UG0680
User Guide
PolarFire FPGA Fabric

a Microchip company

a MICROCHIP company

Microsemi 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
Email: sales.support@microsemi.com
www.microsemi.com

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided “as is, where is” and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center 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; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

Contents

Figures

Tables

1 Revision History

1.1 Revision 7.0

The following is a summary of changes made in this revision.

- Updated Figure 15, page 20 to correct the output data values in the Pipeline Mode.
- Removed collision prevention from Table 4, page 14.
- Updated Table 8, page 19 for 1K x 16 mode.
- Added information about how RAM blocks are cascaded when Write byte Enables option is selected. See, Dual-Port Large SRAM Configurator, page 29 and Two-Port LSRAM Configurator, page 35.
- Updated µPROM Operation, page 51 to mention that µPROM memory file supports only the plain text file.
- Updated Figure 11, page 17 and Figure 17, page 23 to show that A_BYPASS and B_BYPASS signals are control signals of the MUX.
- Removed Simple Write, Feed-Through Write, and Read-Before-Write specific content from two-port LSRAM. These write operations are not supported in the two-port LSRAM configuration.

1.2 Revision 6.0

Updated ECC Mode (For x33 Two-Port Mode Only), page 26.

1.3 Revision 5.0

The following is a summary of the changes in this revision.

- Structural changes were made throughout the document.
- Information about PolarFire LSRAM, µSRAM, µPROM, and sNVM Features were updated. See, Table 2, page 10.
- Math Block Features were updated. See, Features, page 62.
- Libero SoC PolarFire Compile Report is moved to appendix. See, Libero SoC Compile Report, page 103.

1.4 Revision 4.0

Updated the Math Blocks Resources. For more information, see Table 1, page 9.

1.5 Revision 3.0

Revision 3.0 of this document is updated to include features and enhancements introduced in Libero SoC PolarFire v2.0.

1.6 Revision 2.0

The following is a summary of the changes made in this revision.

- Added reference to the ChipPlanner user guide. For more information, see Fabric X-Y Coordinates, page 13
- Added reference to the Synplify Pro RAM block application note for LSRAM. For more information, see RTL Inference during Synthesis, page 28.
- Added reference to the Synplify Pro RAM block application note for µSRAM. For more information, see RTL Inference during Synthesis, page 44.
- Added reference to the Synplify Pro MACC block application note for MACC. For more information, see RTL Inference, page 73.

1.7 Revision 1.0

The first publication of this document.

2 Overview

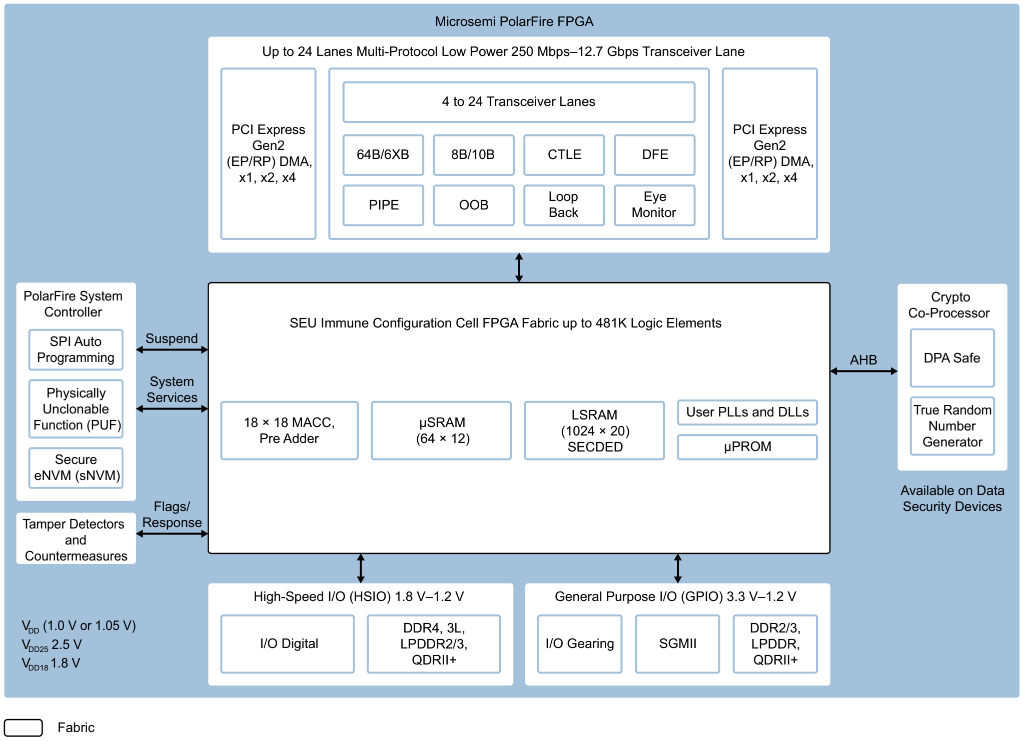
PolarFire® FPGAs are the fifth-generation family of 28nm non-volatile SoC FPGA devices from Microchip.

The PolarFire FPGA fabric consists of the following resources:

- **Logic element**—these are basic building blocks in the PolarFire FPGA.
- **Embedded memory blocks**—these include large SRAM (LSRAM), microSRAM (µSRAM), microPROM (µPROM), and secure non-volatile memory (sNVM)[1].
- **Math blocks (MACC)**—these have a built-in multiplier, pre-adder, and adder.

Microchip's Libero® SoC Design Suite provides LSRAM, µSRAM, µPROM, and Math IP blocks. Figure 1 shows the top-level block diagram of the PolarFire FPGA.

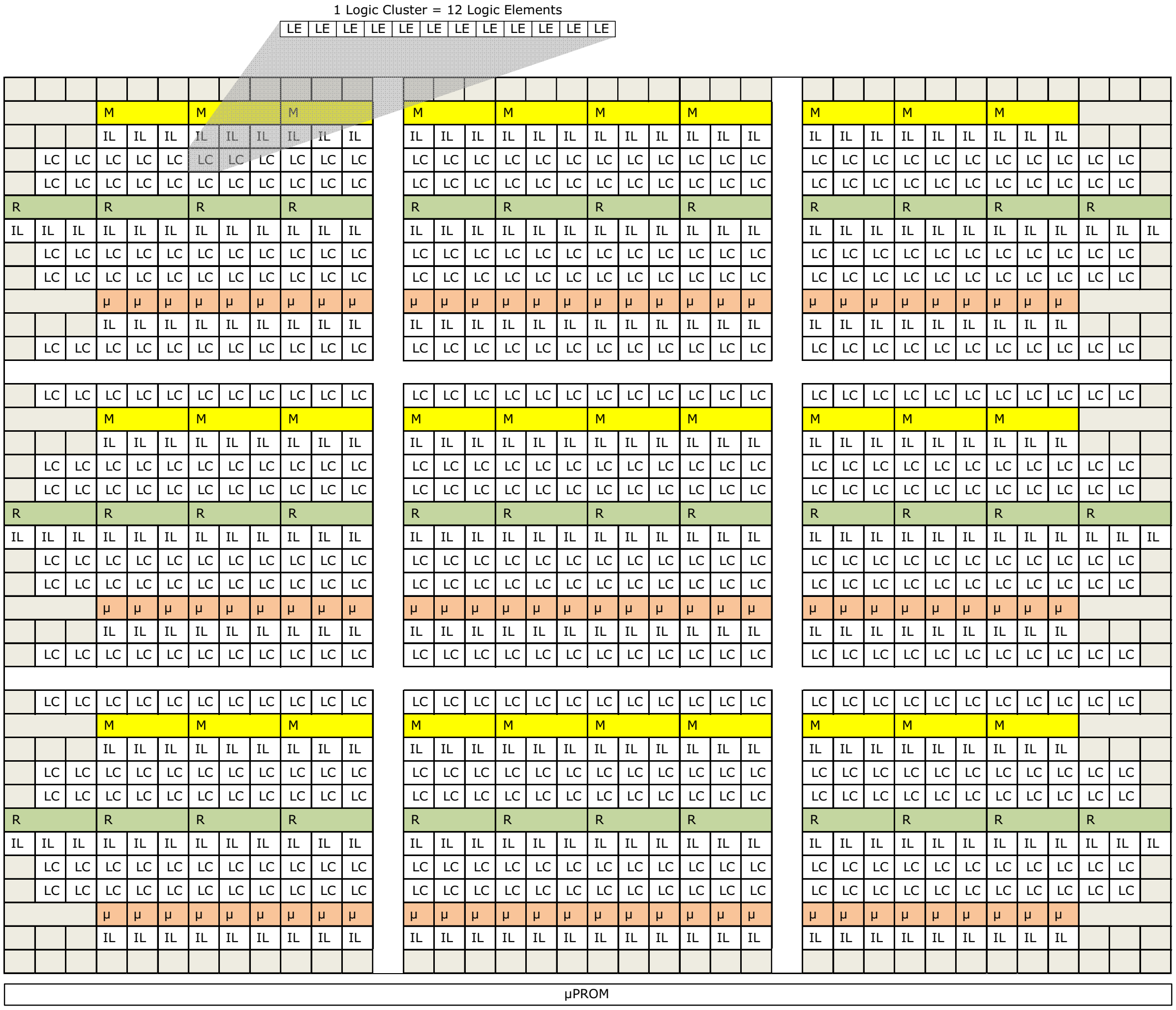
Figure 1 • **PolarFire FPGA Block Diagram**


Microsemi PolarFire FPGA
Up to 24 Lanes Multi-Protocol Low Power 250 Mbps–12.7 Gbps Transceiver Lane
PCI Express Gen2 (EP/RP) DMA, x1, x2, x4
4 to 24 Transceiver Lanes
64B/6XB
8B/10B
CTLE
DFE
PIPE
OOB
Loop Back
Eye Monitor
PCI Express Gen2 (EP/RP) DMA, x1, x2, x4
PolarFire System Controller
SPI Auto Programming
Physically Unclonable Function (PUF)
Secure eNVM (sNVM)
Tamper Detectors and Countermeasures
Suspend
System Services
Flags/ Response
SEU Immune Configuration Cell FPGA Fabric up to 481K Logic Elements
18 × 18 MACC, Pre Adder
µSRAM (64 × 12)
LSRAM (1024 × 20) SECDED
User PLLs and DLLs
µPROM
AHB
Crypto Co-Processor
DPA Safe
True Random Number Generator
Available on Data Security Devices
V_{DD} (1.0 V or 1.05 V)
V_{DD25} 2.5 V
V_{DD18} 1.8 V
High-Speed I/O (HSIO) 1.8 V–1.2 V
I/O Digital
DDR4, 3L, LPDDR2/3, QDRII+
General Purpose I/O (GPIO) 3.3 V–1.2 V
I/O Gearing
SGMII
DDR2/3, LPDDR, QDRII+
Fabric

The fabric layout is shown in Figure 2. The FPGA logic resources are displayed as Logic Clusters (LC) and Interface Logic (IL). Each LC and IL consists of 12 Logic Elements (LE). The embedded memory blocks and math blocks are arranged in rows.

[1] sNVM is part of PolarFire system controller.

Figure 2 • **Fabric Layout**


1 Logic Cluster = 12 Logic Elements
LE LE LE LE LE LE LE LE LE LE LE LE
M
IL
LC
R
µ
µPROM

LC – Logic cluster
IL – Interface logic
M – Math block
R – LSRAMs
µ – µSRAMs

The following table lists the available fabric resources in the PolarFire family devices.

Table 1 • **Fabric Resources for PolarFire FPGA Devices**

Resources	MPF100	MPF200	MPF300	MPF500
Logic elements	71,736	127,896	198,744	319,992
Interface logic	36,864	64,512	100,800	161,280
Total logic	108,600	192,408	299,544	481,272
LSRAM blocks (20 Kb each)	352	616	952	1,520
Total LSRAM bits (Mb)	6.87	12.03	18.59	29.69
µSRAM blocks (768 bits each)	1,008	1,764	2,772	4,440
Total µSRAM bits (Mb)	0.74	1.29	2.03	3.25
Total RAM (Mb)	7.6	13.32	20.62	32.94
Math blocks (18 × 18 MACC)	336	588	924	1,480
µPROM (Kb)	297	297	459	513

Note: 1 Kb = 1024 bits, 1 Mb = 1024 Kb.

3 Logic Element and Routing

The PolarFire fabric includes an array of logic elements grouped in clusters connected by hierarchical routing structures. These clusters are arranged in rows and are used to implement sequential and combinational logic (Figure 9). Table 2 lists the available logic elements in PolarFire devices.

Table 2 • **Number of Logic Elements (PolarFire)**

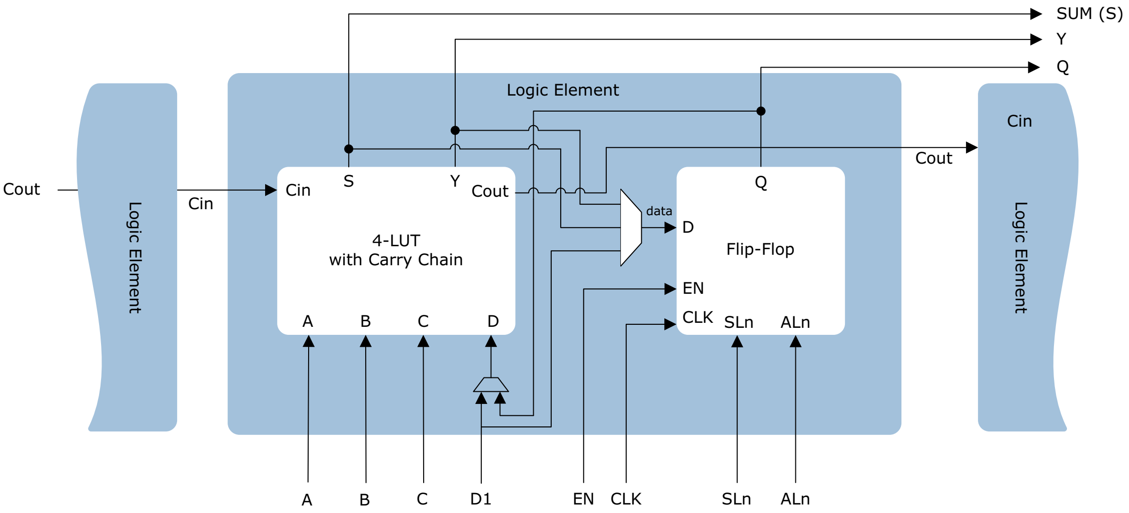
Resources	MPF100	MPF200	MPF300	MPF500
Logic elements	71,736	127,896	198,744	319,992
Interface logic	36,864	64,512	100,800	161,280
Total logic	108,600	192,408	299,544	481,272

Note: The part number lists the approximate number of logic elements that are in the part. For example, 300 in the MPF300 part number notates that there are approximately 300,000 logic elements in the MPF300 device.

3.1 Logic Element

The logic element consists of a 4-input Lookup Table (LUT) with a carry chain and D-type flip-flop, as shown in Figure 3. The logic element is fracturable, which means the LUT can be independently used without flip-flop, or flip-flop can be used without LUT.

Figure 3 • **Functional Block Diagram of Logic Element**



The 4-input LUT with carry chain can be configured to implement any 4-input combinational logical function or arithmetic function. The 4-input LUT generates the output (Y) depending on the four inputs—A, B, C, and D. The carry chain is implemented using a 3-bit carry-look-ahead circuit. This circuit is connected between various logic elements by carry chain input (Cin) signal and carry chain output (Cout) signal. When the LUT is used to implement arithmetic functions, the carry chain input (Cin) is used with LUT output to generate the sum (S) output. However, for non-arithmetic functions, the sum (S) output can still be used as an output along with the other output (Y).

The D-type flip-flop can be used as a register or latch. The data input (D) of the D-type flip-flop can be sourced from one of three inputs: the direct input (D1), the combinational output (Y) of the LUT, or the sum output (S) of the LUT (Figure 3). The ALn and SLn are asynchronous load and synchronous load active low signals that can be configured as reset signal. The flip-flop output (Q) can be an output of the logic element or one of the data inputs to the 4-input LUT (inside the same logic element).

3.2 Interface Logic

The embedded hard IP blocks (LSRAM, µSRAM, and Math blocks) are connected to the fabric through Interface Logic (ILs).

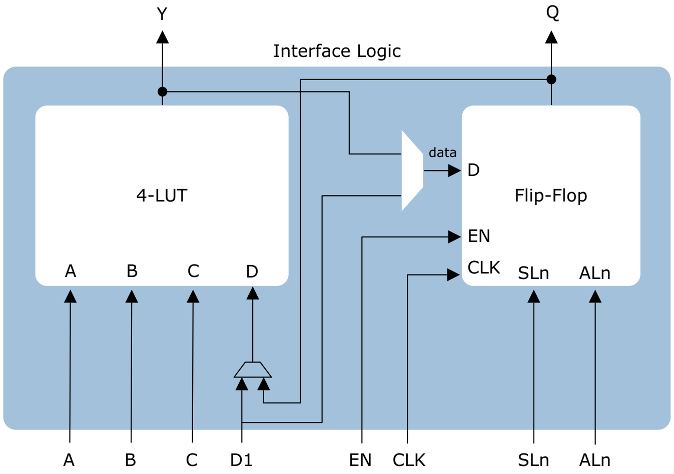
Table 3 lists the total number of ILs associated with each memory block in PolarFire devices.

Table 3 • **ILs for Embedded Hard IP Blocks (PolarFire)**

	MPF100		**MPF200**		**MPF300**		**MPF500**	
Resources	**Number of Blocks**	**Number of ILs**	**Number of Blocks**	**Number of ILs**	**Number of Blocks**	**Number of ILs**	**Number of Blocks**	**Number of ILs**
LSRAM	352	12,672	616	22,176	952	34,272	1,520	54,720
µSRAM	1008	12,096	1,764	21,168	2,772	33,264	4,440	53,280
Math block	336	12,096	588	21,168	924	33,264	1,480	53,280
Total interface logic		36,864		64,512		100,800		161,280

These ILs are structurally similar to LEs with a 4-input LUT and D-type flip-flop, but without a dedicated carry chain, as shown in Figure 4.

Figure 4 • **Functional Block Diagram of Interface Logic**



Each LSRAM and Math block is associated with 36 ILs, and each μSRAM is associated with 12 ILs. For more information, see Figure 5, Figure 6, and Figure 7.

If an embedded hard IP block is used in a design, the associated ILs connect the ports of the embedded hard IP blocks to the fabric routing. Any IL that is not utilized by an embedded hard IP block is automatically available for user logic.

Figure 5 • **LSRAM Interfacing with ILs in a Row**

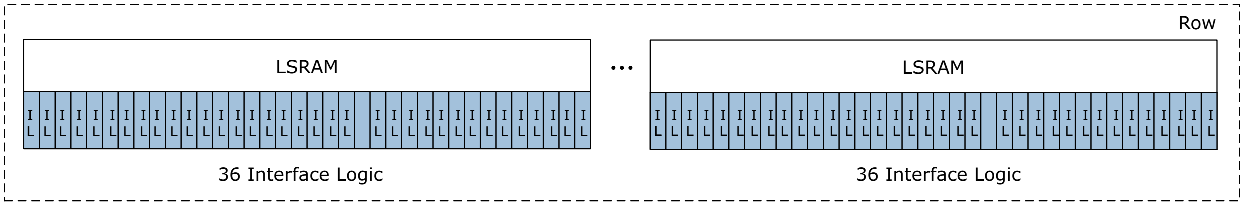


Figure 6 • **Math Block Interfacing with ILs in a Row**

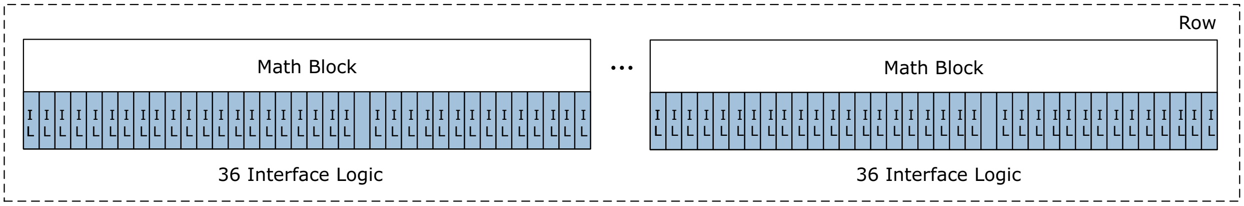
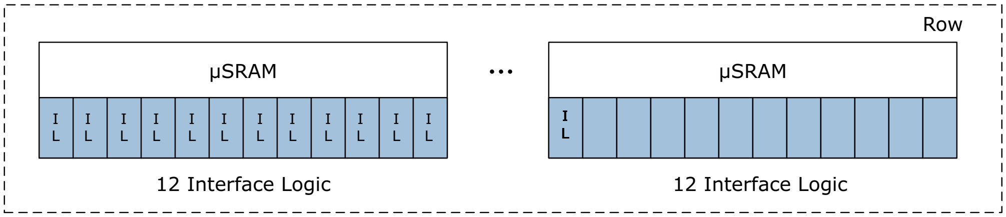


Figure 7 • **μSRAMs Interfacing with ILs in a Row**

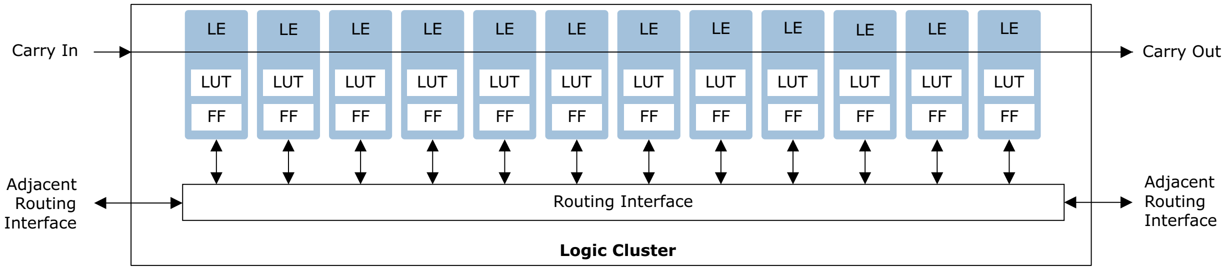


3.3 Logic Cluster

The logic elements in the FPGA fabric are organized in clusters. A logic cluster is a group of 12 LEs. Each logic cluster is connected by a routing interface that connects to its associated LEs and the adjacent routing interfaces.

Figure 8 shows the logic cluster with its routing interface.

Figure 8 • **Logic Cluster**



3.4 Routing Architecture

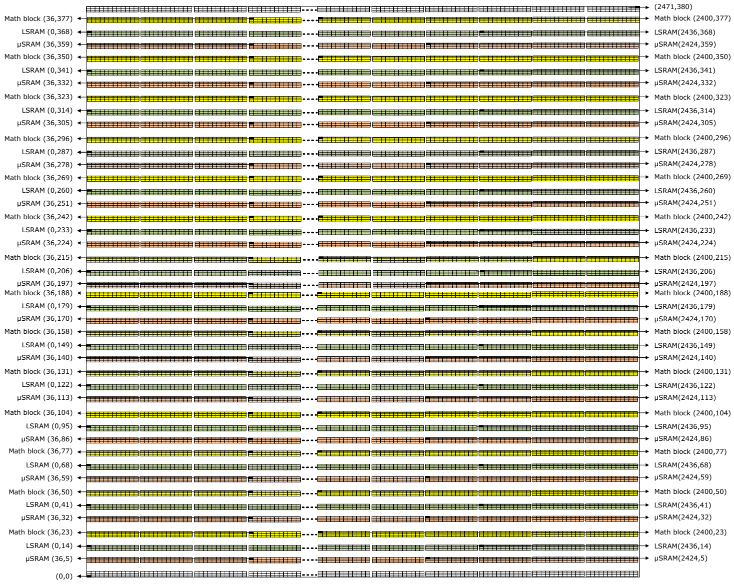
The PolarFire devices support two types of routings—intra-cluster routing and inter-cluster routing. The intra-cluster routing connects the LEs within a cluster, and inter-cluster routing connects LEs between multiple clusters. The intra-cluster routing has lower propagation delay compared to inter-cluster routing. When connecting the adjacent clusters, inter-cluster routing also has additional short routing connections for faster routing.

3.5 Fabric X-Y Coordinates

Each 4-input LUT, D-type flip-flop, carry chain, LSRAM, µSRAM, and math block has individual X-Y coordinates. For manual placement of these blocks, it is possible to set region constraints using these coordinates. The coordinates are measured from the lower left (0, 0) to the top right corner (X, Y); where X, Y values vary for each device. For more information about using coordinates for region/placement constraints, see the *SmartTime User Guide*, *I/O Editor User Guide*, and the *ChipPlanner User Guide*.

Figure 9 shows the available X-Y coordinates of LSRAM, µSRAM, and math block for placement constraints.

Figure 9 • **MPF300 Fabric X-Y Coordinates**



4 Embedded Memory Blocks

The PolarFire FPGA device has the following memory blocks:

- **LSRAM**—The embedded large SRAM blocks are 20Kbits each with 20-bit width and a depth of 1024 locations. The LSRAMs can be configured as either dual port or two port memories. The number of LSRAMs available in a device varies as shown in Table 1. The LSRAMs support ECC when configured in 33-bit data width in two-port mode. The LSRAMs can be configured in various modes as shown in Table 4. LSRAMs can be initialized with user data during power-up. For more information about initialization, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide*.
- **µSRAM**—The embedded 768-bit SRAM blocks (RAM64x12) are arranged in multiple rows within the fabric and can be accessed through the fabric routing architecture. The number of available µSRAM blocks depends on the specific PolarFire device, as shown in Table 1. µSRAMs can be initialized during power-up. For more information about initialization, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide*.
- **µPROM**—The embedded non-volatile PROM is arranged in a single row at the bottom of the fabric and is read only through the fabric interface. µPROM is programmed with the FPGA bitstream during fabric programming and it cannot be programmed independently. µPROM is used to store the initialization data for LSRAM and µSRAM and other user data.
- **sNVM**—Each PolarFire FPGA has 56 KBytes of Secure Non volatile memory (sNVM). The sNVM can be used to initialize LSRAM and µSRAMs with secure data. sNVM can be accessed through the system services. For more information, see *UG0753: PolarFire FPGA Security User Guide*.

The following table lists the LSRAM, µSRAM, and µPROM features.

Table 4 • **PolarFire LSRAM, µSRAM, µPROM, and sNVM Features**

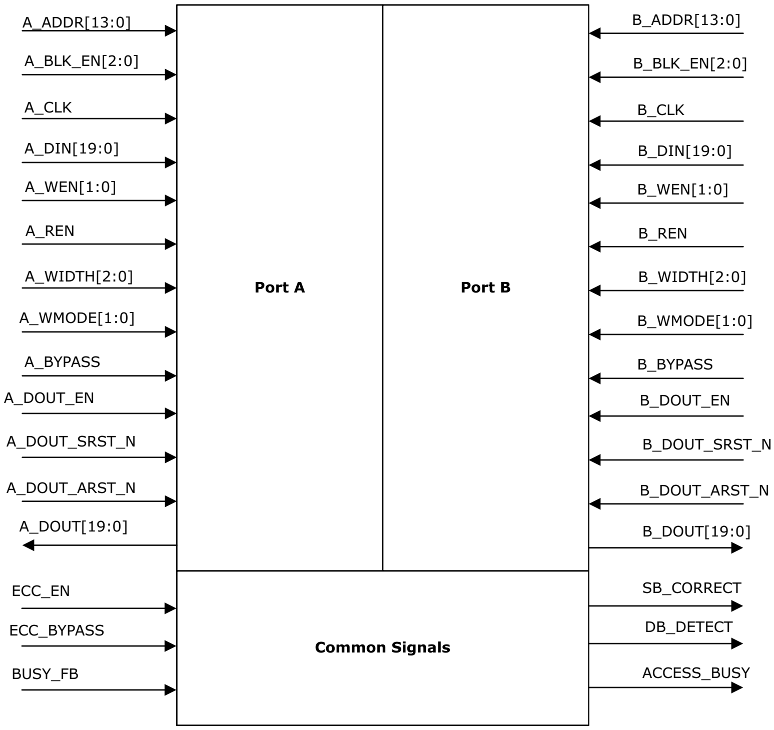
Feature	LSRAM	µSRAM	µPROM	sNVM
Memory size	20,480 bits/block.	768-bit/block.	297 Kbits (MPF100). 297 Kbits (MPF200). 459 Kbits (MPF300). 513 Kbits (MPF500).	56 KBytes
Memory Configuration Options	16K × 1, 8K × 1, 4K × 5, 2K × 10, 1K × 20, 512 × 40[1], and 512 × 33[1] (with ECC).	64 × 12.	Up to 64K × 9.	Not Applicable.
Number of ports	2 read ports, 2 write ports.	1 read port, 1 write port.	1 read port.	Not Applicable.
Memory modes	True dual-port and two-port.	Two-port.	Single-port.	Not Applicable.
Read operation	Synchronous.	Synchronous/Asynchronous.	Asynchronous.	Through system service calls.
Write operation	Simple write, feed-through write, and read-before-write.	Simple write.	Only during device programming.	During device programming and System Service calls.
ECC	Available for two-port mode (512 × 33) only.	Not available.	Not Applicable.	Not Applicable.

[1] ×40 and ×33 are only available in two-port mode.

4.1 LSRAM

Each LSRAM has two independent ports—Port A and Port B, as shown in Figure 10. Both these ports support write and read operations, and can be configured in dual-port mode or two-port mode.

Figure 10 • **LSRAM Input/Output**



Note: When ECC is enabled, if a single-bit error occurs in a word, the data is corrected. If multiple-bit errors occur in a word, the data from the LSRAM is not corrected or modified.

Table 5 lists the ports of LSRAM.

Table 5 • **LSRAM Port List**

Port Name	Direction	Type[1]	Polarity	Description
Port A				
A_ADDR[13:0]	Input	Dynamic		Port A address
A_BLK_EN[2:0]	Input	Dynamic	Active high	Port A block selects
A_CLK	Input	Dynamic	Rising edge	Port A clock
A_DIN[19:0]	Input	Dynamic		Port A write-data
A_DOUT[19:0]	Output	Dynamic		Port A read-data
A_WEN[1:0]	Input	Dynamic	Active high	Port A byte write-enables
A_REN	Input	Dynamic	Active high	Port A read-enable
A_WIDTH[2:0]	Input	Static		Port A width/depth mode select
A_WMODE[1:0]	Input	Static	Active high	Port A read-before-write and feed-through write selects
A_BYPASS	Input	Static	Active low	Port A pipeline register select
A_DOUT_EN	Input	Dynamic	Active high	Port A pipeline register enable
A_DOUT_SRST_N	Input	Dynamic	Active low	Port A pipeline register synchronous-reset
A_DOUT_ARST_N	Input	Dynamic	Active low	Port A pipeline register asynchronous-reset
Port B				
B_ADDR[13:0]	Input	Dynamic		Port B address
B_BLK_EN[2:0]	Input	Dynamic	Active high	Port B block selects
B_CLK	Input	Dynamic	Rising edge	Port B clock
B_DIN[19:0]	Input	Dynamic		Port B write-data
B_DOUT[19:0]	Output	Dynamic		Port B read-data
B_WEN[1:0]	Input	Dynamic	Active high	Port B write-enables (per byte)
B_REN	Input	Dynamic	Active high	Port B read-enable
B_WIDTH[2:0]	Input	Static	Mode select	Port B width/depth
B_WMODE[1:0]	Input	Static	Active high	Port B read-before-write and feed-through write selects
B_BYPASS	Input	Static	Active low	Port B pipeline register select
B_DOUT_EN	Input	Dynamic	Active high	Port B pipeline register enable
B_DOUT_SRST_N	Input	Dynamic	Active low	Port B pipeline register synchronous-reset
B_DOUT_ARST_N	Input	Dynamic	Active low	Port B pipeline register asynchronous-reset
Common Signals				
ECC_EN	Input	Static	Active high	Enable ECC
ECC_BYPASS	Input	Static	Active low	ECC pipeline register select
SB_CORRECT	Output	Dynamic	Active high	Single-bit correct flag

Table 5 • **LSRAM Port List *(continued)***

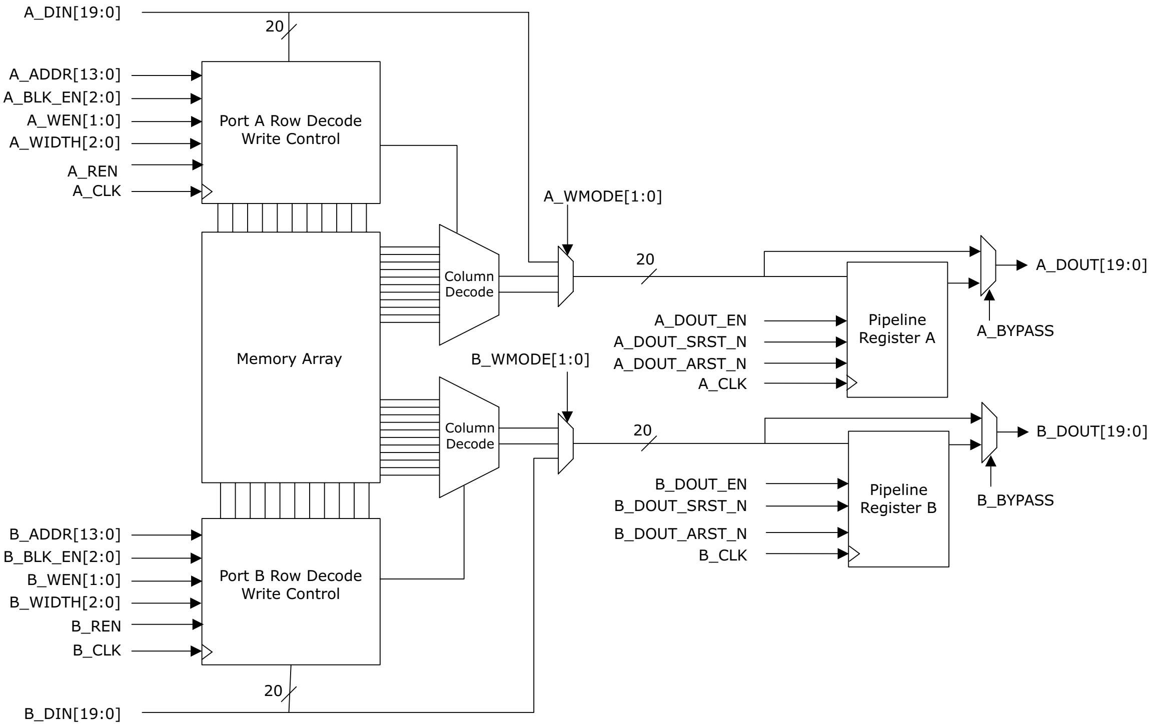
Port Name	Direction	Type[1]	Polarity	Description
DB_DETECT	Output	Dynamic	Active high	Dual-bit error detect flag
BUSY_FB	Input	Static	Active high	Lock access to SmartDebug
ACCESS_BUSY	Output	Dynamic	Active high	Busy signal from SmartDebug

[1] Static inputs are tied to 0 or 1 during design implementation.

4.1.1 Dual-Port Mode

The LSRAM block can be configured as a true dual-port SRAM with independent write and read ports, as shown in Figure 11. Write and read operations can be performed from both ports (A and B) independently at any location as long as there is no write collision. Each port has a unique address, data in, data out, clock, block select, write enable, pipeline registers, and feed-through MUXes. Figure 11 shows the simplified block diagram of LSRAM in dual-port mode.

Figure 11 • **Simplified Functional Block Diagram of LSRAM in Dual-Port Mode**



4.1.1.1 Dual-Port Data Width Configuration

In dual-port mode, both ports A and B have maximum data width of x20. Each port can be configured in multiple data widths. The configuration of one port has a corresponding configuration for the other port, as shown in Table 6.

