DirectC v3.2

User's Guide

Microsemi®

Table of Contents

1 System Overview . . . 5
Systems with Direct Access to Memory . . . 13
Systems with Indirect Access to Memory . . . 15
2 Generating Data Files and Integrating DirectC . . . 16
Data File Compatibility . . . 16
DirectC v3.2 Code Integration . . . 16
3 Required Source Code Modifications . . . 18
Compiler Switches . . . 18
Hardware Interface Components . . . 20
4 Chain Programming . . . 28
Pre/Post Data Variable Declaration . . . 28
5 Data File Format . . . 30
DAT File Description for AGL, AFS, A3PL, A3PEL, A3P/E, and A2F Devices . . . 30
DAT File Description for M2GL and M2S Devices . . . 32
6 Source File Description . . . 34
DPUSER.C and DPUSER.H . . . 34
DPCOM.C and DPCOM.H . . . 34
DPALG.C and DPALG.H . . . 34
DPG3ALG.C and DPG3ALG.H . . . 34
DPCORE.C and DPCORE.H . . . 34
DPFROM.C and DPFROM.H . . . 34
DPNVM.C and DPNVM.H . . . 34
DPSECURITY.C and DPSECURITY.H . . . 34
DPG4ALG.C and DPG4ALG.H . . . 34
DPJTAG.C and DPJTAG.H . . . 34
DPCHAIN.C and DPCHAIN.H . . . 35
DPUTIL.C and DPUTIL.H . . . 35
7 Disabled Features with ENABLE_CODE_SPACE_OPTIMIZATION . . . 36
DMK Verification for ARM Enabled Devices . . . 36
030/015 Device Check . . . 36
8 Data File Bit Orientation . . . 37
9 Error Messages & Troubleshooting Tips . . . 38
A Product Support . . . 41
Customer Service . . . 41
Customer Technical Support Center . . . 41
Technical Support . . . 41

Website 41
Contacting the Customer Technical Support Center 41
ITAR Technical Support 42

Introduction

This document describes how to enable microprocessor-based embedded ISP (In-System Programming) on Microsemi IGLOO2™, SmartFusion2™, ProASIC®3 (including ProASIC3 nano), IGLOO™ (including IGLOO nano), SmartFusion™ and Fusion™ devices. In-System Programming refers to an external processor on board programming a Microsemi device through general purpose IOs using a JTAG interface.

The document assumes that the target system contains a microprocessor with a minimum 256 bytes of RAM, a JTAG interface to the target device from the microprocessor, and access to the programming data to be used for programming the FPGA. Access to programming data can be provided by a telecommunications link for most remote systems.

DirectC v3.2 is a set of C code designed to support embedded In-System Programming for AGL, AFS, A3PL, A3PEL, A3P/E, A2F, M2S and MGL families. To use DirectC v3.2, you must make some minor modifications to the source code, add the necessary API, and compile the source code and the API together to create a binary executable. The binary executable is downloaded to the system along with the programming data file.

The programming data file is a binary file that can be generated by designer version v8.5 or later. The detailed specification of the programming file is included in "Data File Format" on page 30.

DirectC v3.2 contains several compile options to reduce the code size as much as possible. The compile options enable you to disable features that are not needed in the compile. ENABLE_CODE_SPACE_OPTIMIZATION is a new option added in version v2.4 of the DirectC code to reduce code size requirements to below 8500 bytes. See Table 1-3 on page 6 for details. This additional compile option disables code needed to support certain features that are device or silicon revision specific, such as discrete address shifting needed to support revision A of A3PE1500 silicon. In addition, certain checking functions have been disabled or modified with this option enabled to reduce the code size further. These functions do not impact programming; their purpose is to screen for user errors in programming secured devices (such as devices secured with plain text files).

DirectC v3.2 supports systems with direct and indirect access to the memory space containing the data file image. With paging support, it is possible to implement the embedded ISP using DirectC on systems with no direct access to the entire memory space containing the data. Paging support is accomplished by making modifications to the data communication functions defined in *dpuser.h, dpuser.c, dpcom.c and dpcom.h.*

1 – System Overview

To perform In-System Programming (ISP) for the FPGA, the system must contain the following parameters:

- Control logic (a microprocessor with at least 256 bytes RAM or a softcore microprocessor implemented in another FPGA)
- JTAG interface to the target device
- Access to the data file containing the programming data
- Memory to store and run DirectC code

Note: See your device datasheet for information on power requirements for V_{pump}, V_{jtag} and other power supplies.

Memory requirements depend on the options that are enabled. For example, Table 1-1, Table 1-2, Table 1-3, and Table 1-4 show memory requirements (last three columns) if certain options (X) are enabled.
Text - This is the compiled code size memory requirements.
Data - This is the run time memory requirement, i.e. the free data memory space required to execute the code.

BSS - This is the Block Started by Symbol allocation for variables that do not yet have values, i.e. uninitialized data. It is part of the overall Data size.

In Table 1-1, both ENABLED_G3_SUPPORT and ENABLE_G4_SUPPORT are **enabled**. Refer to section 3 for more detailed description of available compiler switches.

***Table 1-1* • Code Memory Requirements- DirectC Code Size on ARM7 32-Bit Mode**

Core		FROM		Embedded Flash Memory Block			Text	Data	BSS
Plain	Encrypt	Plain	Encrypt	Plain	Encrypt	Security	Units in Bytes		
X	X	X	X	X	X	X	49724	1088	256
X							34104	1088	232
	X						34980	1088	232
		X					33192	1088	232
			X				32888	1088	232
				X			37536	1088	240
					X		35068	1088	240
						X	32992	1088	232
X	X						36316	1088	232
		X	X				33728	1088	232
				X	X		39364	1088	256
X		X		X			41836	1088	256
X		X		X		X	43312	1088	256
	X		X		X		39580	1088	256

Note: X = enabled, blank = disabled

In Table 1-2, ENABLE_G3_SUPPORT is **enabled**, and ENABLE_G4_SUPPORT is **disabled**.

***Table 1-2* • Code Memory Requirements - DirectC Code Size on ARM7 32-Bit Mode**

Core		FROM		Embedded Flash Memory Block			Text	Data	BSS
Plain	Encrypt	Plain	Encrypt	Plain	Encrypt	Security	Units in Bytes		
X	X	X	X	X	X	X	41702	1088	176
X							26088	1088	160
	X						26964	1088	160
		X					25176	1088	152
			X				24872	1088	152
				X			29520	1088	168
					X		27052	1088	168
						X	24976	1088	152
X	X						28300	1088	160
		X	X				25712	1088	152
				X	X		31348	1088	168
X		X		X			33820	1088	176
X		X		X		X	35296	1088	176
	X		X		X		31564	1088	176

Note: X = enabled, blank = disabled

In Table 1-3, ENABLE_G3_SUPPORT is **disabled**, and ENABLE_G4_SUPPORT is **enabled**.

***Table 1-3* • Code Memory Requirements - DirectC Code Size on ARM7 32-Bit Mode**

Core		FROM		Embedded Flash Memory Block			Text	Data	BSS
Plain	Encrypt	Plain	Encrypt	Plain	Encrypt	Security	Units in Bytes		
N/A	N/A	N/A	N/A	N/A	N/A	N/A	20748	1088	200

Note: X = enabled, blank = disabled

For M2S and M2GL devices, programming the core and eNVM is data driven and has no impact on code size.

In Table 1-4, ENABLE_G3_SUPPORT and ENABLE_G4_SUPPORT are **enabled**.

