	332

15 – Prototyping With AFS600 for Smaller Devices

This document is designed as an aid for customers who may ultimately wish to use one of the smaller members of the Fusion family (AFS090 and/or AFS250). The first device available in the Fusion family is the AFS600, which can be used as a development or prototyping platform for the smaller devices. This document will help highlight differences between the AFS600 and these smaller devices in order to ease transition to the targeted device when it becomes available.

The Fusion family, based on the highly successful ProASIC®3E and ProASIC3 Flash FPGA architecture, has been designed as a high-performance, programmable, mixed signal platform. By combining an advanced flash FPGA core with embedded flash memory and analog peripherals, Fusion devices dramatically simplify system design, and save overall system cost and board space as a result. Figure 15-1 shows the Fusion device architecture overview.

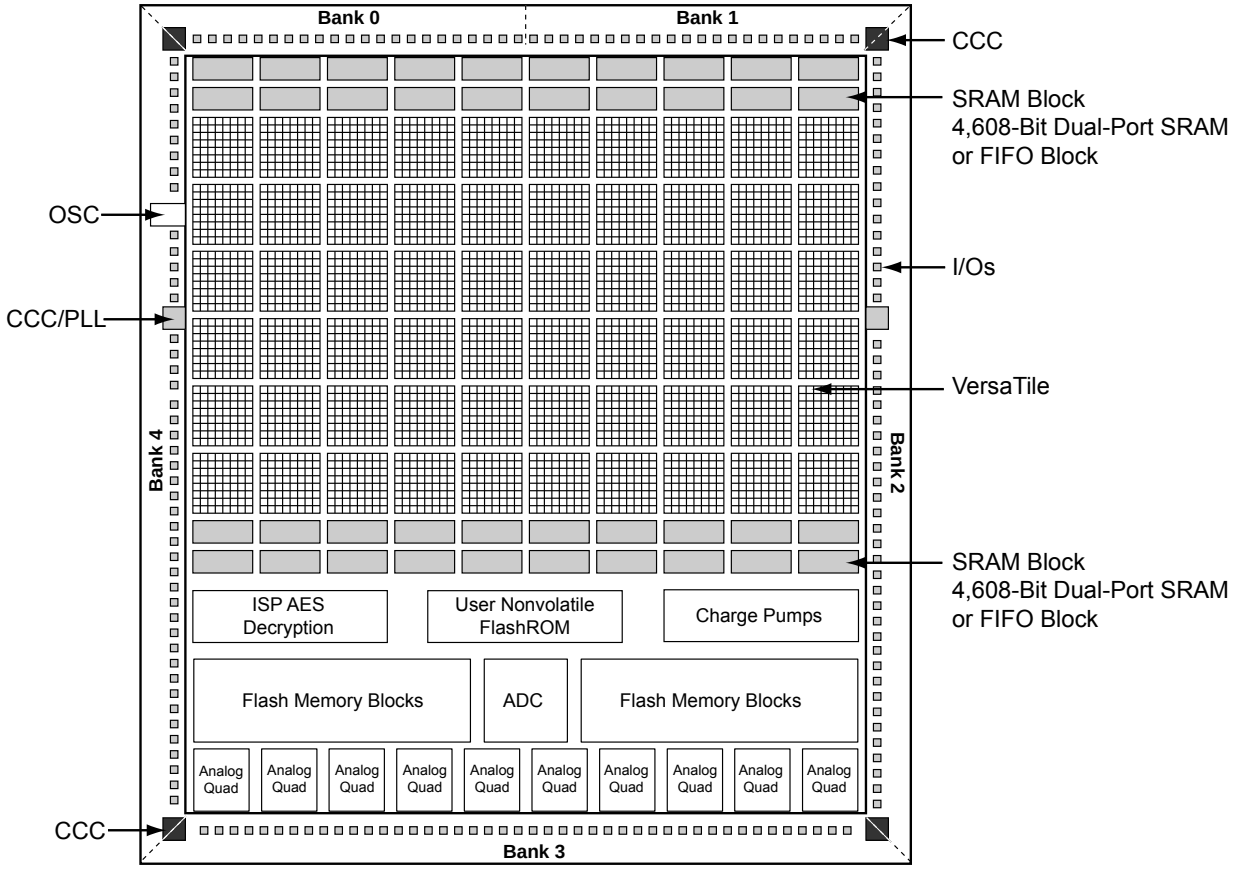


Figure 15-1 • **Fusion Device Architecture Overview**

The state-of-the-art embedded flash memory technology offers high-density integrated flash memory arrays, enabling savings in cost, power, and board area relative to external flash solutions, while providing increased flexibility and performance.

Fusion devices offer a robust and flexible analog mixed signal addition to the high-performance flash FPGA fabric and embedded flash memory. The many built-in analog peripherals include a configurable 32:1 input analog multiplexer, up to 10 independent metal-oxide semiconductor field-effect transistor (MOSFET) gate driver outputs, and a configurable analog-to-digital converter (ADC). The Analog Quad is an I/O structure that contains three adjacent analog inputs and a gate driver output.

The addition of the real-time counter (RTC) system enables Fusion devices to support both standby and sleep modes of operation, greatly reducing power consumption in many applications.

Prototype Guideline

AFS090, AFS250, AFS600, and AFS1500 Device Configuration

AFS600 is a medium size device in the Fusion family. It supports all of the Fusion features, as shown in Table 15-1. The smaller devices (AFS090 and AFS250) in the Fusion family have lower gate counts and fewer memory blocks, I/Os, and PLLs.

***Table 15-1* • AFS090, AFS250, and AFS600 Device Summary**

Part Number		AFS090	AFS250	AFS600	AFS1500
General Information	System Gates	90,000	250,000	600,000	1,500,000
	Tiles (D-Flip-Flop)	2,304	6,144	13,824	38,400
	Secure (AES) ISP	Yes	Yes	Yes	Yes
	PLLs	1	1	2	2
	Globals	18	18	18	18
Memory	Flash Memory Blocks (256 kbytes)	1	1	2	4
	Flash Memory (kbytes)	256	256	512	1,024
	FlashROM Bits	1 k	1 k	1 k	1 k
	RAM Blocks (4,608 bits)	6	8	24	60
	RAM kbits	27	36	108	270
Analog	Analog Quads	5	6	10	10
	Analog Input Channels	15	18	30	30
	Gate Driver Outputs	5	6	10	10
I/O	I/O Types	Analog/ LVDS / Std+	Analog/ LVDS / Std+	Analog/ LVDS / Pro	Analog/ LVDS / Pro
	I/O Banks (+ JTAG)	4	4	5	5
	Maximum Digital I/Os	73	114	172	278
	Analog I/Os	20	24	40	40
I/O: Digital/Analog	QN108	36/14		–	–
	QN180	48/20	62/24	–	–
	PQ208	–	93/24	95/40	–
	FG256	73/20	114/24	119/40	119/40
	FG484	–	–	172/40	228/40
	FG676	–	–	–	278/40

Table 15-2 shows compatible devices for each package. The FG256 package is designed to support migration across all family members.

***Table 15-2* • Package Compatibility Table**

Package Types	PQ208	PQ208	FG256	FG484	FG676	QN108	QN180
Compatible Devices	AFS90	AFS600	AFS090	AFS600	AFS1500	AFS90	AFS090
	AFS250	AFS1500	AFS250	AFS1500			AFS250
			AFS600				
			AFS1500				

List of the Guidelines for Prototyping

Microsemi recommends AFS600-FG256 as the platform for prototyping smaller devices within the same compatible package type. AFS600-FG256 is also the first available Fusion silicon in the rollout roadmap and is used in the Fusion Starter Kit, which can serve as a prototype board to demonstrate the majority of Fusion features.

Memory Blocks

The AFS250 and AFS090 have a single 256-kbyte block of embedded flash memory, whereas the AFS600 has two 256-kbyte blocks (512 kbytes total). Therefore, the user must keep the usage less than 256 kbytes while doing prototyping with the AFS600.

The AFS250 has 8 RAM blocks, while the AFS600 has 24 RAM blocks. A SmartGen analog system generated soft IP uses 3 to 9 blocks of RAM. The user needs to keep track of the RAM block usage, especially if the design contains a RAM initialization application, data storage application or other applications that utilize extra RAM blocks. Usage must be no more than 8 blocks. Likewise, if the user is prototyping for AFS090, then the RAM block usage in AFS600 should not exceed 6 blocks.

PLLs

The AFS250 and AFS090 have one PLL on the west side of the device, whereas AFS600 has two PLLs—one for each side of the device. During prototyping using AFS600, the user should only implement the PLL on the west side and use the corresponding PLL input pin, so that the delays from the PLL input through the PLL to the global network have a minimum variation between AFS090/AFS250 and AFS600.

I/Os

All special function I/Os (V_{CC}, GND, JTAG, Programming Control, etc.) of the AFS250 and AFS090 devices are in exactly the same locations as in the AFS600 device, with one exception for the AFS090 as listed below. The AFS250 device has 6 Analog Quads, whereas the AFS600 device has 10 Analog Quads. The user should only use Analog Quads 0–5 while doing prototyping in AFS600, in order to have exactly the same analog pin map. To prototype AFS090, the user should only use Analog Quads 0–4, since AFS090 has 5 Analog Quads.

While the AFS250 differential I/Os have the same locations as the AFS600, the AFS090 differential I/O locations are slightly different from those of the AFS600. More details on the pin list are available in the *Fusion Family of Mixed Signal FPGAs* datasheet. The user should be aware that if the design has differential I/Os, the pinout needs to be changed from the AFS600 prototype design to an AFS090 production design.

Prototype Consideration in Software

After validating the design in the AFS600, the user needs to create a new Libero® System-on-Chip (SoC) software project for the AFS090 or AFS250, then recreate the SmartGen cores by using the same parameters used for the AFS600. All other source code used in the AFS600 project can be directly imported into the AFS090 or AFS250 project. The same validation process (simulation, static timing analysis, and functional test on silicon) should be performed for the AFS090 or AFS250 design as the user has done for AFS600.

Summary

AFS600-FG256 is the recommended Fusion prototyping vehicle for smaller devices in the same compatible package. It is also used on the Fusion Starter Kit board, which can demonstrate most of the Fusion family features.

16 – Programming Flash Devices

Introduction

This document provides an overview of the various programming options available for the Microsemi flash families. The electronic version of this document includes active links to all programming resources, which are available at http://www.microsemi.com/soc/products/hardware/default.aspx. For Microsemi antifuse devices, refer to the *Programming Antifuse Devices* document.

Summary of Programming Support

FlashPro4 and FlashPro3 are high-performance in-system programming (ISP) tools targeted at the latest generation of low power flash devices offered by the SmartFusion,® Fusion, IGLOO,® and ProASIC®3 families, including ARM-enabled devices. FlashPro4 and FlashPro3 offer extremely high performance through the use of USB 2.0, are high-speed compliant for full use of the 480 Mbps bandwidth, and can program ProASIC3 devices in under 30 seconds. Powered exclusively via USB, FlashPro4 and FlashPro3 provide a VPUMP voltage of 3.3 V for programming these devices.

FlashPro4 replaced FlashPro3 in 2010. FlashPro4 supports SmartFusion, Fusion, ProASIC3,and IGLOO devices as well as future generation flash devices. FlashPro4 also adds 1.2 V programming for IGLOO nano V2 devices. FlashPro4 is compatible with FlashPro3; however it adds a programming mode (PROG_MODE) signal to the previously unused pin 4 of the JTAG connector. The PROG_MODE goes high during programming and can be used to turn on a 1.5 V external supply for those devices that require 1.5 V for programming. If both FlashPro3 and FlashPro4 programmers are used for programming the same boards, pin 4 of the JTAG connector must not be connected to anything on the board because FlashPro4 uses pin 4 for PROG_MODE.

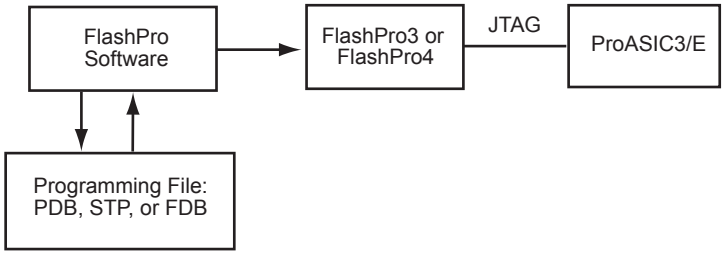


***Figure 16-1* • FlashPro Programming Setup**

Programming Support in Flash Devices

The flash FPGAs listed in Table 16-1 support flash in-system programming and the functions described in this document.

***Table 16-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution, supporting 1.2 V to 1.5 V core voltage with Flash*Freeze technology
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V core voltage with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
SmartFusion	SmartFusion	Mixed-signal FPGA integrating FPGA fabric, programmable microcontroller subsystem (MSS), including programmable analog and ARM® Cortex™-M3 hard processor and flash memory in a monolithic device
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device
ProASIC	ProASIC	First generation ProASIC devices
	ProASIC PLUS	Second generation ProASIC devices

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 16-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 16-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

General Flash Programming Information

Programming Basics

When choosing a programming solution, there are a number of options available. This section provides a brief overview of those options. The next sections provide more detail on those options as they apply to Microsemi FPGAs.

Reprogrammable or One-Time-Programmable (OTP)

Depending on the technology chosen, devices may be reprogrammable or one-time-programmable. As the name implies, a reprogrammable device can be programmed many times. Generally, the contents of such a device will be completely overwritten when it is reprogrammed. All Microsemi flash devices are reprogrammable.

An OTP device is programmable one time only. Once programmed, no more changes can be made to the contents. Microsemi flash devices provide the option of disabling the reprogrammability for security purposes. This combines the convenience of reprogrammability during design verification with the security of an OTP technology for highly sensitive designs.

Device Programmer or In-System Programming

There are two fundamental ways to program an FPGA: using a device programmer or, if the technology permits, using in-system programming. A device programmer is a piece of equipment in a lab or on the production floor that is used for programming FPGA devices. The devices are placed into a socket mounted in a programming adapter module, and the appropriate electrical interface is applied. The programmed device can then be placed on the board. A typical programmer, used during development, programs a single device at a time and is referred to as a single-site engineering programmer.

With ISP, the device is already mounted onto the system printed circuit board when programming occurs. Typically, ISD programming is performed via a JTAG interface on the FPGA. The JTAG pins can be controlled either by an on-board resource, such as a microprocessor, or by an off-board programmer through a header connection. Once mounted, it can be programmed repeatedly and erased. If the application requires it, the system can be designed to reprogram itself using a microprocessor, without the use of any external programmer.

If multiple devices need to be programmed with the same program, various multi-site programming hardware is available in order to program many devices in parallel. Microsemi In House Programming is also available for this purpose.

Programming Features for Microsemi Devices

Flash Devices

The flash devices supplied by Microsemi are reprogrammable by either a generic device programmer or ISP. Microsemi supports ISP using JTAG, which is supported by the FlashPro4 and FlashPro3, FlashPro Lite, Silicon Sculptor 3, and Silicon Sculptor II programmers.

Levels of ISP support vary depending on the device chosen:

- All SmartFusion, Fusion, IGLOO, and ProASIC3 devices support ISP.
- IGLOO, IGLOOe, IGLOO nano V5, and IGLOO PLUS devices can be programmed in-system when the device is using a 1.5 V supply voltage to the FPGA core.
- IGLOO nano V2 devices can be programmed at 1.2 V core voltage (when using FlashPro4 only) or 1.5 V. IGLOO nano V5 devices are programmed with a VCC core voltage of 1.5 V.

Types of Programming for Flash Devices

The number of devices to be programmed will influence the optimal programming methodology. Those available are listed below:

- In-system programming
  - Using a programmer
  - Using a microprocessor or microcontroller
- Device programmers
  - Single-site programmers
  - Multi-site programmers, batch programmers, or gang programmers
  - Automated production (robotic) programmers
- Volume programming services
  - Microsemi in-house programming
  - Programming centers

In-System Programming

Device Type Supported: Flash

ISP refers to programming the FPGA after it has been mounted on the system printed circuit board. The FPGA may be preprogrammed and later reprogrammed using ISP.

The advantage of using ISP is the ability to update the FPGA design many times without any changes to the board. This eliminates the requirement of using a socket for the FPGA, saving cost and improving reliability. It also reduces programming hardware expenses, as the ISP methodology is die-/package-independent.

There are two methods of in-system programming: external and internal.

- Programmer ISP—Refer to the "In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X" section on page 391 for more information.

  Using an external programmer and a cable, the device can be programmed through a header on the system board. In Microsemi SoC Products Group documentation, this is referred to as external ISP. Microsemi provides FlashPro4, FlashPro3, FlashPro Lite, or Silicon Sculptor 3 to perform external ISP. Note that Silicon Sculptor II and Silicon Sculptor 3 can only provide ISP for ProASIC and ProASIC^{PLUS}® families, not for SmartFusion, Fusion, IGLOO, or ProASIC3. Silicon Sculptor II and Silicon Sculptor 3 can be used for programming ProASIC and ProASIC^{PLUS} devices by using an adapter module (part number SMPA-ISP-ACTEL-3).
  - Advantages: Allows local control of programming and data files for maximum security. The programming algorithms and hardware are available from Microsemi. The only hardware required on the board is a programming header.
  - Limitations: A negligible board space requirement for the programming header and JTAG signal routing
- Microprocessor ISP—Refer to the "Microprocessor Programming of Microsemi's Low Power Flash Devices" chapter of an appropriate FPGA fabric user's guide for more information.

  Using a microprocessor and an external or internal memory, you can store the program in memory and use the microprocessor to perform the programming. In Microsemi documentation, this is referred to as internal ISP. Both the code for the programming algorithm and the FPGA programming file must be stored in memory on the board. Programming voltages must also be generated on the board.
  - Advantages: The programming code is stored in the system memory. An external programmer is not required during programming.
  - Limitations: This is the approach that requires the most design work, since some way of getting and/or storing the data is needed; a system interface to the device must be designed; and the low-level API to the programming firmware must be written and linked into the code provided by Microsemi. While there are benefits to this methodology, serious thought and planning should go into the decision.

Device Programmers

Single Device Programmer

Single device programmers are used to program a device before it is mounted on the system board.

The advantage of using device programmers is that no programming hardware is required on the system board. Therefore, no additional components or board space are required.

Adapter modules are purchased with single device programmers to support the FPGA packages used. The FPGA is placed in the adapter module and the programming software is run from a PC. Microsemi supplies the programming software for all of the Microsemi programmers. The software allows for the selection of the correct die/package and programming files. It will then program and verify the device.

- Single-site programmers

  A single-site programmer programs one device at a time. Microsemi offers Silicon Sculptor 3, built by BP Microsystems, as a single-site programmer. Silicon Sculptor 3 and associated software are available only from Microsemi.

  – Advantages: Lower cost than multi-site programmers. No additional overhead for programming on the system board. Allows local control of programming and data files for maximum security. Allows on-demand programming on-site.

  – Limitations: Only programs one device at a time.

- Multi-site programmers

Often referred to as batch or gang programmers, multi-site programmers can program multiple devices at the same time using the same programming file. This is often used for large volume programming and by programming houses. The sites often have independent processors and memory enabling the sites to operate concurrently, meaning each site may start programming the same file independently. This enables the operator to change one device while the other sites continue programming, which increases throughput. Multiple adapter modules for the same package are required when using a multi-site programmer. Silicon Sculptor I, II, and 3 programmers can be cascaded to program multiple devices in a chain. Multi-site programmers, such as the BP2610 and BP2710, can also be purchased from BP Microsystems. When using BP Microsystems multi-site programmers, users must use programming adapter modules available only from Microsemi. Visit the Microsemi SoC Products Group website to view the part numbers of the desired adapter module:

http://www.microsemi.com/soc/products/hardware/program_debug/ss/modules.aspx.

Also when using BP Microsystems programmers, customers must use Microsemi programming software to ensure the best programming result will occur.

– Advantages: Provides the capability of programming multiple devices at the same time. No additional overhead for programming on the system board. Allows local control of programming and data files for maximum security.

– Limitations: More expensive than a single-site programmer

- Automated production (robotic) programmers

Automated production programmers are based on multi-site programmers. They consist of a large input tray holding multiple parts and a robotic arm to select and place parts into appropriate programming sockets automatically. When the programming of the parts is complete, the parts are removed and placed in a finished tray. The automated programmers are often used in volume programming houses to program parts for which the programming time is small. BP Microsystems part number BP4710, BP4610, BP3710 MK2, and BP3610 are available for this purpose. Auto programmers cannot be used to program RTAX-S devices.

Where an auto-programmer is used, the appropriate open-top adapter module from BP Microsystems must be used.

Volume Programming Services

Device Type Supported: Flash and Antifuse

Once the design is stable for applications with large production volumes, preprogrammed devices can be purchased. Table 16-2 describes the volume programming services.

***Table 16-2* • Volume Programming Services**

Programmer	Vendor	Availability
In-House Programming	Microsemi	Contact Microsemi Sales
Distributor Programming Centers	Memec Unique	Contact Distribution
Independent Programming Centers	Various	Contact Vendor

Advantages: As programming is outsourced, this solution is easier to implement than creating a substantial in-house programming capability. As programming houses specialize in large-volume programming, this is often the most cost-effective solution.

Limitations: There are some logistical issues with the use of a programming service provider, such as the transfer of programming files and the approval of First Articles. By definition, the programming file must be released to a third-party programming house. Nondisclosure agreements (NDAs) can be signed to help ensure data protection; however, for extremely security-conscious designs, this may not be an option.

- Microsemi In-House Programming

  When purchasing Microsemi devices in volume, IHP can be requested as part of the purchase. If this option is chosen, there is a small cost adder for each device programmed. Each device is marked with a special mark to distinguish it from blank parts. Programming files for the design will be sent to Microsemi. Sample parts with the design programmed, First Articles, will be returned for customer approval. Once approval of First Articles has been received, Microsemi will proceed with programming the remainder of the order. To request Microsemi IHP, contact your local Microsemi representative.

- Distributor Programming Centers

  If purchases are made through a distributor, many distributors will provide programming for their customers. Consult with your preferred distributor about this option.

Programming Solutions

Details for the available programmers can be found in the programmer user's guides listed in the "Related Documents" section on page 361.

All the programmers except FlashPro4, FlashPro3, FlashPro Lite, and FlashPro require adapter modules, which are designed to support device packages. All modules are listed on the Microsemi SoC Products Group website at http://www.microsemi.com/soc/products/hardware/program_debug/ss/modules.aspx. They are not listed in this document, since this list is updated frequently with new package options and any upgrades required to improve programming yield or support new families.

Table 16-3 • **Programming Solutions**

Programmer	Vendor	ISP	Single Device	Multi-Device	Availability
FlashPro4	Microsemi	Only	Yes	Yes[1]	Available
FlashPro3	Microsemi	Only	Yes	Yes[1]	Available
FlashPro Lite[2]	Microsemi	Only	Yes	Yes[1]	Available
FlashPro	Microsemi	Only	Yes	Yes[1]	Discontinued
Silicon Sculptor 3	Microsemi	Yes[3]	Yes	Cascade option (up to two)	Available
Silicon Sculptor II	Microsemi	Yes[3]	Yes	Cascade option (up to two)	Available
Silicon Sculptor	Microsemi	Yes	Yes	Cascade option (up to four)	Discontinued
Sculptor 6X	Microsemi	No	Yes	Yes	Discontinued
BP MicroProgrammers	BP Microsystems	No	Yes	Yes	Contact BP Microsystems at www.bpmicro.com

Notes:

1. *Multiple devices can be connected in the same JTAG chain for programming.*
2. *If FlashPro Lite is used for programming, the programmer derives all of its power from the target pc board's VDD supply. The FlashPro Lite's VPP and VPN power supplies use the target pc board's VDD as a power source. The target pc board must supply power to both the VDDP and VDD power pins of the ProASIC^{PLUS} device in addition to supplying VDD to the FlashPro Lite. The target pc board needs to provide at least 500 mA of current to the FlashPro Lite VDD connection for programming.*
3. *Silicon Sculptor II and Silicon Sculptor 3 can only provide ISP for ProASIC and ProASIC^{PLUS} families, not for Fusion, IGLOO, or ProASIC3 devices.*

Programmer Ordering Codes

The products shown in Table 16-4 can be ordered through Microsemi sales and will be shipped directly from Microsemi. Products can also be ordered from Microsemi distributors, but will still be shipped directly from Microsemi. Table 16-4 includes ordering codes for the full kit, as well as codes for replacement items and any related hardware. Some additional products can be purchased from external suppliers for use with the programmers. Ordering codes for adapter modules used with Silicon Sculptor are available at http://www.microsemi.com/soc/products/hardware/program_debug/ss/modules.aspx.

Table 16-4 • **Programming Ordering Codes**

Description	Vendor	Ordering Code	Comment
FlashPro4 ISP programmer	Microsemi	FLASHPRO 4	Uses a 2×5, RA male header connector
FlashPro Lite ISP programmer	Microsemi	FLASHPRO LITE	Supports small programming header or large header through header converter (not included)
Silicon Sculptor 3	Microsemi	SILICON-SCULPTOR 3	USB 2.0 high-speed production programmer
Silicon Sculptor II	Microsemi	SILICON-SCULPTOR II	Requires add-on adapter modules to support devices
Silicon Sculptor ISP module	Microsemi	SMPA-ISP-ACTEL-3-KIT	Ships with both large and small header support
ISP cable for small header	Microsemi	ISP-CABLE-S	Supplied with SMPA-ISP-ACTEL-3-KIT
ISP cable for large header	Microsemi	PA-ISP-CABLE	Supplied with SMPA-ISP-ACTEL-3-KIT

Programmer Device Support

Refer to www.microsemi.com/soc for the current information on programmer and device support.

Certified Programming Solutions

The Microsemi-certified programmers for flash devices are FlashPro4, FlashPro3, FlashPro Lite, FlashPro, Silicon Sculptor II, Silicon Sculptor 3, and any programmer that is built by BP Microsystems. All other programmers are considered noncertified programmers.

- FlashPro4, FlashPro3, FlashPro Lite, FlashPro

  The Microsemi family of FlashPro device programmers provides in-system programming in an easy-to-use, compact system that supports all flash families. Whether programming a board containing a single device or multiple devices connected in a chain, the Microsemi line of FlashPro programmers enables fast programming and reprogramming. Programming with the FlashPro series of programmers saves board space and money as it eliminates the need for sockets on the board. There are no built-in algorithms, so there is no delay between product release and programming support. The FlashPro programmer is no longer available.

- Silicon Sculptor 3, Silicon Sculptor II

  Silicon Sculptor 3 and Silicon Sculptor II are robust, compact, single-device programmers with standalone software for the PC. They are designed to enable concurrent programming of multiple units from the same PC with speeds equivalent to or faster than previous Microsemi programmers.

- Noncertified Programmers

  Microsemi does not test programming solutions from other vendors, and DOES NOT guarantee programming yield. Also, Microsemi will not perform any failure analysis on devices programmed on non-certified programmers. Please refer to the *Programming and Functional Failure Guidelines* document for more information.

- Programming Centers

  Microsemi programming hardware policy also applies to programming centers. Microsemi expects all programming centers to use certified programmers to program Microsemi devices. If a programming center uses noncertified programmers to program Microsemi devices, the "Noncertified Programmers" policy applies.

Important Programming Guidelines

Preprogramming Setup

Before programming, several steps are required to ensure an optimal programming yield.

Use Proper Handling and Electrostatic Discharge (ESD) Precautions

Microsemi FPGAs are sensitive electronic devices that are susceptible to damage from ESD and other types of mishandling. For more information about ESD, refer to the *Quality and Reliability Guide,* beginning with page 41.

Use the Latest Version of the Designer Software to Generate Your Programming File (recommended)

The files used to program Microsemi flash devices (*.bit, *.stp, *.pdb) contain important information about the switches that will be programmed in the FPGA. Find the latest version and corresponding release notes at http://www.microsemi.com/soc/download/software/designer/. Also, programming files must always be zipped during file transfer to avoid the possibility of file corruption.

Use the Latest Version of the Programming Software

The programming software is frequently updated to accommodate yield enhancements in FPGA manufacturing. These updates ensure maximum programming yield and minimum programming times. Before programming, always check the version of software being used to ensure it is the most recent. Depending on the programming software, refer to one of the following:

- FlashPro: http://www.microsemi.com/soc/download/program_debug/flashpro/
- Silicon Sculptor: http://www.microsemi.com/soc/download/program_debug/ss/

Use the Most Recent Adapter Module with Silicon Sculptor

Occasionally, Microsemi makes modifications to the adapter modules to improve programming yields and programming times. To identify the latest version of each module before programming, visit http://www.microsemi.com/soc/products/hardware/program_debug/ss/modules.aspx.

Perform Routine Hardware Self-Diagnostic Test

- Adapter modules must be regularly cleaned. Adapter modules need to be inserted carefully into the programmer to make sure the DIN connectors (pins at the back side) are not damaged.
- FlashPro

  The self-test is only applicable when programming with FlashPro and FlashPro3 programmers. It is not supported with FlashPro4 or FlashPro Lite. To run the self-diagnostic test, follow the instructions given in the "Performing a Self-Test" section of http://www.microsemi.com/soc/documents/FlashPro_UG.pdf.
- Silicon Sculptor

  The self-diagnostic test verifies correct operation of the pin drivers, power supply, CPU, memory, and adapter module. This test should be performed with an adapter module installed and before every programming session. At minimum, the test must be executed every week. To perform self-diagnostic testing using the Silicon Sculptor software, perform the following steps, depending on the operating system:

  – DOS: From anywhere in the software, type **ALT + D**.

  – Windows: Click **Device** > choose **Actel Diagnostic** > select the **Test** tab > click **OK**.

  Silicon Sculptor programmers must be verified annually for calibration. Refer to the *Silicon Sculptor Verification of Calibration Work Instruction* document on the website.

Signal Integrity While Using ISP

For ISP of flash devices, customers are expected to follow the board-level guidelines provided on the Microsemi SoC Products Group website. These guidelines are discussed in the datasheets and application notes (refer to the “Related Documents” section of the datasheet for application note links). Customers are also expected to troubleshoot board-level signal integrity issues by measuring voltages and taking oscilloscope plots.

Programming Failure Allowances

Microsemi has strict policies regarding programming failure allowances. Please refer to *Programming and Functional Failure Guidelines* on the Microsemi SoC Products Group website for details.

Contacting the Customer Support Group

Highly skilled engineers staff the Customer Applications Center from 7:00 A.M. to 6:00 P.M., Pacific time, Monday through Friday. You can contact the center by one of the following methods:

Electronic Mail

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. Microsemi monitors the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and contact information for efficient processing of your request. The technical support email address is soc_tech@microsemi.com.

Telephone

Our Technical Support Hotline answers all calls. The center retrieves information, such as your name, company name, telephone number, and question. Once this is done, a case number is assigned. Then the center forwards the information to a queue where the first available applications engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific time, Monday through Friday.

The Customer Applications Center number is (800) 262-1060.

European customers can call +44 (0) 1256 305 600.

Related Documents

Below is a list of related documents, their location on the Microsemi SoC Products Group website, and a brief summary of each document.

Application Notes

Programming Antifuse Devices

http://www.microsemi.com/soc/documents/AntifuseProgram_AN.pdf

Implementation of Security in Actel's ProASIC and ProASIC^{PLUS} Flash-Based FPGAs

http://www.microsemi.com/soc/documents/Flash_Security_AN.pdf

User's Guides

FlashPro Programmers

FlashPro4,[1] FlashPro3, FlashPro Lite, and FlashPro[2]

http://www.microsemi.com/soc/products/hardware/program_debug/flashpro/default.aspx

FlashPro User's Guide

http://www.microsemi.com/soc/documents/FlashPro_UG.pdf

The FlashPro User's Guide includes hardware and software setup, self-test instructions, use instructions, and a troubleshooting / error message guide.

Silicon Sculptor 3 and Silicon Sculptor II

http://www.microsemi.com/soc/products/hardware/program_debug/ss/default.aspx

Other Documents

http://www.microsemi.com/soc/products/solutions/security/default.aspx#flashlock

The security resource center describes security in Microsemi Flash FPGAs.

Quality and Reliability Guide

http://www.microsemi.com/soc/documents/RelGuide.pdf

Programming and Functional Failure Guidelines

http://www.microsemi.com/soc/documents/FA_Policies_Guidelines_5-06-00002.pdf

1. *FlashPro4 replaced FlashPro3 in Q1 2010.*
2. *FlashPro is no longer available.*

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	FlashPro4 is a replacement for FlashPro3 and has been added to this chapter. FlashPro is no longer available.	N/A
	The chapter was updated to include SmartFusion devices.	N/A
	The following were deleted: "Live at Power-Up (LAPU) or Boot PROM" section "Design Security" section Table 14-2 • Programming Features for Actel Devices and much of the text in the "Programming Features for Microsemi Devices" section "Programming Flash FPGAs" section "Return Material Authorization (RMA) Policies" section	N/A
	The "Device Programmers" section was revised.	355
	The Independent Programming Centers information was removed from the "Volume Programming Services" section.	356
	Table 16-3 • Programming Solutions was revised to add FlashPro4 and note that FlashPro is discontinued. A note was added for FlashPro Lite regarding power supply requirements.	357
	Most items were removed from Table 16-4 • Programming Ordering Codes, including FlashPro3 and FlashPro.	358
	The "Programmer Device Support" section was deleted and replaced with a reference to the Microsemi SoC Products Group website for the latest information.	358
	The "Certified Programming Solutions" section was revised to add FlashPro4 and remove Silicon Sculptor I and Silicon Sculptor 6X. Reference to *Programming and Functional Failure Guidelines* was added.	358
	The file type *.pdb was added to the "Use the Latest Version of the Designer Software to Generate Your Programming File (recommended)" section.	359
	Instructions on cleaning and careful insertion were added to the "Perform Routine Hardware Self-Diagnostic Test" section. Information was added regarding testing Silicon Sculptor programmers with an adapter module installed before every programming session verifying their calibration annually.	359
	The "Signal Integrity While Using ISP" section is new.	360
	The "Programming Failure Allowances" section was revised.	360

Date	Changes	Page
v1.3 (December 2008)	The "Programming Support in Flash Devices" section was updated to include IGLOO nano and ProASIC3 nano devices.	352
	The "Flash Devices" section was updated to include information for IGLOO nano devices. The following sentence was added: IGLOO PLUS devices can also be operated at any voltage between 1.2 V and 1.5 V; the Designer software allows 50 mV increments in the voltage.	353
	Table 16-4 · Programming Ordering Codes was updated to replace FP3-26PIN-ADAPTER with FP3-10PIN-ADAPTER-KIT.	358
	Table 14-6 · Programmer Device Support was updated to add IGLOO nano and ProASIC3 nano devices. AGL400 was added to the IGLOO portion of the table.	317
v1.2 (October 2008)	The "Programming Support in Flash Devices" section was revised to include new families and make the information more concise.	352
	Figure 16-1 · FlashPro Programming Setup and the "Programming Support in Flash Devices" section are new.	351, 352
	Table 14-6 · Programmer Device Support was updated to include A3PE600L with the other ProASIC3L devices, and the RT ProASIC3 family was added.	317
v1.1 (March 2008)	The "Flash Devices" section was updated to include the IGLOO PLUS family. The text, "Voltage switching is required in-system to switch from a 1.2 V core to 1.5 V core for programming," was revised to state, "Although the device can operate at 1.2 V core voltage, the device can only be reprogrammed when the core voltage is 1.5 V. Voltage switching is required in-system to switch from a 1.2 V supply (V_{CC}, V_{CCI}, and V_{JTAG}) to 1.5 V for programming."	353
	The ProASIC3L family was added to Table 14-6 · Programmer Device Support as a separate set of rows rather than combined with ProASIC3 and ProASIC3E devices. The IGLOO PLUS family was included, and AGL015 and A3P015 were added.	317

17 – Security in Low Power Flash Devices

Security in Programmable Logic

The need for security on FPGA programmable logic devices (PLDs) has never been greater than today. If the contents of the FPGA can be read by an external source, the intellectual property (IP) of the system is vulnerable to unauthorized copying. Fusion, IGLOO, and ProASIC3 devices contain state-of-the-art circuitry to make the flash-based devices secure during and after programming. Low power flash devices have a built-in 128-bit Advanced Encryption Standard (AES) decryption core (except for 30 k gate devices and smaller). The decryption core facilitates secure in-system programming (ISP) of the FPGA core array fabric, the FlashROM, and the Flash Memory Blocks (FBs) in Fusion devices. The FlashROM, Flash Blocks, and FPGA core fabric can be programmed independently of each other, allowing the FlashROM or Flash Blocks to be updated without the need for change to the FPGA core fabric.