Table 6 • **Port A and Port B Data Width Configurations for LSRAM**

Port A Data Width	Port B Data Width
x1	x1, x2, x4, x8, x16
x2	x1, x2, x4, x8, x16
x4	x1, x2, x4, x8, x16
x5	x5, x10, x20
x8	x1, x2, x4, x8, x16
x10	x5, x10, x20
x16	x1, x2, x4, x8, x16
x20	x5, x10, x20

4.1.1.2 Block Select Operation

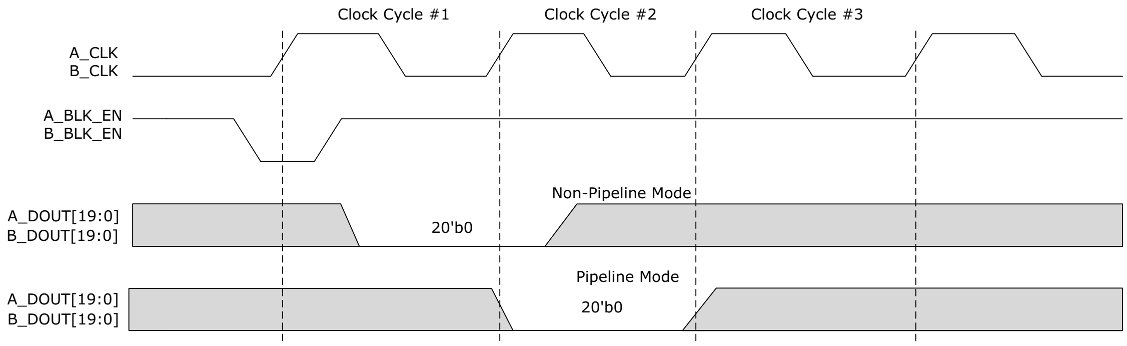
In dual-port mode, to perform two independent write and read operations (on Port A, Port B, or both) the block select signal is required. Table 7 lists the block select operation for Port A and Port B.

Table 7 • **Block Select Operation**

A_BLK_EN[2:0]	B_BLK_EN[2:0]	Operation
Any one bit = 0	Any one bit = 0	No operation on Port A or B. The data output A_DOUT[19:0] and B_DOUT[19:0] will be forced zero.
Any one bit = 0	111	Read or write operation on Port B
111	Any one bit = 0	Read or write operation on Port A
111	111	Read or write operation on both Ports A and B

When the pipeline registers are enabled, the effect of the block select at the outputs is delayed by one clock cycle, as shown in Figure 12.

Figure 12 • **Block Select Inputs for Dual-Port Mode**



4.1.1.3 Byte Write Enables

The byte write enables (A_WEN[1:0], B_WEN[1:0]) enable writing individual bytes of data for x20 and x16 widths. The byte write enables for Port A (A_WEN[1:0]) enables A_DIN[19:10] and A_DIN[9:0] respectively. The byte write enables for Port B (B_WEN[1:0]) enable B_DIN[19:10] and B_DIN[9:0] respectively.

The byte write enables are also used in x1, x2, x4, x5, x8, x10, x16, and x20 widths to select the operational mode (read/write) for a Port A or Port B. If all byte write enables are low, then Port A or Port B is considered to be in read mode and any read operations are controlled by the read enables (A_REN/B_REN).

Table 8 lists the byte write enable settings for Port A and Port B.

Table 8 • **Byte Write Enables Settings for Dual-Port Mode**

Depth x Width	A_WEN / B_WEN	Result
16K x 1, 8K x 2, 4K x 4, 4K x 5, 2K x 8, 2K x 10	00 or 10	Perform a read operation
	01 or 11	Perform a write operation
1K x 16	00	Perform a read operation
	01	Write [7:0]
	10	Write [17:10]
	11	Write [17:10], [7:0]
1K x 20	00	Perform a read operation
	01	Write [9:0]
	10	Write [19:10]
	11	Write [19:0]

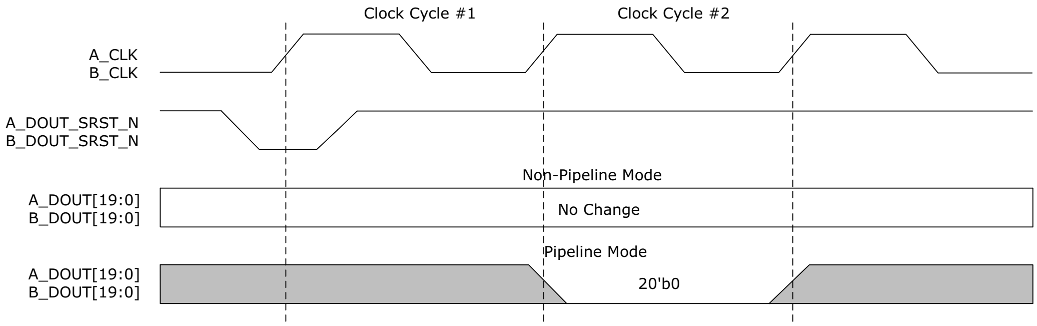
4.1.1.4 Read Enable

The read enable signals, A_REN and B_REN, perform the read operation on ports A and B. When read enable is low, the data outputs retain their previous state and no dynamic read power is consumed on that port. When read enable is high, LSRAM performs read operations and consumes read power.

4.1.1.5 Synchronous Pipeline Register Reset

Each pipeline register has one synchronous reset. In dual-port mode, A_DOUT_SRST_N and B_DOUT_SRST_N drive the synchronous reset of the data output pipeline registers—A_DOUT and B_DOUT. If the synchronous pipeline reset is low, the pipeline data output registers are reset to zero on the next valid clock edge, as shown in Figure 13.

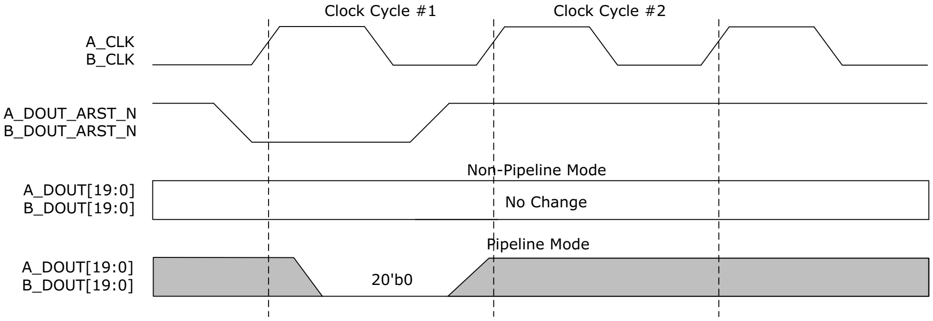
Figure 13 • **Synchronous Pipeline Register Reset in Dual-Port Mode**



4.1.1.6 Asynchronous Pipeline Register Reset

Each pipeline register has one asynchronous reset. In dual-port mode, A_DOUT_ARST_N and B_DOUT_ARST_N drive the asynchronous reset of the data output pipeline registers—A_DOUT and B_DOUT. If the asynchronous pipeline reset is driven low, the pipeline data output registers are immediately reset to zero, as shown in Figure 14.

Figure 14 • **Asynchronous Pipeline Register Reset in Dual-Port Mode**



4.1.1.7 Read Operation

In dual-port mode, LSRAM supports both pipelined and non-pipelined read operations. In a pipelined read operation, the output data is registered at the pipeline registers; as a result the data is available on the corresponding data output on the next clock cycle.

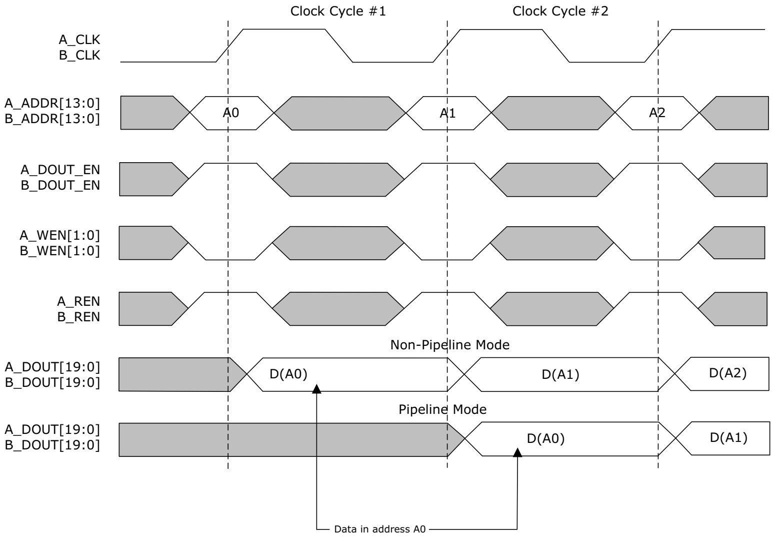
In a non-pipelined read operation, the pipeline registers are bypassed and read data is available on the output port in the same clock cycle.

Note: For high-performance designs, It is recommended to use the LSRAM with pipeline mode to meet the design timing constraints.

Note: When multiple depth-cascaded blocks are used, A_REN and B_REN ports of Dual-Port SRAM are disabled by the configurator GUI.

Figure 15 shows the timing for both pipelined and non-pipelined read operations in dual-port mode:

Figure 15 • **Read Operation in Dual-Port Mode**



4.1.1.8 Write Operation

In dual-port mode, LSRAM supports the following write operations:

- Simple Write
- Feed-Through Write
- Read-Before-Write

The type of write operation is specified while creating or configuring LSRAM in Libero SoC. For more information on LSRAM configuration, see LSRAM Configurator

Note: In dual-port mode, simultaneous write operations from both ports to the same address location are not prevented. Since simultaneous write operations can result in data uncertainty, it is recommended to use external logic in the fabric to avoid collisions.

4.1.1.8.1 Simple Write

In a simple-write operation, data input A_DIN and B_DIN are written to the corresponding address locations A_ADDR and B_ADDR. The data written to the memory is available at the output only after performing a read operation.

4.1.1.8.2 Feed-Through Write

In a feed-through write operation for pipelined operations, the read data is available on the data output bus on the next clock cycle. For non-pipelined operations, data written to the memory is available in the same clock cycle on the corresponding data output bus. For more information, see Figure 16 on page 22.

In dual-port mode during feed-through write, the data output of each port can change in one of the following ways:

- During read port reset, the data output becomes zero.
- If the block select input (A_BLK_EN) of Port A or B is driven low, then the corresponding port's data output becomes zero.
- During valid write operations when read enable (A_REN and B_REN) is high, then write data is available at the data output.
- If there is a valid read operation, then the read data is available at the data output. A valid read happens when read enable (A_REN and B_REN) inputs are high, and byte write enable (A_WEN[1:0] and B_WEN[1:0]) inputs are zero.

4.1.1.8.3 Read-Before-Write

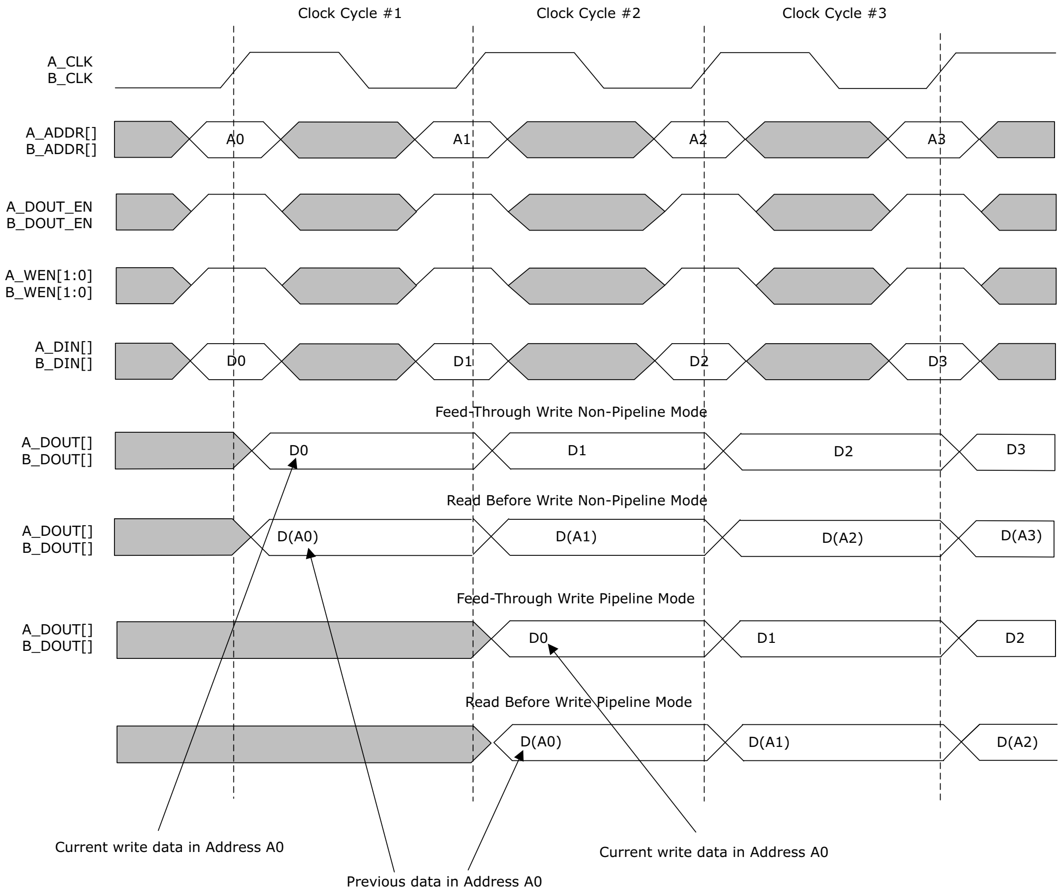
In a read-before-write operation for pipeline mode, read data is available on the data output bus on the next clock cycle. For non-pipeline mode, the previous memory data from the current write address is available on the data output before the new data is written to the address location. For more information, see Figure 16.

In dual-port mode during read-before-write operations, the data output of each port can change in one of the following ways:

- During read port reset, the data output becomes zero.
- If the block select input (A_BLK_EN) of Port A or B is driven low, then the corresponding port's data output becomes zero.
- During valid write operations when read enables (A_REN and B_REN) are driven high, the previous memory data from the current address is available on the data output before the new data is written to the address location.
- If there is a valid read operation, then the read data is available on the data output. A valid read happens when read enable (A_REN and B_REN) inputs are driven high, and byte write enable (A_WEN[1:0] and B_WEN[1:0]) inputs are driven low.

Figure 16 shows the timing for feed-through-write and read-before-write operations for dual-port mode.

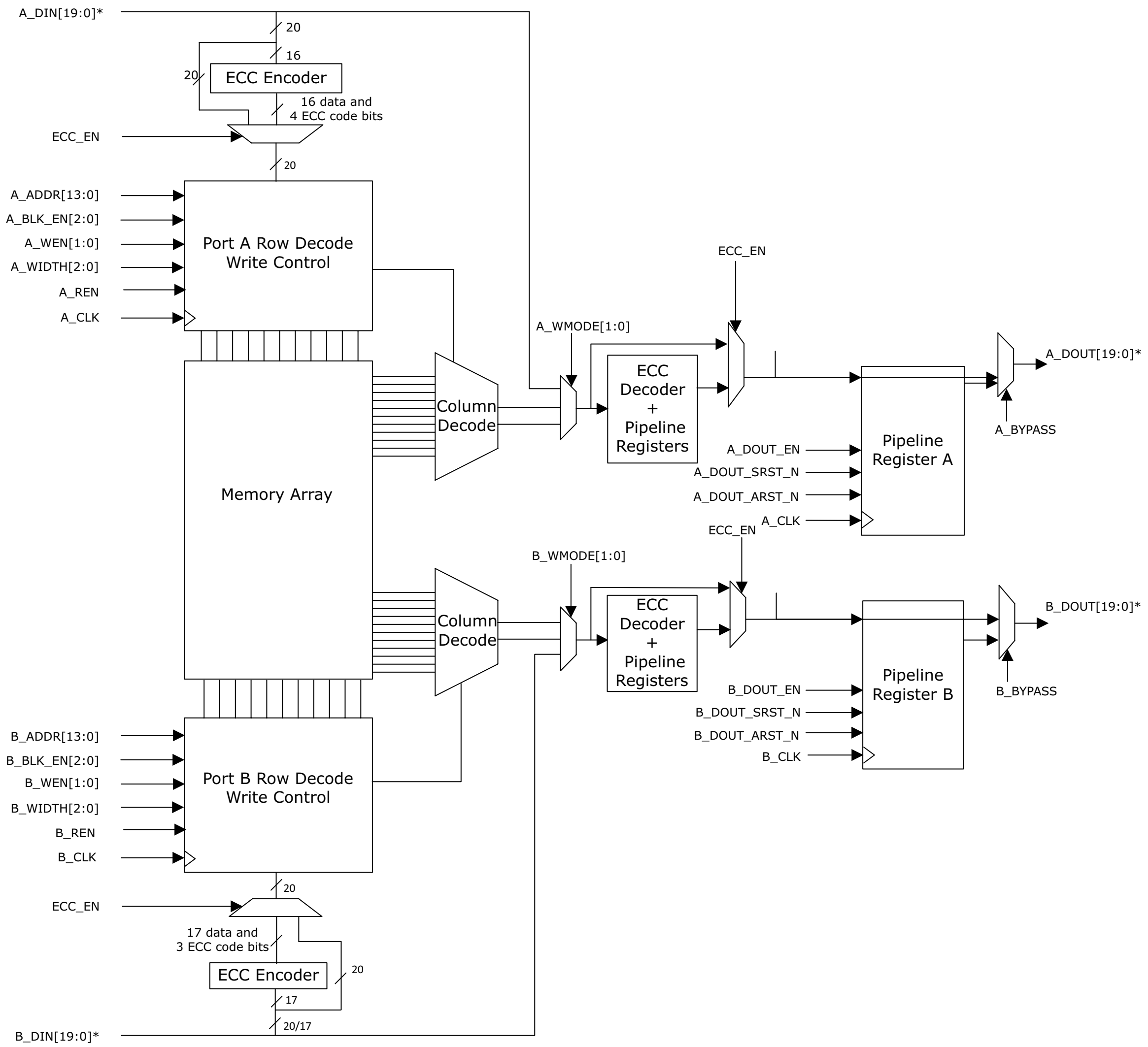
Figure 16 • Write Operations in Dual-Port Mode



4.1.2 Two-Port Mode

The LSRAM block can be configured as a two-port SRAM where Port A is dedicated to read operations and Port B is dedicated to write operations for data widths up to x20. For data widths greater than x20, the read port borrows the unused Port B data output signals, similarly write port borrows the unused Port A data input signals. Figure 17 shows the LSRAM in two-port mode with independent write and read ports, pipeline registers, ECC logic, and feed-through MUXes to enable immediate access to the write data.The ECC is supported only when the LSRAM is configured for 33-bit data width.

Figure 17 • **Simplified Functional Block Diagram for LSRAM in Two-Port Mode**



Note:
* For ECC mode:
A_DIN[15:0] and B_DIN[16:0]
A_DOUT[15:0] and B_DOUT[16:0]

4.1.2.1 Two-Port Data Width Configuration

In two-port mode, the maximum data width is x40. Each port can be configured in different data widths. The configuration of read port has a corresponding configuration for the write port, as shown in Table 9.

Table 9 • **LSRAM Data Width Configurations (Two-Port Mode)**

Read Port	Write Port
x40	x5, x10, x20, x40
x20	x40
x10	x40
x5	x40
x2	x32
x1	x32
x33	x33
x32	x1, x2, x32

4.1.2.2 Byte Write Enables

The byte write enables (A_WEN, B_WEN) enable writing individual bytes of data for x32, x33 and x40 data widths. For x40 width, the byte write enable for the corresponding data is used to enable each of the four bytes; that is, byte write enable for Port A enables A_DIN[19:10] and A_DIN[9:0] and byte write enable for Port B enables B_DIN[19:10] and B_DIN[9:0].

Table 10 lists the byte write enable settings for Port A and Port B.

Table 10 • **Byte Write Enable Settings for Two-Port Mode**

Depth x Width	A_WEN/B_WEN	Result
512 x 32	B_WEN[0] = 1	Write B_DIN[8:5], B_DIN[3:0]
	B_WEN[1] = 1	Write B_DIN[18:15], B_DIN[13:10]
	A_WEN[0] = 1	Write A_DIN[8:5], A_DIN[3:0]
	A_WEN[1] = 1	Write A_DIN[18:15], A_DIN[13:10]
512 x 40	B_WEN[0] = 1	Write B_DIN[9:0]
	B_WEN[1] = 1	Write B_DIN[19:10]
	A_WEN[0] = 1	Write A_DIN[9:0]
	A_WEN[1] = 1	Write A_DIN[19:10]
512 x 33 (with ECC Enabled)	B_WEN[1:0] = 11 A_WEN[1:0] = 11	Write B_DIN[16:0] Write A_DIN[15:0]

Note: For 512x32 configuration, bits 4, 9, 14 and 19 of data input ports are not used.

4.1.2.3 Read Enables

The read enable signals, A_REN and B_REN, perform the read operation on ports A and B. When read enable is low, the data outputs retain their previous state and no dynamic read power is consumed on that port. When read enable is high, LSRAM performs read operation and consumes read power.

Note: In two-port mode, LSRAM Port B read enable (B_REN) is tied to Port A read enable (A_REN).

4.1.2.4 Pipeline Registers

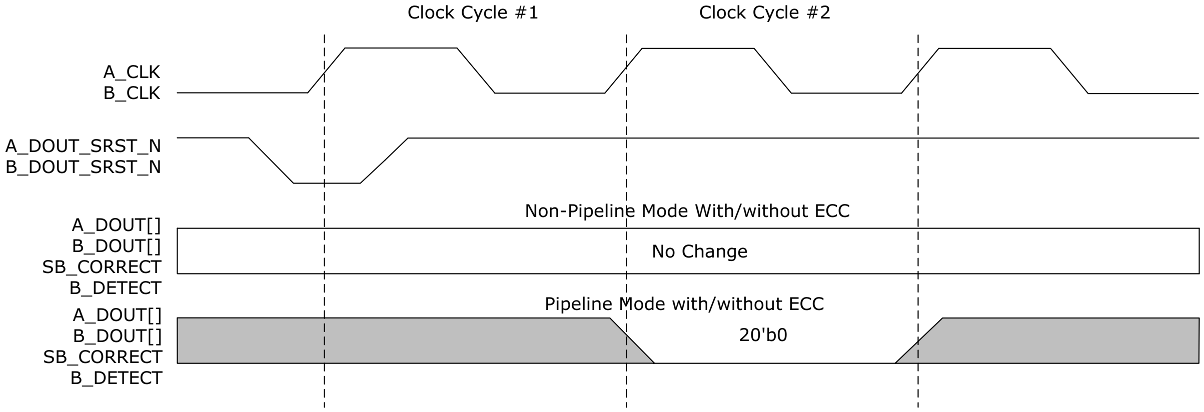
The outputs of the LSRAM have pipeline registers which can be enabled by the user for timing closure. These pipeline registers can be reset synchronously or asynchronously as explained in the following section.

4.1.2.4.1 Synchronous Pipeline Register Reset

Each data output port has its own synchronous reset. In two-port mode, A_DOUT_SRST_N and B_DOUT_SRST_N drive the synchronous reset of the read data output pipeline registers (A_DOUT and B_DOUT). If the synchronous pipeline reset is low, the pipeline data output registers are reset to zero on the next valid clock edge, as shown in Figure 18.

Note: In x33 two-port mode, if ECC is in pipeline mode, then this reset will also reset the ECC flag pipeline registers.

Figure 18 • **Synchronous Pipeline Reset in Two-Port Mode**



4.1.2.5 ECC Mode (For x33 Two-Port Mode Only)

In two-port mode when the port width is set to x33, ECC (single-bit error correction and dual-bit error detection) is available. The ECC encoder provides 40 bits (33 data bits and 7 encoded data bits) of data in the x33 mode (the seven encoded bits are not accessible to the user). Single-bit and dual-bit errors are counted for a full 33-bit read data word.

Note: ECC is supported only in Two-port LSRAM configurations and not supported for RTL inferred RAM blocks. For information about ECC configuration settings, see Figure 28.

The ECC decoder contains an optional pipeline register that adds a clock cycle of latency to the read operation (including the flags). Since the output data can also be pipelined, there are four possible scenarios:

- Pipeline mode with non-pipelined ECC
- Pipeline mode with pipelined ECC
- Non-pipeline mode with non-pipelined ECC
- Non-pipeline mode with pipelined ECC

The ECC logic generates two flags:

- **SB_CORRECT**—asserted when a single-bit error is detected. If SB_CORRECT is asserted, the correction on the read data output happens only when there is no dual-bit error.
- **DB_DETECT**—asserted when a dual-bit error is detected, but not corrected. Multi-bit errors (more than two bits) produce unknown results on the flags and data outputs.

Note: Scrubbing must be performed by user logic.

In pipeline mode, these flags are valid only in the read data output clock cycle. In non-pipeline mode, the ECC flags are valid only in the same clock cycle as the corresponding read data output, as the flags are reset in the next clock cycle. Table 11 lists the ECC error flags.

Table 11 • **Error Flags**

ECC Errors	SB_CORRECT	DB_DETECT	Correction
No error	0	0	NA
Single-bit error	1	0	Correction
Double-bit error	1	1	No correction

In SmartDebug, the ECC bits are included in the 40 bits data divided between adjacent locations. 16 bits of data in one location and 17 bits of data in the next location, the remaining 5 bits are ECC bits. The ECC bits are pre-calculated by Libero SoC and loaded in the background with the SRAM initialization data. For information about loading the initializing client for the SRAM memory IP in Libero SoC, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide* .

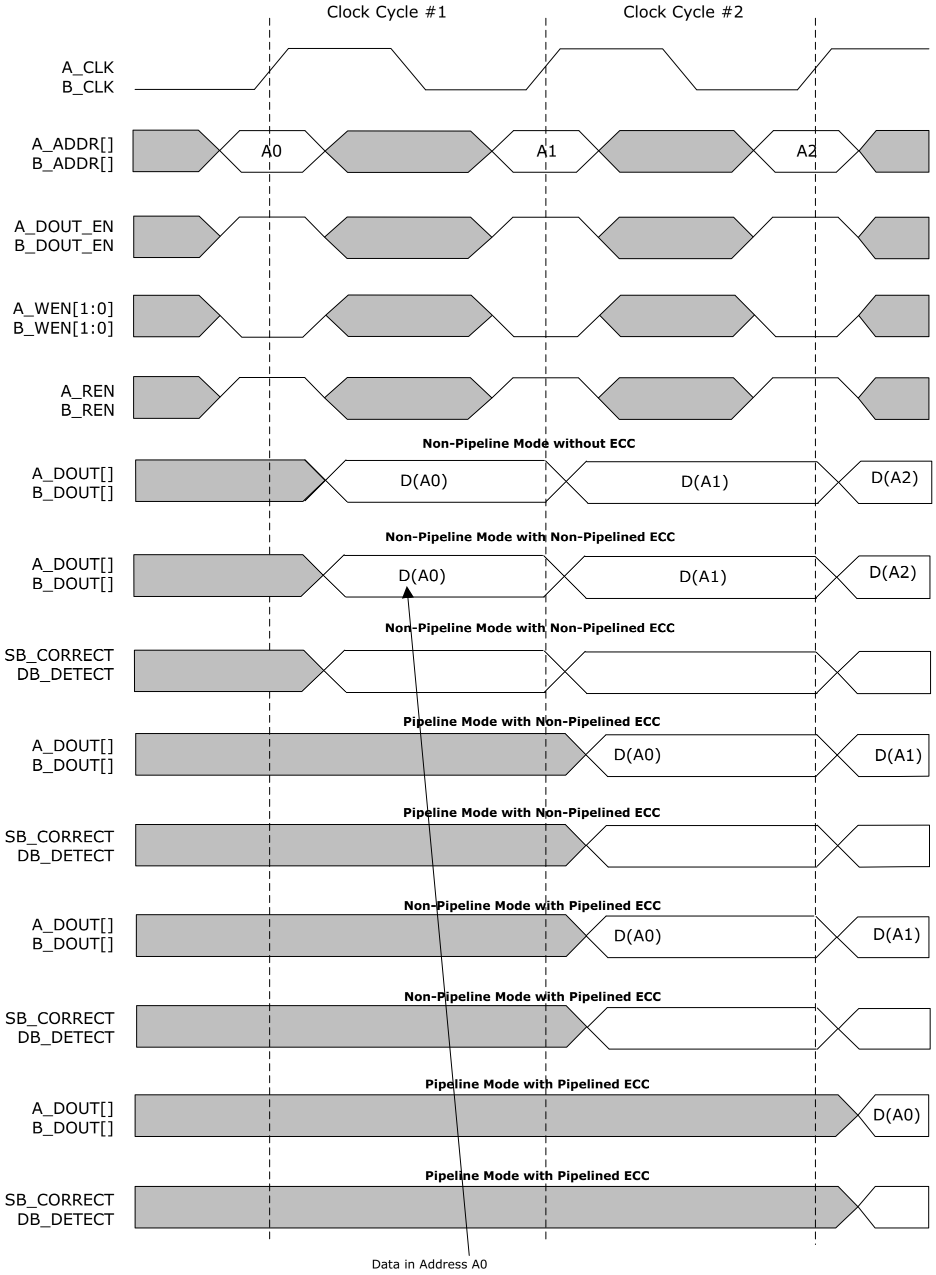
4.1.2.6 Read Operation

In two-port mode, LSRAM supports both pipelined and non-pipelined read operations. In a pipelined read operation, the output data is registered at the pipeline registers making the data available on the corresponding data output on the next clock cycle. If the ECC pipeline mode is enabled, an additional clock cycle is required for read data output. ECC flags are valid in the same clock cycle as the output data. For more information, see ECC Mode (For x33 Two-Port Mode Only).

In non-pipelined read operations, the pipeline registers are bypassed and read data is available on the output port in the same clock cycle. During this operation, LSRAM can generate glitches on the data output buses. Therefore, it is recommended to use LSRAM with pipeline registers to avoid glitches.

Figure 19 shows the timing for both pipelined and non-pipelined read operations in two-port mode.

Figure 19 • **Read Operation in Two-Port Mode**

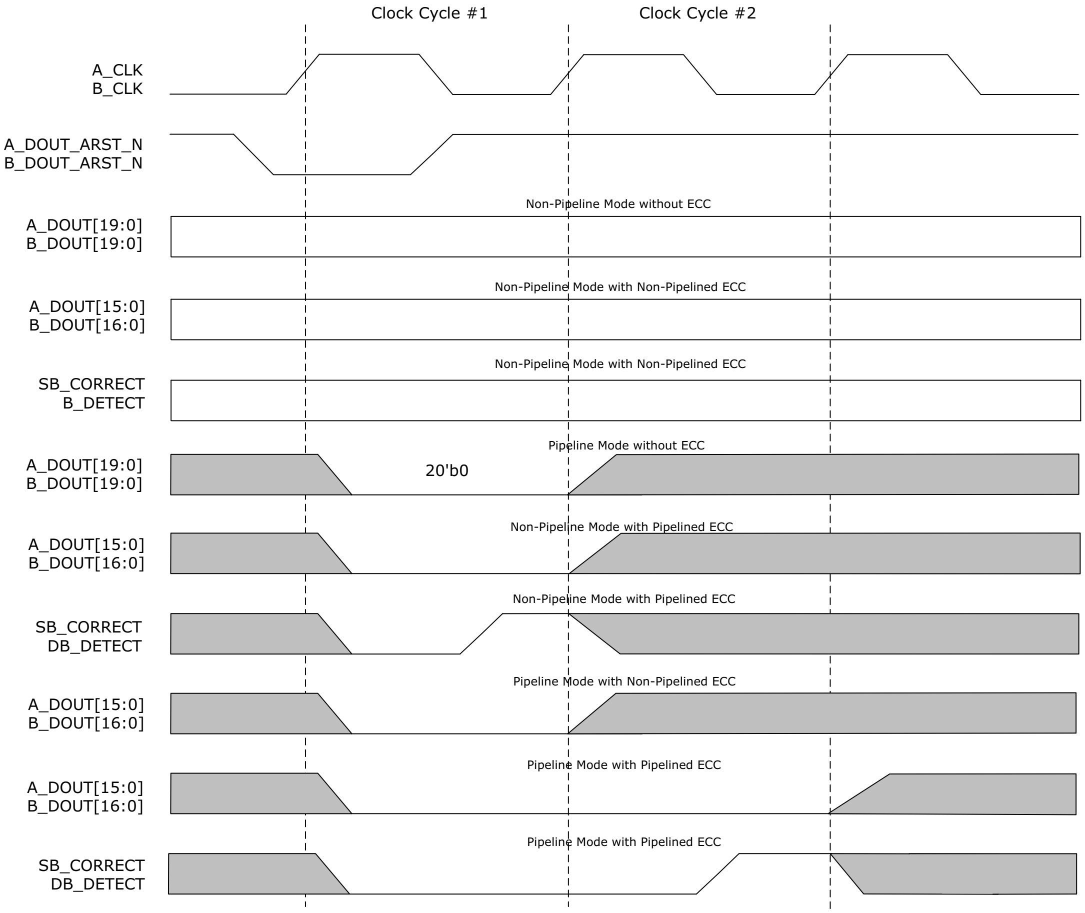


4.1.2.7 Asynchronous Pipeline Register Reset

Each data output port has its own asynchronous reset. In two-port mode, A_DOUT_ARST_N and B_DOUT_SRST_N drive the asynchronous reset of the read data output pipeline registers (A_DOUT and B_DOUT) and ECC pipeline registers. If the asynchronous pipeline reset is driven low, the pipeline data output registers are immediately reset to zero, as shown in Figure 20.

Note: In x33 two-port mode, if ECC is in pipeline mode, then this reset will also reset the ECC flag pipeline registers.

Figure 20 • **Asynchronous Pipeline Register Reset in Two-Port Mode**


Clock Cycle #1
Clock Cycle #2
A_CLK
B_CLK
A_DOUT_ARST_N
B_DOUT_ARST_N
Non-Pipeline Mode without ECC
A_DOUT[19:0]
B_DOUT[19:0]
Non-Pipeline Mode with Non-Pipelined ECC
A_DOUT[15:0]
B_DOUT[16:0]
Non-Pipeline Mode with Non-Pipelined ECC
SB_CORRECT
B_DETECT
Pipeline Mode without ECC
A_DOUT[19:0]
B_DOUT[19:0]
20'b0
Non-Pipeline Mode with Pipelined ECC
A_DOUT[15:0]
B_DOUT[16:0]
Non-Pipeline Mode with Pipelined ECC
SB_CORRECT
DB_DETECT
Pipeline Mode with Non-Pipelined ECC
A_DOUT[15:0]
B_DOUT[16:0]
Pipeline Mode with Pipelined ECC
A_DOUT[15:0]
B_DOUT[16:0]
Pipeline Mode with Pipelined ECC
SB_CORRECT
DB_DETECT

4.1.3 Implementation

An LSRAM block is implemented in a design using the following methods:

- RTL Inference during Synthesis
- LSRAM Configurator
- LSRAM Memory Macro

4.1.3.1 RTL Inference during Synthesis

Synplify Pro ME can infer an LSRAM from RTL automatically based on memory logic used in the design. In this case, synthesis handles all the signal connections of the LSRAM block to the rest of the design and sets the correct values for the static signals needed to configure the appropriate operational mode. The tool ties unused dynamic input signals to ground and provides default values to unused static signals. If a design requires more memory blocks than the blocks available in the device, the Synthesis tool will infer fabric registers for the extra memory blocks.

For more information about LSRAM inference by Synplify Pro, see *Inferring Microsemi RAM Blocks Application Note*.

4.1.3.2 LSRAM Configurator

The Libero SoC software catalog has the following LSRAM configuration tools:

- Dual-Port Large SRAM Configurator
- Two-Port LSRAM Configurator

Using these configurators, the LSRAM can be configured as per design requirements. The generated LSRAM component can be instantiated in the SmartDesign. The configurators generate the HDL wrapper files for LSRAM with the appropriate values assigned to the static signals. The generated HDL wrapper files can be used in the design hierarchy by connecting the ports to the rest of the design.

4.1.3.2.1 Dual-Port Large SRAM Configurator

The PF_DPSRAM configurator is available in the Libero SoC IP catalog -> **Memory & Controllers**.

The PF_DPSRAM configurator automatically cascades LSRAM blocks to create wider and deeper memories by selecting the most efficient aspect ratio. It also handles the grounding of unused bits. The configurator supports the generation of memories that have different aspect ratios on each port. The configurator uses one or more memory blocks to generate a RAM to match the configuration. The configurator also creates the external logic required for the cascading.

The configurator cascades RAM blocks in three different methods:

- Cascaded deep. For example, two blocks of 16384 x 1 are combined to create a 32768 x 1.
- Cascaded wide. For example, two blocks of 16384 x 1 are combined to create a 16384 x 2.
- Cascaded wide and deep. For example, four blocks of 16384 x 1 are combined to create a 32768 x 2, in a two-block width by two-block depth configuration.