***Table 1-4* • Code Memory Requirements - DirectC Code Size on Cortex M3 16-Bit Mode**

Core		FROM		Embedded Flash Memory Block		Security	Text	Data	BSS
Plain	Encrypt	Plain	Encrypt	Plain	Encrypt		Units in Bytes		
X	X	X	X	X	X	X	19898	1218	64
X							12202	1193	64
	X						12622	1193	64
		X					11910	1195	64
			X				11846	1195	64
				X			14326	1204	64
					X		13070	1200	64
						X	12002	1187	64
X	X						13066	1193	64
		X	X				12250	1195	64
				X	X		15366	1204	64
X		X		X			16006	1218	64
X		X		X		X	16830	1218	64
	X		X		X		14778	1214	64

Note: X = enabled, blank = disabled

In Table 1-5, ENABLE_G3_SUPPORT is **enabled** and ENABLE_G4_SUPPORT is **disabled**.

***Table 1-5* • Code Memory Requirements - DirectC Code Size on Cortex M3 16-Bit Mode**

Core		FROM		Embedded Flash Memory Block			Text	Data	BSS
Plain	Encrypt	Plain	Encrypt	Plain	Encrypt	Security	Units in Bytes		
X	X	X	X	X	X	X	16448	1186	64
X							8752	1161	64
	X						9172	1161	64
		X					8460	1163	64
			X				8396	1163	64
				X			10876	1172	64
					X		9620	1168	64
						X	8552	1155	64
X	X						9616	1161	64
		X	X				8800	1163	64
				X	X		11916	1172	64
X		X		X			12556	1186	64
X		X		X		X	13380	1186	64
	X		X		X		11328	1186	64

Note: X = enabled, blank = disabled

In Table 1-6, ENABLE_G3_SUPPORT is **disabled** and ENABLE_G4_SUPPORT is **enabled**.

***Table 1-6* • Code Memory Requirements - DirectC Code Size on Cortex M3 16-Bit Mode**

Core		FROM		Embedded Flash Memory Block			Text	Data	BSS
Plain	Encrypt	Plain	Encrypt	Plain	Encrypt	Security	Units in Bytes		
N/A	N/A	N/A	N/A	N/A	N/A	N/A	5538	1162	60

Note: X = enabled, blank = disabled

For M2S and M2GL devices, programming the core and eNVM is data driven and has no impact on code size.

In Table 1-7, ENABLE_G3_SUPPORT and ENABLE_G4_SUPPORT are **enabled.**

***Table 1-7* • Code Memory Requirements - DirectC Code Size on C8051F120 8-Bit Mode - Optimization Compile Switches OFF**

CORE		FROM		Embedded Flash Memory Blocks			Code Size	RTM
PLAIN	ENCRYPT	PLAIN	ENCRYPT	PLAIN	ENCRYPT	Security	Units in Bytes	
X							13201	275
	X						13423	275
X	X						13677	275
		X					12901	272
			X				12815	272
		X	X				13092	272
				X			15547	286
					X		13971	265
				X	X		16327	268
						X	12935	271
X		X		X		X	17468	291
	X		X		X		15325	270
X	X	X	X	X	X	X	18902	273

Note: X = enabled; blank = disabled

In Table 1-8, ENABLE_G3_SUPPORT is **enabled** and ENABLE_G4_SUPPORT is **disabled.**

***Table 1-8* • Code Memory Requirements - DirectC Code Size on C8051F120 8-Bit Mode - Optimization Compile Switches OFF**

CORE		FROM		Embedded Flash Memory Blocks		Security	Code Size	RTM
PLAIN	ENCRYPT	PLAIN	ENCRYPT	PLAIN	ENCRYPT		Units in Bytes	
X							9737	201
	X						9959	201
X	X						10213	201
		X					9437	198
			X				9351	198
		X	X				9628	198
				X			12083	213
					X		10507	192
				X	X		12863	195
						X	9471	197
X		X		X		X	14004	218
	X		X		X		11861	197
X	X	X	X	X	X	X	15438	200

Note: *X = enabled; blank = disabled*

In Table 1-9, ENABLE_G3_SUPPORT is **disabled** and ENABLE_G4_SUPPORT is **enabled.**

***Table 1-9* • Code Memory Requirements - DirectC Code Size on C8051F120 8-Bit Mode - Optimization Compile Switches OFF**

CORE		FROM		Embedded Flash Memory Blocks		Security	Code Size	RTM
PLAIN	ENCRYPT	PLAIN	ENCRYPT	PLAIN	ENCRYPT		Units in Bytes	
	X				X		7083	255

Note: *X = enabled; blank = disabled*

In Table 1-10, ENABLE_G3_SUPPORT and ENABLE_G4_SUPPORT are **enabled.**

***Table 1-10* • Code Memory Requirements - DirectC Code Size on C8051F120 8-bit Mode - Optimization Compile Switches ON**

CORE		FROM		Embedded Flash Memory Blocks		Security	Code Size	RTM
PLAIN	ENCRYPT	PLAIN	ENCRYPT	PLAIN	ENCRYPT		Units in Bytes	
X							10105	274
	X						10286	274
X	X						10529	274
		X					9879	271
			X				9793	271
		X	X				10078	271
				X			12138	285
					X		10841	264
				X	X		12862	267
						X	9955	270
X		X		X		X	13704	290
	X		X		X		11823	269
X	X	X	X	X	X	X	15022	272

Note: X = enabled, blank = disabled

In Table 1-11, ENABLE_G3_SUPPORT is **enabled** and ENABLE_G4_SUPPORT is **disabled.**

***Table 1-11* • Code Memory Requirements - DirectC Code Size on C8051F120 8-bit Mode - Optimization Compile Switches ON**

CORE		FROM		Embedded Flash Memory Blocks		Security	Code Size	RTM
PLAIN	ENCRYPT	PLAIN	ENCRYPT	PLAIN	ENCRYPT		Units in Bytes	
X							6641	200
	X						6822	200
X	X						7065	200
		X					6415	197
			X				6329	197
		X	X				6614	197
				X			8674	212
					X		7377	191
				X	X		9398	194
						X	6491	196
X		X		X		X	10240	217
	X		X		X		8359	196
X	X	X	X	X	X	X	11558	199

Note: X = enabled, blank = disabled

In Table 1-12, ENABLE_G3_SUPPORT is **disabled** and ENABLE_G4_SUPPORT is **enabled.**

***Table 1-12* • Code Memory Requirements - DirectC Code Size on C8051F120 8-bit Mode - Optimization Compile Switches ON**

CORE		FROM		Embedded Flash Memory Blocks		Security	Code Size	RTM
PLAIN	ENCRYPT	PLAIN	ENCRYPT	PLAIN	ENCRYPT		Units in Bytes	
N/A	N/A	N/A	N/A	N/A	N/A	N/A	7083	255

Note: X = enabled, blank = disabled

For M2S and M2GL devices, programming the core and eNVM is data driven and has no impact on code size.

Systems with Direct Access to Memory

Figure 1-1 shows the overview of a typical system with direct access to the memory space holding the data file. See section 2 for generating DAT files and Table 1-13 for data storage memory requirements.