Microsemi has incorporated the AES decryption core into the low power flash devices and has also included the Microsemi flash-based lock technology, FlashLock.® Together, they provide leading-edge security in a programmable logic device. Configuration data loaded into a device can be decrypted prior to being written to the FPGA core using the AES 128-bit block cipher standard. The AES encryption key is stored in on-chip, nonvolatile flash memory.

This document outlines the security features offered in low power flash devices, some applications and uses, as well as the different software settings for each application.

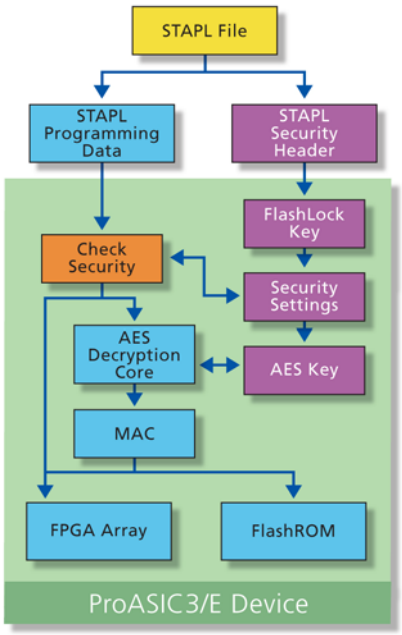


***Figure 17-1* • Overview on Security**

Security Support in Flash-Based Devices

The flash FPGAs listed in Table 17-1 support the security feature and the functions described in this document.

***Table 17-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 17-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

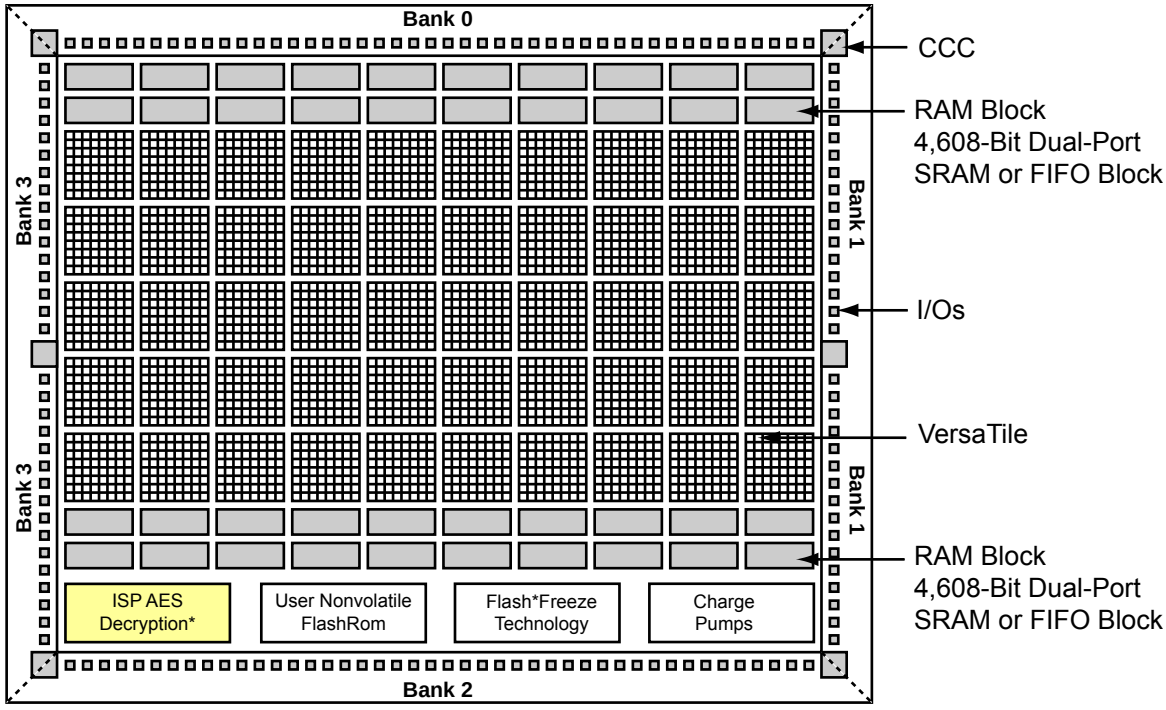
ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 17-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Security Architecture

Fusion, IGLOO, and ProASIC3 devices have been designed with the most comprehensive programming logic design security in the industry. In the architecture of these devices, security has been designed into the very fabric. The flash cells are located beneath seven metal layers, and the use of many device design and layout techniques makes invasive attacks difficult. Since device layers cannot be removed without disturbing the charge on the programmed (or erased) flash gates, devices cannot be easily deconstructed to decode the design. Low power flash devices are unique in being reprogrammable and having inherent resistance to both invasive and noninvasive attacks on valuable IP. Secure, remote ISP is now possible with AES encryption capability for the programming file during electronic transfer. Figure 17-2 shows a view of the AES decryption core inside an IGLOO device; Figure 17-3 on page 368 shows the AES decryption core inside a Fusion device. The AES core is used to decrypt the encrypted programming file when programming.



*Note: *ISP AES Decryption is not supported by 30 k gate devices and smaller. For details of other architecture features by device, refer to the appropriate family datasheet.*

***Figure 17-2* • Block Representation of the AES Decryption Core in IGLOO and ProASIC3 Devices**

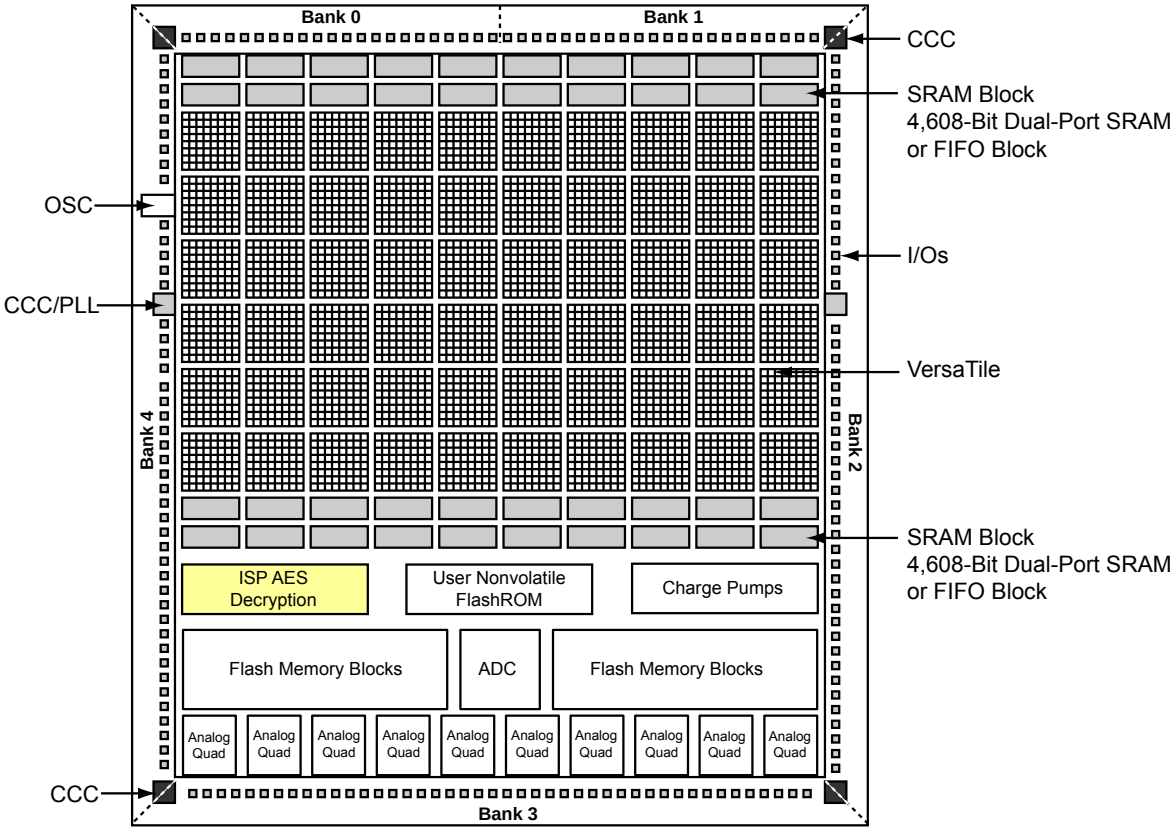


***Figure 17-3* • Block Representation of the AES Decryption Core in a Fusion AFS600 FPGA**

Security Features

IGLOO and ProASIC3 devices have two entities inside: FlashROM and the FPGA core fabric. Fusion devices contain three entities: FlashROM, FBs, and the FPGA core fabric. The parts can be programmed or updated independently with a STAPL programming file. The programming files can be AES-encrypted or plaintext. This allows maximum flexibility in providing security to the entire device. Refer to the "Programming Flash Devices" section on page 351 for information on the FlashROM structure.

Unlike SRAM-based FPGA devices, which require a separate boot PROM to store programming data, low power flash devices are nonvolatile, and the secured configuration data is stored in on-chip flash cells that are part of the FPGA fabric. Once programmed, this data is an inherent part of the FPGA array and does not need to be loaded at system power-up. SRAM-based FPGAs load the configuration bitstream upon power-up; therefore, the configuration is exposed and can be read easily.

The built-in FPGA core, FBs, and FlashROM support programming files encrypted with the 128-bit AES (FIPS-192) block ciphers. The AES key is stored in dedicated, on-chip flash memory and can be programmed before the device is shipped to other parties (allowing secure remote field updates).

Security in ARM-Enabled Low Power Flash Devices

There are slight differences between the regular flash devices and the ARM®-enabled flash devices, which have the M1 and M7 prefix.

The AES key is used by Microsemi and preprogrammed into the device to protect the ARM IP. As a result, the design is encrypted along with the ARM IP, according to the details below.

Cortex-M1 Device Security

Cortex-M1–enabled devices are shipped with the following security features:

- FPGA array enabled for AES-encrypted programming and verification
- FlashROM enabled for AES-encrypted Write and Verify
- Fusion Embedded Flash Memory enabled for AES-encrypted Write

AES Encryption of Programming Files

Low power flash devices employ AES as part of the security mechanism that prevents invasive and noninvasive attacks. The mechanism entails encrypting the programming file with AES encryption and then passing the programming file through the AES decryption core, which is embedded in the device. The file is decrypted there, and the device is successfully programmed. The AES master key is stored in on-chip nonvolatile memory (flash). The AES master key can be preloaded into parts in a secure programming environment (such as the Microsemi In-House Programming center), and then "blank" parts can be shipped to an untrusted programming or manufacturing center for final personalization with an AES-encrypted bitstream. Late-stage product changes or personalization can be implemented easily and securely by simply sending a STAPL file with AES-encrypted data. Secure remote field updates over public networks (such as the Internet) are possible by sending and programming a STAPL file with AES-encrypted data.

The AES key protects the programming data for file transfer into the device with 128-bit AES encryption. If AES encryption is used, the AES key is stored or preprogrammed into the device. To program, you must use an AES-encrypted file, and the encryption used on the file must match the encryption key already in the device.

The AES key is protected by a FlashLock security Pass Key that is also implemented in each device. The AES key is always protected by the FlashLock Key, and the AES-encrypted file does NOT contain the FlashLock Key. This FlashLock Pass Key technology is exclusive to the Microsemi flash-based device families. FlashLock Pass Key technology can also be implemented without the AES encryption option, providing a choice of different security levels.

In essence, security features can be categorized into the following three options:

- AES encryption with FlashLock Pass Key protection
- FlashLock protection only (no AES encryption)
- No protection

Each of the above options is explained in more detail in the following sections with application examples and software implementation options.

Advanced Encryption Standard

The 128-bit AES standard (FIPS-192) block cipher is the NIST (National Institute of Standards and Technology) replacement for DES (Data Encryption Standard FIPS46-2). AES has been designed to protect sensitive government information well into the 21st century. It replaces the aging DES, which NIST adopted in 1977 as a Federal Information Processing Standard used by federal agencies to protect sensitive, unclassified information. The 128-bit AES standard has 3.4×10^{38} possible 128-bit key variants, and it has been estimated that it would take 1,000 trillion years to crack 128-bit AES cipher text using exhaustive techniques. Keys are stored (securely) in low power flash devices in nonvolatile flash memory. All programming files sent to the device can be authenticated by the part prior to programming to ensure that bad programming data is not loaded into the part that may possibly damage it. All programming verification is performed on-chip, ensuring that the contents of low power flash devices remain secure.

Microsemi has implemented the 128-bit AES (Rijndael) algorithm in low power flash devices. With this key size, there are approximately 3.4×10^{38} possible 128-bit keys. DES has a 56-bit key size, which provides approximately 7.2×10^{16} possible keys. In their AES fact sheet, the National Institute of Standards and Technology uses the following hypothetical example to illustrate the theoretical security provided by AES. If one were to assume that a computing system existed that could recover a DES key in a second, it would take that same machine approximately 149 trillion years to crack a 128-bit AES key. NIST continues to make their point by stating the universe is believed to be less than 20 billion years old.[1]

The AES key is securely stored on-chip in dedicated low power flash device flash memory and cannot be read out. In the first step, the AES key is generated and programmed into the device (for example, at a secure or trusted programming site). The Microsemi Designer software tool provides AES key generation capability. After the key has been programmed into the device, the device will only correctly decrypt programming files that have been encrypted with the same key. If the individual programming file content is incorrect, a Message Authentication Control (MAC) mechanism inside the device will fail in authenticating the programming file. In other words, when an encrypted programming file is being loaded into a device that has a different programmed AES key, the MAC will prevent this incorrect data from being loaded, preventing possible device damage. See Figure 17-3 on page 368 and Figure 17-4 on page 370 for graphical representations of this process.

It is important to note that the user decides what level of protection will be implemented for the device. When AES protection is desired, the FlashLock Pass Key must be set. The AES key is a content protection mechanism, whereas the FlashLock Pass Key is a device protection mechanism. When the AES key is programmed into the device, the device still needs the Pass Key to protect the FPGA and FlashROM contents and the security settings, including the AES key. Using the FlashLock Pass Key prevents modification of the design contents by means of simply programming the device with a different AES key.

AES Decryption and MAC Authentication

Low power flash devices have a built-in 128-bit AES decryption core, which decrypts the encrypted programming file and performs a MAC check that authenticates the file prior to programming.

MAC authenticates the entire programming data stream. After AES decryption, the MAC checks the data to make sure it is valid programming data for the device. This can be done while the device is still operating. If the MAC validates the file, the device will be erased and programmed. If the MAC fails to validate, then the device will continue to operate uninterrupted.

This will ensure the following:

- Correct decryption of the encrypted programming file
- Prevention of erroneous or corrupted data being programmed during the programming file transfer
- Correct bitstream passed to the device for decryption

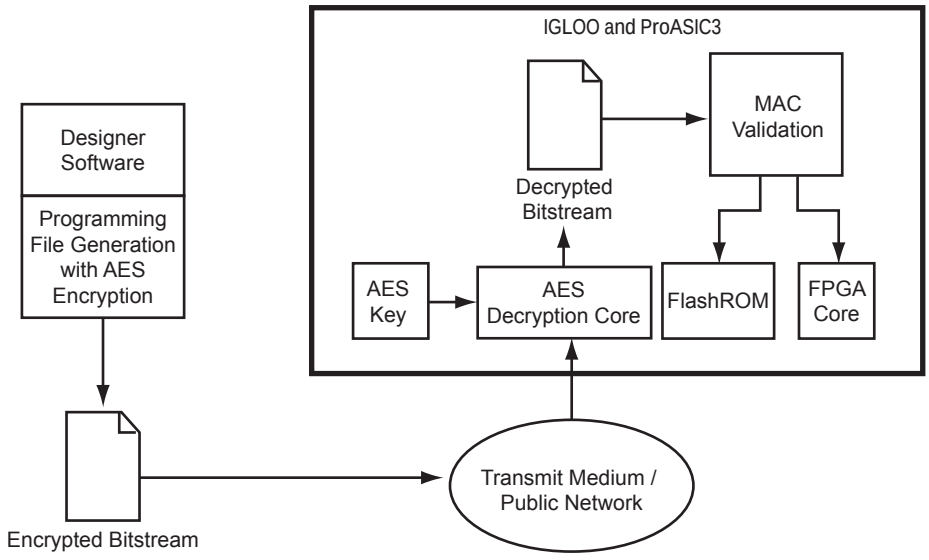


***Figure 17-4* • Example Application Scenario Using AES in IGLOO and ProASIC3 Devices**

1. *National Institute of Standards and Technology, "ADVANCED ENCRYPTION STANDARD (AES) Questions and Answers," 28 January 2002 (10 January 2005). See http://csrc.nist.gov/archive/aes/index1.html for more information.*

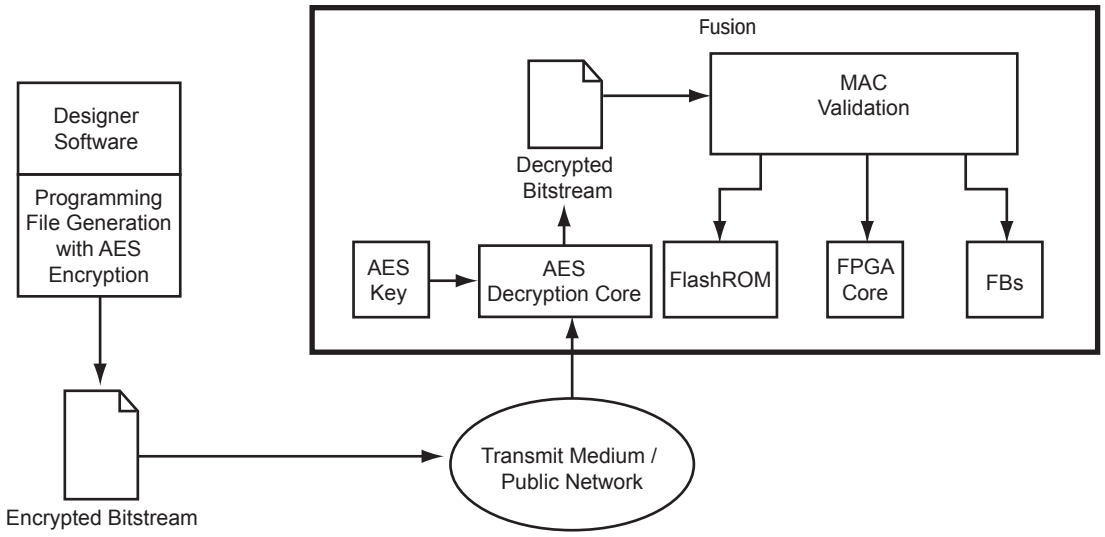


***Figure 17-5* • Example Application Scenario Using AES in Fusion Devices**

FlashLock

Additional Options for IGLOO and ProASIC3 Devices

The user also has the option of prohibiting Write operations to the FPGA array but allowing Verify operations on the FPGA array and/or Read operations on the FlashROM without the use of the FlashLock Pass Key. This option provides the user the freedom of verifying the FPGA array and/or reading the FlashROM contents after the device is programmed, without having to provide the FlashLock Pass Key. The user can incorporate AES encryption on the programming files to better enhance the level of security used.

Permanent Security Setting Options

In applications where a permanent lock is not desired, yet the security settings should not be modifiable, IGLOO and ProASIC3 devices can accommodate this requirement.

This application is particularly useful in cases where a device is located at a remote location and must be reprogrammed with a design or data update. Refer to the "Application 3: Nontrusted Environment—Field Updates/Upgrades" section on page 374 for further discussion and examples of how this can be achieved.

The user must be careful when considering the Permanent FlashLock or Permanent Security Settings option. Once the design is programmed with the permanent settings, it is not possible to reconfigure the security settings already employed on the device. Therefore, exercise careful consideration before programming permanent settings.

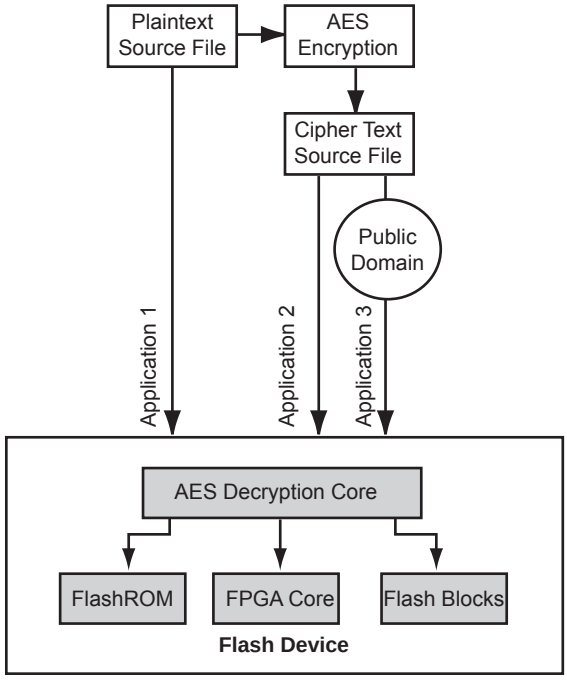
Permanent FlashLock

The purpose of the permanent lock feature is to provide the benefits of the highest level of security to IGLOO and ProASIC3 devices. If selected, the permanent FlashLock feature will create a permanent barrier, preventing any access to the contents of the device. This is achieved by permanently disabling Write and Verify access to the array, and Write and Read access to the FlashROM. After permanently locking the device, it has been effectively rendered one-time-programmable. This feature is useful if the intended applications do not require design or system updates to the device.

Security in Action

This section illustrates some applications of the security advantages of Microsemi's devices (Figure 17-6).

.



Note: Flash blocks are only used in Fusion devices

***Figure 17-6* • Security Options**

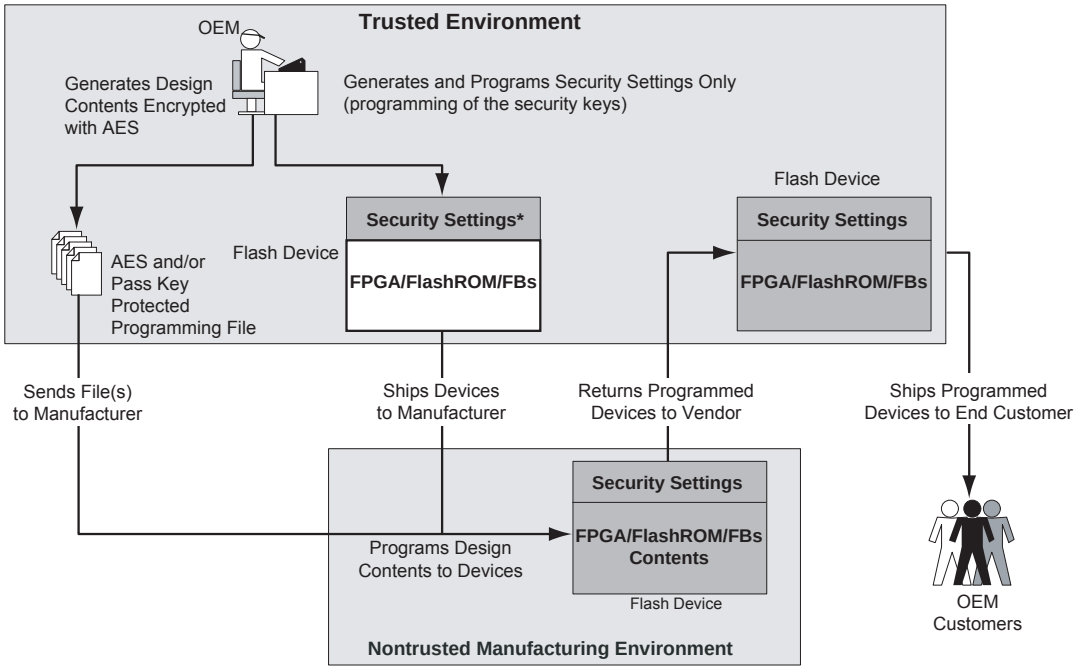
Application 1: Trusted Environment

As illustrated in Figure 17-7, this application allows the programming of devices at design locations where research and development take place. Therefore, encryption is not necessary and is optional to the user. This is often a secure way to protect the design, since the design program files are not sent elsewhere. In situations where production programming is not available at the design location, programming centers (such as Microsemi In-House Programming) provide a way of programming designs at an alternative, secure, and trusted location. In this scenario, the user generates a STAPL programming file from the Designer software in plaintext format, containing information on the entire design or the portion of the design to be programmed. The user can choose to employ the FlashLock Pass Key feature with the design. Once the design is programmed to unprogrammed devices, the design is protected by this FlashLock Pass Key. If no future programming is needed, the user can consider permanently securing the IGLOO and ProASIC3 device, as discussed in the "Permanent FlashLock" section on page 371.

Application 2: Nontrusted Environment—Unsecured Location

Often, programming of devices is not performed in the same location as actual design implementation, to reduce manufacturing cost. Overseas programming centers and contract manufacturers are examples of this scenario.

To achieve security in this case, the AES key and the FlashLock Pass Key can be initially programmed in-house (trusted environment). This is done by generating a programming file with only the security settings and no design contents. The design FPGA core, FlashROM, and (for Fusion) FB contents are generated in a separate programming file. This programming file must be set with the same AES key that was used to program to the device previously so the device will correctly decrypt this encrypted programming file. As a result, the encrypted design content programming file can be safely sent off-site to nontrusted programming locations for design programming. Figure 17-7 shows a more detailed flow for this application.



Notes:

1. *Programmed portion indicated with dark gray.*
2. *Programming of FBs applies to Fusion only.*

Figure 17-7 • **Application 2: Device Programming in a Nontrusted Environment**

Application 3: Nontrusted Environment—Field Updates/Upgrades

Programming or reprogramming of devices may occur at remote locations. Reconfiguration of devices in consumer products/equipment through public networks is one example. Typically, the remote system is already programmed with particular design contents. When design update (FPGA array contents update) and/or data upgrade (FlashROM and/or FB contents upgrade) is necessary, an updated programming file with AES encryption can be generated, sent across public networks, and transmitted to the remote system. Reprogramming can then be done using this AES-encrypted programming file, providing easy and secure field upgrades. Low power flash devices support this secure ISP using AES. The detailed flow for this application is shown in Figure 17-8. Refer to the "Microprocessor Programming of Microsemi's Low Power Flash Devices" chapter of an appropriate FPGA fabric user's guide for more information.

To prepare devices for this scenario, the user can initially generate a programming file with the available security setting options. This programming file is programmed into the devices before shipment. During the programming file generation step, the user has the option of making the security settings permanent or not. In situations where no changes to the security settings are necessary, the user can select this feature in the software to generate the programming file with permanent security settings. Microsemi recommends that the programming file use encryption with an AES key, especially when ISP is done via public domain.

For example, if the designer wants to use an AES key for the FPGA array and the FlashROM, **Permanent** needs to be chosen for this setting. At first, the user chooses the options to use an AES key for the FPGA array and the FlashROM, and then chooses **Permanently lock the security settings**. A unique AES key is chosen. Once this programming file is generated and programmed to the devices, the AES key is permanently stored in the on-chip memory, where it is secured safely. The devices are sent to distant locations for the intended application. When an update is needed, a new programming file must be generated. The programming file must use the same AES key for encryption; otherwise, the authentication will fail and the file will not be programmed in the device.

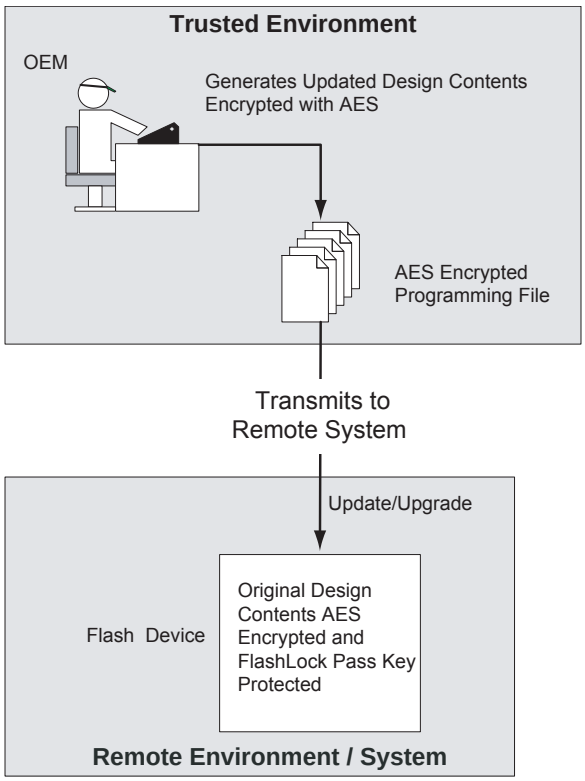


Figure 17-8 • **Application 3: Nontrusted Environment—Field Updates/Upgrades**

FlashROM Security Use Models

Each of the subsequent sections describes in detail the available selections in Microsemi Designer as an aid to understanding security applications and generating appropriate programming files for those applications. Before proceeding, it is helpful to review Figure 17-7 on page 373, which gives a general overview of the programming file generation flow within the Designer software as well as what occurs during the device programming stage. Specific settings are discussed in the following sections.

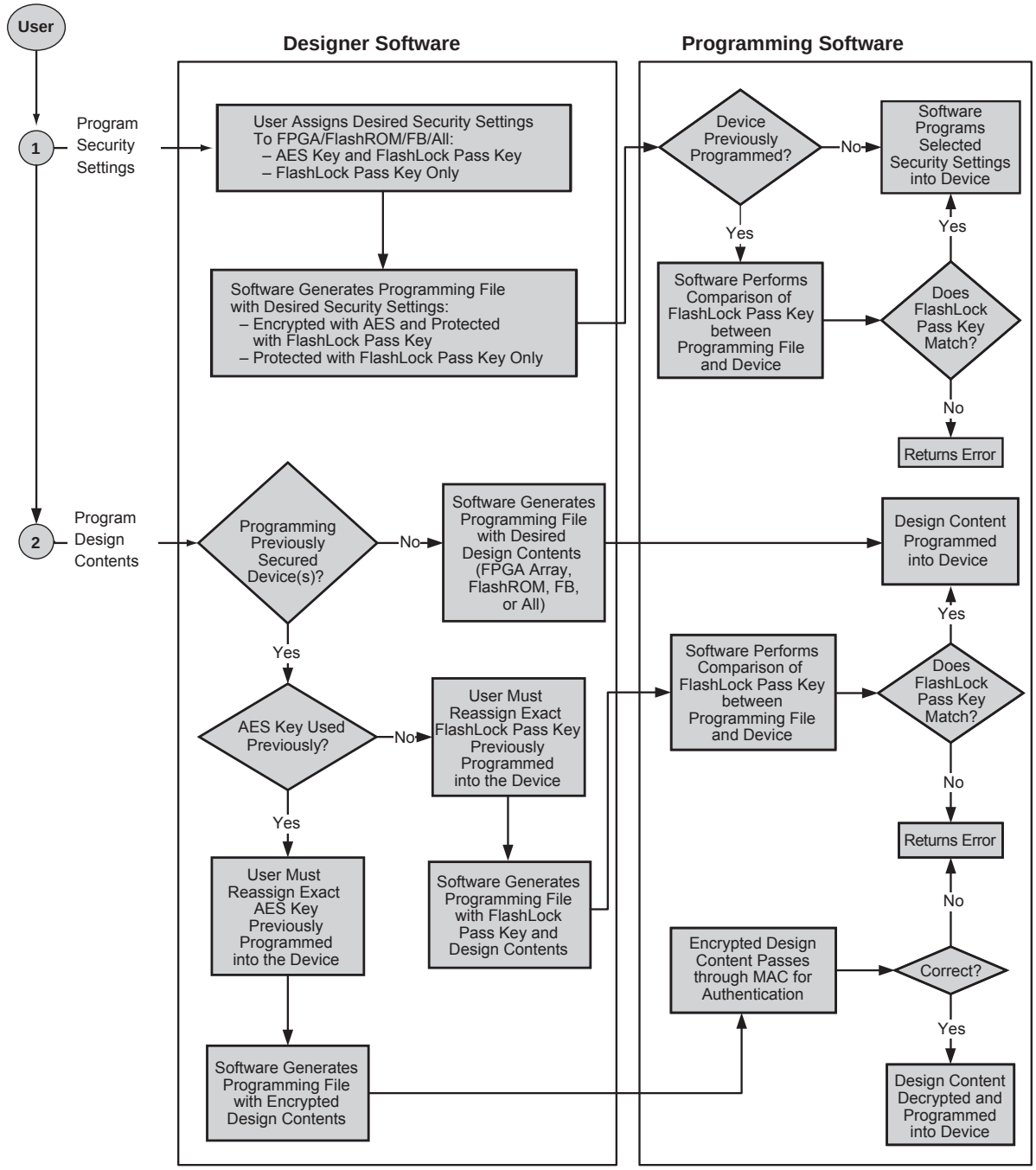
In Figure 17-7 on page 373, the flow consists of two sub-flows. Sub-flow 1 describes programming security settings to the device only, and sub-flow 2 describes programming the design contents only.