For more information on Dual Port Mode, see Dual-Port Mode

Figure 21 • **Dual-Port Large SRAM Configurator: Generated Component**

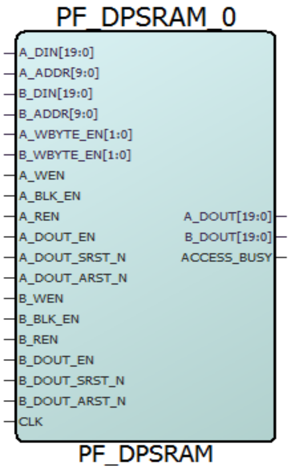


Table 12 • **Dual-Port LSRAM Configurator Signals**

Port	Direction	Polarity	Description
CLK	Input	Rising edge	Single-clock signal that drives both ports with the same clock. Exposed only when single clock is selected.
A_DIN[19:0]	Input		Port A write data
A_ADDR[9:0]	Input		Port A read address
A_BLK_EN	Input	Active high	Port A block select
A_CLK	Input	Rising edge	Port A clock. Applicable only when independent clocks are selected.
A_WEN	Input		Port A signal to switch between write and read modes: Low—Read High—Write
A_REN	Input		Port A read data enable
A_WBYTE_EN[1:0]	Input		Port A write byte enable
A_DOUT[19:0]	Output		Port A read data
A_DOUT_EN	Input	Active high	Port A read data register enable
A_DOUT_SRST_N	Input	Active low	Port A read data register synchronous reset
A_DOUT_ARST_N	Input	Active low	Port A read data register asynchronous reset
B_DIN[19:0]	Input		Port B write data
B_ADDR[9:0]	Input		Port B address
B_BLK_EN	Input	Active high	Port B enable
B_CLK	Input	Rising edge	Port B clock. Applicable only when independent clocks are selected
B_WEN	Input		Port signal to switch between write and read modes: Low—Read High—Write
B_REN	Input		Port B read data enable
B_WBYTE_EN[1:0]	Input		Port B write byte enable
B_DOUT[19:0]	Output		Port B read data
B_DOUT_EN	Input	Active high	Port B read data register enable
B_DOUT_SRST_N	Input	Active low	Port B read data register synchronous reset
B_DOUT_ARST_N	Input	Active low	Port B read data register asynchronous reset
ACCESS_BUSY	Output	Active high	Busy signal when being initialized or accessed using SmartDebug

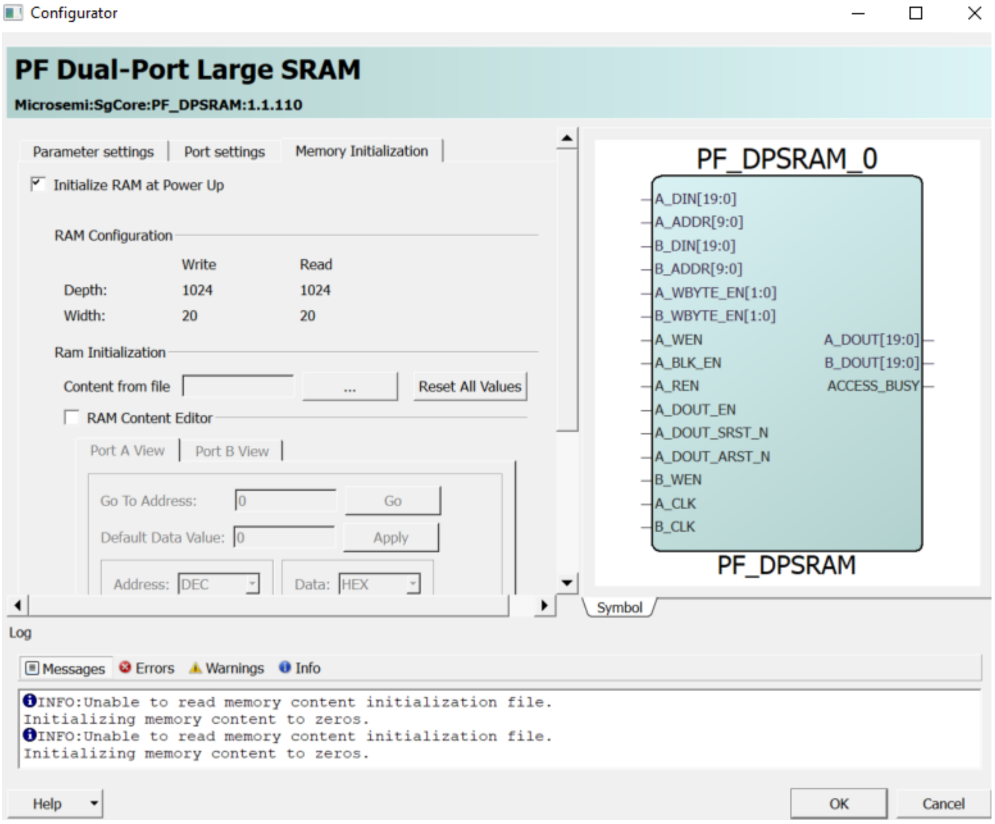
The dual-port LSRAM configurator has three tabs:

- Parameter settings
- Port settings
- Memory initialization

Parameter Settings

The parameter settings include the optimization of LSRAM for High Speed or Low Power, clock signal settings, and optional port settings. Figure 22 shows the Dual Port Large LSRAM block configurator.

Figure 22 • **Dual-Port Large SRAM Configurator: Parameter Settings**



☑ Initialize RAM at Power Up

	Write	Read
Depth:	1024	1024
Width:	20	20

□ RAM Content Editor

Go To Address: 0

Default Data Value: 0

Address: DEC Data: HEX

```
INFO:Unable to read memory content initialization file.
Initializing memory content to zeros.
INFO:Unable to read memory content initialization file.
Initializing memory content to zeros.
```

Optimization for High Speed or Low Power

The user can optimize the LSRAM with one of the following options:

- **High Speed**—optimizes the LSRAM for speed and area by using width cascading.
- **Low Power**—optimizes the LSRAM for low power by using depth cascading, it uses additional logic at the input and output.

Single Clock (CLK) or Independent Clocks (A_CLK and B_CLK)

The user can set the clock signals and the signal polarity:

- **Single clock**—drives both A and B ports with the same clock. This is the default configuration for dual-port LSRAM.
- **Independent clocks**—selects independent clock for each A_CLK for Port A and B_CLK for Port B).
- **Rising edge** or **Falling edge**—changes the signal polarity.

Optional Ports

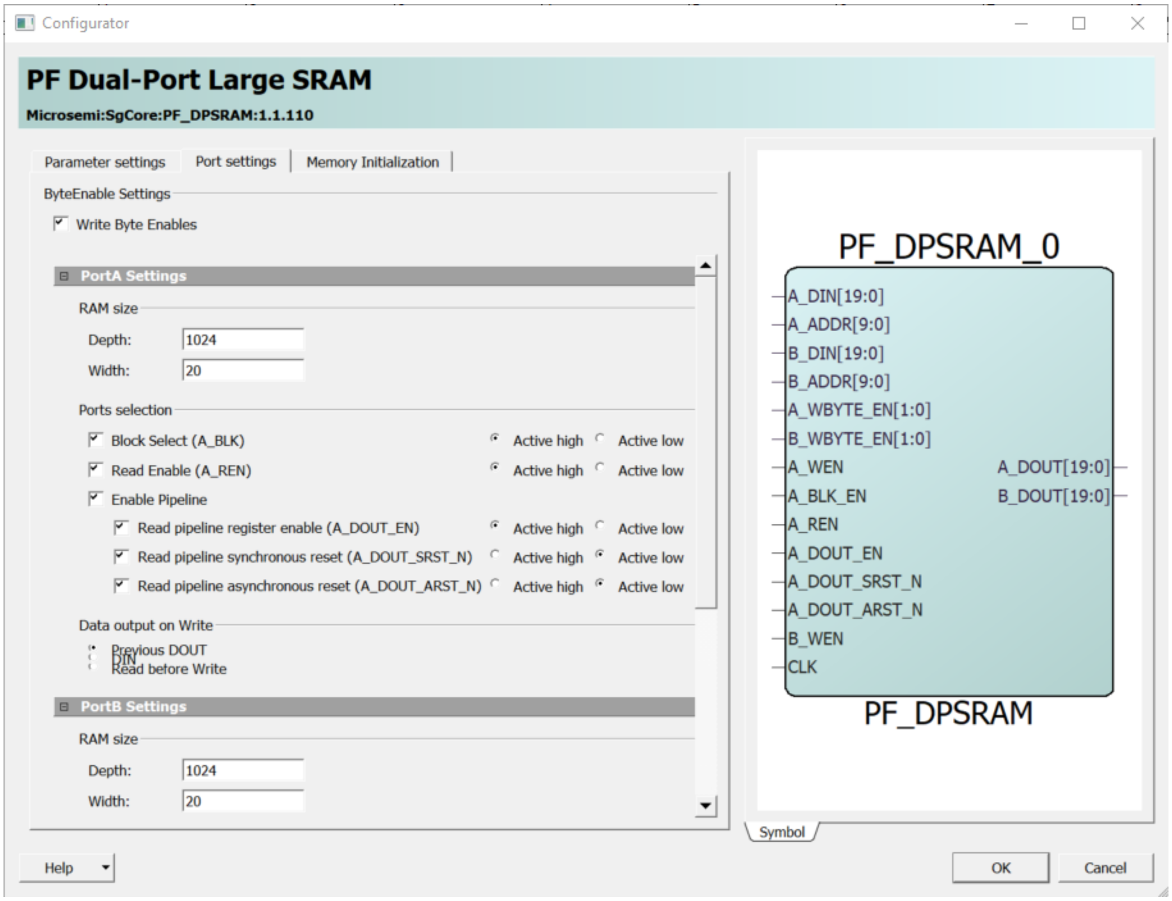
The user can select one of the following optional ports:

- **Lock access to SmartDebug**—when enabled, SmartDebug access to the RAM is disabled.
- **Expose ACCESS_BUSY output**—when enabled, SmartDebug ACCESS_BUSY signal is available as top-level port.

Port Settings

In the Port settings tab, the user can set the RAM size, select ports, and set data output on write settings for both Ports A and B. Figure 23 shows the PF_DPSRAM block port settings.

Figure 23 • **Dual-Port Large SRAM Configurator: Port Settings**



Configurator

PF Dual-Port Large SRAM

Microsemi:SgCore:PF_DPSRAM:1.1.110

Parameter settings | Port settings | Memory Initialization

ByteEnable Settings

☑ Write Byte Enables

⊟ PortA Settings

RAM size

Depth: 1024

Width: 20

Ports selection

☑ Block Select (A_BLK) ● Active high ○ Active low

☑ Read Enable (A_REN) ● Active high ○ Active low

☑ Enable Pipeline

☑ Read pipeline register enable (A_DOUT_EN) ● Active high ○ Active low

☑ Read pipeline synchronous reset (A_DOUT_SRST_N) ○ Active high ● Active low

☑ Read pipeline asynchronous reset (A_DOUT_ARST_N) ○ Active high ● Active low

Data output on Write

● Previous DOUT

○ DIN

○ Read before Write

⊟ PortB Settings

RAM size

Depth: 1024

Width: 20

Byte Enable Settings

Write Byte Enables—enables writing to individual bytes of data (A_WBYTE_EN and B_WBYTE_EN).

When the **Write Byte Enables** option is checked, the RAM Configurator cascades the RAM blocks and the tool distributes the RAM width equally such that the word width is a multiple of 8. Each Byte Write Enable bit controls the writing of 8-bits. For example, generating a 32-bit word width LSRAM, with Write Byte Enables, will cascade the RAMs width-wise such that there are a total of 4 Write Byte Enable bits (2 per RAM block) and each Write Byte Enable bit controls the writing of 8-bits (a byte) of data.

Other Examples are as follows:

- Width of 17, is divided as 9+8.
- Width of 35, is first divided as 18+17, and then, divided as 9+9+9+8.

RAM Size

The user can set the RAM size.

- **Depth**—sets the depth range. The depth range for each port is between 1 and 524288. The maximum value depends on the die.
- **Width**—sets the width range. The width range for each port is between 1 and 19040.

Note: The two ports can be configured independently for any depth and width. However, Port A depth x Port A width must be equal to Port B depth x Port B width. The width range varies between devices.

Ports Selection: Block Select (A_BLK_EN and B_BLK_EN)

- The default configuration for A_BLK_EN and B_BLK_EN is unchecked, which ties the signal to the active state and removes it from the generated component.

Select **Active high** or **Active low** tab to change the signal polarity.

Ports will be populated on the component by checking the respective check-boxes.

Read Enable (A_REN and B_REN)

- The default configuration for A_REN or B_REN is unchecked, which ties the signal to the active state and removes it from the generated macro.
- Select **Active high** or **Active low** to change the signal polarity.

Ports will be populated on the component by checking the respective check-boxes.

Enable Pipeline

Check the **Enable Pipeline** checkbox to enable pipelining of read data (A_DOUT or B_DOUT). If the Enable Pipeline checkbox is not checked, the user cannot configure the A_DOUT_EN/B_DOUT_EN, A_DOUT_SRST_N/B_DOUT_SRST_N, or A_DOUT_ARST_N/B_DOUT_ARST_N signals.

- **Register Enable (A_DOUT_EN and B_DOUT_EN)**—the pipeline registers for ports A and B have active high, enable inputs. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Synchronous Reset (A_DOUT_SRST_N and B_DOUT_SRST_N)**—the pipeline registers for ports A and B have active low, synchronous reset inputs. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Asynchronous Reset (A_DOUT_ARST_N and B_DOUT_ARST_N)**—the pipeline registers for ports A and B have active low, asynchronous reset inputs. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- Select **Active high** or **Active low** to change the signal polarity.

Ports will be populated on the component by checking their respective check-boxes.

Data Output on Write

Select the required option from the following:

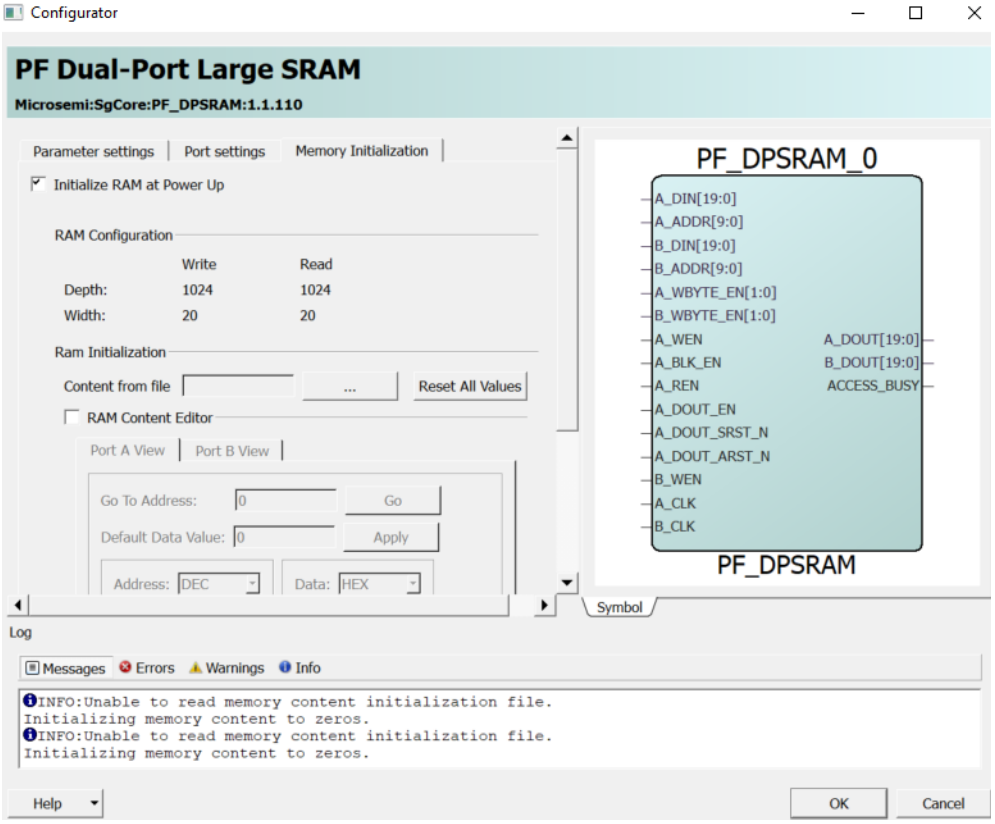
- **Previous DOUT**—the default data on the Read data output (A_DOUT or B_DOUT) during a write cycle is the DOUT data from the previous cycle (Previous DOUT).
- **DIN**—to enable feed-through write mode on Read data output.

- **Read before Write**—to perform a read operation before a write operation overwriting the previous data.

Memory Initialization at Power-Up

In the **Memory Initialization** tab, the user can initialize RAM at power-up. LSRAM can be initialized during device power-up and functional simulation. Figure 24 shows the PF Dual Port Large SRAM IP memory initialization.

Figure 24 • **Dual-Port Large SRAM Configurator: Memory Initialization**



PF Dual-Port Large SRAM

Microsemi:SgCore:PF_DPSRAM:1.1.110

☑ Initialize RAM at Power Up

RAM Configuration

	Write	Read
Depth:	1024	1024
Width:	20	20

Ram Initialization

Content from file

□ RAM Content Editor

Go To Address: 0

Default Data Value: 0

Address: DEC

Data: HEX

```
INFO:Unable to read memory content initialization file.
Initializing memory content to zeros.
INFO:Unable to read memory content initialization file.
Initializing memory content to zeros.
```

Initialize RAM at Power Up

The user can initialize RAM during power up of the device by setting the following:

- **Initialize RAM at Power Up**—loads the RAM content during device operation at power-up and functional simulation.
- **RAM Configuration** - Both write and read depths and widths are displayed as specified in the Port setting tab.
- **Initialize RAM Contents From File**—the RAM's Content can be initialized by importing the memory file. This avoids the simulation cycles required for initializing the memory and reduces the simulation runtime. The configurator partitions the memory file appropriately so that the right content goes to the right block RAM when multiple blocks are cascaded.
- **Import File**—selects and imports a memory content file (Intel-Hex) from the **Import Memory Content** dialog box. File extensions are set to `*.hex` for Intel-Hex files during import. For more information, see Appendix: Supported Memory File Formats for LSRAM and µSRAM. The imported memory content is displayed in the RAM Content Editor.
- **Reset All Values**—resets all the data values.

RAM Content Editor

The RAM Content Editor enables the user to specify the contents of RAM memory manually for both Port A and Port B. It also allows the user to modify imported data.

Port A View/Port B View

- **Go To Address**—enables the user to go to a specific address in the editor. The user can select the number display format (HEX, BIN, DEC) from the **Address** drop-down menu.
- **Default Data Value**—the user can set this with new data in order to change the default. When the data value is changed, all default values in the manager are updated to match the new value. Users can select the number display format (HEX, BIN, DEC) from the **Data** drop-down menu.
- **Address**—the Address column lists the address of a memory location. The drop-down menu specifies the number format of the address list (hexadecimal, binary, or decimal).
- **Data**—controls the data format and data values in the manager.

Note: The dialogs show all data with the MSB down to LSB. For example, if the row showed 0xAABB for a 16-bit word size, the AA is MSB and BB is LSB.

- Click **OK** to close the manager and save all changes made to the memory and its contents.
- Click **Cancel** to close the manager and cancel all the changes.

4.1.3.2.2 Two-Port LSRAM Configurator

The two-port SRAM (TPSRAM) IP configurator is available in the Libero SoC software under **Memory & Controllers**. Figure 25 shows the TPSRAM IP block available in the Libero SoC software. The TPSRAM configurator enables write access on one port and read access on the other port. The RAM configurator automatically cascades LSRAM blocks to create wider and deeper memories by selecting the most efficient aspect ratio. It also handles the grounding of unused bits. The core configurator supports the generation of memories that have different write and read aspect ratios. The configurator uses one or more memory blocks to generate a RAM matching the configuration. In addition, it also creates the surrounding cascading logic.

The configurator cascades RAM blocks in three different methods:

- Cascaded deep. For example, two blocks of 16384 x 1 combined to create a 32768 x 1.
- Cascaded wide. For example, two blocks of 16384 x 1 combined to create a 16384 x 2.
- Cascaded wide and deep. For example, four blocks of 16384 x 1 combined to create a 32768 x 2, in two blocks width-wise by two blocks depth-wise configuration.

Figure 25 • **Two-Port Large SRAM Configurator with ECC Enabled**

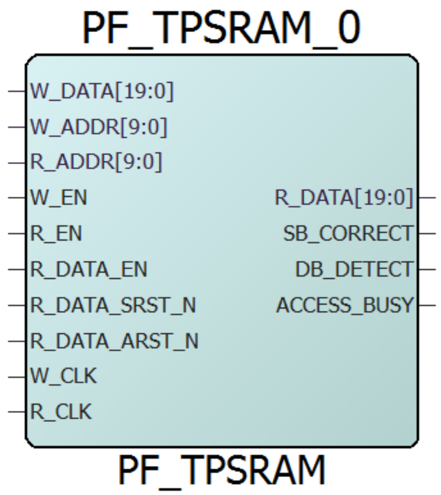


Table 13 • **Two-Port Large SRAM Configurator Signals**

Port	Direction	Polarity	Description
CLK	Input	Rising edge	Single clock to drive both W_CLK and R_CLK. Applicable only when single read/write clock is selected.
W_DATA[19:0]	Input		Write data
W_ADDR[9:0]	Input		Write address
W_EN	Input	Active high	Write port enable
W_CLK	Input	Rising edge	Write clock. Applicable only when independent clocks are selected.
R_CLK	Input	Rising edge	Read clock. Applicable only when independent clocks are selected.
R_EN	Input	Active high	Read data enable.
WBYTE_EN[]	Input		Write enable for byte write.
R_ADDR[9:0]	Input		Read address
R_DATA[19:0]	Output		Read data
R_DATA_EN	Input	Active high	Read data register enable
R_DATA_SRST_N	Input	Active low	Read data register synchronous reset
R_DATA_ARST_N	Input	Active low	Read data register asynchronous reset
SB_CORRECT	Output	Active high	Single-bit correct flag. Applicable only when ECC is
DB_DETECT	Output	Active high	Double-bit detect flag. Applicable only when ECC is
ACCESS_BUSY	Output	Active high	Busy signal from SmartDebug. Exposed only when Expose ACCESS_BUSY output is checked.

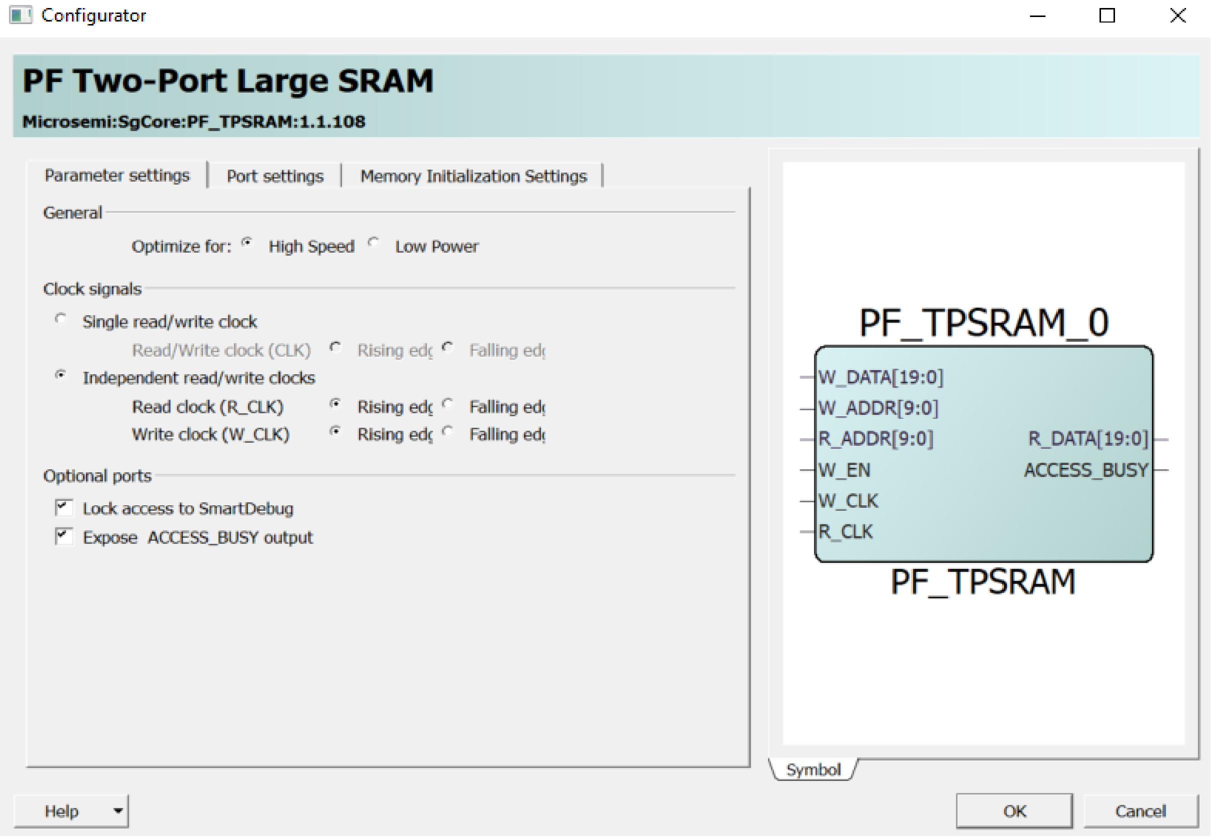
The two-port LSRAM configurator has three tabs:

- Parameter settings
- Port settings
- Memory initialization settings

Parameter Settings

The parameter settings include the Optimization for High Speed or Low Power, clock signals settings, and optional port settings. Figure 26 shows the TPSRAM block configurator.

Figure 26 • **Two-Port Large SRAM Configurator: Parameter Settings**



Configurator

PF Two-Port Large SRAM

Microsemi:SgCore:PF_TPSRAM:1.1.108

Parameter settings | Port settings | Memory Initialization Settings

General

Optimize for: ● High Speed ○ Low Power

Clock signals

○ Single read/write clock

Read/Write clock (CLK) ○ Rising edg ○ Falling edg

● Independent read/write clocks

Read clock (R_CLK) ● Rising edg ○ Falling edg

Write clock (W_CLK) ● Rising edg ○ Falling edg

Optional ports

☑ Lock access to SmartDebug

☑ Expose ACCESS_BUSY output

Optimization for High Speed or Low Power

The user can optimize the LSRAM macro with one of the following options:

- **High Speed**—optimizes the LSRAM macro for speed and area by using width cascading.
- **Low Power**—optimizes the LSRAM macro for low power, but it uses additional logic at the input and output by using depth cascading.

Single Read/Write Clock (CLK) or Independent Read/Write Clocks

The user can set the clock signals and the signal polarity:

- **Single clock**—drives write and read with the same clock. This is the default configuration for two-port LSRAM.
- **Independent clocks**—selects independent clock for Read (R_CLK) and Write (W_CLK)—R_CLK and W_CLK.
- **Rising edge** or **Falling edge**—sets the signal polarity.

Optional Ports

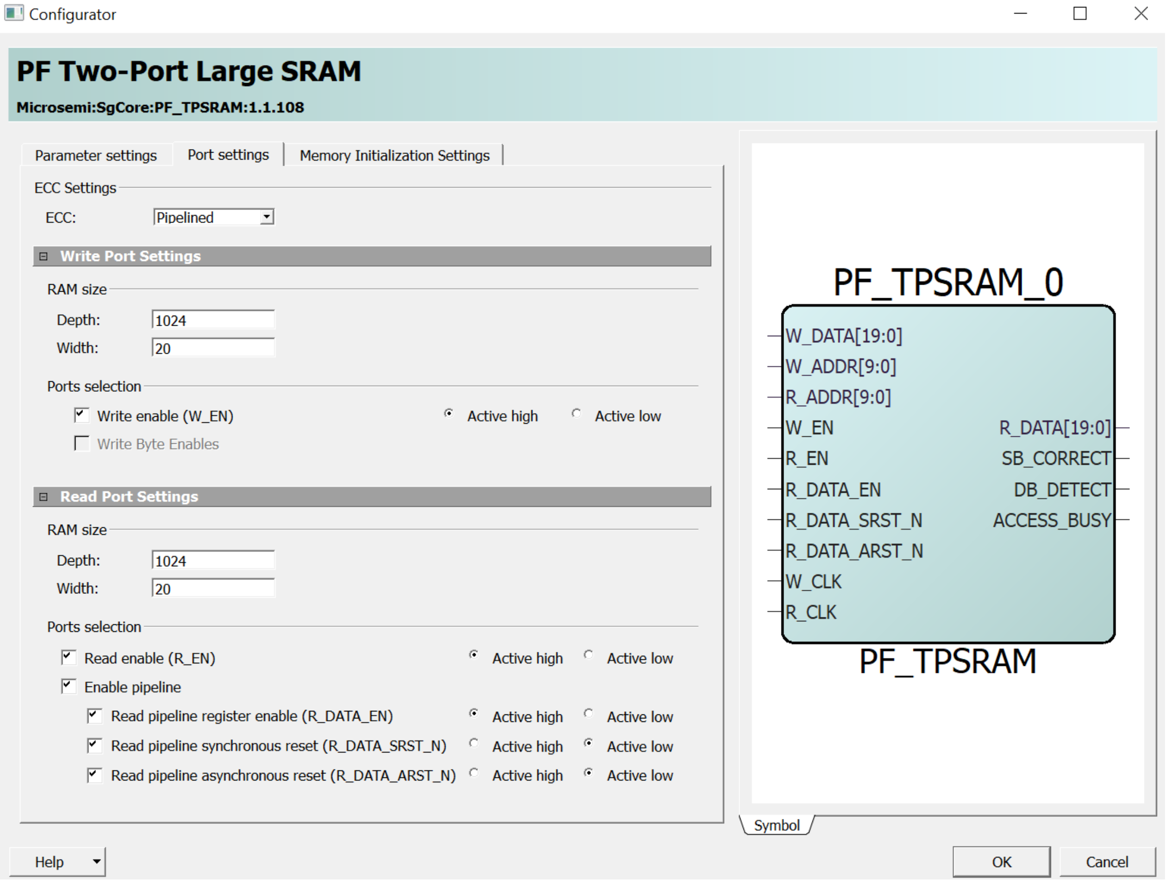
The user can select one of the following optional ports:

- **Lock access to SmartDebug** —when enabled, SmartDebug access to the RAM is disabled.
- **Expose ACCESS_BUSY output**—when enabled, SmartDebug ACCESS_BUSY signal is available as top-level port.

Port Settings

In the Port settings tab, the user can set RAM size, select ports, and set data output on write settings for both ports. Figure 27 shows the TPSRAM block port settings.

Figure 27 • **Two-Port Large SRAM Configurator: Port Settings**



Configurator

PF Two-Port Large SRAM

Microsemi:SgCore:PF_TPSRAM:1.1.108

Parameter settings | Port settings | Memory Initialization Settings

ECC Settings

ECC: Pipelined

⊟ Write Port Settings

RAM size

Depth: 1024

Width: 20

Ports selection

☑ Write enable (W_EN) ● Active high ○ Active low

☐ Write Byte Enables

⊟ Read Port Settings

RAM size

Depth: 1024

Width: 20

Ports selection

☑ Read enable (R_EN) ● Active high ○ Active low

☑ Enable pipeline

☑ Read pipeline register enable (R_DATA_EN) ● Active high ○ Active low

☑ Read pipeline synchronous reset (R_DATA_SRST_N) ○ Active high ● Active low

☑ Read pipeline asynchronous reset (R_DATA_ARST_N) ○ Active high ● Active low

ECC

The following three Error Correction Code (ECC) options are available:

- Disabled
- Pipelined
- Non-pipelined

Note: When ECC is enabled (pipelined or non-pipelined), both ports have data widths equal to 33 bits. The SB_CORRECT and DB_DETECT output ports are exposed when ECC is enabled.

Write Byte Enable Settings

Write Byte Enables—enables the writing of individual bytes of data (WBYTE_EN). This port is disabled while the ECC is enabled.

When the **Write Byte Enables** option is checked, the RAM Configurator cascades the RAM blocks and the tool distributes the RAM width equally such that the word width is a multiple of 8. Each Byte Write Enable bit controls the writing of 8-bits. For example, generating a 32-bit word width LSRAM, with Write Byte Enables, will cascade the RAMs width-wise such that there are a total of 4 Write Byte Enable bits (2 per RAM block) and each Write Byte Enable bit controls the writing of 8-bits (a byte) of data.

Other Examples are as follows:

- Width of 17, is divided as 9+8.
- Width of 35, is first divided as 18+17, and then, divided as 9+9+9+8.

RAM Size

The user can set the RAM size.

- **Depth**—sets the depth range. The depth range for each port is between 1 and 524288. The maximum value depends on the die.
- **Width**—sets the width range. The width range for each port is between 1 and 38080.

Note: The two ports can be configured independently for any depth and width. The write port depth x write port width must be equal to read port depth x read port width. The width range varies for different devices. The performance of the RAM will be affected if width and depth are too large.

Write Enable (W_EN)

The following list describes the result of asserting and de-asserting the W_EN signal:

- The default configuration for W_EN is checked (enabled). Unchecking the W_EN option ties the signal to the active state and removes it from the generated macro.
- Use **Active high** or **Active low** to change the signal polarity.

Read Enable (R_EN)

The following list describes the result of asserting and de-asserting R_EN signal:

- The default configuration for R_EN is unchecked (disabled), which ties the signal to the active state and removes it from the generated macro. Selecting the checkbox enables R_EN and the associated functionality.
- Use **Active high** or **Active low** to change the signal polarity.

Note: The user can insert the signal on the generated macro by checking its respective check-boxes.

Enable Pipeline

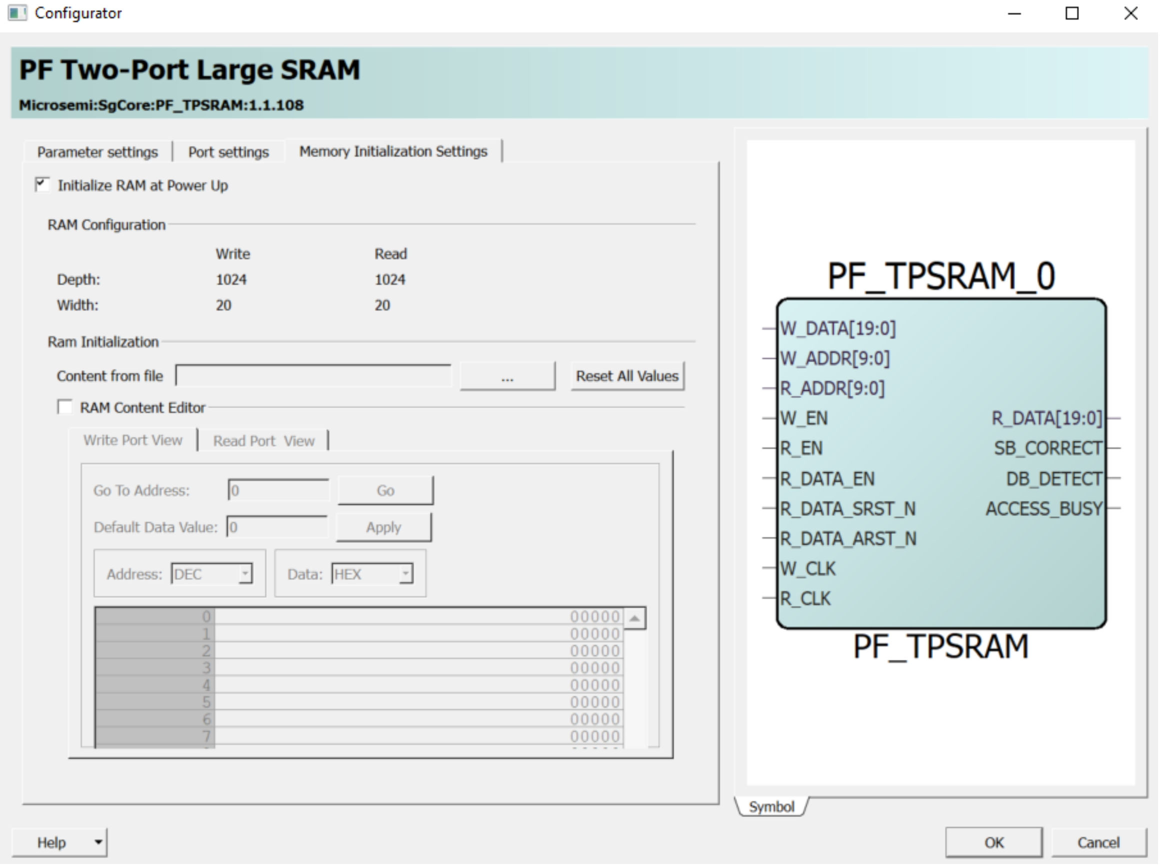
Check the **Enable Pipeline** checkbox to enable pipelining of Read data (R_DATA). This is a static selection and cannot be changed dynamically by driving it with a signal. If the Enable Pipeline checkbox is not checked, the user cannot configure R_DATA_EN, R_DATA_SRST_N, or R_DATA_ARST_N signals.

