

Introduction (Ask a Question)

This user guide describes the fabric architecture and its components available in the PolarFire® family. The FPGA fabric is common to the PolarFire family, which consists of the following FPGA devices.

- PolarFire FPGAs** Microchip's PolarFire FPGAs are the fifth-generation family of non-volatile FPGA devices, built on state-of-the-art 28 nm non-volatile process technology. PolarFire FPGAs deliver the lowest power at mid-range densities. PolarFire FPGAs lower the cost of mid-range FPGAs by integrating the industry's lowest power FPGA fabric, lowest power 12.7 Gbps transceiver lane, built-in low power dual PCI Express Gen2 (EP/RP), and, on select data security (S) devices, an integrated low-power crypto co-processor.
- PolarFire SoC FPGAs** Microchip's PolarFire SoC FPGAs are the fifth-generation family of non-volatile SoC FPGA devices, built on state-of-the-art 28 nm non-volatile process technology. The PolarFire SoC family offers industry's first RISC-V® based SoC FPGAs capable of running Linux®. It combines a powerful 64-bit 5x core RISC-V Microprocessor Subsystem (MSS), based on SiFive's U54-MC family, with the PolarFire FPGA fabric in a single device.
- RT PolarFire FPGAs** Microchip's RT PolarFire FPGAs combine our 60 years of space flight heritage with the industry's lowest-power PolarFire FPGA family to enable new capabilities for space and mission-critical applications. RT PolarFire FPGA family includes RTPF500T, RTPF500TL, RTPF500TS, RTPF500TLS, RTPF500ZT, RTPF500ZTL, RTPF500ZTS, and RTPF500ZTLS devices.
- RT PolarFire SoC FPGAs** Microchip's RT PolarFire SoC FPGAs are designed to enable high-performance data processing. Our radiation-tolerant PolarFire SoC FPGA is the industry's first embedded, real-time, Linux®-capable, RISC-V®-based Microprocessor Subsystem (MSS) on the flight-proven RT PolarFire FPGA fabric. With our extensive Mi-V ecosystem, designers can develop lower-power solutions for the challenging thermal environments seen in space. RT PolarFire SoC FPGA family includes RTPFS160ZT, RTPFS160ZTL, RTPFS160ZTS, RTPFS160ZTLS, RTPFS460ZT, RTPFS460ZTL, RTPFS460ZTS, and RTPFS460ZTLS devices.

The following table summarizes fabric components available in the PolarFire family.

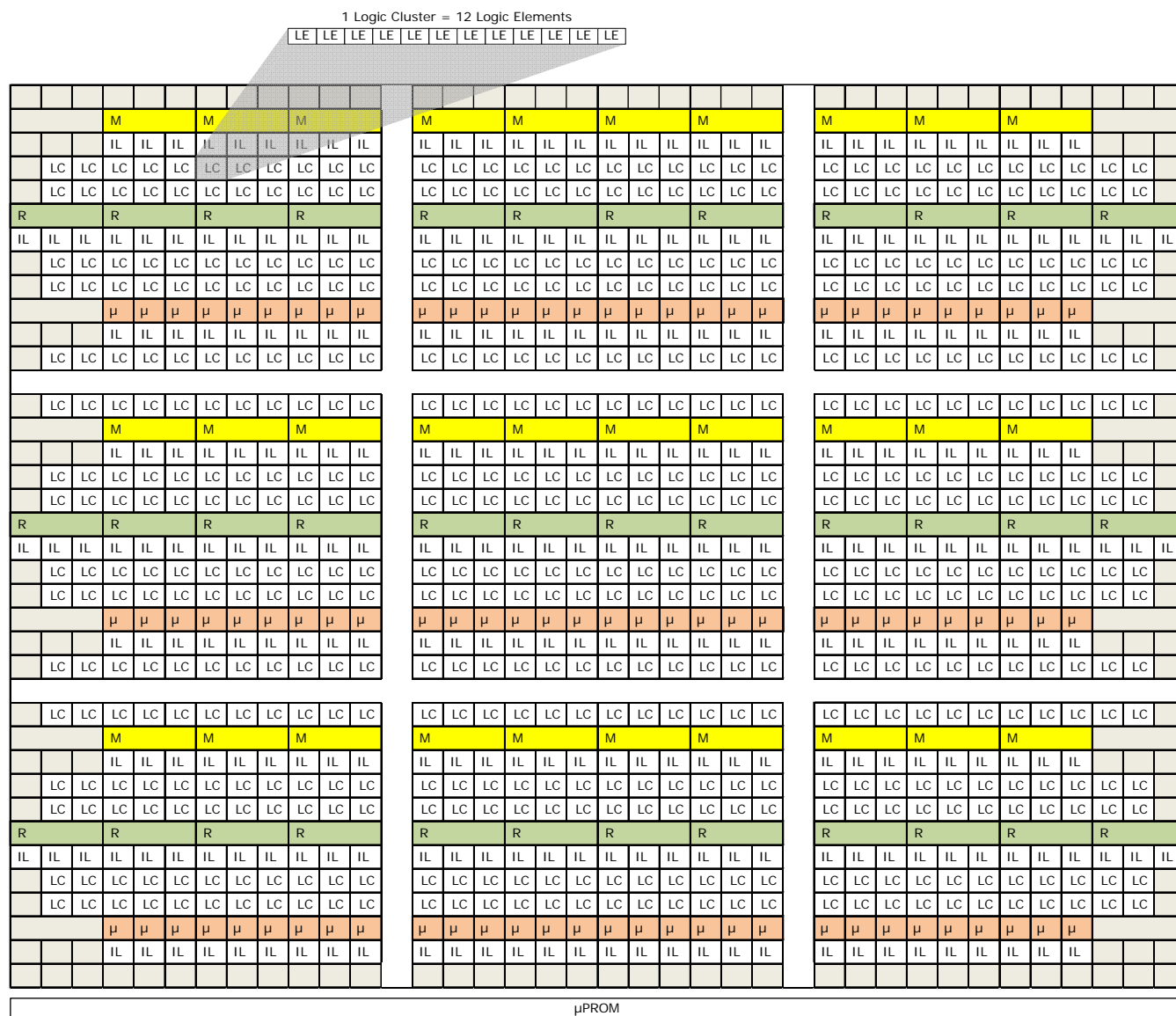
Table 1. Fabric Components

Component	PolarFire® FPGA (MPF)	PolarFire SoC FPGA (MPFS)	RT PolarFire FPGA (RTPF)	RT PolarFire SoC FPGA (RTPFS)
Logic Elements	✓	✓	✓	✓
Embedded Memory Blocks	large SRAM (LSRAM)	✓	✓	✓
	microSRAM (μSRAM)	✓	✓	✓
	microPROM (μPROM)	✓	✓	✓
	secure non-volatile memory (sNVM)	✓	✓	✓
	eNVM	—	✓	—
Math Blocks	✓	✓	✓	✓

Microchip's Libero® SoC Design Suite provides LSRAM, μSRAM, μPROM, and Math IP blocks. All these IP blocks belong to the PolarFire family and can be seamlessly used in PolarFire SoC, RT PolarFire and RT PolarFire SoC designs.

The fabric layout is shown in Figure 1. 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.

Figure 1. Fabric Layout



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

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

Table 2. Fabric Resources in PolarFire Devices

Resources	PolarFire® Devices			
	MPF100	MPF200	MPF300	MPF500
Logic elements (4LUT + DFF)	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.

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

Table 3. Fabric Resources in PolarFire SoC Devices

Resources	PolarFire® SoC Devices				
	MPFS025	MPFS095	MPFS160	MPFS250	MPFS460
Logic elements (4LUT + DFF)	23,000	93,000	161,000	254,000	461,000
Interface logic	7,920	32,112	54,576	85,680	1,54,800
LSRAM blocks (20 Kb each)	84	308	520	812	1,460
μSRAM blocks (768 bits each)	204	876	1,494	2,352	4,260
Total RAM (Mb)	1.8	6.7	11.3	17.6	31.6
Math blocks (18 × 18 MACC)	68	292	498	784	1,420
μPROM (Kb)	194	387	415	470	553

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

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

Table 4. Fabric Resources in RT PolarFire Devices

Resources	RT PolarFire® Devices (RTPF500T and RTPF500ZT)
Logic elements (4LUT + DFF)	319,992
Interface logic	161,280
Total logic	481,272
LSRAM blocks (20 Kb each)	1,520
Total LSRAM bits (Mb)	29.69
μSRAM blocks (768 bits each)	4,440
Total μSRAM bits (Mb)	3.25
Total RAM (Mb)	32.94
Math blocks (18 × 18 MACC)	1,480
μPROM (Kb)	513

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

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

Table 5. Fabric Resources in RT PolarFire SoC Devices

Resources	RT PolarFire® SoC Devices	
	RTPFS160ZT	RTPFS460ZT
Logic elements (4LUT + DFF)	161,000	461,000
Interface logic	54,576	1,54,800
LSRAM blocks (20 Kb each)	520	1,460
μSRAM blocks (768 bits each)	1,494	4,260
Total μSRAM bits (Mb)	11.3	31.6
Math blocks (18 × 18 MACC)	498	1,420
μPROM (Kb)	415	553

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

References [\(Ask a Question\)](#)

The following is the list of reference documents.

- For information about using Libero SoC for PolarFire, RT PolarFire, PolarFire SoC, and RT PolarFire SoC, see [Libero SoC Documentation](#).
- For information about the initialization of the fabric memory blocks during power-up, see [PolarFire Family Power-Up and Resets User Guide](#).
- For information about the security features in both product families, see [PolarFire Family FPGA Security User Guide](#).
- For information about the PolarFire SoC MSS, see [PolarFire SoC MSS Technical Reference Manual](#).
- For information about the RT PolarFire SoC MSS, see *RT PolarFire SoC FPGA MSS Technical Reference Manual*.

Table of Contents

Introduction.....	1
References.....	4
1. Logic Element and Routing.....	6
1.1. Logic Element.....	6
1.2. Interface Logic.....	6
1.3. Logic Cluster.....	9
1.4. Routing Architecture.....	9
1.5. Fabric X-Y Coordinates.....	9
2. Embedded Memory Blocks.....	11
2.1. LSRAM.....	12
2.2. μ SRAM.....	45
2.3. μ PROM.....	55
2.4. sNVM.....	69
2.5. eNVM (PolarFire SoC and RT PolarFire SoC Only).....	70
3. Math Blocks.....	71
3.1. Features.....	71
3.2. Math Block Resources.....	71
3.3. Functional Description.....	71
3.4. Cascading Math Blocks.....	76
3.5. Operational Modes.....	78
3.6. Implementation.....	81
4. Appendix: SRAM Optimization for High-Speed Versus Low-Power.....	89
5. Appendix: Supported Memory File Formats for LSRAM and μ SRAM.....	91
5.1. Write Port Width Alignment.....	93
6. Appendix: Macro Configuration.....	101
6.1. LSRAM Macro.....	101
6.2. μ SRAM Macro.....	108
6.3. Math Block Macro.....	110
6.4. Libero SoC Compile Report.....	121
7. Revision History.....	126
Microchip FPGA Support.....	130
Microchip Information.....	131
Trademarks.....	131
Legal Notice.....	131
Microchip Devices Code Protection Feature.....	131

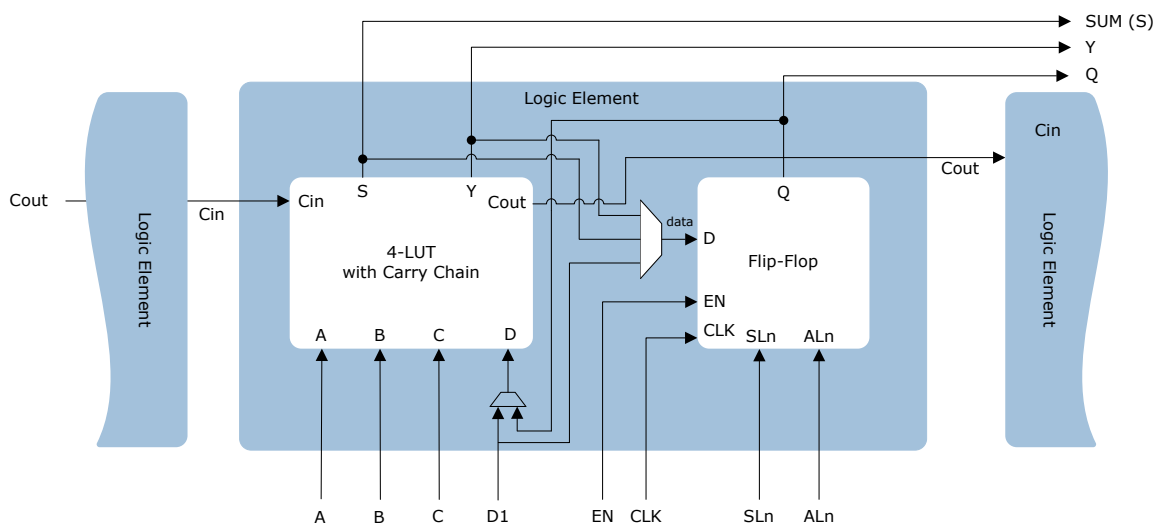
1. Logic Element and Routing [\(Ask a Question\)](#)

The 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. See [Figure 1-7](#) and [Figure 1-8](#) for PolarFire and PolarFire SoC fabric coordinates respectively. See [Figure 1-9](#) and [Figure 1-10](#) for RT PolarFire and RT PolarFire SoC fabric coordinates respectively.

1.1. Logic Element [\(Ask a Question\)](#)

The logic element consists of a 4-input Lookup Table (LUT) with a carry chain and D-type flip-flop, as shown in [Figure 1-1](#). The logic element is fracturable, which means that the LUT can be independently used without flip-flop, or flip-flop can be used without LUT.

Figure 1-1. 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 1-1](#)). 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).

1.2. Interface Logic [\(Ask a Question\)](#)

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

The following table lists the total number of ILs associated with each memory block in the PolarFire devices.

Table 1-1. ILs for Embedded Hard IP Blocks (PolarFire Devices)

Resources	MPF100		MPF200		MPF300		MPF500	
	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs	No. of Blocks	No. 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

The following table lists the total number of ILs associated with each memory block in the PolarFire SoC devices.

Table 1-2. ILs for Embedded Hard IP Blocks (PolarFire SoC Devices)

Resources	MPFS025		MPFS095		MPFS160		MPFS250		MPFS460	
	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs
LSRAM	84	3024	308	11088	520	18720	812	29232	1460	52560
μSRAM	204	2448	876	10512	1494	17928	2352	28224	4260	51120
Math block	68	2448	292	10512	498	17928	784	28224	1420	51120
Total interface logic	—	7920	—	32112	—	54576	—	85680	—	154800

The following table lists the total number of ILs associated with each memory block in the RT PolarFire devices.

Table 1-3. ILs for Embedded Hard IP Blocks (RT PolarFire Devices)

Resources	RT PolarFire® Devices	
	No. of Blocks	No. of ILs
LSRAM	1,520	54,720
μSRAM	4,440	53,280
Math block	1,480	53,280
Total interface logic	—	161,280

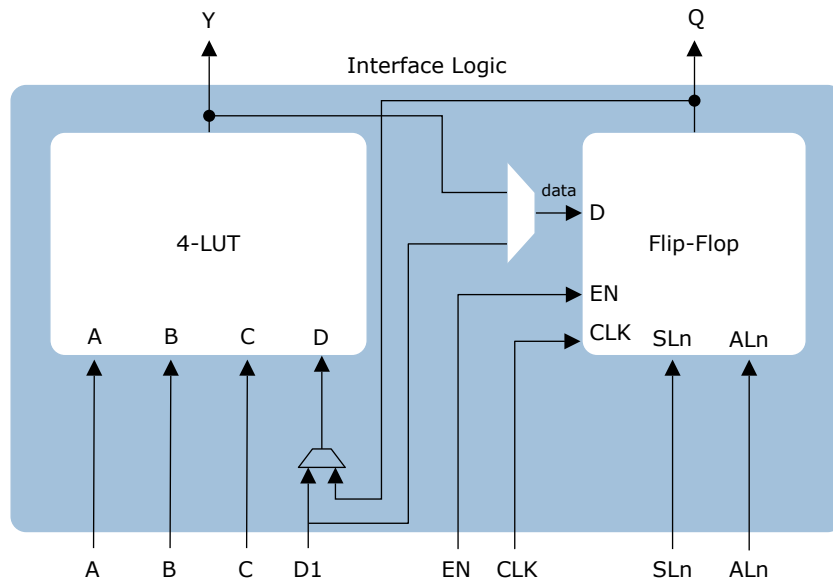
The following table lists the total number of ILs associated with each memory block in the RT PolarFire SoC devices.

Table 1-4. ILs for Embedded Hard IP Blocks (RT PolarFire SoC Devices)

Resources	RTPFS160		RTPFS460	
	No. of Blocks	No. of ILs	No. of Blocks	No. of ILs
LSRAM	520	18720	1460	52560
μSRAM	1494	17928	4260	51120
Math block	498	17928	1420	51120
Total interface logic	—	54576	—	154800

The 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 the following figure.

Figure 1-2. Functional Block Diagram of Interface Logic



Each LSRAM and Math block is associated with 36 ILs, and each μ SRAM is associated with 12 ILs. For more information, see [Figure 1-3](#), [Figure 1-4](#), and [Figure 1-5](#).

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 1-3. LSRAM Interfacing with ILs in a Row

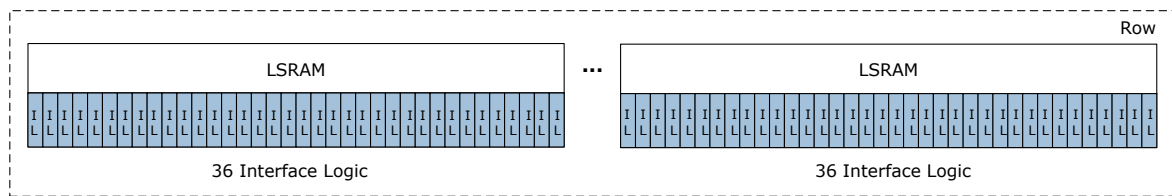


Figure 1-4. Math Block Interfacing with ILs in a Row

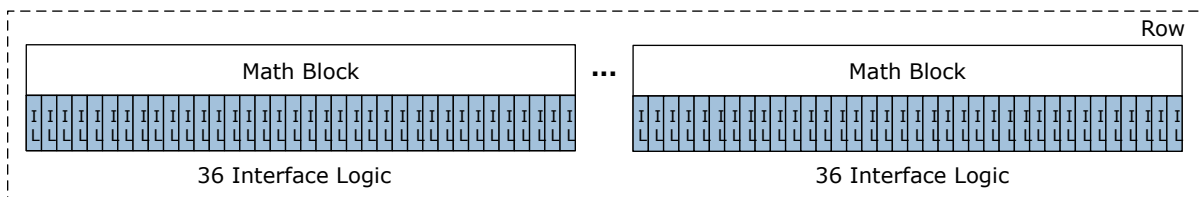
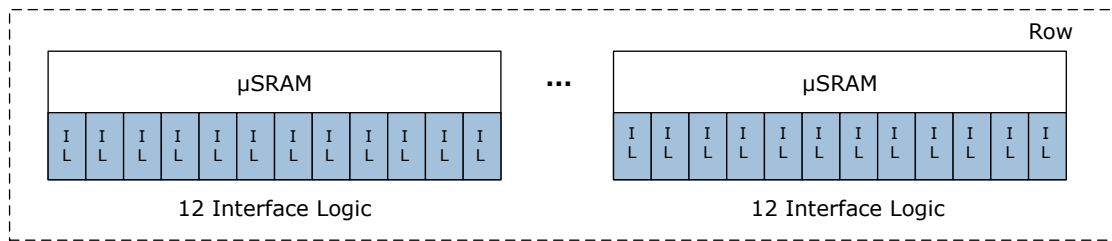


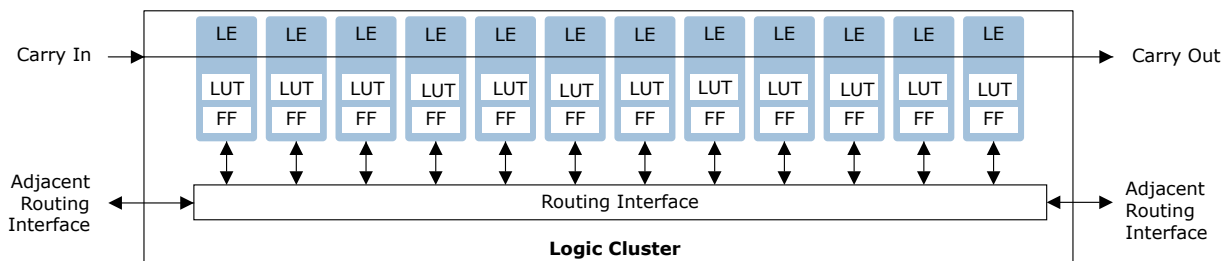
Figure 1-5. μ SRAMs Interfacing with ILs in a Row

1.3. Logic Cluster [\(Ask a Question\)](#)

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.

The following figure shows the logic cluster with its routing interface.

Figure 1-6. Logic Cluster



1.4. Routing Architecture [\(Ask a Question\)](#)

The PolarFire family supports 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.

1.5. Fabric X-Y Coordinates [\(Ask a Question\)](#)

Each 4-input LUT, D-type flip-flop, carry chain, LSRAM, μ SRAM, 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 details, see *SmartTime User Guide*, *I/O Editor User Guide*, and *ChipPlanner User Guide* on the [Liberio SoC](#) page.

The following figure shows the available X-Y coordinates of LSRAM, μ SRAM, and Math block for placement constraints for the PolarFire MPF300 device.