In Application 1, described in the "Application 1: Trusted Environment" section on page 373, the user does not need to generate separate files but can generate one programming file containing both security settings and design contents. Then programming of the security settings and design contents is done in one step. Both sub-flow 1 and sub-flow 2 are used.

In Application 2, described in the "Application 2: Nontrusted Environment—Unsecured Location" section on page 373, the trusted site should follow sub-flows 1 and 2 separately to generate two separate programming files. The programming file from sub-flow 1 will be used at the trusted site to program the device(s) first. The programming file from sub-flow 2 will be sent off-site for production programming.

In Application 3, described in the "Application 3: Nontrusted Environment—Field Updates/Upgrades" section on page 374, typically only sub-flow 2 will be used, because only updates to the design content portion are needed and no security settings need to be changed.

In the event that update of the security settings is necessary, see the "Reprogramming Devices" section on page 385 for details. For more information on programming low power flash devices, refer to the "In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X" section on page 391.



Note: If programming the Security Header only, just perform sub-flow 1.
If programming design content only, just perform sub-flow 2.

Figure 17-9 • **Security Programming Flows**

Generating Programming Files

Generation of the Programming File in a Trusted Environment—Application 1

As discussed in the "Application 1: Trusted Environment" section on page 373, in a trusted environment, the user can choose to program the device with plaintext bitstream content. It is possible to use plaintext for programming even when the FlashLock Pass Key option has been selected. In this application, it is not necessary to employ AES encryption protection. For AES encryption settings, refer to the next sections.

The generated programming file will include the security setting (if selected) and the plaintext programming file content for the FPGA array, FlashROM, and/or FBs. These options are indicated in Table 17-2 and Table 17-3.

***Table 17-2* • IGLOO and ProASIC3 Plaintext Security Options, No AES**

Security Protection	FlashROM Only	FPGA Core Only	Both FlashROM and FPGA
No AES / no FlashLock	✓	✓	✓
FlashLock only	✓	✓	✓
AES and FlashLock	–	–	–

***Table 17-3* • Fusion Plaintext Security Options**

Security Protection	FlashROM Only	FPGA Core Only	FB Core Only	All
No AES / no FlashLock	✓	✓	✓	✓
FlashLock	✓	✓	✓	✓
AES and FlashLock	–	–	–	–

Note: For all instructions, the programming of Flash Blocks refers to Fusion only.

For this scenario, generate the programming file as follows:

1. Select the **Silicon features to be programmed** (Security Settings, FPGA Array, FlashROM, Flash Memory Blocks), as shown in Figure 17-10 on page 378 and Figure 17-11 on page 378. Click **Next**.

   If **Security Settings** is selected (i.e., the FlashLock security Pass Key feature), an additional dialog will be displayed to prompt you to select the security level setting. If no security setting is selected, you will be directed to Step 3.

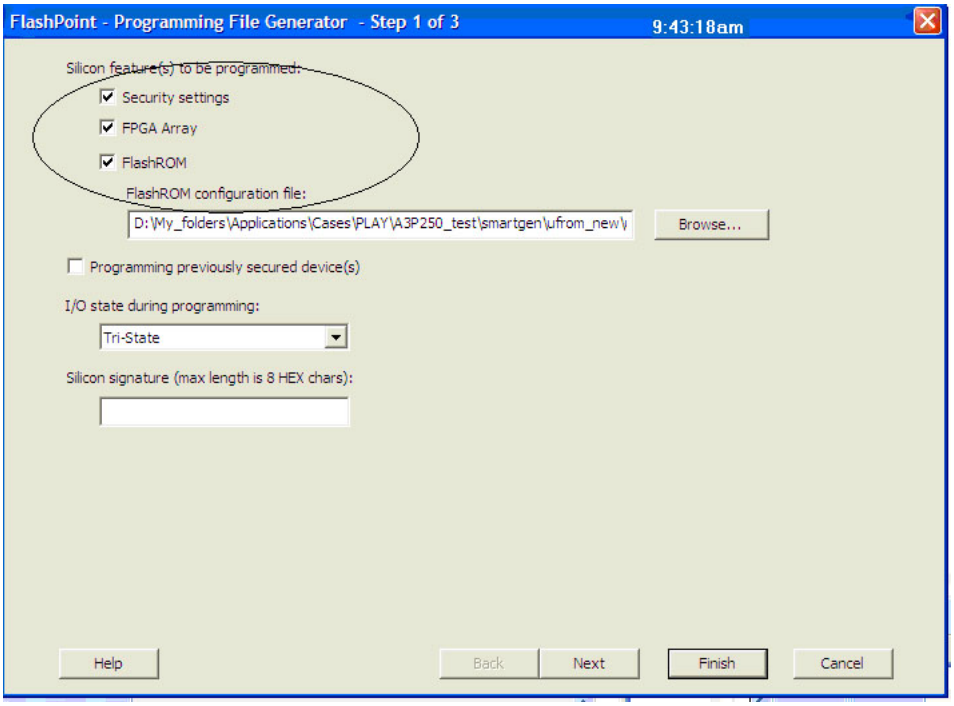


Figure 17-10 • **All Silicon Features Selected for IGLOO and ProASIC3 Devices**

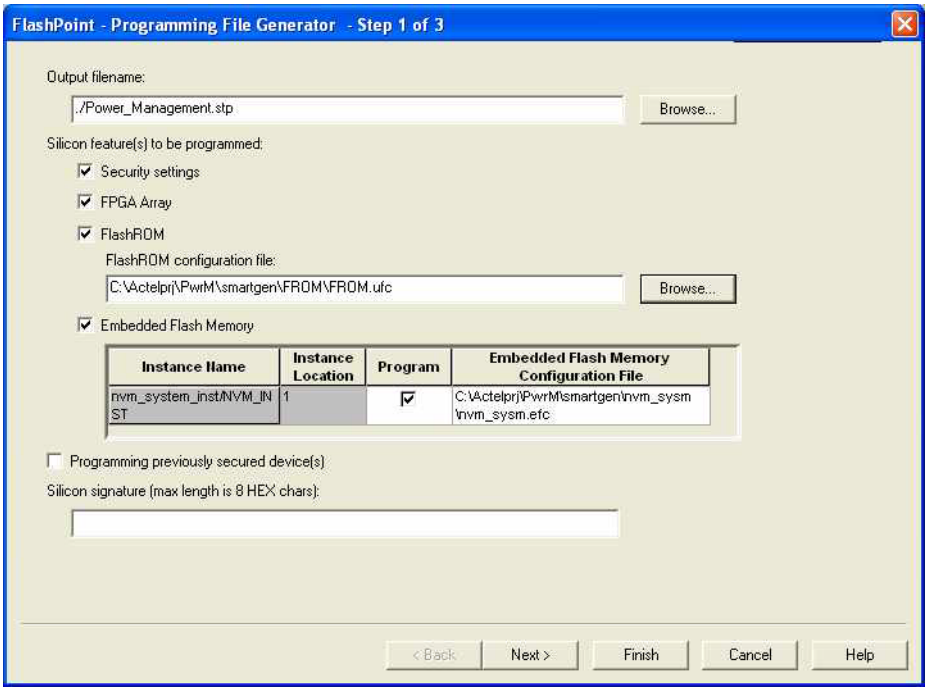


Figure 17-11 • **All Silicon Features Selected for Fusion**

2. Choose the appropriate security level setting and enter a FlashLock Pass Key. The default is the **Medium** security level (Figure 17-12). Click **Next**.

   If you want to select different options for the FPGA and/or FlashROM, this can be set by clicking **Custom Level**. Refer to the "Advanced Options" section on page 386 for different custom security level options and descriptions of each.

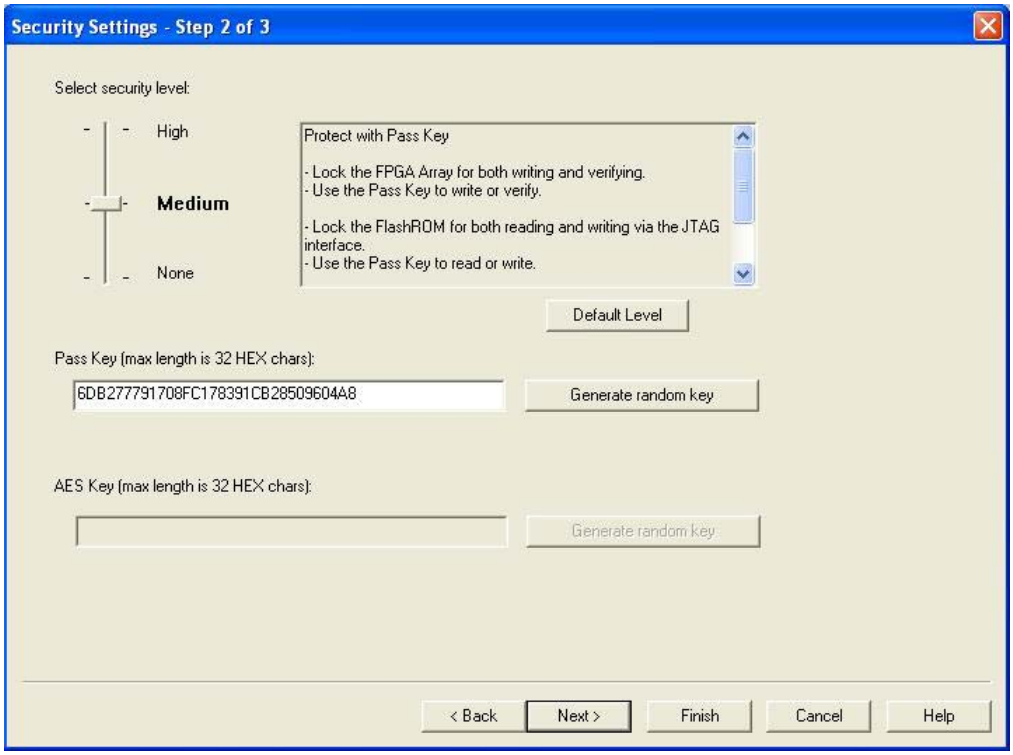


Figure 17-12 • **Medium Security Level Selected for Low Power Flash Devices**

3. Choose the desired settings for the FlashROM configurations to be programmed (Figure 17-13). Click **Finish** to generate the STAPL programming file for the design.

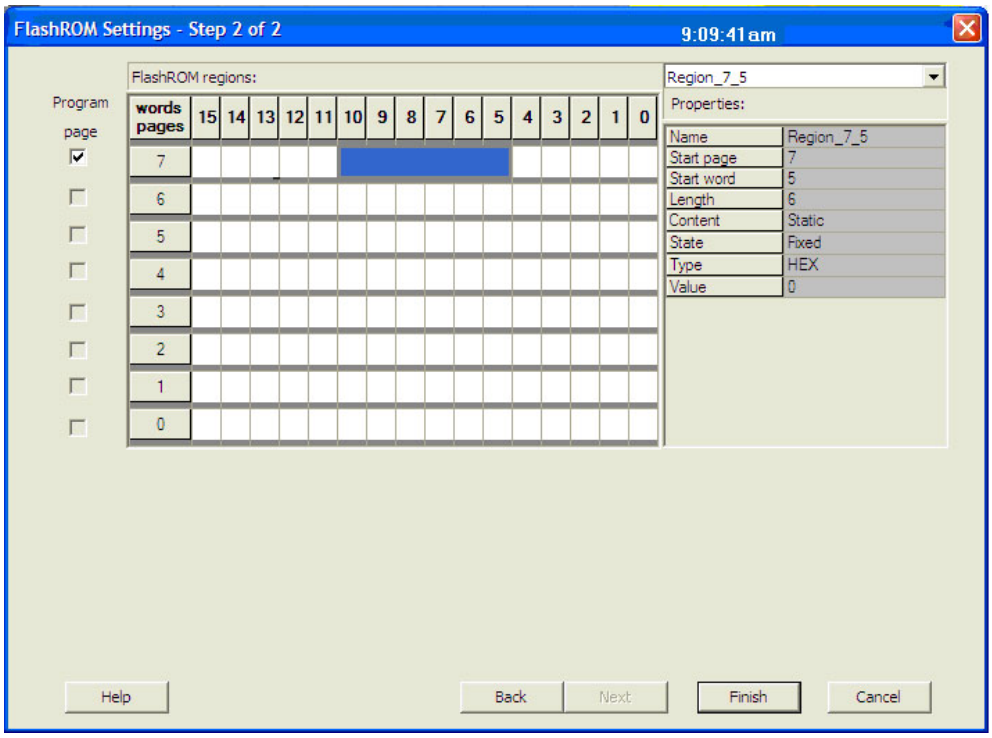


***Figure 17-13* • FlashROM Configuration Settings for Low Power Flash Devices**

Generation of Security Header Programming File Only—Application 2

As mentioned in the "Application 2: Nontrusted Environment—Unsecured Location" section on page 373, the designer may employ FlashLock Pass Key protection or FlashLock Pass Key with AES encryption on the device before sending it to a nontrusted or unsecured location for device programming. To achieve this, the user needs to generate a programming file containing only the security settings desired (Security Header programming file).

Note: If AES encryption is configured, FlashLock Pass Key protection must also be configured.

The available security options are indicated in Table 17-4 and Table 17-5 on page 381.

***Table 17-4* • FlashLock Security Options for IGLOO and ProASIC3**

Security Option	FlashROM Only	FPGA Core Only	Both FlashROM and FPGA
No AES / no FlashLock	–	–	–
FlashLock only	✓	✓	✓
AES and FlashLock	✓	✓	✓

***Table 17-5* • FlashLock Security Options for Fusion**

Security Option	FlashROM Only	FPGA Core Only	FB Core Only	All
No AES / no FlashLock	–	–	–	–
FlashLock	✓	✓	✓	✓
AES and FlashLock	✓	✓	✓	✓

For this scenario, generate the programming file as follows:

1. Select only the **Security settings** option, as indicated in Figure 17-14 and Figure 17-15 on page 382. Click **Next**.

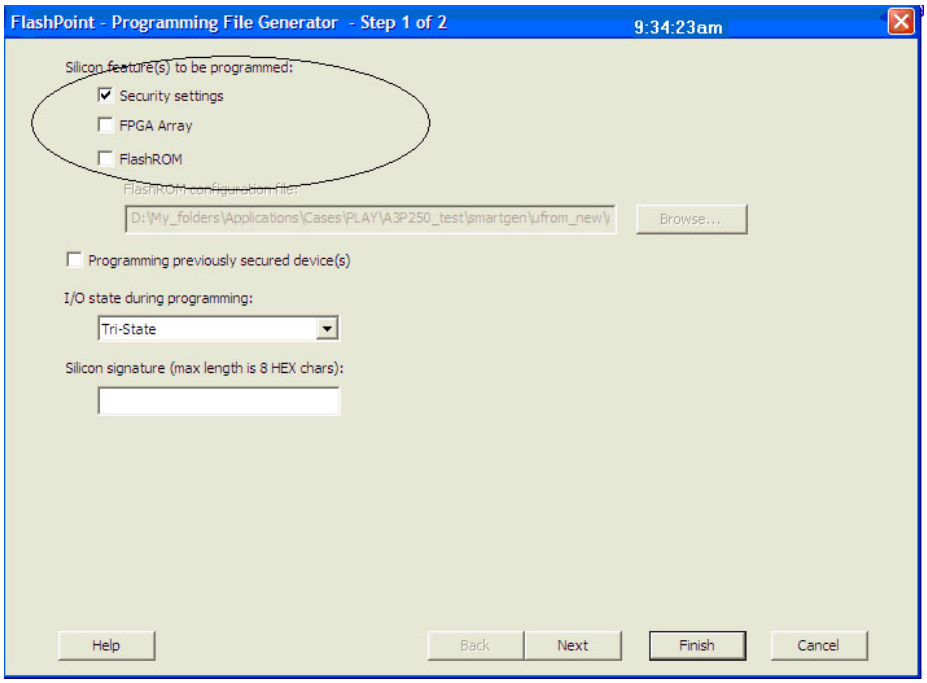


***Figure 17-14* • Programming IGLOO and ProASIC3 Security Settings Only**

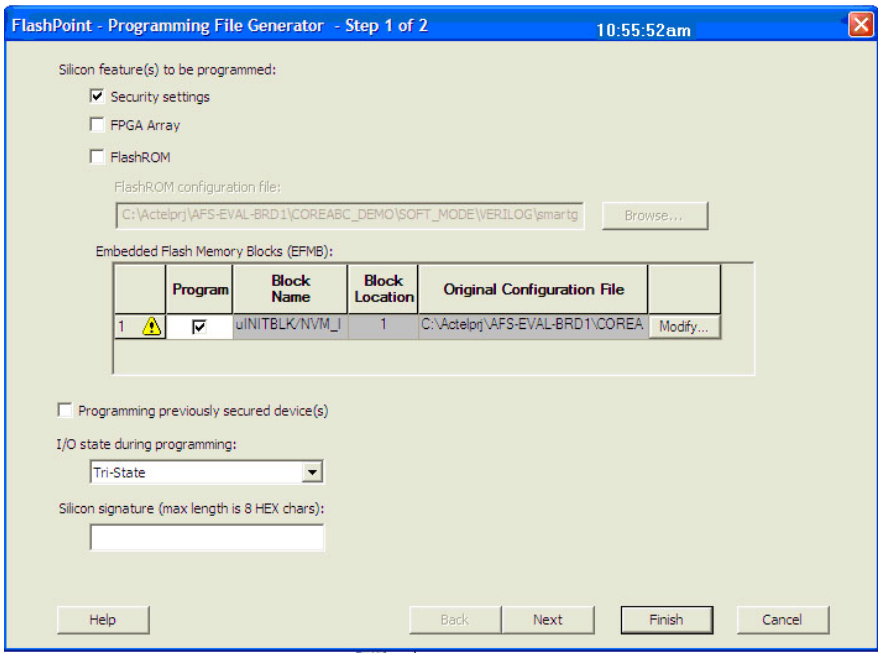


***Figure 17-15* • Programming Fusion Security Settings Only**

2. Choose the desired security level setting and enter the key(s).
   – The **High** security level employs FlashLock Pass Key with AES Key protection.
   – The **Medium** security level employs FlashLock Pass Key protection only.

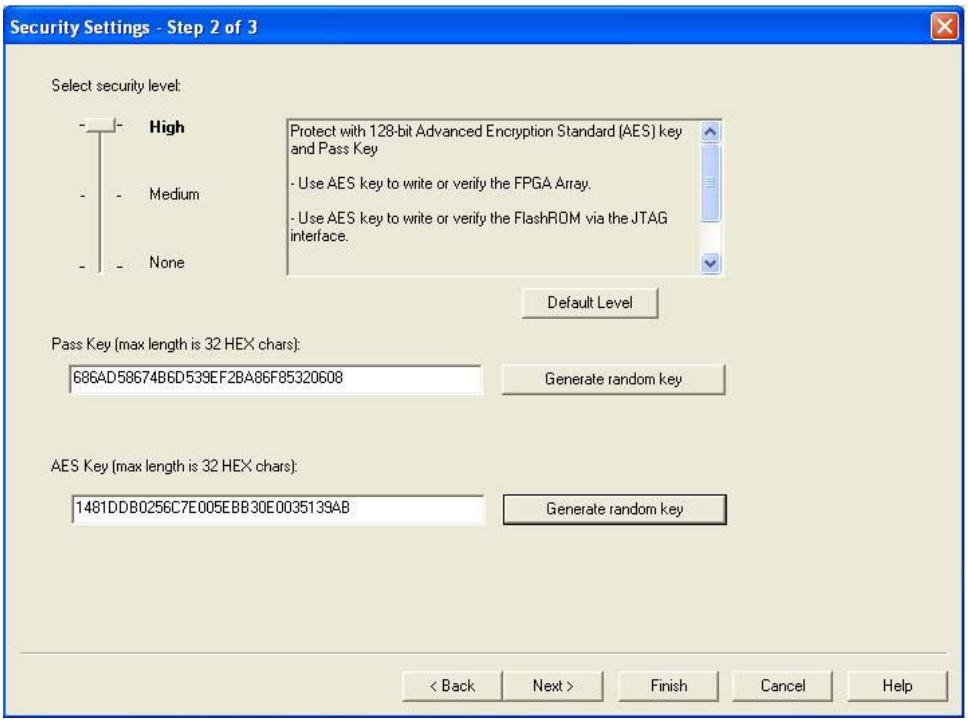


***Figure 17-16* • High Security Level to Implement FlashLock Pass Key and AES Key Protection**

Table 17-6 and Table 17-7 show all available options. If you want to implement custom levels, refer to the "Advanced Options" section on page 386 for information on each option and how to set it.

3. When done, click **Finish** to generate the Security Header programming file.

***Table 17-6* • All IGLOO and ProASIC3 Header File Security Options**

Security Option	FlashROM Only	FPGA Core Only	Both FlashROM and FPGA
No AES / no FlashLock	✓	✓	✓
FlashLock only	✓	✓	✓
AES and FlashLock	✓	✓	✓

Note: ✓ = options that may be used

***Table 17-7* • All Fusion Header File Security Options**

Security Option	FlashROM Only	FPGA Core Only	FB Core Only	All
No AES / No FlashLock	✓	✓	✓	✓
FlashLock	✓	✓	✓	✓
AES and FlashLock	✓	✓	✓	✓

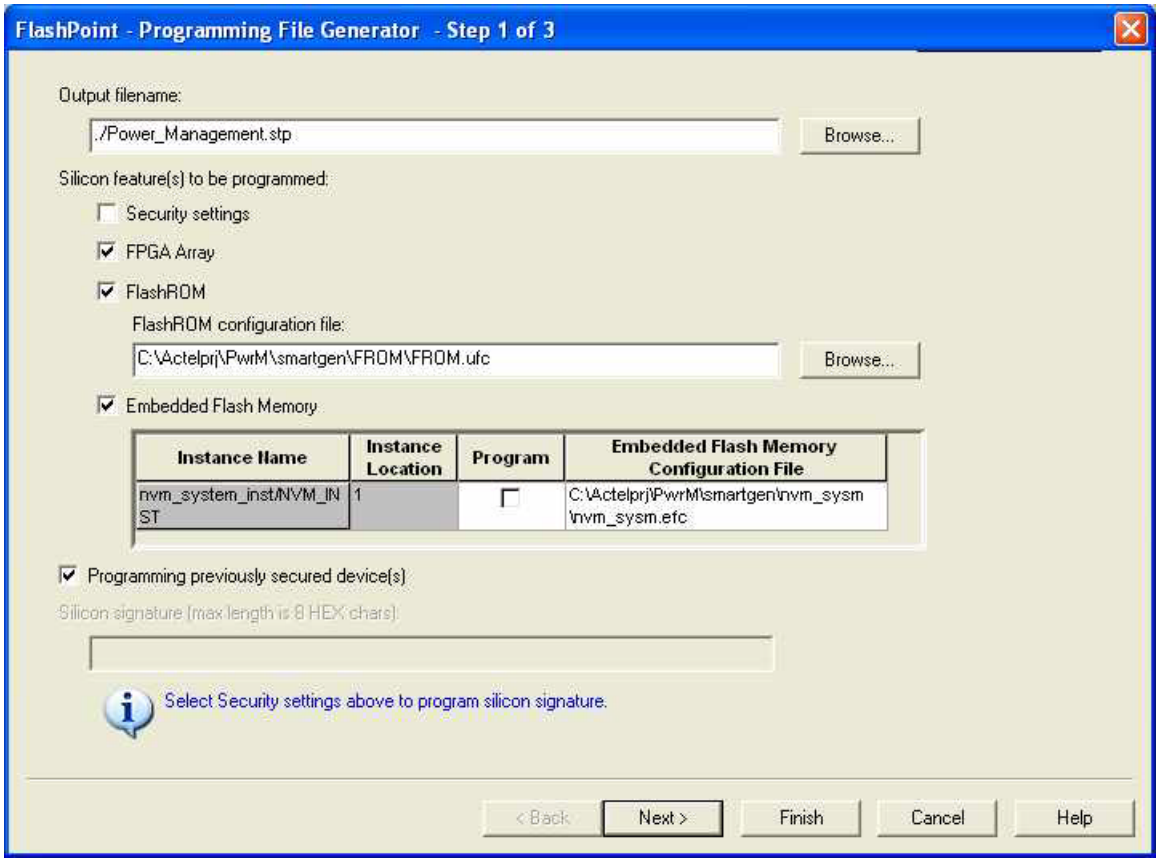
Generation of Programming Files with AES Encryption—Application 3

This section discusses how to generate design content programming files needed specifically at unsecured or remote locations to program devices with a Security Header (FlashLock Pass Key and AES key) already programmed ("Application 2: Nontrusted Environment—Unsecured Location" section on page 373 and "Application 3: Nontrusted Environment—Field Updates/Upgrades" section on page 374). In this case, the encrypted programming file must correspond to the AES key already programmed into the device. If AES encryption was previously selected to encrypt the FlashROM, FBs, and FPGA array, AES encryption must be set when generating the programming file for them. AES encryption can be applied to the FlashROM only, the FBs only, the FPGA array only, or all. The user must ensure both the FlashLock Pass Key and the AES key match those already programmed to the device(s), and all security settings must match what was previously programmed. Otherwise, the encryption and/or device unlocking will not be recognized when attempting to program the device with the programming file.

The generated programming file will be AES-encrypted.

In this scenario, generate the programming file as follows:

1. Deselect **Security settings** and select the portion of the device to be programmed (Figure 17-17 on page 384). Select **Programming previously secured device(s**). Click **Next**.



Note: The settings in this figure are used to show the generation of an AES-encrypted programming file for the FPGA array, FlashROM, and FB contents. One or all locations may be selected for encryption.

Figure 17-17 • **Settings to Program a Device Secured with FlashLock and using AES Encryption**

Choose the **High** security level to reprogram devices using both the FlashLock Pass Key and AES key protection (Figure 17-18 on page 385). Enter the AES key and click **Next**.

A device that has already been secured with FlashLock and has an AES key loaded must recognize the AES key to program the device and generate a valid bitstream in authentication. The FlashLock Key is only required to unlock the device and change the security settings.

This is what makes it possible to program in an untrusted environment. The AES key is protected inside the device by the FlashLock Key, so you can only program if you have the correct AES key. In fact, the AES key is not in the programming file either. It is the key used to encrypt the data in the file. The same key previously programmed with the FlashLock Key matches to decrypt the file.

An AES-encrypted file programmed to a device without FlashLock would not be secure, since without FlashLock to protect the AES key, someone could simply reprogram the AES key first, then program with any AES key desired or no AES key at all. This option is therefore not available in the software.

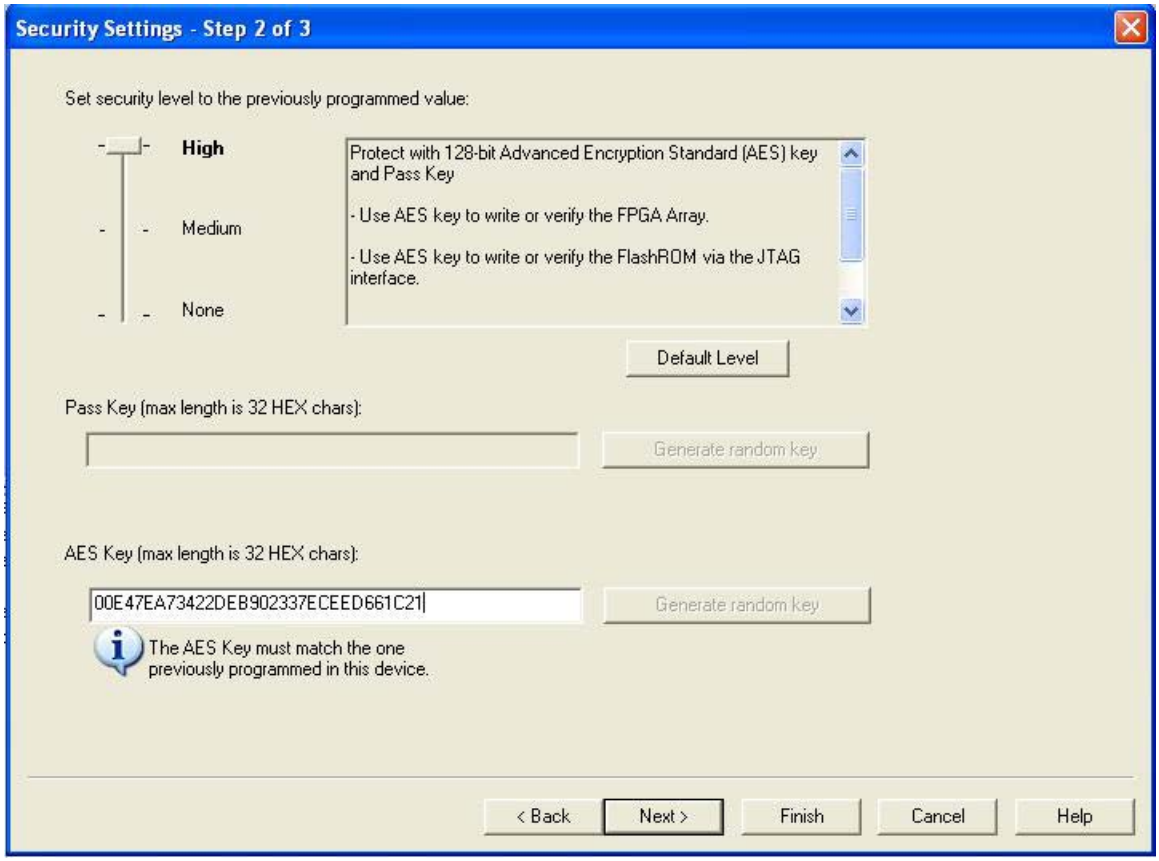


Figure 17-18 • **Security Level Set High to Reprogram Device with AES Key**

Programming with this file is intended for an unsecured environment. The AES key encrypts the programming file with the same AES key already used in the device and utilizes it to program the device.

Reprogramming Devices

Previously programmed devices can be reprogrammed using the steps in the "Generation of the Programming File in a Trusted Environment—Application 1" section on page 377 and "Generation of Security Header Programming File Only—Application 2" section on page 380. In the case where a FlashLock Pass Key has been programmed previously, the user must generate the new programming file with a FlashLock Pass Key that matches the one previously programmed into the device. The software will check the FlashLock Pass Key in the programming file against the FlashLock Pass Key in the device. The keys must match before the device can be unlocked to perform further programming with the new programming file.

Figure 17-10 on page 378 and Figure 17-11 on page 378 show the option **Programming previously secured device(s)**, which the user should select before proceeding. Upon going to the next step, the user will be notified that the same FlashLock Pass Key needs to be entered, as shown in Figure 17-19 on page 386.

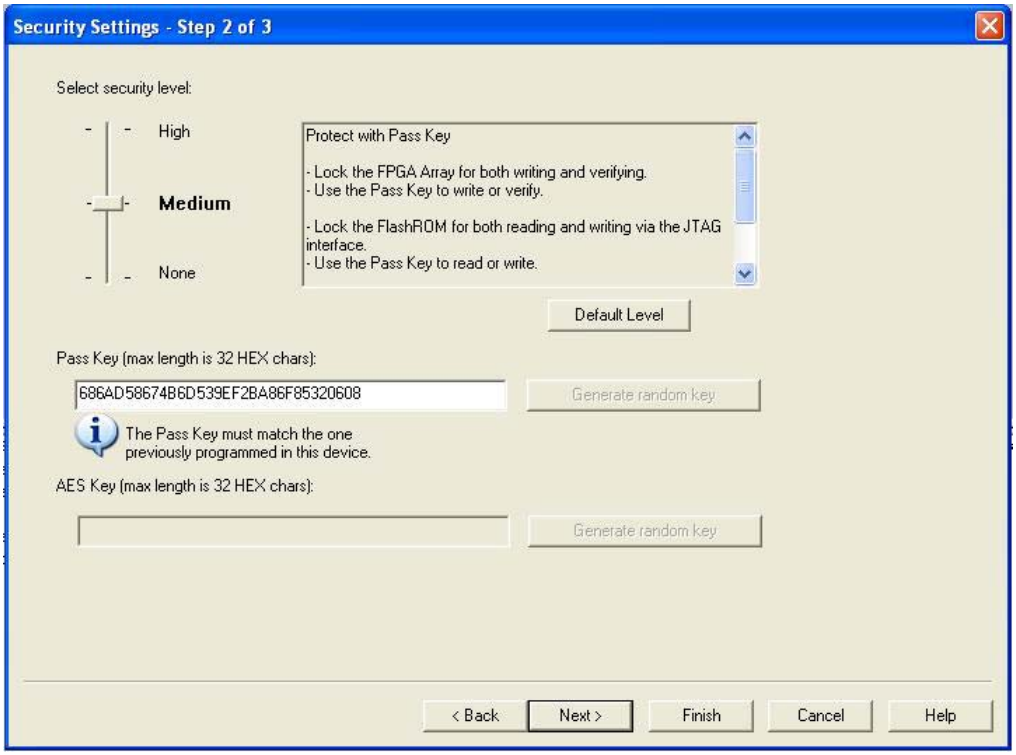


Figure 17-19 • **FlashLock Pass Key, Previously Programmed Devices**

It is important to note that when the security settings need to be updated, the user also needs to select the **Security settings** check box in Step 1, as shown in Figure 17-10 on page 378 and Figure 17-11 on page 378, to modify the security settings. The user must consider the following:

- If only a new AES key is necessary, the user must re-enter the same Pass Key previously programmed into the device in Designer and then generate a programming file with the same Pass Key and a different AES key. This ensures the programming file can be used to access and program the device and the new AES key.
- If a new Pass Key is necessary, the user can generate a new programming file with a new Pass Key (with the same or a new AES key if desired). However, for programming, the user must first load the original programming file with the Pass Key that was previously used to unlock the device. Then the new programming file can be used to program the new security settings.

Advanced Options

As mentioned, there may be applications where more complicated security settings are required. The “Custom Security Levels” section in the *FlashPro User's Guide* describes different advanced options available to aid the user in obtaining the best available security settings.

Programming File Header Definition

In each STAPL programming file generated, there will be information about how the AES key and FlashLock Pass Key are configured. Table 17-8 shows the header definitions in STAPL programming files for different security levels.