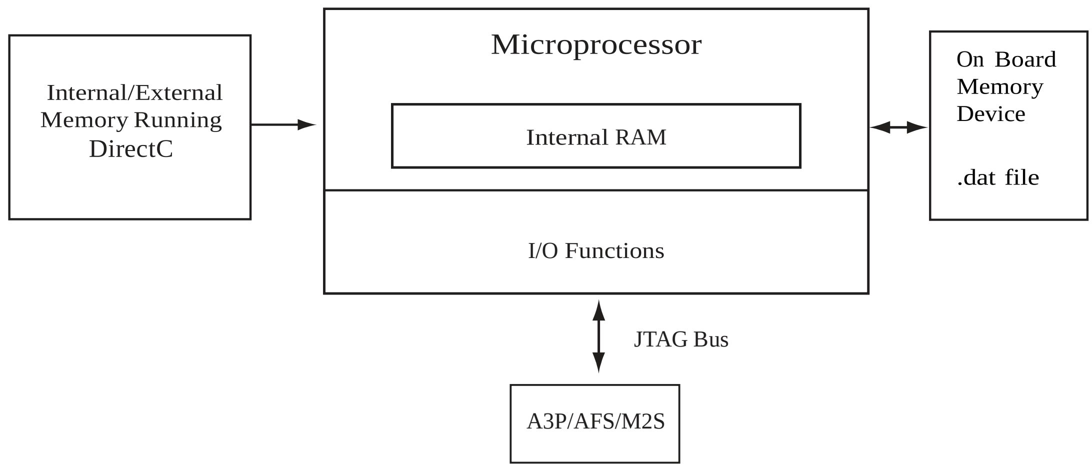

Figure 1-1 **• System with Direct Access to Memory**

Systems with Indirect Access to Memory

Figure 1-2 is an overview of a system with no direct access to the memory space holding the data file. For example, the programming data may be received via a communication interface peripheral that exists between the processor memory, and the remote system holding the data file. *dpcom.h* and *dpcom.c* must be modified to interface with the communication peripheral.

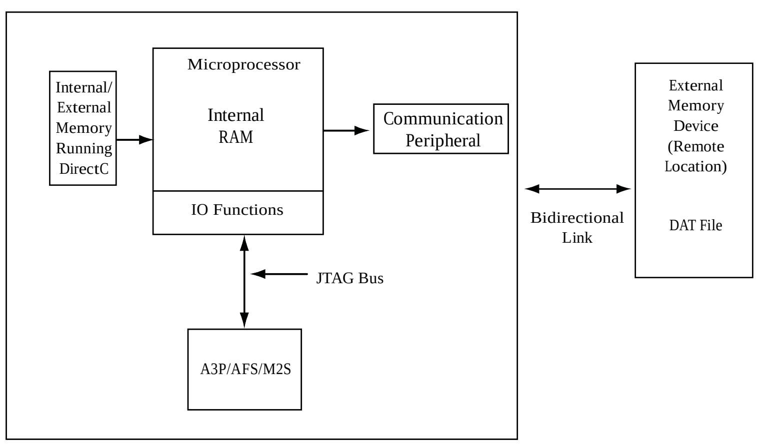

Figure 1-2 **• System With Indirect Access to Memory**

Table 1-13 **• Data Storage Memory Requirements - Data Image Size**

	Data Image Size						
	Core/FPGA Array		FROM		Embedded Flash Memory Block		
Device	Plain (kB)	Encrypt (kB)	Plain (kB)	Encrypt (kB)	Plain (kB)	Encrypt (kB)	Security (kB)
A3PE600	526	647	1	1	N/A	N/A	1
A3PE1500*	1434	1765	1	1	N/A	N/A	1
A3PE3000	2790	3433	1	1	N/A	N/A	1
A3P015	32	N/A	1	N/A	N/A	N/A	1
A3P030	32	N/A	1	N/A	N/A	N/A	1
A3P060	64	79	1	1	N/A	N/A	1
A3P125	127	156	1	1	N/A	N/A	1
A3P250	235	288	1	1	N/A	N/A	1
A3P400	351	432	1	1	N/A	N/A	1
A3P600	523	647	1	1	N/A	N/A	1
A3P1000	915	1126	1	1	N/A	N/A	1
AFS090	96	117	1	1	256	545	1
AFS250	234	288	1	1	256	545	1
AFS600	526	647	1	1	512	1090	1
AFS1500	1434	1765	1	1	2048	2180	1
A2F200M3F	181	222	1	1	256	545	1
A2F500M3G	455	560	1	1	512	1090	1
M2GL010	N/A	557	N/A	N/A	N/A	267	N/S
M2GL025	N/A	1197	N/A	N/A	N/A	267	N/S
M2GL050	N/A	2364	N/A	N/A	N/A	267	N/S
M2S005	N/A	297	N/A	N/A	N/A	137	N/S
M2S010	N/A	557	N/A	N/A	N/A	272	N/S
M2S025	N/A	1197	N/A	N/A	N/A	272	N/S
M2S050	N/A	2364	N/A	N/A	N/A	272	N/S

A3PE1500 is not supported with an 8-bit processor.

INA - Information not available at this time.

N/A - Not applicable

N/S - Not supported

All data in the table for base FPGA devices applies equally to the M1, M7, P1 and U1 encrypted versions of the devices, e.g. data for AFS1500 is equally applicable to M1AFS1500, P1AFS1500 and U1AFS1500. Not all combinations of M1, M7, P1 and U1 are available for all devices. Refer to the product datasheets for available devices.

The total image size is the sum of all the corresponding enabled blocks for the specific target device.

2 – Generating Data Files and Integrating DirectC

This chapter describes the flows for data file generation and DirectC code integration.

To generate your data file:

1. Generate the DAT file using Designer v8.5 or later. See the latest Libero SoC online help for information on generating a DAT file.
2. Program the DAT file into the storage memory.

Data File Compatibility

DirectC data files can be generated from Designer v8.5 and above. Data files generated from Designer v8.5 are identical to the files generated by the original datgen tool with the exception of the file title. However, data files generated by Designer version v8.6 are enhanced to support nano devices. DirectC v3.2 can detect which version of the file is being used and handle it accordingly.

DirectC v3.2 Code Integration

Figure 2-1 shows the DirectC integration use flow.

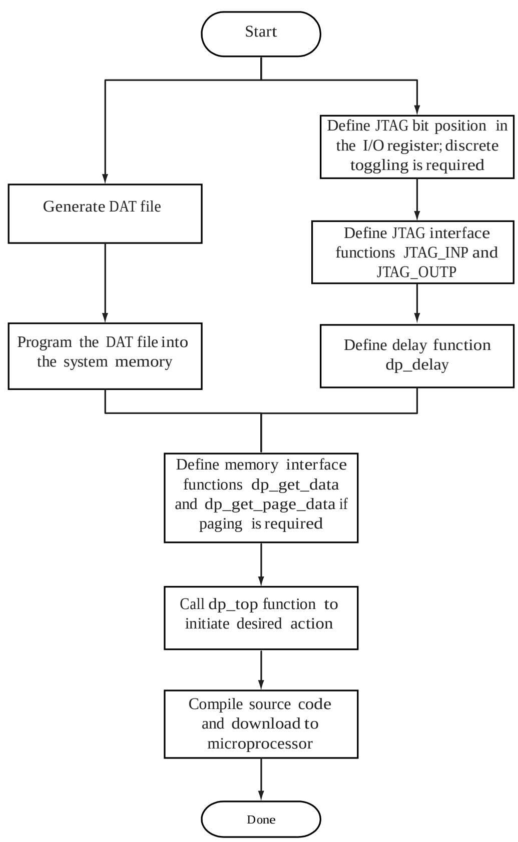

Figure 2-1 **• Importing DirectC Files**

To use DirectC v3.2 code integration:

1. Import the DirectC v3.2 files shown in Figure 2-2 into your development environment.

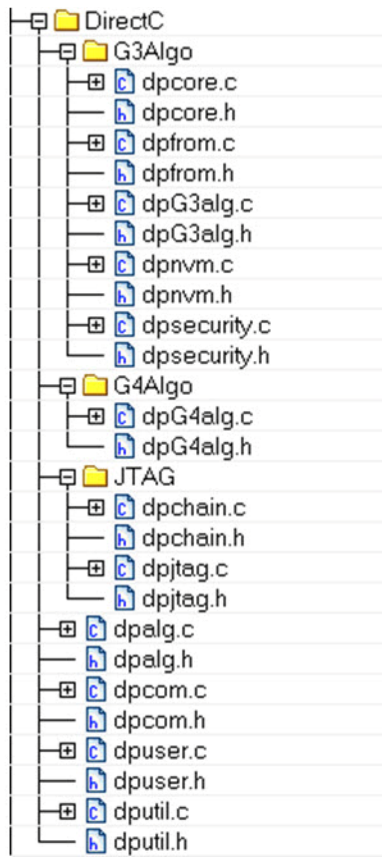

Figure 2-2 **• DirectC v3.2 Files to import into your Development Environment**

2. Modify the DirectC code. Refer to flow chart in figure 2-1.
   – Define JTAG pin bit locations in the I/O register.
   – Add API to support discrete toggling of the individual JTAG pins.
   – Modify the hardware interface functions (*inp_jtag* and *outp_jtag*) to use the hardware API functions designed to control the JTAG port.
   – Modify the delay function (*dp_delay*).
   – Modify memory access functions to access the data blocks within the image file programmed into the system memory. See "Data File Bit Orientation" on page 37.
   – Call d*p_top* function with the action code desired. See "DPG3ALG.C and DPG3ALG.H" on page 34 for supported actions and their corresponding codes.
3. Compile the source code. This creates a binary executable that is downloaded to the system for execution.

3 – Required Source Code Modifications

You must modify the *dpuser.h* and *dpuser.c* files when using the DirectC source code. "Source File Description" on page 34 contains a short description of DirectC source code and their functions. Functions that must be modified are listed in Table 3-2.

Table 3-1 **• Compiler Switches**

Function	Source File	Purpose
Jtag_inp	Dpuser.c	Hardware interface function used to set JTAG pins and read TDO
Jtag_outp	Dpuser.c	Hardware interface function used to set JTAG pins
dp_get_page_data	Dpcom.c	Programming file interface function
dp_delay	Dpuser.c	Delay function
dp_display_text	Dpuser.c	Function to display text to an output device
dp_display_value	Dpuser.c	Function to display value of a variable to an output device

Compiler Switches

The compiler switches in Table 3-2 are designed to allow you to easily adjust the compiled code size by enabling or disabling specific support in DirectC. For example, to enable FPGA Array (Core) plain text programming, CORE_SUPPORT and CORE_PLAIN must be defined. Table 3-2 lists the available compiler switches in the project.

Table 3-2 **• Compiler Switches**

Compiler Switch	Source File	Purpose
CORE_SUPPORT	dpG3alg.h	Enables FPGA Array Programming support
CORE_ENCRYPT	dpG3alg.h	Specify to include FPGA Array Encrypted programming support
CORE_PLAIN	dpG3alg.h	Specify to include FPGA Array Plain Text programming support
FROM_SUPPORT	dpG3alg.h	Enables FlashROM Programming support
FROM_ENCRYPT	dpG3alg.h	Specify to include FlashROM Encrypted programming support
FROM_PLAIN	dpG3alg.h	Specify to include FlashROM Plain Text programming support
NVM_SUPPORT	dpG3alg.h	Enables eNVM Programming support
NVM_ENCRYPT	dpG3alg.h	Specify to include eNVM Encrypted programming support
NVM_PLAIN	dpG3alg.h	Specify to include eNVM Plain Text programming support

***Table 3-2* • Compiler Switches**

Compiler Switch	Source File	Purpose
SECURITY_SUPPORT	dpG3alg.h	Enables Security Programming support
SILSIG_SUPPORT	dpG3alg.h	Enables SILSIG Programming support
ENABLE_DAS_SUPPORT	dpG3alg.h	Enables support for A3PE1500 rev A devices; support for this feature is not available on some 8-bit microcontrollers due to Run Time Memory requirements
ENABLE_GPIO_SUPPORT	dpuser.h	This switch must be defined to enable external device programming
ENABLE_G3_SUPPORT	dpuser.h	Enables support for AGL, AFS, A3PL, A3PEL, A3P/E, and A2F devices
ENABLE_G4_SUPPORT	dpuser.h	Enables support for M2S and MGL devices
ENABLE_DISPLAY	dpuser.h	Enables display functions
USE_PAGING	dpuser.h	Used to enable paging implementation for memory access
CHAIN_SUPPORT	dpuser.h	Used to enable support for chain programming as described in Table 4-2 on page 29.
BSR_SAMPLE	dpuser.h	Enable this option to maintain the last known IO states during programming. **BSR loading and BSR_SAMPLE are not supported for IAP.**
ENABLE_CODE_SPACE_OPTIMIZATION	dpG3alg.h	See "Disabled Features with ENABLE_CODE_SPACE_OPTIMIZATION" on page 36
DISABLE_CORE_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable array specific actions such as erase, program and verify array actions.
DISABLE_FROM_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable FROM specific actions such as erase, program and verify FROM actions.
DISABLE_NVM_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable NVM specific actions such as program and verify NVM actions.
DISABLE_SEC_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable security specific actions such as erase and program security actions.

Note: Make sure that the appropriate compiler options are enabled to support all features available in the STAPL/ Dat file. Otherwise, DirectC may report an error depending on the requested action. Avoid using source files that have all options enabled. The number of options selected incrementally increases the number of variables that need to be maintained and the amount of memory that is used.

Compiler options defined in dpG3alg.h are specific to the AGL, AFS, A3PL, A3PEL, A3P/E and A2F families of devices, whereas compiler switches defined in dpuser.h are common to all devices.

Hardware Interface Components

Define JTAG Hardware Bit Assignments (dpuser.h)

Define the JTAG bits corresponding to each JTAG pin. This is usually the bit location of the I/O register controlling the JTAG port of the target device.

```
#define TCK 0x1 /* ... user code goes here ... */
#define TDI 0x2 /* ... user code goes here ... */
#define TMS 0x4 /*... user code goes here ... */
#define TRST 0x0 /* ... user code goes here ... set to zero if does not exist !!!*/
#define TDO 0x80 /*.. user code goes here ... */
```

Hardware Interface Function (dpuser.c)

jtag_inp and *jtag_outp* functions are used to interface with the JTAG port. A register *jtag_port_reg* is an 8 bit register already defined in DirectC. DirectC uses it to track the logical states of all the JTAG pins.

jtag_inp Function

This function returns the logical state of the TDO pin. If it is logic level zero, this function must return zero. If the logical state is 1, it must return 0x80.

jag_outp Function

This function takes one argument that is the value of the JTAG port register containing the states of all the JTAG pins. It sets the JTAG pins to the values in this argument.

Delay Function

dp_delay function, defined in dpuser.c, takes one argument which is the amount of time in microseconds. Its purpose is to pause for a minimum of time passed in its argument.

Longer delay time does not impact programming other than programming time.