Figure 1-7. MPF300 Fabric X-Y Coordinates

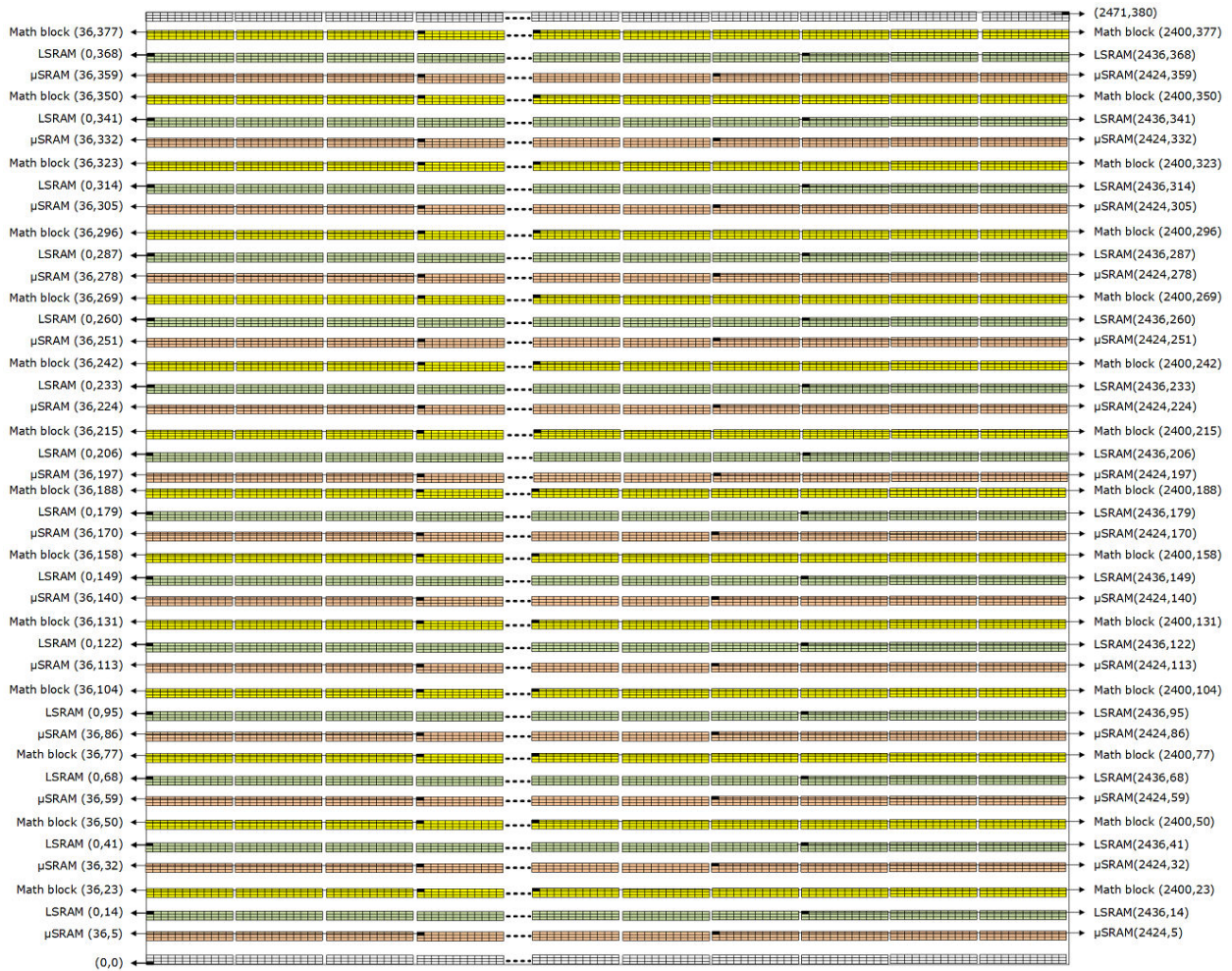


Figure 1-8 shows the available X-Y coordinates of LSRAM, μSRAM, and Math block for placement constraints for the PolarFire SoC MPFS250 device.

Figure 1-8. MPFS250 Fabric X-Y Coordinates

To be updated.

The following figure shows the available X-Y coordinates of LSRAM, μSRAM, and Math block for placement constraints for the RT PolarFire RTPF500 device.

Figure 1-9. RTPF500 Fabric X-Y Coordinates

To be updated.

The following figure shows the available X-Y coordinates of LSRAM, μSRAM, and Math block for placement constraints for the RT PolarFire SoC RTPFS460 device.

Figure 1-10. RTPFS460 Fabric X-Y Coordinates

To be updated.

2. Embedded Memory Blocks (Ask a Question)

The PolarFire family 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 2](#) and [Table 3](#). The LSRAMs support ECC when configured in 33-bit data width in two-port mode. LSRAMs can be configured in various modes as shown in [Table 2-1](#). LSRAMs can be initialized with user data during power-up.
- **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 device, as shown in [Table 2](#) and [Table 3](#). μSRAMs can be initialized during power-up.
- **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 KB 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. The usable sNVM is 48 KB on RTPF500ZT, RTPFS160ZT and RTPFS460ZT due to the addition of built-in SECDED ECC.

Along with the above memory blocks, the PolarFire SoC and RT PolarFire SoC family includes the following memory block:

- **eNVM (PolarFire SoC and RT PolarFire SoC Only)**—The 128 KB of eNVM is located in the MSS. eNVM is used to store the first stage bootloader program for booting the E51 monitor core. eNVM is programmed with the device bitstream during the device programming, it cannot be programmed independently.

 **Important:** eNVM is 102 KB for RT PolarFire SoC devices when using added ECC for radiation mitigation. For more information, see [AN4903: Differences Between RT PolarFire RTPF500T and RTPF500ZT FPGAs](#).

The following table lists the features of the memory blocks of the PolarFire family.

Table 2-1. LSRAM, μSRAM, μPROM, and sNVM Features

Feature	LSRAM	μSRAM	μPROM	sNVM	eNVM (Only PolarFire SoC and RT PolarFire SoC)
Memory size	20,480 bits/block	768-bit/block	See Table 2 and Table 3	<ul style="list-style-type: none"> • 56 Kbytes • 48 Kbytes (RTPF500ZT, RTPFS160ZT and RTPFS460ZT) 	128 Kbytes. See the preceding note .
Memory Configuration Options	16K × 1, 8K × 2, 4K × 5, 2K × 10, 1K × 20, 512 × 40 ¹ , and 512 × 33 ¹ (with ECC)	64 × 12	Up to 64K × 9	Not Applicable	Not Applicable
Number of ports	2 read ports, 2 write ports	1 read port, 1 write port	1 read port	Not Applicable	Not Applicable

Feature	LSRAM	μ SRAM	μ PROM	sNVM	eNVM (Only PolarFire SoC and RT PolarFire SoC)
Memory modes	True dual-port and two-port	Two-port	Single-port	Not Applicable	Not Applicable
Read operation	Synchronous	Synchronous/Asynchronous	Asynchronous	Through system service calls	Asynchronous
Write operation	Simple write, feed-through write, and read-before-write	Simple write	Only during device programming	During device programming and System Service calls	During device programming
ECC	Available for two-port mode (512 \times 33) only	Not available	Not Applicable	Built-in for RTPFS500ZT, RTPFS160ZT and RTPFS460ZT.	Not Applicable for MPFS devices. Software defined ECC is available for RTPFS160ZT and RTPFS460ZT.

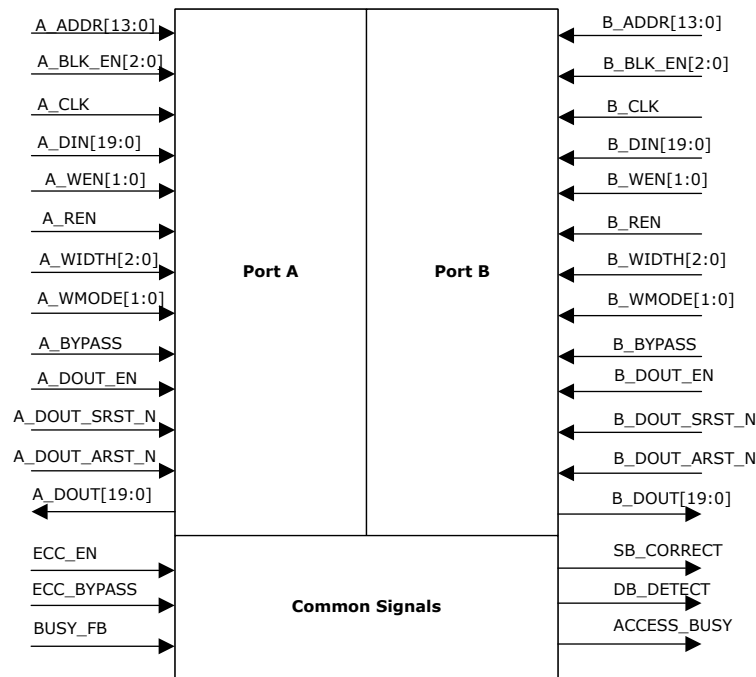
Note:


1. \times 40 and \times 33 are available only in two-port mode.

2.1. LSRAM [\(Ask a Question\)](#)

Each LSRAM has two independent ports—Port A and Port B, as shown in [Figure 2-1](#). Both these ports support write and read operations, and can be configured in dual-port mode or two-port mode.

Figure 2-1. LSRAM Input/Output



 **Important:** 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.

The following table lists the ports of LSRAM.

Table 2-2. 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 high	Port A read data pipeline register bypassed when High
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 high	Port B read data pipeline register bypassed when High
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 high	ECC pipeline register bypassed when High.
SB_CORRECT	Output	Dynamic	Active high	Single-bit correct flag
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

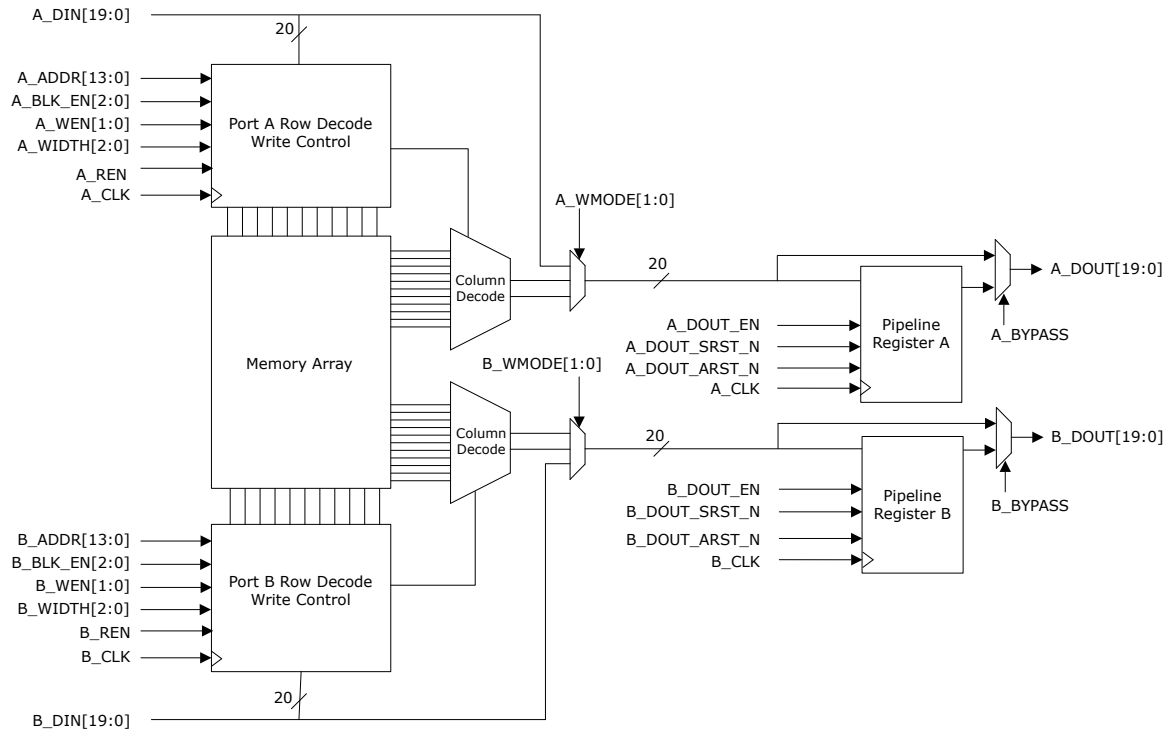
Note:

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

2.1.1. Dual-Port Mode [\(Ask a Question\)](#)

The LSRAM block can be configured as a true dual-port SRAM with independent write and read ports, as shown in [Figure 2-2](#). 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 2-2. Simplified Functional Block Diagram of LSRAM in Dual-Port Mode



2.1.1.1. Dual-Port Data Width Configuration [\(Ask a Question\)](#)

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 2-3](#).

Table 2-3. 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

2.1.1.2. Block Select Operation in Dual-Port Mode (Ask a Question)

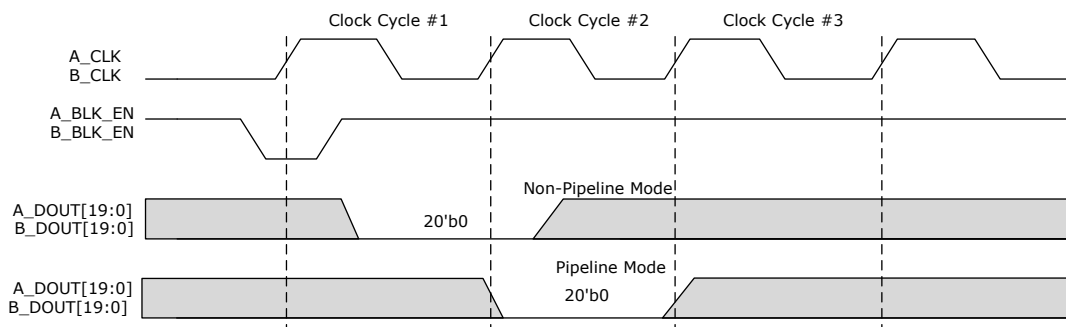
In Dual-Port mode, to perform two independent write and read operations (on Port A, Port B, or both) the block select signal is required. Table 2-4 lists the block select operation for Port A and Port B.

Table 2-4. Block Select Operation in Dual-Port Mode

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 the following figure.

Figure 2-3. Block Select Inputs for Dual-Port Mode



2.1.1.3. Byte Write Enables (Ask a Question)

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 2-5 lists the byte write enable settings for Port A and Port B.

Table 2-5. 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]

Table 2-5. Byte Write Enables Settings for Dual-Port Mode (continued)

Depth x Width	A_WEN/B_WEN	Result
1K x 20	00	Perform a read operation
	01	Write [9:0]
	10	Write [19:10]
	11	Write [19:0]

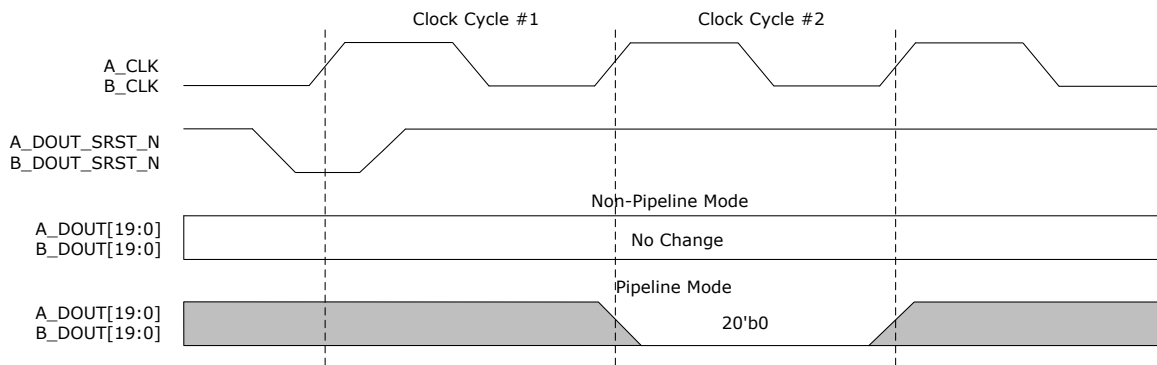
2.1.1.4. Read Enable [\(Ask a Question\)](#)

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.

2.1.1.5. Synchronous Pipeline Register Reset [\(Ask a Question\)](#)

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 the following figure.

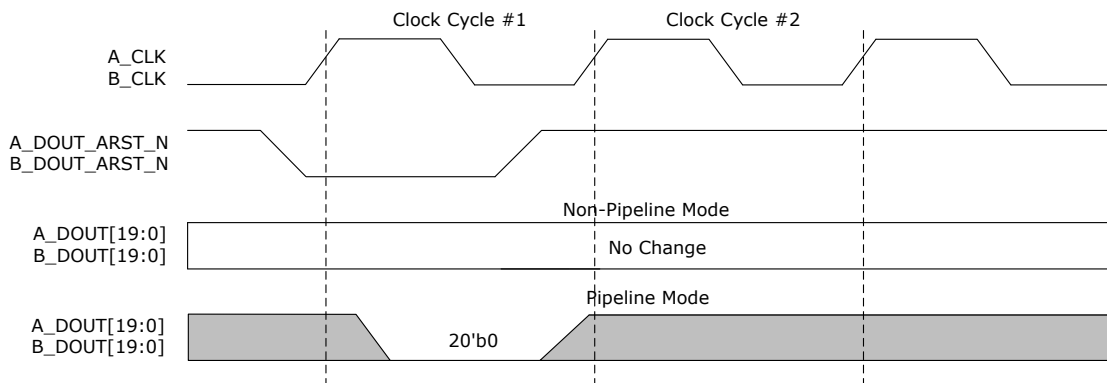
Figure 2-4. Synchronous Pipeline Register Reset in Dual-Port Mode



2.1.1.6. Asynchronous Pipeline Register Reset [\(Ask a Question\)](#)

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 the following figure.

Figure 2-5. Asynchronous Pipeline Register Reset in Dual-Port Mode



2.1.1.7. Read Operation [\(Ask a Question\)](#)

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.

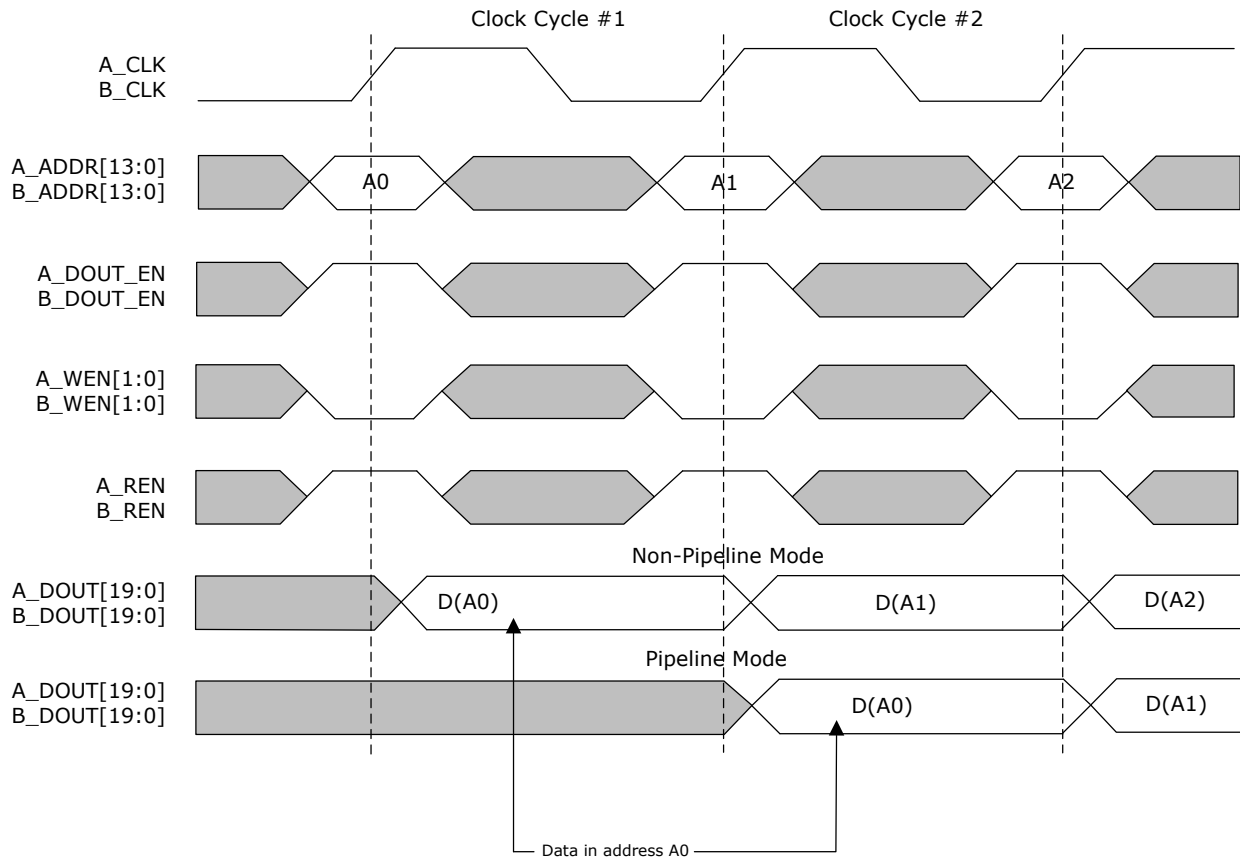


Important:

- For high-performance designs, It is recommended to use the LSRAM with pipeline mode to meet the design timing constraints.
- When multiple depth-cascaded blocks are used, A_REN and B_REN ports of Dual-Port SRAM are disabled by the configurator GUI.

The following figure shows the timing for both pipelined and non-pipelined read operations in the Dual-Port mode.

Figure 2-6. Read Operation in Dual-Port Mode



2.1.1.8. Write Operation [\(Ask a Question\)](#)

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. As simultaneous write operations can result in data uncertainty, it is recommended to use external logic in the fabric to avoid collisions.

2.1.1.8.1. Simple Write [\(Ask a Question\)](#)

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.

2.1.1.8.2. Feed-Through Write [\(Ask a Question\)](#)

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 2-7](#).

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.

2.1.1.8.3. Read-Before-Write [\(Ask a Question\)](#)

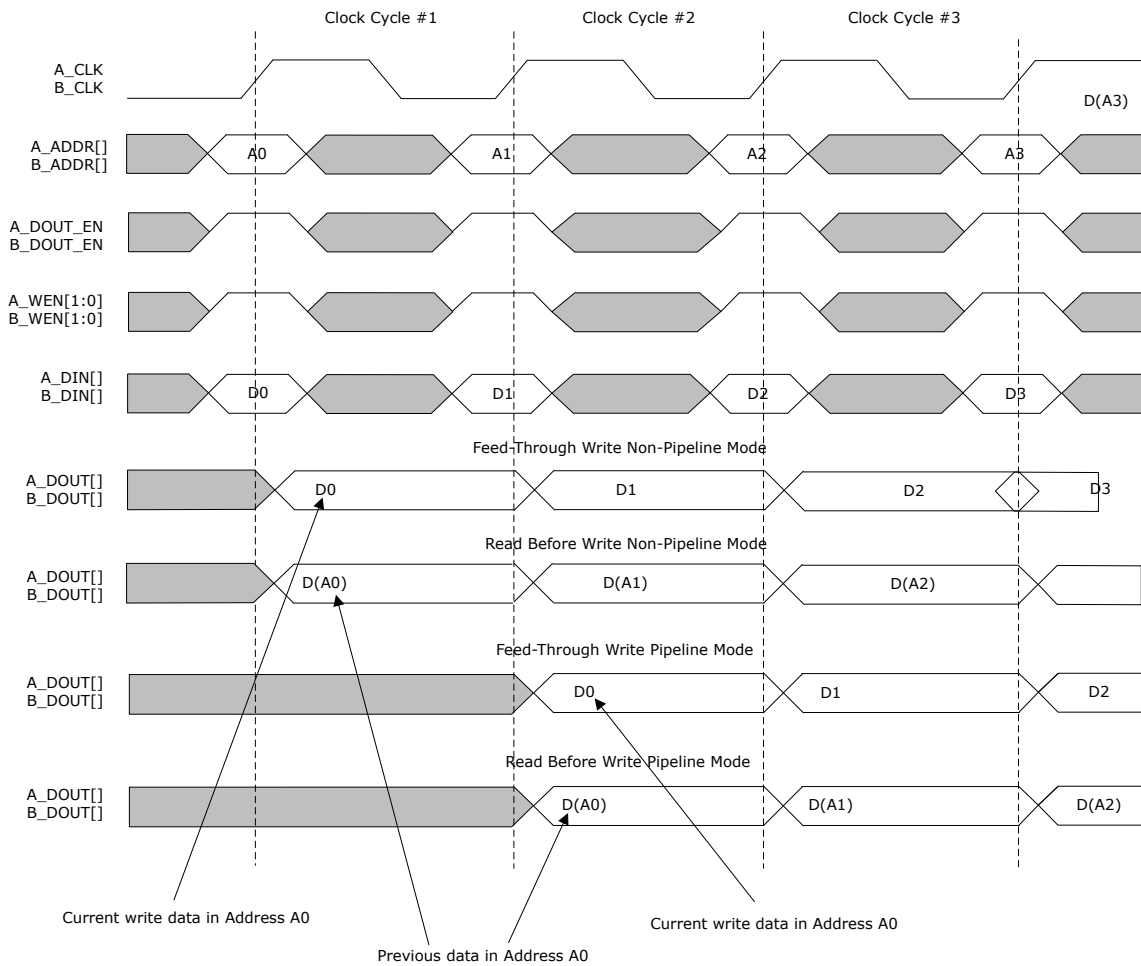
In a read-before-write operation for pipeline mode, read data is available on the data output bus on the next clock cycle. For non-pipeline mode, the previous memory data from the current write address is available on the data output before the new data is written to the address location. For more information, see [Figure 2-7](#).