***Table 17-8* • STAPL Programming File Header Definitions by Security Level**

Security Level	STAPL File Header Definition
No security (no FlashLock Pass Key or AES key)	`NOTE "SECURITY" "Disable";`
FlashLock Pass Key with no AES key	`NOTE "SECURITY" "KEYED ";`
FlashLock Pass Key with AES key	`NOTE "SECURITY" "KEYED ENCRYPT ";`
Permanent Security Settings option enabled	`NOTE "SECURITY" "PERMLOCK ENCRYPT ";`
AES-encrypted FPGA array (for programming updates)	`NOTE "SECURITY" "ENCRYPT CORE ";`
AES-encrypted FlashROM (for programming updates)	`NOTE "SECURITY" "ENCRYPT FROM ";`
AES-encrypted FPGA array and FlashROM (for programming updates)	`NOTE "SECURITY" "ENCRYPT FROM CORE ";`

Example File Headers

STAPL Files Generated with FlashLock Key and AES Key Containing Key Information

- FlashLock Key / AES key indicated in STAPL file header definition
- Intended ONLY for secured/trusted environment programming applications

```
================================================
NOTE "CREATOR" "Designer Version: 6.1.1.108";
NOTE "DEVICE" "A3PE600";
NOTE "PACKAGE" "208 PQFP";
NOTE "DATE" "2005/04/08";
NOTE "STAPL_VERSION" "JESD71";
NOTE "IDCODE" "$123261CF";
NOTE "DESIGN" "counter32";
NOTE "CHECKSUM" "$EDB9";
NOTE "SAVE_DATA" "FRomStream";
NOTE "SECURITY" "KEYED ENCRYPT ";
NOTE "ALG_VERSION" "1";
NOTE "MAX_FREQ" "20000000";
NOTE "SILSIG" "$00000000";
NOTE "PASS_KEY" "$00123456789012345678901234567890";
NOTE "AES_KEY" "$ABCDEFABCDEFABCDEFABCDEFABCDEFAB";
================================================
```

STAPL File with AES Encryption

- Does not contain AES key / FlashLock Key information
- Intended for transmission through web or service to unsecured locations for programming

```
============================================
NOTE "CREATOR" "Designer Version: 6.1.1.108";
NOTE "DEVICE" "A3PE600";
NOTE "PACKAGE" "208 PQFP";
NOTE "DATE" "2005/04/08";
NOTE "STAPL_VERSION" "JESD71";
NOTE "IDCODE" "$123261CF";
NOTE "DESIGN" "counter32";
NOTE "CHECKSUM" "$EF57";
NOTE "SAVE_DATA" "FRomStream";
NOTE "SECURITY" "ENCRYPT FROM CORE ";
NOTE "ALG_VERSION" "1";
NOTE "MAX_FREQ" "20000000";
NOTE "SILSIG" "$00000000";
```

Conclusion

The new and enhanced security features offered in Fusion, IGLOO, and ProASIC3 devices provide state-of-the-art security to designs programmed into these flash-based devices. Microsemi low power flash devices employ the encryption standard used by NIST and the U.S. government—AES using the 128-bit Rijndael algorithm.

The combination of an on-chip AES decryption engine and FlashLock technology provides the highest level of security against invasive attacks and design theft, implementing the most robust and secure ISP solution. These security features protect IP within the FPGA and protect the system from cloning, wholesale “black box” copying of a design, invasive attacks, and explicit IP or data theft.

Glossary

Term	Explanation
Security Header programming file	Programming file used to program the FlashLock Pass Key and/or AES key into the device to secure the FPGA, FlashROM, and/or FBs.
AES (encryption) key	128-bit key defined by the user when the AES encryption option is set in the Microsemi Designer software when generating the programming file.
FlashLock Pass Key	128-bit key defined by the user when the FlashLock option is set in the Microsemi Designer software when generating the programming file. The FlashLock Key protects the security settings programmed to the device. Once a device is programmed with FlashLock, whatever settings were chosen at that time are secure.
FlashLock	The combined security features that protect the device content from attacks. These features are the following: • Flash technology that does not require an external bitstream to program the device • FlashLock Pass Key that secures device content by locking the security settings and preventing access to the device as defined by the user • AES key that allows secure, encrypted device reprogrammability

References

National Institute of Standards and Technology. “ADVANCED ENCRYPTION STANDARD (AES) Questions and Answers.” 28 January 2002 (10 January 2005).
See http://csrc.nist.gov/archive/aes/index1.html for more information.

Related Documents

User's Guides

FlashPro User's Guide

http://www.microsemi.com/soc/documents/flashpro_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.5 (August 2009)	The "CoreMP7 Device Security" section was removed from "Security in ARM-Enabled Low Power Flash Devices", since M7-enabled devices are no longer supported.	368
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 17-1 • Flash-Based FPGAs.	366
v1.3 (October 2008)	The "Security Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	366
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 17-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	366
v1.1 (March 2008)	The chapter was updated to include the IGLOO PLUS family and information regarding 15 k gate devices.	N/A
	The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	366

18 – In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X

Introduction

Microsemi's low power flash devices are all in-system programmable. This document describes the general requirements for programming a device and specific requirements for the FlashPro4/3/3X programmers[1].

IGLOO, ProASIC3, SmartFusion, and Fusion devices offer a low power, single-chip, live-at-power-up solution with the ASIC advantages of security and low unit cost through nonvolatile flash technology. Each device contains 1 kbit of on-chip, user-accessible, nonvolatile FlashROM. The FlashROM can be used in diverse system applications such as Internet Protocol (IP) addressing, user system preference storage, device serialization, or subscription-based business models. IGLOO, ProASIC3, SmartFusion, and Fusion devices offer the best in-system programming (ISP) solution, FlashLock® security features, and AES-decryption-based ISP.

ISP Architecture

Low power flash devices support ISP via JTAG and require a single VPUMP voltage of 3.3 V during programming. In addition, programming via a microcontroller in a target system is also supported.

Refer to the "Microprocessor Programming of Microsemi's Low Power Flash Devices" chapter of an appropriate FPGA fabric user's guide.

Family-specific support:

- ProASIC3, ProASIC3E, SmartFusion, and Fusion devices support ISP.
- ProASIC3L devices operate using a 1.2 V core voltage; however, programming can be done only at 1.5 V. Voltage switching is required in-system to switch from a 1.2 V core to 1.5 V core for programming.
- IGLOO and IGLOOe V5 devices can be programmed in-system when the device is using a 1.5 V supply voltage to the FPGA core.
- IGLOO nano V2 devices can be programmed at 1.2 V core voltage (when using FlashPro4 only) or 1.5 V. IGLOO nano V5 devices are programmed with a VCC core voltage of 1.5 V. Voltage switching is required in-system to switch from a 1.2 V supply (VCC,VCCI, and VJTAG) to 1.5 V for programming. The exception is that V2 devices can be programmed at 1.2 V VCC with FlashPro4.

IGLOO devices cannot be programmed in-system when the device is in Flash*Freeze mode. The device should exit Flash*Freeze mode and be in normal operation for programming to start. Programming operations in IGLOO devices can be achieved when the device is in normal operating mode and a 1.5 V core voltage is used.

JTAG 1532

IGLOO, ProASIC3, SmartFusion, and Fusion devices support the JTAG-based IEEE 1532 standard for ISP. To start JTAG operations, the IGLOO device must exit Flash*Freeze mode and be in normal operation before starting to send JTAG commands to the device. As part of this support, when a device is in an unprogrammed state, all user I/O pins are disabled. This is achieved by keeping the global IO_EN

1. *FlashPro4 replaced FlashPro3/3X in 2010 and is backward compatible with FlashPro3/3X as long as there is no connection to pin 4 on the JTAG header on the board. On FlashPro3/3X, there is no connection to pin 4 on the JTAG header; however, pin 4 is used for programming mode (Prog_Mode) on FlashPro4. When converting from FlashPro3/3X to FlashPro4, users should make sure that JTAG connectors on system boards do not have any connection to pin 4. FlashPro3X supports discrete TCK toggling that is needed to support non-JTAG compliant devices in the chain. This feature is included in FlashPro4.*

signal deactivated, which also has the effect of disabling the input buffers. The SAMPLE/PRELOAD instruction captures the status of pads in parallel and shifts them out as new data is shifted in for loading into the Boundary Scan Register (BSR). When the device is in an unprogrammed state, the OE and output BSR will be undefined; however, the input BSR will be defined as long as it is connected and being used. For JTAG timing information on setup, hold, and fall times, refer to the *FlashPro User's Guide*.

ISP Support in Flash-Based Devices

The flash FPGAs listed in Table 18-1 support the ISP feature and the functions described in this document.

***Table 18-1* • Flash-Based FPGAs Supporting ISP**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
SmartFusion	SmartFusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable microcontroller subsystem (MSS) which includes programmable analog and an ARM® Cortex™-M3 hard processor and flash memory in a monolithic device
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device
ProASIC	ProASIC	First generation ProASIC devices
	ProASIC PLUS	Second generation ProASIC devices

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 18-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 18-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Programming Voltage (VPUMP) and VJTAG

Low-power flash devices support on-chip charge pumps, and therefore require only a single 3.3 V programming voltage for the VPUMP pin during programming. When the device is not being programmed, the VPUMP pin can be left floating or can be tied (pulled up) to any voltage between 0 V and 3.6 V[1]. During programming, the target board or the FlashPro4/3/3X programmer can provide VPUMP. FlashPro4/3/3X is capable of supplying VPUMP to a single device. If more than one device is to be programmed using FlashPro4/3/3X on a given board, FlashPro4/3/3X should not be relied on to supply the VPUMP voltage. A FlashPro4/3/3X programmer is not capable of providing reliable VJTAG voltage. The board must supply VJTAG voltage to the device and the VJTAG pin of the programmer header must be connected to the device VJTAG pin. Microsemi recommends that VPUMP[2] and VJTAG power supplies be kept separate with independent filtering capacitors rather than supplying them from a common rail. Refer to the "Board-Level Considerations" section on page 401 for capacitor requirements.

Low power flash device I/Os support a bank-based, voltage-supply architecture that simultaneously supports multiple I/O voltage standards (Table 18-2). By isolating the JTAG power supply in a separate bank from the user I/Os, low power flash devices provide greater flexibility with supply selection and simplify power supply and printed circuit board (PCB) design. The JTAG pins can be run at any voltage from 1.5 V to 3.3 V (nominal). Microsemi recommends that TCK be tied to GND through a 200 ohm to 1 Kohm resistor. This prevents a possible totempole current on the input buffer stage. For TDI, TMS, and TRST pins, the devices provide an internal nominal 10 Kohm pull-up resistor. During programming, all I/O pins, except for JTAG interface pins, are tristated and weakly pulled up to VCCI. This isolates the part and prevents the signals from floating. The JTAG interface pins are driven by the FlashPro4/3/3X during programming, including the TRST pin, which is driven HIGH.

***Table 18-2* • Power Supplies**

Power Supply	Programming Mode	Current during Programming
VCC	1.2 V / 1.5 V	< 70 mA
VCCI	1.2 V / 1.5 V / 1.8 V / 2.5 V / 3.3 V (bank-selectable)	I/Os are weakly pulled up.
VJTAG	1.2 V / 1.5 V / 1.8 V / 2.5 V / 3.3 V	< 20 mA
VPUMP	3.15 V to 3.45 V	< 80 mA

Note: All supply voltages should be at 1.5 V or higher, regardless of the setting during normal operation, except for IGLOO nano, where 1.2 V VCC and VJTAG programming is allowed.

Nonvolatile Memory (NVM) Programming Voltage

SmartFusion and Fusion devices need stable VCCNVM/VCCENVM[3] (1.5 V power supply to the embedded nonvolatile memory blocks) and VCCOSC/VCCROSC[3] (3.3 V power supply to the integrated RC oscillator). The tolerance of VCCNVM/VCCENVM is ± 5% and VCCOSC/VCCROSC is ± 5%.

Unstable supply voltage on these pins can cause an NVM programming failure due to NVM page corruption. The NVM page can also be corrupted if the NVM reset pin has noise. This signal must be tied off properly.

Microsemi recommends installing the following capacitors[4] on the VCCNVM/VCCENVM and VCCOSC/VCCROSC pins:

- Add one bypass capacitor of 10 µF for each power supply plane followed by an array of decoupling capacitors of 0.1 µF.
- Add one 0.1 µF capacitor near each pin.

1. *During sleep mode in IGLOO devices connect VPUMP to GND.*
2. *VPUMP has to be quiet for successful programming. Therefore VPUMP must be separate and required capacitors must be installed close to the FPGA VPUMP pin.*
3. *VCCROSC is for SmartFusion.*
4. *The capacitors cannot guarantee reliable operation of the device if the board layout is not done properly.*

IEEE 1532 (JTAG) Interface

The supported industry-standard IEEE 1532 programming interface builds on the IEEE 1149.1 (JTAG) standard. IEEE 1532 defines the standardized process and methodology for ISP. Both silicon and software issues are addressed in IEEE 1532 to create a simplified ISP environment. Any IEEE 1532 compliant programmer can be used to program low power flash devices. Device serialization is not supported when using the IEEE1532 standard. Refer to the standard for detailed information about IEEE 1532.

Security

Unlike SRAM-based FPGAs that require loading at power-up from an external source such as a microcontroller or boot PROM, Microsemi nonvolatile devices are live at power-up, and there is no bitstream required to load the device when power is applied. The unique flash-based architecture prevents reverse engineering of the programmed code on the device, because the programmed data is stored in nonvolatile memory cells. Each nonvolatile memory cell is made up of small capacitors and any physical deconstruction of the device will disrupt stored electrical charges.

Each low power flash device has a built-in 128-bit Advanced Encryption Standard (AES) decryption core, except for the 30 k gate devices and smaller. Any FPGA core or FlashROM content loaded into the device can optionally be sent as encrypted bitstream and decrypted as it is loaded. This is particularly suitable for applications where device updates must be transmitted over an unsecured network such as the Internet. The embedded AES decryption core can prevent sensitive data from being intercepted (Figure 18-1 on page 395). A single 128-bit AES Key (32 hex characters) is used to encrypt FPGA core programming data and/or FlashROM programming data in the Microsemi tools. The low power flash devices also decrypt with a single 128-bit AES Key. In addition, low power flash devices support a Message Authentication Code (MAC) for authentication of the encrypted bitstream on-chip. This allows the encrypted bitstream to be authenticated and prevents erroneous data from being programmed into the device. The FPGA core, FlashROM, and Flash Memory Blocks (FBs), in Fusion only, can be updated independently using a programming file that is AES-encrypted (cipher text) or uses plain text.

Security in ARM-Enabled Low Power Flash Devices

There are slight differences between the regular flash device and the ARM-enabled flash devices, which have the M1 prefix.

The AES key is used by Microsemi and preprogrammed into the device to protect the ARM IP. As a result, the design will be encrypted along with the ARM IP, according to the details below.

Cortex-M1 and Cortex-M3 Device Security

Cortex-M1–enabled and Cortex-M3 devices are shipped with the following security features:

- FPGA array enabled for AES-encrypted programming and verification
- FlashROM enabled for AES-encrypted write and verify
- Embedded Flash Memory enabled for AES encrypted write

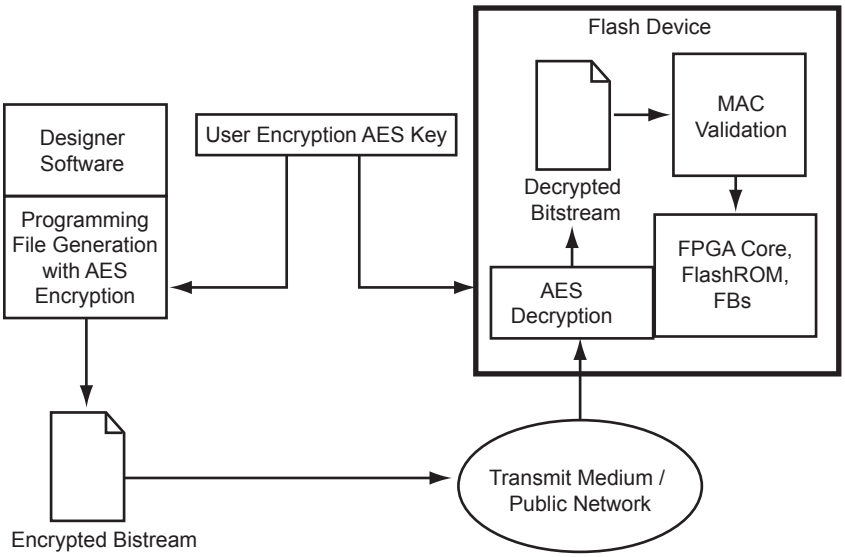


***Figure 18-1* • AES-128 Security Features**

Figure 18-2 shows different applications for ISP programming.

1. In a trusted programming environment, you can program the device using the unencrypted (plaintext) programming file.
2. You can program the AES Key in a trusted programming environment and finish the final programming in an untrusted environment using the AES-encrypted (cipher text) programming file.
3. For the remote ISP updating/reprogramming, the AES Key stored in the device enables the encrypted programming bitstream to be transmitted through the untrusted network connection.

Microsemi low power flash devices also provide the unique Microsemi FlashLock feature, which protects the Pass Key and AES Key. Unless the original FlashLock Pass Key is used to unlock the device, security settings cannot be modified. Low power flash devices do not support read-back of FPGA core-programmed data; however, the FlashROM contents can selectively be read back (or disabled) via the JTAG port based on the security settings established by the Microsemi Designer software. Refer to the "Security in Low Power Flash Devices" section on page 365 for more information.

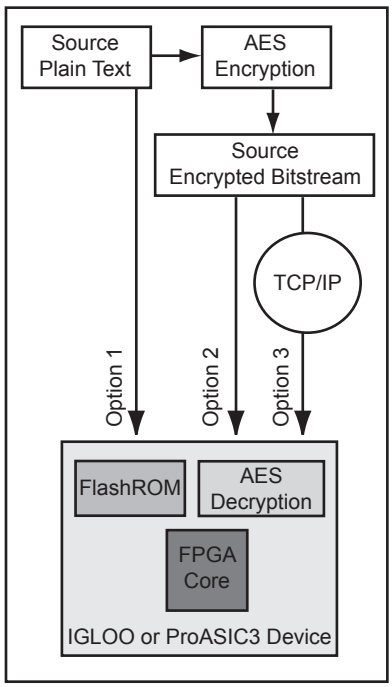


***Figure 18-2* • Different ISP Use Models**

FlashROM and Programming Files

Each low power flash device has 1 kbit of on-chip, nonvolatile flash memory that can be accessed from the FPGA core. This nonvolatile FlashROM is arranged in eight pages of 128 bits (Figure 18-3). Each page can be programmed independently, with or without the 128-bit AES encryption. The FlashROM can only be programmed via the IEEE 1532 JTAG port and cannot be programmed from the FPGA core. In addition, during programming of the FlashROM, the FPGA core is powered down automatically by the on-chip programming control logic.

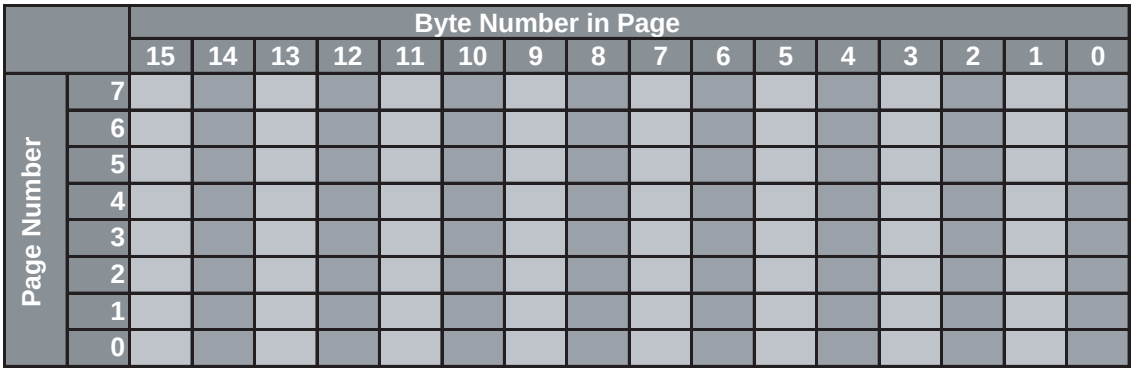


***Figure 18-3* • FlashROM Architecture**

When using FlashROM combined with AES, many subscription-based applications or device serialization applications are possible. The FROM configurator found in the Libero SoC Catalog supports easy management of the FlashROM contents, even over large numbers of devices. The FROM configurator can support FlashROM contents that contain the following:

- Static values
- Random numbers
- Values read from a file
- Independent updates of each page

In addition, auto-incrementing of fields is possible. In applications where the FlashROM content is different for each device, you have the option to generate a single STAPL file for all the devices or individual serialization files for each device. For more information on how to generate the FlashROM content for device serialization, refer to the "FlashROM in Microsemi's Low Power Flash Devices" section on page 191.

Libero SoC includes a unique tool to support the generation and management of FlashROM and FPGA programming files. This tool is called FlashPoint.

Depending on the applications, designers can use the FlashPoint software to generate a STAPL file with different contents. In each case, optional AES encryption and/or different security settings can be set.

In Designer, when you click the Programming File icon, FlashPoint launches, and you can generate STAPL file(s) with four different cases (Figure 18-4 on page 398). When the serialization feature is used during the configuration of FlashROM, you can generate a single STAPL file that will program all the devices or an individual STAPL file for each device.

The following cases present the FPGA core and FlashROM programming file combinations that can be used for different applications. In each case, you can set the optional security settings (FlashLock Pass Key and/or AES Key) depending on the application.

1. A single STAPL file or multiple STAPL files with multiple FlashROM contents and the FPGA core content. A single STAPL file will be generated if the device serialization feature is not used. You can program the whole FlashROM or selectively program individual pages.
2. A single STAPL file for the FPGA core content

3. A single STAPL file or multiple STAPL files with multiple FlashROM contents. A single STAPL file will be generated if the device serialization feature is not used. You can program the whole FlashROM or selectively program individual pages.
4. A single STAPL file to configure the security settings for the device, such as the AES Key and/or Pass Key.

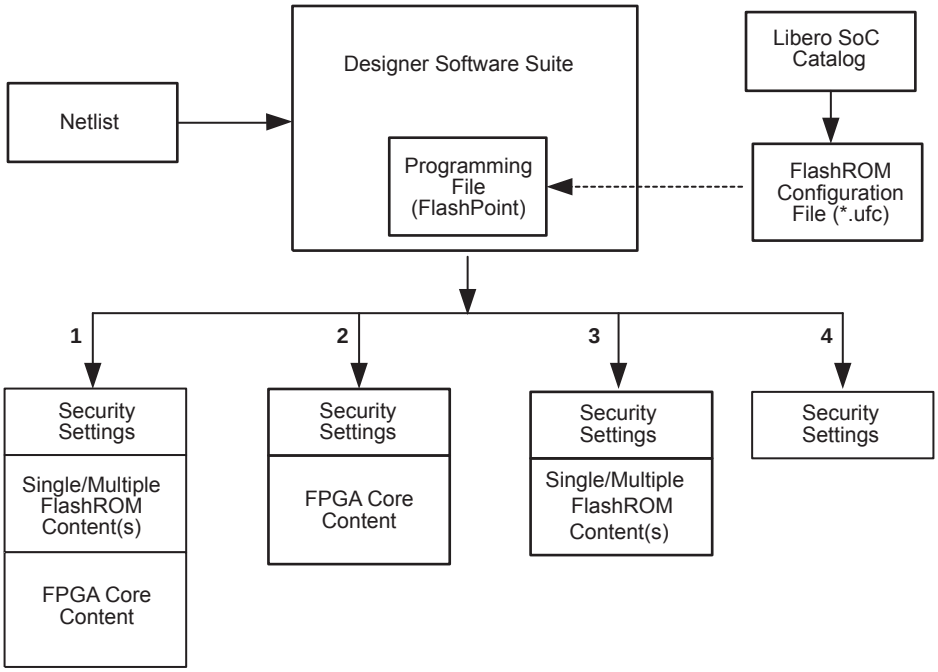


***Figure 18-4* • Flexible Programming File Generation for Different Applications**

Programming Solution

For device programming, any IEEE 1532–compliant programmer can be used; however, the FlashPro4/3/3X programmer must be used to control the low power flash device's rich security features and FlashROM programming options. The FlashPro4/3/3X programmer is a low-cost portable programmer for the Microsemi flash families. It can also be used with a powered USB hub for parallel programming. General specifications for the FlashPro4/3/3X programmer are as follows:

- Programming clock – TCK is used with a maximum frequency of 20 MHz, and the default frequency is 4 MHz.
- Programming file – STAPL
- Daisy chain – Supported. You can use the ChainBuilder software to build the programming file for the chain.
- Parallel programming – Supported. Multiple FlashPro4/3/3X programmers can be connected together using a powered USB hub or through the multiple USB ports on the PC.
- Power supply – The target board must provide VCC, VCCI, VPUMP, and VJTAG during programming. However, if there is only one device on the target board, the FlashPro4/3/3X programmer can generate the required VPUMP voltage from the USB port.

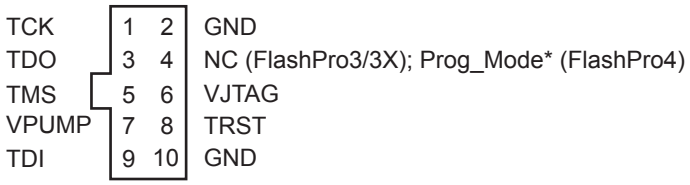
ISP Programming Header Information

The FlashPro4/3/3X programming cable connector can be connected with a 10-pin, 0.1"-pitch programming header. The recommended programming headers are manufactured by AMP (103310-1) and 3M (2510-6002UB). If you have limited board space, you can use a compact programming header manufactured by Samtec (FTSH-105-01-L-D-K). Using this compact programming header, you are required to order an additional header adapter manufactured by Microsemi SoC Products Group (FP3-10PIN-ADAPTER-KIT).

Existing ProASIC^{PLUS} family customers who are using the Samtec Small Programming Header (FTSH-113-01-L-D-K) and are planning to migrate to IGLOO or ProASIC3 devices can also use FP3-10PIN-ADAPTER-KIT.

***Table 18-3* • Programming Header Ordering Codes**

Manufacturer	Part Number	Description
AMP	103310-1	10-pin, 0.1"-pitch cable header (right-angle PCB mount angle)
3M	2510-6002UB	10-pin, 0.1"-pitch cable header (straight PCB mount angle)
Samtec	FTSH-113-01-L-D-K	Small programming header supported by FlashPro and Silicon Sculptor
Samtec	FTSH-105-01-L-D-K	Compact programming header
Samtec	FFSD-05-D-06.00-01-N	10-pin cable with 50 mil pitch sockets; included in FP3-10PIN-ADAPTER-KIT.
Microsemi	FP3-10PIN-ADAPTER-KIT	Transition adapter kit to allow FP3 to be connected to a micro 10-pin header (50 mil pitch). Includes a 6 inch Samtec FFSD-05-D-06.00-01-N cable in the kit. The transition adapter board was previously offered as FP3-26PIN-ADAPTER and includes a 26-pin adapter for design transitions from ProASIC^{PLUS} based boards to ProASIC3 based boards.



*Note: *Prog_Mode on FlashPro4 is an output signal that goes High during device programming and returns to Low when programming is complete. This signal can be used to drive a system to provide a 1.5 V programming signal to IGLOO nano, ProASIC3L, and RT ProASIC3 devices that can run with 1.2 V core voltage but require 1.5 V for programming. IGLOO nano V2 devices can be programmed at 1.2 V core voltage (when using FlashPro4 only), but IGLOO nano V5 devices are programmed with a VCC core voltage of 1.5 V.*

***Figure 18-5* • Programming Header (top view)**

***Table 18-4* • Programming Header Pin Numbers and Description**

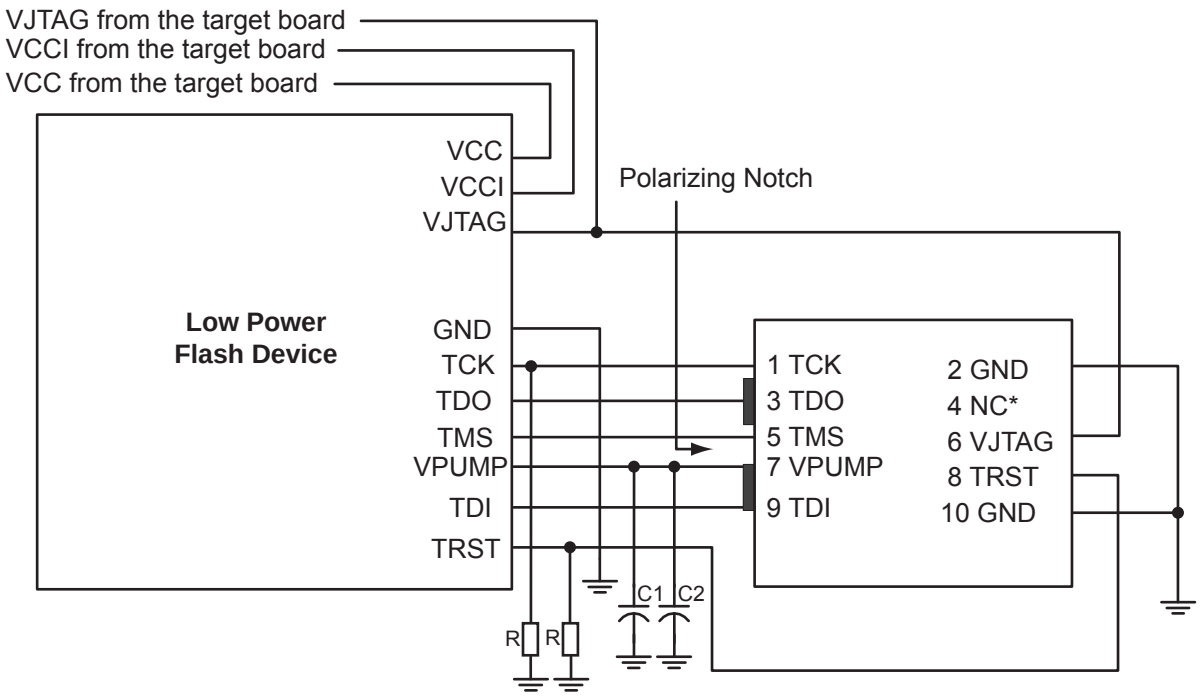
Pin	Signal	Source	Description
1	TCK	Programmer	JTAG Clock
2	GND[1]	–	Signal Reference
3	TDO	Target Board	Test Data Output
4	NC	–	No Connect (FlashPro3/3X); Prog_Mode (FlashPro4). See note associated with Figure 18-5 on page 399 regarding Prog_Mode on FlashPro4.
5	TMS	Programmer	Test Mode Select
6	VJTAG	Target Board	JTAG Supply Voltage
7	VPUMP[2]	Programmer/Target Board	Programming Supply Voltage
8	nTRST	Programmer	JTAG Test Reset (Hi-Z with 10 kΩ pull-down, HIGH, LOW, or toggling)
9	TDI	Programmer	Test Data Input
10	GND[1]	–	Signal Reference

Notes:

1. *Both GND pins must be connected.*
2. *FlashPro4/3/3X can provide VPUMP if there is only one device on the target board.*

Board-Level Considerations

A bypass capacitor is required from VPUMP to GND for all low power flash devices during programming. This bypass capacitor protects the devices from voltage spikes that may occur on the VPUMP supplies during the erase and programming cycles. Refer to the "Pin Descriptions and Packaging" chapter of the appropriate device datasheet for specific recommendations. For proper programming, 0.01 µF and 0.33 µF capacitors (both rated at 16 V) are to be connected in parallel across VPUMP and GND, and positioned as close to the FPGA pins as possible. The bypass capacitor must be placed within 2.5 cm of the device pins.



*Note: *NC (FlashPro3/3X); Prog_Mode (FlashPro4). Prog_Mode on FlashPro4 is an output signal that goes High during device programming and returns to Low when programming is complete. This signal can be used to drive a system to provide a 1.5 V programming signal to IGLOO nano, ProASIC3L, and RT ProASIC3 devices that can run with 1.2 V core voltage but require 1.5 V for programming. IGLOO nano V2 devices can be programmed at 1.2 V core voltage (when using FlashPro4 only), but IGLOO nano V5 devices are programmed with a VCC core voltage of 1.5 V.*

***Figure 18-6* • Board Layout and Programming Header Top View**

Troubleshooting Signal Integrity

Symptoms of a Signal Integrity Problem

A signal integrity problem can manifest itself in many ways. The problem may show up as extra or dropped bits during serial communication, changing the meaning of the communication. There is a normal variation of threshold voltage and frequency response between parts even from the same lot. Because of this, the effects of signal integrity may not always affect different devices on the same board in the same way. Sometimes, replacing a device appears to make signal integrity problems go away, but this is just masking the problem. Different parts on identical boards will exhibit the same problem sooner or later. It is important to fix signal integrity problems early. Unless the signal integrity problems are severe enough to completely block all communication between the device and the programmer, they may show up as subtle problems. Some of the FlashPro4/3/3X exit codes that are caused by signal integrity problems are listed below. Signal integrity problems are not the only possible cause of these

errors, but this list is intended to show where problems can occur. FlashPro4/3/3X allows TCK to be lowered from 6 MHz down to 1 MHz to allow you to address some signal integrity problems that may occur with impedance mismatching at higher frequencies. Customers are expected to troubleshoot board-level signal integrity issues by measuring voltages and taking scope plots.

Scan Chain Failure

Normally, the FlashPro4/3/3X Scan Chain command expects to see 0x1 on the TDO pin. If the command reports reading 0x0 or 0x3, it is seeing the TDO pin stuck at 0 or 1. The only time the TDO pin comes out of tristate is when the JTAG TAP state machine is in the Shift-IR or Shift-DR state. If noise or reflections on the TCK or TMS lines have disrupted the correct state transitions, the device's TAP state controller might not be in one of these two states when the programmer tries to read the device. When this happens, the output is floating when it is read and does not match the expected data value. This can also be caused by a broken TDO net. Only a small amount of data is read from the device during the Scan Chain command, so marginal problems may not always show up during this command. Occasionally a faulty programmer can cause intermittent scan chain failures.

Exit 11

This error occurs during the verify stage of programming a device. After programming the design into the device, the device is verified to ensure it is programmed correctly. The verification is done by shifting the programming data into the device. An internal comparison is performed within the device to verify that all switches are programmed correctly. Noise induced by poor signal integrity can disrupt the writes and reads or the verification process and produce a verification error. While technically a verification error, the root cause is often related to signal integrity.

Refer to the *FlashPro User's Guide* for other error messages and solutions. For the most up-to-date known issues and solutions, refer to http://www.microsemi.com/soc/support.

Conclusion

IGLOO, ProASIC3, SmartFusion, and Fusion devices offer a low-cost, single-chip solution that is live at power-up through nonvolatile flash technology. The FlashLock Pass Key and 128-bit AES Key security features enable secure ISP in an untrusted environment. On-chip FlashROM enables a host of new applications, including device serialization, subscription-based applications, and IP addressing. Additionally, as the FlashROM is nonvolatile, all of these services can be provided without battery backup.