- **Read Pipeline Register Enable (R_DATA_EN)**—the pipeline registers for R_DATA have an active high, enable input. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Read Pipeline Synchronous Reset (R_DATA_SRST_N)**—the pipeline registers for R_DATA have an active low, synchronous reset input. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Read Pipeline Asynchronous Reset (R_DATA_ARST_N)**—the pipeline registers for R_DATA have an active low, asynchronous reset input. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Active high** or **Active low**—sets the signal polarity.

Note: The user can insert the signal on the generated macro by checking the respective check-boxes.

Memory Initialization at Power-Up

In the **Memory Initialization** tab, the user can initialize RAM at power-up. LSRAM can be initialized during device power-up and functional simulation. Figure 28 shows the TPSRAM IP memory initialization. For more information about LSRAM memory initialization, see Memory Initialization at Power-Up.



Configurator

PF Two-Port Large SRAM

Microsemi:SgCore:PF_TPSRAM:1.1.108

☑ Initialize RAM at Power Up

RAM Configuration

	Write	Read
Depth:	1024	1024
Width:	20	20

Ram Initialization

Content from file:

□ RAM Content Editor

Go To Address: 0

Default Data Value: 0

Address: DEC

Data: HEX

0	00000
1	00000
2	00000
3	00000
4	00000
5	00000
6	00000
7	00000

Figure 28 • **Two-Port Large SRAM Configurator: Memory Initialization**

4.1.3.3 LSRAM Memory Macro

An LSRAM primitive is available as a component that can be used directly in the HDL file or instantiated in SmartDesign. For more information about configuring LSRAM, see LSRAM Macro.

4.2 µSRAM

Each µSRAM has one read port and one write port for two-port memory requirements. Figure 29 shows the µSRAM I/O diagram.

Figure 29 • **µSRAM Input/Output**

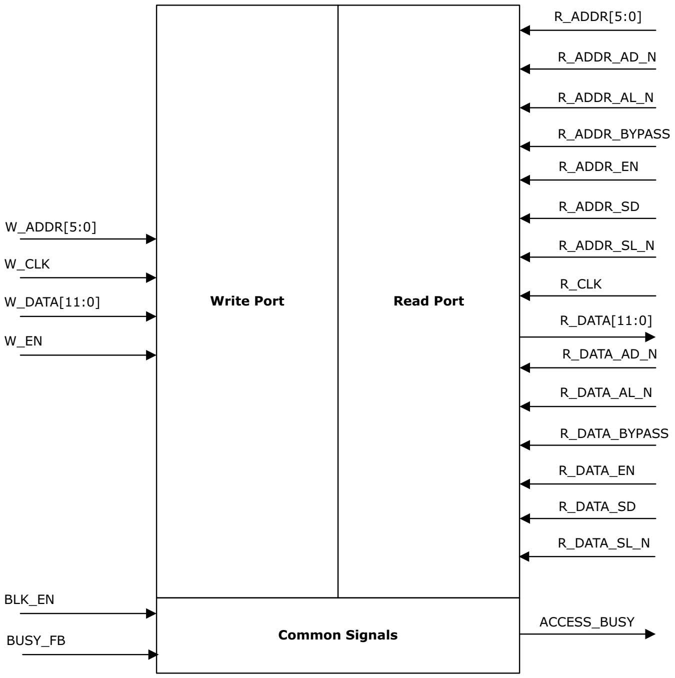


Table 14 lists the ports available in µSRAM.

Table 14 • **Port List for µSRAM**

Pin Name	Direction	Type[1]	Polarity	Description
W_EN	Input	Dynamic	Active high	Write enable
W_CLK	Input	Dynamic	Rising edge	Write clock
W_ADDR[5:0]	Input	Dynamic		Write address
W_DATA[11:0]	Input	Dynamic		Write-data
BLK_EN	Input	Dynamic	Active high	Read port enable
R_CLK	Input	Dynamic	Rising edge	Read clock
R_ADDR[5:0]	Input	Dynamic		Read-address
R_ADDR_BYPASS	Input	Static	Active low	Read-address and BLK_EN register select
R_ADDR_EN	Input	Dynamic	Active high	Read-address register Enable
R_ADDR_SL_N	Input	Dynamic	Active low	Read-address register synchronous load
R_ADDR_SD	Input	Static	Active high	Read-address register synchronous load data
R_ADDR_AL_N	Input	Dynamic	Active low	Read-address register asynchronous load
R_ADDR_AD_N	Input	Static	Active low	Read-address register asynchronous load data
R_DATA[11:0]	Output	Dynamic		Read-data

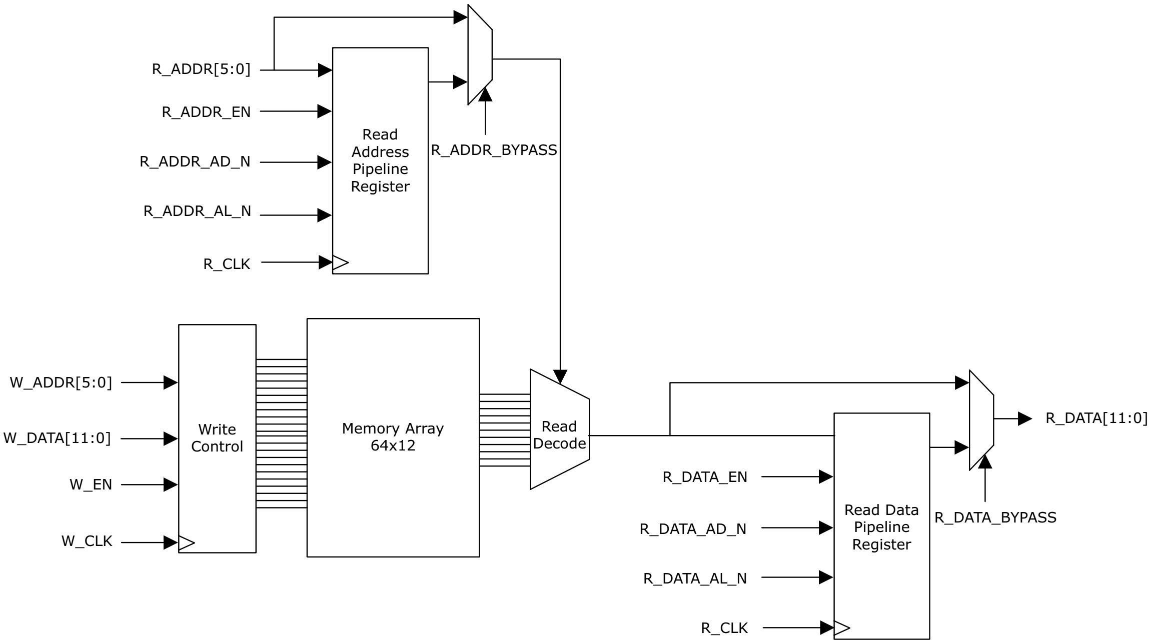
Table 14 • **Port List for µSRAM *(continued)***

Pin Name	Direction	Type[1]	Polarity	Description
R_DATA_BYPASS	Input	Static	Active low	Read-data pipeline register select
R_DATA_EN	Input	Dynamic	Active high	Read-data pipeline register enable
R_DATA_SL_N	Input	Dynamic	Active low	Read-data pipeline register synchronous load
R_DATA_SD	Input	Static	Active high	Read-data pipeline register synchronous load data
R_DATA_AL_N	Input	Dynamic	Active low	Read-data pipeline register asynchronous load
R_DATA_AD_N	Input	Static	Active low	Read-data pipeline register asynchronous load data
BUSY_FB	Input	Static	Active high	Lock access to SmartDebug
ACCESS_BUSY	Output	Dynamic	Active high	Busy signal when the RAM is being initialized or accessed using SmartDebug

[1] Static inputs are tied to 0 or 1 during design implementation.

Figure 30 shows the µSRAM with independent write and read ports and read data pipeline registers.

Figure 30 • **Simplified Functional Block Diagram of µSRAM**



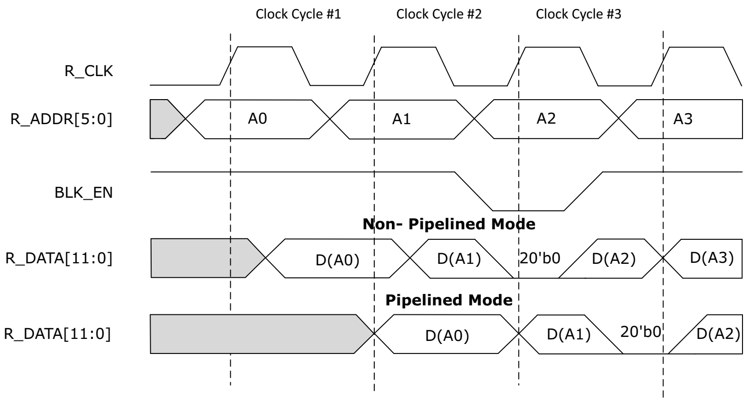
4.2.1 Read Operation

Read operations are independent of write operations and are performed asynchronously. Synchronous read operations can be performed by using fabric flip-flops as pipeline registers. These flip-flops are located in the associated interface cluster at the read address input and read data output.

When the input address (R_ADDR[]) is provided, the output data is available on the output data bus. When BLK_EN is high, the read operations are enabled. R_DATA contains the contents of the memory location selected by R_ADDR. When BLK_EN is low, the R_DATA is driven to zero.

Figure 31 shows the timing of read operation.

Figure 31 • **Read Operation in µSRAM**

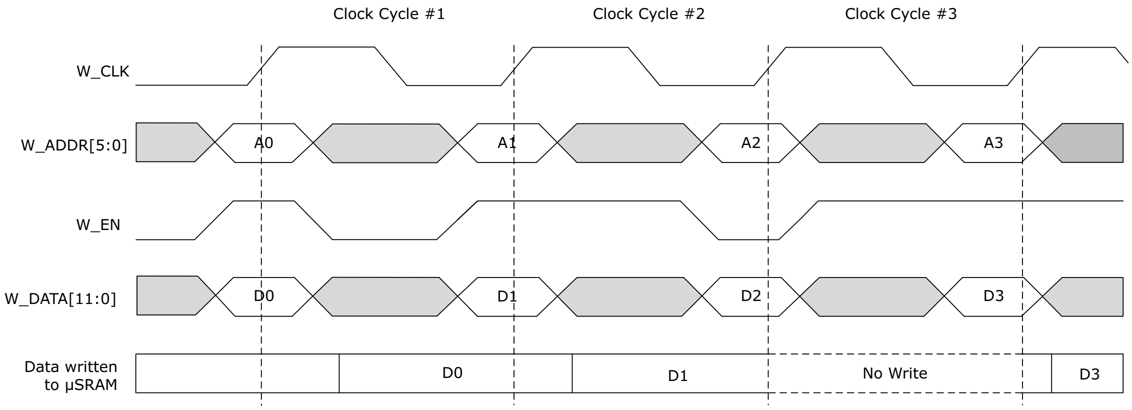


4.2.2 Write Operation

µSRAM supports synchronous write operation. The write port inputs are registered on the rising edge of the write port clock, W_CLK.

When write enable (W_EN) is high, the data (W_DATA) is written to the RAM at the address (W_ADDR) after write delay. When the write enable (W_EN) is low, no write operation is performed.

Figure 32 • **Write Operation in µSRAM**



4.2.3 Collision

Collision occurs when write and read-write operations requested for the same address at the same time. Simultaneous write and read operations at the same location are not supported. In µSRAM, collision is not supported and write operations supersede read operations. Therefore, during collision the read operation generates invalid data at the output until the write operation is completed.

4.2.4 Implementation

An µSRAM block can be implemented in a design by the following methods:

- RTL Inference during Synthesis
- µSRAM Configurator
- µSRAM Memory Macro

4.2.4.1 RTL Inference during Synthesis

Synplify Pro ME can infer a µSRAM from RTL automatically based on memory logic used in the design. In this case, synthesis handles all the signal connections of the µSRAM block to the rest of the design and sets the correct values for the static signals needed to configure the appropriate operational mode. The tool ties unused dynamic input signals to ground and provides default values to unused static signals. If a design requires more memory blocks than the blocks available in the device, the Synthesis tool will infer fabric registers for the extra memory blocks. For more information about µSRAM inference by Synplify Pro, see *Inferring Microsemi RAM Blocks Application Note*.

4.2.4.2 µSRAM Configurator

The µSRAM configurator is available in the Libero SoC software under **Memory & Controllers**. Figure 33 shows µSRAM available in the Libero SoC software. The RAM configurator automatically cascades µSRAM blocks to create wider and deeper memories by selecting the most efficient aspect ratio. It also handles the grounding of unused bits. The core configurator supports the generation of memories that have same write/read depth and width. The configurator uses one or more memory blocks to generate a RAM matching the configuration. In addition, it also creates the surrounding cascading logic.

The configurator cascades RAM blocks in three different methods:

- Cascaded deep. For example, two blocks of 64 x 12 combined to create a 128 x 12.
- Cascaded wide. For example, two blocks of 64 x 12 combined to create a 64 x 24.
- Cascaded wide and deep. For example, four blocks of 64 x 12 combined to create a 128 x 24, in two blocks width-wise by two blocks depth-wise configuration.

Write operations are synchronous for setting up the address, and writing the data. The memory write operations will be triggered at the rising edge of the clock.

Read operations for setting up the address and reading the data can be either asynchronous or synchronous. An optional pipeline register is available for the read-address to improve the setup. An optional pipeline register is available at the read data to improve the clock-to-output delay. Disabling both the address and read data registers creates the asynchronous mode for read operations. For synchronous read operations, the memory read operations will be triggered at the rising edge of the clock.

Figure 33 • **µSRAM Configurator - Generated Component**

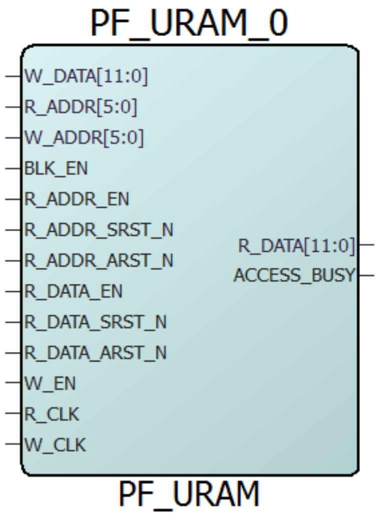


Table 15 • **µSRAM Configurator Signals**

Port	Direction	Polarity	Description
CLK	Input	Rising edge	Single clock signal that drives both ports with the same clock. Applicable only when Single clock is selected.
BLK_EN	Input	Active high	Read port enable
R_ADDR[5:0]	Input		Read address
R_ADDR_EN	Input	Active high	Read address register enable
R_ADDR_SRST_N	Input	Active low	Read address register synchronous reset
R_ADDR_ARST_N	Input	Active low	Read address register asynchronous reset
R_CLK	Input	Rising edge	Read clock. Applicable only when independent clocks are selected.
R_DATA[11:0]	Output		Read data
R_DATA_EN	Input	Active high	Read data register enable
R_DATA_SRST_N	Input	Active low	Read data register synchronous reset
R_DATA_ARST_N	Input	Active low	Read data register asynchronous reset
W_ADDR[5:0]	Input		Write address
W_CLK	Input	Rising edge	Write clock. Applicable only when independent clocks are selected.
W_EN	Input	Active low	Write enable
W_DATA[11:0]	Input		Write data
ACCESS_BUSY	Output	Active high	Busy signal from SmartDebug

This section also describes the µSRAM configuration and defines how the signals are connected.

The µSRAM configurator window has three tabs for settings:

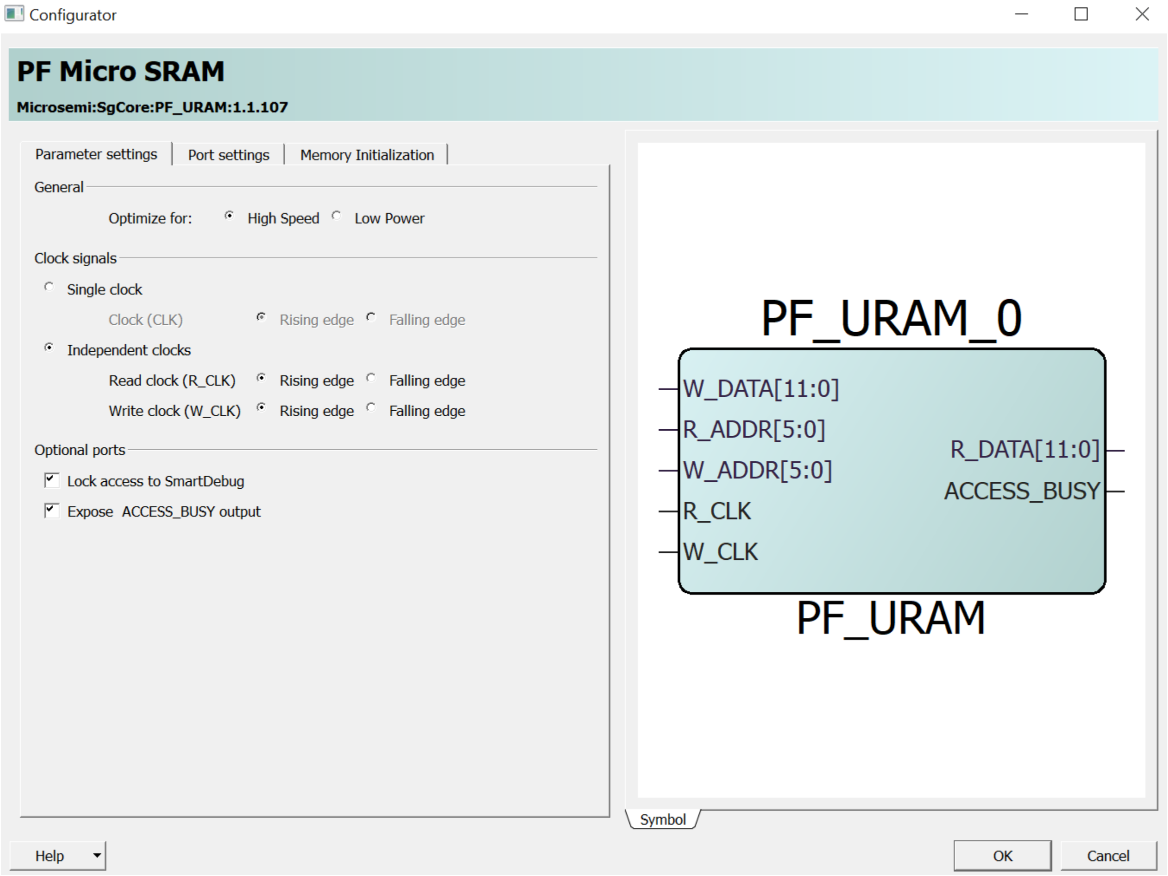
- Parameter settings
- Port settings
- Memory Initialization settings

4.2.4.2.1 Parameter Settings

The parameter settings include the optimization of µSRAM for High Speed or Low Power, clock signal settings, and optional port settings.

Figure 34 shows the µSRAM configurator.

Figure 34 • **MicroSRAM Configurator: Parameter Settings**



Configurator

PF Micro SRAM

Microsemi:SgCore:PF_URAM:1.1.107

Parameter settings | Port settings | Memory Initialization

General

Optimize for: ● High Speed ○ Low Power

Clock signals

○ Single clock

Clock (CLK) ● Rising edge ○ Falling edge

● Independent clocks

Read clock (R_CLK) ● Rising edge ○ Falling edge

Write clock (W_CLK) ● Rising edge ○ Falling edge

Optional ports

☑ Lock access to SmartDebug

☑ Expose ACCESS_BUSY output

Optimization for High Speed or Low Power

The user can optimize the µSRAM macro with one of the following options:

- **High Speed**—to optimizes the µSRAM macro for speed and area by using width cascading.
- **Low Power**—to optimizes the µSRAM macro for low power, but it also uses additional logic at the input and output by using depth cascading.

Single Clock (CLK) or Independent Clocks (R_CLK and W_CLK)

The user can set the clock signals and the signal polarity:

- **Single clock**—drives both write and read ports with the same clock. This is the default configuration for µSRAM.
- **Independent clocks**—selects the independent clock for each port (R_CLK for read port and W_CLK for write port).
- **Rising edge** or **Falling edge**—changes the signal polarity.

Optional Ports

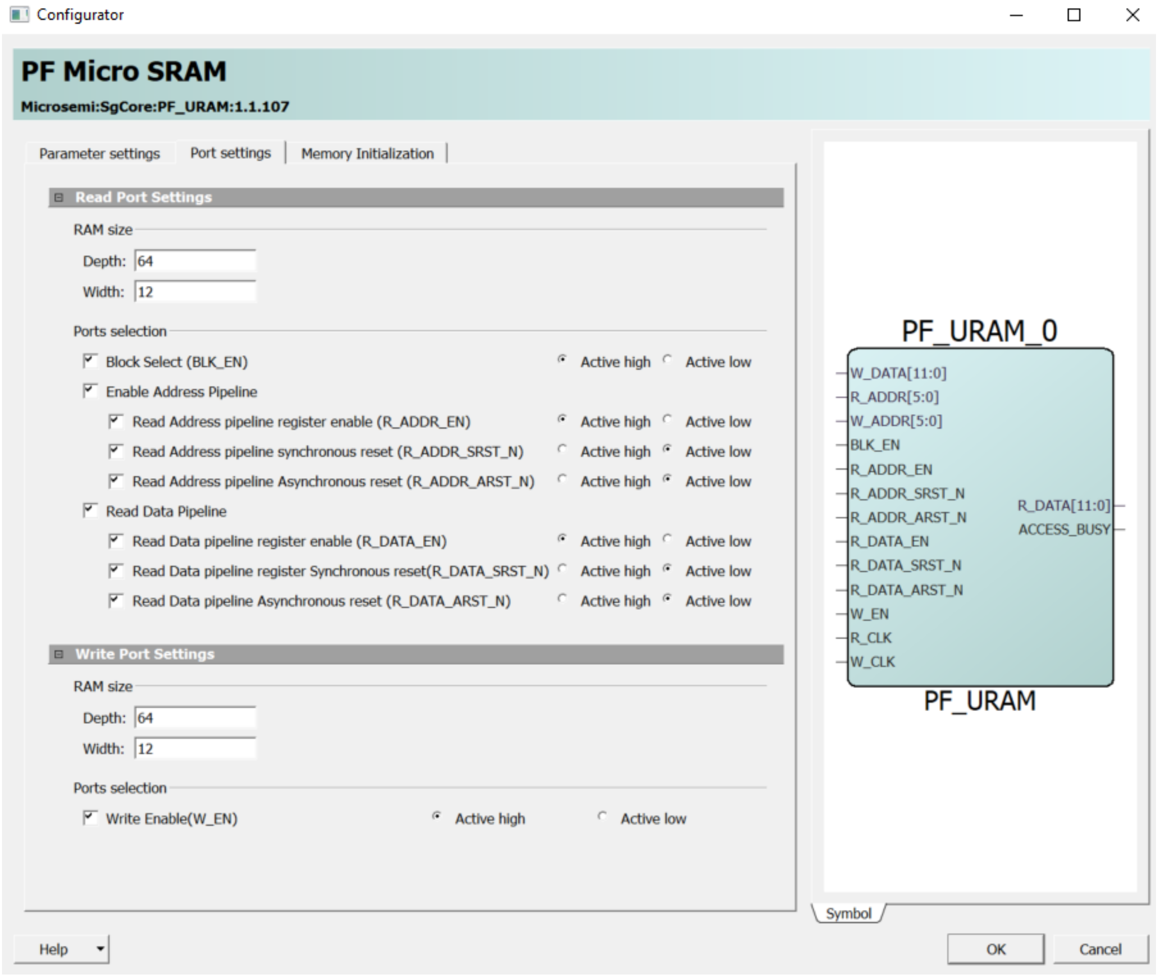
The user can select one of the following optional ports:

- **Lock access to SmartDebug**—when enabled, SmartDebug access to the RAM is disabled.
- **Expose ACCESS_BUSY output** —when enabled, SmartDebug ACCESS_BUSY signal is available as top-level port.

4.2.4.2.2 Port Settings

In the Port settings tab, the user can set RAM size, select ports, and set data output on write settings for both write and read ports. Figure 35 shows the μSRAM IP block port settings.

Figure 35 • **MicroSRAM Configurator: Port Settings**



Configurator

PF Micro SRAM

Microsemi:SgCore:PF_URAM:1.1.107

Parameter settings | Port settings | Memory Initialization

⊟ Read Port Settings

RAM size

Depth: 64

Width: 12

Ports selection

☑ Block Select (BLK_EN) ● Active high ○ Active low

☑ Enable Address Pipeline

☑ Read Address pipeline register enable (R_ADDR_EN) ● Active high ○ Active low

☑ Read Address pipeline synchronous reset (R_ADDR_SRST_N) ○ Active high ● Active low

☑ Read Address pipeline Asynchronous reset (R_ADDR_ARST_N) ○ Active high ● Active low

☑ Read Data Pipeline

☑ Read Data pipeline register enable (R_DATA_EN) ● Active high ○ Active low

☑ Read Data pipeline register Synchronous reset(R_DATA_SRST_N) ○ Active high ● Active low

☑ Read Data pipeline Asynchronous reset (R_DATA_ARST_N) ○ Active high ● Active low

⊟ Write Port Settings

RAM size

Depth: 64

Width: 12

Ports selection

☑ Write Enable(W_EN) ● Active high ○ Active low

RAM Size

The user can set the RAM size.

- **Depth**—sets the depth range. The depth range for each port is 1 to 2048. The maximum value depends on the die.
- **Width**—sets the width range. The width range for each port is 1 to 53280.

Note: The two ports can be configured independently for any depth and width. Write depth x write width must be equal to read depth x read width. The width and depth range varies for different devices. The performance of the RAM will be affected if width and depth are too large.

Port Selection: Block Select for Read Port (BLK_EN)

The default configuration for BLK_EN is unchecked, which ties the signal to the active state and removes it from the generated macro. For more information, see Read Operation.

Select **Active high** or **Active low** to change the signal polarity.

Note: Ports will be populated on the component by checking its respective check-boxes.

Write Enable (W_EN)

The default configuration for W_EN is unchecked (disabled), which ties the signal to the active state and removes it from the generated macro. For more information, see Write Operation, for more information.

Select **Active high** or **Active low** to change the signal polarity.

Note: The user can insert the signal on the generated macro by checking the respective check-boxes.

Enable Address Pipeline

Check the **Enable Address Pipeline** checkbox to enable pipelining of Read data (R_ADDR_EN). This is a static selection and cannot be changed dynamically by driving it with a signal. If the Enable Address Pipeline checkbox is not checked, the user cannot configure R_ADDR_EN,R_ ADDR_SRST_N, or R_ADDR_ARST_N signals.

- **Register Enable (R_ADDR_EN and R_DATA_EN)**: the pipeline registers for read ports have active high enable inputs. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Synchronous Reset (R_ADDR_SRST_N and R_DATA_SRST_N)**: the pipeline registers for read ports have active low, synchronous reset inputs. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Asynchronous Reset (R_ADDR_ARST_N and R_DATA_ARST_N)**: the pipeline registers for read ports have active low, asynchronous reset inputs. By default, the checkbox is disabled. Selecting this checkbox adds the signal to the top-level port.
- **Active high** or **Active low**: changes the signal polarity.

Note: Ports will be populated on the component by checking the respective check-boxes.

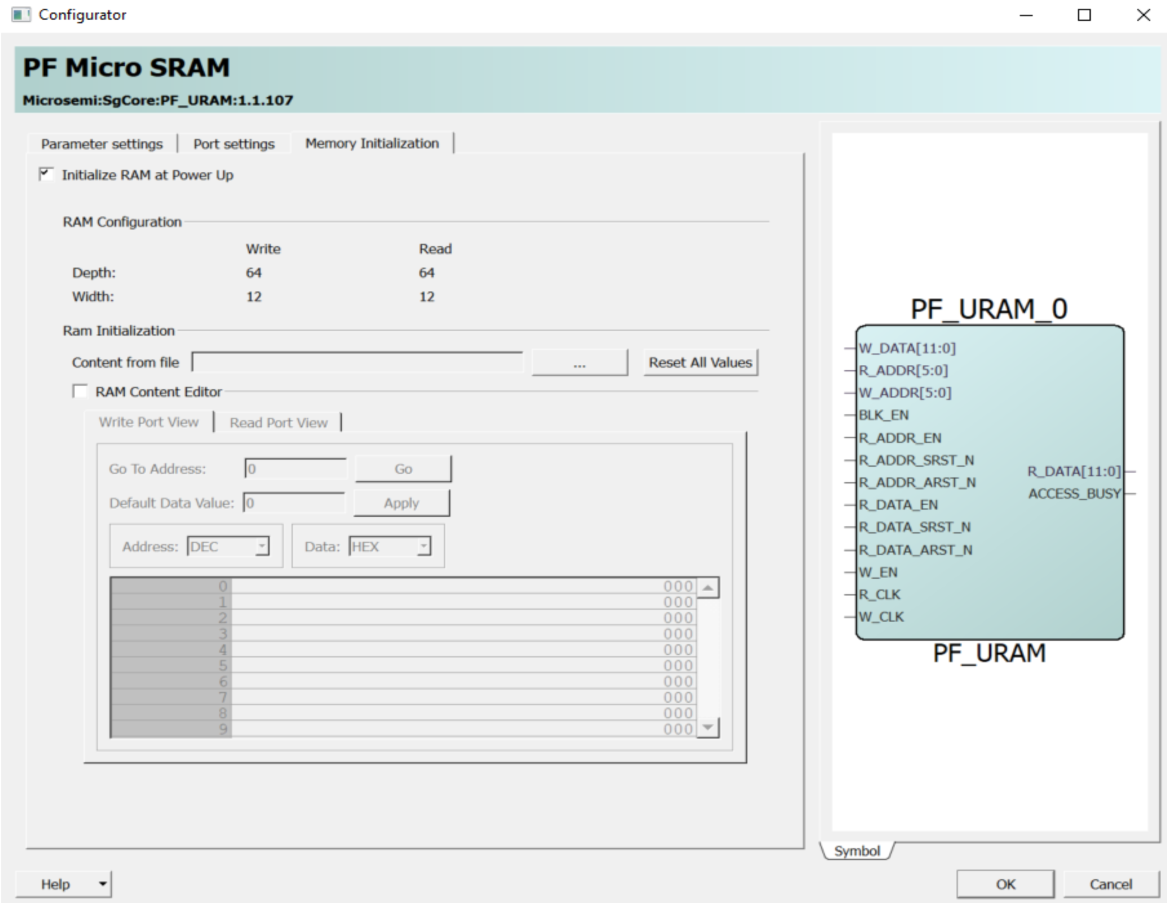
Read Data Pipeline

- The default configuration for µSRAM is to enable the pipeline of read data (R_DATA).
- Click the Pipeline checkbox to enable pipelining of Read data (R_DATA). This is a static selection and cannot be changed dynamically by driving it with a signal.
- Turning off pipelining of Read data also disables the configuration options of the respective R_DATA_EN, R_DATA_SRST_N, and R_DATA_ARST_N signals.

4.2.4.2.3 Memory Initialization at Power-Up

In the **Memory Initialization** tab, the user can initialize RAM at power-up. µSRAM can be initialized during device power-up and functional simulation. Figure 36 shows the µSRAM IP memory initialization. For more information about the usage of memory initialization, see LSRAM Memory Initialization at Power-Up.

Figure 36 • **MicroSRAM Configurator: Memory Initialization**



Configurator

PF Micro SRAM

Microsemi:SgCore:PF_URAM:1.1.107

Parameter settings | Port settings | Memory Initialization

☑ Initialize RAM at Power Up

RAM Configuration

	Write	Read
Depth:	64	64
Width:	12	12

Ram Initialization

Content from file [ ] ... Reset All Values

□ RAM Content Editor

Write Port View | Read Port View

Go To Address: 0 Go

Default Data Value: 0 Apply

Address: DEC Data: HEX

0	000
1	000
2	000
3	000
4	000
5	000
6	000
7	000
8	000
9	000

4.2.4.3 µSRAM Memory Macro

A µSRAM primitive is available as a component that can be used directly in the HDL file or instantiated in SmartDesign. For more information about configuring µSRAM, see µSRAM Macro.

4.3 µPROM

PolarFire devices include a single User Programmable Read-Only Memory (µPROM) row located at the bottom of the fabric, providing up to 513 Kb of non-volatile, read-only memory. The address bus is 16 bits wide, and the read data bus is 9-bit wide. Fabric logic has write access to the entire µPROM data. Figure 37 shows the high-level block diagram of µPROM.

Figure 37 • **Simplified Functional Block Diagram of µPROM**

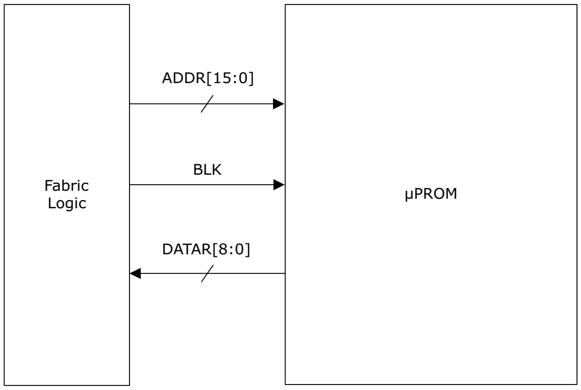


Table 16 lists the ports of µPROM.

Table 16 • **µPROM Port List**

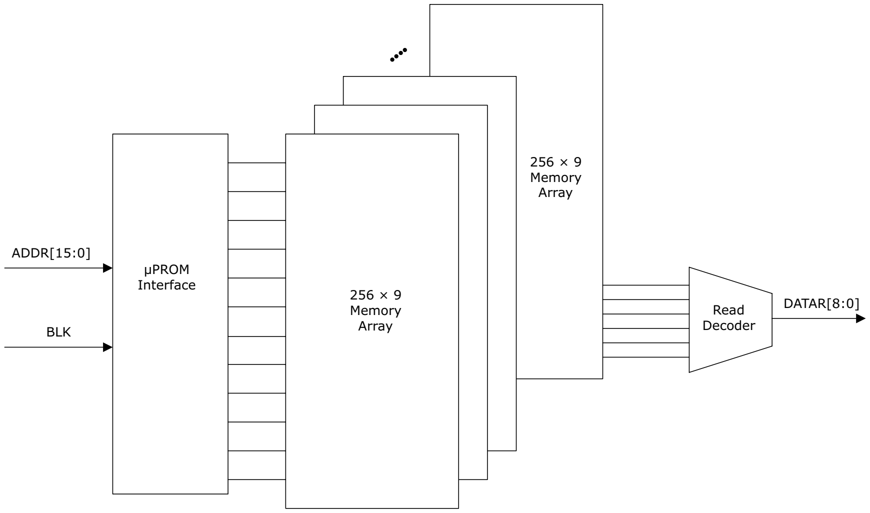
Port Name	Direction	Polarity	Description
ADDR[15:0]	Input		Address input
BLK	Input	Active high	Block select
DATAR[8:0]	Output		Read data output

4.3.1 µPROM Architecture and Address Space

Architecturally, the µPROM is structured into different memory arrays with 8-bit addressing. Each array is 256 x 9 bit words. The total number of memory array per device is die-dependent and can vary from 194 for the smallest die (MPFS025) to 553 for the largest die (MPFS460). The address bus (ADDR) is 16 bits wide. The lower 8 bits, ADDR [7:0], are used to address the individual 9-bit words while the upper 8 bits, ADDR[15:8], are used to address the individual memory array blocks inside the device.

Figure 38 shows a simplified block diagram of the µPROM memory.