Display Functions

Display functions are only enabled if the ENABLE_DISPLAY compiler switch is enabled. Three functions, *dp_display_array, dp_display_text* and *dp_display_value*, are available to display text as well as numeric values. You must modify these functions for proper operation.

Memory Interface Functions

All access to the memory blocks within the data file is done through the *dp_get_data* function within the
DirectC code. This is true for all system types.

This function returns an address pointer to the byte containing the first requested bit.

The dp_get_data function takes two arguments as follows:

- var_ID: an integer variable that contains an identifier specifying which block within the data file needs to be accessed.
- bit_index: The bit index addressing the bit to address within the data block specified in Var_ID.

Upon completion of this function, return_bytes variable must hold the total number of valid bytes available for the calling function.

See "Systems with Direct Access to the Memory Containing the Data File" and "Systems with Indirect Access to the Data File" on page 21 for details.

Systems with Direct Access to the Memory Containing the Data File

Since the memory space holding the data file is accessible by the microprocessor, it can be treated as an array of unsigned characters. In this case, complete these steps:

1. Disable the USE_PAGING compiler switch. See "Compiler Switches" on page 18.

2. Assign the physical address pointer to the first element of the data memory location (*image_buffer* defined in *dpcom.c)*. *Image_buffer* is used as the base memory for accessing the information in the programming data in storage memory.

The dp_get_data function calculates the address offset to the requested data and adds it to *image_buffer*. *Return_bytes* is the requested data.

An example of the *dp_get_data* function implementation follows.

```
DPUCHAR* dp_get_data(DPUCHAR var_ID,DPULONG bit_index)
{
      DPULONG image_requested_address;
      if (var_ID == Header_ID)
            current_block_address = 0;
      else
dp_get_data_block_address(var_ID);

    if ((current_block_address ==0) && (var_ID != Header_ID))
    {
        return_bytes = 0;
        return NULL;
    }

/* Calculating the relative address of the data block needed within the image */
      image_requested_address = current_block_address + bit_index / 8;

     return_bytes=image_size - image_requested_address;
     return image_buffer+image_requested_address;
}
```

Systems with Indirect Access to the Data File

These systems access programming data indirectly via a paging mechanism. Paging is a method of copying a certain range of data from the memory containing the data file and pasting it into a limited size memory buffer that DirectC can access.

To implement paging:

1. Enable the USE_PAGING compiler option. See "Compiler Switches" on page 18.
2. Define *Page_buffer_size*. The recommended minimum buffer size is 16 bytes for efficiency purposes. If eNVM encrypted programming support is required on SmartFusion or Fusion devices, two buffers are needed of Page_buffer_size. Therefore, the run time memory required must be able to hold 2 x Page_buffer_size.
3. Modify the *dp_get_data* function. Follow these rules for correct operation:
   a. The function must return a pointer to the byte which contains the first bit to be processed.
   b. The function must update *return_bytes* variable which specifies the number of valid bytes in the page buffer.
4. Modify the *dp_get_page_data* function. This function copies the requested data from the external memory device into the page buffer. See "Data File Bit Orientation" on page 37 for additional information. Follow these rules for correct operation:
   – Fill the entire page unless the end of the image is reached. See "Data File Format" on page 30.
   – Update *return_bytes* to reflect the number of valid bytes in the page.

DirectC programming functions call *dp_get_data* function every time access to a data block within the image data file is needed. The *dp_get_data* function calculates the relative address location of the requested data and checks if it already exists in the current page data. The paging mechanism is triggered if the requested data is not within the page buffer.

Example of dp_get_data Function Implementation

```
DPUCHAR* dp_get_data(DPUCHAR var_ID,DPULONG bit_index)
{
      DPULONG image_requested_address;

      DPULONG page_address_offset;

      if (var_ID == Header_ID)
            current_block_address = 0;
      else
            dp_get_data_block_address(var_ID);

    if ((current_block_address ==0) && (var_ID != Header_ID))
    {
        return_bytes = 0;
        return NULL;
    }

      /* Calculating the relative address of the data block needed within the image */
      image_requested_address = current_block_address + bit_index / 8;

      /* If the data is within the page, adjust the pointer to point to the particular
element requested */
      if ((image_requested_address >= start_page_address) && (image_requested_address
<= end_page_address))
      {
            page_address_offset = image_requested_address - start_page_address;
            return_bytes = end_page_address - image_requested_address + 1;
      }
      /* Otherwise, call dp_get_page_data which would fill the page with a new data
block */
      else
      {
            dp_get_page_data(image_requested_address);
            page_address_offset = 0;
      }
      return &page_global_buffer[page_address_offset];
}
```

Example of dp_get_page_data Function Implementation

dp_get_page_data is the only function that must interface with the communication peripheral of the image data file. Since the requested data blocks may not be contiguous, it must have random access to the data blocks. Its purpose is to fill the page buffer with valid data.

In addition, this function must maintain *start_page_address* and *end_page_address* that contain the range of data currently in the page.

dp_get_page_data takes two arguments:

- address_offset - Contains the relative address of the needed element within the data block of the image file.

- *return_bytes* - Points to the *return_bytes* variable that should be updated with the number of valid bytes available.

```
void dp_get_page_data(DPULONG image_requested_address)
{
      DPULONG image_address_index;
      start_page_address=0;

      image_address_index=image_requested_address;
      return_bytes = PAGE_BUFFER_SIZE;
      if (image_requested_address + return_bytes > image_size)
            return_bytes = image_size - image_requested_address;

      while (image_address_index < image_requested_address + return_bytes)
      {
            page_global_buffer[start_page_address]=image_buffer[image_address_index];
            start_page_address++;
            image_address_index++;
      }
      start_page_address = image_requested_address;
      end_page_address = image_requested_address + return_bytes - 1;

      return;
}
```

Main Entry Function

The main entry function is *dp_top* defined in *dpalg.c*. It must be called to initiate the programming operation. Prior to calling the *dp_top* function, a global variable Action_code must be assigned a value as defined in *dpalg.h*. Action codes are listed below.

```
    #define DP_DEVICE_INFO_ACTION_CODE              1
    #define DP_READ_IDCODE_ACTION_CODE              2
    #define DP_ERASE_ACTION_CODE                    3
    #define DP_PROGRAM_ACTION_CODE                  5
    #define DP_VERIFY_ACTION_CODE                   6
    #define DP_ENC_DATA_AUTHENTICATION_ACTION_CODE  7
    #define DP_ERASE_ARRAY_ACTION_CODE              8
    #define DP_PROGRAM_ARRAY_ACTION_CODE            9
    #define DP_VERIFY_ARRAY_ACTION_CODE             10
    #define DP_ERASE_FROM_ACTION_CODE               11
    #define DP_PROGRAM_FROM_ACTION_CODE             12
    #define DP_VERIFY_FROM_ACTION_CODE              13
    #define DP_ERASE_SECURITY_ACTION_CODE           14
    #define DP_PROGRAM_SECURITY_ACTION_CODE         15
    #define DP_PROGRAM_NVM_ACTION_CODE              16
    #define DP_VERIFY_NVM_ACTION_CODE               17
    #define DP_VERIFY_DEVICE_INFO_CODE              18
    #define DP_READ_USERCODE_ACTION_CODE            19
    #define DP_PROGRAM_NVM_ACTIVE_ARRAY_CODE        20
    #define DP_VERIFY_NVM_ACTIVE_ARRAY_CODE         21
    #define DP_IS_CORE_CONFIGURED_ACTION_CODE       22

/* Smart Fusion specific actions */
#define DP_PROGRAM_PRIVATE_CLIENTS_ACTION_CODE  23u
#define DP_VERIFY_PRIVATE_CLIENTS_ACTION_CODE   24u
#define DP_PROGRAM_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE  25u
```

```
#define DP_VERIFY_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE   26u
```

The following are the only actions supported on the SmartFusion2 and IGLOO2 families of devices.

```
#define DP_DEVICE_INFO_ACTION_CODE 1
#define DP_READ_IDCODE_ACTION_CODE 2
#define DP_ERASE_ACTION_CODE 3
#define DP_PROGRAM_ACTION_CODE 5
#define DP_VERIFY_ACTION_CODE 6
#define DP_ENC_DATA_AUTHENTICATION_ACTION_CODE 7
```

Note: Programming of individual blocks, such as array only or eNVM only, is not possible with one data file because of how the data is constructed. If you wish to use such a feature you must generate multiple data files.