Related Documents

User's Guides

FlashPro User's Guide

http://www.microsemi.com/soc/documents/flashpro_ug.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
August 2012	This chapter will now be published standalone as an application note in addition to being part of the IGLOO/ProASIC3/Fusion FPGA fabric user's guides (SAR 38769).	N/A
	The "ISP Programming Header Information" section was revised to update the description of FP3-10PIN-ADAPTER-KIT in Table 18-3 • Programming Header Ordering Codes, clarifying that it is the adapter kit used for ProASIC^{PLUS} based boards, and also for ProASIC3 based boards where a compact programming header is being used (SAR 36779).	399
June 2011	The VPUMP programming mode voltage was corrected in Table 18-2 • Power Supplies. The correct value is 3.15 V to 3.45 V (SAR 30668).	393
	The notes associated with Figure 18-5 • Programming Header (top view) and Figure 18-6 • Board Layout and Programming Header Top View were revised to make clear the fact that IGLOO nano V2 devices can be programmed at 1.2 V (SAR 30787).	399, 401
	Figure 18-6 • Board Layout and Programming Header Top View was revised to include resistors tying TCK and TRST to GND. Microsemi recommends tying off TCK and TRST to GND if JTAG is not used (SAR 22921). RT ProASIC3 was added to the list of device families.	401
	In the "ISP Programming Header Information" section, the kit for adapting ProASIC^{PLUS} devices was changed from FP3-10PIN-ADAPTER-KIT to FP3-26PIN-ADAPTER-KIT (SAR 20878).	399
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
	References to FlashPro4 and FlashPro3X were added to this chapter, giving distinctions between them. References to SmartGen were deleted and replaced with Libero IDE Catalog.	N/A
	The "ISP Architecture" section was revised to indicate that V2 devices can be programmed at 1.2 V VCC with FlashPro4.	391
	SmartFusion was added to Table 18-1 • Flash-Based FPGAs Supporting ISP.	392
	The "Programming Voltage (VPUMP) and VJTAG" section was revised and 1.2 V was added to Table 18-2 • Power Supplies.	393
	The "Nonvolatile Memory (NVM) Programming Voltage" section is new.	393
	Cortex-M3 was added to the "Cortex-M1 and Cortex-M3 Device Security" section.	395
	In the "ISP Programming Header Information" section, the additional header adapter ordering number was changed from FP3-26PIN-ADAPTER to FP3-10PIN-ADAPTER-KIT, which contains 26-pin migration capability.	399
	The description of NC was updated in Figure 18-5 • Programming Header (top view), Table 18-4 • Programming Header Pin Numbers and Description and Figure 18-6 • Board Layout and Programming Header Top View.	399, 400
	The "Symptoms of a Signal Integrity Problem" section was revised to add that customers are expected to troubleshoot board-level signal integrity issues by measuring voltages and taking scope plots. "FlashPro4/3/3X allows TCK to be lowered from 6 MHz down to 1 MHz to allow you to address some signal integrity problems" formerly read, "from 24 MHz down to 1 MHz." "The Scan Chain command expects to see 0x2" was changed to 0x1.	401

Date	Changes	Page
July 2010 (continued)	The "Chain Integrity Test Error Analyze Chain Failure" section was renamed to the "Scan Chain Failure" section, and the Analyze Chain command was changed to Scan Chain. It was noted that occasionally a faulty programmer can cause scan chain failures.	402
v1.5 (August 2009)	The "CoreMP7 Device Security" section was removed from "Security in ARM-Enabled Low Power Flash Devices", since M7-enabled devices are no longer supported.	395
v1.4 (December 2008)	The "ISP Architecture" section was revised to include information about core voltage for IGLOO V2 and ProASIC3L devices, as well as 50 mV increments allowable in Designer software.	391
	IGLOO nano and ProASIC3 nano devices were added to Table 18-1 • Flash-Based FPGAs Supporting ISP.	392
	A second capacitor was added to Figure 18-6 • Board Layout and Programming Header Top View.	401
v1.3 (October 2008)	The "ISP Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	392
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 18-1 • Flash-Based FPGAs Supporting ISP: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	392
v1.1 (March 2008)	The "ISP Architecture" section was updated to included the IGLOO PLUS family in the discussion of family-specific support. The text, "When 1.2 V is used, the device can be reprogrammed in-system at 1.5 V only," was revised to state, "Although the device can operate at 1.2 V core voltage, the device can only be reprogrammed when all supplies (VCC, VCCI, and VJTAG) are at 1.5 V."	391
	The "ISP Support in Flash-Based Devices" section and Table 18-1 • Flash-Based FPGAs Supporting ISP were updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	392
	The "Security" section was updated to mention that 15 k gate devices do not have a built-in 128-bit decryption core.	394
	Table 18-2 • Power Supplies was revised to remove the Normal Operation column and add a table note stating, "All supply voltages should be at 1.5 V or higher, regardless of the setting during normal operation."	393
	The "ISP Programming Header Information" section was revised to change FP3-26PIN-ADAPTER to FP3-10PIN-ADAPTER-KIT. Table 18-3 • Programming Header Ordering Codes was updated with the same change, as well as adding the part number FFSD-05-D-06.00-01-N, a 10-pin cable with 50-mil-pitch sockets.	399
	The "Board-Level Considerations" section was updated to describe connecting two capacitors in parallel across VPUMP and GND for proper programming.	401
v1.0 (January 2008)	Information was added to the "Programming Voltage (VPUMP) and VJTAG" section about the JTAG interface pin.	393
51900055-2/7.06	ACTgen was changed to SmartGen.	N/A
	In Figure 18-6 • Board Layout and Programming Header Top View, the order of the text was changed to: VJTAG from the target board VCCI from the target board VCC from the target board	401

19 – Microprocessor Programming of Microsemi's Low Power Flash Devices

Introduction

The Fusion, IGLOO, and ProASIC3 families of flash FPGAs support in-system programming (ISP) with the use of a microprocessor. Flash-based FPGAs store their configuration information in the actual cells within the FPGA fabric. SRAM-based devices need an external configuration memory, and hybrid nonvolatile devices store the configuration in a flash memory inside the same package as the SRAM FPGA. Since the programming of a true flash FPGA is simpler, requiring only one stage, it makes sense that programming with a microprocessor in-system should be simpler than with other SRAM FPGAs. This reduces bill-of-materials costs and printed circuit board (PCB) area, and increases system reliability.

Nonvolatile flash technology also gives the low power flash devices the advantage of a secure, low power, live-at-power-up, and single-chip solution. Low power flash devices are reprogrammable and offer time-to-market benefits at an ASIC-level unit cost. These features enable engineers to create high-density systems using existing ASIC or FPGA design flows and tools.

This document is an introduction to microprocessor programming only. To explain the difference between the options available, user's guides for DirectC and STAPL provide more detail on implementing each style.

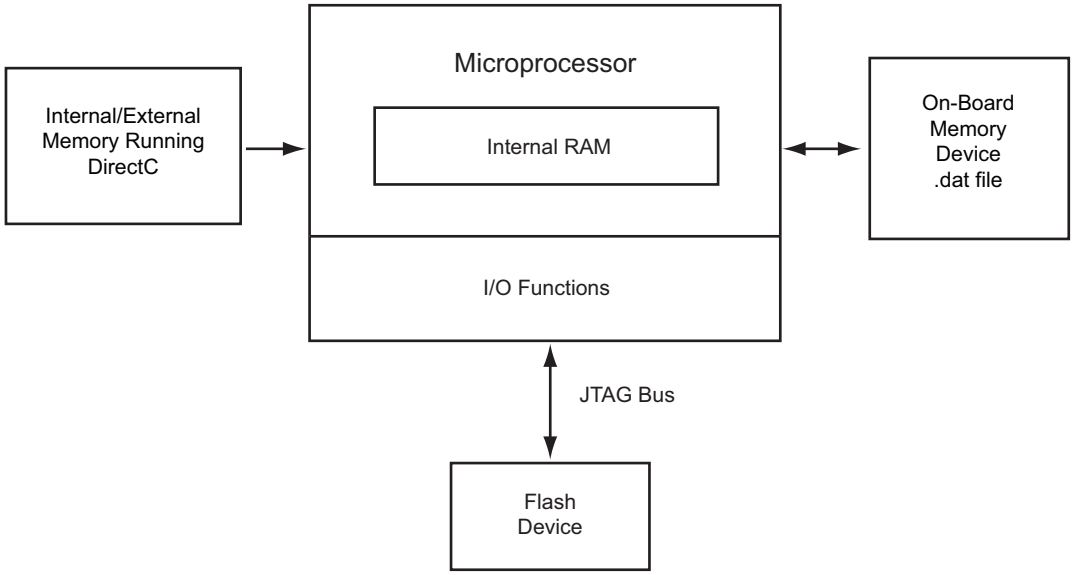


***Figure 19-1* • ISP Using Microprocessor**

Microprocessor Programming Support in Flash Devices

The flash-based FPGAs listed in Table 19-1 support programming with a microprocessor and the functions described in this document.

***Table 19-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 19-1. Where the information applies to only one device or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 19-1. Where the information applies to only one device or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Programming Algorithm

JTAG Interface

The low power flash families are fully compliant with the IEEE 1149.1 (JTAG) standard. They support all the mandatory boundary scan instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS) as well as six optional public instructions (USERCODE, IDCODE, HIGHZ, and CLAMP).

IEEE 1532

The low power flash families are also fully compliant with the IEEE 1532 programming standard. The IEEE 1532 standard adds programming instructions and associated data registers to devices that comply with the IEEE 1149.1 standard (JTAG). These instructions and registers extend the capabilities of the IEEE 1149.1 standard such that the Test Access Port (TAP) can be used for configuration activities. The IEEE 1532 standard greatly simplifies the programming algorithm, reducing the amount of time needed to implement microprocessor ISP.

Implementation Overview

To implement device programming with a microprocessor, the user should first download the C-based STAPL player or DirectC code from the Microsemi SoC Products Group website. Refer to the website for future updates regarding the STAPL player and DirectC code.

http://www.microsemi.com/soc/download/program_debug/stapl/default.aspx

http://www.microsemi.com/soc/download/program_debug/directc/default.aspx

Using the easy-to-follow user's guide, create the low-level application programming interface (API) to provide the necessary basic functions. These API functions act as the interface between the programming software and the actual hardware (Figure 19-2).

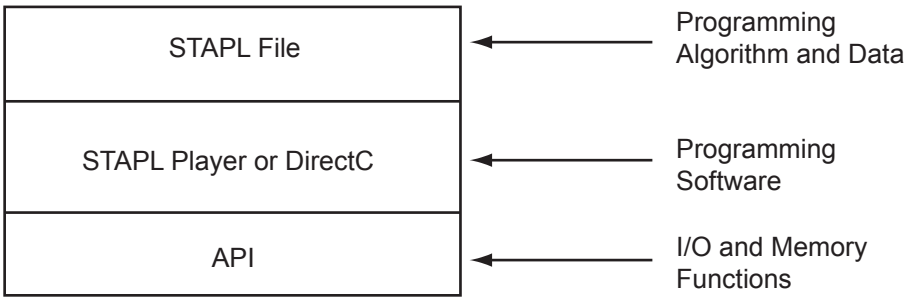


***Figure 19-2* • Device Programming Code Relationship**

The API is then linked with the STAPL player or DirectC and compiled using the microprocessor's compiler. Once the entire code is compiled, the user must download the resulting binary into the MCU system's program memory (such as ROM, EEPROM, or flash). The system is now ready for programming.

To program a design into the FPGA, the user creates a bitstream or STAPL file using the Microsemi Designer software, downloads it into the MCU system's volatile memory, and activates the stored programming binary file (Figure 19-3 on page 408). Once the programming is completed, the bitstream or STAPL file can be removed from the system, as the configuration profile is stored in the flash FPGA fabric and does not need to be reloaded at every system power-on.

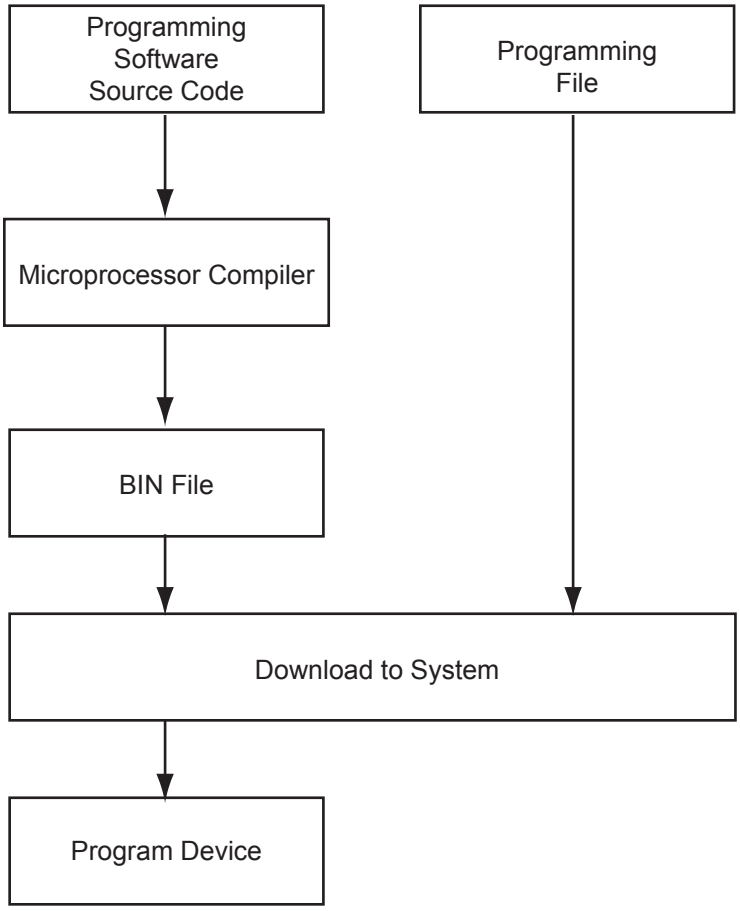


***Figure 19-3* • MCU FPGA Programming Model**

FlashROM

Microsemi low power flash devices have 1 kbit of user-accessible, nonvolatile, FlashROM on-chip. This nonvolatile FlashROM can be programmed along with the core or on its own using the standard IEEE 1532 JTAG programming interface.

The FlashROM is architected as eight pages of 128 bits. Each page can be individually programmed (erased and written). Additionally, on-chip AES security decryption can be used selectively to load data securely into the FlashROM (e.g., over public or private networks, such as the Internet). Refer to the "FlashROM in Microsemi's Low Power Flash Devices" section on page 191.

STAPL vs. DirectC

Programming the low power flash devices is performed using DirectC or the STAPL player. Both tools use the STAPL file as an input. DirectC is a compiled language, whereas STAPL is an interpreted language. Microprocessors will be able to load the FPGA using DirectC much more quickly than STAPL. This speed advantage becomes more apparent when lower clock speeds of 8- or 16-bit microprocessors are used. DirectC also requires less memory than STAPL, since the programming algorithm is directly implemented. STAPL does have one advantage over DirectC—the ability to upgrade. When a new programming algorithm is required, the STAPL user simply needs to regenerate a STAPL file using the latest version of the Designer software and download it to the system. The DirectC user must download the latest version of DirectC from Microsemi, compile everything, and download the result into the system (Figure 19-4).

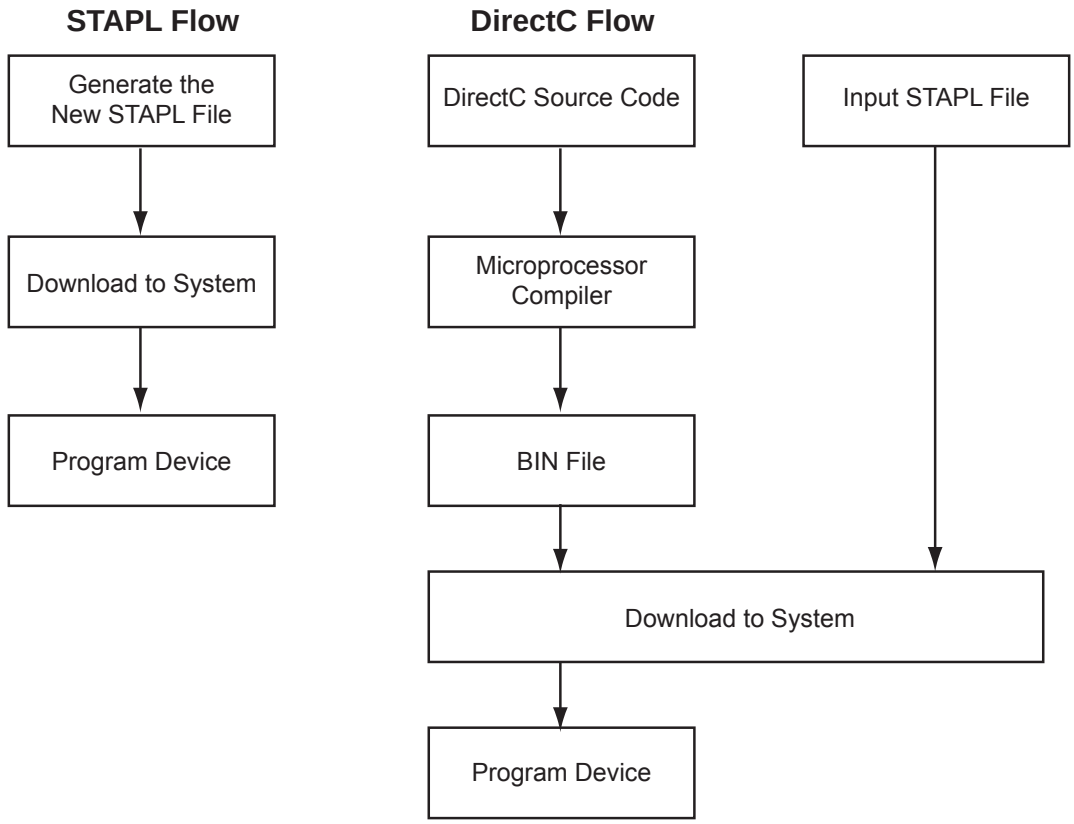


***Figure 19-4* • STAPL vs. DirectC**

Remote Upgrade via TCP/IP

Transmission Control Protocol (TCP) provides a reliable bitstream transfer service between two endpoints on a network. TCP depends on Internet Protocol (IP) to move packets around the network on its behalf. TCP protects against data loss, data corruption, packet reordering, and data duplication by adding checksums and sequence numbers to transmitted data and, on the receiving side, sending back packets and acknowledging the receipt of data.

The system containing the low power flash device can be assigned an IP address when deployed in the field. When the device requires an update (core or FlashROM), the programming instructions along with the new programming data (AES-encrypted cipher text) can be sent over the Internet to the target system via the TCP/IP protocol. Once the MCU receives the instruction and data, it can proceed with the FPGA update. Low power flash devices support Message Authentication Code (MAC), which can be used to validate data for the target device. More details are given in the "Message Authentication Code (MAC) Validation/Authentication" section.

Hardware Requirement

To facilitate the programming of the low power flash families, the system must have a microprocessor (with access to the device JTAG pins) to process the programming algorithm, memory to store the programming algorithm, programming data, and the necessary programming voltage. Refer to the relevant datasheet for programming voltages.

Security

Read-Back Prevention

The low power flash devices are designed with security in mind. Even without any security measures (such as FlashLock with AES), it is not possible to read back the programming data from a programmed device. Upon programming completion, the programming algorithm will reload the programming data into the device. The device will then use built-in circuitry to determine if it was programmed correctly.

As an additional security measure, the devices are equipped with AES decryption. AES works in two steps. The first step is to program a key into the devices in a secure or trusted programming center (such as Microsemi SoC Products Group In-House Programming (IHP) center). The second step is to encrypt any programming files with the same encryption key. The encrypted programming file will only work with the devices that have the same key. The AES used in the low power flash families is the 128-bit AES decryption engine (Rijndael algorithm).

Message Authentication Code (MAC) Validation/Authentication

As part of the AES decryption flow, the devices are equipped with a MAC validation/authentication system. MAC is an authentication tag, also called a checksum, derived by applying an on-chip authentication scheme to a STAPL file as it is loaded into the FPGA. MACs are computed and verified with the same key so they can only be verified by the intended recipient. When the MCU system receives the AES-encrypted programming data (cipher text), it can validate the data by loading it into the FPGA and performing a MAC verification prior to loading the data, via a second programming pass, into the FPGA core cells. This prevents erroneous or corrupt data from getting into the FPGA.

Low power flash devices with AES and MAC are superior to devices with only DES or 3DES encryption. Because the MAC verifies the correctness of the data, the FPGA is protected from erroneous loading of invalid programming data that could damage a device (Figure 19-5 on page 411).

The AES with MAC enables field updates over public networks without fear of having the design stolen. An encrypted programming file can only work on devices with the correct key, rendering any stolen files

useless to the thief. To learn more about the low power flash devices' security features, refer to the "Security in Low Power Flash Devices" section on page 365.

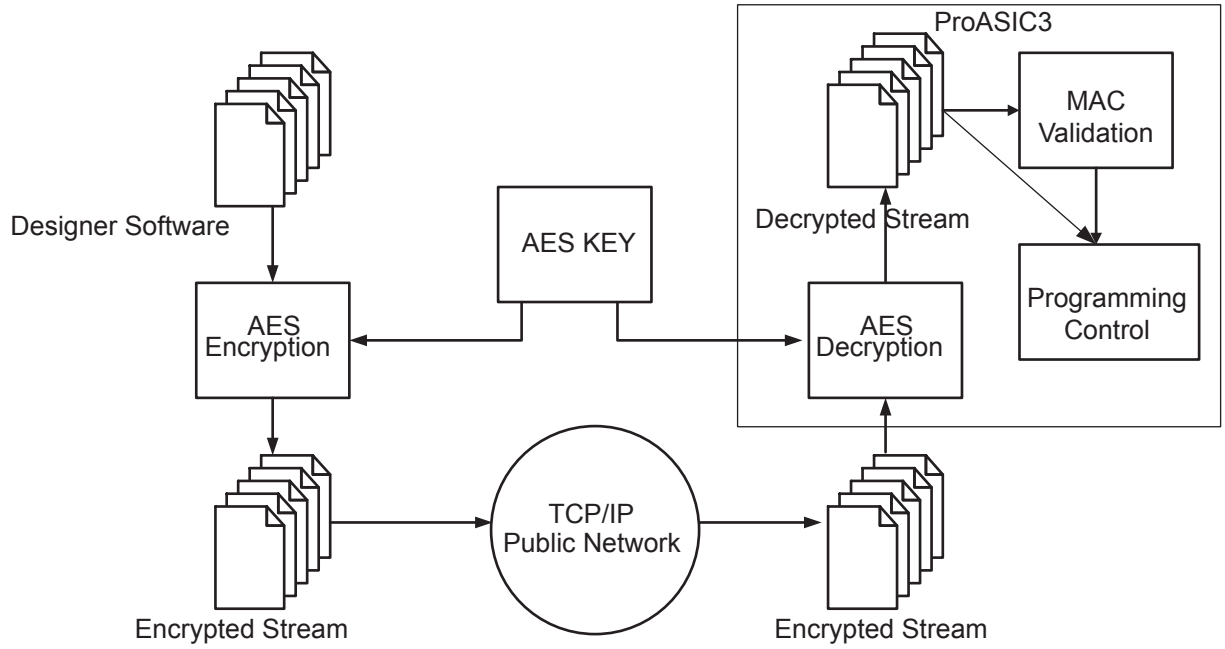


Figure 19-5 • **ProASIC3 Device Encryption Flow**

Conclusion

The Fusion, IGLOO, and ProASIC3 FPGAs are ideal for applications that require field upgrades. The single-chip devices save board space by eliminating the need for EEPROM. The built-in AES with MAC enables transmission of programming data over any network without fear of design theft. Fusion, IGLOO, and ProASIC3 FPGAs are IEEE 1532–compliant and support STAPL, making the target programming software easy to implement.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 19-1 • Flash-Based FPGAs.	406
v1.3 (October 2008)	The "Microprocessor Programming Support in Flash Devices" section was revised to include new families and make the information more concise.	406
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 19-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	406
v1.1 (March 2008)	The "Microprocessor Programming Support in Flash Devices" section was updated to include information on the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	406

20 – Boundary Scan in Low Power Flash Devices

Boundary Scan

Low power flash devices are compatible with IEEE Standard 1149.1, which defines a hardware architecture and the set of mechanisms for boundary scan testing. JTAG operations are used during boundary scan testing.

The basic boundary scan logic circuit is composed of the TAP controller, test data registers, and instruction register (Figure 20-2 on page 416).

Low power flash devices support three types of test data registers: bypass, device identification, and boundary scan. The bypass register is selected when no other register needs to be accessed in a device. This speeds up test data transfer to other devices in a test data path. The 32-bit device identification register is a shift register with four fields (LSB, ID number, part number, and version). The boundary scan register observes and controls the state of each I/O pin. Each I/O cell has three boundary scan register cells, each with serial-in, serial-out, parallel-in, and parallel-out pins.

TAP Controller State Machine

The TAP controller is a 4-bit state machine (16 states) that operates as shown in Figure 20-1.

The 1s and 0s represent the values that must be present on TMS at a rising edge of TCK for the given state transition to occur. IR and DR indicate that the instruction register or the data register is operating in that state.

The TAP controller receives two control inputs (TMS and TCK) and generates control and clock signals for the rest of the test logic architecture. On power-up, the TAP controller enters the Test-Logic-Reset state. To guarantee a reset of the controller from any of the possible states, TMS must remain HIGH for five TCK cycles. The TRST pin can also be used to asynchronously place the TAP controller in the Test-Logic-Reset state.

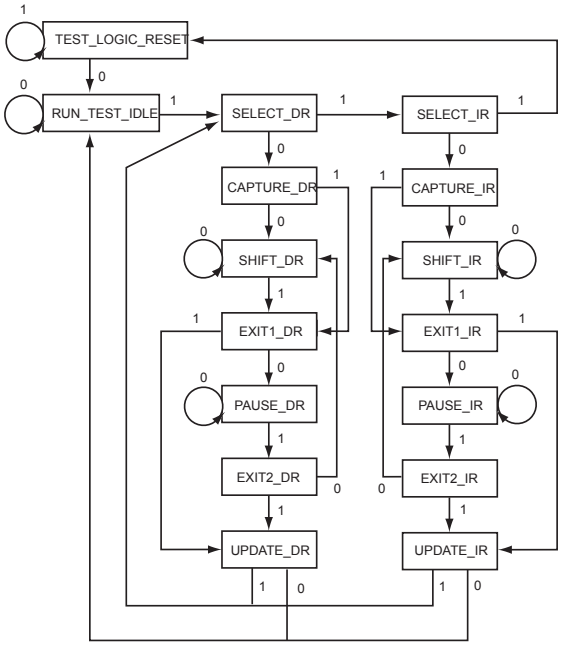


***Figure 20-1* • TAP Controller State Machine**

Microsemi's Flash Devices Support the JTAG Feature

The flash-based FPGAs listed in Table 20-1 support the JTAG feature and the functions described in this document.

***Table 20-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC®3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 20-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 20-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

Boundary Scan Support in Low Power Devices

The information in this document applies to all Fusion, IGLOO, and ProASIC3 devices. For IGLOO, IGLOO PLUS, and ProASIC3L devices, the Flash*Freeze pin must be deasserted for successful boundary scan operations. Devices cannot enter JTAG mode directly from Flash*Freeze mode.

Boundary Scan Opcodes

Low power flash devices support all mandatory IEEE 1149.1 instructions (EXTEST, SAMPLE/PRELOAD, and BYPASS) and the optional IDCODE instruction (Table 20-2).

***Table 20-2* • Boundary Scan Opcodes**

	Hex Opcode
EXTEST	00
HIGHZ	07
USERCODE	0E
SAMPLE/PRELOAD	01
IDCODE	0F
CLAMP	05
BYPASS	FF

Boundary Scan Chain

The serial pins are used to serially connect all the boundary scan register cells in a device into a boundary scan register chain (Figure 20-2 on page 416), which starts at the TDI pin and ends at the TDO pin. The parallel ports are connected to the internal core logic I/O tile and the input, output, and control ports of an I/O buffer to capture and load data into the register to control or observe the logic state of each I/O.

Each test section is accessed through the TAP, which has five associated pins: TCK (test clock input), TDI, TDO (test data input and output), TMS (test mode selector), and TRST (test reset input). TMS, TDI, and TRST are equipped with pull-up resistors to ensure proper operation when no input data is supplied to them. These pins are dedicated for boundary scan test usage. Refer to the "JTAG Pins" section in the "Pin Descriptions and Packaging" chapter of the appropriate device datasheet for pull-up/-down recommendations for TCK and TRST pins. Pull-down recommendations are also given in Table 20-3 on page 416

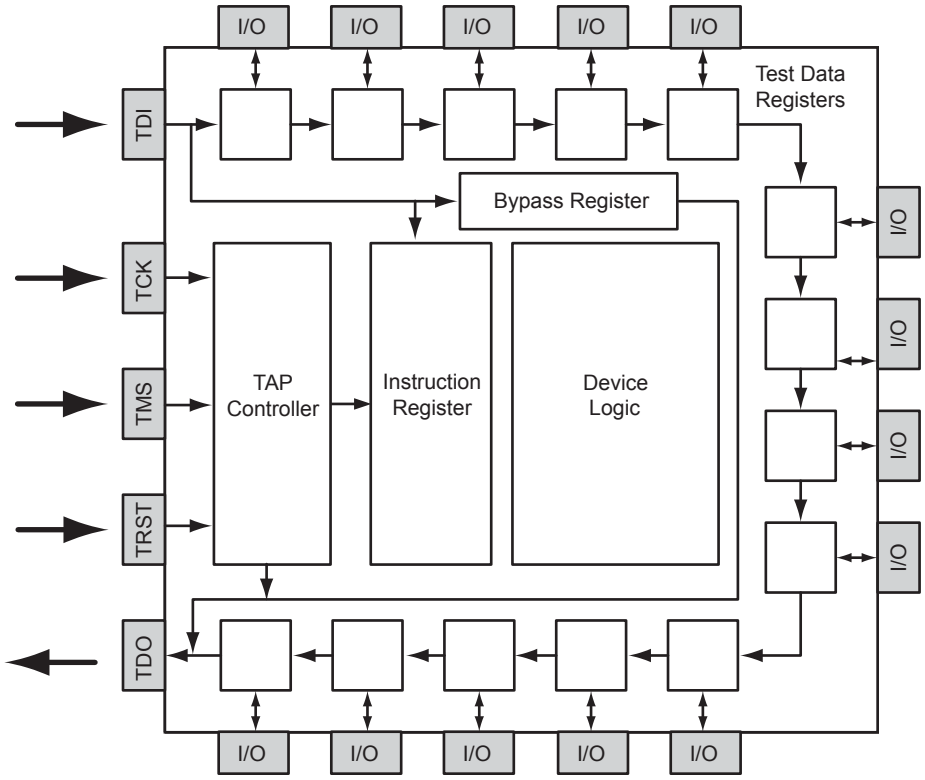


***Figure 20-2* • Boundary Scan Chain**

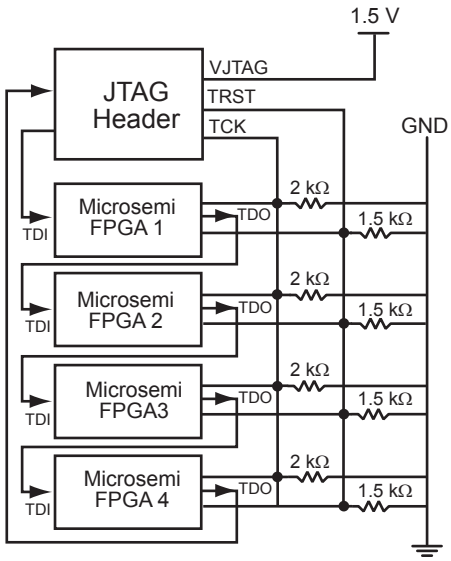
Board-Level Recommendations

Table 20-3 gives pull-down recommendations for the TRST and TCK pins.

***Table 20-3* • TRST and TCK Pull-Down Recommendations**

VJTAG	Tie-Off Resistance*
VJTAG at 3.3 V	200 Ω to 1 kΩ
VJTAG at 2.5 V	200 Ω to 1 kΩ
VJTAG at 1.8 V	500 Ω to 1 kΩ
VJTAG at 1.5 V	500 Ω to 1 kΩ
VJTAG at 1.2 V	TBD

Note: Equivalent parallel resistance if more than one device is on JTAG chain (Figure 20-3)



Note: TCK is correctly wired with an equivalent tie-off resistance of 500 Ω, which satisfies the table for VJTAG of 1.5 V. The resistor values for TRST are not appropriate in this case, as the tie-off resistance of 375 Ω is below the recommended minimum for VJTAG = 1.5 V, but would be appropriate for a VJTAG setting of 2.5 V or 3.3 V.

***Figure 20-3* • Parallel Resistance on JTAG Chain of Devices**

Advanced Boundary Scan Register Settings

You will not be able to control the order in which I/Os are released from boundary scan control. Testing has produced cases where, depending on I/O placement and FPGA routing, a 5 ns glitch has been seen on exiting programming mode. The following setting is recommended to prevent such I/O glitches:

1. In the FlashPro software, configure the advanced BSR settings for **Specify I/O Settings During Programming**.
2. Set the input BSR cell to **Low** for the input I/O.

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
August 2012	In the "Boundary Scan Chain" section, the reference made to the datasheet for pull-up/-down recommendations was changed to mention TCK and TRST pins rather than TDO and TCK pins. TDO is an output, so no pull resistor is needed (SAR 35937).	415
	The "Advanced Boundary Scan Register Settings" section is new (SAR 38432).	417
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
	Table 20-3 • TRST and TCK Pull-Down Recommendations was revised to add VJTAG at 1.2 V.	416
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 20-1 • Flash-Based FPGAs.	414
v1.3 (October 2008)	The "Boundary Scan Support in Low Power Devices" section was revised to include new families and make the information more concise.	415
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 20-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	414
v1.1 (March 2008)	The chapter was updated to include the IGLOO PLUS family and information regarding 15 k gate devices.	N/A
	The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	414

21 – UJTAG Applications in Microsemi's Low Power Flash Devices

Introduction

In Fusion, IGLOO, and ProASIC3 devices, there is bidirectional access from the JTAG port to the core VersaTiles during normal operation of the device (Figure 21-1). User JTAG (UJTAG) is the ability for the design to use the JTAG ports for access to the device for updates, etc. While regular JTAG is used, the UJTAG tiles, located at the southeast area of the die, are directly connected to the JTAG Test Access Port (TAP) Controller in normal operating mode. As a result, all the functional blocks of the device, such as Clock Conditioning Circuits (CCCs) with PLLs, SRAM blocks, embedded FlashROM, flash memory blocks, and I/O tiles, can be reached via the JTAG ports. The UJTAG functionality is available by instantiating the UJTAG macro directly in the source code of a design. Access to the FPGA core VersaTiles from the JTAG ports enables users to implement different applications using the TAP Controller (JTAG port). This document introduces the UJTAG tile functionality and discusses a few application examples. However, the possible applications are not limited to what is presented in this document. UJTAG can serve different purposes in many designs as an elementary or auxiliary part of the design. For detailed usage information, refer to the "Boundary Scan in Low Power Flash Devices" section on page 413.