Figure 38 • **µPROM Memory Blocks**

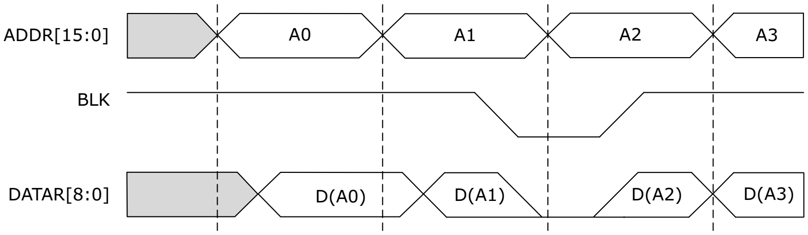


4.3.2 µPROM Operation

In µPROM, the write operation (program/erase) is performed during FPGA Programming. To configure the µPROM, the Libero SoC µPROM configurator writes the memory file (`*.mem`) to the configuration bit-stream. The memory file is a plain text file. The device programmer (FlashPro 5 or later) writes this memory file to µPROM during FPGA programming. Read operations are performed only through the fabric interface. All µPROM read operations are asynchronous. To perform a synchronous read operation, the µPROM output needs to be pipelined using fabric registers.

Figure 39 shows the read timing diagram for the µPROM.

Figure 39 • **Read Operation in µPROM**

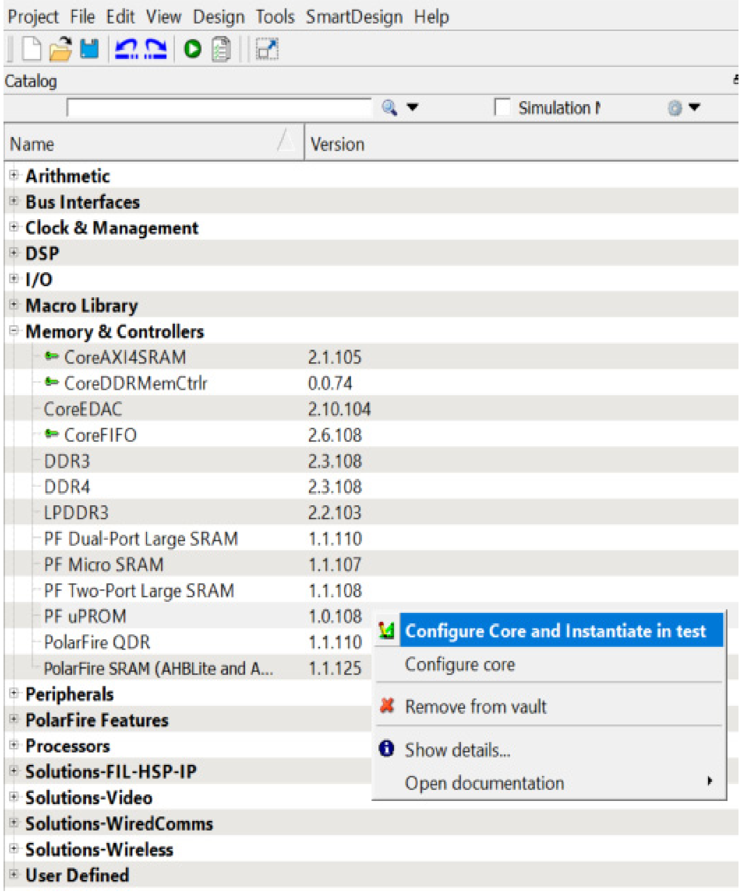


4.3.3 Implementation

The µPROM can be implemented using the Configurator available in the Catalog tab. To invoke the configurator:

1. Expand **Memory & Controllers** in the catalog.
2. Do one of the following to invoke the µPROM Configurator:
   - Double-click or right-click **PF µPROM** and select **Instantiate in <design_name>** to instantiate the µPROM in the SmartDesign canvas.
   - Double-click or right-click **PF µPROM** and choose **Configure Core**. Enter a component name for the µPROM when prompted.

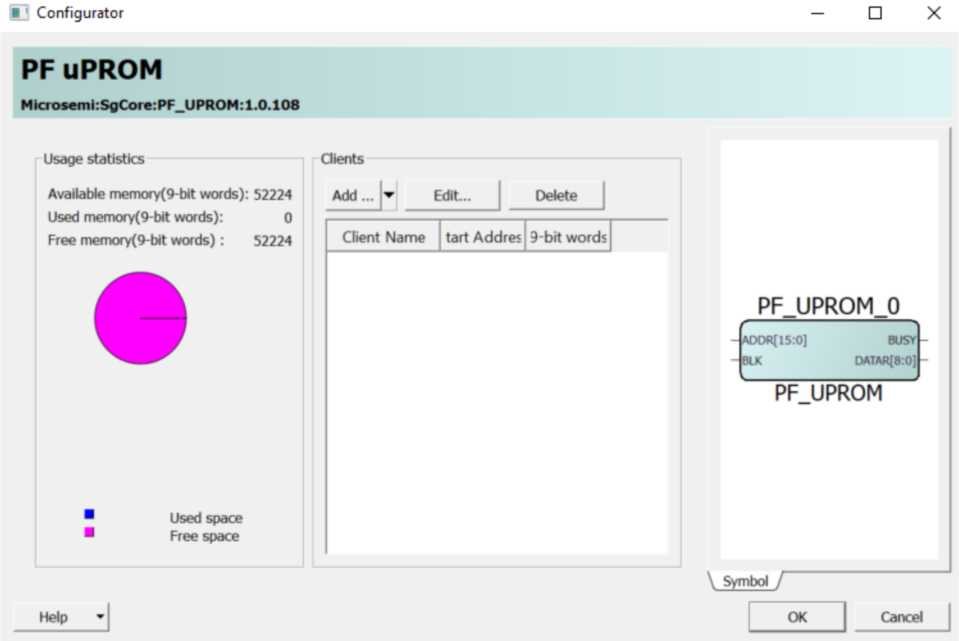
Figure 40 • **µPROM Core in Catalog**


Project File Edit View Design Tools SmartDesign Help
Catalog
Simulation I
Configure Core and Instantiate in test
Configure core
Remove from vault
Show details...
Open documentation

Name	Version
Arithmetic	
Bus Interfaces	
Clock & Management	
DSP	
I/O	
Macro Library	
Memory & Controllers	
CoreAXI4SRAM	2.1.105
CoreDDRMemCtrlr	0.0.74
CoreEDAC	2.10.104
CoreFIFO	2.6.108
DDR3	2.3.108
DDR4	2.3.108
LPDDR3	2.2.103
PF Dual-Port Large SRAM	1.1.110
PF Micro SRAM	1.1.107
PF Two-Port Large SRAM	1.1.108
PF uPROM	1.0.108
PolarFire QDR	1.1.110
PolarFire SRAM (AHBLite and A...	1.1.125
Peripherals	
PolarFire Features	
Processors	
Solutions-FIL-HSP-IP	
Solutions-Video	
Solutions-WiredComms	
Solutions-Wireless	
User Defined	

3. In the µPROM Configurator, click **Add Clients to System** to add a client to the µPROM. Figure 41 shows the µPROM Configurator. Only the client will be programmed and not all the content is erased during programming.

Figure 41 • **µPROM Configurator**



Configurator

PF uPROM

Microsemi:SgCore:PF_UPROM:1.0.108

Usage statistics

Available memory(9-bit words): 52224
Used memory(9-bit words): 0
Free memory(9-bit words) : 52224

4.3.3.1 Usage Statistics

Usage statistics display the total memory size of the µPROM, the size of used memory, and available free memory. All memory sizes are expressed in terms of the number of 9-bit words.

4.3.3.1.1 Available Memory

The µPROM can hold upto 58,368 9-bit words (total 525312 bits), depends on the die. For more information, see Table 1.

4.3.3.1.2 Used Memory

When memory clients are added, the used memory displays the total amount of memory (number of 9-bit words) used by all clients. This is displayed in blue in the pie chart.

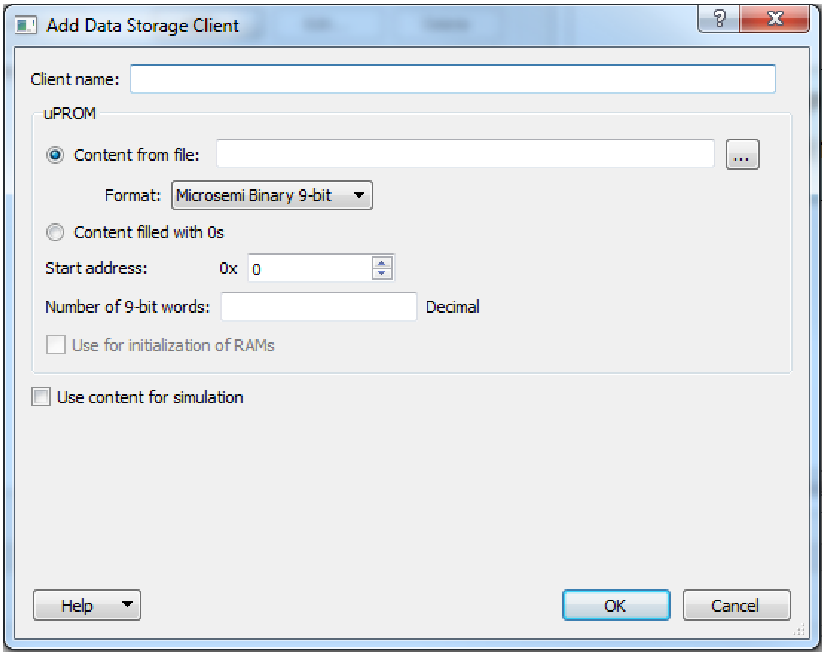
4.3.3.1.3 Free Memory

Free memory (number of 9-bit words) is displayed in magenta in the pie chart.

4.3.3.2 Add Clients to System

1. Click **Add Clients to System** to open the **Add Data Storage Client** dialog box (Figure 42).
2. Specify the start address, client size, the content of the client, and whether or not to use the memory content for simulation.

Figure 42 • **Add Data Storage Client Dialog Box**



Add Data Storage Client

Client name:

uPROM

● Content from file:

Format: Microsemi Binary 9-bit

○ Content filled with 0s

Start address: 0x 0

Number of 9-bit words: Decimal

□ Use for initialization of RAMs

□ Use content for simulation

4.3.3.2.1 Client Name

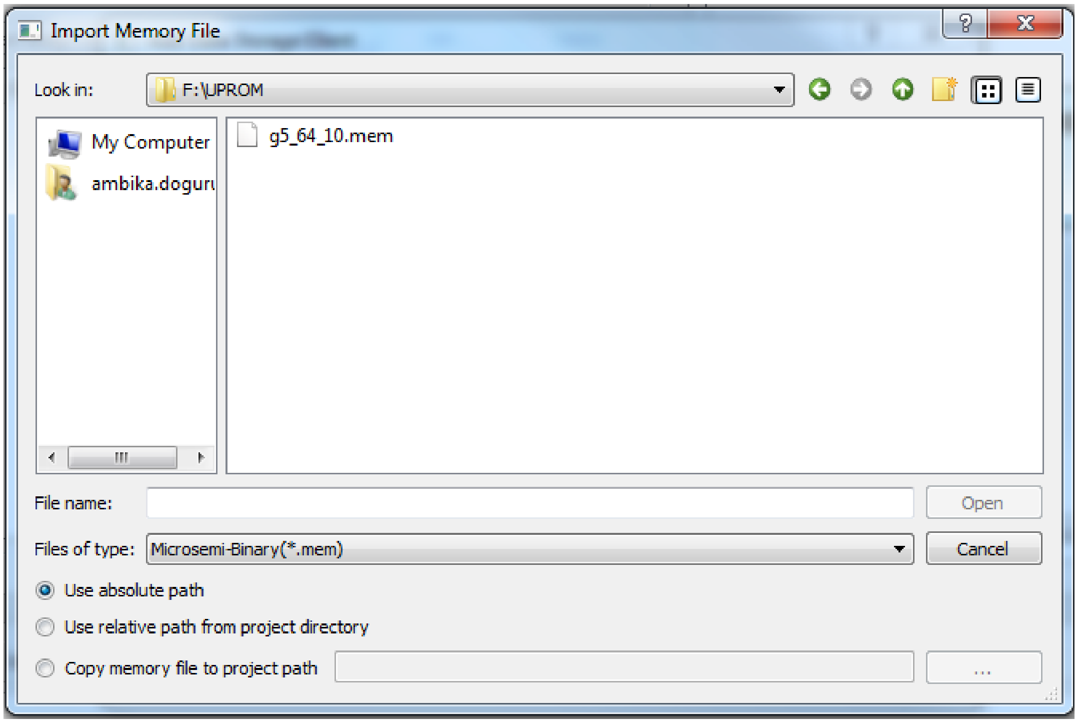
Enter a name for your memory client.

4.3.3.2.2 Content from File

Import the memory client from a memory file with this option. Click **Browse** to navigate to the location of the memory file and import. Select the Memory File and click **Open**.

Note: The memory file must have the `*.mem` file extension.

Figure 43 • **Import Memory File Dialog Box**



Import Memory File

Look in: F:\UPROM

g5_64_10.mem

File name:

Files of type: Microsemi-Binary(*.mem)

● Use absolute path

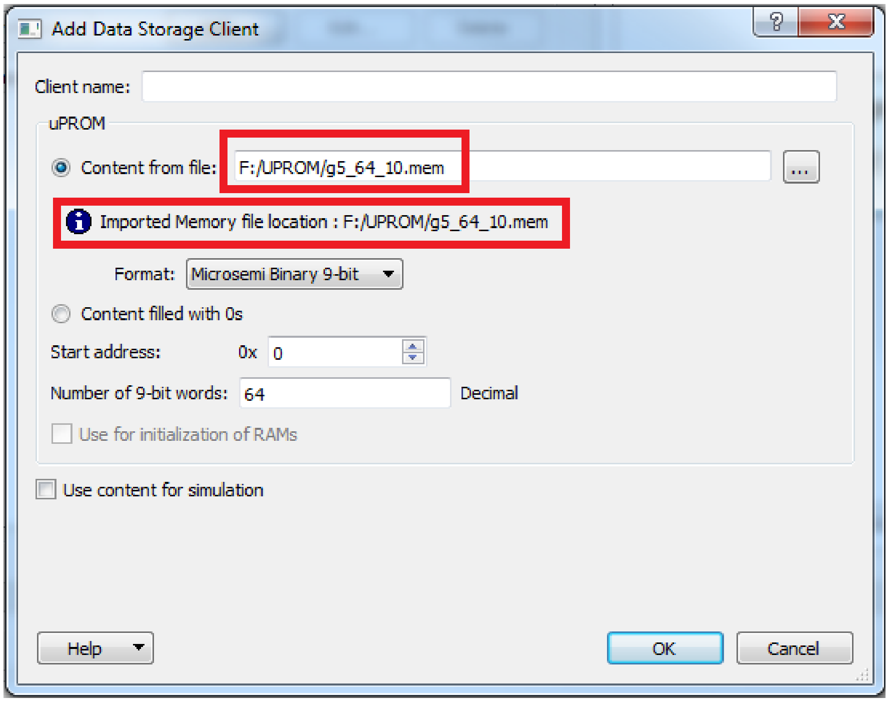
○ Use relative path from project directory

○ Copy memory file to project path

Use Absolute Path

When this is selected, the absolute path of the memory file appears in the **Content from File** field.

Figure 44 • **Absolute Path of Memory File**



Add Data Storage Client

Client name:

uPROM

● Content from file: F:/UPROM/g5_64_10.mem

Imported Memory file location : F:/UPROM/g5_64_10.mem

Format: Microsemi Binary 9-bit

○ Content filled with 0s

Start address: 0x 0

Number of 9-bit words: 64 Decimal

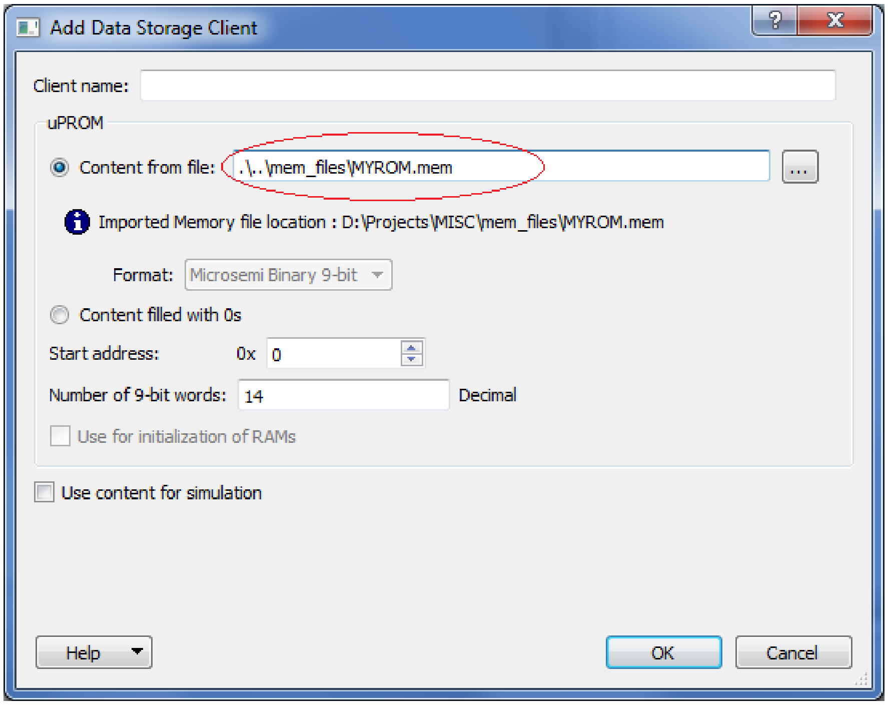
□ Use for initialization of RAMs

□ Use content for simulation

Use Relative Path from Project Directory

When this is selected, the Relative Path of the Memory File (relative to the Project location) is displayed in the **Content from File** field.

Figure 45 • **Relative Path of Memory File**



Add Data Storage Client

Client name:

uPROM

● Content from file: .\..\mem_files\MYROM.mem

Imported Memory file location : D:\Projects\MISC\mem_files\MYROM.mem

Format: Microsemi Binary 9-bit

○ Content filled with 0s

Start address: 0x 0

Number of 9-bit words: 14 Decimal

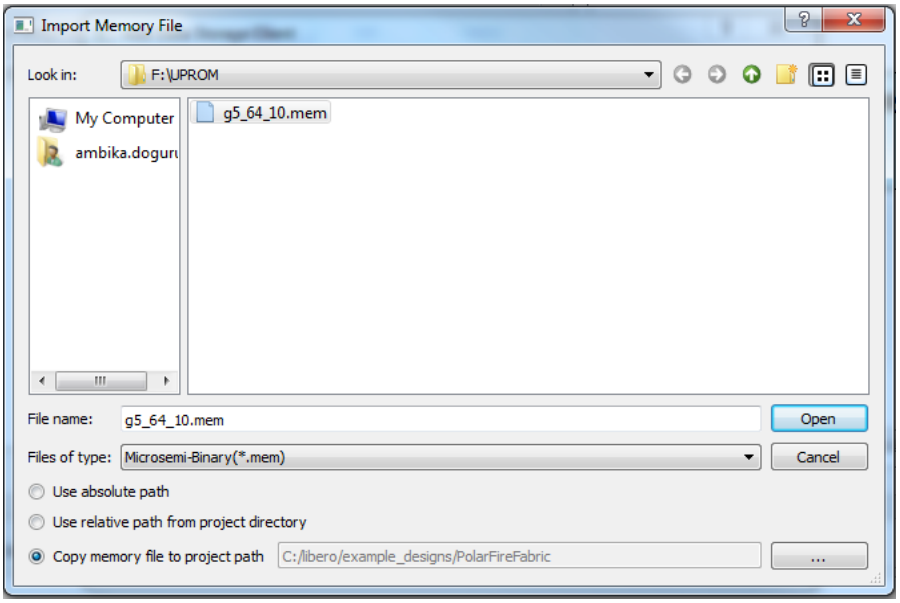
□ Use for initialization of RAMs

□ Use content for simulation

Copy Memory File to Project Path

Select this option and click **Browse** to navigate to the location of the memory file to copy from. The memory file is copied to the project location.

Figure 46 • **Location of Memory File to Copy From**



Look in: F:\UPROM

g5_64_10.mem

File name: g5_64_10.mem

Files of type: Microsemi-Binary(*.mem)

○ Use absolute path

○ Use relative path from project directory

● Copy memory file to project path C:/libero/example_designs/PolarFireFabric

Note: On the Windows systems, if the memory file and the Project location are on different drives, the Absolute Path is used even if Relative Path is selected.

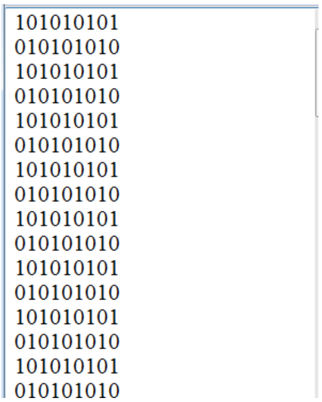
The memory file cannot be copied to and stored in the project's subfolders: component, smartgen, synthesis, designer, simulation, stimulus, tool data, and constraint. To prevent users from inadvertently copying the memory file into these sub-folders, these project subfolders are hidden from view when you select the project folder. Copy the memory file to the same project folder as the `*.prjx` file.

Note: The copied Memory File path is internally stored as relative path. Once this is copied to the project, user must update the content of the Memory File to make it current.

μPROM supports only the Microsemi Binary format (`*.mem`) for the memory content. The `*.mem` file must meet the following requirements:

- Each row is one 9-bit binary word (only 0s and 1s).
- The number of rows in the file (word count) must be less than or equal to the memory space of the μPROM (up to 58,368 words).
- The memory file must have the `*.mem` file extension. Figure 47 shows an example memory file.

Figure 47 • **Microsemi Binary File (`*.mem`) Example**



```
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
```

4.3.3.2.3 Content filled with 0s

Fill the content of the memory client with 0s as a place holder and update the memory client after Place and Route and before Programming. There is no need to rerun Place and Route after updating the µPROM Memory Content. For more information, see Update µPROM Memory Content.

4.3.3.2.4 Start Address

Enter the Start address (16-bit) of the client in HEX.

4.3.3.2.5 Number of 9-bit Words

Enter the size of the client (displayed as the number of 9-bit words) in decimal.

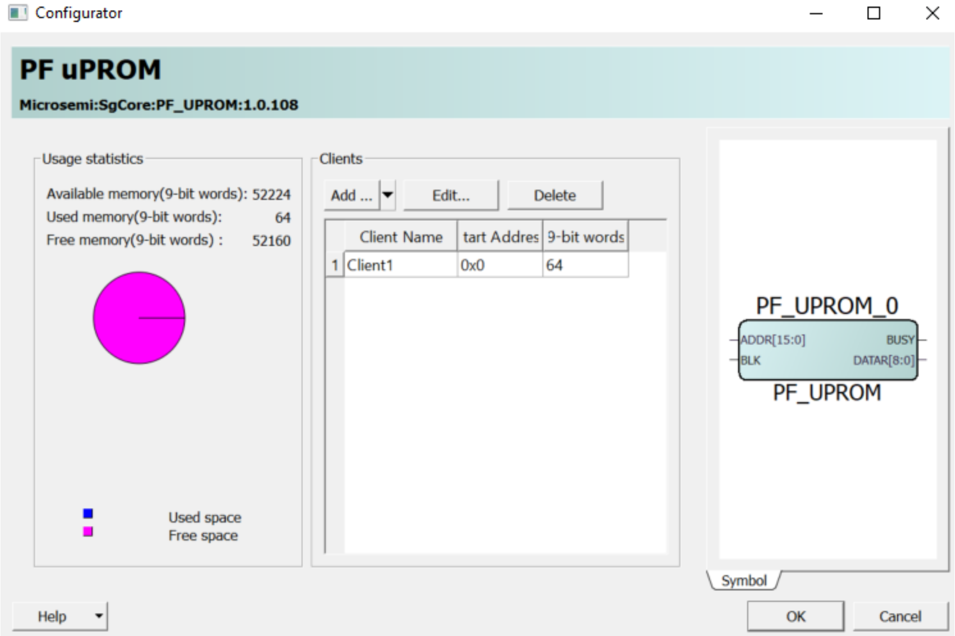
Note: When multiple clients are added, ensure that the address range of each client does not overlap with the other clients. Overlapping of address range is not allowed and is flagged as an error when it occurs.

4.3.3.2.6 Use Content for Simulation

Select to include the memory content for simulation. When this checked, a `UPROM.mem` file is automatically created in the *<prj_location>/simulation* folder when simulation is invoked in the **Design Flow** window. The `UPROM.mem` file is read by the µPROM simulation model to initialize the µPROM content when the simulation starts. Only clients with the “Use Client for Simulation” check box checked have the contents added to the `UPROM.mem` file for simulation.

Figure 48 shows the added clients under **User clients in µPROM**.

Figure 48 • **User Clients**



Configurator

PF uPROM

Microsemi:SgCore:PF_UPROM:1.0.108

Usage statistics

Available memory(9-bit words): 52224
Used memory(9-bit words): 64
Free memory(9-bit words) : 52160

Clients

	Client Name	tart Addres	9-bit words
1	Client1	0x0	64

4.3.3.3 DRC Rules and Error Messages

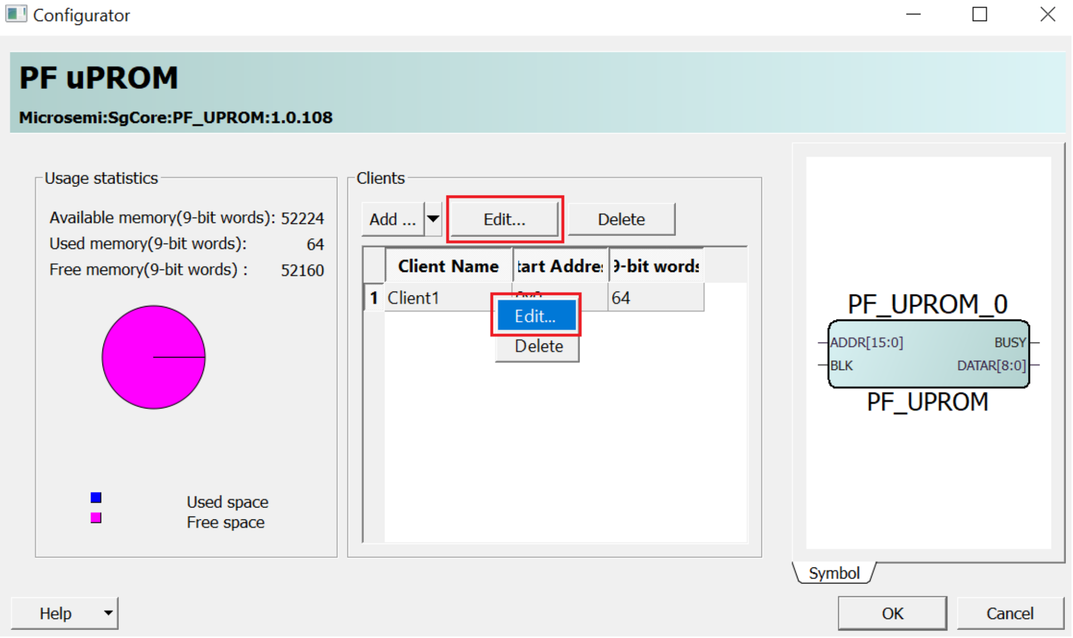
To prevent out-of-bound memory addressing and overlapping of address space, DRC rules are enforced and error messages are given when:

- An invalid start address (outside of the μPROM memory space) is entered.
  DRC Error: The specified start address is invalid; legal addresses range from 0x0 to <max_possible_address_for_the_die>.
- The start address and the number of words entered put the user client beyond the memory space of the μPROM.
  DRC Error: For the specified start address, the number of words cannot exceed the total number of words of <max_possible_words_for_die>.
- The number of 9-bit words entered is less than the number of words in the memory file used to fill the content of the client.
  DRC Error: The number of words cannot be less than the number of words <mem_file_word_count> specified in the memory file <mem_file_name>.
- There is more than one user client and the address range of one client overlaps with that of another.
  DRC Error: This client overlaps with: <client name >.
- The memory file (`*.mem`) size exceeds the total μPROM memory space.
  DRC Error: The memory file <memoryFileName> size exceeds the total μPROM space.

4.3.3.4 Editing a Client

1. To edit a client, click **Edit** or right-click the client name and select **Edit** to open the **Edit Data Storage Client** dialog box.

Figure 49 • **Editing User Clients**

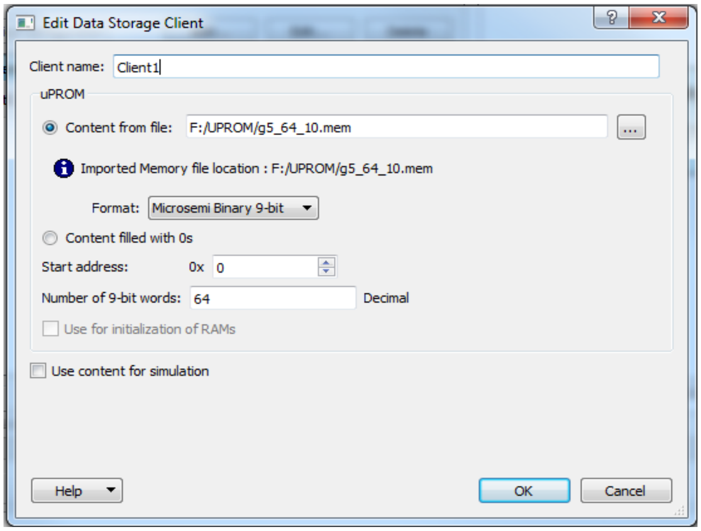


Available memory(9-bit words): 52224
Used memory(9-bit words): 64
Free memory(9-bit words) : 52160

	Client Name	tart Addre:	)-bit word:
1	Client1	0x0	64

2. Make changes in the Edit Data Storage Client dialog box and click **OK** to save edits.

Figure 50 • **Edit Data Storage Client Dialog Box**



Edit Data Storage Client

Client name: Client1

uPROM

● Content from file: F:/UPROM/g5_64_10.mem

Imported Memory file location : F:/UPROM/g5_64_10.mem

Format: Microsemi Binary 9-bit

○ Content filled with 0s

Start address: 0x 0

Number of 9-bit words: 64 Decimal

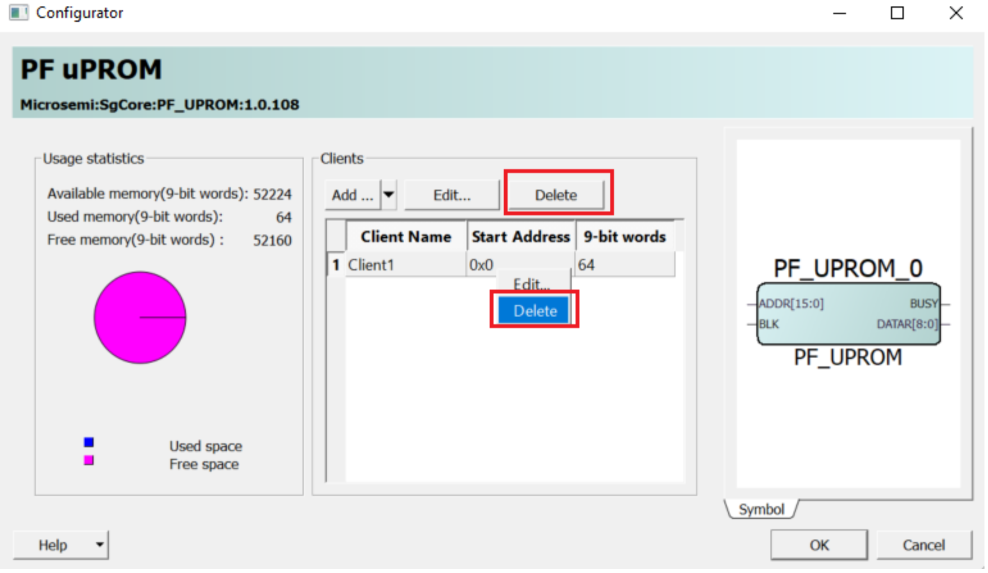
□ Use for initialization of RAMs

□ Use content for simulation

4.3.3.5 Deleting a Client

Right-click the client and select **Delete**.

Figure 51 • **Deleting a Client**



Configurator

PF uPROM

Microsemi:SgCore:PF_UPROM:1.0.108

Usage statistics

Available memory(9-bit words): 52224
Used memory(9-bit words): 64
Free memory(9-bit words) : 52160

■ Used space
■ Free space

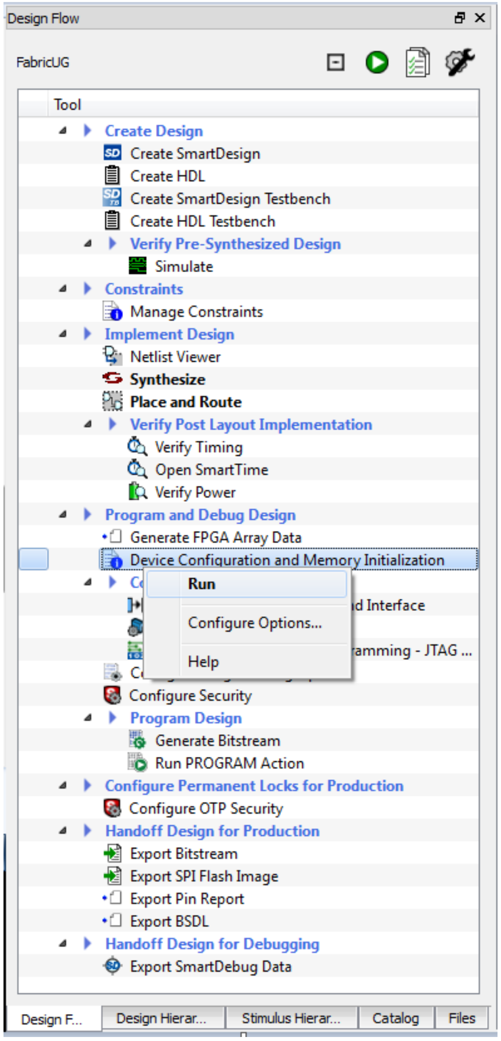
Clients

	Client Name	Start Address	9-bit words
1	Client1	0x0	64

4.3.3.6 Update µPROM Memory Content

The µPROM Memory Content can be updated after Place and Route using **Device Configuration and Memory Initialization** under **Program and Debug Design** in the Design Flow tab. For more information about updating µPROM memory content, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide* .

Figure 52 • **Update µPROM Memory Content**



4.4 sNVM

PolarFire devices include 56 KBytes of sNVM. The sNVM is organized into 221 pages of 236 bytes or 252 bytes depending on whether the data is stored as plain text or encrypted/authenticated data. Pages within the sNVM can be marked as ROM during bitstream programming. Data written to the sNVM can be protected by the PUF. The sNVM is readable and writable by the designer’s application during runtime and is an ideal storage location for locating the boot code for soft processors and user keys.

4.4.1 Implementation

The sNVM is not accessible to the fabric logic. This can be accessed through CoreSysService IP using system service calls. The sNVM content can be used for device initialization for LSRAM, µSRAM, PCIe, and transceiver data. Few pages of available 56 Kb are used for storing the device and peripheral configuration data and remaining pages are available for the user data. For more information about device initialization using sNVM, see *UG0725: PolarFire FPGA Device Power-Up and Resets User Guide*.

5 Math Blocks

PolarFire fabric includes embedded math blocks optimized for Digital Signal Processing (DSP) applications such as Finite Impulse Response (FIR) filters, Infinite Impulse Response (IIR) filters, Fast Fourier Transform (FFT) functions, and encoders that require high data throughput.

The math block has a built-in multiplier, a pre-adder, and an adder. These built-in features minimize the external logic required to implement multiplication, multiply-add, and multiply-accumulate (MACC) functions. For more information about math block inference by Synplify Pro ME, see *Inferring Microsemi MACC Blocks Application Note*. Implementation of these arithmetic functions using mathblocks results in efficient resource usage and improved performance for DSP applications. Math blocks can also be used in conjunction with fabric logic and embedded memories (LSRAM, µSRAM, and µPROM) to implement complex DSP algorithms.

5.1 Features

Key features of the Math block are as follows:

- High-performance and power optimized multiplication operations.
- Full-precision 48-bit output width.
- Supports 18 x 19 signed multiplication.
- Supports 17 x 18 unsigned multiplications.
- Supports input and output pipeline registers.
- Supports dot-product (DOTP) mode.
- Supports Single-Instruction Multiple-Data (SIMD) mode (dual-independent mode).
- Internal pre-adder block enables the efficient implementation of symmetric filters.
- Supports input cascade chain to form the tap-delay line for filtering applications.
- Built-in addition, subtraction, and accumulation units to combine multiplication results efficiently.
- Independent 48-bit registered third input.
- Supports signed and unsigned operations.
- Internal cascade signals (48-bit CDIN and CDOUT) enable cascading of the math blocks.