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.

The following figure shows the timing for feed-through-write and read-before-write operations for Dual-Port mode.

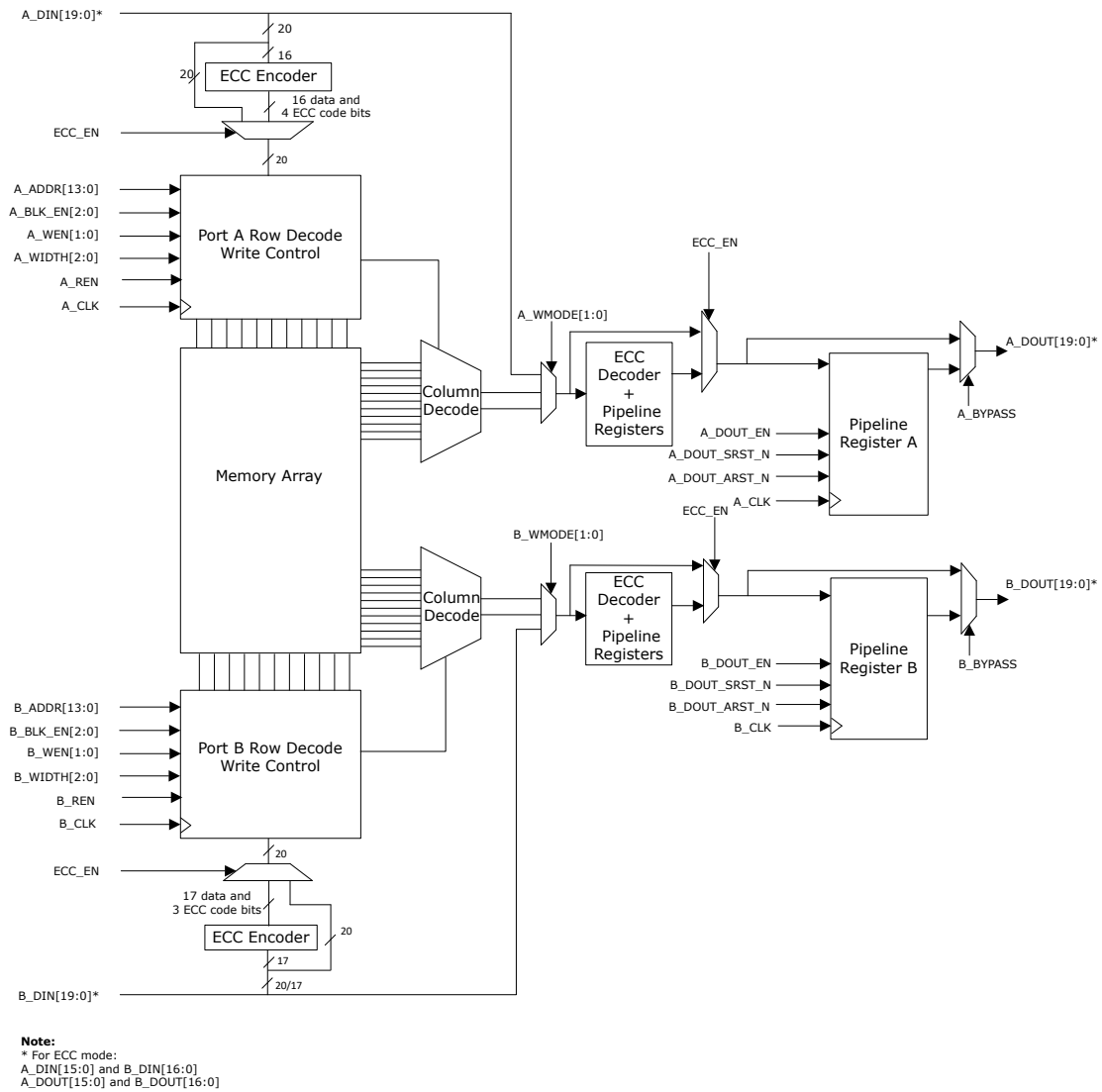
Figure 2-7. Write Operations in Dual-Port Mode



2.1.2. Two-Port Mode [\(Ask a Question\)](#)

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 2-8 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 2-8. Simplified Functional Block Diagram for LSRAM in Two-Port Mode



2.1.2.1. Two-Port Data Width Configuration (Ask a Question)

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 2-6.

Table 2-6. 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

Table 2-6. LSRAM Data Width Configurations (Two-Port Mode) (continued)

Read Port	Write Port
x33	x33
x32	x1, x2, x32

2.1.2.2. Block Select Operation in Two-Port Mode [\(Ask a Question\)](#)

In Two-Port mode, independent read and write operations require the use of the block select signals. A_BLK_EN[2:0] controls the read port (Port A), while B_BLK_EN[2:0] controls the write port (Port B). The block selection for Port A operates independently of the block selection for Port B.

The table below summarizes the block select operation for the read and write ports in Two-Port mode.

Table 2-7. Block Select Operation in Two-Port Mode

A_BLK_EN[2:0]	B_BLK_EN[2:0]	Operation
Any one bit = 0	Any one bit = 0	No operation on the read or write port The data output A_DOUT[19:0] is forced zero.
Any one bit = 0	111	Write operation on Port B No read operation on Port A
111	Any one bit = 0	Read operation on Port A No write operation on Port B
111	111	Read operation on Port A and write operation on Port B

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


2.1.2.3. Byte Write Enables [\(Ask a Question\)](#)

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 2-8 lists the byte write enable settings for Port A and Port B.

Table 2-8. 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]

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

2.1.2.4. Read Enables [\(Ask a Question\)](#)

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.

➔ Important: In Two-Port mode, LSRAM Port B read enable (B_REN) is tied to Port A read enable (A_REN).

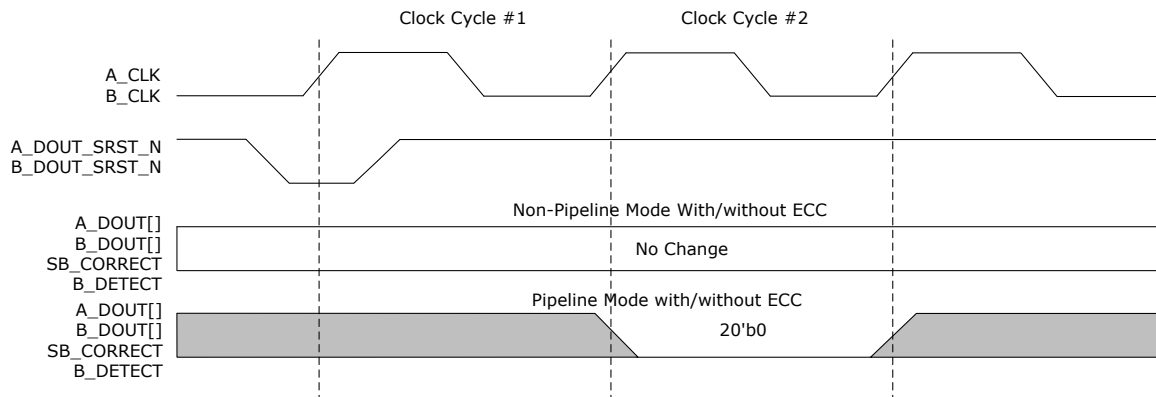
2.1.2.5. Pipeline Registers [\(Ask a Question\)](#)

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.

2.1.2.5.1. Synchronous Pipeline Register Reset [\(Ask a Question\)](#)

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 the following figure.

Figure 2-9. Synchronous Pipeline Reset in Two-Port Mode



➔ Important: In x33 two-port mode, if ECC is in pipeline mode, this reset also resets the ECC flag pipeline registers.

2.1.2.6. ECC Mode (For x33 Two-Port Mode Only) [\(Ask a Question\)](#)

In the 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.

➔ Important: 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 2-20](#).

The ECC decoder contains an optional pipeline register that adds a clock cycle of latency to the read operation (including the flags). As 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 following table lists the two flags generated by the ECC logic.

Table 2-9. Error Flags

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

Any multiple bit errors greater than one has both flags asserted, and the 33-bit read data word not corrected. No scrubbing is done inside the LSRAM when ECC decoder detects any bit errors. All scrubbing must be done in the fabric design. ECC simulation is not supported.

In the 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.

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 [PolarFire Family Power-Up and Resets User Guide](#).

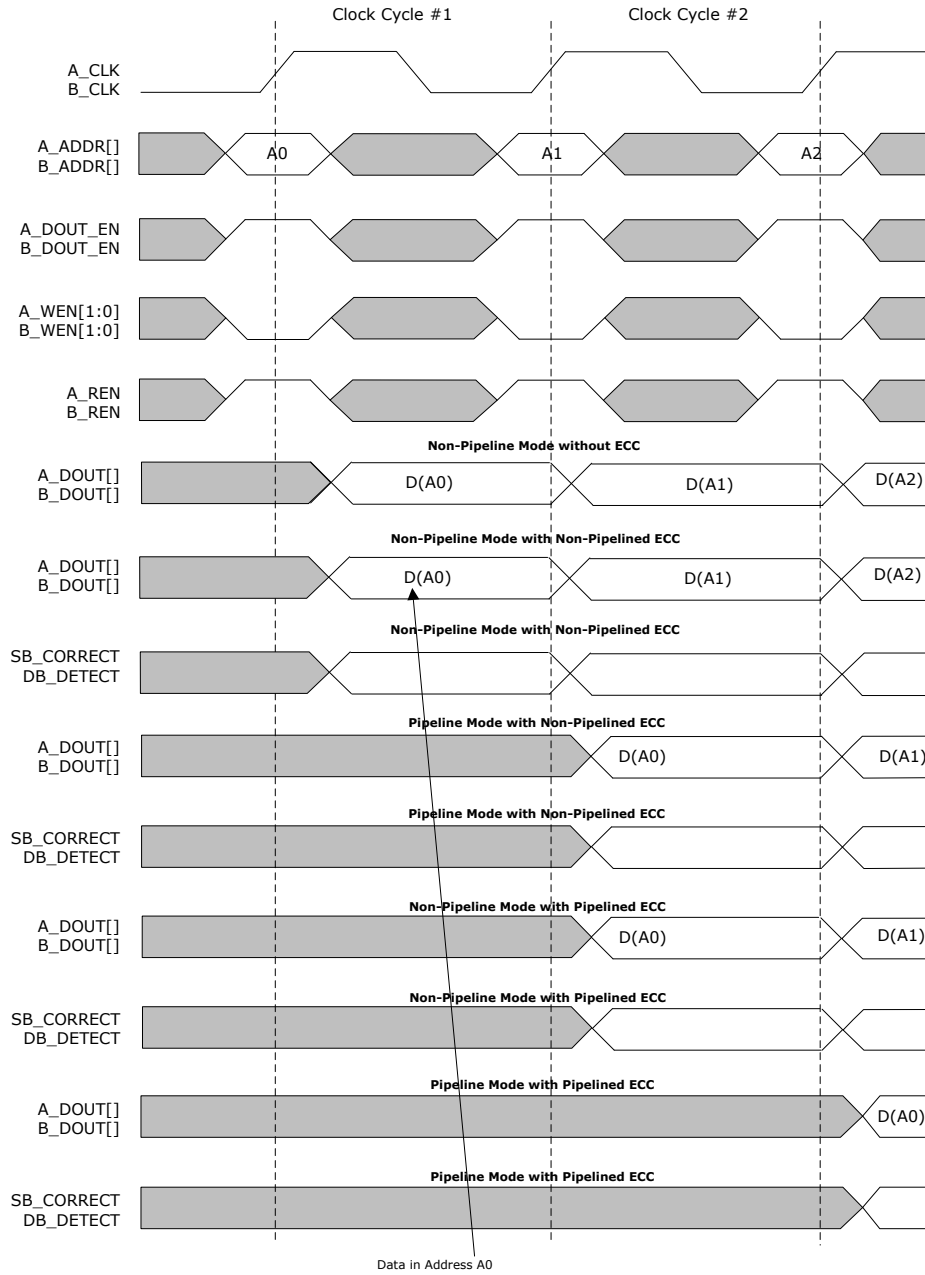
2.1.2.7. Read Operation [\(Ask a Question\)](#)

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.

The following figure shows the timing for both pipelined and non-pipelined read operations in two-port mode.

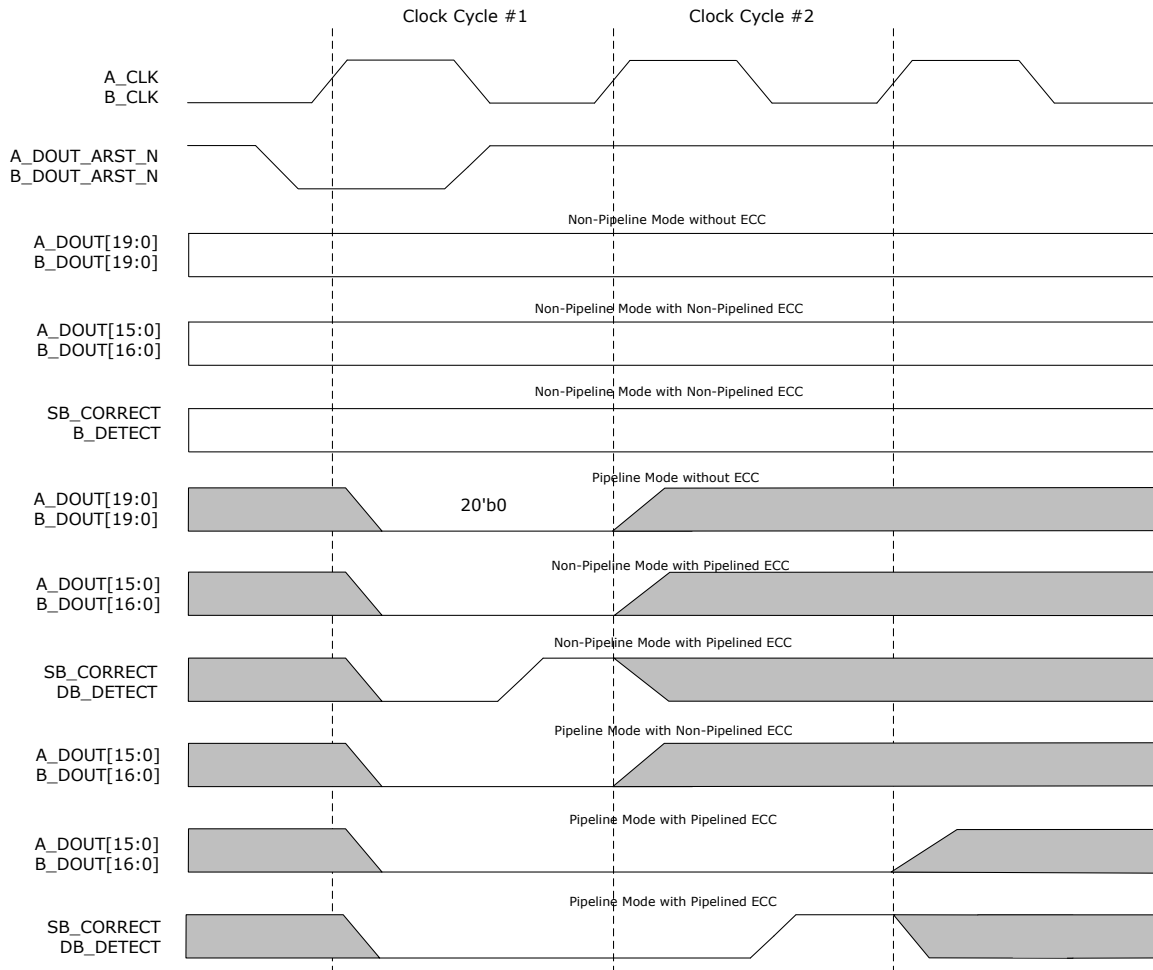
Figure 2-10. Read Operation in Two-Port Mode



2.1.2.8. Asynchronous Pipeline Register Reset [\(Ask a Question\)](#)

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 the following figure.

Figure 2-11. Asynchronous Pipeline Register Reset in Two-Port Mode



Important: In x33 Two-Port mode, if ECC is in pipeline mode, then this reset also resets the ECC flag pipeline registers.

2.1.2.9. Write Operation [\(Ask a Question\)](#)

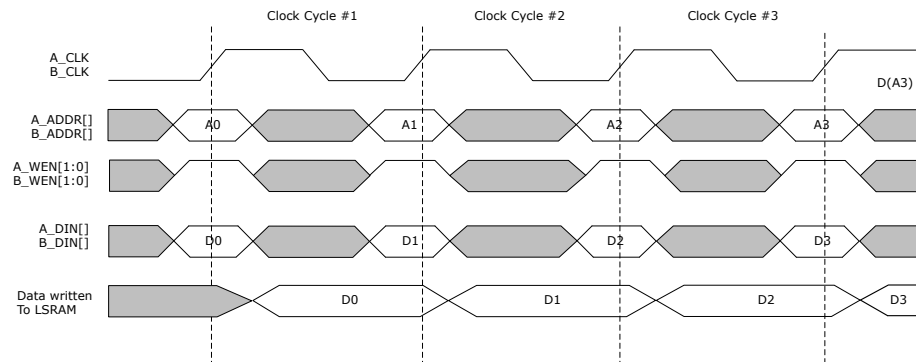
In Two-Port mode, LSRAM supports only Simple Write operations. Port B is the dedicated write port for data widths up to x20. For data widths greater than x20 (x32, x33 with ECC, and x40), the write operation borrows the unused Port A data input signals (A_DIN) to extend the data width.

Simple Write mode is selected by setting B_WMODE[1:0] = 00 and A_WMODE[1:0] = 00 when Port A input pins are borrowed.

All write port inputs such as B_DIN, borrowed A_DIN, B_ADDR, B_WEN, A_WEN, and B_BLK_EN are synchronous to B_CLK.

The following figure shows the write operation timing.

Figure 2-12. Write Operation in Two-Port Mode



Once a valid read operation is performed, the written data is available on the data output ports.

2.1.3. Implementation [\(Ask a Question\)](#)

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

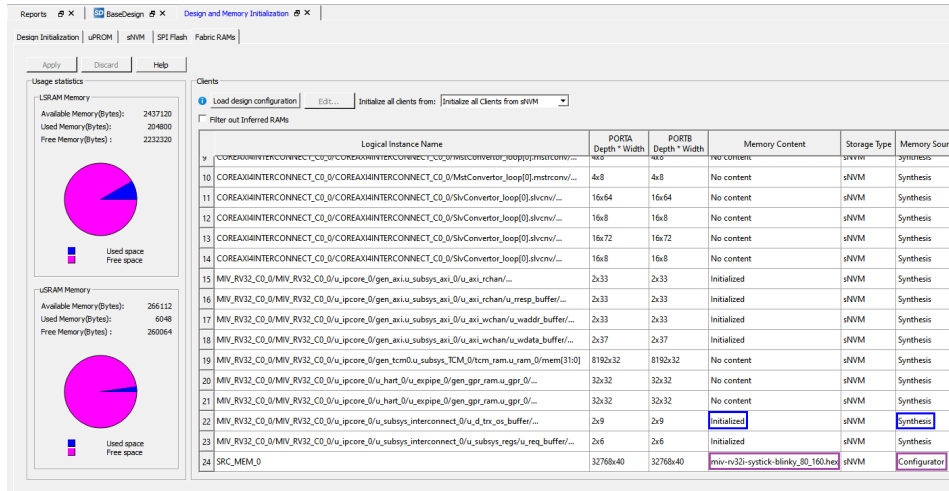
- [RTL Inference During Synthesis](#)
- [LSRAM Configurator](#)
- [LSRAM Memory Macro](#)

2.1.3.1. RTL Inference During Synthesis [\(Ask a Question\)](#)

Synplify Pro[®] 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 PolarFire RAM Blocks Application Note](#).

Synplify Pro can also infer initialized RAM blocks, as well as ROMs, if the prescribed coding styles in the [Inferring PolarFire RAM Blocks Application Note](#) are followed. As noted in [Inferring PolarFire RAM Blocks Application Note](#) and [Appendix: SRAM Optimization for High-Speed Versus Low-Power](#), there are Synthesis directives that can be used to confirm the RAM block cascading styles used. Ultimately, this impacts the way the user memory's data width is split across underlying PolarFire embedded memory block primitive macros and thus determines the user word bit mapping to individual data bits of the primitive macros. In Libero SoC design flow, after a synthesized netlist containing inferred initialized RAM blocks completes **Place and Route** and **Generate FPGA Array Data**, the **Configure Design Initialization Data and Memories** tool lists the Fabric RAMs by Logical Instance Name. Each inferred initialized instance is described as having Initialized Memory Content and lists the Memory Source as Synthesis instead of the LSRAM configurator, as shown in the following figure.

Figure 2-13. Inferred Initialized RAM vs Configurator



For more information about the Libero SoC **Design and Memory Initialization** tool, see the "How To Set Up Design and Memory Initialization" section in [PolarFire Family Power-Up and Resets User Guide](#).

When inferring RAM blocks during Synthesis, Synplify Pro might use RAM cascading styles and data word bit mapping to RAM block data bits that differ from the LSRAM configurator in the Libero SoC IP Catalog. In the Libero SoC design flow, although the **Configure Design Initialization Data and Memories** step allows the user to modify the RAM initialization content for an inferred RAM block, this is not recommended because it can result in incorrect design functionality when the data word bit mapping to individual RAM block data bit pins differs from the LSRAM core configurator's bit mapping. Instead, the initialization content for inferred RAMs must be provided along with the user HDL source files, such as using the `$readmemb`, `$readmemh`, or similar syntax, as shown in the code examples 61–67 for ROMs and 79–82 for initialized RAMs in [Inferring PolarFire RAM Blocks Application Note](#). These HDL examples are synthesizable for the PolarFire family of devices using the Synplify Pro[®] Synthesis software. If the recommended syntax is followed, the fabric RAM blocks are inferred and the user supplied initialization content is forward annotated to the netlist. The **Configure Design Initialization Data and Memories** step, in the Libero SoC design flow, includes the **Fabric RAMs** tab that lists these inferred RAM instances with table entries indicating that the **Memory Content** is "Initialized" and the **Memory Source** is "Synthesis".

Updates to the initialization data for inferred RAMs must be treated as an update to RTL source files and must be re-synthesized. This approach also ensures that RTL functional simulation matches all post-synthesis simulation steps and the initial RAM values in the programmed device.