Data Type Definitions

Microsemi uses *DPUCHAR*, *DPUINT*, *DPULONG*, *DPBOOL*, *DPCHAR*, *DPINT*, and *DPLONG* in the DirectC source code. Change the corresponding variable definition if different data type names are used.

```
/************************************************/
/* DPCHAR    -- 8-bit Windows (ANSI) character */
/*              i.e. 8-bit signed integer      */
/* DPINT     -- 16-bit signed integer          */
/* DPLONG    -- 32-bit signed integer          */
/* DPBOOL    -- boolean variable (0 or 1)      */
/* DPUCHAR   -- 8-bit unsigned integer         */
/* DPUSHORT  -- 16-bit unsigned integer        */
/* DPUINT    -- 16-bit unsigned integer        */
/* DPULONG   -- 32-bit unsigned integer        */
\/************************************************/
typedef unsigned char  DPUCHAR;
typedef unsigned short DPUSHORT;
typedef unsigned int   DPUINT;
typedef unsigned long  DPULONG;
typedef unsigned char  DPBOOL;
typedef          char  DPCHAR;
typedef          int   DPINT;
typedef          long  DPLONG;
```

Supported Actions

Action: DP_DEVICE_INFO_ACTION
Purpose: Displays device security settings and the content of the FROM if note encrypted.

Action: DP_READ_IDCODE_ACTION
Purpose: Reads and displays the content of the IDCODE register.

Action: DP_ERASE_ACTION
Purpose: Erases all supported blocks in the data file.

Action: DP_PROGRAM_ACTION
Purpose: Performs erase, program and verify operations for all the supported blocks in the data file including SmartFusion MSS private clients.

Action: DP_VERIFY_ACTION
Purpose: Performs verify operation for all the supported blocks in the data file including SmartFusion MSS private clients.

Action: DP_ENC_DATA_AUTHENTICATION_ACTION
Purpose: Valid for encrypted array devices and files only. Performs data authentication for the array to make sure the data was encrypted with the same encryption key as the device.

Action: DP_ERASE_ARRAY_ACTION
Purpose: Performs erase operation on the array blocks.

Action: DP_PROGRAM_ARRAY_ACTION
Purpose: Performs erase, program and verify operations on the array block and SmartFusion MSS private clients.

Action: DP_VERIFY_ARRAY_ACTION
Purpose: Performs verify operation on the array block and SmartFusion MSS private clients.

Action: DP_ERASE_FROM_ACTION
Purpose: Performs erase operation on the FROM block.

Action: DP_PROGRAM_FROM_ACTION
Purpose: Performs erase, program and verify operations on the FROM block.

Action: DP_VERIFY_FROM_ACTION
Purpose: Performs verify operation on the FROM block.

Action: DP_ERASE_SECURITY_ACTION
Purpose: Performs erase operation on the security registers.

Action: DP_PROGRAM_FROM_ACTION
Purpose: Performs erase and program operations on the security registers.

Action: DP_PROGRAM_NVM_ACTION
Purpose: Performs program and verify operations on all supported NVM blocks in the data file including SmartFusion MSS private clients.

Action: DP_VERIFY_NVM_ACTION
Purpose: Performs verify operation on all supported NVM blocks in the data file including SmartFusion MSS private clients.

Action: DP_VERIFY_DEVICE_INFO_ACTION
Purpose: Performs verification of the security settings of the device against the data file security setting.

Action: DP_READ_USERCODE_ACTION
Purpose: Reads and displays the device usercode while the FPGA Array remains active.

Action: DP_PROGRAM_NVM_ACTIVE_ARRAY
Purpose: Programs the targeted EFMBs while the FPGA Array remains active including SmartFusion MSS private clients.

Action: DP_VERIFY_NVM_ACTIVE_ARRAY
Purpose: Verifies the targeted EFMBs while the FPGA Array remains active including SmartFusion MSS private clients.

Action: DP_PROGRAM_SECURITY_ACTION
Purpose: Performs erase, and program operation of the security registers.

Action: DP_IS_CORE_CONFIGURED_ACTION_CODE
Purpose: Performs a quick check on the array to determine if the core is programmed and enabled.

Action: DP_PROGRAM_PRIVATE_CLIENTS_ACTION_CODE
Purpose: SmartFusion specific action. This action programs the system boot code as well as initialization clients in smart fusion used by the MSS.

Action: DP_VERIFY_PRIVATE_CLIENTS_ACTION_CODE
Purpose: SmartFusion specific action. This action verifies the system boot code as well as initialization clients in smart fusion used by the MSS.

Action: DP_PROGRAM_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE
Purpose: SmartFusion specific action. This action updates the system boot code as well as initialization clients in smart fusion used by the MSS while the FPGA array remains active.

Action: DP_VERIFY_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE
Purpose: SmartFusion specific action. This action updates the system boot code as well as initialization clients in smart fusion used by the MSS while the FPGA array remains active.

4 – Chain Programming

Chain programming refers to a chain of devices (from various vendors) connected together serially through a JTAG port. When devices are joined together in a JTAG chain, all of their Instruction Registers (IR) and Data Registers (DR) are put in a long shift register from TDI to TDO. The IR length differs from device to device and the DR length depends on the instruction that shifts into the instruction register.

Pre/Post Data Variable Declaration

The pre/post data variable declaration variables are initialized and used in the *dpchain.c* file. Their default values are 0s. You do not need to change these values if you are programming a standalone device. However, you must correctly set these variables if you are programming Microsemi devices in a daisy chain.

The variables that must be set are defined in dpchain.c and are listed below.

```
DPUINT dp_preir_length = PREIR_LENGTH_VALUE;
DPUINT dp_predr_length = PREDR_LENGTH_VALUE;
DPUINT dp_postir_length = POSTIR_LENGTH_VALUE;
DPUINT dp_postdr_length = POSTDR_LENGTH_VALUE;
```

These variables are used to hold the pre and post IR and DR data. `DPUCHAR`

```
dp_preir_data[PREIR_DATA_SIZE];
DPUCHAR dp_predr_data[PREDR_DATA_SIZE];
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE];
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE];

PREIR_DATA_SIZE = (dp_preir_length + 7) / 8;
PREDR_DATA_SIZE = (dp_predr_length + 7) / 8;
POSTIR_DATA_SIZE = (dp_postir_length + 7) / 8;
POSTDR_DATA_SIZE = (dp_postdr_length + 7) / 8;
```

In the example below, the devices sitting in a chain between the need-programming A3P device and the TDO of the programming header are called pre-devices. The devices between the need-programming A3P device and the TDI of the programming header are called post-devices. In Figure 4-1, devices one and two are pre-devices, devices four, five and six are post-devices, and the A3P 3 is the device that is programmed.