5.2 Math Block Resources

Table 1 lists the number of Math blocks available in PolarFire devices.

5.3 Functional Description

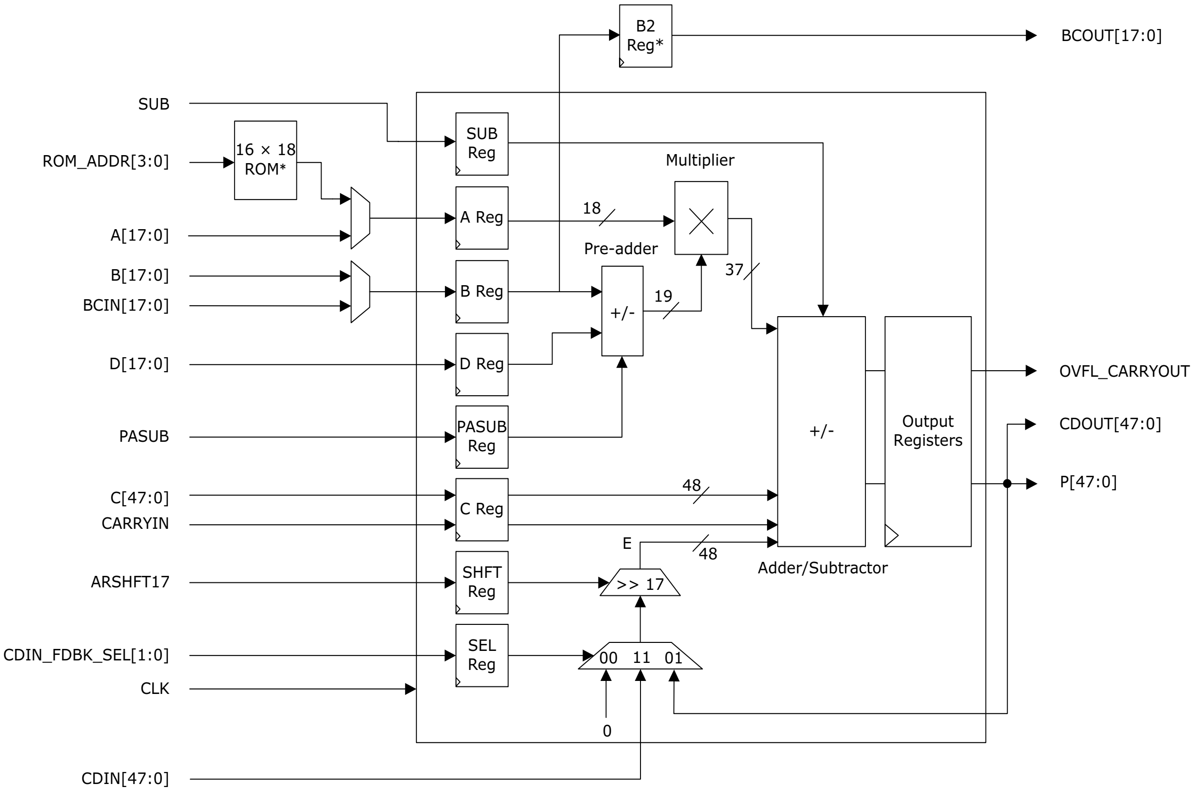
Math blocks are arranged in rows in the FPGA fabric and can be cascaded in a chain, starting from the left-most block to the right-most block within a row. For more information, see Figure 9.

Each math block consists of:

- Pre-Adder
- Multiplier
- Adder/Subtractor
- Math Block Ports
- 16 x 18 Coefficient ROM and B2 Register

Figure 53 shows the simplified block diagram of the math block.

Figure 53 • **Simplified Functional Block Diagram of Math Block**



Note: *B2 register, 16 × 18 coefficient ROM, and outside MUXes use the Interface Logic in the fabric.

Table 17 shows the port list for math block.

Table 17 • **Port List for Math Block**

Port Name	Direction	Type	Polarity	Description
A[17:0]	Input	Dynamic	Active high	Input data for operand A when USE_ROM = 0
ARSHFT17	Input	Dynamic	Active high	Arithmetic right-shift for operand E. When asserted, a 17-bit arithmetic right-shift is performed on operand E
B[17:0]	Input	Dynamic	Active high	Input data B to pre-adder with data D
B2[17:0]	Output	Dynamic	Active high	Pipelined output of input data B. Result P must be floating when B2 is used.
BCOUT[17:0]	Output	Cascade	Active high	Cascade output of B2. Value of BCOUT is the same as B2. The entire bus must either be dangling or drive an entire B input of another MACC_PA or MACC_PA_BC_ROM block.
C[47:0]	Input	Dynamic	Active high	Input data C When DOTP = 1, connect C[8:0] to CARRYIN. When SIMD = 1, connect C[8:0] to 0.
CLK	Input	Dynamic	Rising edge	Clock for A, B, C, CARRYIN, D, P, OVFL_CARRYOUT, ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers
CARRYIN	Input	Dynamic	Active high	CARRYIN for input data C
CDIN[47:0]	Input	Cascade	Active high	Cascaded input for operand E The entire bus must be driven by an entire CDOUT of another math block. In Dot-product mode, the driving CDOUT must also be generated by a math block in Dot-product mode.
CDIN_FDBK_SEL[1:0]	Input	Dynamic	Active high	Select CDIN, P, or 0 for operand E
CDOUT[47:0]	Output	Cascade	Active high	Cascade output of result P Value of CDOUT is the same as P. The entire bus must either be dangling or drive an entire CDIN of another math block in cascaded mode.
D[17:0]	Input	Dynamic	Active high	Input data D to pre-adder with data B When SIMD = 1, connect D[8:0] to 0.
OVFL_CARRYOUT	Output		Active high	OVERFLOW or CARRYOUT
P[47:0]	Output		Active high	Result data
PASUB	Input	Dynamic	Active high	Subtract operation for pre-adder of B and D
ROM_ADDR[3:0]	Input	Dynamic	Active high	Address of ROM data for operand A when USE_ROM = 1
SUB	Input	Dynamic	Active high	Subtract operation

Note: For information about asynchronous reset, synchronous reset, bypass, and enable signal details for each input/output register, see Table 37.

5.3.1 Pre-Adder

Pre-adder performs the upstream addition/subtraction operation prior to multiplication. It can be configured for the following operations on the inputs B[17:0] and D[17:0]:

- 18-bit addition/subtraction, producing the result B[17:0] ± D[17:0]. For more information, see Figure 58.
- Two 9-bit additions/subtractions, producing the results B[8:0] ± D[8:0], B[17:9] ± D[17:9]. For more information, see Figure 59. In the two 9-bit adder/subtractor modes, the type of operation (addition/subtraction) performed on the lower pre-adder of bits B[8:0] and D[8:0], and the type of operation performed on the upper pre-adder of bits B[17:9], D[17:9] must be the same.
- Single 9-bit addition/subtraction, producing the result B[17:9] ± D[17:9]. For more information, see Figure 60.

5.3.2 Multiplier

Inputs to multiplier are Port A data and pre-adder output data. The math block multiplier can be configured to perform any of the following:

- 19 x 18-bit multiplication.
- Two 10 x 9-bit multiplications (dot-product).
- Two independent multiplications, one 9 x 9-bit and one 10 x 9-bit SIMD/(dual-independent mode).

5.3.3 Adder/Subtractor

The adder/subtractor performs the final addition/subtraction and accumulation operation. This operation produces the final math block output with 48-bit precision.

The adder/subtractor can be configured to compute any of the following:

- (B ± D) x A + C + CARRYIN or (B ± D) x A + E.
- (B ± D) x A + C + CARRYIN + E.

If this block is configured as a subtractor, the output is (C + E) - (B ± D) x A.

5.3.4 Math Block Ports

Math blocks have built-in, by-passable registers on the data inputs (A, B, C, and D), data output (P), and control signals.

5.3.4.1 C Input and CARRYIN

The C input port allows the formation of several 3-input mathematical functions, such as 3-input addition or 2-input multiplication with addition. The CARRYIN signal is the carry input of the adder or accumulator. The C input can also be used as a dynamic input to achieve the following functionalities:

- Wrapping-around the cascade chain of math blocks from one row to the next row through the fabric.
- Rounding the multiplication outputs.
- Trimming the lower-order bits of the final sum, partial sum, or the product.

5.3.4.2 CDIN, CDOUT, and CDIN_FDBK_SEL

Higher-level DSP functions are supported by cascading individual math blocks in a row. The two data signals, CDIN[47:0] and CDOUT[47:0], provide the cascading capability with an input select (CDIN_FDBK_SEL). Table 19 lists the possible settings of CDIN_FDBK_SEL for propagating CDIN to the E input of the adder.

To cascade math blocks, the CDOUT of one block must feed the CDIN of an adjacent block. The CDOUT-to-CDIN connection is hardwired between the blocks within a row. Two different rows can be cascaded using fabric routing between two rows. Extra pipeline registers might be required to compensate for the extra delays added due to the fabric routing, which increases the latency of the chain.

The ability to cascade math blocks is useful in filter designs. For example, an FIR filter can be constructed by cascading inputs to arrange a series of input data samples and cascading outputs to arrange a series of partial output results. Since the general routing in the fabric is not used, the cascading ability provides a high-performance and low-power implementation of DSP filter functions. For more information, see Cascading Math Blocks.

5.3.4.3 BCOUT

BCOUT signal forms a chain in a row of math blocks for the B input. This signal can be used to form the input delay chain used in filter applications. The BCOUT signal is not hardwired and is routed using fabric routing.

5.3.4.4 OVFL_CARRYOUT

Each math block has an overflow signal, OVFL_CARRYOUT. This signal indicates any overflow from the addition operation performed by the adder. This signal is also used to extend the adder data widths from the existing 48 bits using fabric resources. It is also used to implement saturation capabilities. Saturation refers to catching an overflow condition and replacing the output with either the maximum (most positive) or minimum (most negative) value that can be represented.

Table 18 • **Truth Table for Computation of OVFL_CARRYOUT**

OVFL_CARRYOUT_SEL	OVFL_CARRYOUT	Description
0	(SUM[49] XOR SUM[48]) OR (SUM[48] XOR SUM[47])	True, if overflow or underflow occurred
1	C[47] XOR E[47] XOR SUM[48]	A signal that can be used to extend the final adder in the fabric.

5.3.4.5 ARSHFT17

For multi-precision arithmetic, math blocks provide a right shift by 17 that is controlled by the shift input, ARSHFT17. Hence, a partial product from one math block can be shifted to the right and added to the next partial product computed in an adjacent math block. Using this technique, math blocks can be used to build wide multipliers.

5.3.4.6 CDIN_FDBK_SEL

For accumulation operations, the math block output must be fed back to the E input of the adder block. Selection of this input is controlled by combinations of the CDIN_FDBK_SEL and ARSHFT17 inputs. Following is the truth table for operand E.

Table 19 • **Truth Table for Propagating Operand E of the Adder or Accumulator**

CDIN_FDBK_SEL	ARSHFT17	Operand E
00	0	0
00	1	0
01	0	P[47:0]
01	1	{17{P[47]}, P[47:18]}
11	0	CDIN[47:0]
11	1	{17{CDIN[47]}, CDIN[47:18]}

5.3.4.7 PASUB and SUB Inputs

The PASUB signal controls the mode of pre-adder (subtraction or addition). The SUB signal controls whether the multiplier product is to be subtracted or added.

Table 20 • **Truth Table for Computation of Result P and CDOUT**

SIMD	DOTP	SUB	PASUB	Result P and CDOUT
0	0	0	0	CARRYIN + C[47:0] + E[47:0] + { (B[17:0] + D[17:0]) x A[17:0] }
0	0	0	1	CARRYIN + C[47:0] + E[47:0] + { (B[17:0] - D[17:0]) x A[17:0] }
0	0	1	0	CARRYIN + C[47:0] + E[47:0] - { (B[17:0] + D[17:0]) x A[17:0] }
0	0	1	1	CARRYIN + C[47:0] + E[47:0] - { (B[17:0] - D[17:0]) x A[17:0] }
0	1	0	0	CARRYIN + C[47:0] + E[47:0] + { (B[8:0] + D[8:0]) x A[17:9] + (B[17:9] + D[17:9]) x A[8:0] } x 2^9
0	1	0	1	CARRYIN + C[47:0] + E[47:0] + { (B[8:0] - D[8:0]) x A[17:9] + (B[17:9] - D[17:9]) x A[8:0] } x 2^9
0	1	1	0	CARRYIN + C[47:0] + E[47:0] + { (B[8:0] + D[8:0]) x A[17:9] - (B[17:9] + D[17:9]) x A[8:0] } x 2^9
0	1	1	1	CARRYIN + C[47:0] + E[47:0] + { (B[8:0] - D[8:0]) x A[17:9] - (B[17:9] - D[17:9]) x A[8:0] } x 2^9

1	0	0	0	P[17:0] = CARRYIN + { B[8:0] x A[8:0] }
				P[47:18] = C[47:18] + E[47:18] + { (B[17:9] + D[17:9]) x A[17:9] }

1	0	0	1	P[17:0] = CARRYIN + { B[8:0] x A[8:0] }
				P[47:18] = C[47:18] + E[47:18] + { (B[17:9] - D[17:9]) x A[17:9] }

1	0	1	0	P[17:0] = CARRYIN + { B[8:0] x A[8:0] }
				P[47:18] = C[47:18] + E[47:18] - { (B[17:9] + D[17:9]) x A[17:9] }

1	0	1	1	P[17:0] = CARRYIN + { B[8:0] x A[8:0] }
				P[47:18] = C[47:18] + E[47:18] - { (B[17:9] - D[17:9]) x A[17:9] }

5.3.5 16 x 18 Coefficient ROM and B2 Register

Each math block has a 16 x 18 coefficient ROM for storing filter coefficients, and a B2 register for delay chains. By changing the address of the coefficient ROM, a set of coefficients can be cycled through for folded, interpolation or decimation filters, and/or switch between two more sets of filter coefficients. Three IL clusters are associated with each math block providing 36 LUTs and 36 flip-flops for connecting to the fabric and building complex structures. These ILs can be used to implement the following:

- 18 IL LUTs are used to implement a 16 x 18-bit coefficient ROM. This coefficient ROM feeds the data input of the IL registers associated with the A input of the math block and provides storage for 16 fixed coefficient values. These coefficients are loaded during device programming. The coefficient ROM content is accessed through ADDR[3:0] for multiplication.
- 18 ILs LUTs are used to implement a 2:1 mux for selecting the BCIN.
- 18 interface logic flip-flops are used to implement the B2 register, as shown in Figure 53. This register is used in input delay chains for filtering applications.

Note: The coefficient ROM is part of the interface logic associated with the math block. It is different from µPROM.

5.4 Cascading Math Blocks

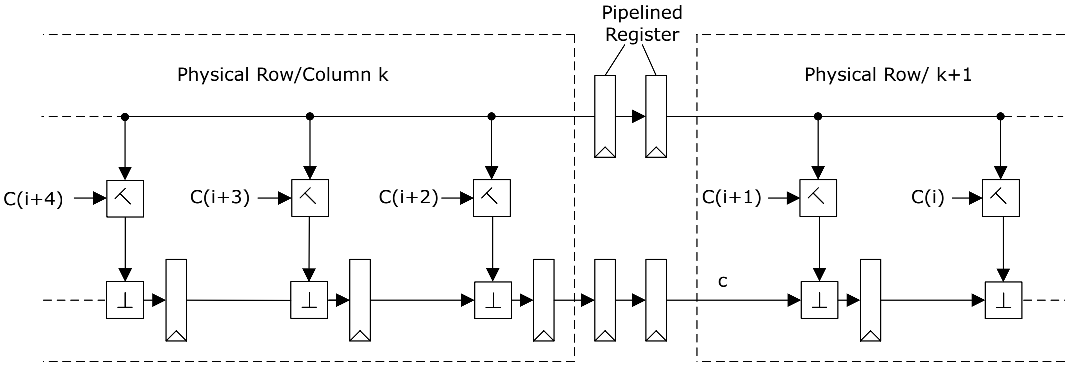
The math blocks in each row cascade in a chain from left to right and terminate at the end of each row. Fabric routing is required to extend the cascading chain from one row to another. The following are the two options to connect cascading chains across multiple rows:

- Wrap-Around to the Start of the Cascade Chain
- Wrap-Around to the Inside of the Cascade Chain

5.4.1 Wrap-Around to the Start of the Cascade Chain

The cascade chain output of one row is connected to the cascade chain input of another row through fabric routing, with one or more pipeline registers added in the fabric to improve performance. If latency is required at the output, additional compensating registers can be added on the input side. Figure 54 shows an example implementation of a transpose FIR filter with the cascade chain of one row being extended to the chain of another row through pipeline registers.

Figure 54 • **Wrap-Around of Cascade Chain Using Pipeline Registers in Fabric**

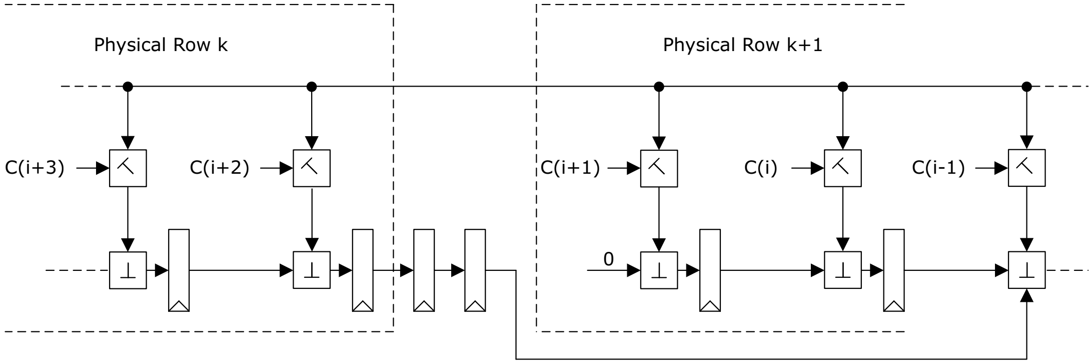


5.4.2 Wrap-Around to the Inside of the Cascade Chain

The cascade chain output of one row is connected to any adder input of another row. This type of connection is useful when designing systolic or folded FIR filters. With this approach, there is no need to insert compensating pipeline registers on the input side or create additional latency on the output side.

Figure 55 shows an example implementation of a wrap-around to the inside of a cascade chain.

Figure 55 • **Wrap-Around to Inside of Cascade Chain**



5.4.2.1 Merging Multiple Cascade Chains

Multiple cascade chains can be merged using either the C input of math block or adders constructed in the fabric. Extra pipelined register must be added to adjust the latency.

Figure 56 shows multiple cascaded chains merged using a math block C input.

Figure 56 • **Merging Cascade Chains Using C Input**

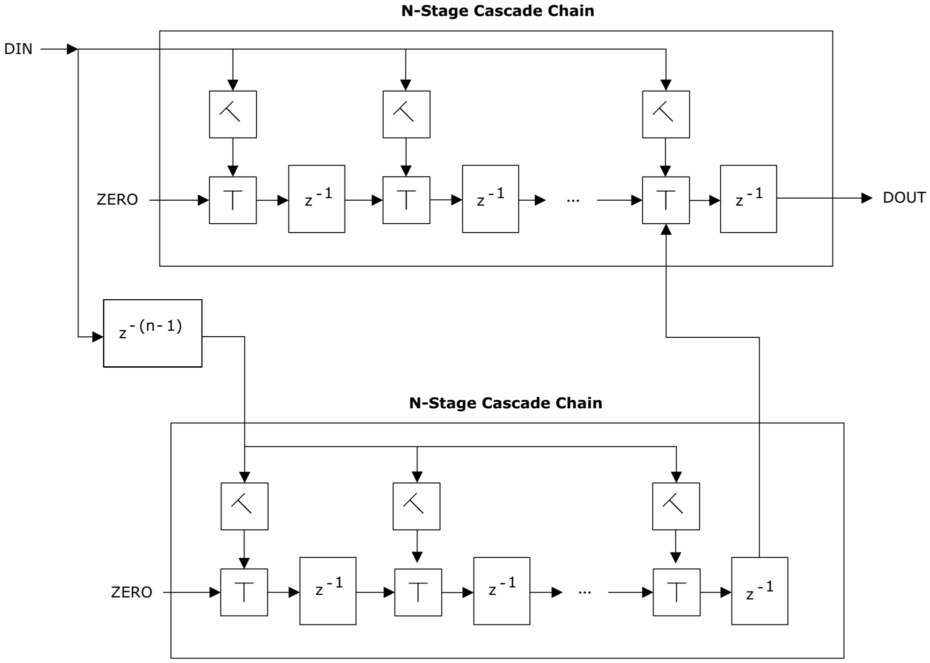
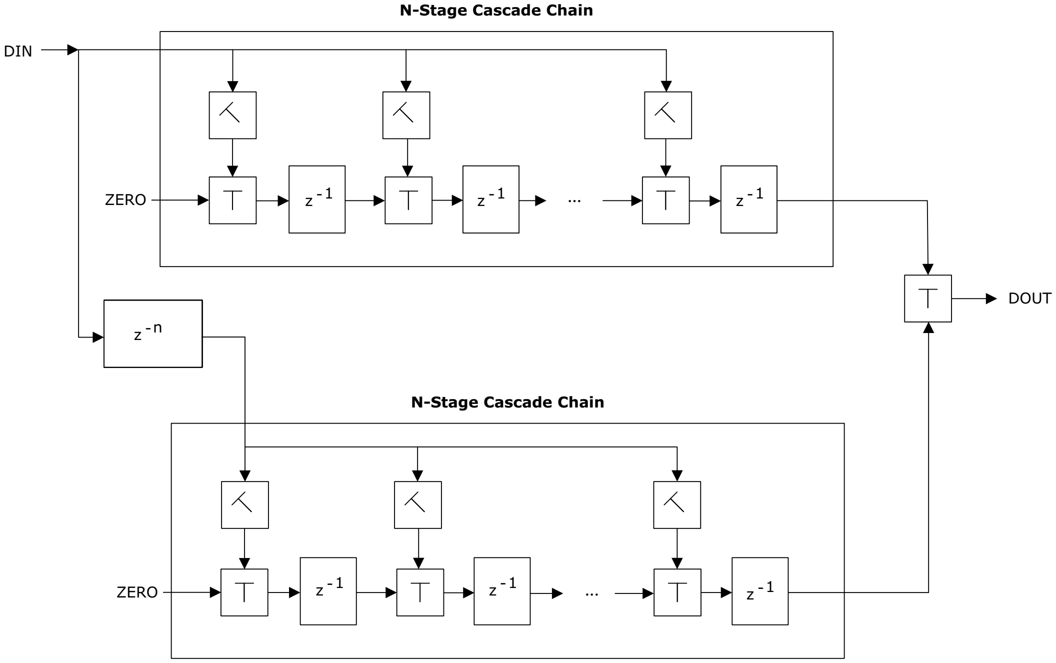


Figure 57 shows multiple cascaded chains merged with fabric adders.

Figure 57 • **Merging Cascade Chains Using Fabric Adders**



5.5 Operational Modes

The math block supports the following operational modes:

- Normal Mode
- DOTP Mode
- SIMD Mode

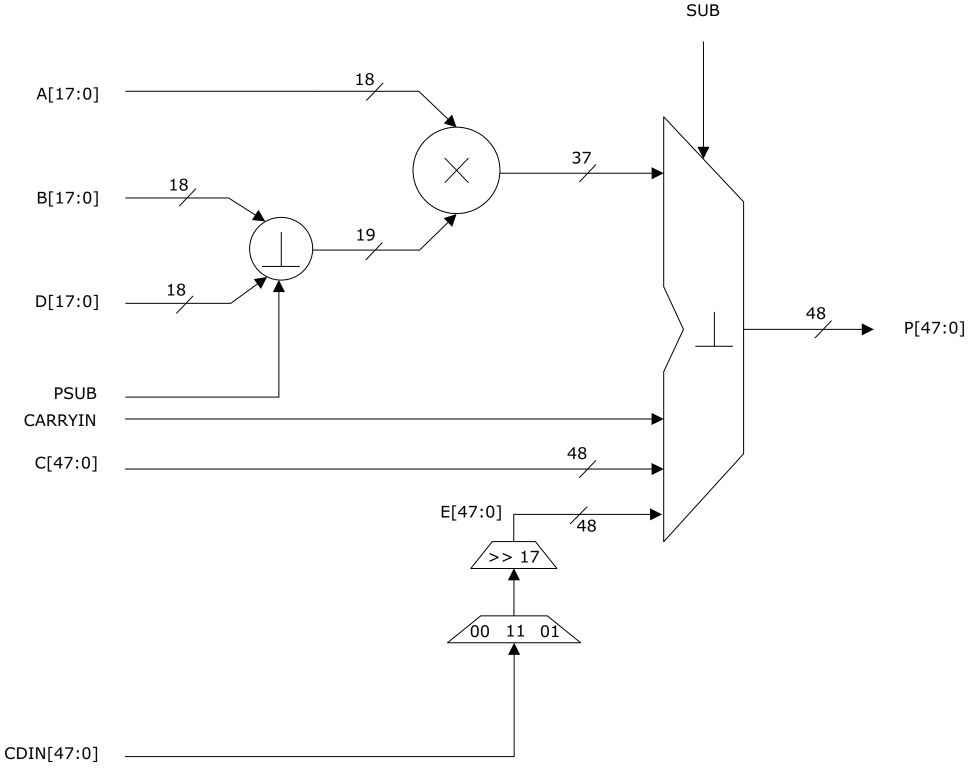
5.5.1 Normal Mode

Normal mode performs a 18-bit addition (pre-adder) and a 19 x 18 multiplication operation as per the following output equation:

P[47:0] = ((B[17:0] ± D[17:0]) x A[17:0]) + C[47:0] + E[47:0] + CARRYIN

Figure 58 shows the functional block diagram of the math block in normal mode.

Figure 58 • **Functional Block Diagram of the Math Block in Normal Mode**



5.5.2 DOTP Mode

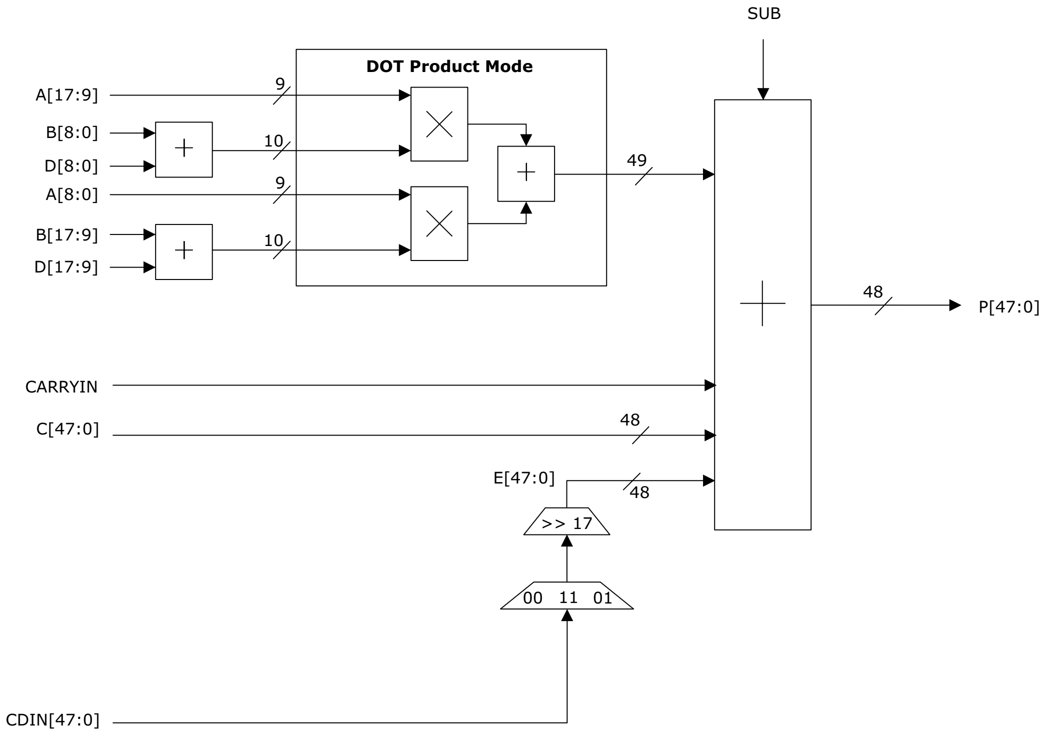
DOTP mode computes the sum of two 10 x 9-bit multiplications. In this mode, the pre-adder is split into two 9-bit adders. The two 9-bit pre-adders must be in the same mode, either addition or subtraction.

In DOTP mode, the math block implements the following output equation:

$P[47:0] = ((B[8:0] \pm D[8:0]) \times A[17:9] \pm (B[17:9] \pm D[17:9]) \times A[8:0]) \times 2^9 + C[47:0] + E[47:0] + CARRYIN$

Figure 59 shows the functional block diagram of the math block in DOTP mode.

Figure 59 • **Functional Block Diagram of the Math Block in DOTP Mode**



5.5.3 SIMD Mode

In SIMD mode, the 18 x 19 multiplier operates as two independent 9 x 9 multipliers. It doubles the number of multiplications for smaller widths (say 9 bits). It computes two independent 9 x 9-bit multiplications using the 9-bit upper pre-adder. The lower pre-adder is not used (D[8:0] input must be zero). The lower portion of the final adder is not used, that is, E[17:0] is ignored and C[17:0] must be zero. The ARSHFT17 input is also ignored in SIMD mode.

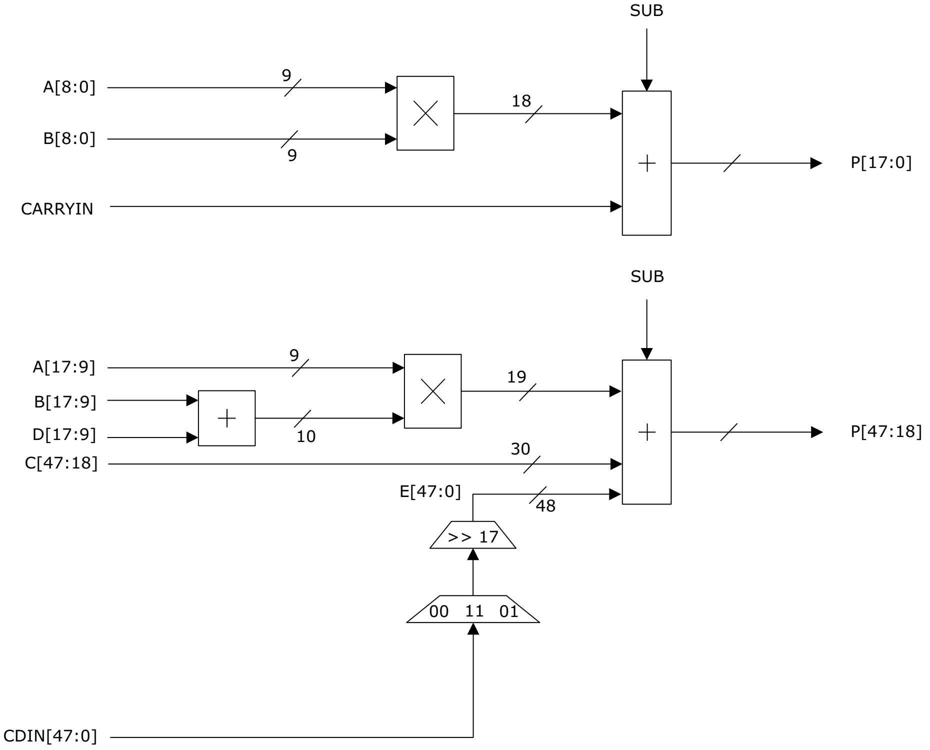
SIMD mode implements the following output equations:

P[17:0] = (B[8:0] x A[8:0]) + CARRYIN

P[47:18] = ((B[17:9] ± D[17:9]) x A[17:9]) + C[47:18] + E[47:18]

Figure 60 shows the functional block diagram of the math block in SIMD mode.

Figure 60 • **Functional Block Diagram of the Math Block in SIMD Mode**



5.6 Implementation

A math block is implemented through one of the following methods:

- RTL Inference
- Math Block Macro
- Use Models

5.6.1 RTL Inference

Synplify Pro ME infers math blocks and configures the appropriate modes automatically if the RTL contains any specific multiply, multiply-accumulate, multiply-add, or multiply-subtract, pre-adder multiply functions. Synthesis handles all the signal connections of the math block to the rest of the design and sets the correct values for the static signals needed to configure the appropriate operational mode. Synplify Pro ME ties unused dynamic input signals to ground and provides default values to unused static signals.

Synplify Pro ME automatically maps any multiplication functions with input widths of three or more to math blocks. For input widths less than three, mathblocks can be inferred by using synthesis attribute (Syn_multstyle= “dsp”), the mapping of multiplication functions with input widths less than three, (which are implemented in fabric logic by default), can be controlled by the synthesis attribute (Syn_multstyle = “dsp”) if using math blocks is preferred. The tool is also capable of cascading multiple math blocks if the function crosses the limits of a single math block. For example, if the RTL contains a 35 x 35 multiplication, synthesis implements this block using four math blocks cascaded in a chain.

Synplify Pro ME also has the capability to utilize the input and output registers inside the math block boundary, provided they are in the same clock domain. If the registers are in different clock domains, the clock that drives the output register has priority, and all registers driven by that clock are placed in the math block. If the outputs are unregistered and the inputs are registered but not all on the same clock domain, the input registers with the larger input have priority and are placed in the math block. The synthesis tool supports inferencing of math block components across hierarchical boundaries. In this case even if the multipliers, input registers, output registers, and adders/subtractors are present at different levels of a design's hierarchy, they can be placed into the same math block. For more information about math block inference by Synplify Pro ME, see *Inferring Microsemi MACC Blocks Application Note*.

5.6.2 Math Block Macro

The math block macro is available in the Libero SoC -> IP Catalog as a component that can be used directly in an HDL file or instantiated in SmartDesign. Math block has the following two macros.

- MACC_PA (MACC with Pre-Adder)
- MACC_PA_BC_ROM (MACC with Pre-Adder, BCOUT Register, and Coefficient ROM)

For more information about configuring math block macro, see Math Block Macro.

5.6.3 Use Models

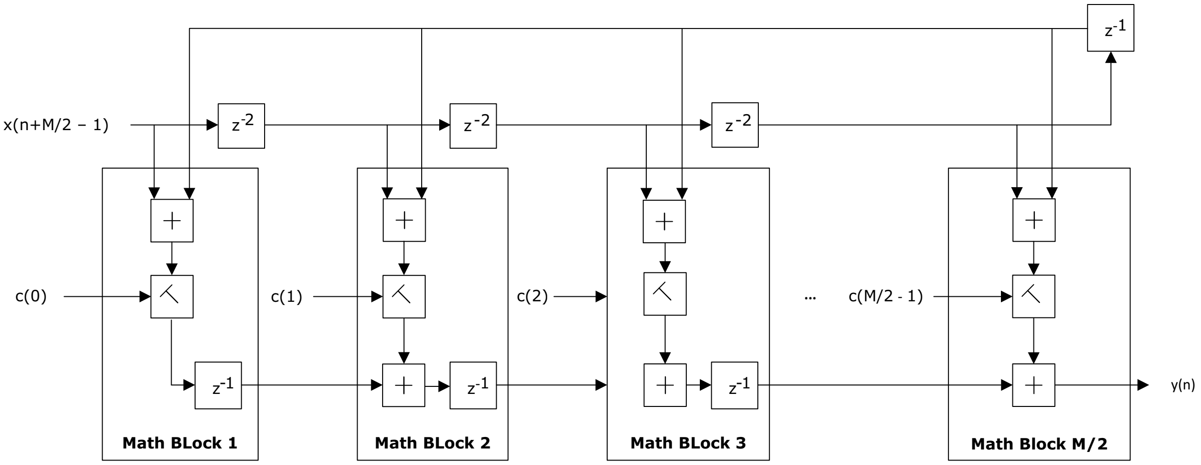
Math blocks can be used in a wide variety of DSP applications and this section describes the following use models:

- Symmetric FIR Filter
- 9 x 9 Systolic FIR Filter
- 9 x 9 Symmetric FIR Filter
- 9 x 9 Complex Multiplier
- Non-Pipelined 35 x 35 Multiplier Using Cascade Chain
- Pipelined 35 x 35 Multiplier Using Cascade Chain
- Non-Pipelined 35 x 35 Multiplier Using Fabric Adders
- Extension Adder

5.6.3.1 Symmetric FIR Filter

In symmetrical FIR filters, the coefficients are symmetrical. Because of the symmetry, an M-tap FIR filter can be implemented using M/2 math blocks. The math block pre-adder is used to add the input signal feeding the symmetric filter coefficients, as shown in Figure 61.