Updating the inferred RAM content in the **Configure Design Initialization Data and Memories** step only works as expected if the breakdown of the inferred RAM data bit mapping matches the LSRAM core configurator's generated bit mapping. For example, when using Low-Power RAM decomposition/cascading style with byte-write enables, the resulting inferred RAM data bit partitioning and mapping should match the Libero SoC LSRAM Core Configurator generated bit partitioning and mapping.

2.1.3.2. LSRAM Configurator [\(Ask a Question\)](#)

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. Use the generated HDL wrapper files in the design hierarchy by connecting the ports to the rest of the design.

2.1.3.2.1. Dual-Port Large SRAM Configurator [\(Ask a Question\)](#)

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 an 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 2-14. Dual-Port Large SRAM Configurator: Generated Component

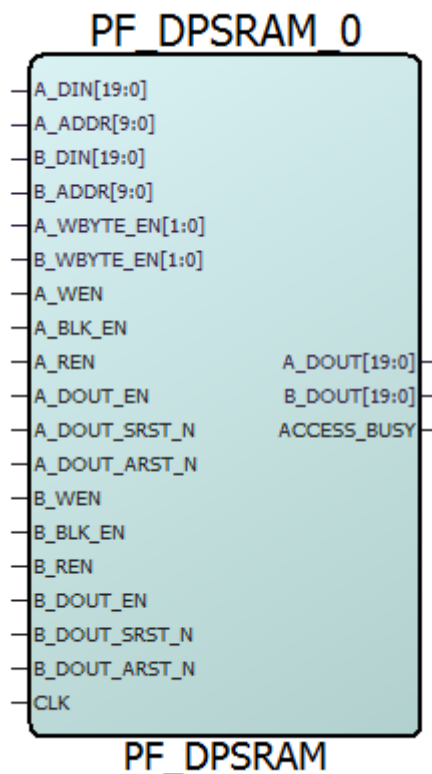


Table 2-10. Dual-Port LSRAM Configurator Signals

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.

Table 2-10. Dual-Port LSRAM Configurator Signals (continued)

Port	Direction	Polarity	Description
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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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

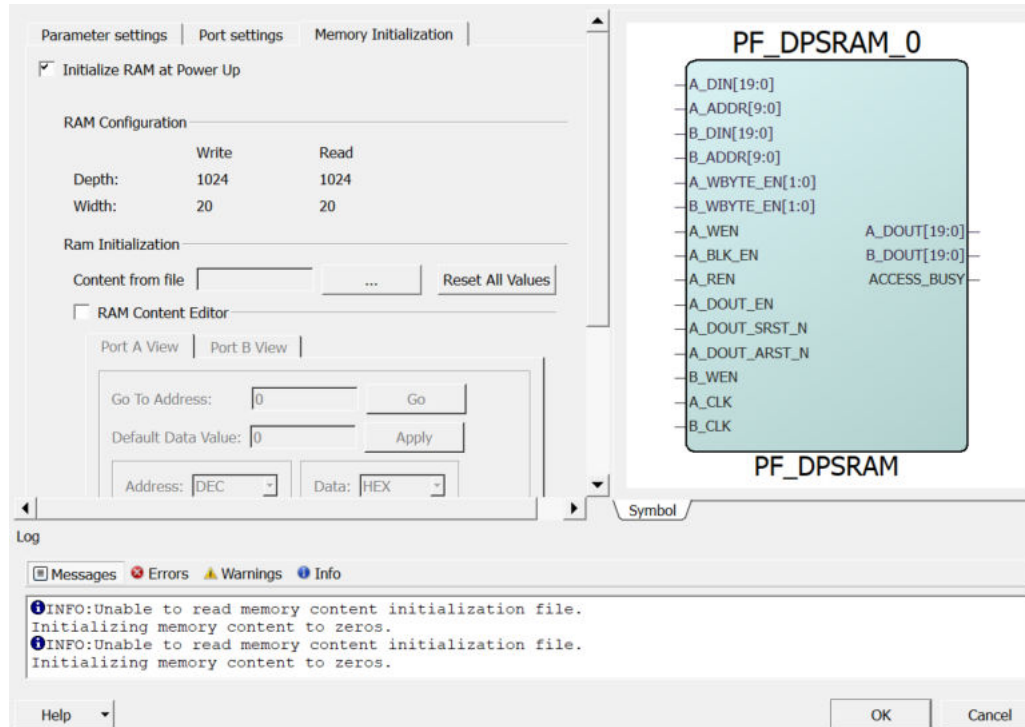
The dual-port LSRAM configurator has three tabs:

- 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. The following figure shows the Dual-Port Large LSRAM block configurator.

Figure 2-15. Dual-Port Large SRAM Configurator: Parameter Settings



Optimization for High Speed or Low Power

You 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)

You 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

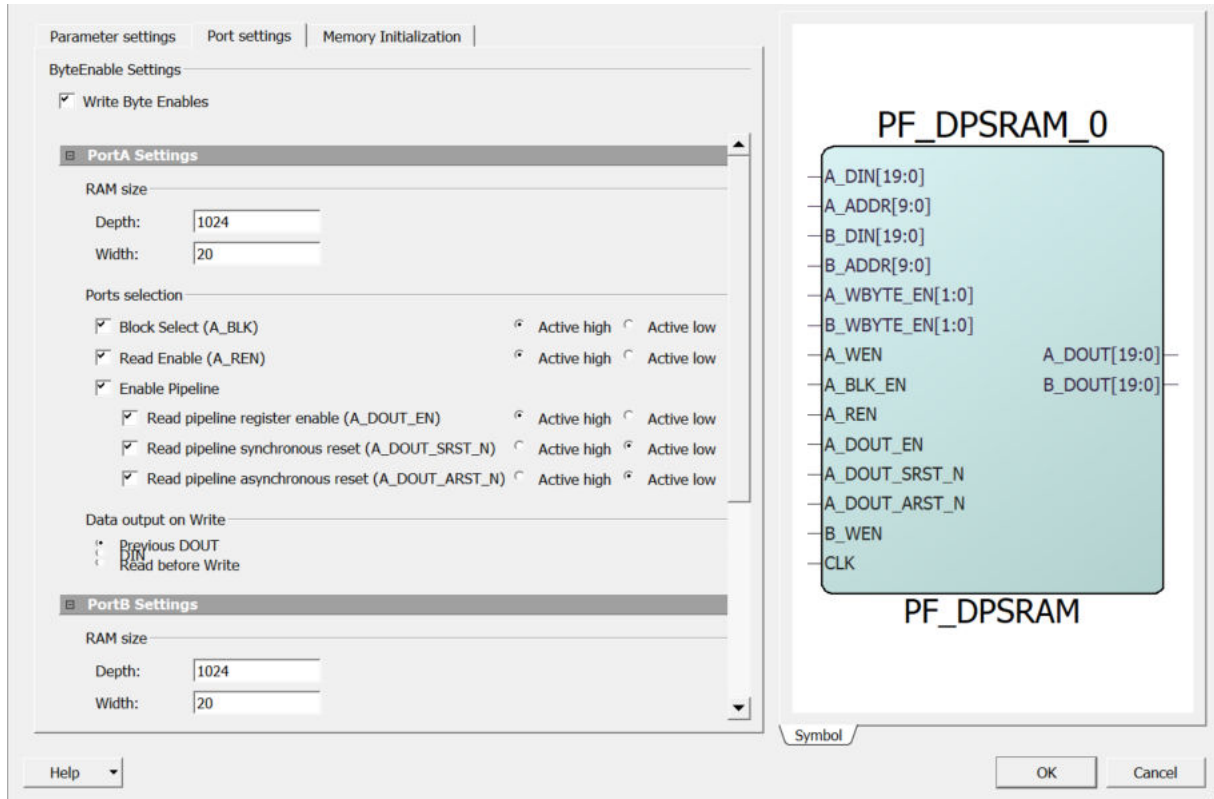
You can select one of the following optional ports:

- **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, you can set the RAM size, select ports, and set data output on write settings for both Ports A and B. The following figure shows the PF_DPSRAM block port settings.

Figure 2-16. Dual-Port Large SRAM Configurator: Port Settings



Byte Enable Settings

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

The Write Byte Enable bits are all the control signals exposed by each column of LSRAM blocks when the implementation splits the word width. The LSRAM configurator generates the most efficient configuration of the depth and width for high-speed or low-power selection. Depending on the generated configuration, each Write Byte Enable bit may control up to 10 bits of data. In other words, the Write Byte Enable option does not govern the implementation.

In low-power mode for word widths that are multiples of 8, each Write Byte Enable bit controls 8 bits unless the word width is also a multiple of 10. For example, generating a 32-bit word width LSRAM, with Write Byte Enables, cascades 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 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

You 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.

 **Important:** 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** to change the signal polarity. Ports are 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 are populated on the component by checking the respective check boxes.

Enable Pipeline

Check the **Enable Pipeline** check box to enable pipelining of read data (A_DOUT or B_DOUT). If the Enable Pipeline check box is disabled, you 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 check box is disabled. Selecting this check box 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 check box is disabled. Selecting this check box 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 check box is disabled. Selecting this check box adds the signal to the top-level port.

Select **Active high** or **Active low** to change the signal polarity. Ports are populated on the component by checking their respective check boxes.

Data Output on Write

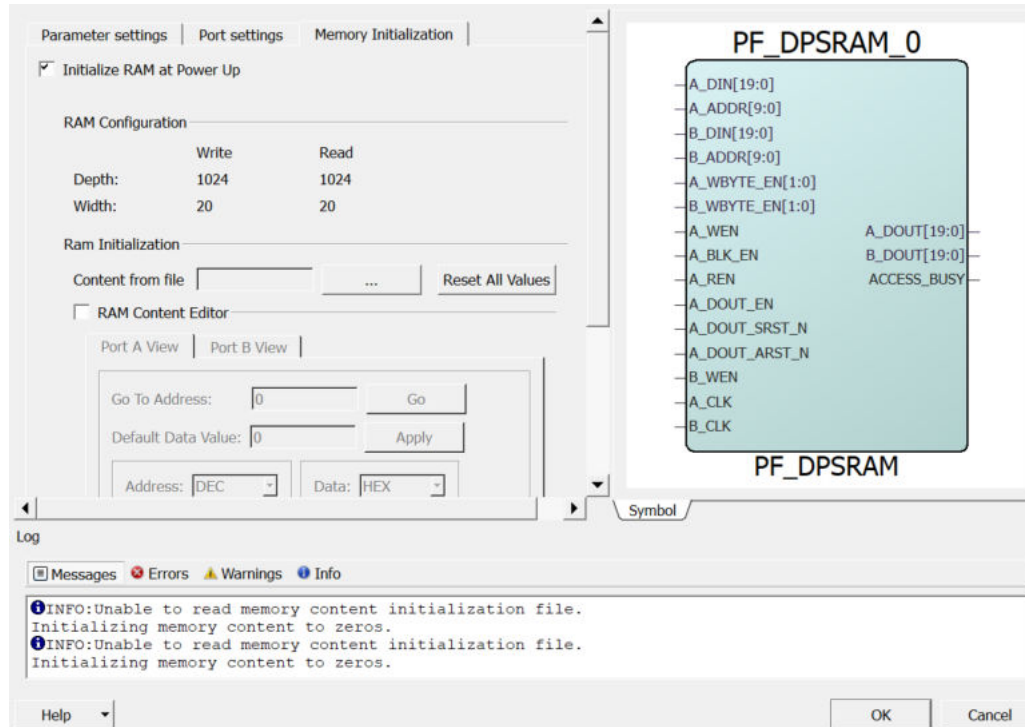
Select the required option from the following:

- **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, you can initialize RAM at power-up. LSRAM can be initialized during device power-up and functional simulation. The following figure shows the PF Dual Port Large SRAM IP memory initialization.

Figure 2-17. Dual-Port Large SRAM Configurator: Memory Initialization



Initialize RAM at Power Up

You 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 settings** 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 \$\mu\$ SRAM](#). The imported memory content is displayed in the RAM Content Editor.
- **Reset All Values**—Resets all the data values. This option is enabled when the **RAM Content Editor** check box is selected.

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 you to modify imported data.

Port A View/Port B View

- **Go To Address**—Enables you to go to a specific address in the editor. You can select the number display format (HEX, BIN, and DEC) from the Address menu.
- **Default Data Value**—You can set this with new data to change the default. When the data value is changed, all default values in the manager are updated to match the new value. You can select the number display format (HEX, BIN, and DEC) from the Data menu.

- **Address**—The Address column lists the address of a memory location. The menu specifies the number format of the address list (hexadecimal, binary, or decimal).
- **Data**—Controls the data format and data values in the manager.

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.

➔ Important: 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.

2.1.3.2.2. Two-Port LSRAM Configurator [\(Ask a Question\)](#)

The Two-Port SRAM (TPSRAM) IP configurator is available in the Libero SoC software under Memory & Controllers. The following figure 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 2-18. Two-Port Large SRAM Configurator with ECC Enabled

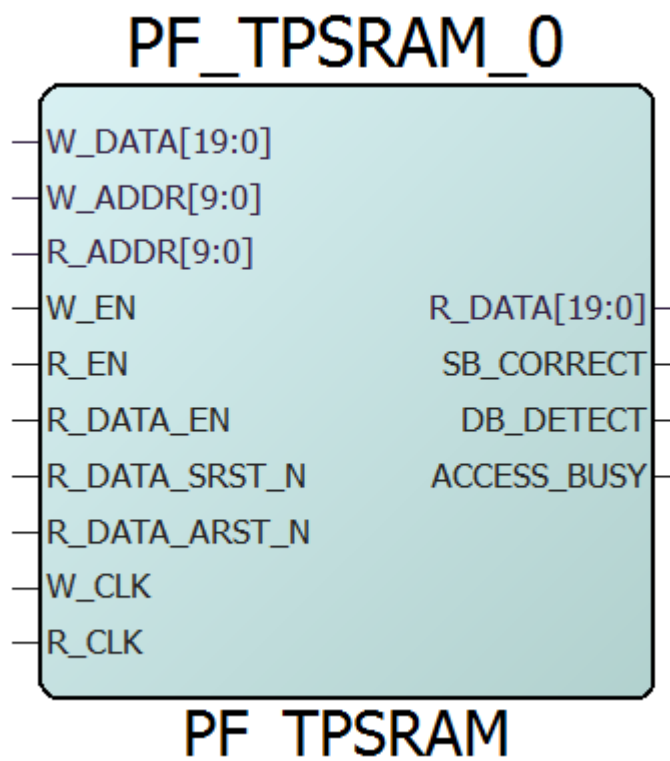


Table 2-11. Two-Port Large SRAM Configurator Signals

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 enabled.
DB_DETECT	Output	Active High	Double-bit detect flag. Applicable only when ECC is enabled.
ACCESS_BUSY	Output	Active High	Busy signal from SmartDebug. Exposed only when Expose ACCESS_BUSY output is checked.

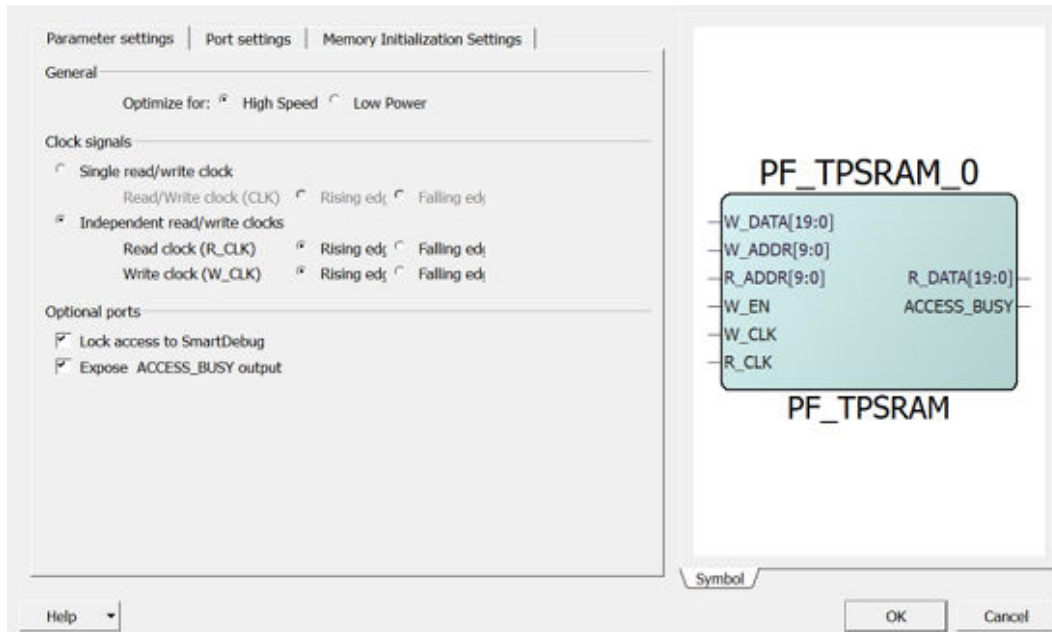
The Two-Port LSRAM configurator has three tabs:

- 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. The following figure shows the TPSRAM block configurator.

Figure 2-19. Two-Port Large SRAM Configurator: Parameter Settings



Optimization for High Speed or Low Power

You 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

You can set the clock signals and the signal polarity as follows:

- **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

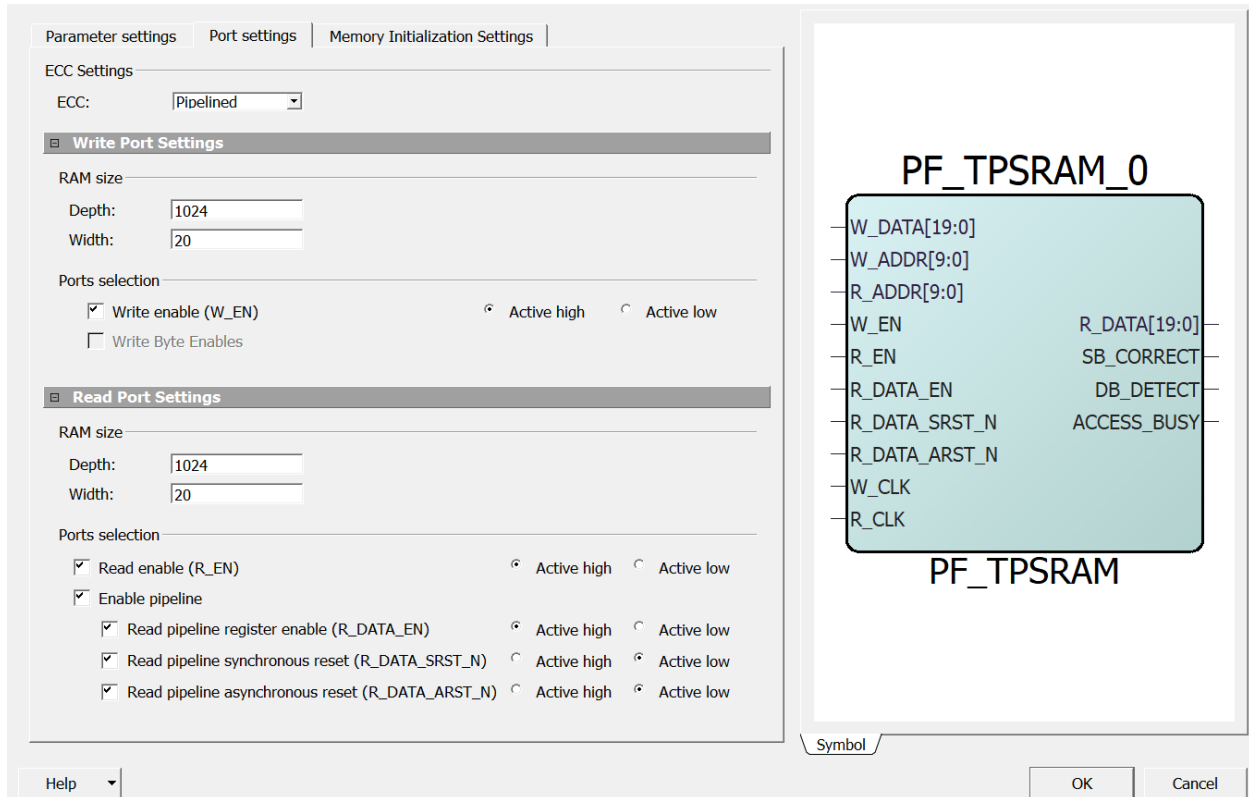
The user can select one of the following optional ports:

- **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. The following figure shows the TPSRAM block port settings.

Figure 2-20. Two-Port Large SRAM Configurator: Port Settings



ECC

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

- Disabled
- Pipelined
- Non-pipelined

➔ Important: 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.

The Write Byte Enable bits are all the control signals exposed by each column of LSRAM blocks when the implementation splits the word width. The LSRAM configurator generates the most efficient configuration of the depth and width for high-speed or low-power selection. Depending on the generated configuration, each Write Byte Enable bit may control up to 10 bits of data. In other words, the Write Byte Enable option does not govern the implementation.

In low-power mode for word widths that are multiples of 8, each Write Byte Enable bit will control 8 bits unless the word width is also a multiple of 10. 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 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

You can set the RAM size using the following options.

- **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.



Important: 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 is affected if width and depth are too large.

Write Enable (W_EN)—The default configuration for W_EN is checked (enabled). Clearing 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 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 check box enables R_EN and the associated functionality. Use **Active high** or **Active low** to change the signal polarity.



Important: You can insert the signal on the generated macro by checking its respective check boxes.

Enable Pipeline

Select the **Enable Pipeline** check box 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 check box 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 check box is disabled. Selecting this check box 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 check box is disabled. Selecting this check box 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 check box is disabled. Selecting this check box adds the signal to the top-level port.
- **Active high** or **Active low**—sets the signal polarity.



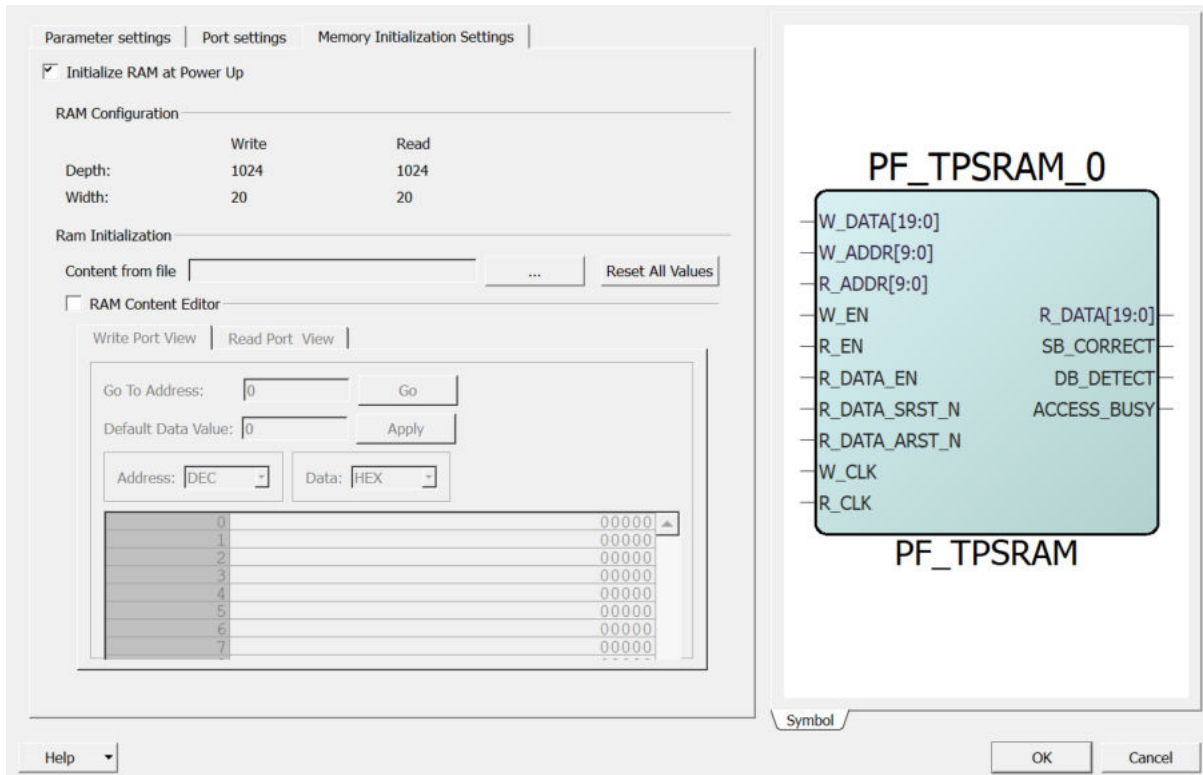
Important: You can insert the signal on the generated macro by checking the respective check boxes.