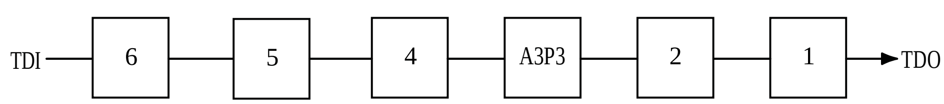

Figure 4-1 **• Devices in the Chain**

If there are N1 pre-devices and N2 post-devices in a chain, L1 is the sum of IR lengths of all the pre-devices. L2 is the sum of IR lengths of all the post-devices. Table 4-2 is an example of how to set the values for the dpchain.c file using the variables assuming the values shown in Table 4-1.

Table 4-1 **• Device IR Length**

Device	IR Length
Dev 1	5
Dev 2	8

Table 4-1 • **Device IR Length (continued)**

Device	IR Length
Dev 3	8
Dev 4	3
Dev 5	12
Dev 6	5

L1 = 5 + 8 = 13

L2 = 3 + 12 + 5 = 20

Table 4-2 • **Example Variable Values for dpchain.c File**

Pre/Post Data Values	Comments
#define PREIR_LENGTH_VALUE 13	L1
#define PREDR_LENGTH_VALUE 2	N1
#define POSTIR_LENGTH_VALUE 20	L2
#define POSTDR_LENGTH_VALUE 3	N2
#define PREIR_DATA_SIZE 2	Number of bytes needed to hold L1
#define PREDR_DATA_SIZE 1	Number of bytes needed to hold N1
#define POSTIR_DATA_SIZE 3	Number of bytes needed to hold L2
#define POSTDR_DATA_SIZE 1	Number of bytes needed to hold N2

Initialize the following arrays as follows for this particular example:

```
DPUCHAR dp_preir_data[PREIR_DATA_SIZE]={0xff,0x1f};
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x3};
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0xff,0xff,0xf};
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x1f};
```

Note: Chain programming does not support programming multiple devices simultaneously. Instead, it is a method to communicate with one device to perform programming. All other devices must be placed in bypass mode, as implemented in the above example.

5 – Data File Format

DAT File Description for AGL, AFS, A3PL, A3PEL, A3P/E, and A2F Devices

The AGL / AFS / A3PL / A3PEL / A3P/3 A2F data file contains the following sections:

- **Header Block** - Contains information identifying the type of the binary file, data size blocks, target device ID and different flags needed in the DirectC code to identify which block is supported and its associated options.
- **Data Lookup Table** - Contains records identifying the starting relative location of all the different data blocks used in the DirectC code and data size of each block. The format is described in Table 5-1.
- **Data Block** - Contains the raw data for all the variables specified in the lookup table.

Table 5-1 **• DAT Image Description**

Header Section of DAT File	
Information	# of Bytes
Designer Version Number	24
Header Size	1
Image Size	4
Data Compression Flag	1
M1/P1/M7 Flag	1
Target Device ID	4
Tools Version Number	2
Map Version Number	2
Core Support Flag	1
FORM Support Flag	1
NVM Support Flag	1
NVM Block 0 Support Flag	1
NVM Block 1 Support Flag	1
NVM Block 2 Support Flag	1
NVM Block 3 Support Flag	1
NVM Verify Support Flag	1
PASS Key Support Flag	1
AES Key Support Flag	1

Table 5-1 **• DAT Image Description (continued)**

Header Section of DAT File	
Core Encryption Flag	1
FROM Encryption Flag	1
NVM Block 0 Encryption Flag	1
NVM Block 1 Encryption Flag	1
NVM Block 2 Encryption Flag	1
NVM Block 3 Encryption Flag	1
Device Exception Flag	2
ID Mask	4
SD Tiles	1
Mapped rows	2
BSR Length	2
SE Wait	1
Dual Key Support Flag	1
Number of DirectC data blocks in file	1
Look Up Table	
Information	# of Bytes
Data Identifier # 1	1
Pointer to data 1 memory location in the data block section	4
# of bytes of data 1	4
Data Identifier # 2	1
Pointer to data 2 memory location in the data block section	4
# of bytes of data 2	4
Data Identifier # x	1
Pointer to data x memory location in the data block section	4
# of bytes of data x	4
Data Block	
Information	# of Bytes
Binary Data	Variable
CRC of the entire image	2

DAT File Description for M2GL and M2S Devices

The MGL and M2S data file contains the following sections:

- **Header Block** - Contains information identifying the type of the binary file and data size blocks.
- **Constant Data Block** - Includes device ID, silicon signature and other information needed for programming.
- **Data Lookup Table** - Contains records identifying the starting relative location of all the different data blocks used in the DirectC code and data size of each block. The format is described in Table 5-2.
- **Data Block** - Contains the raw data for all the different variables specified in the lookup table.

***Table 5-2* • DAT Image Description**

Header Section of DAT File	
Information	# of Bytes
Designer Version Number	24
Header Size	1
Image Size	4
DAT File Version	1
Tools Version Number	2
Map Version Number	2
Feature Flag	2
Device Family	1
Constant Data Block	
Device ID	4
Device ID Mask	4
Silicon Signature	4
Checksum	2
Number of BSR Bits	2
Number of Components	2
Data Size	2
Erase Data Size	2
Verify Data Size	2
ENVM Data Size	2
ENVM Verify Data Size	2
Number of Records	1
Look Up Table	
Information	# of Bytes

Table 5-2 • **DAT Image Description (continued)**

Header Section of DAT File	
Data Identifier # 1	1
Pointer to data 1 memory location in the data block section	4
# of bytes of data 1	4
Data Identifier # 2	1
Pointer to data 2 memory location in the data block section	4
# of bytes of data 2	4
Data Identifier # x	1
Pointer to data x memory location in the data block section	4
# of bytes of data x	4
Data Block	
Information	# of Bytes
Binary Data	Variable
CRC of the entire image	2

6 – Source File Description

DPUSER.C and DPUSER.H

These files contain hardware interface functions and require user modification.

DPCOM.C and DPCOM.H

These files contain memory interface functions.

DPALG.C and DPALG.H

dpalg.c contains the main entry function *dp_top*.

dpalg.h contains definitions of all the STAPL actions and their corresponding codes.

DPG3ALG.C and DPG3ALG.H

dpG3alg.c contains the main entry function dp_top_g3 and all other functions common to AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families.

dpG3alg.h contains compile options specific to AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families.

DPCORE.C and DPCORE.H

Files that contain the specific functions to support array erase, program and verify actions of AGL, AFS, A3PL, A3PEL, A3P/E and A2F families.

DPFROM.C and DPFROM.H

Files that contain the specific functions to support FROM erase, program and verify actions of AGL, AFS, A3PL, A3PEL, A3P/E and A2F families.

DPNVM.C and DPNVM.H

Files that contain the specific functions to support NVM program and verify actions of AFS and A2F families.

DPSECURITY.C and DPSECURITY.H

Files that contain the specific functions to support security erase, program actions of AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families.

DPG4ALG.C and DPG4ALG.H

dpG4alg.c contains the main entry function dp_top_g4 and all other functions common to M2S and MGL families.

DPJTAG.C and DPJTAG.H

The JTAG related functions are declared in dpjtag.h and implemented in *dpjtag.c*.