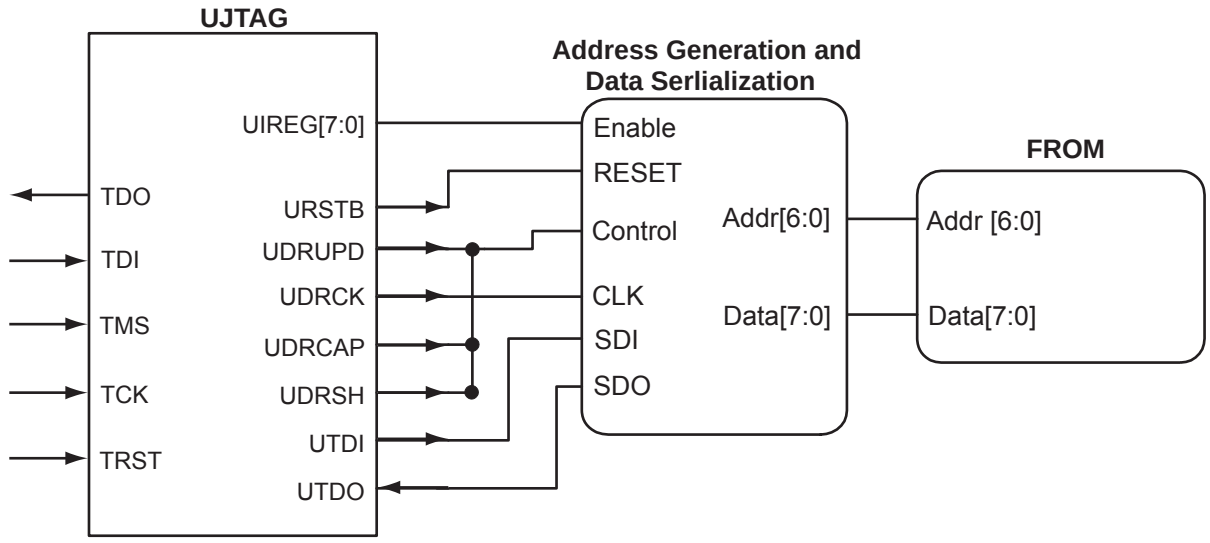


***Figure 21-1* • Block Diagram of Using UJTAG to Read FlashROM Contents**

UJTAG Support in Flash-Based Devices

The flash-based FPGAs listed in Table 21-1 support the UJTAG feature and the functions described in this document.

***Table 21-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 21-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 21-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

UJTAG Macro

The UJTAG tiles can be instantiated in a design using the UJTAG macro from the Fusion, IGLOO, or ProASIC3 macro library. Note that "UJTAG" is a reserved name and cannot be used for any other user-defined blocks. A block symbol of the UJTAG tile macro is presented in Figure 21-2. In this figure, the ports on the left side of the block are connected to the JTAG TAP Controller, and the right-side ports are accessible by the FPGA core VersaTiles. The TDI, TMS, TDO, TCK, and TRST ports of UJTAG are only provided for design simulation purposes and should be treated as external signals in the design netlist. However, these ports must NOT be connected to any I/O buffer in the netlist. Figure 21-3 on page 422 illustrates the correct connection of the UJTAG macro to the user design netlist. Microsemi Designer software will automatically connect these ports to the TAP during place-and-route. Table 21-2 gives the port descriptions for the rest of the UJTAG ports:

Table 21-2 • **UJTAG Port Descriptions**

Port	Description
UIREG [7:0]	This 8-bit bus carries the contents of the JTAG Instruction Register of each device. Instruction Register values 16 to 127 are not reserved and can be employed as user-defined instructions.
URSTB	URSTB is an active-low signal and will be asserted when the TAP Controller is in Test-Logic-Reset mode. URSTB is asserted at power-up, and a power-on reset signal resets the TAP Controller. URSTB will stay asserted until an external TAP access changes the TAP Controller state.
UTDI	This port is directly connected to the TAP's TDI signal.
UTDO	This port is the user TDO output. Inputs to the UTDO port are sent to the TAP TDO output MUX when the IR address is in user range.
UDRSH	Active-high signal enabled in the ShiftDR TAP state
UDRCAP	Active-high signal enabled in the CaptureDR TAP state
UDRCK	This port is directly connected to the TAP's TCK signal.
UDRUPD	Active-high signal enabled in the UpdateDR TAP state

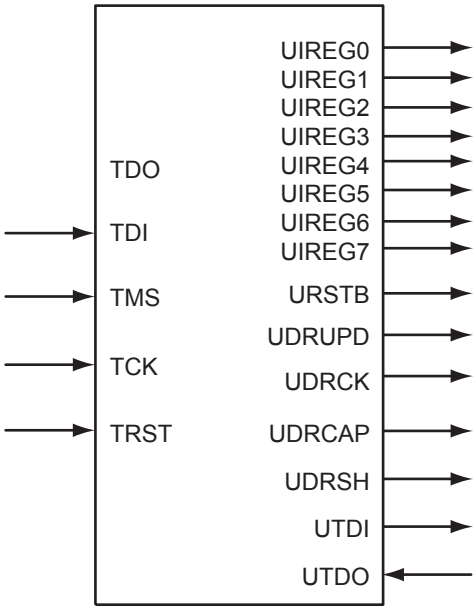
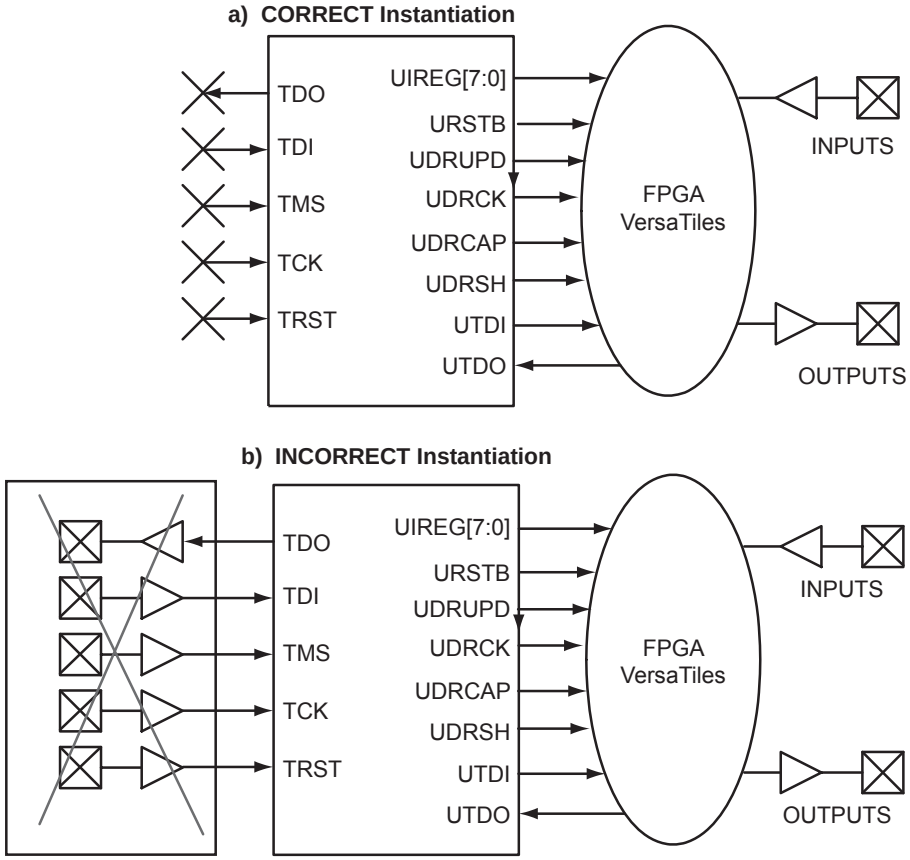


Figure 21-2 • **UJTAG Tile Block Symbol**



Note: Do not connect JTAG pins (TDO, TDI, TMS, TCK, or TRST) to I/Os in the design.

***Figure 21-3* • Connectivity Method of UJTAG Macro**

UJTAG Operation

There are a few basic functions of the UJTAG macro that users must understand before designing with it. The most important fundamental concept of the UJTAG design is its connection with the TAP Controller state machine.

TAP Controller State Machine

The 16 states of the TAP Controller state machine are shown in Figure 21-4 on page 423. The 1s and 0s, shown adjacent to the state transitions, represent the TMS values that must be present at the time of a rising TCK edge for a state transition to occur. In the states that include the letters "IR," the instruction register operates; in the states that contain the letters "DR," the test data register operates. The TAP Controller receives two control inputs, TMS and TCK, and generates control and clock signals for the rest of the test logic.

On power-up (or the assertion of TRST), the TAP Controller enters the Test-Logic-Reset state. To reset the controller from any other state, TMS must be held HIGH for at least five TCK cycles. After reset, the TAP state changes at the rising edge of TCK, based on the value of TMS.

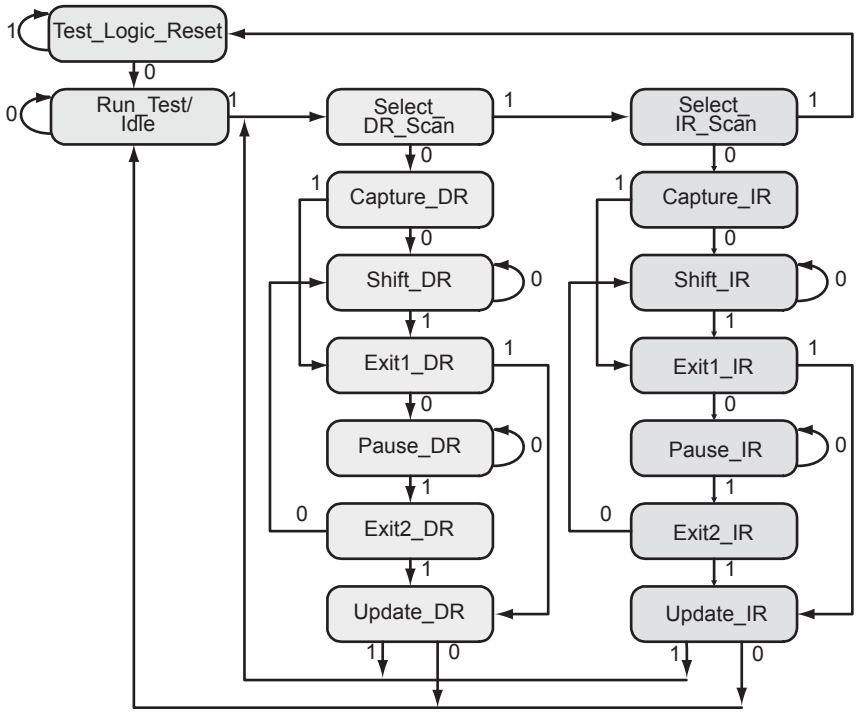


Figure 21-4 • **TAP Controller State Diagram**

UJTAG Port Usage

UIREG[7:0] hold the contents of the JTAG instruction register. The UIREG vector value is updated when the TAP Controller state machine enters the Update_IR state. Instructions 16 to 127 are user-defined and can be employed to encode multiple applications and commands within an application. Loading new instructions into the UIREG vector requires users to send appropriate logic to TMS to put the TAP Controller in a full IR cycle starting from the Select IR_Scan state and ending with the Update_IR state.

UTDI, UTDO, and UDRCK are directly connected to the JTAG TDI, TDO, and TCK ports, respectively. The TDI input can be used to provide either data (TAP Controller in the Shift_DR state) or the new contents of the instruction register (TAP Controller in the Shift_IR state).

UDRSH, UDRUPD, and UDRCAP are HIGH when the TAP Controller state machine is in the Shift_DR, Update_DR, and Capture_DR states, respectively. Therefore, they act as flags to indicate the stages of the data shift process. These flags are useful for applications in which blocks of data are shifted into the design from JTAG pins. For example, an active UDRSH can indicate that UTDI contains the data bitstream, and UDRUPD is a candidate for the end-of-data-stream flag.

As mentioned earlier, users should not connect the TDI, TDO, TCK, TMS, and TRST ports of the UJTAG macro to any port or net of the design netlist. The Designer software will automatically handle the port connection.

Typical UJTAG Applications

Bidirectional access to the JTAG port from VersaTiles—without putting the device into test mode—creates flexibility to implement many different applications. This section describes a few of these. All are based on importing/exporting data through the UJTAG tiles.

Clock Conditioning Circuitry—Dynamic Reconfiguration

In low power flash devices, CCCs, which include PLLs, can be configured dynamically through either an 81-bit embedded shift register or static flash programming switches. These 81 bits control all the characteristics of the CCC: routing MUX architectures, delay values, divider values, etc. Table 21-3 lists the 81 configuration bits in the CCC.

***Table 21-3* • Configuration Bits of Fusion, IGLOO, and ProASIC3 CCC Blocks**

Bit Number(s)	Control Function
80	RESET ENABLE
79	DYNCSEL
78	DYNBSEL
77	DYNASEL
<76:74>	VCOSEL [2:0]
73	STATCSEL
72	STATBSEL
71	STATASEL
<70:66>	DLYC [4:0]
<65:61>	DLYB {4:0]
<60:56>	DLYGLC [4:0]
<55:51>	DLYGLB [4:0]
<50:46>	DLYGLA [4:0]
45	XDLYSEL
<44:40>	FBDLY [4:0]
<39:38>	FBSEL
<37:35>	OCMUX [2:0]
<34:32>	OBMUX [2:0]
<31:29>	OAMUX [2:0]
<28:24>	OCDIV [4:0]
<23:19>	OBDIV [4:0]
<18:14>	OADIV [4:0]
<13:7>	FBDIV [6:0]
<6:0>	FINDIV [6:0]

The embedded 81-bit shift register (for the dynamic configuration of the CCC) is accessible to the VersaTiles, which, in turn, have access to the UJTAG tiles. Therefore, the CCC configuration shift register can receive and load the new configuration data stream from JTAG.

Dynamic reconfiguration eliminates the need to reprogram the device when reconfiguration of the CCC functional blocks is needed. The CCC configuration can be modified while the device continues to operate. Employing the UJTAG core requires the user to design a module to provide the configuration data and control the CCC configuration shift register. In essence, this is a user-designed TAP Controller requiring chip resources.

Similar reconfiguration capability exists in the ProASIC$^{PLUS®}$ family. The only difference is the number of shift register bits controlling the CCC (27 in ProASIC^{PLUS} and 81 in IGLOO, ProASIC3, and Fusion).

Fine Tuning

In some applications, design constants or parameters need to be modified after programming the original design. The tuning process can be done using the UJTAG tile without reprogramming the device with new values. If the parameters or constants of a design are stored in distributed registers or embedded SRAM blocks, the new values can be shifted onto the JTAG TAP Controller pins, replacing the old values. The UJTAG tile is used as the "bridge" for data transfer between the JTAG pins and the FPGA VersaTiles or SRAM logic. Figure 21-5 shows a flow chart example for fine-tuning application steps using the UJTAG tile.

In Figure 21-5, the TMS signal sets the TAP Controller state machine to the appropriate states. The flow mainly consists of two steps: a) shifting the defined instruction and b) shifting the new data. If the target parameter is constantly used in the design, the new data can be shifted into a temporary shift register from UTDI. The UDRSH output of UJTAG can be used as a shift-enable signal, and UDRCK is the shift clock to the shift register. Once the shift process is completed and the TAP Controller state is moved to the Update_DR state, the UDRUPD output of the UJTAG can latch the new parameter value from the temporary register into a permanent location. This avoids any interruption or malfunctioning during the serial shift of the new value.

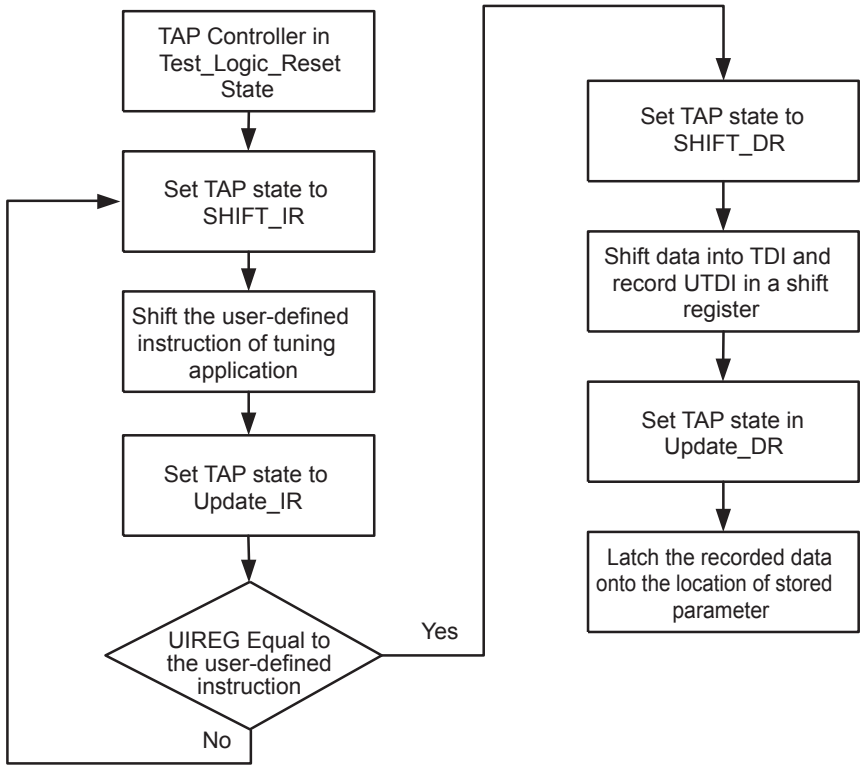


***Figure 21-5* • Flow Chart Example of Fine-Tuning an Application Using UJTAG**

Silicon Testing and Debugging

In many applications, the design needs to be tested, debugged, and verified on real silicon or in the final embedded application. To debug and test the functionality of designs, users may need to monitor some internal logic (or nets) during device operation. The approach of adding design test pins to monitor the critical internal signals has many disadvantages, such as limiting the number of user I/Os. Furthermore, adding external I/Os for test purposes may require additional or dedicated board area for testing and debugging.

The UJTAG tiles of low power flash devices offer a flexible and cost-effective solution for silicon test and debug applications. In this solution, the signals under test are shifted out to the TDO pin of the TAP Controller. The main advantage is that all the test signals are monitored from the TDO pin; no pins or additional board-level resources are required. Figure 21-6 illustrates this technique. Multiple test nets are brought into an internal MUX architecture. The selection of the MUX is done using the contents of the TAP Controller instruction register, where individual instructions (values from 16 to 127) correspond to different signals under test. The selected test signal can be synchronized with the rising or falling edge of TCK (optional) and sent out to UTDO to drive the TDO output of JTAG.

For flash devices, TDO (the output) is configured as low slew and the highest drive strength available in the technology and/or device. Here are some examples:

1. If the device is A3P1000 and VCCI is 3.3 V, TDO will be configured as LVTTL 3.3 V output, 24 mA, low slew.
2. If the device is AGLN020 and VCCI is 1.8 V, TDO will be configured as LVCMOS 1.8 V output, 4 mA, low slew.
3. If the device is AGLE300 and VCCI is 2.5 V, TDO will be configured as LVCMOS 2.5 V output, 24 mA, low slew.

The test and debug procedure is not limited to the example in Figure 21-5 on page 425. Users can customize the debug and test interface to make it appropriate for their applications. For example, multiple test signals can be registered and then sent out through UTDO, each at a different edge of TCK. In other words, *n* signals are sampled with an F_{TCK} / *n* sampling rate. The bandwidth of the information sent out to TDO is always proportional to the frequency of TCK.

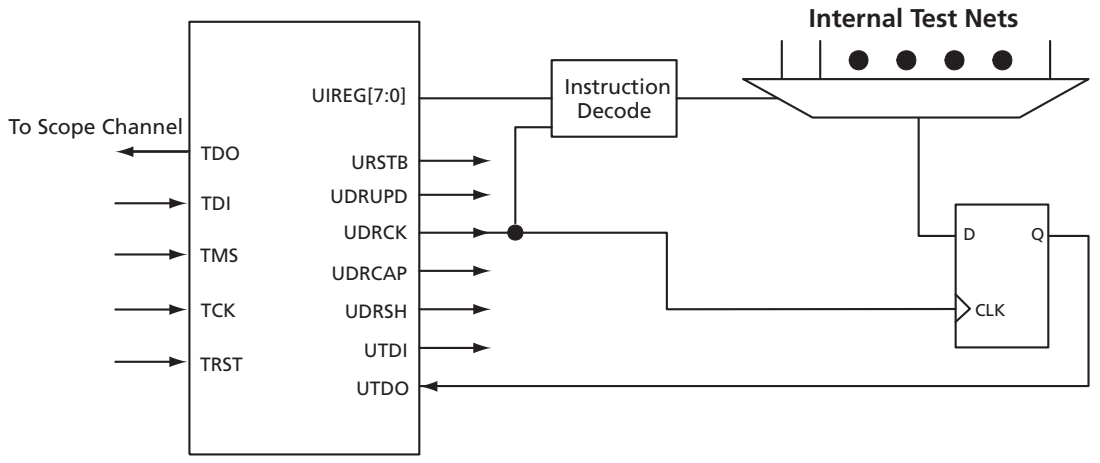


***Figure 21-6* • UJTAG Usage Example in Test and Debug Applications**

SRAM Initialization

Users can also initialize embedded SRAMs of the low power flash devices. The initialization of the embedded SRAM blocks of the design can be done using UJTAG tiles, where the initialization data is imported using the TAP Controller. Similar functionality is available in ProASIC^{PLUS} devices using JTAG. The guidelines for implementation and design examples are given in the *RAM Initialization and ROM Emulation in ProASIC^{PLUS} Devices* application note.

SRAMs are volatile by nature; data is lost in the absence of power. Therefore, the initialization process should be done at each power-up if necessary.

FlashROM Read-Back Using JTAG

The low power flash architecture contains a dedicated nonvolatile FlashROM block, which is formatted into eight 128-bit pages. For more information on FlashROM, refer to the "FlashROM in Microsemi's Low Power Flash Devices" section on page 191. The contents of FlashROM are available to the VersaTiles during normal operation through a read operation. As a result, the UJTAG macro can be used to provide the FlashROM contents to the JTAG port during normal operation. Figure 21-7 illustrates a simple block diagram of using UJTAG to read the contents of FlashROM during normal operation.

The FlashROM read address can be provided from outside the FPGA through the TDI input or can be generated internally using the core logic. In either case, data serialization logic is required (Figure 21-7) and should be designed using the VersaTile core logic. FlashROM contents are read asynchronously in parallel from the flash memory and shifted out in a synchronous serial format to TDO. Shifting the serial data out of the serialization block should be performed while the TAP is in UDRSH mode. The coordination between TCK and the data shift procedure can be done using the TAP state machine by monitoring UDRSH, UDRCAP, and UDRUPD.

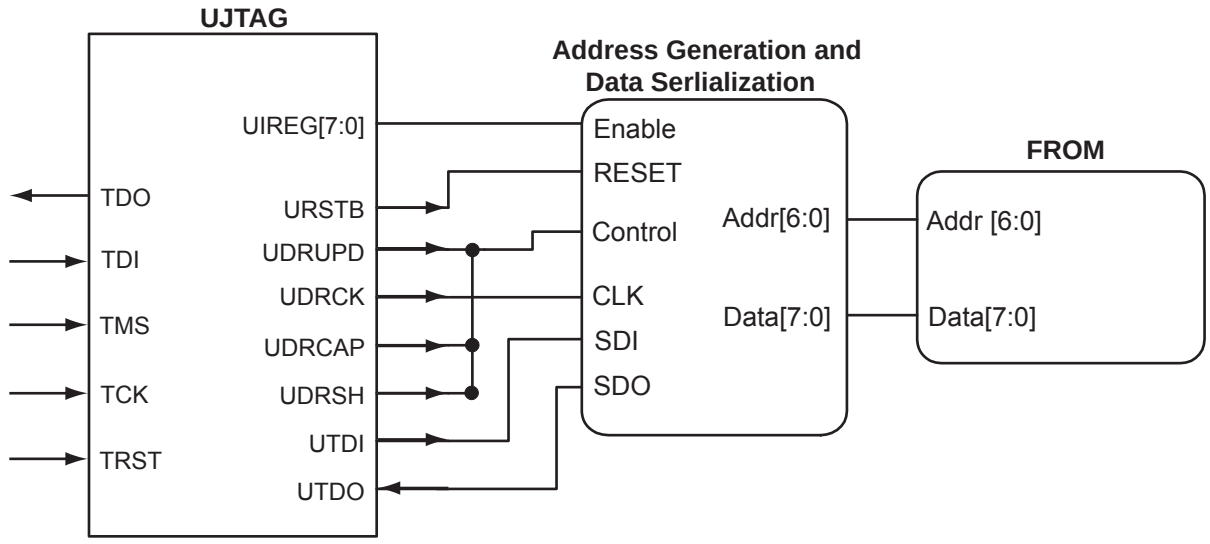


***Figure 21-7* • Block Diagram of Using UJTAG to Read FlashROM Contents**

Conclusion

Microsemi low power flash FPGAs offer many unique advantages, such as security, nonvolatility, reprogrammablity, and low power—all in a single chip. In addition, Fusion, IGLOO, and ProASIC3 devices provide access to the JTAG port from core VersaTiles while the device is in normal operating mode. A wide range of available user-defined JTAG opcodes allows users to implement various types of applications, exploiting this feature of these devices. The connection between the JTAG port and core tiles is implemented through an embedded and hardwired UJTAG tile. A UJTAG tile can be instantiated in designs using the UJTAG library cell. This document presents multiple examples of UJTAG applications, such as dynamic reconfiguration, silicon test and debug, fine-tuning of the design, and RAM initialization. Each of these applications offers many useful advantages.

Related Documents

Application Notes

RAM Initialization and ROM Emulation in ProASIC^{PLUS} Devices

http://www.microsemi.com/soc/documents/APA_RAM_Initd_AN.pdf

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
December 2011	Information on the drive strength and slew rate of TDO pins was added to the "Silicon Testing and Debugging" section (SAR 31749).	426
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 21-1 • Flash-Based FPGAs.	420
v1.3 (October 2008)	The "UJTAG Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	420
	The title of Table 21-3 • Configuration Bits of Fusion, IGLOO, and ProASIC3 CCC Blocks was revised to include Fusion.	424
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 21-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	420
v1.1 (March 2008)	The chapter was updated to include the IGLOO PLUS family and information regarding 15 k gate devices.	N/A
	The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	420

22 – Fusion Board-Level Design Guidelines

Objective

The successful design of Printed Circuit Boards (PCBs) incorporating the Fusion mixed signal FPGA requires good understanding of the mixed signal nature of the Fusion chips. Good board design practices are required to achieve the expected performance from the PCB and Fusion device and are essential to achieve high quality and reliable results, such as minimal noise levels, and adequate isolation between digital and analog domain.

This chapter presents guidelines for board-level design specific to applications using Fusion mixed signal FPGAs. Note that these guidelines should be treated as a supplement to standard board-level design practices. This document assumes readers are experienced in digital and analog board layout and knowledgeable in the electrical characteristics of mixed signal systems. Background information on the key theories and concepts of mixed signal board-level design is available in *High Speed Digital Design: A Handbook of Black Magic*[1], as well as in many reference text books and literature.

Analog and Digital Plane Isolation

Since Fusion is a mixed signal product in which both analog and digital components exist, it requires both analog and digital supply and ground planes. In addition, there are several voltage supply and ground pins on the device to power different components on the die. This section discusses the layout of the different analog or digital planes and recommends schemes to efficiently isolate different digital and analog domains from each other. This section also describes all ground and supply pins of the Fusion device, required to operate the chip, and explains how to connect them to the existing digital or analog supply or ground planes.

Placement of Fusion Device and Isolation of Ground Planes

In applications using Fusion devices, two separate grounds to the device should be provided: GND (digital ground) and GNDA (analog ground). The ground pins of the device are to be connected to one of the aforementioned ground planes appropriately, as discussed in "Isolation of Ground Planes" on page 431. GND is the digital ground plane that connects to all GND pins of a Fusion device. GNDA is the analog ground plane that connects to all GNDA pins of a Fusion device.

To avoid noise propagation from one plane to another (e.g. from digital to analog ground), the ground planes should be well isolated from each other. Correct layout of the ground planes on the board for current and return paths in the board will prevent the noise in one plane to affect others. For example if the return path of a digital signal trace on the board passes through the ground analog plane, the analog ground will be vulnerable to noise induced by the digital signal. Therefore, it is critical for digital traces and components on the board to be routed and placed only in the area of their corresponding layer that is covered by digital ground in the ground plane. Similar regulation should be applied to analog traces and components with respect to the analog ground as well. Figure 22-1 on page 430 illustrates the aforementioned regulation.

In Figure 22-1 on page 430, digital component C and the traces that connect to it overlap with the analog ground layout in the ground plane. This may cause some of the digital signaling current and return paths to pass through the analog domain and induce noise in this noise-sensitive domain.

Figure 22-1 on page 430 brings up a critical point: How a mixed signal device, such as a Fusion device, should be placed on the board.

1. *Johnson, Howard, and Martin Graham, High Speed Digital Design: A Handbook of Black Magic. Prentice Hall PTR, 1993. ISBN-10 0133957241 or ISBN-13: 978-0133957242*

Placement of Fusion Device on Board

Fusion devices contain both analog and digital components and can interface with other digital and analog components on the board.

A Fusion device should be placed on boards such that analog signaling of the system falls within the boundaries of the analog ground and supply domain. Similarly, digital signaling of the system should fall within the boundaries of the digital domain. Figure 22-2 shows a simple illustration of the placement of a Fusion device on the board.

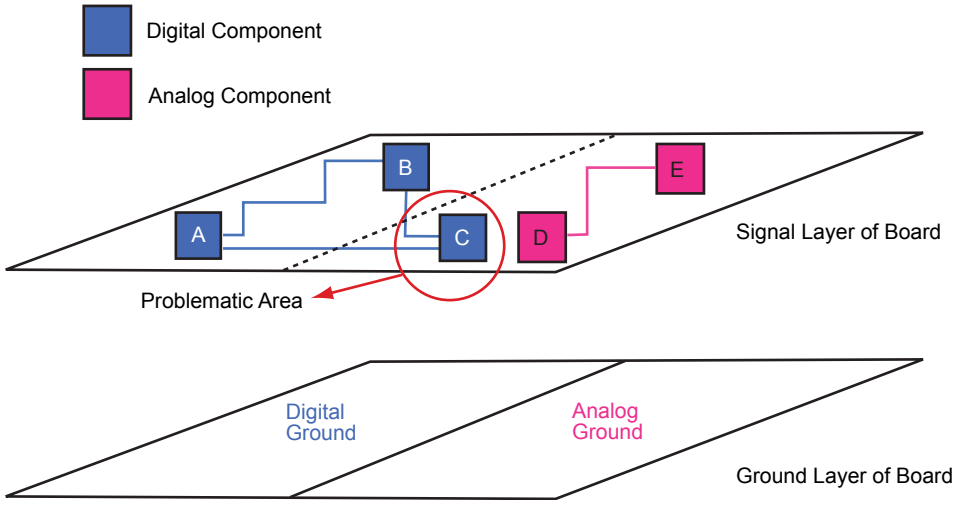


***Figure 22-1* • Illustration of Analog and Digital Components Placement on Boards**

As shown in Figure 22-2, the Fusion chip is placed on the boundary of analog and digital domains, so that the analog pins of the Fusion device are within the analog ground domain and the digital portion of the chip is placed within the digital ground domain.

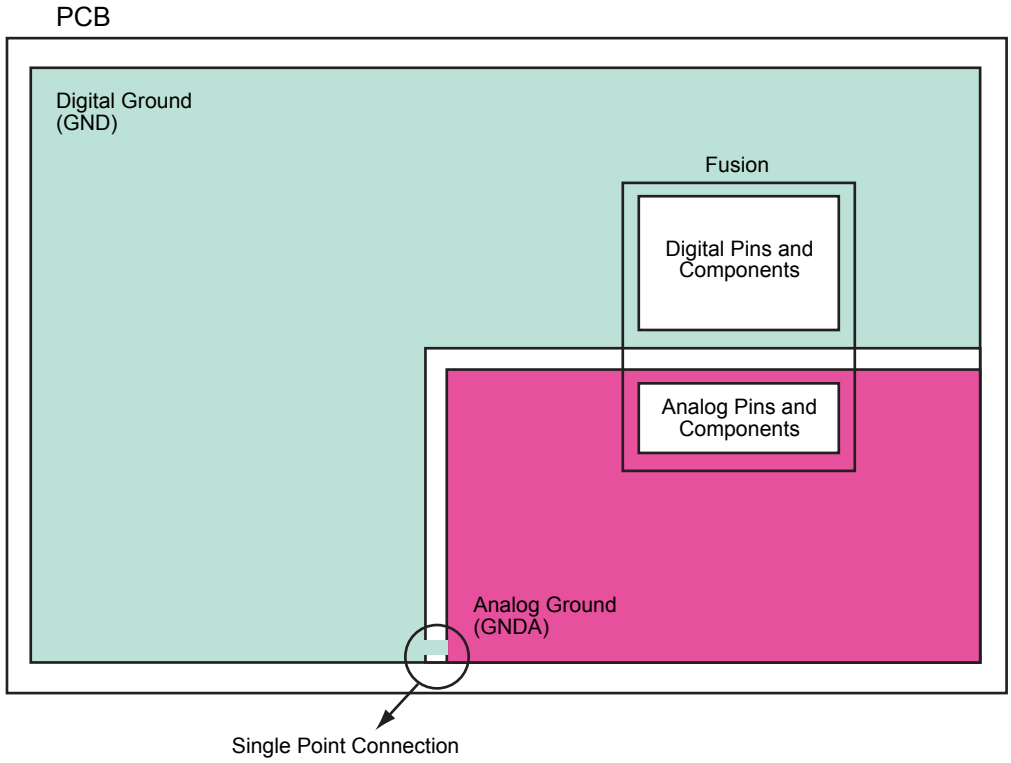


***Figure 22-2* • Simple Illustration of Fusion Device Placement on Boards**

In complicated system designs and more complicated device packages, the placement of a Fusion device may not be as straight forward as shown in the simplified diagram of Figure 22-2. However, in any board layout, it is critical to keep digital signals and their return paths well isolated from the analog domain. The "Isolation of Ground Planes" section discusses an example of Fusion placement and ground plane layout in a real-world mixed signal system design.