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 as described in **Memory Initialization at Power-**

Up of [Dual-Port Large SRAM Configurator](#). The following figure shows the TPSRAM IP memory initialization.

Figure 2-21. Two-Port Large SRAM Configurator: Memory Initialization



The **Reset All Values** option resets all the data values. This option is enabled when the **RAM Content Editor** check box is selected.

2.1.3.3. LSRAM Memory Macro [\(Ask a Question\)](#)

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](#).

2.1.3.4. Simulating ECC Errors in LSRAM [\(Ask a Question\)](#)

The simulation models for LSRAM support two signals. The SB_CORRECT signal to indicate a single-bit ECC error has occurred and the DB_DETECT signal to indicate that a double-bit, non-correctable error has occurred. These signals are asserted in conjunction with each other to pass this information.

Table 2-12. ECC Errors Flag

ECC Error Flag	SB_CORRECT	DB_DETECT
No Error	0	0
Single-bit Error Corrected	1	0
Double-bit Error	1	1

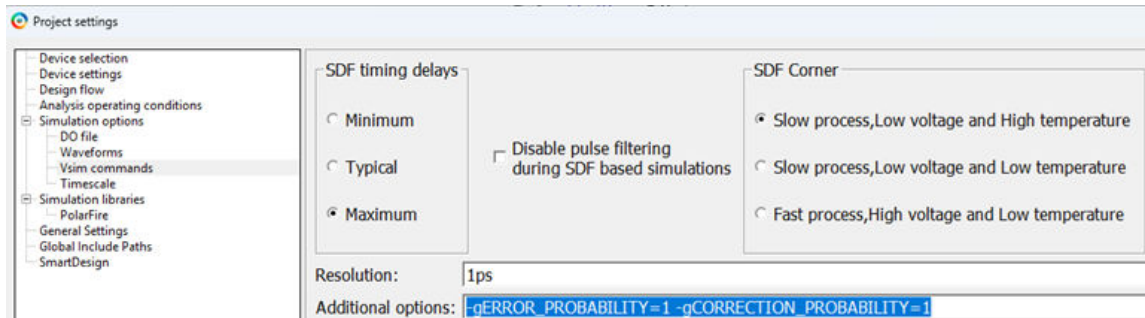
2.1.3.4.1. Probability-Based ECC Flag Assertion [\(Ask a Question\)](#)

With this feature, ECC flags are asserted depending on the probability parameter provided by you through vsim commands. The read data is unaffected by the flag behavior and is always corrected in the simulation regardless of the flag state.

The added logic also ensures that the flag outputs from the PF_TPSRAM block are always assigned a value and do not become unknown during periods when the read enable is not asserted high. For more information, see Libero SoC v11.9 SP2, v11.9 SP3, or Libero SoC v12.1 release notes.

Enable the simulation to report errors, the following options must be added to the `vsim` command. These are set on the command-line when entering the `vsim` command or through the Libero Project settings as shown in the following figure.

Figure 2-22. Libero Project Settings



-gERROR_PROBABILITY

- Legal values are $-0 \leq \text{value} \leq 1$.
- The simulation uses a random number generator and compares its value with the value assigned to `ERROR_PROBABILITY`.
- `SB_CORRECT asserted = "1"` indicates a single bit error has occurred.
- The assertion rate of `SB_CORRECT` is directly related to the value of `ERROR_PROBABILITY`. Increasing the value, results in `SB_CORRECT` being asserted more often and decreasing the value, results in `SB_CORRECT` being asserted less often.

-gCORRECTION_PROBABILITY

- Legal values are $-0 \leq \text{value} \leq 1$.
- The simulation uses a random number generator and compares its value with the value assigned to `DB_DETECT asserted = "1"` indicates more than one bit error has occurred and could not be corrected.
- `ERROR_CORRECTION`
- The assertion rate of `DB_DETECT` is inversely related to the value of `CORRECTION_PROBABILITY`. Increasing the value results in `DB_DETECT` being asserted less often and decreasing the value results in `DB_DETECT` being asserted more often.

2.1.3.4.2. Verilog Task-Based Flag Assertion [\(Ask a Question\)](#)

The LSRAM simulation model feature allows you to control the ECC error flags through your environment by using Verilog tasks. While using this feature, ensure that you have not activated the probability based ECC error flag simulation as described in [Probability-Based ECC Flag Assertion](#).

The Verilog task-based simulation can be performed on any of your projects without regenerating any of the cores. To use this feature, you must enter the address for which you want an ECC flag along with data mask bits, which decide to assert either `SB_CORRECT` or `DB_DETECT` flags in the `user_ecc_wrapper.v`, and include the `generated_ecc_wrapper.v` file in the active test bench.

To enable or disable the ECC error flags feature in the simulation model, execute the `ecc_err_inject_inst.enable_ecc_err_inject(1'b<user-input>)` task with the values described in the following table:

Table 2-13. Enable/Disable ECC Error Flag–User-Input Values

User-Input	Result
1	The ECC error flag feature is enabled.
0	The ECC error flag feature is disabled (default). Note: If this feature is disabled, the simulation model does not respond to other related tasks.

To control the ECC error flags for a single address, execute the `inject_single_addr_ecc_errors(<address>, <data_mask>)` task with the values described in the following table:

Table 2-14. Controlling ECC Error Flag–User-Input Values

Argument	Description
<code>address</code>	The user address to which the simulation model must drive the ECC error flags.
<code>data_mask</code>	The data mask bits that decide single or dual bit ECC flag.

During read operation, when the read address is equal to the `<address>` specified in this task, the following conditional results are achieved:

Table 2-15. Controlling ECC Error Flag–Results

Result...	if...
The <code>SB_CORRECT</code> error flag is asserted to 1 and read data remains unchanged.	The specified <code><data_mask></code> has only one non-zero bit.
The <code>DB_DETECT</code> and <code>SB_CORRECT</code> error flags are asserted to 1 and the read data bit is negated where <code><data_mask></code> bit is 1. Note: For some configurations, even though you provide the Data Mask bits > 1, it may still turn into SB flag only and not DB flag, as each of those Data mask bits belong to different physical instance block.	The specified <code><data_mask></code> has more than one non-zero bit.

2.1.3.4.3. Simulation Flow Example [\(Ask a Question\)](#)


The following is an example to simulate your design with the ECC error flag feature enabled.

1. Enter all ECC related Verilog tasks as per the Logical Address in the `user_ecc_wrapper.v` file in the following format.

```
sd_0.PF_TPSRAM_C0.ecc_err_inject_inst.enable_ecc_err_inject(1'b1);
sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(m'hLA1,n'hDM_LA1);
...
```

Table 2-16. ECC Include File Format

Format	Description
<code>sd_0</code>	Design instance name in the testbench
<code>PF_TPSRAM_C0</code>	Logical instance name of PolarFire Two-Port Large SRAM in the design
<code>m</code>	Size of the logical address
<code>LA1</code>	Logical address
<code>n</code>	Size of the data mask
<code>DM_LA1</code>	Data mask bits for the logical address

 **Important:** Ensure that you provide the accurate hierarchy up to the LSRAM instance before populating the `ecc_err_inject_inst.enable_ecc_err_inject` and `inject_single_addr_ecc_errors` tasks in the `user_ecc_wrapper.v` file.

The following is an example with the logical configuration of depth set to 4096, width set to 20 of PF_TPSRAM in the design sd_0 with logical instance PF_TPSRAM_C0.

```
initial
begin
  tb.sd_0.PF_TPSRAM_C0.enable_ecc_err_inject(1'b1);
  tb.sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(12'h100 ,20'h00002);
  tb.sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(12'h1a0 ,20'h00003);
  tb.sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(12'h1ff ,20'h00004);
end
```

2. Download the [PolarFire_ecc_include_file_gen.py](#) Python script file to your computer.
3. Extract the contents of the .zip file to any location on the computer.
4. Use any standard Python executor and change the directory to the location where the script is extracted and execute the `PolarFire_ecc_include_file_gen.py` python script with the following arguments.

```
python PolarFire_ecc_include_file_gen.py <libero_project_location> <user_ecc_wrapper.v>
```

Example 1: In the following example, the `user_ecc_wrapper.v` file and the script file are placed in the same location as that of the Libero project.

```
python PolarFire_ecc_include_file_gen.py . .
```

Example 2: In the following example, the `user_ecc_wrapper.v` file is located inside the stimulus folder under the Libero project folder, which is located in the path `D:/user/test_ecc`.

```
python PolarFire_ecc_include_file_gen.py D:/user/test_ecc D:/user/test_ecc/stimulus
```

After successful execution of the script, the `generated_ecc_wrapper.v` file is generated with the physical block instance name embedded in the hierarchical path and both Address and Data Mask breakdown, as shown in the following example:

```
initial
begin
  //tb.sd_0.PF_TPSRAM_C0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R0C0.u0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R1C0.u0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R2C0.u0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R3C0.u0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R4C0.u0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R5C0.u0.enable_ecc_err_inject(1'b1);

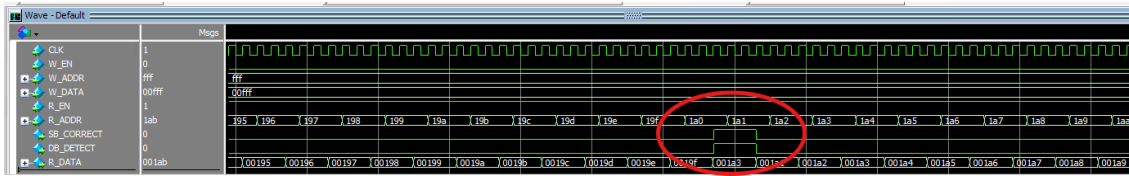
  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R6C0.u0.enable_ecc_err_inject(1'b1);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R7C0.u0.enable_ecc_err_inject(1'b1);
  //tb.sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(12'h100 ,20'h00002);

  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R0C0.u0.inject_single_addr_ecc_errors(9'h100,33'h2);
  //tb.sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(12'h1a0 ,20'h00003);

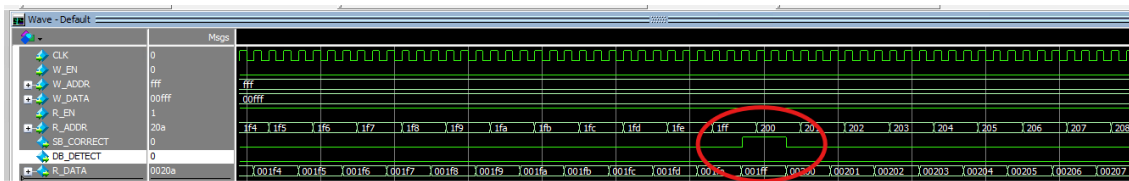
  tb.sd_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_0.PF_TPSRAM_C0_P0.PF_TPSRAM_C0_0_P0.PF_TPSRAM_R0C0.u0.inject_single_addr_ecc_errors(9'h1a0,33'h3);
  //tb.sd_0.PF_TPSRAM_C0.inject_single_addr_ecc_errors(12'h1ff ,20'h00004);
```


Figure 2-25. Assertion of SB_CORRECT and DB_DETECT ECC Flags as per user_ecc_wrapper.v File



In the following waveform, the highlight indicates that there is a SB_CORRECT flag at address 12'h1ff with no read data change.

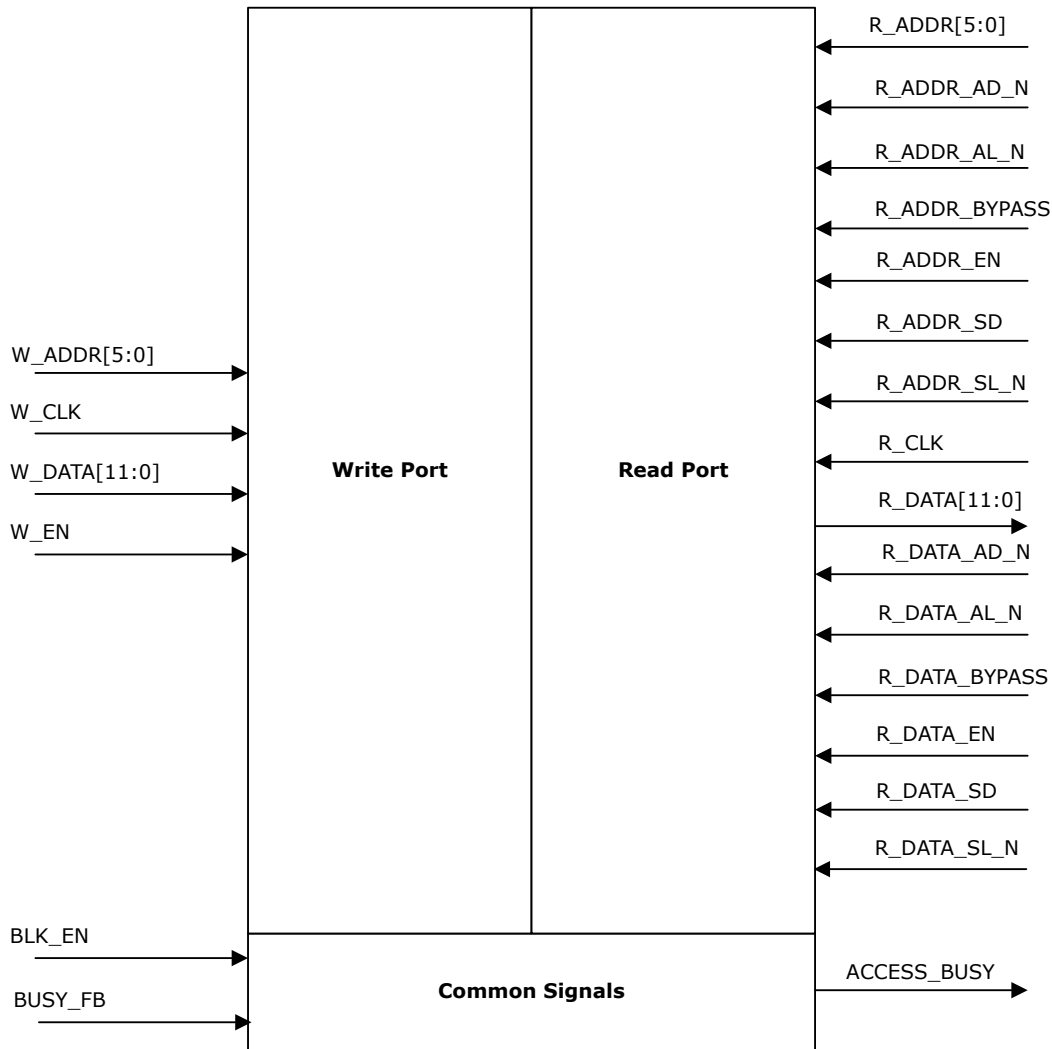
Figure 2-26. Assertion of ECC Flags as per user_ecc_wrapper.v File



2.2. μ SRAM [\(Ask a Question\)](#)

Each μ SRAM has one read port and one write port for Two-Port memory requirements. The following figure shows the μ SRAM I/O diagram.

Figure 2-27. μ SRAM Input/Output



The following table lists the ports available in μ SRAM.

Table 2-17. Port List for μ SRAM

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 High	Read-address and BLK_EN bypassed when High
R_ADDR_EN	Input	Dynamic	Active High	Read-address register Enable
R_ADDR_SL_N	Input	Dynamic	Active Low	Read-address register synchronous load

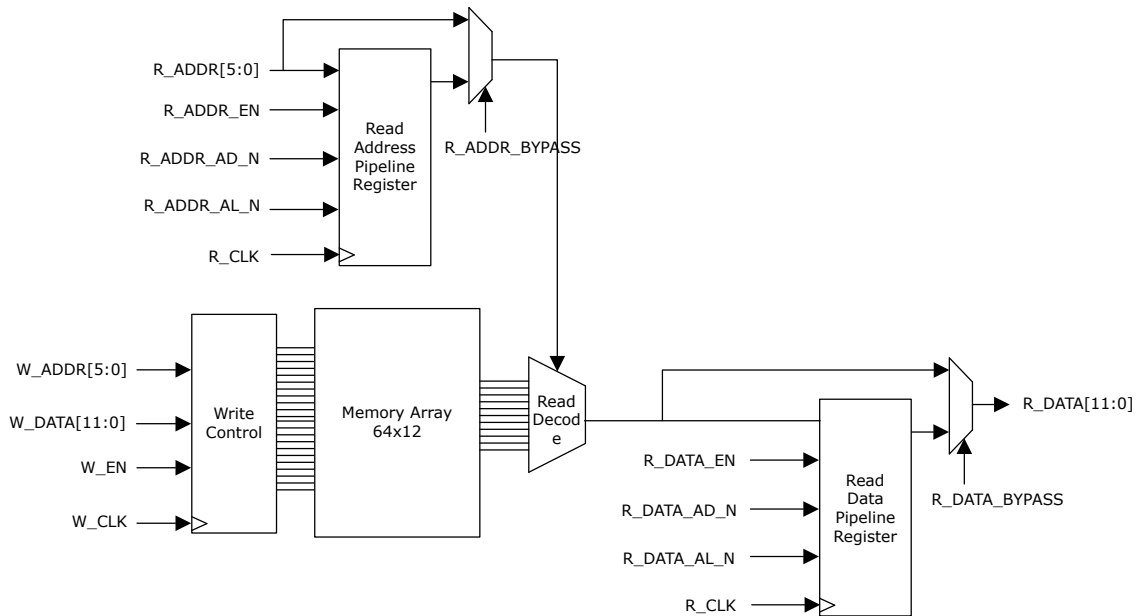
Pin Name	Direction	Type ¹	Polarity	Description
R_ADDR_SD	Input	Static	Active High	Read-address register synchronous load data
R_ADDR_AL_N	Input	Dynamic	Active Low	Read-address register asynchronous load
R_ADDR_AD_N	Input	Static	Active Low	Read-address register asynchronous load data
R_DATA[11:0]	Output	Dynamic	—	Read-data
R_DATA_BYPASS	Input	Static	Active High	Read-data pipeline register bypassed when High
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

Note:

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

The following figure shows the μ SRAM with independent write and read ports and read data pipeline registers.

Figure 2-28. Simplified Functional Block Diagram of μ SRAM



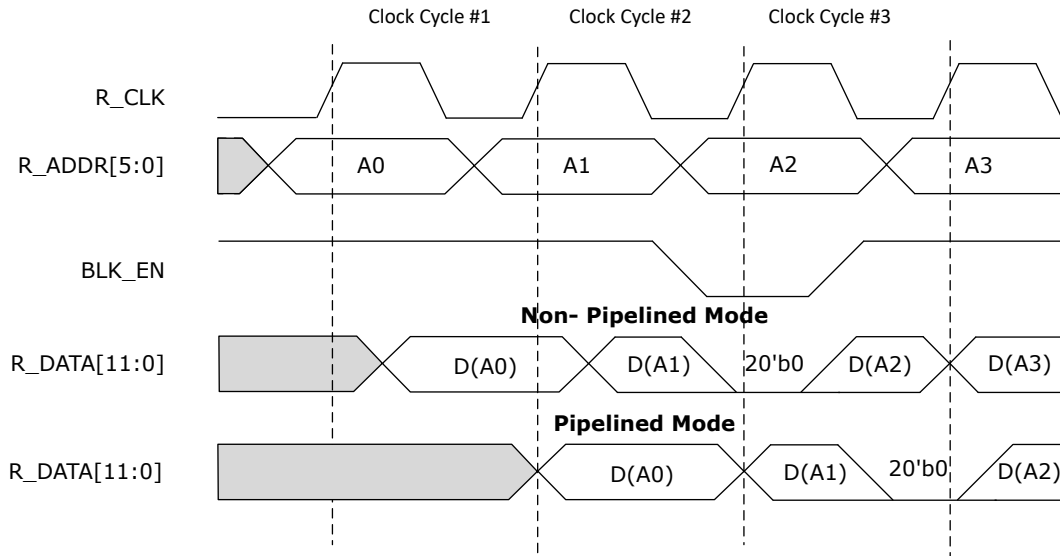
2.2.1. Read Operation [\(Ask a Question\)](#)

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.

The following figure shows the timing of read operation.

Figure 2-29. Read Operation in μ SRAM

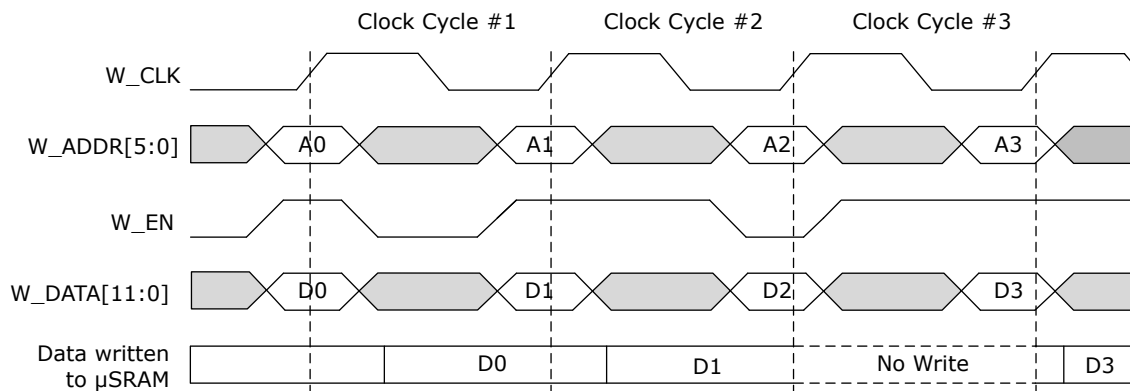


2.2.2. Write Operation [\(Ask a Question\)](#)

μ SRAM 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 2-30. Write Operation in μ SRAM



2.2.3. Collision [\(Ask a Question\)](#)

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 μ SRAM, 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.