DPCHAIN.C and DPCHAIN.H

Files that contain the specific functions to support chain programming.

dpchain.c contains pre- and post-IR/DR data definition to support chain programming.

DPUTIL.C and DPUTIL.H

These files contain utility functions needed in the DirectC code.

7 – Disabled Features with ENABLE_CODE_SPACE_OPTIMIZATION

DMK Verification for ARM Enabled Devices

This feature identifies whether the target device is an M1, M7, or P1

device. Affected devices: ARM enabled devices

Impact if removed - DirectC will be unable to identify if the device is standard Fusion or ARM enabled device. DirectC still support programming; however, it relies on the data file processing the target device as an ARM enabled device.

030/015 Device Check

This feature identifies if the target device is a 015 or 030 device; needed to prevent the wrong design from being programmed into the device.

Affected devices: A3P and AGL 015 / 030 device

Impact if removed: If the design does not match the target device, programming may pass, but the device may not function

8 – Data File Bit Orientation

This chapter specifies the data orientation of the binary data file generated by the Libero software. DirectC implementation must be in sync with the specified data orientation. Table 8-1 illustrates how the data is stored in the binary data file. See "Data File Format" on page 30 for additional information on the data file.

Table 8-1 **• Binary Data File Example**

Byte O	**Byte 1**	**Byte 2**	**Byte 3**	**..**	**..**	**Byte N**
Bit7..Bit0	Bit15..Bit8	Bit23..Bit16	Bit35..Bit24	..	..	Bit(8N+7)..Bit(8N)
Valid Data	Valid Data	Valid Data	Valid Data	..	..	o <-Valid Data

If the number of bits in a data block is not a multiple of eight, the rest of the most significant bits (msb) in the last byte are filled with zeros. The example below shows a given 70-bit data to be shifted into the target shift register from the least significant bit (lsb) to the most significant bit (msb). A binary representation of the same data follows.

20E60A9AB06FAC78A6	tdi
10000011100110 00001010100110101011000001101111101011000111100010100110 tdi	
Bit 69	Bit 0

This data is stored in the data block section. Table 8-2 shows how the data is stored in the data block.

Table 8-2 **• Data Block Section Example**

Byte O	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**	**..**	**Byte 8**
Bit7...Bit0	Bit15..Bit8	Bit23..Bit16	Bit35..Bit24	Bit43..Bit36	..	Bit71..Bit64
10100110	01111000	10101100	01101111	10110000		00100000
A6	78	AC	6F	B0		20

9 – Error Messages & Troubleshooting Tips

The information in this chapter may help you solve or identify a problem when using DirectC code. If you have a problem that you cannot solve, visit the Microsemi website at http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support or contact Microsemi Customer Technical Support at tech@microsemi.com or call our hotline at 1-800-262-1060.

See Table 9-1 for a description of exit codes and their solutions.

Table 9-1 **• Exit Codes**

Exit Code	Error Message	Action/Solution
0	This code does not indicate an error.	This message indicates success.
6	JEDEC standard message. The IDCODE of the target device does not match the expected value in the DAT file image.	Possible Causes: - The data file loaded was compiled for a different device. Example AFS250 DAT file loaded to program AFS600 device. - Device TRST pin is grounded. - Noise or reflections on one or more of the JTAG pins causing incorrect read-back of the IR Bits. Solution: - Choose the correct DAT file for the target device. - Measure JTAG pins and noise or reflection. TRST should be floating or tied high. - Cut down the extra length of ground connection.
8	This message occurs when the FPGA failed during the Erase operation.	Possible Causes: - The device is secured, and the corresponding data file is not loaded. The device has been permanently secured and cannot be unlocked. Solution: - Load the correct DAT file.
10	Failed to program FlashROM.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
11	The message occurs when the FPGA failed verify.	Possible Cause: - The device is secured, and the corresponding DAT file is not loaded. - The device is programmed with an incorrect design. Solution: - Load the correct DAT file. - Check Vpump level. - Measure JTAG pins and noise or reflection.
14	Failed to program Silicon Signature.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.

Microsemi

Table 9-1 **• Exit Codes (continued)**

Exit Code	Error Message	Action/Solution
18	Failed to authenticate the encrypted data.	- Make sure the AES key used to encrypt the data matches the AES key programmed in the device.
20	Failed to verify FlashROM at row ###.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection. -Make sure the device is programmed with the correct design.
22	Failed to program pass key.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
23	Failed to program AES key.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
24	Failed to program UROW.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection. - Make sure you mounted 0.01uF and 0.33ufF caps on Vpump (close to the pin).
25	Failed to enter programming mode.	- Try programming with a new device. - Measure JTAG pins and noise or reflection.
27	FlashROM Write/Erase is protected by the pass key. A valid pass key needs to be provided.	- Provide a data file with a pass key.
30	FPGA Array verification is protected by a pass key. A valid pass key needs to be provided.	- Provide a data file with a valid pass key.
31	Failed to program DMK.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
33	FPGA Array encryption is enforced. Plain text programming is prohibited.	Provide a data file with an encrypted FPGA Array.
34	FlashROM encryption is enforced. Plain text programming is prohibited.	- Provide a data file with an encrypted FlashROM.
35	Pass key match failure.	- Provide a data file with correct pass key.
36	FlashROM Encryption is not enforced. AES key may not be present in the target device. Unable to proceed with Encrypted FlashROM programming.	- Make sure the device is properly secured with AES encryption protection turned on. - Provide correct DAT file for programming.
37	FPGA Array Encryption is not enforced. Cannot guarantee valid AES key present in target device. Unable to proceed with Encrypted FPGA Array programming.	- Make sure the device is properly secured with the AES encryption protection turned on for FPGA Array. - Provide the correct data file for programming.

Microsemi

Table 9-1 **• Exit Codes (continued)**

Exit Code	Error Message	Action/Solution
38	Failed to program pass key.	- Check that the device is not already secured with a different pass key. - Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
39	Failed the Embedded Flash Block verification.	- Check that the device is not read secured already with a different pass key. - Measure JTAG pins and noise or reflection.
41	Failed to program Embedded Flash Block.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
42	User lock bits do not match the lock bits in the data file.	Provide a data file with the correct lock bits data.
43	User urow information does not match the urow information in the data file.	Provide a data file with the correct urow information data.
47	NVM encryption is enforced. Plain text programming is prohibited.	Provide a data file with an encrypted NVM.
49	NVM encryption is not enforced. Cannot guarantee valid AES key present in target device. Unable to proceed with encrypted NVM programming.	- Make sure the device is properly secured with the AES encryption protection turned on for NVM. - Provide the correct data file for programming.
100	CRC data error. Data file is corrupted or programming on system board is not successful.	- Regenerate data file. - Reprogram data file into system memory.
150	Request action is not found.	Check spelling.
151	Action is not supported because required data block is missing from the data file.	Regenerate STAPL/DAT file with the needed block/ feature support.
152	Compiled code does not support the requested action.	Compile DirectC code with the appropriate compile options enabled.
153	Data file contains data for the protected portion of NVM0 block.	Regenerate the data file from the latest Designer software.
154	Device security settings do not match with the data file.	Regenerate the data file with the correct device security settings.

A – 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

For Microsemi SoC Products Support, visit http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://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. Visit About Us for sales office listings and corporate contacts.

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@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.

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs, and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,600 employees globally. Learn more at **www.microsemi.com**.

Microsemi Corporate Headquarters

One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

5-13-00109-1/10.15