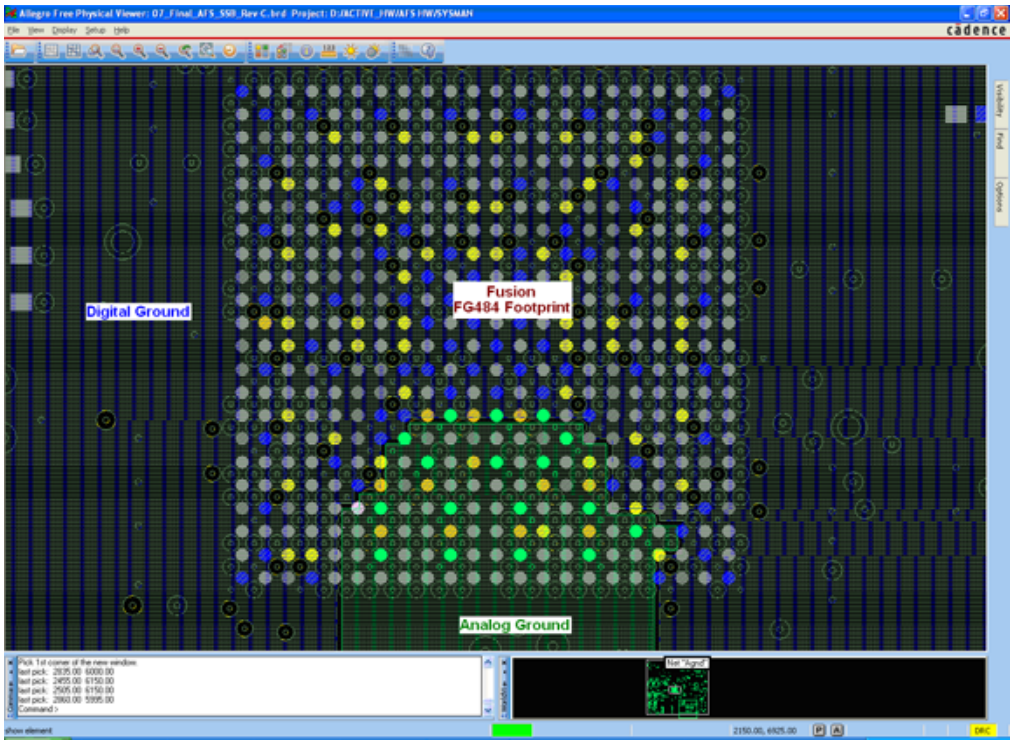
Figure 22-2 on page 431 also shows that the analog and digital grounds are to be connected to each other at a single point. The layout of the ground planes, as well as the power supply planes, plays a key role in reducing the noise and hence enhancing the performance and accuracy of the system.

Isolation of Ground Planes

As mentioned in "Placement of Fusion Device and Isolation of Ground Planes" on page 429, the ground and supply planes should be divided in two main domains: digital (e.g. GND) and analog (e.g. GNDA). Though there is no technical limitation in implementing more ground and supply domains for other necessary ground and supply pins of a Fusion device, the rest of the ground and supply pins can be connected to one of the aforementioned domains.

In order to isolate the ground of different domains from each other, the ground plane (or planes) of the board should be split into two domains: GND and GNDA, as an example. The components and signaling in each domain should remain within the boundaries of each ground as discussed in "Placement of Fusion Device and Isolation of Ground Planes" on page 429 and "Placement of Fusion Device on Board" on page 430. However, since data and control signals usually exchange between different domains, a common connection between analog and digital grounds is needed to ensure the two planes are at the same potential. Connection between two grounds should be made only through a single point as shown Figure 22-2 on page 431. More than a single connection point between two grounds can result in inter-domain current paths that can induce noise from one domain to another. Furthermore, the single point connection should be as far as possible from the Fusion device.

Figure 22-3 shows a real-world example of a ground plane layout and the relative placement of the Fusion chip. Refer to the "Analog and Digital Plane Isolation" section on page 429 for board layout recommendations.



Note: Blue = GND, Yellow = VCCI, and Green = GNDA

Figure 22-3 • **Example of Ground Plane Layout and Fusion Device Placement**

Other ground pins of the Fusion device can connect to one of the two grounds using traces on the board if necessary. However, the length of the traces should be kept as short as possible to reduce the trace inductance between ground pins and the ground plane. Table 22-1 lists all the ground pins of a Fusion device and the ground plane that they connect to.

Table 22-1 • **Ground Pin Connections to Ground Plate on Board**

Ground Pin Name	Ground Domain
GND	Digital(GND)
GNDQ	Digital(GND)
ADCGNDREF	Analog(GNDA)
GNDA	Analog(GNDA)
GNDAQ	Analog(GNDA)
GNDNVM	Digital(GND)
GNDOSC	Digital(GND)
VCOMPPLA/B*	Digital(GND)

*Note: *Older revisions of datasheets referred to a signal called VCOMPLF. This is now called VCOMPLA/B.*

Analog and Digital Voltage Supply Isolation

Digital and analog voltage supplies should be isolated from each other similar to the grounds as discussed in "Placement of Fusion Device and Isolation of Ground Planes" on page 429. There are three main power supplies to Fusion devices: VCC33A (3.3 V analog supply), VCC (1.5 V digital core supply), and VCCI (digital I/O supply). There may be multiple VCCI levels (for digital I/Os) since Fusion devices offer multiple I/O banks. Regardless of the number of power supply voltage levels, the layout of the board's power plans should conform to the same specifications as recommended for the ground plane in "Placement of Fusion Device and Isolation of Ground Planes" on page 429. None of the digital power domains should overlap with the analog power supply domain (VCC33A). This ensures that digital signaling and its return paths are well isolated form the analog power supply, minimizing noise in the analog domain. Figure 22-4 on page 433 shows a simple illustration of mixed signal board layers and relative layout of the digital and analog domains.

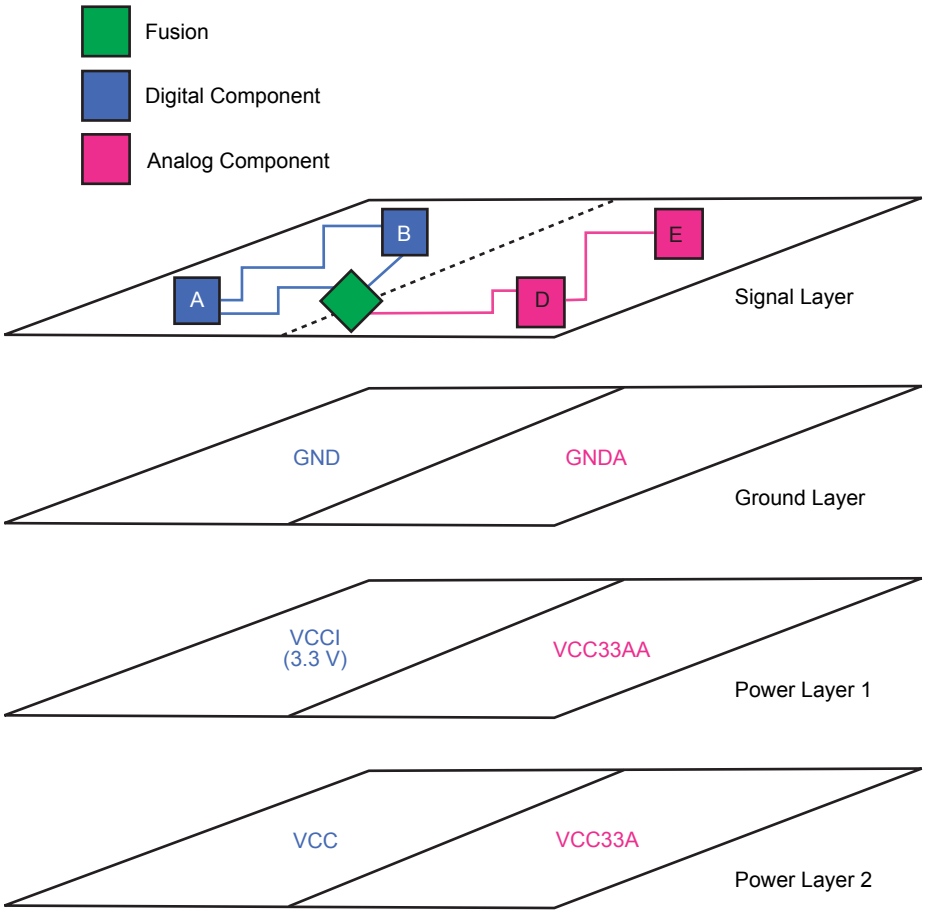


***Figure 22-4* • Simplified Illustration of a Mixed Signal Board Stack Up**

As shown in Figure 22-4, no digital grounds or voltage supplies overlap with the analog domain. The ground plane is divided into two domains of analog and digital ground. The power planes in the Figure 22-4 board stack up follow the same layout as the ground plane. The Fusion device is placed on the boundary of the digital and analog domains as recommended in "Placement of Fusion Device and Isolation of Ground Planes" on page 429. Digital planes may be split if needed to accommodate additional supplies. For example, the VCCI plane can be split into 3.3 V and 2.5 V planes. The addition of another plane just to support the additional supply is typically not needed.

Additionally, Figure 22-4 emphasizes the layout of the signal traces in the signal layers of the board stack up. The digital signal traces are laid out within the digital domain and the analog traces are contained within the analog area of the layer.

Other power pins of the Fusion device can connect to one of the two domains using traces on the board. However, the length of the traces should be kept as short as possible to reduce the trace inductance between power pins and the power plane, induced by board traces, to a minimum. Figure 22-2 on page 431 shows that the analog and digital grounds are to be connected to each other at a single point. The same technique should be applied to digital and analog 3.3 V supplies.

Table 22-2 lists all the power pins of a Fusion device and the power plane that they connect to.

***Table 22-2* • Power Pin Connections to Power Plane on Board**

Supply Pin Name	Supply Domain
VCC	Digital (VCC)
VCC15A	Digital (VCC)
VCC33A	Analog (VCC33A)
VCC33PMP	Analog (VCC33A)
VCCNVM	Digital (VCC)
VCCOSC	Digital (3.3V)[1]
VCCIBx	Digital (VCCI)[2]
VCCPLA/B	Digital (VCC through recommended capacitors)[3]

Notes:

1. *Can be tied to any digital 3.3 V rail available in the application board (e.g. VCCIBx if the bank requires a 3.3 V supply)*
2. *If multiple banks are powered with different supply levels, different VCCI planes are needed for each voltage level*
3. *Capacitor recommendations for VCCPLA/B pins are similar to those for the ProASIC3 family device and can be found in the "Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" section on page 53.*

Similar to any other board-level designs, decoupling/bypass capacitors or other power supply filtering techniques should be used between power supply pins and ground to reduce any potential fluctuation on the supply lines. The *Board-Level Considerations* application note provides additional recommendations on using decoupling capacitors. There are numerous other industry publications and guidelines available on the subject.

Other Special Function Pins

In addition to the general power and ground pins discussed in "Analog and Digital Plane Isolation" on page 429, there are a few other special pins that require special board considerations to ensure proper functionality of the Fusion device. This section of the document lists these pins and describes their connectivity in the board-level design.

VAREF

This pin is the voltage reference for Fusion's analog to digital converter (ADC). The Fusion device can provide a 2.56 V internal reference voltage. While using the internal reference, the reference voltage is output on VAREF for use as a system reference. If a different reference voltage is required, it can be externally supplied to VAREF and used by the ADC.

Since VAREF is the reference voltage for the ADC, it is critical for VAREF (either internal or external) to be very clean. Noise on VAREF affects the accuracy of the ADC and may cause the analog system to operate outside the specification listed in the *Fusion Family of Mixed Signal Flash FPGAs* datasheet. For the internal VAREF use model, Microsemi recommends an external capacitor be placed between VAREF and the ADCGNDREF, as close as possible to the VAREF pin. Microsemi recommends the capacitor value to be between 3.3 µF to 22 µF. High capacitor values (up to 22 µF) result in better noise filtering

and higher ADC accuracy. However, the larger the capacitor value, the longer the rise time of VAREF at power-up. Longer time of VAREF will in turn delay the power-up time to functional of the analog block. Smaller capacitor values cause faster power-up time for VAREF, but noise filtering will be relatively less. The choice of capacitor values also depends on the total amount of noise existing on the user's board. Boards with relatively higher noise levels may need to have capacitor values close to 22 µF and a VAREF pin, and may not perform to expectation if the capacitor values are close to 3.3 µF.

When VAREF is provided by an external source, the source must be clean to ensure the highest accuracy of the ADC. Refer to the *Fusion Family of Mixed Signal Flash FPGAs* datasheet and the "Interfacing with the Fusion Analog System: Processor/Microcontroller Interface" section on page 253 for more information on selecting the right capacitor value for internally generated VAREF.

VCC33N, PCAP, and NCAP

These three pins are associated with the -3.3 V charge pump. This charge pump uses two external capacitors in order to generate the -3.3 V supply. One capacitor is connected between the NCAP and PCAP pins, while the other is connected between VCC33N and the analog ground. The impulse charging of the capacitors, while the charge pump is in operation, is a source of electromagnetic interference (EMI). To reduce EMI, each of these capacitors consist of a 0.1 µF ceramic capacitor in parallel with a tantalum capacitor. The ceramic capacitors should be mounted as close as possible to the pins, using capacitors of small physical size. For the BGA package, these capacitors are to be mounted on the bottom layer, directly underneath the respective pins. The tantalum capacitors can be mounted a little further off, but users should try to minimize the distance. Ceramic capacitors are also available in higher values such as 2.2 µF. If such a capacitor is used, the 0.1 µF capacitor might not be needed.

XTAL1 and XTAL2

These pins are input from external oscillators to Fusion devices. Very slow rise and fall times of typical oscillator output (input to XTAL1 and XTAL2 pins) are much more prone to any noise induced by the system, and can result in the oscillator frequency to be misinterpreted by the Fusion device. Typical crystal oscillators generating low frequency signals, such as 32.768 KHz, typically have very slow rise and fall times (sinusoidal signal). Any small noise on the generated sine wave can result in misinterpretation of the frequency of the sine wave for the Fusion device and affect the correct functionality of the design. Therefore, for very low frequency signals, such as 32.768 KHz input to XTAL inputs of the Fusion device, Microsemi recommends users utilize a digital oscillator (fast rise and fall times) or to use Schmitt trigger buffers to shorten the rise and fall time of typical oscillators. For signals at 1 MHz and above, since the rise and fall times are inherently fast, a typical analog crystal oscillator can be used with no specific precaution.

For the layout and connection of the external crystal and the associated capacitors, keep stray capacitance and inductance to a minimum. It is important to keep any noise from coupling to the on-chip oscillator by way of the power supply, the crystal, the two load capacitors, or the copper traces used to connect these components. It is also important to prevent noise from coupling from the oscillator into the analog power supply, affecting the performance and accuracy of other analog circuitry. The following guidelines help achieve these objectives:

- The oscillator power supply pins should be decoupled by a 0.1 µF capacitor connected as close as possible between the VCCOSC and GNDOSC pins of the Fusion device. For a BGA package, this capacitor can be placed on the bottom layer; and if it is size 402, it fits between the pins.
- The crystal should be placed as close as possible to the XTAL1 and XTAL2 pins.
- The spacing between traces connecting crystal to XTAL1 and XTAL2 pins and nearby traces should be increased beyond the minimum spacing dictated by the PCB design rules to prevent any noise from coupling into these traces. In addition, copper traces carrying high speed digital signals should not be routed in parallel to the copper traces connected to the XTAL1 and XTAL2 pins, either on the same layer or on the other layers.
- To reduce electromagnetic emissions and provide good mechanical stability to the crystal, a copper pad slightly larger than the crystal and grounded to GNDOSC should be placed on the top layer of the PCB. The metal package of the crystal should be grounded to this pad with a suitable clip. Copper traces connected to this grounded pad and extending around the copper traces leading from the crystal to XTAL1 and XTAL2 pins shield these pins and further increase noise

immunity of the oscillator. The shields add a very small amount of stray capacitance and this can be accounted for in the selection of the load capacitors.

Application-Specific Recommendations

This section of the document discusses some recommendations that are specific to temperature, voltage, or current monitoring applications. These recommendations are merely for improving the accuracy of the applications.

Temperature Monitor

The temperature monitor generates a voltage of about 2.5 mV/K (per degree Kelvin), as seen by the ADC. However, the voltage change that appears across the external discreet bipolar transistor may be much smaller. Such low levels mean that precautions should be taken to not couple noisy signals to the conductors connecting the transistor to the temperature monitor pins.

If the temperature sensing the diode/transistor is connected to a Fusion device through cables, Microsemi recommends using a twin lead shielded cable to carry the AT and ATRTN traces with the shield of the cable grounded at the board.

If the connections are made by copper traces on the PCB, AT and ATRTN traces should be routed in such a way that traces carrying digital signal are not parallel to them above, below, or on the sides. To achieve this, lay the AT and ATRTN traces on the top layer, so that the next adjacent layer in the PCB stack is the ground layer. This provides for shielding against digital signals that can couple to the signals on the copper traces connected to the AT and ATRTN pins.

If digital signal carrying traces can not be avoided in the vicinity of the traces connecting to the transistor, sufficient distance is to be created between the offending trace and the AT or ATRTN traces.

It is important to minimize the resistance of the conductors connecting the external discrete bipolar transistor to the AT and ATRTN pins of the Fusion chip. If PCB copper traces are used as the interconnecting conductors, they should be of such a width that, taking into account their length, they contribute only a negligible voltage drop compared to 200 µV. The current through the bipolar transistor used for sensing the temperature changes by 90 µA during the measurement process. This current, multiplied by the total resistance of the copper trace from the AT pin to the transistor and from the transistor back to the ATRTN pin, should be negligible compared to 200 µV. If a shielded cable is used, the wire gauge of its conductors should be appropriately selected. If the system using the Fusion devices is to be operated at other than room temperature, the effect of temperature on the resistance of the wire or copper traces should also be taken into account.

Voltage and Current Monitor

If any of the AV channels are used in the direct mode that is directly connecting to the ADC without prescalers, it is recommend that a ceramic capacitor of the NPO or COG variety, or better yet, a polyester capacitor of 2200 pF be placed from the corresponding AV channel pin to the analog ground, and as close as possible to the AV pin.

A resistor of 100 Ω should then be connected between the AV pin and whatever point is being monitored by the particular AV channel. If the accuracy requirements are not stringent, one may be able to get by without using the above mentioned resistor/capacitor combination. However, it is good practice to at least make provision for these components on the prototype PCB. The ADC is a switched capacitor design and needs to be driven from a low impedance. It draws a charging current every time a channel is sampled, and the capacitor helps to maintain the voltage steady at the particular AV pin during such intervals. All copper traces connecting to the AV or the AC pins should stay within the area covered by the analog ground plane. The power for the ADC, voltage and current monitors, and the internal voltage reference is provided from the same pins. These pins are to be adequately decoupled with 0.1 µF ceramic X7R dielectric capacitors in parallel with a tantalum capacitor of 22 µF capacity.

In applications using current monitor, it is important to route the AV and AC signals of each channel in parallel and keep the two traces matched as much as possible. Large differences in the nets bringing AV and AC signals to the device may cause significant inaccuracy in differential voltage across the AV and AC pin.

In current monitor applications, the current sense resistor should be chosen carefully so that optimal accuracy and resolution can be achieved. The *Fusion Family of Mixed Signal Flash FPGAs* datasheet describes the recommended resistor values for various current ranges.

Voltage Regulator Power Supply Monitor (VRPSM)

The on-chip voltage regulator generates a 1.5 V output from a 3.3 V external power supply. The 1.5 V output is intended to supply power to all 1.5 V resources in the Fusion device and must be routed externally to the appropriate VCC pads. When using the internal 1.5 V regulator, the TRST pad must be pulled down using a 470 Ω resister to ensure proper functionality.

Connection to PLL

Table 22-1 on page 432 and Table 22-2 on page 434 describe the connection of the VCCPLA/B and VCOMPLA/B pins of the Fusion device to the power and ground planes. This section of the document discusses how these pins and the dedicated clock pins of the Fusion device connect to the PLLs on the chip. Connecting external signals into PLL and powering them up should be done considering that AFS090 and AFS250 devices contain only one PLL, while AFS600 and AFS1500 devices contain two PLL blocks. In AFS090 and AFS250 devices, the PLL is located on the west side of the die. In devices with two PLLs, the second PLL is placed on east side of the die[1].

Table 22-3 shows the corresponding power and ground pins for each PLL block

***Table 22-3* • Power and Ground Pin Names For Fusion Device PLLs**

PLL/Device	AFS090	AFS250	AFS600	AFS1500
West PLL	VCCPLA/VCOMPLA	VCCPLA/VCOMPLA	VCCPLA/VCOMPLA	VCCPLA/VCOMPLA
East PLL	–	–	VCCPLB/VCOMPLB	VCCPLB/VCOMPLB

In addition to hardwire clock pins, Fusion device PLLs can be driven by any internal net or external I/O pins. Although the hardwire I/Os can be used as any user I/O, if designers are required to minimize the propagation from external clock to the PLL, hardwire clock pins of the PLL provide the shortest paths from board to PLL clock input. Table 22-4 lists the hardwire clock pins for each PLL on the device.

***Table 22-4* • Hardwire Clock Pin Connections to PLL**

PLL/Device	AFS090	AFS250	AFS600	AFS1500
West PLL	GFA0/GFA1/GFA2*	GFA0/GFA1/GFA2*	GFA0/GFA1/GFA2	GFA0/GFA1/GFA2
East PLL	–	–	GCA0/GCA1/GCA2	GCA0/GCA1/GCA2

*Note: * Depending on the selected package, not all three hardwire clock I/Os may be available.*

1. *Refer to the Fusion Family of Mixed Signal Flash FPGAs datasheet for more information on Clock Conditioning Circuits and location of PLLs.*

List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
August 2012	The "VAREF" section was updated to change the recommendation for the internal VAREF use model to putting external capacitor between VAREF and ADCGNDREF rather than between VAREF and the analog ground (SAR 36725).	434
December 2011	The "Voltage Regulator Power Supply Monitor (VRPSM)" section is new (SAR 21829).	437
July 2010	This chapter is no longer published separately with its own part number and version but is now part of the Fusion FPGA Fabric User's Guide.	N/A
v1.1 (November 2008)	The "Temperature Monitor" section was revised to remove information about capacitors.	436

23 – Power-Up/-Down of Fusion FPGAs

Introduction

The Fusion mixed signal FPGA satisfies the demand from system architects for a device that simplifies design and unleashes their creativity. As the world's first mixed signal FPGA family, Fusion integrates configurable analog, flash memory, and FPGA fabric in a monolithic mixed signal FPGA. Fusion devices enable designers to quickly move from concept to completed design and deliver feature-rich systems to market. This new technology takes advantage of the unique properties of flash-based FPGAs, including a high isolation, triple-well process and the ability to support high voltage transistors to meet the demanding requirements of mixed signal system design.

As a flash-based FPGA, a Fusion device is nonvolatile, offering a single-chip solution. Being live at power-up and capable of operating from a single 3.3 V power supply make Fusion an ideal system master for initiating, controlling, and monitoring multiple voltage supplies. An on-chip voltage regulator can be used to provide a 1.5 V power source to the FPGA fabric portion of Fusion and embedded flash memory.

Glitches and brownouts in system power will not corrupt the Fusion device logic configuration. Fusion devices have no power-on surge, inrush current, or configuration current, as do SRAM-based FPGAs. Fusion devices offer additional power savings with both sleep and standby modes of operation.

Power Supply Requirements

The main power supplies for digital and analog components of Fusion devices are given in Table 23-1.

***Table 23-1* • Main Power Supplies of Fusion Devices**

Voltage	Power Supplies	Notes
1.5 V	Core supply voltage (VCC)	Supply voltage to the FPGA core, nominal 1.5 V. VCC is also required for powering the JTAG state machine in addition to VJTAG. Even when a Fusion device is in bypass mode in a JTAG chain of interconnected devices, both VCC and VJTAG must remain powered to allow JTAG signals to pass through the Fusion device.
	Embedded flash power supply (VCCNVM)	1.5 V power supply used by the Fusion device's flash memory block module(s).
	PLL supply voltage (VCCPLA/B)	Supply voltage to analog PLL, nominal 1.5 V. If unused, VCCPLA/B should be tied to GND.
	Analog power supply (VCC15A)	1.5 V clean analog power supply input for use by the ADC, AQ, and ACM.
3.3 V	I/O supply voltage (VCCIBx)	Supply voltage to the bank's I/O output buffers and I/O logic. VCCI can be 1.5 V, 1.8 V, 2.5 V, or 3.3 V nominal voltage. Unused I/O banks should have their corresponding V_{CCI} pins tied to GND.
	JTAG supply voltage (VJTAG)	The JTAG pins can be run at any voltage from 1.5 V to 3.3 V (nominal). If the JTAG interface is not used nor planned to be used, the VJTAG pin together with the TRST pin could be tied to GND.
3.3 V (cont'd)	Oscillator power supply (VCCOSC)	Power supply for both integrated RC oscillator and crystal oscillator circuit

***Table 23-1* • Main Power Supplies of Fusion Devices (continued)**

Voltage	Power Supplies	Notes
	Analog power supply (VCC33A)	3.3 V clean analog power supply input for use by the ADC and RTC
	Negative 3.3 V output (VCC33N)	–3.3 V output from the voltage converter. A 2.2 µF capacitor must be connected from this pin to ground
	Analog power supply (VCC33PMP)	3.3 V clean analog power supply input for use by the analog charge pump
	Programming supply voltage (VPUMP)	For programming, VPUMP should be 3.3 V nominal. During normal device operation, VPUMP can be left floating or can be tied (pulled up) to any voltage between 0 V and 3.6 V.

Table 23-2 shows which power supplies are required for the operation of major functional blocks inside the Fusion device.

***Table 23-2* • Required Power Supplies for Operation of Fusion Blocks**

Fusion Blocks	3.3 V Supply Only	3.3 V and 1.5 V Supply
Fabric	Off	On
JTAG	Off	On
RCO/XTALOSC	On	On
I/O Banks	On	On
Embedded flash memory (NVM)	Off	On
Real-time counter (RTC)	On	On
Analog-to-digital converter (ADC)	Reset state	On
Analog Quad (AQ)	Off	On
Analog configuration multiplexer (ACM)	Off	On

Fusion devices can operate from a single 3.3 V supply by utilizing the Fusion integrated 1.5 V voltage regulator and an external pass transistor. The 1.5 V will drive the FPGA core and embedded flash memory. The 1.5 V is not supplied to the device internally, but is routed on the board to the appropriate power supply pins on the Fusion device. This allows users the flexibility to either implement this power scheme or utilize other 1.5 V supplies that may already exist within their system.

Power-Up/Down Sequence and Transient Current

Fusion devices have no power-up or power-down sequencing requirements. To maximize power savings, Microsemi identifies three power conditions that will result in relatively high transient current:

1. When VCC33PMP is powered up before VCC33A, there will be approximately 30 mA transient current on VCC33PMP until VCC33A is powered up. In order to avoid this increase in current, Microsemi recommends that the 3.3 V supplies, VCC33A and VCC33PMP, be tied together.
2. Microsemi recommends tying VCCNVM to VCC.
    a. If VCCNVM is powered up before VCC, a static current of 30 mA (typical) can be measured on VCCNVM.
        i. The current vanishes as soon as VCC reaches VCCNVM voltage level.
        ii. The same current is observed at power-down (VCC before VCCNVM).
    b. If VCC is powered up before VCCNVM, a static current of 30 mA (typical) can be measured on VCC.
        i. The current vanishes as soon as VCCNVM reaches VCC voltage level.
        ii. The same current is observed at power-down (VCCNVM before VCC).
    c. If VCCNVM is powered up simultaneously with VCC, the maximum transient current on VCC does not exceed the maximum standby current specified in the datasheet.
3. Microsemi recommends tying VCCPLX to VCC and using proper filtering circuits to decouple VCC noise from the PLL.
    a. If VCCPLX is powered up before VCC, a static current of up to 5 mA (typical) per PLL can be measured on VCCPLX.
        i. The current vanishes as soon as VCC reaches VCCPLX voltage level.
        ii. The same current is observed at power-down (VCC before VCCPLX).
    b. If VCCPLX is powered up simultaneously or after VCC, the maximum transient current on VCC does not exceed the maximum standby current specified in the datasheet.

In any other power-up/down sequence, no transient current is observed (i.e., the transient current during power-up remains less than the quiescent current of the Fusion device).

Digital I/O Behavior at Power-Up/Down

This section discusses the behavior of device I/Os, used and unused, during power-up/down of VCC and VCCI.

The voltage pins used for Fusion I/O activation are as follows:

- VCC: Voltage supply to the FPGA core
- VCCIBx: Supply voltage to the bank's I/O output buffers and I/O logic. Bx is the I/O bank number.

In Fusion devices, the JTAG I/O bank is powered separately and need only be powered when JTAG operations are desired. When implementing JTAG commands, both the FPGA core and the JTAG I/O bank must be powered.

I/O State During Power-Up/Down

During power-up of the Fusion device, all I/Os are in tristate mode. Only after both VCC and VCCI power supplies are powered up to their functional level are the I/Os configured to the desired user configuration (power supply functional levels are discussed in the "Power-Up to Functional Time" section on page 23-442). Similarly, during power-down, I/Os of each bank are tristated after the first power supply reaches its brownout deactivation voltage.

I/O State During Standby and Sleep Modes

During standby and sleep modes of operation, VCC (1.5 V for the FPGA core) is off, VCC33A (3.3 V supply to the voltage regulator) is on, and VCCI (I/O power rails) is on or off. The state of each type of I/O is defined as follows:

- Analog inputs
  - Inputs are pulled-down to ground through a 1 M Ω resistor.
- Gate driver
  - Output is floating (or tristated). No extra current on VCC33A.
- Digital inputs (with or without pull-up/down)
  - Inputs are tristated.
- Digital outputs
  - Outputs are tristated.
- Digital Bibufs (input/output)
  - Outputs and inputs are tristated.

Power-Up to Functional Time

At power-up, device I/Os exit the tristate mode and become functional when the last voltage supply in the power-up sequence (VCCI or VCC) reaches its functional activation level.

Fusion devices meet Level 0 live at power-up (LAPU); they can be functional prior to VCC reaching the regulated voltage required. This important advantage distinguishes Fusion flash devices from their SRAM-based counter parts. SRAM-based FPGAs, due to their volatile technology, require hundreds of milliseconds after power-up to configure the design bitstream before they become functional (Figure 23-1).

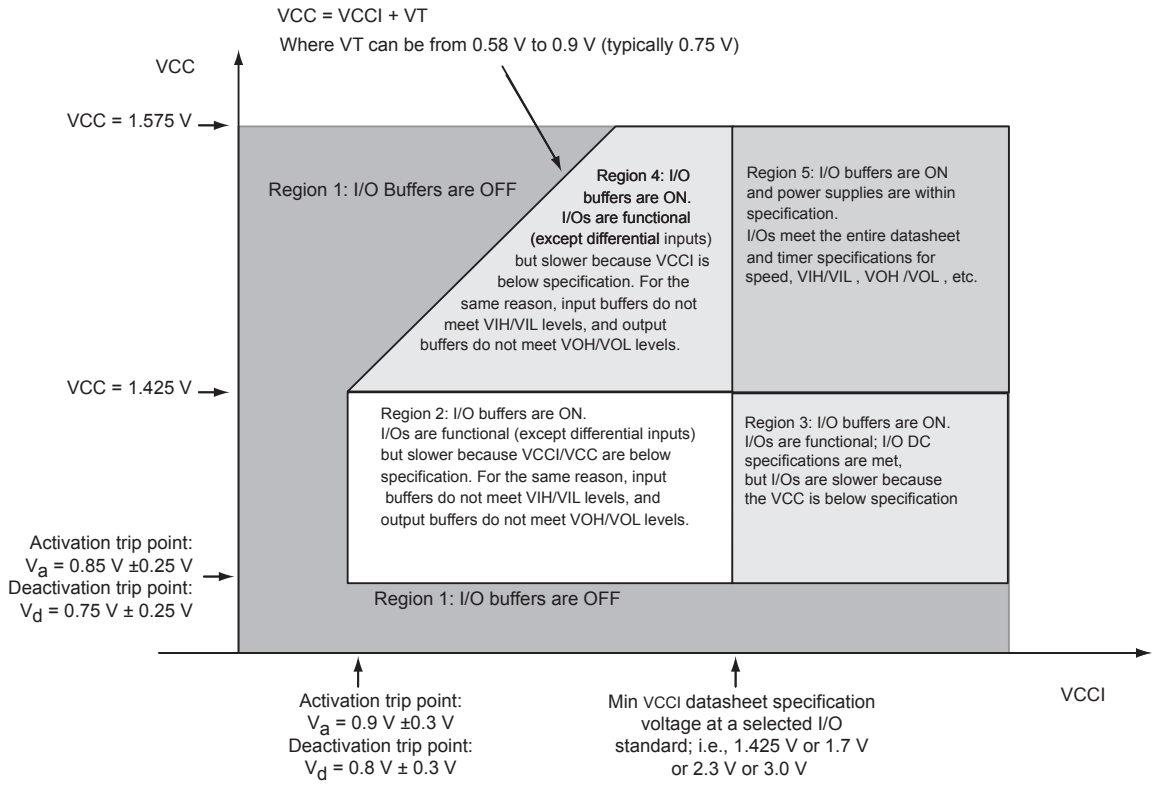


***Figure 23-1* • I/O State as a Function of VCCI and VCC Voltage Levels**

Brownout Voltage

Brownout is a condition in which the voltage supplies are lower than normal, causing the device to malfunction as a result of insufficient power. In general, Microsemi does not guarantee the functionality of the design inside Fusion devices if voltage supplies are below their minimum recommended operating condition. Microsemi has performed measurements to characterize the brownout levels of FPGA power supplies. The brownout levels of the power supplies for Fusion devices are designed to be 0.75 V ±0.25 V for VCC and 0.8 V ±0.3 V for VCCI. Characterization tests performed on two AFS600-PQ208E Rev A devices in typical operating conditions showed the brownout voltage levels to be within the specification.

PLL Behavior at Brownout Condition

When PLL power supply voltage and/or VCC levels drop below the VCC brownout levels (0.75 V ±0.25 V), the PLL output lock signal goes low and/or the output clock is lost. The following sections explain PLL behavior during and after the brownout condition.

VCCPLx and VCC Tied Together

In this condition, both VCC and VCCPLx drop below the 0.75 V ±0.25 V brownout level. During the brownout recovery, once VCCPLx and VCC reach the activation point (0.85 ±0.25 V) again, the PLL output lock signal may still remain low with the PLL output clock signal toggling. If this condition occurs, there are two ways to recover the PLL output lock signal:

1. Recycle the power supplies of the PLL (power off and on) by using the PLL POWERDOWN signal.
2. Turn off the input reference clock to the PLL and then turn it back on.