2.2.4. Implementation [\(Ask a Question\)](#)

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

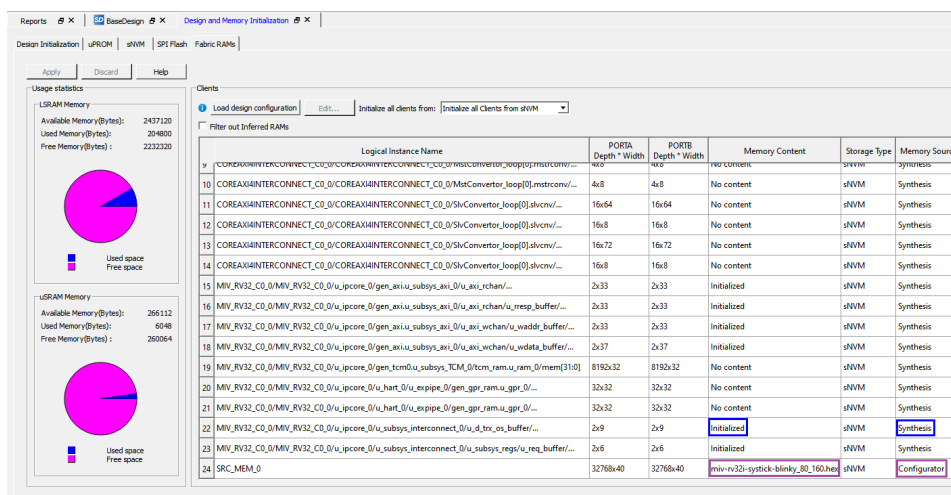
- [RTL Inference During Synthesis](#)
- [\$\mu\$ SRAM Configurator](#)
- [\$\mu\$ SRAM Memory Macro](#)

2.2.4.1. RTL Inference During Synthesis [\(Ask a Question\)](#)

Synplify Pro[®] can infer a μ SRAM from RTL automatically based on memory logic used in the design. In this case, Synthesis handles all the signal connections of the μ SRAM 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 infers fabric registers for the extra memory blocks. For more information about μ SRAM inference by Synplify Pro, see [Inferring PolarFire RAM Blocks Application Note](#).

Synplify Pro can also infer initialized RAM blocks, as well as ROMs, if the prescribed coding styles in the [Inferring PolarFire RAM Blocks Application Note](#) are followed. As noted in [Inferring PolarFire RAM Blocks Application Note](#) and [Appendix: SRAM Optimization for High-Speed Versus Low-Power](#), there are synthesis directives that can be used to confirm the RAM block cascading styles used. Ultimately, this impacts the way the user memory's data width is split across underlying PolarFire embedded memory block primitive macros and thus determines the user word bit mapping to individual data bits of the primitive macros. In Libero SoC design flow, after a synthesized netlist containing inferred initialized RAM blocks completes **Place and Route** and **Generate FPGA Array Data**, the **Configure Design Initialization Data and Memories** tool lists the Fabric RAMs by Logical Instance Name. Each inferred initialized instance is described as having Initialized Memory Content and lists the Memory Source as Synthesis instead of the μ SRAM configurator, as shown in the following figure.

Figure 2-31. Inferred Initialized RAM Vs Configurator



For more information about the Libero SoC **Design and Memory Initialization** tool, see the "How To Set Up Design and Memory Initialization" section in [PolarFire Family Power-Up and Resets User Guide](#).

When inferring RAM blocks during Synthesis, Synplify Pro might use RAM cascading styles and data word bit mapping to RAM block data bits that differ from the μ SRAM configurator in the Libero SoC IP Catalog. In the Libero SoC design flow, although the **Configure Design Initialization Data and Memories** step allows the user to modify the RAM initialization content for an inferred RAM block, this is not recommended because it can result in incorrect design functionality when the data word bit mapping to individual RAM block data bit pins differs from the μ SRAM core configurator's bit mapping. Instead, the initialization content for inferred RAMs must be provided along with the user HDL source files, such as using the `$readmemb`, `$readmemh`, or similar syntax, as shown in the code examples 61–67 for ROMs and 79–82 for initialized RAMs in [Inferring PolarFire RAM Blocks Application Note](#). These HDL examples are synthesizable for the PolarFire family of devices using the Synplify Pro Synthesis software. If the recommended syntax is followed, the fabric RAM blocks are inferred and the user supplied initialization content is forward annotated to the netlist. The **Configure Design Initialization Data and Memories** step, in the Libero SoC design flow, includes the **Fabric RAMs** tab that lists these inferred RAM instances with table entries indicating that the **Memory Content** is "Initialized" and the **Memory Source** is "Synthesis".

Updates to the initialization data for inferred RAMs must be treated as an update to RTL source files and must be re-synthesized. This approach also ensures that RTL functional simulation matches all post-synthesis simulation steps and the initial RAM values in the programmed device.

Updating the inferred RAM content in the **Configure Design Initialization Data and Memories** step only works as expected if the breakdown of the inferred RAM data bit mapping matches the μ SRAM core configurator's generated bit mapping.

2.2.4.2. μ SRAM Configurator [\(Ask a Question\)](#)

The μ SRAM configurator is available in the Libero SoC software under Memory & Controllers. [Figure 2-32](#) shows μ SRAM available in the Libero SoC software. The RAM configurator automatically cascades μ SRAM 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 are 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 are triggered at the rising edge of the clock.

Figure 2-32. μ SRAM Configurator: Generated Component

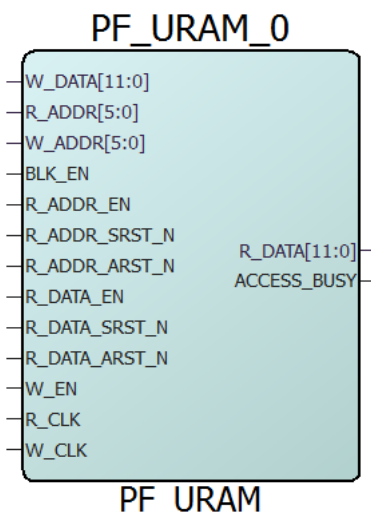


Table 2-18. μ SRAM Configurator Signals

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 μ SRAM configuration and defines how the signals are connected.

The μ SRAM configurator window has three tabs for settings:

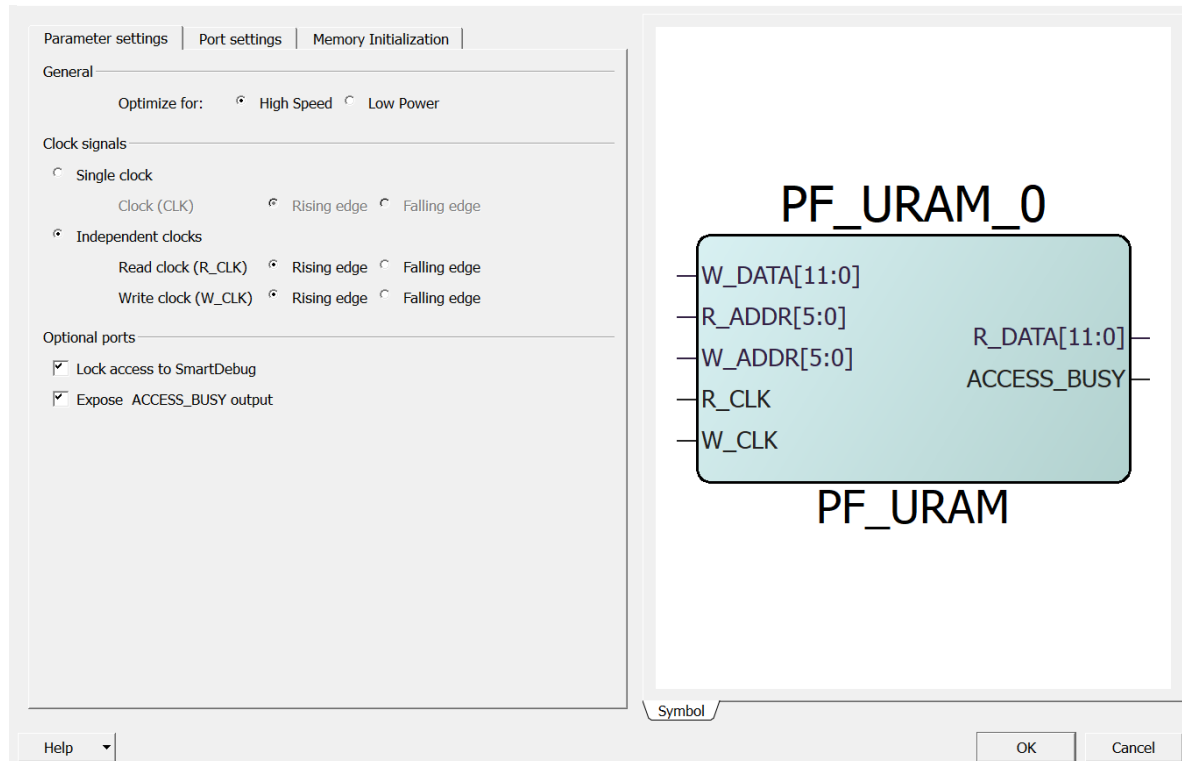
- Parameter settings
- Port settings
- Memory Initialization settings

2.2.4.2.1. Parameter Settings [\(Ask a Question\)](#)

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

The following figure shows the μ SRAM configurator.

Figure 2-33. MicroSRAM Configurator: Parameter Settings



Optimization for High Speed or Low Power

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

- **High Speed**—to optimizes the μ SRAM macro for speed and area by using width cascading.
- **Low Power**—to optimizes the μ SRAM 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 as follows:

- **Single clock**—drives both write and read ports with the same clock. This is the default configuration for μ SRAM.
- **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

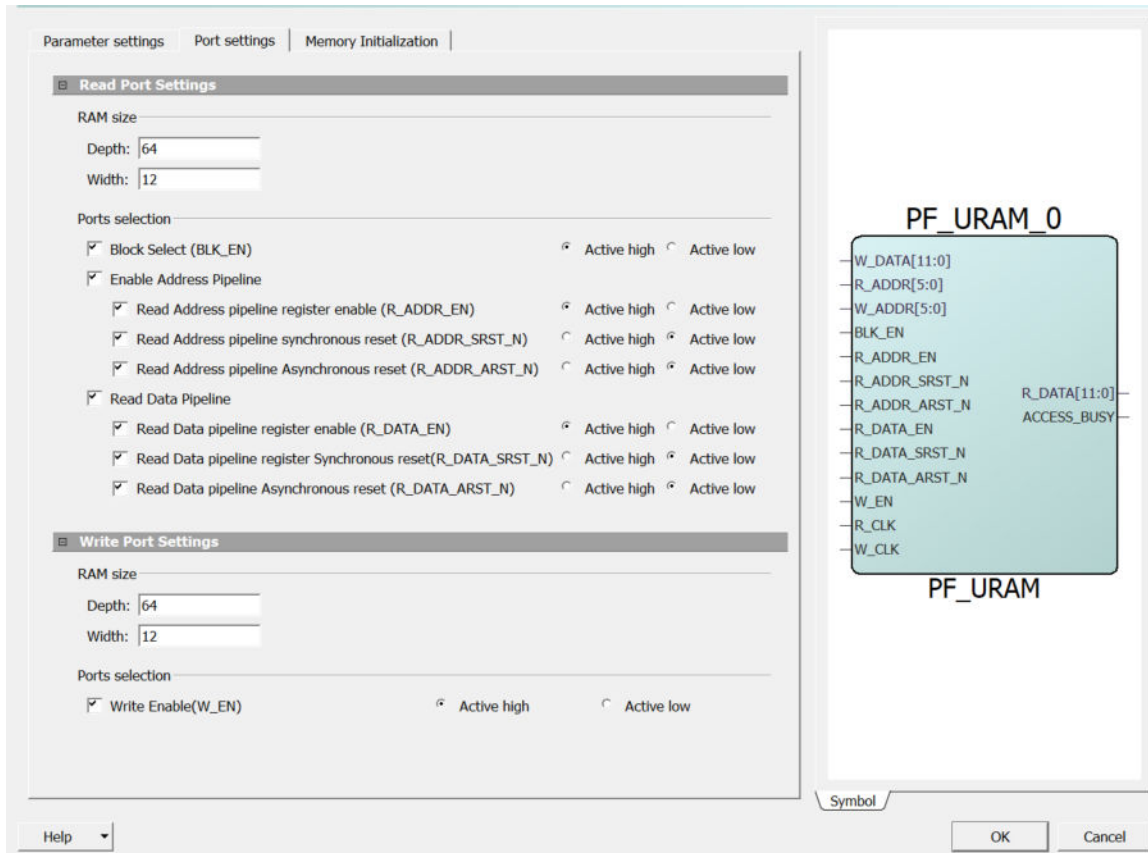
The user can select one of the following optional ports:

- **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.

2.2.4.2.2. Port Settings [\(Ask a Question\)](#)

In the Port settings tab, you can set RAM size, select ports, and set data output on write settings for both write and read ports. The following figure shows the μ SRAM IP block port settings.

Figure 2-34. MicroSRAM Configurator: Port Settings



RAM Size


The user can set the RAM size using the following options.

- **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.

➔ Important: 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 is 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.

 **Important:** Ports are 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](#). Select **Active high** or **Active low** to change the signal polarity.

 **Important:** You can insert the signal on the generated macro by checking the respective check-boxes.

Enable Address Pipeline

Check the **Enable Address Pipeline** check box 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 check box 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 check box is disabled. Selecting this check box 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 check box is disabled. Selecting this check box 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 check box is disabled. Selecting this check box adds the signal to the top-level port.
- **Active high or Active low:** changes the signal polarity.

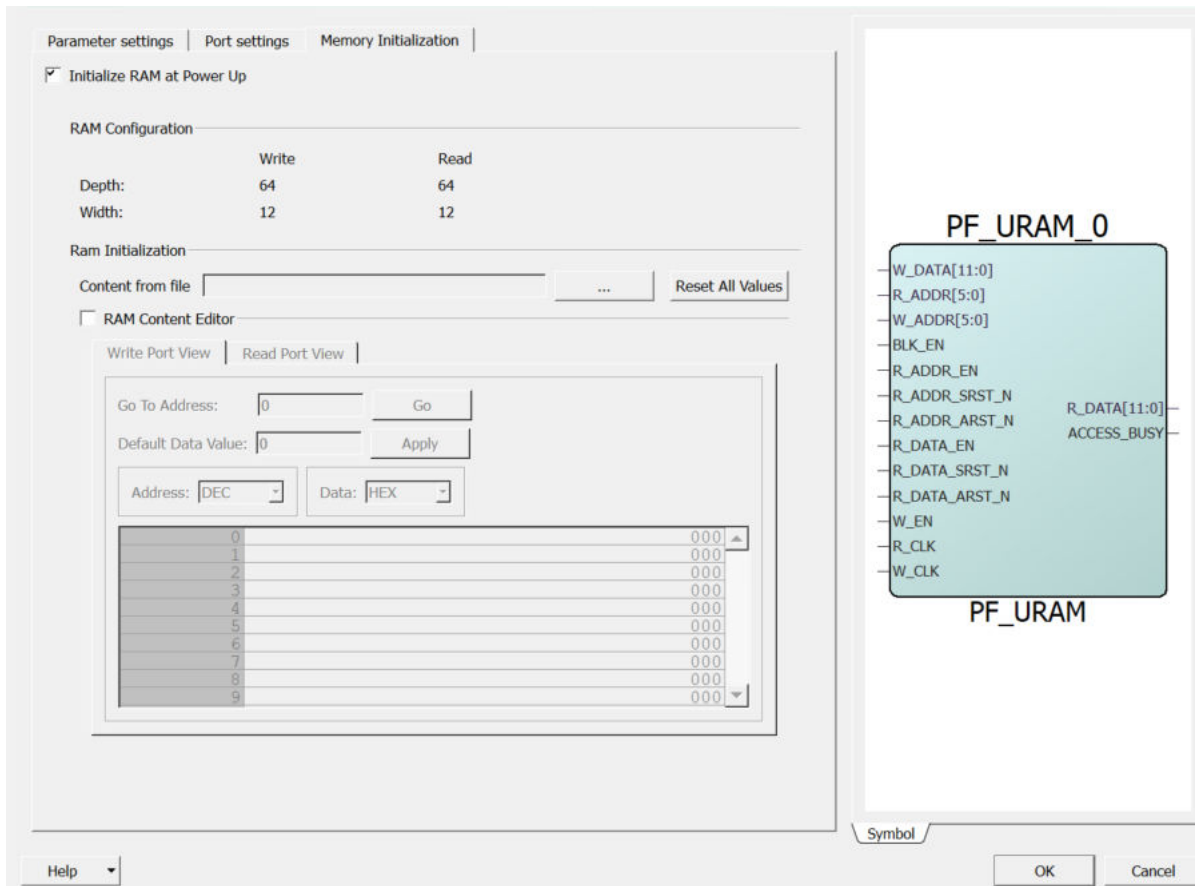
 **Important:** Ports are populated on the component by checking the respective check boxes.

Read Data Pipeline—This option is disabled by default. Select the Pipeline check box 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.

2.2.4.2.3. Memory Initialization at Power-Up [\(Ask a Question\)](#)

In the Memory Initialization tab, the user can initialize RAM at power-up. μ SRAM can be initialized during device power-up and functional simulation as described in **Memory Initialization at Power-Up** of [Dual-Port Large SRAM Configurator](#). The following figure shows the μ SRAM IP memory initialization.

Figure 2-35. MicroSRAM Configurator: Memory Initialization



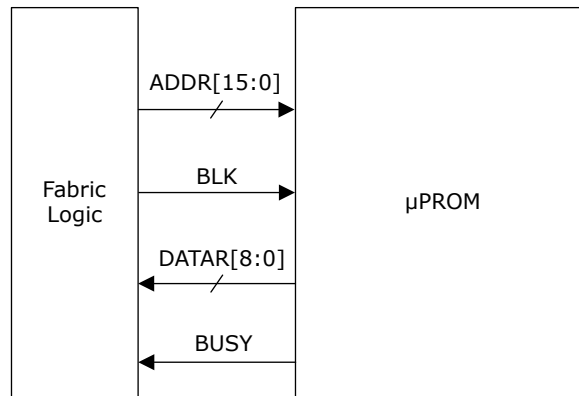
The **Reset All Values** option resets all the data values. This option is enabled when the **RAM Content Editor** check box is selected.

2.2.4.3. μ SRAM Memory Macro [\(Ask a Question\)](#)

A μ SRAM primitive is available as a component that can be used directly in the HDL file or instantiated in SmartDesign. For more information about configuring μ SRAM, see [\$\mu\$ SRAM Macro](#).

2.3. μ PROM [\(Ask a Question\)](#)

Both device families include a single User Programmable Read-Only Memory (μ PROM) row located at the bottom of the fabric providing up to 513 Kb of non-volatile, read-only memory. The address bus is 16-bit wide, and the read data bus is 9-bit wide. Fabric logic has read-only access to the entire μ PROM data. The following figure shows the high-level block diagram of μ PROM.

Figure 2-36. Simplified Functional Block Diagram of μ PROM

The following table lists the ports of μ PROM.

Table 2-19. μ PROM Port List

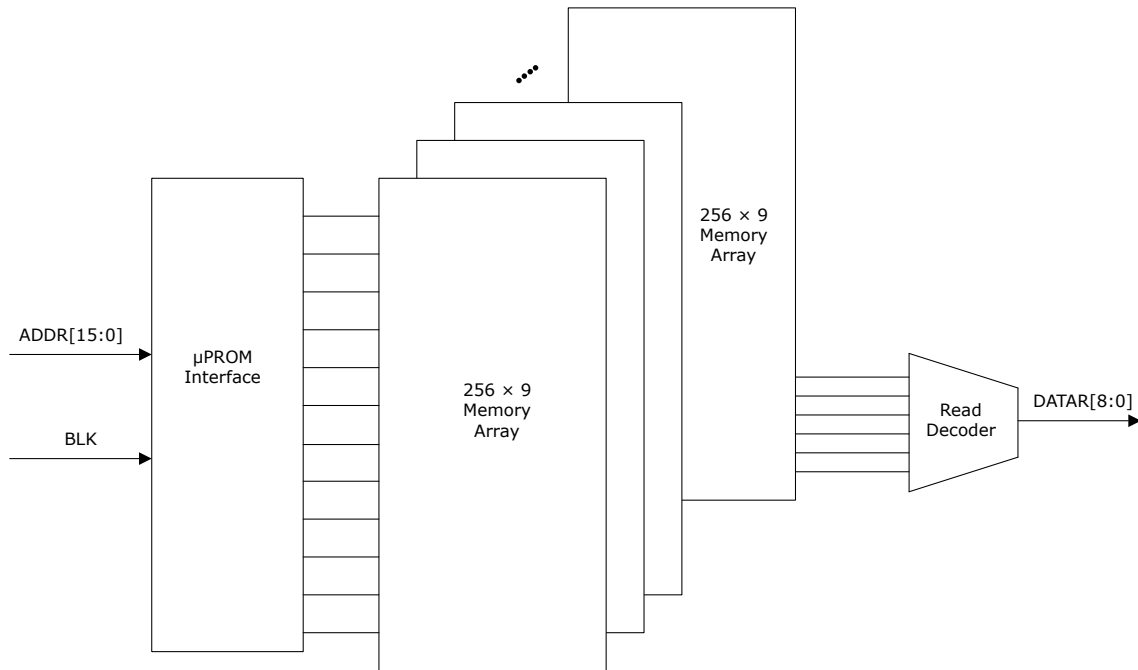
Port Name	Direction	Polarity	Description
ADDR[15:0]	Input	—	Address input
BLK	Input	Active-High	Block select
DATAR[8:0]	Output	—	Read data output
BUSY	Output	Active-High	Asserted when system controller or SmartDebug is accessing the μ PROM

2.3.1. μ PROM Architecture and Address Space [\(Ask a Question\)](#)

Architecturally, the μ PROM 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.

The following figure shows a simplified block diagram of the μ PROM memory.

Figure 2-37. μ PROM Memory Blocks

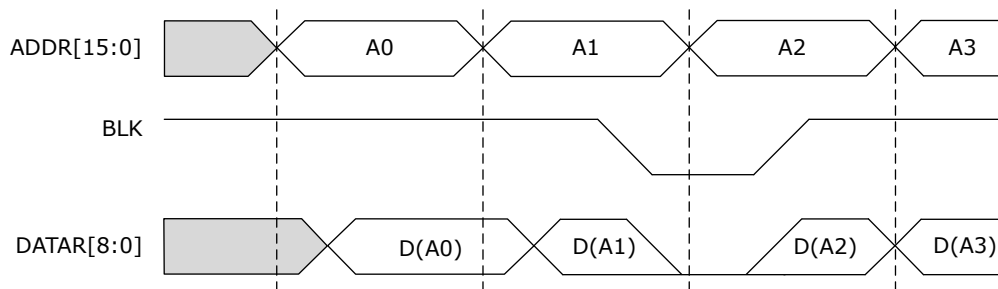


2.3.2. μ PROM Operation [\(Ask a Question\)](#)

In μ PROM, the write operation (program/erase) is performed during FPGA Programming. To configure the μ PROM, the Libero SoC μ PROM 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 μ PROM during FPGA programming. Read operations are performed only through the fabric interface. All μ PROM read operations are asynchronous. To perform a synchronous read operation, the μ PROM output needs to be pipelined using fabric registers.

The following figure shows the read timing diagram for the μ PROM.