Figure 61 • **Symmetric FIR Filter Implementation**

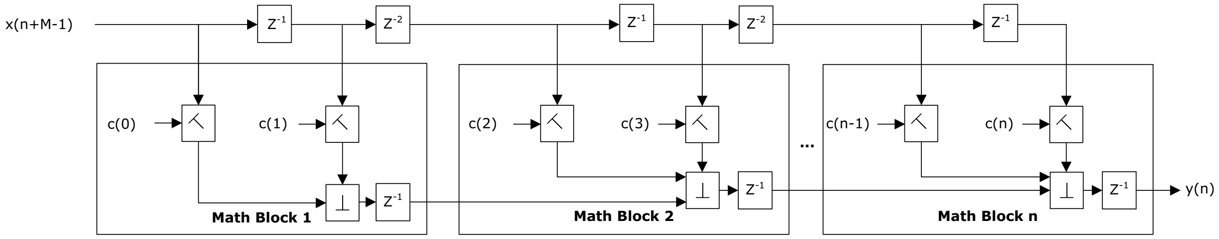


Anti-symmetric FIR filters can also be implemented in the same way, but with the pre-adder configured to subtract the input signals feeding the anti-symmetric filter coefficients, as shown in Figure 62.

5.6.3.2 9 x 9 Systolic FIR Filter

A 9 x 9 systolic FIR filter can be implemented using the 9 x 9 DOTP mode of the math block, see Figure 62. In DOTP mode, two multipliers share the same output register. As a result, the latencies have to be adjusted properly on the input side to achieve correct filter operation.

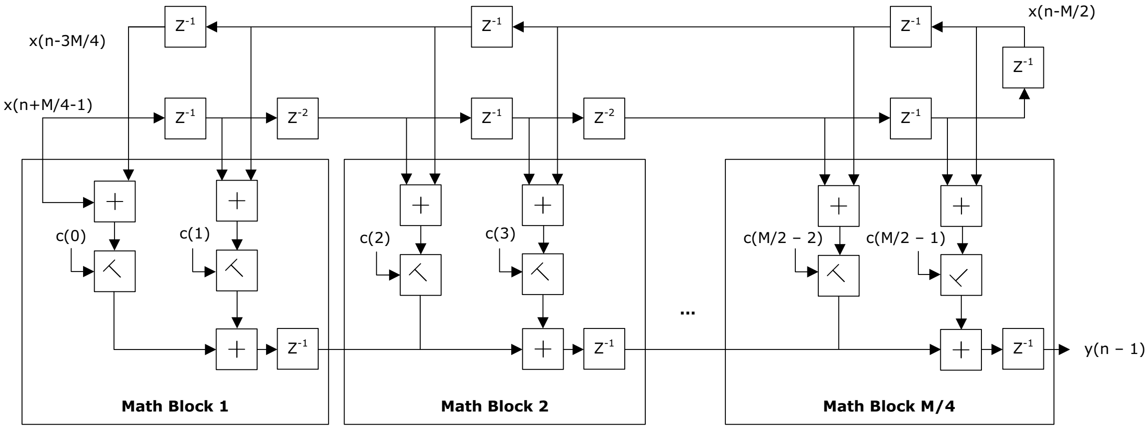
Figure 62 • **9 x 9 Systolic FIR Filter**



5.6.3.3 9 x 9 Symmetric FIR Filter

A 9 x 9 symmetric FIR filter can be implemented by using the 9 x 9 DOTP mode with the math block pre-adder configured as two 9-bit pre-adders. Figure 63 shows an example of a 9 x 9 symmetric FIR filter for M number (divisible by 4) of taps.

Figure 63 • **9 x 9 Symmetric FIR Filter**



5.6.3.4 9 x 9 Complex Multiplier

To implement a 9 x 9 complex multiplier, 2 math blocks and an additional 2's complement logic (in the fabric) are required. The 2's complement logic in the fabric is needed for negating the Q input, as shown in Figure 64, and this logic consumes minimal fabric resources.

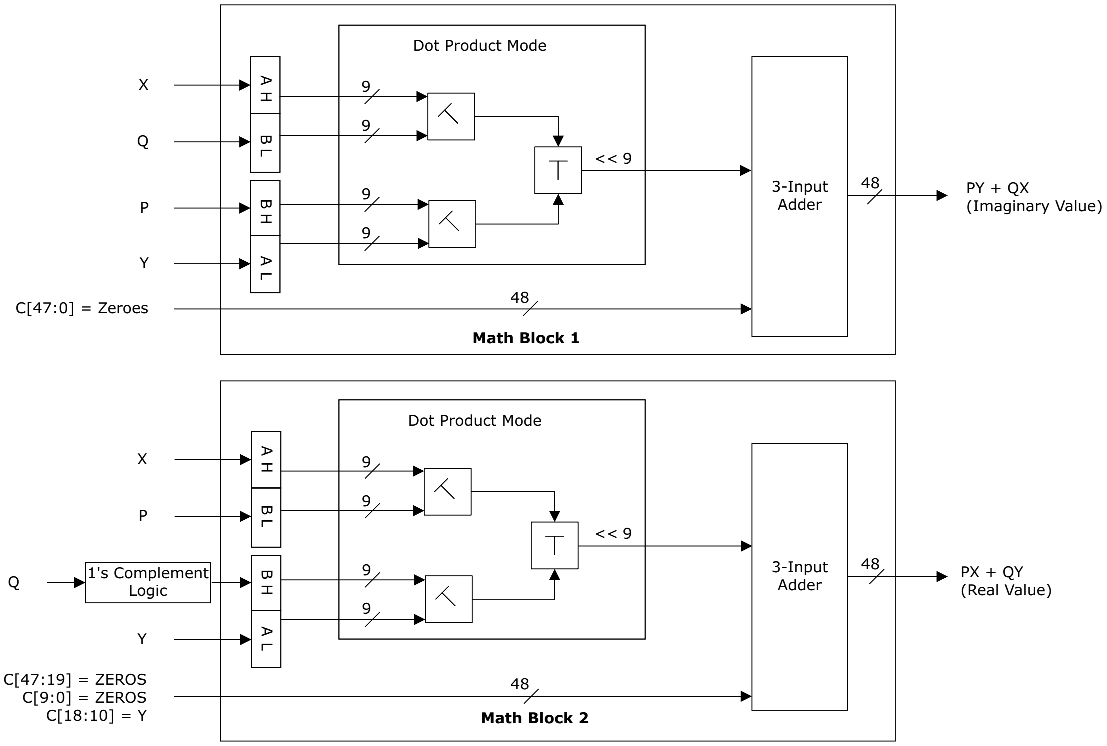
For two complex numbers, X + jY and P + jQ, the complex multiplication is:

Multiplication result = real part + imaginary part = (PX – QY) + j (PY + QX).

The real part (PX - QY) requires -Q for the multiplication result. In 2's complement arithmetic this value can be computed using the 1's complement of Q and adding the Y using the C input (-Q = ~Q + 1).

Real part = P x X + (~Q) x Y + Y.

Figure 64 • **9-Bit Complex Multiplication Using DOTP Mode**

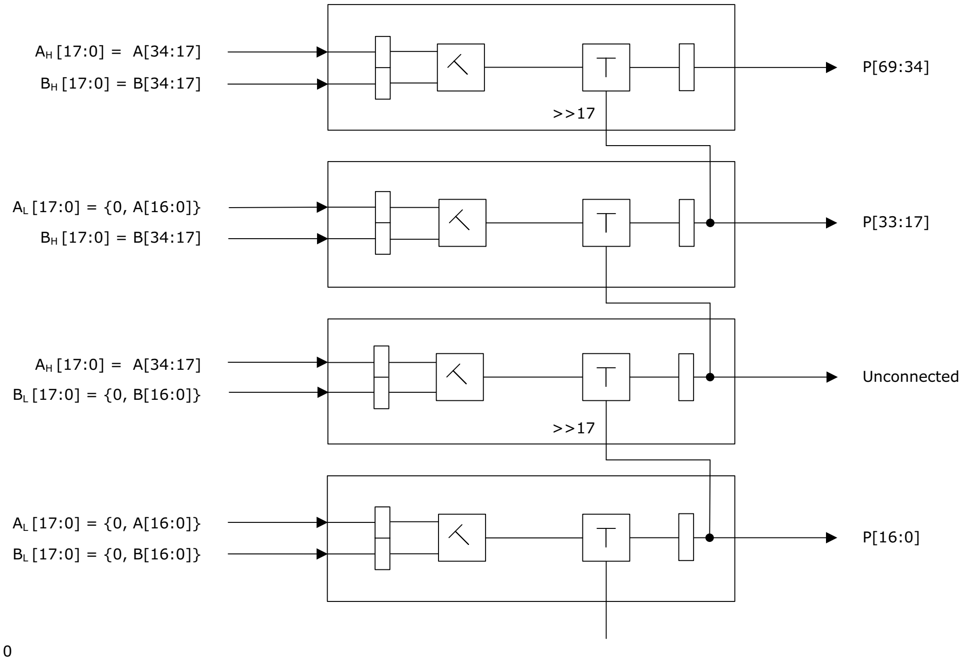


5.6.3.5 Non-Pipelined 35 x 35 Multiplier Using Cascade Chain

A 35 x 35 multiplier is useful for applications requiring more than 18-bit precision. A non-pipelined implementation is typically used for low-speed applications. A non-pipelined 35 x 35 multiplier can be implemented using the cascaded math blocks in a single row. Figure 65 shows an example of such an implementation.

The inputs are assumed to be A[34:0] and B[34:0] with a product of P[69:0].

Figure 65 • **Non-Pipelined Implementation Using Cascade Chain**

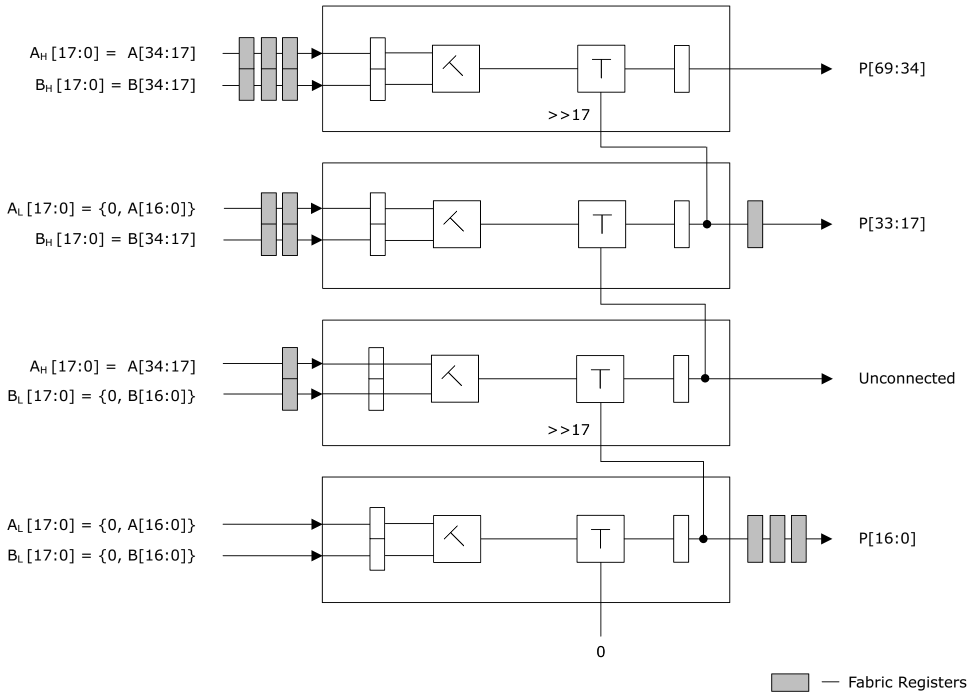


5.6.3.6 Pipelined 35 x 35 Multiplier Using Cascade Chain

Math blocks have IL registers accessible to all input and output ports. To implement pipelined multipliers, these IL registers are added to the input or output side of the non-pipelined implementation. These registers are needed for balancing the pipeline latency.

Figure 66 shows a typical 35 x 35 multiplier implementation with fabric pipeline registers.

Figure 66 • **Pipelined Implementation Using Cascade Chain**

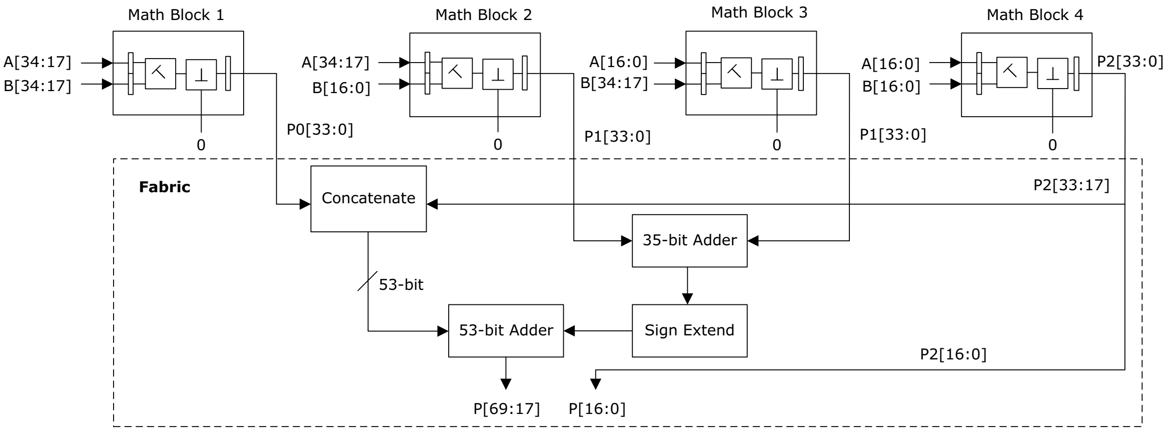


5.6.3.7 Non-Pipelined 35 x 35 Multiplier Using Fabric Adders

A non-pipelined 35 x 35 multiplier can be implemented using fabric adders. This reduces the output latency compared to the non-pipelined cascade-chain implementation.

Figure 67 shows the block diagram of the 35 x 35 multiply using fabric adders.

Figure 67 • **35 x 35 Multiply Using Fabric Adders**



5.6.3.8 Extension Adder

A math block's adder/accumulator width can be extended using fabric logic. Figure 68 and Figure 69 show example implementations of 2-input and 3-input extension adders in fabric. In this extension adder, the MSB bits, P[47], C[47], and E[47], are not treated as sign bits. Instead, P[n-1] represents the sign bit. The static input EXT_SEL is set to 1 causing the appropriate signal to appear on each math block's OVFL_CARRYOUT output.

Figure 68 shows a case where input C = 0, and input E is used for the cascade chain.

Figure 68 • **Extension of 2-Input Adder in Fabric**

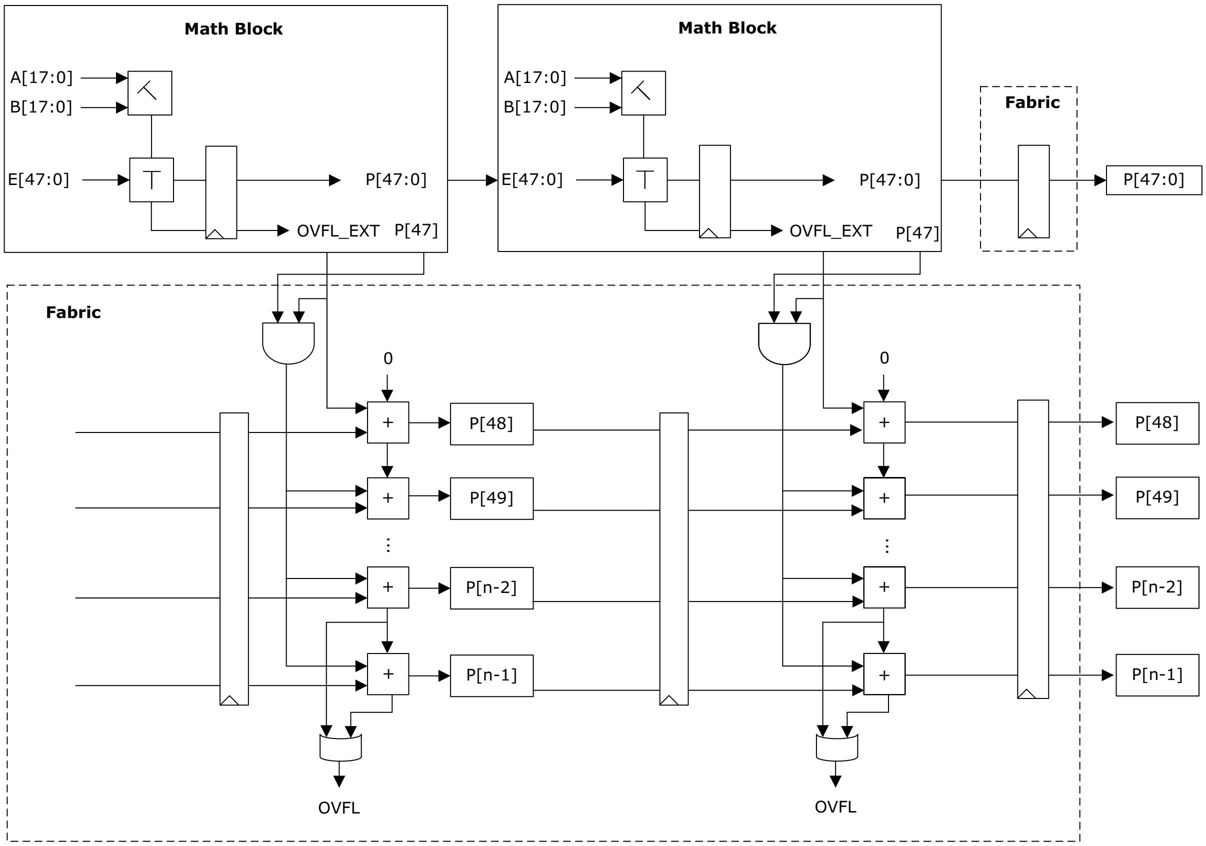
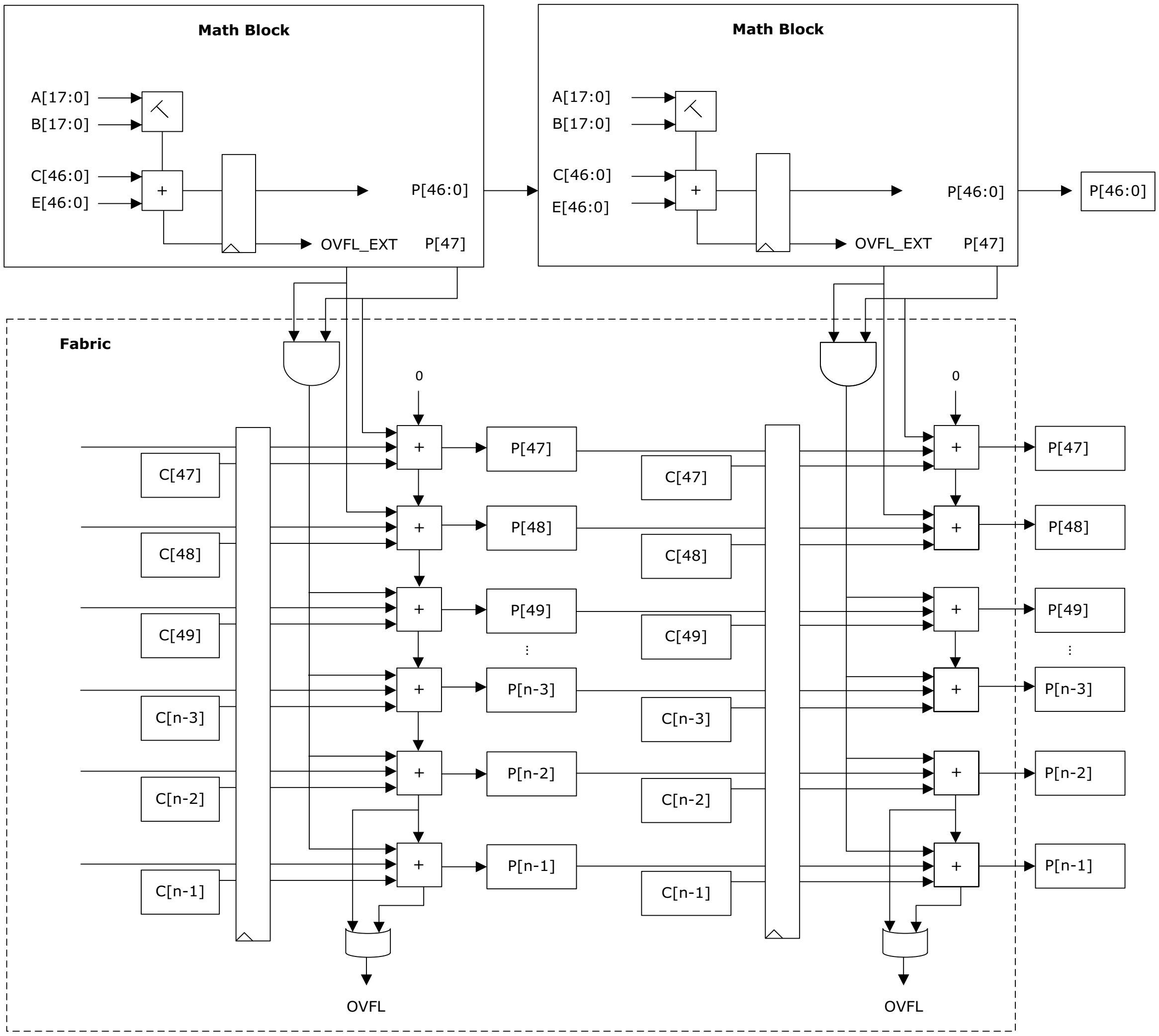


Figure 68 shows a case where both inputs C and E are used.

Figure 69 • Extension of 3-Input Adder in Fabric



6 Appendix: Supported Memory File Formats for LSRAM and µSRAM

Microchip supports Intel-Hex and Motorola-S file formats. Implementation of these formats interprets data sets in bytes. If the memory width is 7 bits then every 8th bit in the data set is ignored. If the data width is 9 bits, two bytes are assigned to each memory address, and the upper 7 bits of each 2-byte pair are ignored.

The following examples illustrate how the data is interpreted for various word sizes:

FF 11 EE 22 DD 33 CC 44 BB 55 (where 55 is the MSB and FF is the LSB) for 32-bit word size:

```
0x22EE11FF (address 0)
0x44CC33DD (address 1)
0x000055BB (address 2)
```

For 16-bit word size:

```
0x11FF (address 0)
0x22EE (address 1)
0x33DD (address 2)
0x44CC (address 3)
0x55BB (address 4)
```

For 8-bit word size:

```
0xFF (address 0)
0x11 (address 1)
0xEE (address 2)
0x22 (address 3)
0xDD (address 4)
0x33 (address 5)
0xCC (address 6)
0x44 (address 7)
0xBB (address 8)
0x55 (address 9)
```

For 9-bit word size:

```
0x11FF -> 0x01FF (address 0)
0x22EE -> 0x00EE (address 1)
0x33DD -> 0x01DD (address 2)
0x44CC -> 0x00CC (address 3)
0x55BB -> 0x01BB (address 4)
```

Note: For 9-bit, the upper 7-bits of the 2-bytes are ignored.

INTEL-HEX

Intel Hex is an industry standard file format created by Intel. File extensions for these files are `.hex` and `.ihx`.

Memory contents are stored in ASCII files using hexadecimal characters. Each file contains a series of records (lines of text) delimited by new line, '\n', characters and each record starts with a ':' character. For more information about this format, see the Intel-Hex Record Format Specification.

The Intel Hex record is composed of five fields and is arranged as:

```
:llaaaatt[dd...]cc
```

Where:

- :—start code of every Intel Hex record.
- ll—byte count of the data field.
- aaaa—16-bit address of the beginning of the memory position for the data. Address is big endian.
- tt—record type, defines the data field:
  - 00—data record.
  - 01—end of file record.
  - 02—extended segment address record.
  - 03—start segment address record (ignored by Microsemi SoC tools).
  - 04—extended linear address record.
  - 05—start linear address record (ignored by Microsemi SoC tools).
- [dd...]—sequence of n bytes of the data. n is equivalent to what was specified in the ll field.
- cc—checksum of count, address, and data.

Example Intel Hex record:

```
:0300300002337A1E
```

MOTOROLA S-Record

Motorola S-Record is an industry standard file format created by Motorola. The file extension for these files is `.s`.

This format uses ASCII files, hex characters, and records to specify memory content similar to the Intel-Hex format. See the Motorola S-record description document for more information about this format. The RAM Content Manager uses only the S1 through S3 record types. The other record types are ignored.

The major difference between Intel-Hex and Motorola S is the record formats and extra error checking features that are incorporated into the Motorola S-record.

In both formats, memory content is specified by providing a starting address and a data set. The upper bits of the data set are loaded into the starting address and leftovers overflow into the adjacent addresses until the entire data set has been used.

The Motorola S-record is composed of six fields and arranged as follows:

```
Stllaaaa[dd...]cc
```

Where:

- S—start code of every Motorola S-record.
- t—record type, defines the data field.
- ll—byte count of the data field.
- aaaa—16-bit address at the beginning of the memory position for the data. Address is big endian.
- [dd...]—sequence of n bytes of the data; n is equivalent to what was specified in the ll field.
- cc—checksum of count, address, and data.

Example Motorola S-record:

```
S10a0000112233445566778899FFFA
```

7 Appendix: Macro Configuration

7.1 LSRAM Macro

The LSRAM macro (RAM1K20) in the Libero SoC IP macro library can be used directly to instantiate the LSRAM block in the design. The LSRAM block must be configured with appropriate values of the static signals. Instantiating LSRAM primitives in a design is not recommended. For the recommended methods of instantiating memory in a user design, see LSRAM Memory Macro, page 40. Figure 70 shows the LSRAM macro (RAM1K20).

Figure 70 • **RAM1K20 Macro**

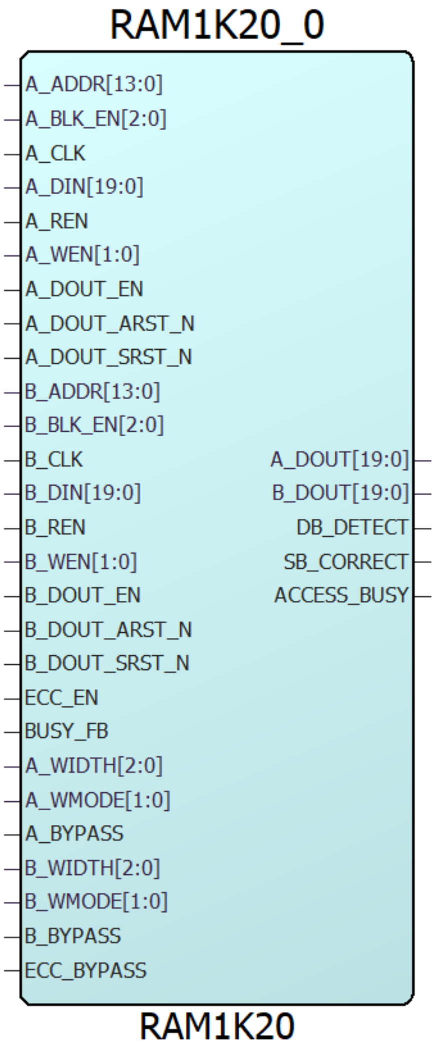


Table 21 lists the ports of RAM1K20.

Table 21 • **Port List of RAM1K20**

Pin Name	Direction	Type[1]	Polarity	Description
A_ADDR[13:0]	Input	Dynamic		Port A address
BLK_EN[2:0]	Input	Dynamic	Active high	Port A block selects
A_CLK	Input	Dynamic	Rising edge	Port A clock
A_DIN[19:0]	Input	Dynamic		Port A write data
A_DOUT[19:0]	Output	Dynamic		Port A read data
A_WEN[1:0]	Input	Dynamic	Active high	Port A write-enables (per byte)
A_REN	Input	Dynamic	Active high	Port A read-enable
A_WIDTH[2:0]	Input	Static		Port A width/depth mode select
A_WMODE[1:0]	Input	Static	Active high	Port A read-before-write and feed-through write selects
A_BYPASS	Input	Static	Active low	Port A pipeline register select
A_DOUT_EN	Input	Dynamic	Active high	Port A pipeline register enable
A_DOUT_SRST_N	Input	Dynamic	Active low	Port A pipeline register synchronous reset
A_DOUT_ARST_N	Input	Dynamic	Active low	Port A pipeline register asynchronous reset
B_ADDR[13:0]	Input	Dynamic		Port B address
B_BLK_EN[2:0]	Input	Dynamic	Active high	Port B block selects
B_CLK	Input	Dynamic	Rising edge	Port B clock
B_DIN[19:0]	Input	Dynamic		Port B write data
B_DOUT[19:0]	Output	Dynamic		Port B read data
B_WEN[1:0]	Input	Dynamic	Active high	Port B write-enables (per byte)
B_REN	Input	Dynamic	Active high	Port B read-enable
B_WIDTH[2:0]	Input	Static	Mode select	Port B width/depth mode select
B_WMODE[1:0]	Input	Static	Active high	Port B read-before-write and feed-through write selects
B_BYPASS	Input	Static	Active low	Port B pipeline register select
B_DOUT_EN	Input	Dynamic	Active high	Port B pipeline register enable
B_DOUT_SRST_N	Input	Dynamic	Active low	Port B pipeline register synchronous-reset
B_DOUT_ARST_N	Input	Dynamic	Active low	Port B pipeline register asynchronous-reset
ECC_EN	Input	Static	Active high	Enable ECC
ECC_BYPASS	Input	Static	Active low	ECC pipeline register select
SB_CORRECT	Output	Dynamic	Active high	Single-bit correct flag
DB_DETECT	Output	Dynamic	Active high	Double-bit detect flag
BUSY_FB	Input	Static	Active high	Lock access to SmartDebug
ACCESS_BUSY	Output	Dynamic	Active high	Busy signal when being initialized or accessed using SmartDebug

[1] Static inputs are defined at design time and need to be tied to 0 or 1

7.1.1 A_WIDTH and B_WIDTH

Table 22 lists the width/depth mode selections for each port. Two-port mode is in effect when the width of at least one port is greater than 20 bits, and A_WIDTH indicates the read width while B_WIDTH indicates the write width.

Table 22 • **Width/Depth Mode Selection**

Depth x Width	A_WIDTH/B_WIDTH
16K x 1	000
8K x 2	001
4K x 4, 4K x 5	010
2K x 8, 2K x 10	011
1K x 16, 1K x 20	100
512 x 32 (two-port) 512 x 40 (two-port) 512 x 33 (two-port ECC)	101

7.1.2 A_WEN and B_WEN

Table 23 lists the write/read control signals for each port. Two-port mode is in effect when the width of at least one port is greater than 20 bits, and read operation is always enabled.

Table 23 • **Write/Read Operation Select**

Depth x Width	A_WEN/B_WEN	Result
16K x 1, 8K x 2, 4K x 5, 2K x 10	x0	Perform a read operation
16K x 1, 8K x 2, 4K x 5, 2K x 10	x1	Perform a write operation
1K x 16	00	Perform a read operation
	01	Write [8:5], [3:0]
	10	Write [18:15], [13:10]
	11	Write [18:15], [13:10], [8:5], [3:0]
1K x 20	00	Perform a read operation
	01	Write [9:0]
	10	Write [19:10]
	11	Write [19:0]
512 x 32 (two-port write)	B_WEN[0] = 1	Write B_DIN[8:5], B_DIN[3:0]
	B_WEN[1] = 1	Write B_DIN[18:15], B_DIN[13:10]
	A_WEN[0] = 1	Write A_DIN[8:5], A_DIN[3:0]
	A_WEN[1] = 1	Write A_DIN[18:15], A_DIN[13:10]
512 x 40 (two-port write)	B_WEN[0] = 1	Write B_DIN[9:0]
	B_WEN[1] = 1	Write B_DIN[19:10]
	A_WEN[0] = 1	Write A_DIN[9:0]
	A_WEN[1] = 1	Write A_DIN[19:10]
512 x 33 (two-port ECC)	B_WEN[1:0] = 11 A_WEN[1:0] = 11	Write B_DIN[16:0] Write A_DIN[15:0]

7.1.3 A_ADDR and B_ADDR

Table 24 lists the address buses for the two ports. 14 bits are required to address the 16K independent locations in x1 mode. In wider modes, fewer address bits are used. The required bits are MSB justified and unused LSB bits must be tied to 0. A_ADDR is synchronized by A_CLK, while B_ADDR is synchronized to B_CLK. Two-port mode is in effect when the width of at least one port is greater than 20, and A_ADDR provides the read-address while B_ADDR provides the write-address.

Table 24 • **Write/Read Operation Select**

	A_ADDR/B_ADDR	
Depth x Width	**Used Bits**	**Unused Bits (Must be Tied to 0)**
16K x 1	[13:0]	None
8K x 2	[13:1]	[0]
4K x 4, 4K x 5	[13:2]	[1:0]
2K x 8, 2K x 10	[13:3]	[2:0]
1K x 16, 1K x 20	[13:4]	[3:0]
512 x 32 (two-port) 512 x 40 (two-port) 512 x 33 (two-port ECC)	[13:5]	[4:0]

7.1.4 A_DIN and B_DIN

Table 25 lists the data input buses for the two ports. The required bits are LSB justified and unused MSB bits must be tied to 0. Two-port mode is in effect when the width of at least one port is greater than 20 bits, and A_DIN provides the MSB of the write-data while B_DIN provides the LSB of the write-data.

Table 25 • **Data Input Buses Used and Unused Bits**

	A_DIN/B_DIN	
Depth x Width	**Used Bits**	**Unused Bits (must be tied to 0)**
16K x 1	[0]	[19:1]
8K x 2	[1:0]	[19:2]
4K x 4	[3:0]	[19:4]
4K x 5	[4:0]	[19:5]
2K x 8	[8:5] => [7:4], [3:0] => [3:0]	[19:9], [4]
2K x 10	[9:0]	[19:10]
1K x 16	[18:15] => [15:12] [13:10] => [11:8] [8:5] => [7:4] [3:0] => [3:0]	[19] [14] [9] [4]
1K x 20	[19:0]	None

Table 25 • **Data Input Buses Used and Unused Bits *(continued)***

	A_DIN/B_DIN	
Depth x Width	**Used Bits**	**Unused Bits (must be tied to 0)**
512 x 32 (two-port write)	A_DIN[18:15] => [31:28] A_DIN[13:10] => [27:24] A_DIN[8:5] => [23:20] A_DIN[3:0] => [19:16] B_DIN[18:15] => [15:12] B_DIN[13:10] =>[11:8] B_DIN[8:5] => [7:4] B_DIN[3:0] => [3:0]	A_DIN[19] A_DIN[14] A_DIN[9] A_DIN[4] B_DIN[19] B_DIN[14] B_DIN[9] B_DIN[4]
512 x 40 (two-port write)	A_DIN[19:0] => [39:20] B_DIN[19:0] => [19:0]	None
512 x 33 (two-port ECC)	A_DIN[15:0] => [32:17] B_DIN[16:0] => [16:0]	A_DIN[19:16] B_DIN[19:17]

7.1.5 A_DOUT and B_DOUT

Table 26 lists the data output buses for the two ports. The required bits are LSB-justified. Two-port mode is in effect when the width of at least one port is greater than 20, and A_DOUT provides the MSB of the read-data while B_DOUT provides the LSB of the read-data.