Only VCCPLx Is at Brownout

In this case, only VCCPLx drops below the 0.75 V ±0.25 V brownout level and the VCC supply remains at nominal recommended operating voltage (1.5 V ±0.075 V). In this condition, the PLL behavior after brownout recovery is similar to the initial power-up condition, and the PLL will regain lock automatically after VCCPLx is ramped up above the activation level (0.85 ±0.25 V). No intervention is necessary in this case.

Only VCC Is at Brownout

In this condition, VCC drops below the 0.75 V ±0.25 V brownout level and VCCPLx remains at nominal recommended operating voltage (1.5 V ±0.075 V). During the brownout recovery, once VCC reaches the activation point again (0.85 ±0.25 V), the PLL output lock signal may still remain low with the PLL output clock signal toggling. If this condition occurs, there are two ways to recover the PLL output lock signal:

1. Recycle the power supplies of the PLL (power off and on) by using the PLL POWERDOWN signal.
2. Turn off the input reference clock to the PLL and then turn it back on.

It is important to note that Microsemi recommends using a monotonic power supply or voltage regulator to ensure proper power-up behavior.

Internal Pull-Up/Down

Fusion device I/Os are equipped with internal weak pull-up/-down resistors that can be used by designers. If used, these internal pull-up/-down resistors will be activated during power-up, once both VCC and VCCI are past their functional activation level. Similarly, during power-down these internal pull-up/-down resistors will turn off when the first supply voltage falls below its brownout deactivation level.

Power-Up of Fusion FPGA Fabric Using Internal Voltage Regulator

The internal voltage regulator, when powered on, drives the FPGA core and embedded flash memory, supplying all 1.5 V needs of the Fusion device.

1.5 V Voltage Regulator

In conjunction with an external pass transistor, the voltage regulator (VR) generates a 1.5 V power supply from the 3.3 V power supply (Figure 23-2). The VR can drive up to 20 mA of current to the base of the transistor. The maximum 1.5 V current supplied will depend upon the gain of the transistor utilized and the clamp current of the 3.3 V power supply. Using a typical transistor with a gain of 25, the 1.5 V output can easily supply current requirements for the Fusion device. If there are additional 1.5 V board requirements, the transistor can be sized accordingly. Microsemi recommends using the following transistors with the VR: PN2222A or 2N2222A.

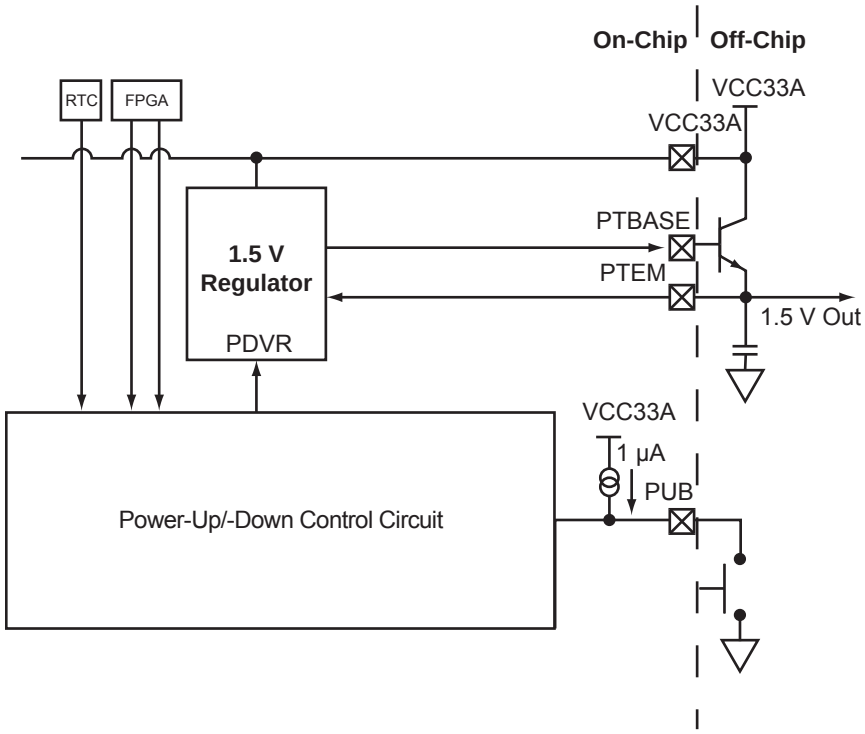


***Figure 23-2* • 1.5 V Voltage Regulator (VR)**

Voltage Regulator and Power Supply Monitor

Since the functions of the voltage regulator logic and power system work together to control the power-up state of the FPGA core, these functions were combined into a single voltage regulator power supply monitor (VRPSM), as shown in Figure 23-3.

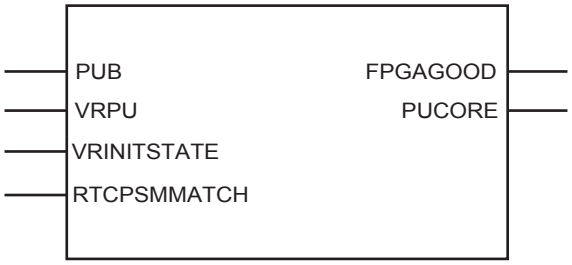


***Figure 23-3* • VRPSM Macro Block**

For more details on the components of the VRPSM, refer to Figure 23-5 on page 23-447.

The voltage regulator can be triggered by several signals: PUB, VRPU, MATCH signal from RTC or VRINITSTATE signal from the VR Init block.

The power-up bar (PUB) input comes from the PUB pin on the device, and hence can be a signal external to the Fusion device. The PUB is an active low input.

The VR Init block decides if the voltage regulator should turn on when the 3.3 V supply is first applied. Since the FPGA is not operating when the 3.3 V supply is off, the VR Init block lets the user define VR behavior at design time. When using the SmartGen tool, the VRINITSTATE input of the VRPSM macro is configured. If turning on the VR at device power-up is desired, the VRINITSTATE is configured to logic 1.

The VRINITSTATE value is also the initial power-up value of the FPGAGOOD output signal of the VRPSM macro. This allows the user to drive FPGAGOOD to logic 1 or 0 even before the 3.3 V supply is up. When at logic 1 value, the FPGAGOOD signal indicates that the FPGA is logically functional. The default power-up value of FPGAGOOD is '0'. If at power-up, VRINITSTATE is '1' or the condition on VRPU, PUB, and RTCPSMMATCH triggers a VR power-up, then FPGAGOOD will be '1'. If VRPU, PUB, and RTCPSMMATCH are not driven, then FPGAGOOD is '0', unless it is already at '1'. In that case, it remains at '1'.

The PUB, VRPU, and RTCPSMMATCH values are never unknown. Pull-up logic for PUB and pull-down logic for VRPU ensure that PUB and VRPU are always at their default inactive values when they are not driven.

VRPU can not be driven to '1' before FPGAGOOD is '1'.

The VRINITSTATE input has equal priority with PUB, RTCPSMMATCH, or VRPU signals. For example, if at power-up both the VRINITSTATE and PUB signals are logic 0, the FPGAGOOD signal will be logic 1.

The PUCORE output is the PUB input inverted. The PUCORE output can be left floating. By connecting it to logic inside the fabric, it can be used as an indicator of the PUB pad status.

Power-Down of FPGA Fabric Using the Internal Voltage Regulator

Powering down the VR turns off the 1.5 V power supply to the FPGA fabric.

Once triggered, the voltage regulator remains on because of the latching functions of flip-flops. Only the FPGA fabric can reset these flip-flops and turn off the VR. Power-down of the VR is controlled by the input signal VRPU to the VRPSM macro. VRPU can be connected to internal logic in the FPGA fabric. Shown in Figure 23-5 on page 23-447, VRPU is also known as the FPGA_VRON signal.

Logic 1 on the VRPU input indicates that the VR should remain on, while logic 0 shuts off the regulator. A transition from logic 1 to logic 0 indicates an FPGA command to shut off the VR (and to shut off the FPGA if the FPGA is powered by the VR). The logic 0 state must be maintained until the FPGA shuts down in order to complete the voltage regulator shutdown. However, if the PUB pad is held low, and hence active, the VR remains on regardless of the state of the VRPU signal.

JTAG Restriction to Powering-Off FPGA Fabric

The Fusion voltage regulator has been designed to be enabled whenever the JTAG port is active in order to make sure that the FPGA is powered-on during different JTAG operations (i.e., programming). The JTAG port is active when the TRST pin is at logic 1.

The TRST pin functions as an active low input to reset the boundary scan circuitry.

If the JTAG interface is active, the voltage regulator cannot be shut off, regardless of the state of the VRPU signal. Hence, in order to turn off VR, the TRST pin must be grounded.

As already mentioned, by using the SmartGen tool, the user can specify that the VR should be turned off at device power-up via VRINITSTATE signal. However, unless the TRST is grounded through a 470 Ω resistor, the VR will turn on as soon the board is powered up.

Periodic Power-Up/Down of FPGA Fabric

The Fusion real-time counter (RTC) system can be configured to power-up the FPGA fabric at a specific time or periodically. The RTC provides time-matching events that may be used so that whenever a match event occurs, the RTC controls power-up of the 1.5 V VR. After the FPGA fabric has completed its power-up sequence, its custom logic or soft microcontroller takes the appropriate actions, after which it controls the power-down of the voltage regulator, which turns the 1.5 V FPGA supply off.

Real-Time Counter (RTC)

The RTC operation requires a 3.3 V power supply, distinguishing it from the 1.5 V FPGA core and embedded flash memory (Figure 23-4). The crystal oscillator must also be enabled in order to be used as a clock source for the RTC.

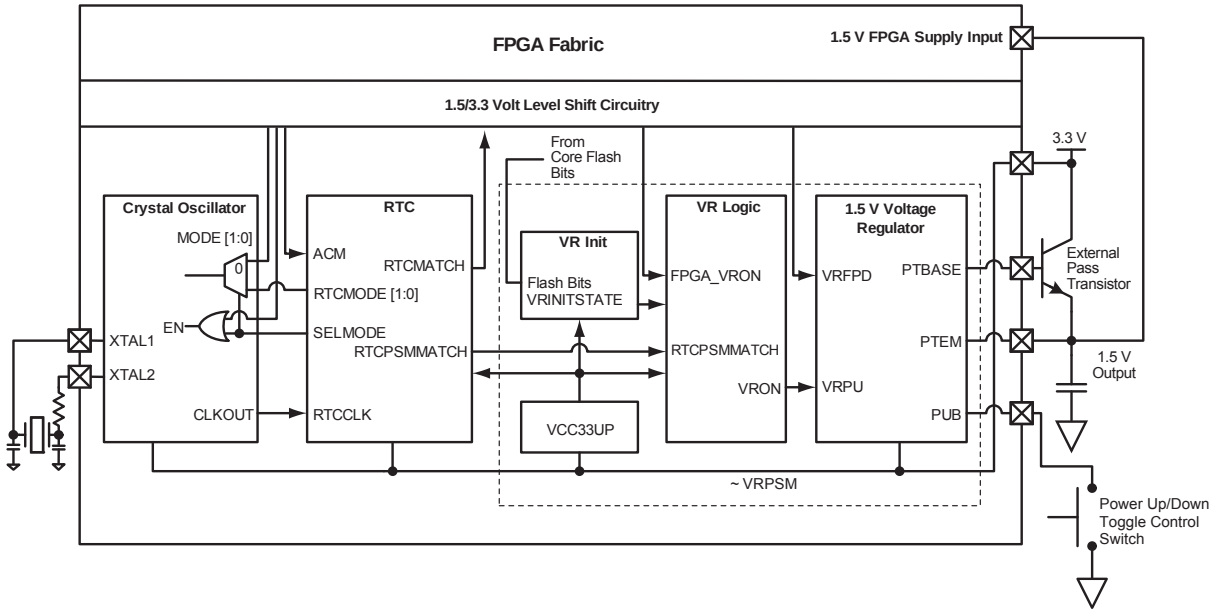


***Figure 23-4* • Real-Time Counter System**

The RTC, built from and controlled by a set of registers, is accessed via the analog configuration multiplexer (ACM). In Figure 23-5, a 40-bit loadable counter is used as the primary timekeeping element within the RTC. This counter can be configured to reset itself when a count value is reached that matches the value set within a 40-bit match register. As its reference clock, the counter logic uses a prescaled clock signal generated by the 7-bit prescaler block. Both the counter and match registers are addressable (read/write) from the FPGA and through a JTAG instruction.

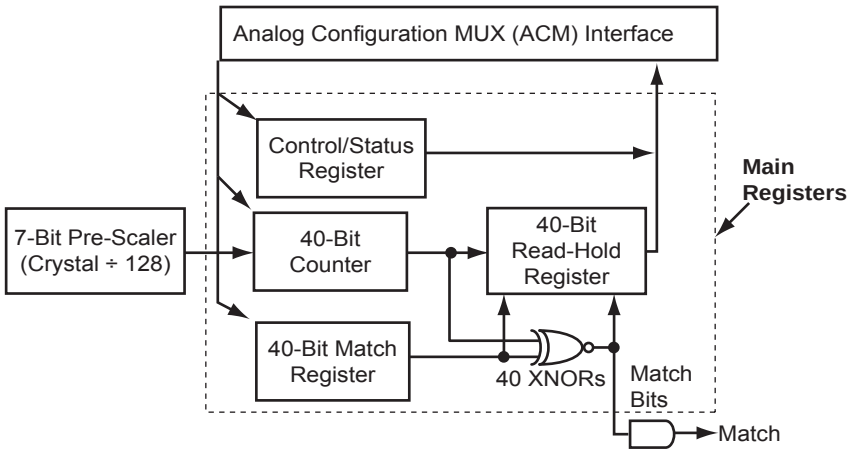


***Figure 23-5* • RTC Block Diagram**

When the device is first powered up (i.e., when the 3.3 V supply becomes valid), the 40-bit counter and 40-bit match register are cleared to logic 0. At this time, the MATCH output signal (provided by RTC) is active (logic 1). At any instance when the 40-bit counter value does not match the value in the 40-bit match register, the MATCH output signal will become inactive (logic 0).

The FPGA fabric portion of the Fusion device must be powered up and active at least once to write to the various registers within the RTC to initialize them for the user's application. Users set up the RTC by configuring the RTC from the SmartGen tool, implementing custom logic, or programming a soft microcontroller.

The operation of the RTC is defined by the 8-bit control status register that is configured through ACM. For a periodic power-up/down of the VR using RTC, the following bits of the control status register must be set:

- The rtc_rst bit must be logic 0 so that the RTC reset is removed synchronously after 2 rising edges of ACM_CLK.
- The rstb_cnt bit must be logic 1 in order to allow the counter to count.
- The vr_en_mat bit must be logic 1 in order to enable the MATCH output port when a match occurs between the 40-bit register and the 40-bit match register.
- The rst_cnt_omat bit must be logic 1, allowing the counter to clear itself when a match occurs.
- The cntr_en bit must be logic 1, enabling the counter.

When using the SmartGen tool to configure the RTC, the bits of the control status register are set by SmartGen. The user only needs to set these control bits when using a microcontroller.

During the power-down of the 1.5 V VR, the 3.3 V supply must remain active in order to keep the RTC actively counting for the next match event, which will cause a repeat power-up/power-down sequence to occur. In order to accomplish this, the match register must be set to the period of time that will be repeated for the power-up/power-down sequence.

Basic Operation Modes

The addition of the RTC system enables Fusion devices to support both standby and sleep modes of operation.

Standby

In standby mode with the 3.3 V power supply applied, the FPGA is not powered on. The crystal oscillator is turned on and the RTC is running off the 3.3 V input to the Fusion device. Furthermore, the 1.5 V voltage regulator is off. The voltage regulator can be turned on by reaching a match count in the RTC or by grounding the PUB pad.

Sleep

With the 3.3 V power supply applied, the FPGA is powered off in the sleep operation mode. The crystal oscillator, RTC, and VR are all turned off. The voltage regulator can be turned on by an external stimulus (i.e., push button) applied to the PUB pin.

It takes 100 µs for the voltage regulator to re-power if PUB pin is not driven to '0' when JTAG comes out of the reset. In order to avoid this power-up delay, PUB should be driven to '0' when TRST changes from '0' to '1'.

Conclusion

Fusion family devices are live at power-up and do not suffer from high the inrush and configuration currents associated with SRAM technologies. Fusion offers further power savings with standby and sleep modes of operation. Utilizing the integrated 1.5 V voltage regulator, Fusion devices can operate from a single 3.3 V power supply, thereby eliminating the need for any power supply sequencing, making Fusion an ideal system master.

List of Changes

The following table lists critical changes that were made in the current version of the document.

Previous Version	Changes in Current Version (Revision 3)	Page
August 2012	This chapter was added to the Fusion and Extended Temperature Fusion FPGA Fabric User's Guide. It had been removed from the Fusion Handbook in 2008 (SAR 29841).	N/A
51900150-1/4.08	The document was converted to the new handbook format.	N/A
51900150-0/7.06	In Table 23-1, Quiet I/O Supply Voltage (VMVx) was deleted.	439
	The two conditions in the "Power-Up/Down Sequence and Transient Current" section were updated and condition 3 is new.	441
	The "PLL Behavior at Brownout Condition" section and all its subsections are new.	443
	V_{CCPLL} was changed to V_{CCPLX}.	N/A

24 – Fusion Solutions, Design Examples, and Reference Designs

The unprecedented level of integration of Fusion enables a wide variety of functionality, such as power and thermal management, remote communications, and system clocking, in a single mixed signal FPGA.

Microsemi, the world leader of mixed signal FPGAs, now offers the only single-chip system management solution. The Fusion mixed signal FPGA integrates configurable analog, large flash memory blocks, comprehensive clock generation and management circuitry, and high-performance programmable logic in a monolithic device. Microsemi has developed turn-key solutions, including a development kit and a software GUI, for system management. This level of integration, configurability, and support establishes Fusion as the definitive system management solution.

This chapter captures the design examples and reference design information for Fusion applications in various aspects.

System Management Applications

System management continues to gain importance in the design of all electronic systems. Smaller process geometries drive more multivolt devices and are more susceptible to voltage and temperature fluctuations. Whereas system management designs can run into hundreds of discrete components, the Fusion mixed signal FPGA solution can integrate these system management functions and provide programmable flexibility and system-level integration—all in a single chip. Unprecedented integration in Fusion devices can offer cost and space savings of 50% or greater relative to current implementations.

White paper*: System Management Using a Mixed Signal FPGA*

Power Management

- Control up to 10 power supplies
  - Voltage monitoring
  - Current monitoring
- Power-on detection and reset
- Custom power mode topology support for total system power reduction
- Power-up sequencing

Power Supply Monitoring

Application note: *Multi-Channel Analog Voltage Comparator in Fusion FPGAs*

- VHDL Design Files
- Verilog Design Files

Power Sequencing

Application note: *Fusion Power Sequencing and Ramp-Rate Control*

- Design Files

Motor Control

An obstacle to wide-scale use of electronic motor control has been the high cost of the computer and power electronics. This obstacle is diminishing due to tremendous technology improvement in semiconductor processes and integration. The Fusion mixed signal FPGA offers unprecedented integration by combining mixed signal analog, flash memory, and FPGA fabric in a monolithic PSC. This means that for the first time, engineers can combine the motor control analog front end, high-speed flash lookup tables, and deterministic algorithm processing capabilities of programmable logic in a single-chip solution.

Reference board: Motor Control Reference Board

Application note: *PID Control*

MicroTCA

Microsemi provides highly integrated solutions for MicroTCA system management. Microsemi supports Fusion-based solutions with reference designs, semiconductor intellectual property (IP), software, and customization services that enable quicker time to market for Microsemi customers with reduced risk, lower costs, and improved availability over existing solutions. The high integration of Microsemi MicroTCA solutions also provides increased functionality in a fixed form factor.

Nearly every electronic system needs system management, especially those with telecommunications-driven standards like MicroTCA, ATCA, and AMC, as well as those with server-driven standards like IPMI. Fusion and ProASIC®3 are adept at supporting both proprietary and standards-based implementations.

Application note: *Actel Fusion FPGAs Supporting Intelligent Peripheral Management Interface (IPMI) Applications*

Application note: *MicroTCA*

Reference design: MicroTCA Power Module Reference Design

Other Applications

Single chip, live at power-up, embedded flash memory, and low power make Fusion suitable for many other applications.

Application note: *Context Save and Reload*

- Design Files

Application note: *Real-Time Calendar Applications in Actel Fusion Devices*

- Design Files

Technical brief: *Using Fusion FIFO for Generating Periodic Waveforms*

- Fusion Sine Table

Application note: *Using Fusion RAM as Multipliers*

Application note: *Configuring CorePWM Using RTL Blocks*

- Verilog Design Files
- VHDL Design Files

Application brief: *Smart Battery Management Applications*

Development System

Fusion Starter Kit

The Fusion Starter Kit is an all-inclusive, low-cost evaluation kit for the Fusion family. Users can utilize this starter kit to explore the voltage, current, and temperature monitor, real-time counter for low power use model, and embedded flash for context-saving applications.

User's guide: *Fusion Starter Kit User's Guide & Tutorial*

- Design Files for Fusion Starter Kit Tutorial

System Management Development Kit

The Microsemi System Management Development Kit provides an excellent platform for developing system management applications and applications with a microprocessor. The kit includes an ARM-enabled Fusion device, a system management GUI, and a platform for systems that performs these functions:

- Power-up detection
- Power sequencing
- Thermal management
- Sleep modes
- System diagnostics
- Remote communications
- Clock generation and management
- Data logging

User's guide: *System Management Board User's Guide*

A – Fusion Glossary

AB

Analog Block

ACM

Analog Configuration Multiplexer. Stores configuration data for the Analog Quads.

ACMCLK

The clock input to the ACM used for configuration/initialization of the Analog Quads: 10 MHz max.

ADC

Analog-to-digital converter

ADCCLK

The clock input to the AB used by the ADC: 10 MHz max.

ADCSTART

Request to the ADC to begin sampling and convert a defined channel

AHB

Advanced High-Performance Bus. The AHB sits above the APB in the AMBA architecture and implements the features required for high-performance, high-clock-frequency systems, including burst transfers, split transactions, single-cycle bus master handover, single-clock-edge operation, non-tristate implementation, and wider data bus configurations (64/128 bits).

AMBA

Advanced Microcontroller Bus Architecture. The AMBA protocol is an open-standard, on-chip bus specification that details a strategy for the interconnection and management of functional blocks.

APB

Advanced Peripheral Bus. As part of the AMBA architecture, the APB is optimized for reduced interface complexity and is used to interface with low-bandwidth peripherals.

ASB

Analog System Builder

ASSC

ADC Sample Sequence Controller

CCC

Clock Conditioning Circuit, which may include a PLL

CoreABC

AMBA Bus Controller soft IP

CoreAI

Analog Interface IP for microprocessors/-controllers

CTRL_STAT

8-bit register located within the ACM that defines the operation of the RTC

Designer

Microsemi's place-and-route tool, which allows users to assign I/Os, evaluate timing, and generate programming files

FPGAGOOD

Signal from VRPSM indicating the internal voltage regulator is on

INITCLK

Initialization clock used in the initialization client: 10 MHz max.

Libero System-on-Chip (SoC) software

Microsemi's FPGA design environment, integrating synthesis, simulation, place-and-route, design entry, and IP generation tools

MATCH

Signal indicating that RTC count register (COUNTER) matches MATCHREG

MATCHREG

40-bit register containing a user-defined value to be compared with RTC count register

PCLK

APB interface clock

PUB

Power Up (Bar). FPGA input with weak internal pull-up used to force power-up of internal voltage regulator

PUCORE

Inverse of PUB

RTC

Real-Time Counter

RTCPSMMATCH

Match signal from RTC that is passed to VRPSM to power up voltage regulator

SmartTime

Microsemi's static timing analysis tool, integrated into Designer

SMEV

System Monitor Evaluation Phase State Machine

SMTR

System Monitor Transition Phase State Machine

STC

Sample Time Control setting for the acquisition time in ASB

TVC

8-bit user-configurable register that determines the division value required to bring the system clock to less that 10 MHz for the ADC

VRINITSTATE

Defines the voltage regulator state upon power-up

VRPSM

Voltage Regulator Power Supply Monitor

VRPU

Voltage Regulator Power-Up. Internal, user-accessible signal to turn off internal voltage regulator from FPGA core

B – Summary of Changes

History of Revision to Chapters

The following table lists chapters that were affected in each revision of this document. Each chapter includes its own change history because it may appear in other device family user's guides. Refer to the individual chapter for a list of specific changes.

Revision (month/year)	Chapter Affected	List of Changes (page number)
Revision 3 (August 2012)	"FPGA Array Architecture in Low Power Flash Devices" was revised.	22
	"Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" was revised.	105
	"SRAM and FIFO Memories in Microsemi's Low Power Flash Devices" was revised.	231
	"Interfacing with the Fusion Analog System: IP Interface" was revised.	293
	"Temperature, Voltage, and Current Calibration in Fusion FPGAs" was revised.	309
	"In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X" was revised.	403
	"Boundary Scan in Low Power Flash Devices" was revised.	418
	"Fusion Board-Level Design Guidelines" was revised.	438
	The "Power-Up/-Down of Fusion FPGAs" chapter was added (SAR 29841).	439
Revision 2 (December 2011)	"Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" was revised.	105
	"Fusion Embedded Flash Memory Blocks" was revised.	190
	"Designing the Fusion Analog System" was revised.	245
	"Fusion Design Solutions and Methodologies" was revised.	252
	"Interfacing with the Fusion Analog System: Processor/Microcontroller Interface" was revised.	272
	"UJTAG Applications in Microsemi's Low Power Flash Devices" was revised.	428
	"Fusion Board-Level Design Guidelines" was revised.	438
	CoreConsole was removed from the "Fusion Glossary".	455
Revision 1 (June 2011)	The title was of this user's guide was changed from "Fusion FPGA Fabric User's Guide" to "Fusion and Extended Temperature Fusion FPGA Fabric User's Guide."	N/A
	"Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" was revised.	105
	"I/O Software Control in Low Power Flash Devices" was revised.	330
	"In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X" was revised.	403

Revision (month/year)	Chapter Affected	List of Changes (page number)
Revision 0 (July 2010)	The Fusion Handbook was divided into two parts to create the Fusion Mixed Signal FPGAs datasheet and the Fusion FPGA Fabric User's Guide.	N/A
	"Global Resources in Low Power Flash Devices" was revised.	51
	"Clock Conditioning Circuits in Low Power Flash Devices and Mixed Signal FPGAs" was revised.	105
	"I/O Software Control in Low Power Flash Devices" was revised.	330
	"DDR for Microsemi's Low Power Flash Devices" was revised.	345
	"Programming Flash Devices" was revised.	362
	"In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X" was revised.	403
	"Boundary Scan in Low Power Flash Devices" was revised.	418

C – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call 800.262.1060
From the rest of the world, call 650.318.4460
Fax, from anywhere in the world, 650.318.8044

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Visit the Customer Support website (www.microsemi.com/soc/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

Website

You can browse a variety of technical and non-technical information on the SoC home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office. Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

Index

A

ACMCLK 256
acquisition time calculation 241
ADC
  background 233
  clock 236
  configuration and calibration 261
  settings 255
ADCCLK 256
AES encryption 369
Analog Configuration MUX (ACM) 243, 290
  configuration 253
  initialization 258
  reset 258
Analog Quad
  configuration 254
Analog System
  Analog Block settings 283
  design 233
  interconnects 144
  interface components 274
  IP interface 273
  microcontroller interface 253
  operation 276
  processor interface 253
  soft IP design 273
    flow 247
Analog System Client 143
applications
  current monitoring 265
  gate driver 267
  temperature monitoring 266
  voltage monitoring 262
architecture 205
  four I/O banks 15
  global 23
  IGLOO 14
  IGLOO nano 13
  IGLOO PLUS 15
  IGLOOe 16
  ProASIC3 nano 13
  ProASIC3E 16
  routing 20
  spine 33
  SRAM and FIFO 209
architecture overview 13
array coordinates 18

B

boundary scan 413
  board-level recommendations 416
  chain 415
  opcodes 415

C

CCC 74
  board-level considerations 104
  cascading 101
  Fusion locations 75
  global resources 54
  hardwired I/O clock input 100
  IGLOO locations 73
  IGLOOe locations 74
  locations 72
  overview 53
  ProASIC3 locations 73
  ProASIC3E locations 74
  programming 54
  software configuration 88
  with integrated PLLs 71
  without integrated PLLs 71
chip global aggregation 35
CLKDLY macro 57
clock aggregation 36
clock macros 38
clock resources 109
clock sources
  core logic 68
  PLL and CLKDLY macros 65
clocking scheme 256
  ACMCLK 256
  ADCCLK 256
  initialization clock 256
clocks
  delay adjustment 78
  detailed usage information 96
  multipliers and dividers 77
  phase adjustment 79
  physical constraints for quadrant clocks 100
  SmartGen settings 97
  static timing analysis 99
Common Flash Interface (CFI)
  CoreAhbNvm supported commands 185
  data client 169
compiling 321
  report 321
contacting Microsemi SoC Products Group
  customer service 459
  email 459
  web-based technical support 459
CoreAhbNvm
  configuration 181
  port signal descriptions 184

status register flags 186
supported Common Flash Interface (CFI) commands 185
CoreAI 253
settings 253
crystal oscillator 114
package connections 123
current monitoring applications 265
customer service 459

D

Data Storage Client 164
DDR
architecture 331
design example 342
I/O options 333
input/output support 335
instantiating registers 336
design
analog system soft IP flow 247
examples 268, 451
flow 251
microprocessor/microcontroller 250
solutions and methodologies 247
state management 249
design example 47
design recommendations 38
device architecture 205
DirectC code 407

E

efficient long-line resources 21
embedded flash memory 137
busy signal handling 174
initialization IP interface 138
interconnects 144, 146
macro and interface 158
page programming 175
priority of operations 173
read operations 178
updating contents 180
using for general data storage 158
using for initialization 137
write operations 175
encryption 411
examples, design 451

F

FIFO
features 215
initializing 222
memory block consumption 221
software support 228
usage 218
flash memory, embedded
see embedded flash memory
flash switch for programming 11
FlashLock
IGLOO and ProASIC devices 371
permanent 371
FlashROM
access using JTAG port 197
architecture 397
architecture of user nonvolatile 191
configuration 194
custom serialization 203
design flow 198
generation 199
programming and accessing 196
programming file 201
programming files 397
SmartGen 200
FlashROM read-back 427

G

gate driver applications 267
global architecture 23
global buffers
no programmable delays 56
with PLL function 59
with programmable delays 56
global macros
Synplicity 42
globals
designer flow 45
networks 50
spines and rows 33
glossary 455

H

HLD code
instantiating 318

I

I/O banks
standards 32
I/O standards 69
global macros 38
I/Os
assigning technologies 324
assignments defined in PDC file 319
automatically assigning 328
buffer schematic cell 317
cell architecture 333
configuration with SmartGen 314
global, naming 27
manually assigning technologies 324
software-controlled attributes 313
user I/O assignment flow chart 311
initialization clients 137
Analog System 143
RAM 146
Standalone
see Standalone Initialization Client

initialization clock 256
ISP 353, 354
  architecture 391
  board-level considerations 401
  microprocessor 405

J
JTAG 1532 391
JTAG interface 407

L
layout
  device-specific 70

M
MAC validation/authentication 410
macros
  CLKBUF 69
  CLKBUF_LVDS/LVPECL 69
  CLKDLY 57, 65
  FIFO4KX18 215
  PLL 65
  PLL macro signal descriptions 60
  RAM4K9 211
  RAM512X18 213
  supported basic RAM macros 210
  UJTAG 421
MCU FPGA programming model 408
memory
  changing content 249
  embedded flash
    see embedded flash memory
memory availability 220
memory blocks 209
microcontroller
  design 250
  interface 181
microprocessor
  design 250
  interface 181
microprocessor programming 405
Microsemi SoC Products Group
  email 459
  web-based technical support 459
  website 459

N
No-Glitch Multiplexer (NGMUX) 123
  connections 130
  modes of operation 124
  placement 130
  timing analysis 128
  tips 128
  usage 126

O
OTP 353

P
PDC
  global promotion and demotion 43
place-and-route 319
PLL
  configuration bits 82
  core specifications 76
  dynamic PLL configuration 79
  functional description 77
  power supply decoupling scheme 104
PLL block signals 60
PLL macro block diagram 61
prescaler selection 243
product support
  customer service 459
  email 459
  My Cases 460
  outside the U.S. 460
  technical support 459
  website 459
programmers 355
  device support 358
programming
  AES encryption 383
  basics 353
  features 353
  file header definition 387
  flash and antifuse 355
  flash devices 353
  glossary 388
  guidelines for flash programming 359
  header pin numbers 400
  microprocessor 405
  power supplies 393
  security 377
  solution 398
  solutions 357
  voltage 393
  volume services 356
programming support 351

R
RAM
  memory block consumption 221
RAM Initialization Client 146
RAM interconnects 146
RC oscillator, internal 110
  package connections 114
Real-Time Counter (RTC) 131
  designing with 272
  interconnection 136
  tips 135
  usage 131
reference designs 451

remote upgrade via TCP/IP 410
routing structure 20

S

sample code 291
sample rate calculation 240
sample sequencing
  calculation 240
  overview 239
security 394
  architecture 367
  examples 372
  features 368
  FlashLock 371
  FlashROM 195
  FlashROM use models 375
  in programmable logic 365
  overview 365
  read-back prevention 410
signal integrity problem 401
silicon testing 426
SmartDesign
  design state management 249
SmartGen 228
  soft IP blocks 277
    ADC Sample Sequence Controller (ASSC) 277
    System Monitor Evaluation Phase State Machine (SMEV) 281
    System Monitor Transition Phase State Machine (SMTR) 283
solutions 451
spine architecture 33
spine assignment 44
SRAM
  features 211
  initializing 222
  software support 228
  usage 215
Standalone Initialization Client 150
  flash memory ports 150
  usages 152
STAPL player 407
STAPL vs. DirectC 409
synthesizing 318

T

TAP controller state machine 413, 422
tech support
  ITAR 460
  My Cases 460
  outside the U.S. 460
technical support 459
temperature monitoring applications 266
tools overview 251

U

UJTAG
  CCC dynamic reconfiguration 424
  fine tuning 425
  macro 421
  operation 422
  port usage 423
  use to read FlashROM contents 419
ultra-fast local lines 20

V

variable aspect ratio and cascading 219
VersaNet global networks 25
VersaTile 17
very-long-line resources 21
ViewDraw 317
voltage monitoring applications 262
Voltage Regulator Power Supply Monitor (VRPSM) 429
VREF pins
  manually assigning 325

W

web-based technical support 459

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

© 2012 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

50200265-3/8.12