Figure 2-38. Read Operation in μ PROM



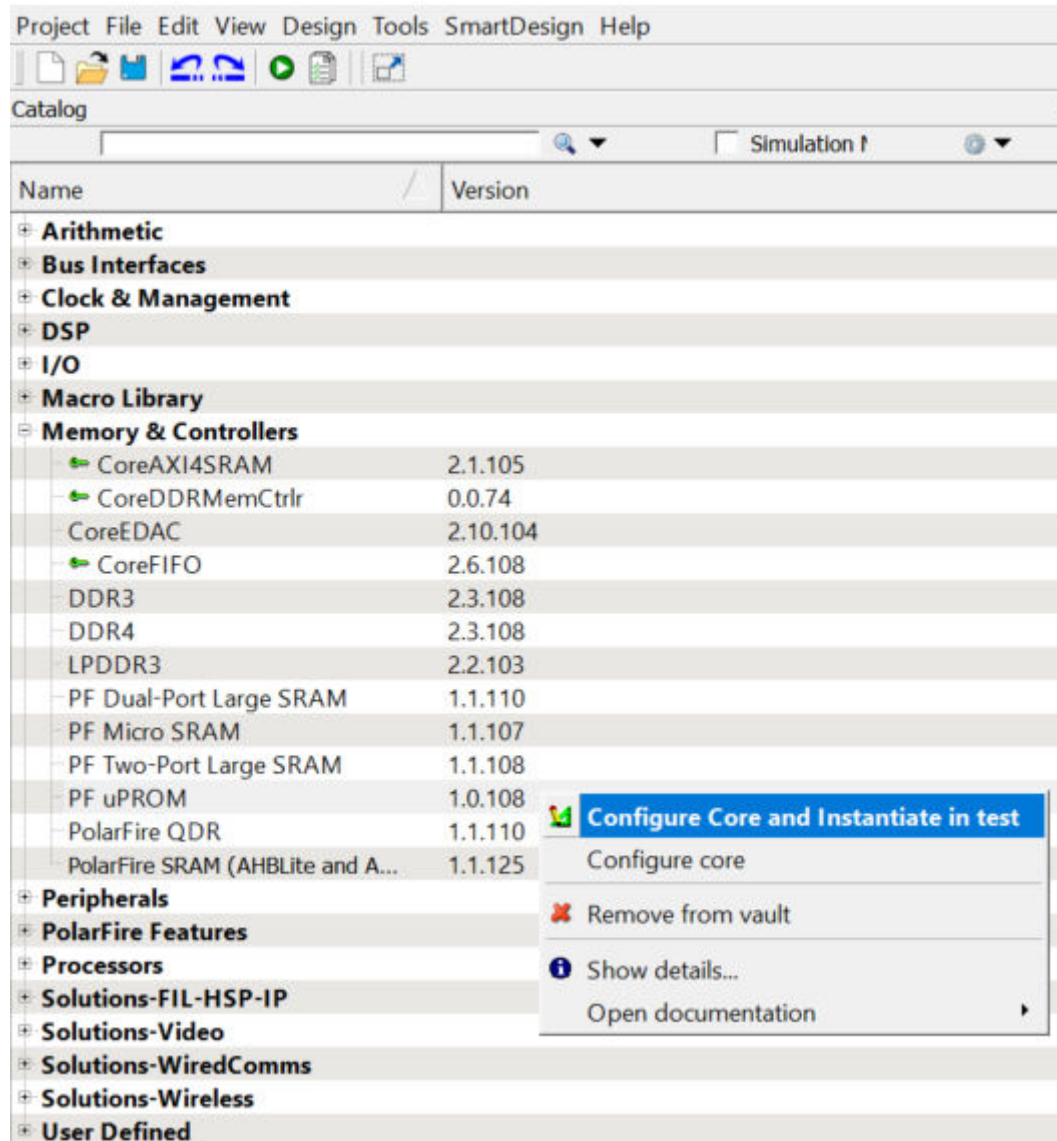
2.3.3. Implementation [\(Ask a Question\)](#)

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

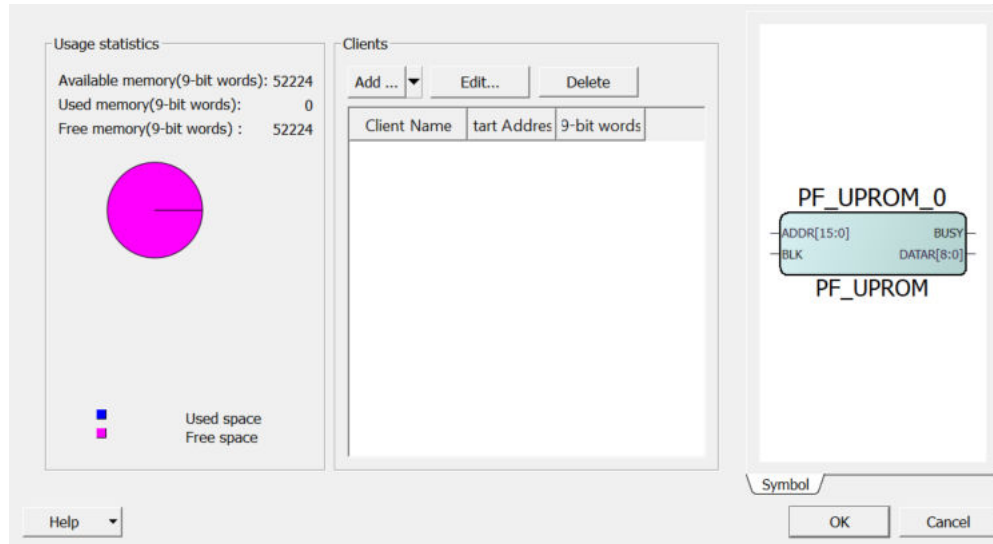
1. Expand **Memory & Controllers** in the **Catalog**.

2. Do one of the following to invoke the μ PROM Configurator:
 - Double click or right click PF μ PROM and select **Instantiate in *design_name*** to instantiate the μ PROM in the SmartDesign canvas.
 - Double click or right click PF μ PROM and select **Configure Core**. Enter a component name for the μ PROM when prompted.

Figure 2-39. μ PROM Core in Catalog



3. In the μ PROM Configurator, click **Add Clients to System** to add a client to the μ PROM. The following figure shows the μ PROM Configurator. Only the client is programmed and not all the content is erased during programming.

Figure 2-40. μ PROM Configurator

2.3.3.1. Usage Statistics [\(Ask a Question\)](#)

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

2.3.3.1.1. Available Memory [\(Ask a Question\)](#)

The μ PROM can hold up to 58,368 9-bit words (total 525312 bits), depends on the die. For more information, see [Table 2](#).

2.3.3.1.2. Used Memory [\(Ask a Question\)](#)

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.

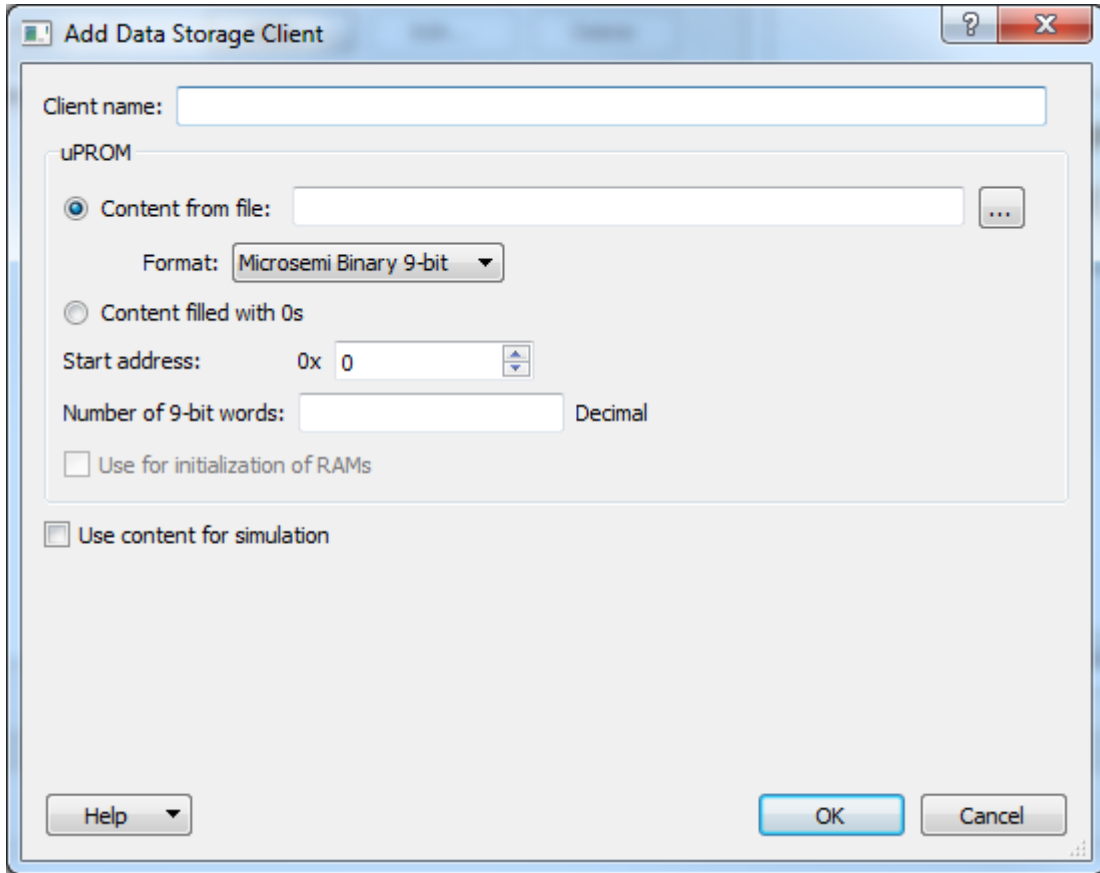
2.3.3.1.3. Free Memory [\(Ask a Question\)](#)

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

2.3.3.2. Add Clients to System [\(Ask a Question\)](#)

1. Click **Add Clients to System** to open the Add Data Storage Client dialog box ([Figure 2-41](#)).
2. Specify the start address, client size, the content of the client, and whether to use the memory content for simulation or not.

Figure 2-41. Add Data Storage Client Dialog Box



2.3.3.2.1. Client Name [\(Ask a Question\)](#)

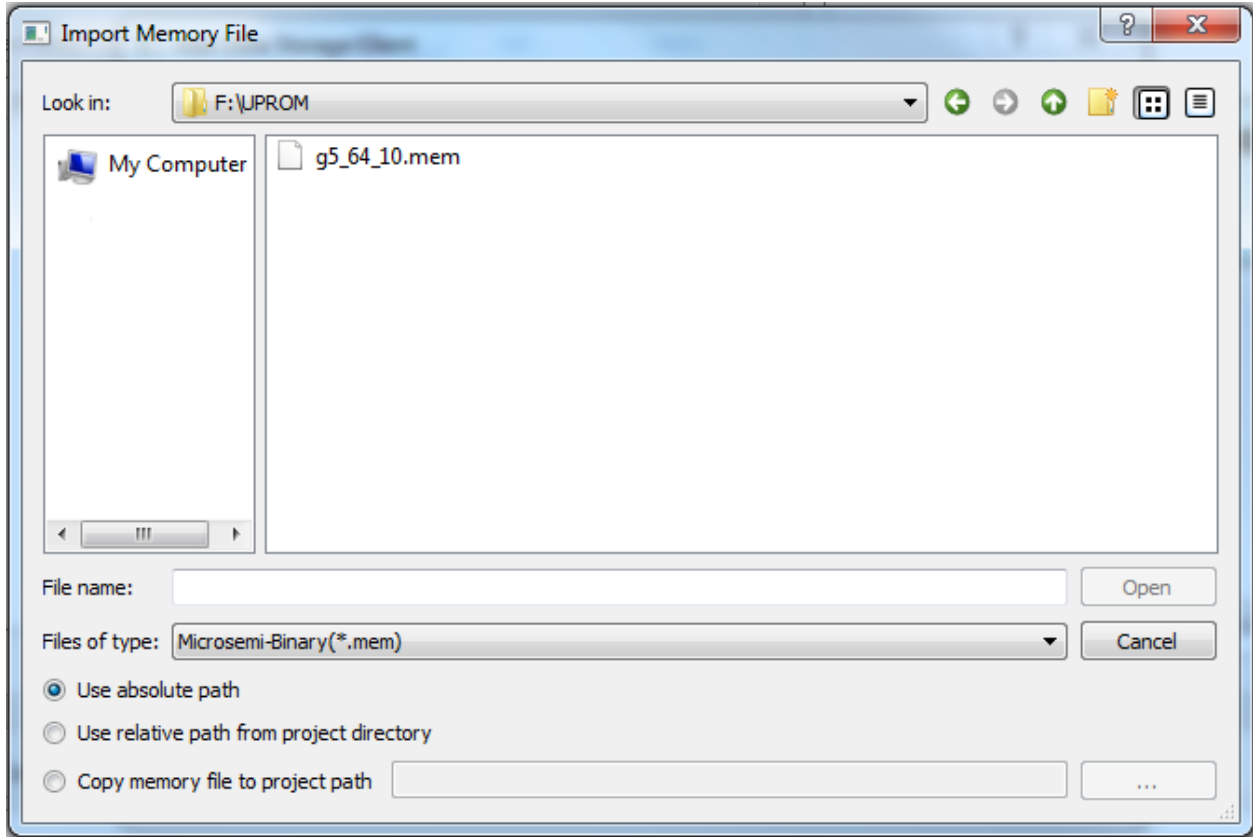
Enter a name for your memory client.

2.3.3.2.2. Content from File [\(Ask a Question\)](#)

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: In this example, * .mem file is selected.

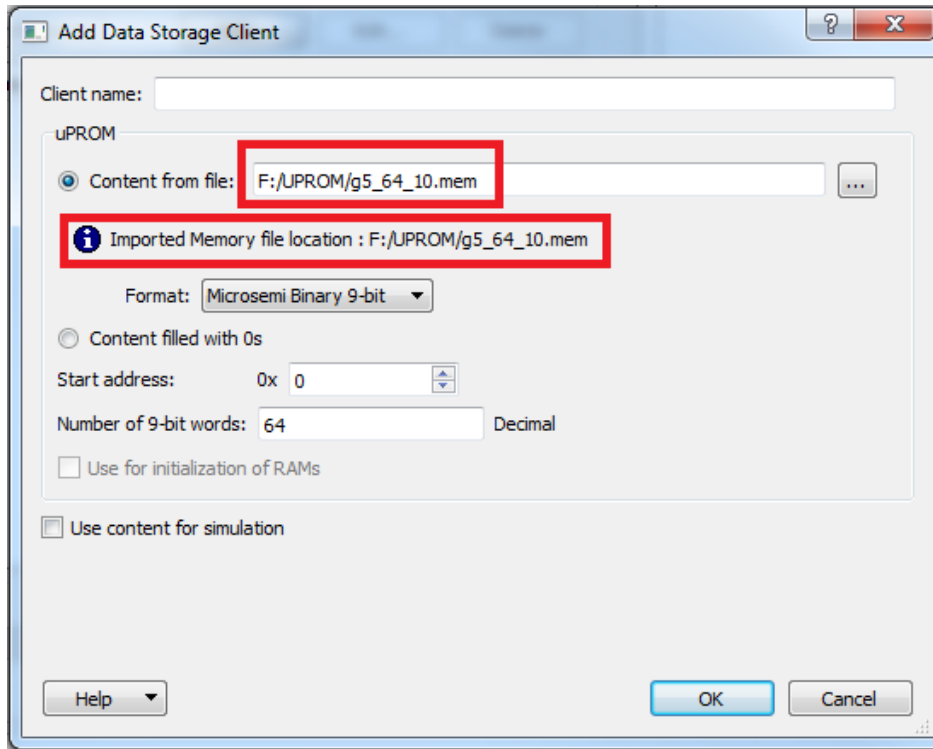
Figure 2-42. Import Memory File Dialog Box



Use Absolute Path

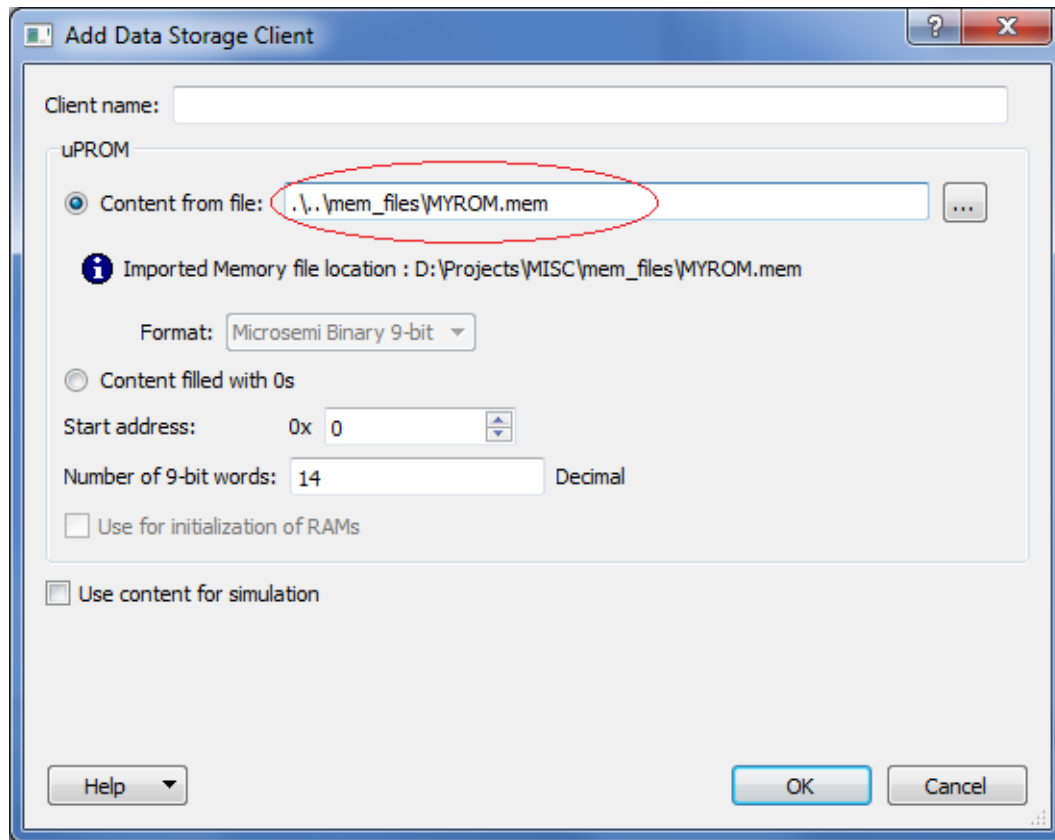
When this is selected, the absolute path of the memory file appears in the **Content from File** field.

Figure 2-43. Absolute Path of Memory File



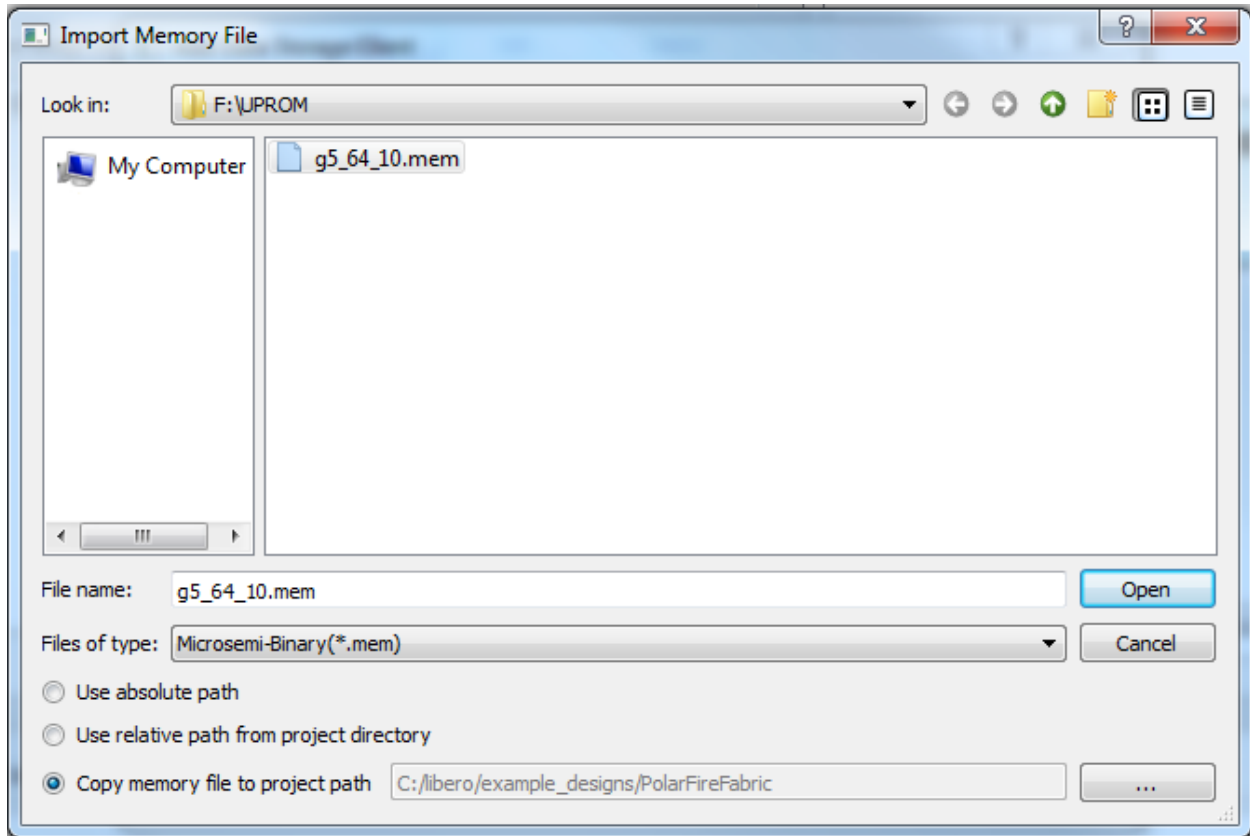
Use Relative Path from Project Directory

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 2-44. Relative Path of Memory File

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 2-45. Location of Memory File to Copy From

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 Intel-Hex (*.hex, *.ihx), Motorola-S (*.s), Simple-Hex (*.shx), and Microsemi-Binary (*.mem) memory file formats. For more information about the supported memory file formats, see [Appendix: Supported Memory File Formats for LSRAM and µSRAM](#).

In this example, *.mem is used. 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. The following figure shows an example memory file.

Figure 2-46. Microsemi Binary File (*.mem) Example

```

101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010
101010101
010101010

```

2.3.3.2.3. Content filled with 0s [\(Ask a Question\)](#)

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 μ PROM Memory Content. For more information, see [Update \$\mu\$ PROM Memory Content](#).