Table 26 • **Data Output Buses Used and Unused Bits**

	A_ADDR/B_ADDR	
Depth x Width	**Used Bits**	**Unused Bits (must be tied to 0)**
16K x 1	[0]	[19:1]
8K x 2	[1:0]	[19:2]
4K x 4	[3:0]	[19:4]
4K x 5	[4:0]	[19:5]
2K x 8	[8:5] => [7:4] [3:0] => [3:0]	[19:9] [4]
2K x 10	[9:0]	[19:10]
1K x 16	[18:15] => [15:12] [13:10] => [11:8] [8:5] => [7:4] [3:0] => [3:0]	[19] [14] [9] [4]
1K x 20	[19:0]	None
512 x 32 (two-port write)	A_DIN[18:15] => [31:28] A_DIN[13:10] => [27:24] A_DIN[8:5] => [23:20] A_DIN[3:0] => [19:16] B_DIN[18:15] => [15:12] B_DIN[13:10] => [11:8] B_DIN[8:5] => [7:4] B_DIN[3:0] => [3:0]	A_DIN[19] A_DIN[14] A_DIN[9] A_DIN[4] B_DIN[19] B_DIN[14] B_DIN[9] B_DIN[4]
512 x 40 (two-port write)	A_DOUT[19:0] => [39:20] B_DOUT[19:0] => [19:0]	None

Table 26 • **Data Output Buses Used and Unused Bits *(continued)***

	A_ADDR/B_ADDR	
Depth x Width	**Used Bits**	**Unused Bits (must be tied to 0)**
512 x 33 (two-port ECC)	A_DOUT[15:0] => [32:17] B_DOUT[16:0] => [16:0]	A_DOUT[19:16] B_DOUT[19:17]

7.1.6 A_BLK_EN and B_BLK_EN

Table 27 lists the block-port select control signals for the two ports. A_BLK_EN is synchronized to A_CLK, while B_BLK_EN is synchronized to B_CLK. When two-port mode is in effect, the width of at least one port is greater than 20 bits, A_BLK_EN controls the read operation, and B_BLK_EN controls the write operation.

Table 27 • **Block-Port Select**

Block-Port Select Signal	Value	Result
A_BLK_EN[2:0]	111	Perform read or write operation on Port A. If the width is greater than 20 bits, a read is performed from both ports A and B.
A_BLK_EN[2:0]	Any one bit is 0	No operation in memory from Port A. Port A read-data will be forced to 0. If the width is greater than 20 bits, the read-data from both ports A and B will be forced to 0.
B_BLK_EN[2:0]	111	Perform read or write operation on Port B, unless the width is greater than 20 bits and a write is performed to both ports A and B.
B_BLK_EN[2:0]	Any one bit is 0	No operation in memory from Port B. Port B read-data will be forced to 0, unless the width is greater than 20 bits and write operation to both ports A and B is gated.

7.1.7 A_WMODE and B_WMODE

In dual-port write mode, each port has a feed-through write or read-before-write option. When A_WMODE/B_WMODE is equal to:

- Logic 00 = Simple Write. Read-data port holds the previous value.
- Logic 01 = Feed-through. Write-data appears on the corresponding read-data port. This setting is invalid when the width of at least one port is greater than 20 bits and the two-port mode is in effect.
- Logic 10 = Read-before-write. The previous content of the memory appears on the corresponding read-data port before it is overwritten. This setting is invalid when the width of at least one port is greater than 20 bits and the two-port mode is in effect.

7.1.8 A_CLK and B_CLK

All signals in ports A and B are synchronous to the corresponding port clock. All addresses, data, block-port select, write-enable, and read-enable inputs must be set up before the rising edge of the clock. The read or write operation begins with the rising edge. Two-port mode is in effect when the width of at least one port is greater than 20 bits, and A_CLK provides the read clock while B_CLK provides the write clock.

7.1.9 A_REN and B_REN

Enables read operation from the memory on the corresponding port. Two-port read mode is in effect when the width of Port A is greater than 20 bits, and A_REN controls the read operation.

7.1.9.1 Read-Data Pipeline Register Control Signals

- A_BYPASS and B_BYPASS
- A_DOUT_EN and B_DOUT_EN

- A_DOUT_SRST_N and B_DOUT_SRST_N
- A_DOUT_ARST_N and B_DOUT_ARST_N

Two-port mode is in effect when the width of at least one port is greater than 20 bits, and the A_DOUT register signals control the MSB of the read-data, while the B_DOUT register signals control the LSB of the read-data.

Table 28 lists the functionality of the control signals on the A_DOUT and B_DOUT pipeline registers.

Table 28 • **Truth Table for A_DOUT and B_DOUT Registers**

ARST_N	A_BYPASS/ B_BYPASS	A_CLK/B_CLK	A_EN/B_EN	A_SRST_N/ B_SRST_N	D	Q_{n+1}
0	X	X	X	X	X	0
1	0	Not rising	X	X	X	Q_n
1	0	↑	0	X	X	Q_n
1	0	↑	1	0	X	0
1	0	↑	1	1	D	D
1	1	X	X	X	D	D

7.1.10 ECC_EN and ECC_BYPASS

The ECC operation is only allowed in two-port mode when the width of both ports is greater than 20 bits.

- ECC_EN = 0—disable ECC.
- ECC_EN = 1, ECC_BYPASS= 0—enable ECC Pipelined.
  - ECC Pipelined mode inserts an additional clock cycle to read-data. In addition, write-feed-through, and read-before-write modes add another clock cycle to read-data.
- ECC_EN = 1, ECC_BYPASS= 1—enable ECC Non-pipelined.

7.1.11 SB_CORRECT and DB_DETECT

ECC flags become available when the ECC operation is enabled in two-port mode and the width of both ports is greater than 20. Table 29 lists the functionality of the error detection and correction flags.

Table 29 • **Error Detection and Correction Flags**

DB_DETECT	SB_CORRECT	Flag
0	0	No errors detected
0	1	A single bit error is detected and corrected in the data output.
1	1	Multiple bit errors are detected, but are not corrected.

7.1.12 BUSY_FB

When the control signal is set to 1, the entire RAM1K20 memory is locked off from being accessed by the SmartDebug.

7.1.13 ACCESS_BUSY

This output indicates that the RAM1K20 memory is being accessed by SmartDebug.

7.2 µSRAM Macro

The µSRAM macro (RAM64x12) in Libero SoC can be used directly to instantiate µSRAM in the design. µSRAM must be configured correctly with appropriate values provided to the static signals before instantiating in the design. Instantiating µSRAM primitives in a design is not recommended. For the recommended methods of instantiating memory into a user design, see µSRAM Memory Macro, page 49. Figure 71 shows the µSRAM macro (RAM64x12) available in the Libero SoC macro library.

Figure 71 • **RAM64x12 Macro**

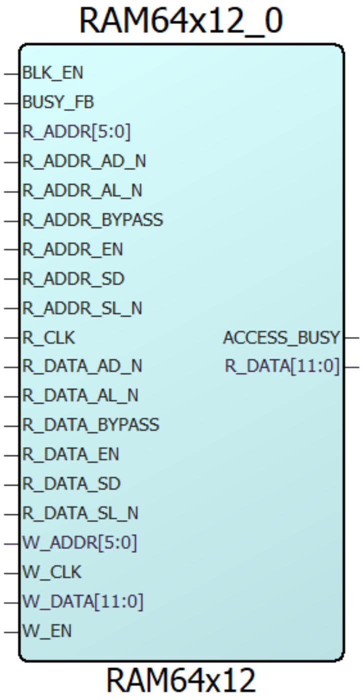


Table 30 lists the ports of RAM64x12.

Table 30 • **Port List for RAM64x12**

Pin Name	Direction	Type[1]	Polarity	Description
W_EN	Input	Dynamic	Active high	Write port enable
W_CLK	Input	Dynamic	Rising edge	Write clock. All write-address, write-data, and write-enable inputs must be set up before the rising edge of the clock. The write operation begins with the rising edge.
W_ADDR[5:0]	Input	Dynamic		Write address
W_DATA[11:0]	Input	Dynamic		Write-data
BLK_EN	Input	Dynamic	Active high	Read port block select. When High, a read operation is performed. When Low, read-data will be forced to zero. BLK_EN signal is registered through R_CLK when R_ADDR_BYPASS is Low.
R_CLK	Input	Dynamic	Rising edge	Read registers clock. All read-address, block-port select, and read-enable inputs must be set up before the rising edge of the clock. The read operation begins with the rising edge.
R_ADDR[]	Input	Dynamic		Read-address

Table 30 • **Port List for RAM64x12 *(continued)***

Pin Name	Direction	Type[1]	Polarity	Description
R_ADDR_BYPASS	Input	Static	Active low	Read-address and BLK_EN register select
R_ADDR_EN	Input	Dynamic	Active high	Read-address register enable
R_ADDR_SL_N	Input	Dynamic	Active low	Read-address register synchronous load
R_ADDR_SD	Input	Static	Active high	Read-address register synchronous load data
R_ADDR_AL_N	Input	Dynamic	Active low	Read-address register asynchronous load
R_ADDR_AD_N	Input	Static	Active low	Read-address register asynchronous load data
R_DATA[]	Output	Dynamic		Read-data
R_DATA_BYPASS	Input	Static	Active low	Read-data pipeline register select
R_DATA_EN	Input	Dynamic	Active high	Read-data pipeline register enable
R_DATA_SL_N	Input	Dynamic	Active low	Read-data pipeline register synchronous load
R_DATA_SD	Input	Static	Active high	Read-data pipeline register synchronous load data
R_DATA_AL_N	Input	Dynamic	Active low	Read-data pipeline register asynchronous load
R_DATA_AD_N	Input	Static	Active low	Read-data pipeline register asynchronous load data
BUSY_FB	Input	Static	Active high	Lock access to SmartDebug
ACCESS_BUSY	Output	Dynamic	Active high	Busy signal from SmartDebug

1. Static inputs are defined at design time and need to be tied to 0 or 1.

7.2.1 Read-Address and Read-Data Pipeline Register Control Signals

Table 31 lists the functionality of the control signals on the R_ADDR and R_DATA registers.

Table 31 • **Truth Table for A_DOUT and B_DOUT Registers**

A_AL_N/ B_AL_N	A_AD_N/ B_AD_N	A_BYPASS/ B_BYPASS	A_CLK/ B_CLK	A_EN/ B_EN	A_SL_N/ B_SL_N	A_SD/ B_SD	D	Q_{n+1}
0	ADn	X	X	X	X	X	X	!ADn
1	X	0	Not rising	X	X	X	X	Q_n
1	X	0	↑	0	X	X	X	Q_n
1	X	0	↑	1	0	SD	X	SD
1	X	0	↑	1	1	X	D	D
1	X	1	X	X	X	X	D	D

7.3 Math Block Macro

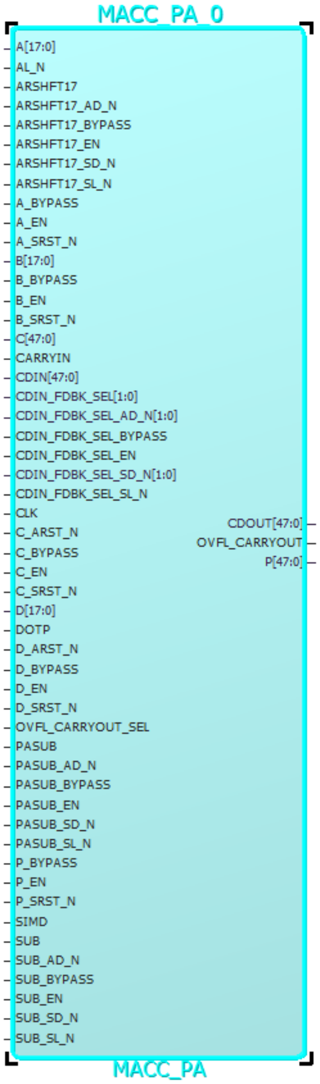
Two math block macros, MACC_PA and MACC_PA_BC_ROM, are available in Libero SoC IP catalog macro library. These macros can be used in designs created with SmartDesign or by directly instantiating the macro wrapper in an HDL file as a component. Instantiating math block primitives in a design is not recommended. For the recommended methods of instantiating math blocks into a user design, see Implementation, page 73. When using the macros, the inputs and outputs must be connected manually to the design signals. Proper values for the static signals must also be provided to ensure that the math block is configured in the correct operational mode. For example, to configure the math block in DOTP mode, the DOTP signal must be tied to logic 1.

7.3.1 MACC_PA (MACC with Pre-Adder)

The MACC_PA is the multiply and accumulator with pre-adder macro block. The MACC_PA macro implements multiplication, multiply-add, and multiply-accumulate functions. The MACC_PA block can accumulate the current multiplication product with a previous result, a constant, a dynamic value, or a result from another MACC_PA block. Each MACC_PA block can also be configured to perform a DOTP operation. All the signals of the MACC_PA block have optional registers.

Figure 72 shows the MACC_PA available in the macro library.

Figure 72 • **MACC_PA Macro**



7.3.1.1 Port List

Table 32 • **MACC_PA Pin Descriptions**

Port Name	Direction	Type[1]	Polarity	Description
DOTP	Input	Static	Active high	DOTP mode When DOTP = 1, MACC_PA block performs DOTP of two pairs of 9-bit operands. • SIMD must not be 1 • C[8:0] must be connected to CARRYIN.
SIMD	Input	Static	Active high	SIMD mode When SIMD = 1, MACC_PA block performs dual-independent multiplication of two pairs of 9-bit operands. DOTP must not be 1 ARSHFT17 must be 0 D[8:0] must be 0 C[17:0] must be 0 E[17:0] must be 0. For more information about how operand E is obtained from P, CDIN, or 0, see Table 19.
OVFL_CARRYOUT_SEL	Input	Static	Active high	Generate OVERFLOW or CARRYOUT with result P. OVERFLOW when OVFL_CARRYOUT_SEL = 0 CARRYOUT when OVFL_CARRYOUT_SEL = 1
CLK	Input	Dynamic	Rising edge	Clock for A, B, C, CARRYIN, D, P, OVFL_CARRYOUT, ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers.
AL_N	Input	Dynamic	Active low	Asynchronous load for A, B, P, OVFL_CARRYOUT, ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers. Connect to 1 if not registered. When asserted, A, B, P, and OVFL_CARRYOUT registers are loaded with zero, while the ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers are loaded with the complementary value of the respective _AD_N.
A[17:0]	Input	Dynamic	Active high	Input data A
A_BYPASS	Input	Static	Active high	Bypass data A registers. Connect to 1 if not registered. For more information, see Table 34.
A_SRST_N	Input	Dynamic	Active low	Synchronous reset for data A registers. Connect to 1 if not registered. For more information, see Table 34.
A_EN	Input	Dynamic	Active high	Enable for data A registers. Connect to 1 if not registered. For more information, see Table 34.
B[17:0]	Input	Dynamic	Active high	Input data B to pre-adder with data D.
B_BYPASS	Input	Static	Active high	Bypass data B registers. Connect to 1 if not registered. For more information, see Table 34.
B_SRST_N	Input	Dynamic	Active low	Synchronous reset for data B registers. Connect to 1 if not registered. For more information, see Table 34.
B_EN	Input	Dynamic	Active high	Enable for data B registers. Connect to 1 if not registered. For more information, see Table 34.

Table 32 • MACC_PA Pin Descriptions *(continued)*

Port Name	Direction	Type[1]	Polarity	Description
D[17:0]	Input	Dynamic	Active high	Input data D to pre-adder with data B. When SIMD = 1, connect D[8:0] to 0.
D_BYPASS	Input	Static	Active high	Bypass data D registers. Connect to 1 if not registered. For more information, see Table 35.
D_ARST_N	Input	Dynamic	Active low	Asynchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 35.
D_SRST_N	Input	Dynamic	Active low	Synchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 35.
D_EN	Input	Dynamic	Active high	Enable for data D registers. Connect to 1 if not registered. For more information, see Table 35.
CARRYIN	Input	Dynamic	Active high	CARRYIN for input data C.
C[47:0]	Input	Dynamic	Active high	Input data C. When DOTP = 1, connect C[8:0] to CARRYIN. When SIMD = 1, connect C[8:0] to 0.
C_BYPASS	Input	Static	Active high	Bypass CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
C_ARST_N	Input	Dynamic	Active low	Asynchronous reset for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
C_SRST_N	Input	Dynamic	Active low	Synchronous reset for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
C_EN	Input	Dynamic	Active high	Enable for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
CDIN[47:0]	Input	Cascade	Active high	Cascaded input for operand E. The entire bus must be driven by an entire CDOUT of another MACC_PA or MACC_PA_BC_ROM block. In DOTP mode, the driving CDOUT must also be generated by a MACC_PA or MACC_PA_BC_ROM block in DOTP mode. For more information about how CDIN is propagated to operand E, see Table 19.
P[47:0]	Output		Active high	Result data. For more information, see Table 20.
OVFL_CARRYOUT	Output		Active high	OVERFLOW or CARRYOUT. For more information, see Table 20.
P_BYPASS	Input	Static	Active high	Bypass P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 34.
P_SRST_N	Input	Dynamic	Active low	Synchronous reset for P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 34.
P_EN	Input	Dynamic	Active high	Enable for P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 34.

Table 32 • **MACC_PA Pin Descriptions *(continued)***

Port Name	Direction	Type[1]	Polarity	Description
CDOUT[47:0]	Output	Cascade	Active high	Cascade output of result P. For more information, see Table 20. Value of CDOUT is the same as P. The entire bus must either be dangling or drive an entire CDIN of another MACC_PA or MACC_PA_BC_ROM block in cascaded mode.
PASUB	Input	Dynamic	Active high	Subtract operation for pre-adder of B and D.
PASUB_BYPASS	Input	Static	Active high	Bypass PASUB register. Connect to 1 if not registered. For more information, see Table 33.
PASUB_AD_N	Input	Static	Active low	Asynchronous load data for PASUB register. For more information, see Table 33.
PASUB_SL_N	Input	Dynamic	Active low	Synchronous load for PASUB register. Connect to 1 if not registered. For more information, see Table 33.
PASUB_SD_N	Input	Static	Active low	Synchronous load data for PASUB register. For more information, see Table 33.
PASUB_EN	Input	Dynamic	Active high	Enable for PASUB register. Connect to 1 if not registered. For more information, see Table 33.
CDIN_FDBK_SEL[1:0]	Input	Dynamic	Active high	Select CDIN, P or 0 for operand E. For more information, see Table 19.
CDIN_FDBK_SEL_BYPASS	Input	Static	Active high	Bypass CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 33.
CDIN_FDBK_SEL_AD_N [1:0]	Input	Static	Active low	Asynchronous load data for CDIN_FDBK_SEL register. For more information, see Table 33.
CDIN_FDBK_SEL_SL_N	Input	Dynamic	Active low	Synchronous load for CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 33.
CDIN_FDBK_SEL_SD_N [1:0]	Input	Static	Active low	Synchronous load data for CDIN_FDBK_SEL register. For more information, see Table 33.
CDIN_FDBK_SEL_EN	Input	Dynamic	Active high	Enable for CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 33.
ARSHFT17	Input	Dynamic	Active high	Arithmetic right-shift for operand E. When asserted, a 17-bit arithmetic right-shift is performed on operand E. For information on how operand E is obtained from P, CDIN or 0, see Table 19. When SIMD = 1, ARSHFT17 must be 0.
ARSHFT17_BYPASS	Input	Static	Active high	Bypass ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 33.
ARSHFT17_AD_N	Input	Static	Active low	Asynchronous load data for ARSHFT17 register. For more information, see Table 33.
ARSHFT17_SL_N	Input	Dynamic	Active low	Synchronous load for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 33.
ARSHFT17_SD_N	Input	Static	Active low	Synchronous load data for ARSHFT17 register. For more information, see Table 33.

Table 32 • **MACC_PA Pin Descriptions *(continued)***

Port Name	Direction	Type[1]	Polarity	Description
ARSHFT17_EN	Input	Dynamic	Active high	Enable for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 33.
SUB	Input	Dynamic	Active high	Subtract operation.
SUB_BYPASS	Input	Static	Active high	Bypass SUB register. Connect to 1 if not registered. For more information, see Table 33.
SUB_AD_N	Input	Static	Active low	Asynchronous load data for SUB register. For more information, see Table 33.
SUB_SL_N	Input	Dynamic	Active low	Synchronous load for SUB register. Connect to 1 if not registered. For more information, see Table 33.
SUB_SD_N	Input	Static	Active low	Synchronous load data for SUB register. For more information, see Table 33.
SUB_EN	Input	Dynamic	Active high	Enable for SUB register. Connect to 1 if not registered. For more information, see Table 33.

[1] Static inputs are defined at design time and need to be tied to 0 or 1.

Note: SUM[49:0] is defined similarly to P[47:0] as listed in Table 20, except that SUM is a 50-bit quantity so that overflow does not occur. SUM[48] is the carry out bit of a 48-bit final adder that produces P[47:0].

Table 33 • **Truth Table for Control Registers ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB**

AL_N	_AD_N	_BYPASS	CLK	_EN	_SL_N	_SD_N	D	Q_{n+1}
0	AD_N	0	X	X	X	X	X	!AD_N
1	X	0	Not rising	X	X	X	X	Q_n
1	X	0	↑	0	X	X	X	Q_n
1	X	0	↑	1	0	SD_N	X	!SD_N
1	X	0	↑	1	1	X	D	D
X	X	1	X	0	X	X	X	Q_n
X	X	1	X	1	0	SD_N	X	!SD_N
X	X	1	X	1	1	X	D	D

Table 34 • **Truth Table for Data Registers A, B, P, and OVFL_CARRYOUT**

AL_N	_BYPASS	CLK	_EN	_SRST_N	D	Q_{n+1}
0	0	X	X	X	X	0
1	0	Not rising	X	X	X	Q_n
1	0	↑	0	X	X	Q_n
1	0	↑	1	0	X	0
1	0	↑	1	1	D	D
X	1	X	0	X	X	Q_n
X	1	X	1	0	X	0
X	1	X	1	1	D	D

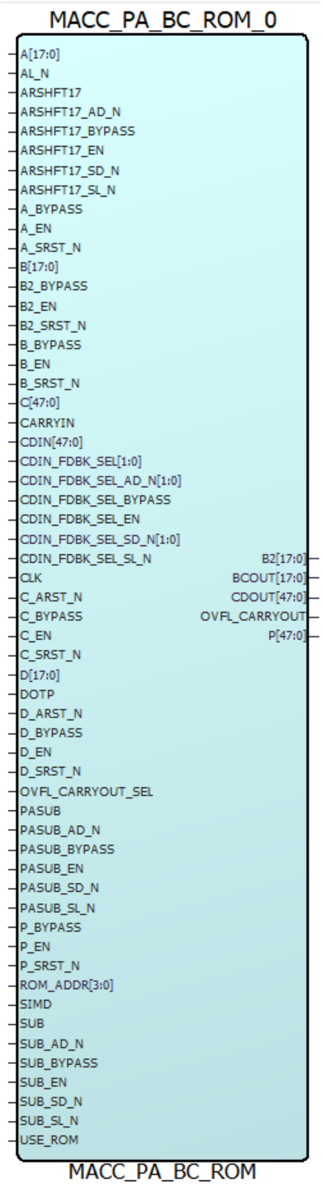
Table 35 • **Truth Table for Data Registers C, CARRYIN, and D**

_ARST_N	_BYPASS	CLK	_EN	_SRST_N	D	Q_{n+1}
0	0	X	X	X	X	0
1	0	Not rising	X	X	X	Q_n
1	0	↑	0	X	X	Q_n
1	0	↑	1	0	X	0
1	0	↑	1	1	D	D
X	1	X	0	X	X	Q_n
X	1	X	1	0	X	0
X	1	X	1	1	D	D

7.3.2 MACC_PA_BC_ROM (MACC with Pre-Adder, BCOUT Register, and Coefficient ROM)

The MACC_PA_ROM is the multiply accumulator with pre-adder, B register cascading, and built-in ROM macro block. The MACC_PA_BC_ROM macro extends the functionality of the MACC_PA macro to provide a 16 x 18 ROM at the A input along with a pipelined output of B for cascading.

Figure 73 • MACC_PA_BC_ROM Macro



7.3.2.1 Parameters

Coefficients are loaded using INIT parameter. It holds the 16 x 18 ROM content as a linear array. The first 18 bits are word 0, the next 18 bits are word 1, and so on. Table 36 lists the INIT declaration for loading coefficients.

Table 36 • **MACC_PA_BC_ROM Parameter Descriptions**

Parameter	Dimensions	Description

INIT	parameter [287:0] INIT = {
	18'h0, 18'h0, 18'h0, 18'h0, 18'h0, 18'h0, 18'h0, 18'h0,
	18'h0, 18'h0, 18'h0, 18'h0, 18'h0, 18'h0, 18'h0, 18'h0
	};
		16 x 18 ROM content specified in Verilog.

INIT	generic map(INIT => (B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"))	16 x 18 ROM content specified in VHDL.

7.3.2.2 Port List

Table 37 • **MACC_PA_BC_ROM Pin Descriptions**

Port Name	Direction	Type[1]	Polarity	Description
DOTP	Input	Static	Active high	DOTP mode. When DOTP = 1, MACC_PA_BC_ROM block performs DOTP of two pairs of 9-bit operands. SIMD must not be 1. C[8:0] must be connected to CARRYIN.
SIMD	Input	Static	Active high	SIMD mode When SIMD = 1, MACC_PA_BC_ROM block performs dual independent multiplication of two pairs of 9-bit operands. DOTP must not be 1 ARSHFT17 must be 0 D[8:0] must be 0 C[17:0] must be 0 E[17:0] must be 0. For information on how operand E is obtained from P, CDIN or 0, see Table 19.
OVFL_CARRYOUT_SEL	Input	Static	Active high	Generate OVERFLOW or CARRYOUT with result P. OVERFLOW when OVFL_CARRYOUT_SEL = 0 CARRYOUT when OVFL_CARRYOUT_SEL = 1
CLK	Input	Dynamic	Rising edge	Clock for A, B, C, CARRYIN, D, P, OVFL_CARRYOUT, ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers.

Table 37 • **MACC_PA_BC_ROM Pin Descriptions *(continued)***

Port Name	Direction	Type[1]	Polarity	Description
AL_N	Input	Dynamic	Active low	Asynchronous load for A, B, P, OVFL_CARRYOUT, ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers. Connect to 1, if none are registered. When asserted, A, B, P, and OVFL_CARRYOUT registers are loaded with zero, while the ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB registers are loaded with the complementary value of the respective _AD_N.
USE_ROM	Input	Static (virtual)	Active high	Selection for operand A. When USE_ROM = 0, select input data A. When USE_ROM = 1, select ROM data at ROM_ADDR.
ROM_ADDR[3:0]	Input	Dynamic	Active high	Address of ROM data for operand A when USE_ROM = 1
A[17:0]	Input	Dynamic	Active high	Input data for operand A when USE_ROM = 0
A_BYPASS	Input	Static	Active high	Bypass data A registers. Connect to 1 if not registered. For more information, see Table 34.
A_SRST_N	Input	Dynamic	Active low	Synchronous reset for data A registers. Connect to 1 if not registered. For more information, see Table 34.
A_EN	Input	Dynamic	Active high	Enable for data A registers. Connect to 1 if not registered. For more information, see Table 34.
B[17:0]	Input	Dynamic	Active high	Input data B to pre-adder with data D
B_BYPASS	Input	Static	Active high	Bypass data B registers. Connect to 1 if not registered. For more information, see Table 34.
B_SRST_N	Input	Dynamic	Active low	Synchronous reset for data B registers. Connect to 1 if not registered. For more information, see Table 34.
B_EN	Input	Dynamic	Active high	Enable for data B registers. Connect to 1 if not registered. For more information, see Table 34.
B2[17:0]	Output	Dynamic	Active high	Pipelined output of input data B. Result P must be floating when B2 is used.
B2_BYPASS	Input	Static	Active high	Bypass data B2 registers. Connect to 1 if not registered. For more information, see Table 34.
B2_SRST_N	Input	Dynamic	Active low	Synchronous reset for data B2 registers. Connect to 1 if not registered. For more information, see Table 34.
B2_EN	Input	Dynamic	Active high	Enable for data B2 registers. Connect to 1 if not registered. For more information, see Table 34.

Table 37 • **MACC_PA_BC_ROM Pin Descriptions *(continued)***

Port Name	Direction	Type[1]	Polarity	Description
BCOUT[17:0]	Output	Cascade	Active high	Cascade output of B2. Value of BCOUT is the same as B2. The entire bus must either be dangling or drive an entire B input of another MACC_PA or MACC_PA_BC_ROM block.
D[17:0]	Input	Dynamic	Active high	Input data D to pre-adder with data B. When SIMD = 1, connect D[8:0] to 0.
D_BYPASS	Input	Static	Active high	Bypass data D registers. Connect to 1 if not registered. For more information, see Table 35.
D_ARST_N	Input	Dynamic	Active low	Asynchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 35.
D_SRST_N	Input	Dynamic	Active low	Synchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 35.
D_EN	Input	Dynamic	Active high	Enable for data D registers. Connect to 1 if not registered. For more information, see Table 35.
CARRYIN	Input	Dynamic	Active high	CARRYIN for input data C
C[47:0]	Input	Dynamic	Active high	Input data C. When DOTP = 1, connect C[8:0] to CARRYIN. When SIMD = 1, connect C[8:0] to 0.
C_BYPASS	Input	Static	Active high	Bypass CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
C_ARST_N	Input	Dynamic	Active low	Asynchronous reset for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
C_SRST_N	Input	Dynamic	Active low	Synchronous reset for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
C_EN	Input	Dynamic	Active high	Enable for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 35.
CDIN[47:0]	Input	Cascade	Active high	Cascaded input for operand E. The entire bus must be driven by an entire CDOUT of another MACC_PA or MAC_PA_BC_ROM block. In Dot-product mode, the driving CDOUT must also be generated by a MACC_PA or MAC_PA_BC_ROM block in Dot-product mode. For more information about how CDIN is propagated to operand E, see Table 19.
P[47:0]	Output		Active high	Result data. For more information, see Table 19. B2 output must be floating when P is used.
OVFL_CARRYOUT	Output		Active high	OVERFLOW or CARRYOUT. For more information, see Table 18.
P_BYPASS	Input	Static	Active high	Bypass P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 34.

Table 37 • **MACC_PA_BC_ROM Pin Descriptions *(continued)***

Port Name	Direction	Type[1]	Polarity	Description
P_SRST_N	Input	Dynamic	Active low	Synchronous reset for P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 34.
P_EN	Input	Dynamic	Active high	Enable for P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 34.
CDOUT[47:0]	Output	Cascade	Active high	Cascade output of result P. For more information, see Table 20. Value of CDOUT is the same as P. The entire bus must either be dangling or drive an entire CDIN of another MACC_PA or MAC_PA_BC_ROM block in cascaded mode.
PASUB	Input	Dynamic	Active high	Subtract operation for pre-adder of B and D
PASUB_BYPASS	Input	Static	Active high	Bypass PASUB register. Connect to 1 if not registered. For more information, see Table 33.
PASUB_AD_N	Input	Static	Active low	Asynchronous load data for PASUB register. For more information, see Table 33.
PASUB_SL_N	Input	Dynamic	Active low	Synchronous load for PASUB register. Connect to 1 if not registered. For more information, see Table 33.
PASUB_SD_N	Input	Static	Active low	Synchronous load data for PASUB register. For more information, see Table 33.
PASUB_EN	Input	Dynamic	Active high	Enable for PASUB register. Connect to 1 if not registered. For more information, see Table 33.
CDIN_FDBK_SEL[1:0]	Input	Dynamic	Active high	Select CDIN, P or 0 for operand E. For more information, see Table 19.
CDIN_FDBK_SEL_BYPASS	Input	Static	Active high	Bypass CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 33.
CDIN_FDBK_SEL_AD_N [1:0]	Input	Static	Active low	Asynchronous load data for CDIN_FDBK_SEL register. For more information, see Table 33.
CDIN_FDBK_SEL_SL_N	Input	Dynamic	Active low	Synchronous load for CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 33.
CDIN_FDBK_SEL_SD_N [1:0]	Input	Static	Active low	Synchronous load data for CDIN_FDBK_SEL register. For more information, see Table 33.
CDIN_FDBK_SEL_EN	Input	Dynamic	Active high	Enable for CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 33.
ARSHFT17	Input	Dynamic	Active high	Arithmetic right-shift for operand E. When asserted, a 17-bit arithmetic right-shift is performed on operand E. For more information about how operand E is obtained from P, CDIN or 0, see Table 19. When SIMD = 1, ARSHFT17 must be 0.
ARSHFT17_BYPASS	Input	Static	Active high	Bypass ARSHFT17 register. Connect to 1, if not registered. For more information, see Table 33.
ARSHFT17_AD_N	Input	Static	Active low	Asynchronous load data for ARSHFT17 register. For more information, see Table 33.

Table 37 • **MACC_PA_BC_ROM Pin Descriptions *(continued)***

Port Name	Direction	Type[1]	Polarity	Description
ARSHFT17_SL_N	Input	Dynamic	Active low	Synchronous load for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 33.
ARSHFT17_SD_N	Input	Static	Active low	Synchronous load data for ARSHFT17 register. For more information, see Table 33.
ARSHFT17_EN	Input	Dynamic	Active high	Enable for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 33.
SUB	Input	Dynamic	Active high	Subtract operation~~.~~
SUB_BYPASS	Input	Static	Active high	Bypass SUB register. Connect to 1 if not registered. For more information, see Table 33.
SUB_AD_N	Input	Static	Active low	Asynchronous load data for SUB register. For more information, see Table 33.
SUB_SL_N	Input	Dynamic	Active low	Synchronous load for SUB register. Connect to 1 if not registered. For more information, see Table 33.
SUB_SD_N	Input	Static	Active low	Synchronous load data for SUB register. For more information, see Table 33.
SUB_EN	Input	Dynamic	Active high	Enable for SUB register. Connect to 1 if not registered. For more information, see Table 33.

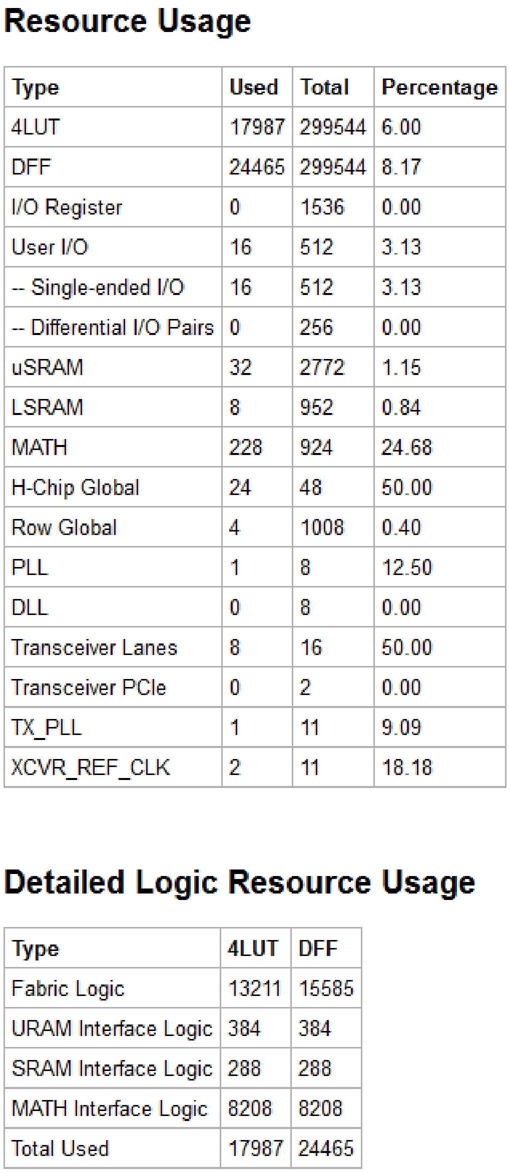
1. Static inputs are defined at design time and need to be tied to 0 or 1.

7.4 Libero SoC Compile Report

Libero SoC Design Suite offers high productivity with its comprehensive, easy-to-learn, easy-to-adopt development tools for designing with the PolarFire family. The compile report contains fabric resource utilization and the total number of resources available. This report provides the number of 4LUTs, DFFs, µSRAMs, LSRAMs, math blocks, I/O, DLLs, PLLs, transceivers, and globals used in a design. It also contains the additional 4LUTs and DFFs required for RAMs and MACC interface logic.

For a sample compile report on MPF300, see Figure 74.

Figure 74 • **Sample Compile Report for MPF300**



Resource Usage

Type	Used	Total	Percentage
4LUT	17987	299544	6.00
DFF	24465	299544	8.17
I/O Register	0	1536	0.00
User I/O	16	512	3.13
-- Single-ended I/O	16	512	3.13
-- Differential I/O Pairs	0	256	0.00
uSRAM	32	2772	1.15
LSRAM	8	952	0.84
MATH	228	924	24.68
H-Chip Global	24	48	50.00
Row Global	4	1008	0.40
PLL	1	8	12.50
DLL	0	8	0.00
Transceiver Lanes	8	16	50.00
Transceiver PCIe	0	2	0.00
TX_PLL	1	11	9.09
XCVR_REF_CLK	2	11	18.18

Detailed Logic Resource Usage

Type	4LUT	DFF
Fabric Logic	13211	15585
URAM Interface Logic	384	384
SRAM Interface Logic	288	288
MATH Interface Logic	8208	8208
Total Used	17987	24465