2.3.3.2.4. Start Address [\(Ask a Question\)](#)

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

2.3.3.2.5. Number of 9-bit Words [\(Ask a Question\)](#)

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

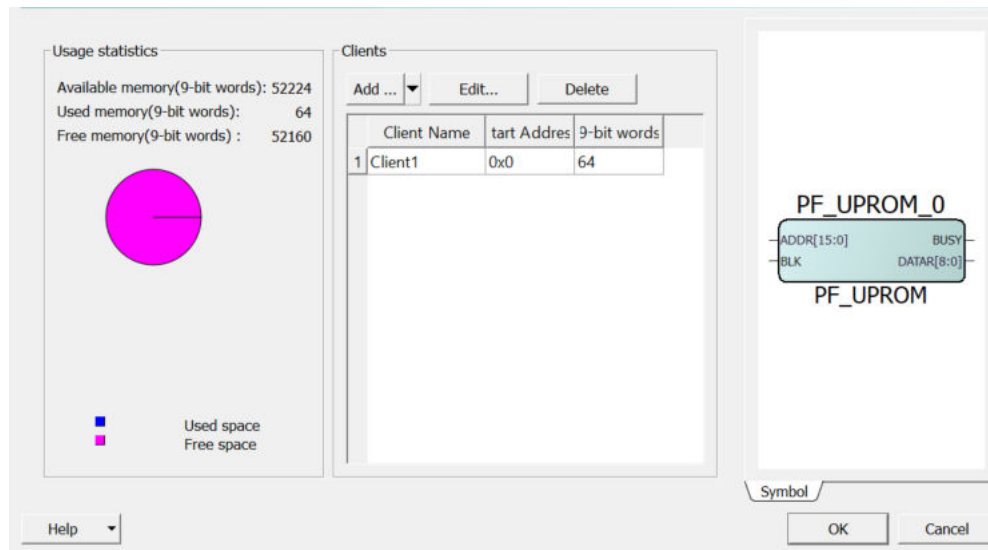
Note: When multiple clients are added, ensure that the address range of each client does not overlap with the other clients. Overlapping of address range is not allowed and is flagged as an error when it occurs.

2.3.3.2.6. Use Content for Simulation [\(Ask a Question\)](#)

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 μ PROM simulation model to initialize the μ PROM 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.

The following figure shows the added clients under **User clients in μ PROM**.

Figure 2-47. User Clients



2.3.3.3. DRC Rules and Error Messages [\(Ask a Question\)](#)

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

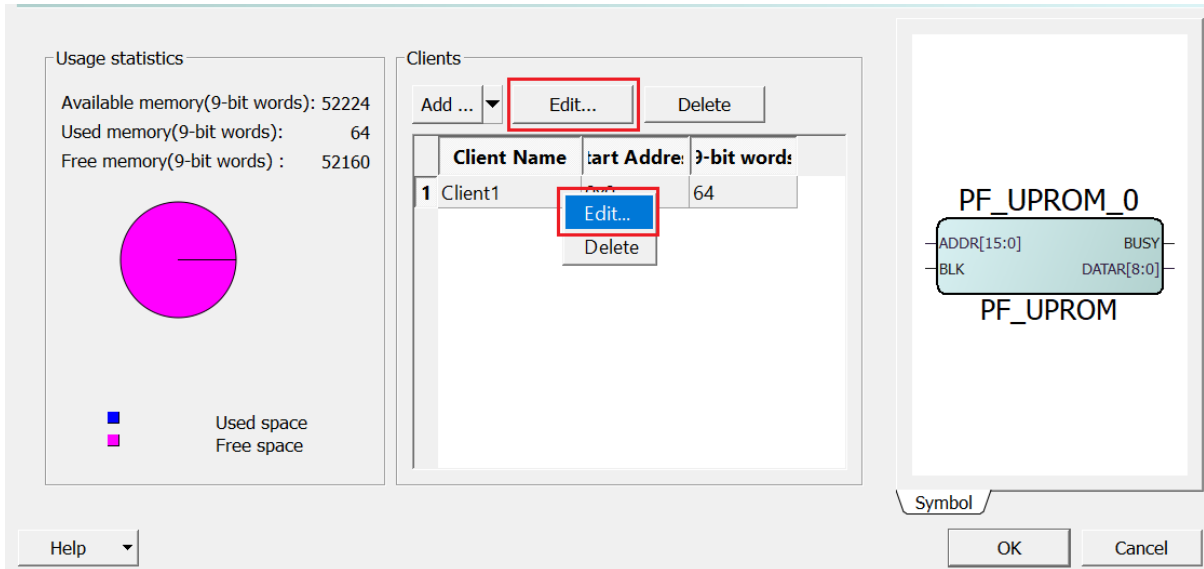
- An invalid start address (outside of the μ PROM memory space) is entered.
DRC Error: The specified start address is invalid; legal addresses range from $0x0$ to $\langle \text{max_possible_address_for_the_die} \rangle$.
- The start address and the number of words entered put the user client beyond the memory space of the μ PROM.
DRC Error: For the specified start address, the number of words cannot exceed the total number of words of $\langle \text{max_possible_words_for_die} \rangle$.
- 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 $\langle \text{mem_file_word_count} \rangle$ specified in the memory file $\langle \text{mem_file_name} \rangle$.
- 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: $\langle \text{client name} \rangle$.
- The memory file ($*.mem$) size exceeds the total μ PROM memory space.
DRC Error: The memory file $\langle \text{memoryFileName} \rangle$ size exceeds the total μ PROM space.

2.3.3.4. Editing a Client [\(Ask a Question\)](#)

To edit a client:

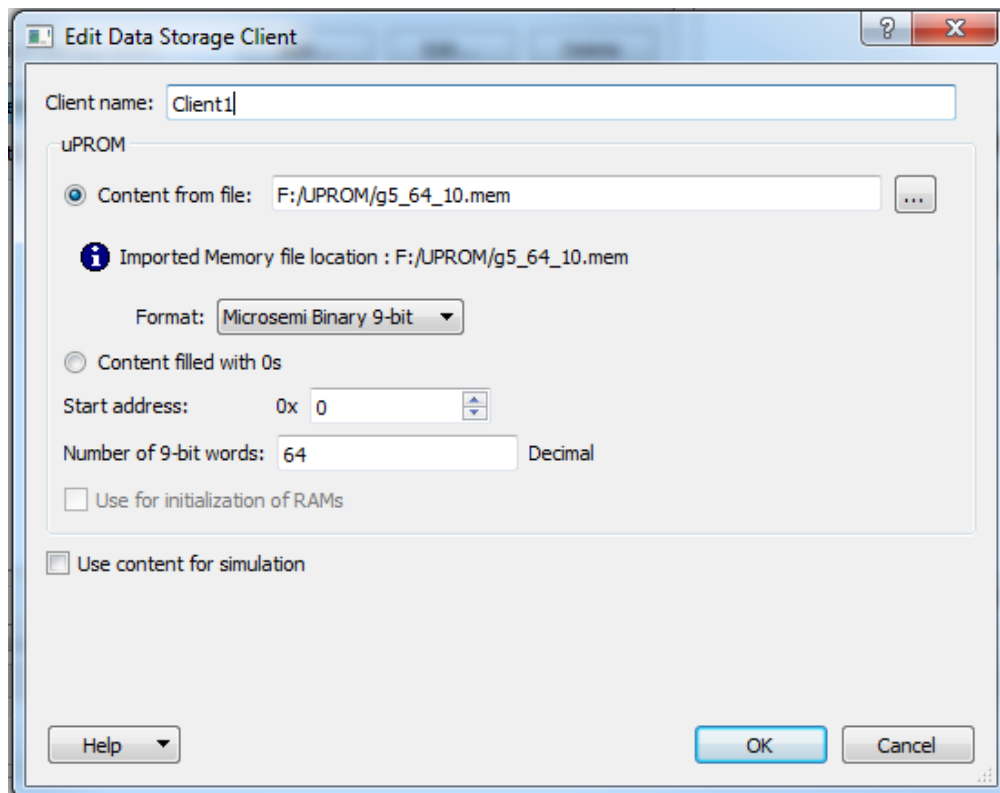
1. Click **Edit** or right-click the client name and select **Edit** to open the **Edit Data Storage Client** dialog box.

Figure 2-48. Editing User Clients



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

Figure 2-49. Edit Data Storage Client Dialog Box



2.3.3.5. Deleting a Client [\(Ask a Question\)](#)

Right-click the client and select Delete.

Figure 2-50. Deleting a Client

The screenshot displays the Embedded Memory Blocks configuration interface. On the left, the 'Usage statistics' panel shows:

- Available memory(9-bit words): 52224
- Used memory(9-bit words): 6320
- Free memory(9-bit words): 45904

A pie chart below the statistics shows a small blue slice representing 'Used space' and a large pink slice representing 'Free space'. A legend at the bottom left identifies the colors: a blue square for 'Used space' and a pink square for 'Free space'.

On the right, the 'Clients' panel features a table with the following data:

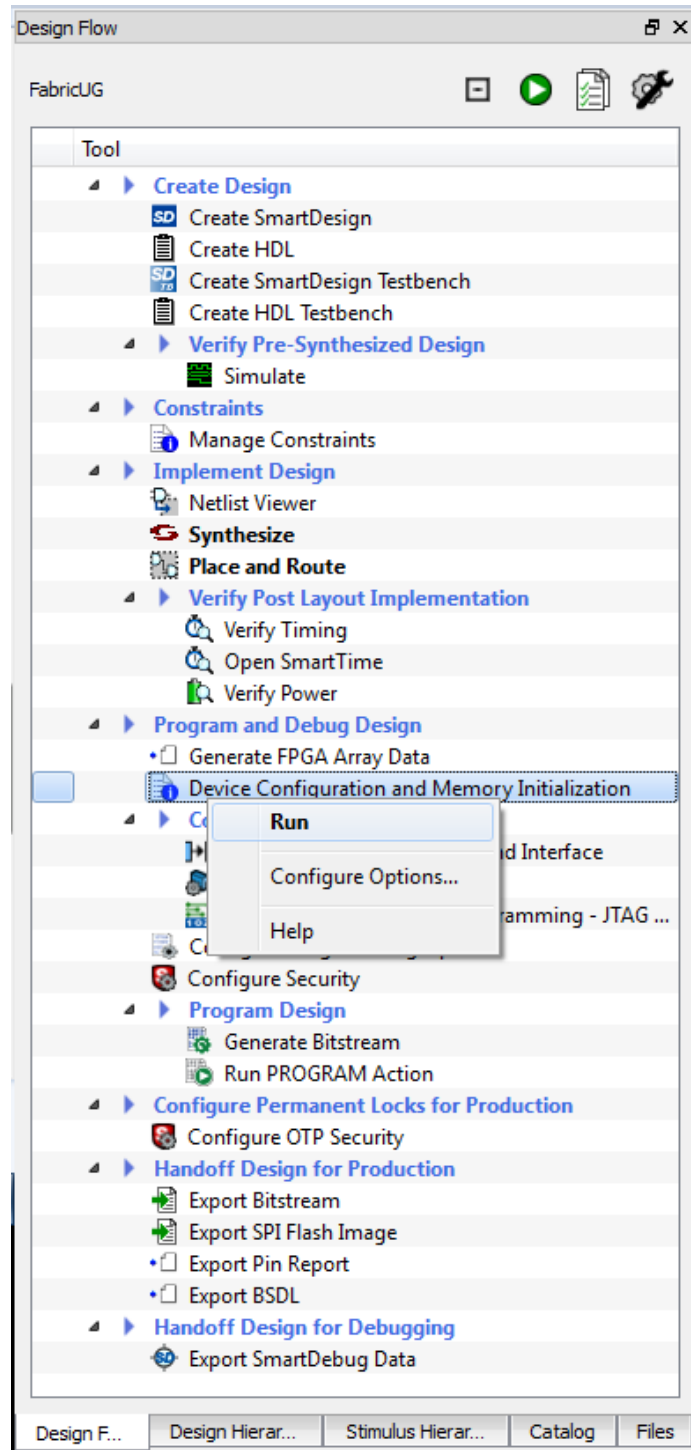
	Client Name	Start Address	9-bit words
1	client_1	0x0	6320

Buttons for 'Add ...', 'Edit...', and 'Delete' are located at the top of the Clients panel. The 'Delete' button is highlighted with a red box. A context menu is open over the first row of the table, with 'Edit...' and 'Delete' options. The 'Delete' option in the context menu is also highlighted with a red box.

2.3.3.6. Update μ PROM Memory Content [\(Ask a Question\)](#)

The μ PROM 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 μ PROM memory content, see [PolarFire Family Power-Up and Resets User Guide](#).

Figure 2-51. Update μ PROM Memory Content



2.4. sNVM [\(Ask a Question\)](#)

The PolarFire family includes 56 KB 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 Physically Unclonable Function (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.

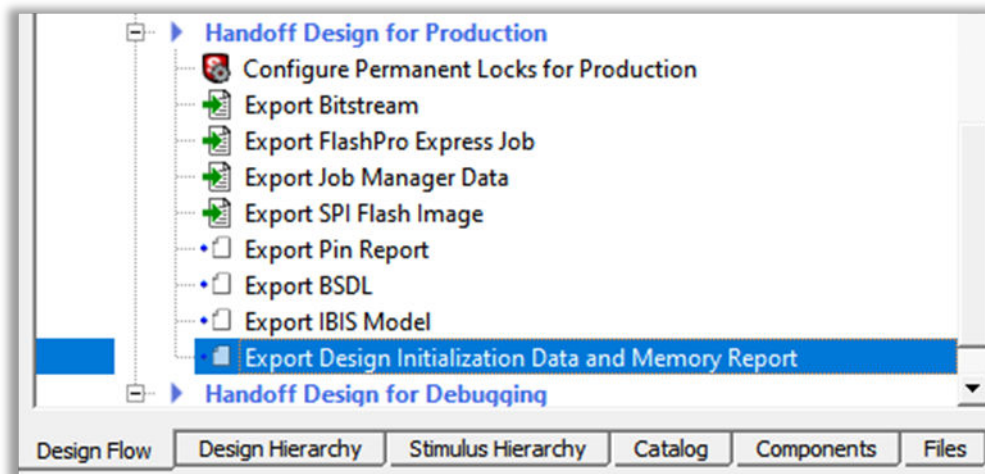
2.4.1. Implementation [\(Ask a Question\)](#)

The sNVM is not accessible to the fabric logic. You can access sNVM through CoreSysService IP using system service calls. The sNVM content is used for device initialization for LSRAM, μ SRAM, PCIe®, and transceiver data. Few pages of available 56 KB are used for storing the device and peripheral configuration data and the remaining pages store the user data. For more information about device initialization using sNVM, see [PolarFire Family Power-Up and Resets User Guide](#).

4136 bytes of sNVM storage space is required to initialize a single LSRAM block (1024 x 20 bits) with custom data or all zeros. 168 bytes of sNVM storage space is required to initialize a single Micro SRAM block (64 x 12 bits) with custom data or all zeros. Each page of sNVM can store 252 bytes. So, to store 168 bytes of a single μ SRAM block, you need one sNVM page.

The selected logical width/depth determines the number of LSRAM or Micro SRAM blocks instantiated. Therefore, the number of fabric RAM blocks, the user chooses to initialize during Power-Up, determines how much non-volatile storage space (such as sNVM) is required for the system controller operation, and how long the device initialization process takes to complete. To obtain a design specific summary of the sNVM usage and Power-Up to Functional Time (PUFT), the user must run the Libero SoC Design Flow Production Handoff step called **Export Design Initialization Data and Memory Report**, as shown in the following figure.

Figure 2-52. Export Design Initialization Data and Memory Report



2.5. eNVM (PolarFire SoC and RT PolarFire SoC Only) [\(Ask a Question\)](#)

PolarFire SoC and RT PolarFire SoC devices include 128 Kbytes of eNVM within the MSS. eNVM is used for storing the first stage bootloader during the device programming. At device power-up, the E51 monitor core boots from the eNVM and holds the U54 processors in reset. The E51 core boots the U54 cores based on the boot addresses specified in the first stage bootloader. For more information about device initialization using eNVM, see [PolarFire Family Power-Up and Resets User Guide](#). For more information about booting and configuration, see [PolarFire SoC Software Development and Tool Flow User Guide](#).

3. Math Blocks [\(Ask a Question\)](#)

In the PolarFire family, 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.

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, see [Inferring PolarFire MACC Blocks Application Note](#). Implementation of these arithmetic functions using math blocks 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, μ SRAM, and μ PROM) to implement complex DSP algorithms.

3.1. Features [\(Ask a Question\)](#)

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.

3.2. Math Block Resources [\(Ask a Question\)](#)

[Table 2](#) and [Table 3](#) lists the number of Math blocks available in both the device families.

3.3. Functional Description [\(Ask a Question\)](#)

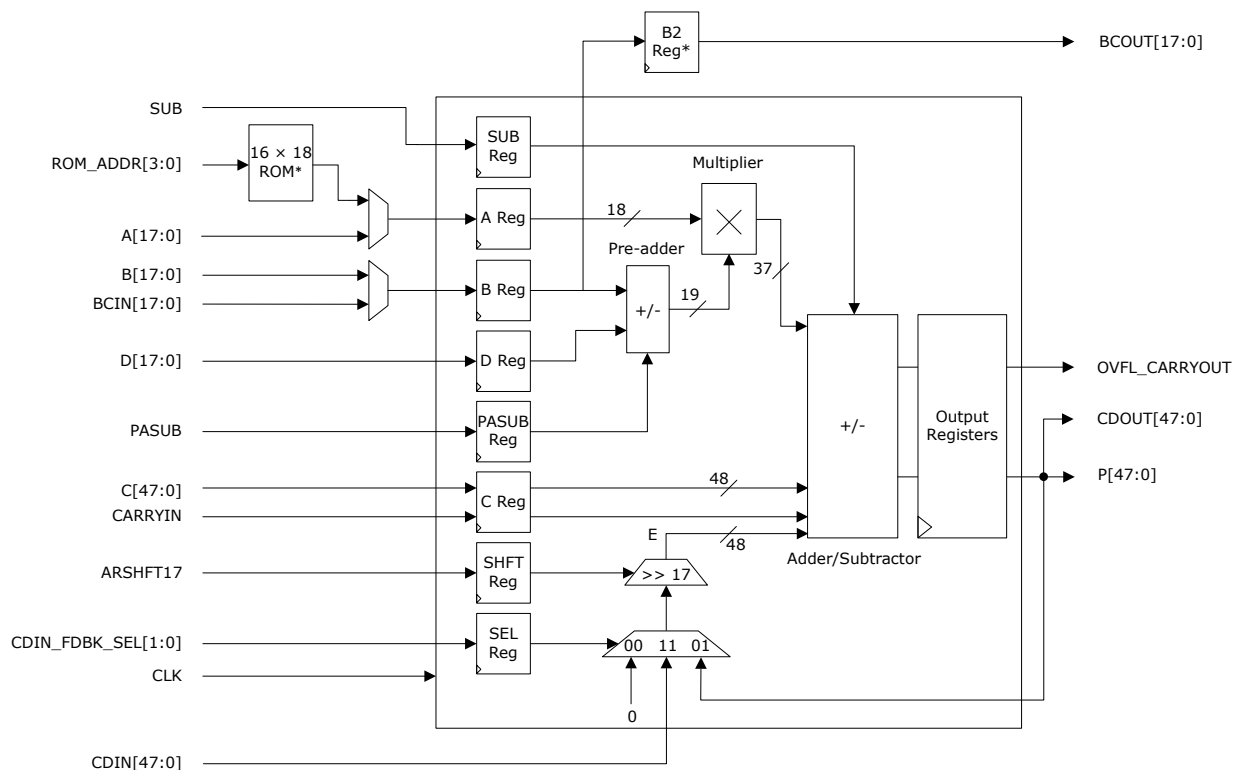
Math blocks are arranged in rows in the FPGA fabric and can be cascaded in a chain, starting from the left-most block to the right-most block within a row.

Each Math block consists of:

- [Pre-Adder](#)
- [Multiplier](#)
- [Adder/Subtractor](#)
- [Math Block Ports](#)
- [16 x 18 Coefficient ROM and B2 Register](#)

The following figure shows the simplified block diagram of the Math block.

Figure 3-1. Simplified Functional Block Diagram of Math Block



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

The following table lists the ports of the Math block.

Table 3-1. 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.

Table 3-1. Port List for Math Block (continued)

Port Name	Direction	Type	Polarity	Description
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 6-17](#).

3.3.1. Pre-Adder [\(Ask a Question\)](#)

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

- 18-bit addition/subtraction, producing the result $B[17:0] \pm D[17:0]$. For more information, see [Figure 3-6](#).
- Two 9-bit additions/subtractions, producing the results $B[8:0] \pm D[8:0]$, $B[17:9] \pm D[17:9]$. For more information, see [Figure 3-7](#). 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 $B[17:9] \pm D[17:9]$. For more information, see [Figure 3-8](#).

3.3.2. Multiplier [\(Ask a Question\)](#)

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).

3.3.3. Adder/Subtractor [\(Ask a Question\)](#)

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:

- $(B \pm D) \times A + C + \text{CARRYIN}$ or $(B \pm D) \times A + E$.
- $(B \pm D) \times A + C + \text{CARRYIN} + E$.

If this block is configured as a subtractor, the output is $(C + E) - (B \pm D) \times A$.

3.3.4. Math Block Ports [\(Ask a Question\)](#)

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

3.3.4.1. C Input and CARRYIN [\(Ask a Question\)](#)

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.

3.3.4.2. CDIN, CDOUT, and CDIN_FDBK_SEL [\(Ask a Question\)](#)

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 3-3](#) 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](#).

3.3.4.3. BCOUT [\(Ask a Question\)](#)

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.

3.3.4.4. OVFL_CARRYOUT [\(Ask a Question\)](#)

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 3-2. 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.

3.3.4.5. ARSHFT17 [\(Ask a Question\)](#)

For multi-precision arithmetic, Math blocks provide a right shift by 17 that is controlled by the shift input, ARSHFT17. Therefore, 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.

3.3.4.6. CDIN_FDBK_SEL [\(Ask a Question\)](#)

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 3-3. 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]}

3.3.4.7. PASUB and SUB Inputs [\(Ask a Question\)](#)

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 3-4. Truth Table for Computation of Result P and CDOUT

SIMD	DOTP	SUB	PASUB	Result P and CDOUT
0	0	0	0	$CARRYIN + C[47:0] + E[47:0] + \{ (B[17:0] + D[17:0]) \times A[17:0] \}$
0	0	0	1	$CARRYIN + C[47:0] + E[47:0] + \{ (B[17:0] - D[17:0]) \times A[17:0] \}$
0	0	1	0	$CARRYIN + C[47:0] + E[47:0] - \{ (B[17:0] + D[17:0]) \times A[17:0] \}$
0	0	1	1	$CARRYIN + C[47:0] + E[47:0] - \{ (B[17:0] - D[17:0]) \times A[17:0] \}$
0	1	0	0	$CARRYIN + C[47:0] + E[47:0] + \{ (B[8:0] + D[8:0]) \times A[17:9] + (B[17:9] + D[17:9]) \times A[8:0] \} \times 2^9$
0	1	0	1	$CARRYIN + C[47:0] + E[47:0] + \{ (B[8:0] - D[8:0]) \times A[17:9] + (B[17:9] - D[17:9]) \times A[8:0] \} \times 2^9$
0	1	1	0	$CARRYIN + C[47:0] + E[47:0] + \{ (B[8:0] + D[8:0]) \times A[17:9] - (B[17:9] + D[17:9]) \times A[8:0] \} \times 2^9$
0	1	1	1	$CARRYIN + C[47:0] + E[47:0] + \{ (B[8:0] - D[8:0]) \times A[17:9] - (B[17:9] - D[17:9]) \times A[8:0] \} \times 2^9$
1	0	0	0	$P[17:0] = CARRYIN + \{ B[8:0] \times A[8:0] \}$ $P[47:18] = C[47:18] + E[47:18] + \{ (B[17:9] + D[17:9]) \times A[17:9] \}$
1	0	0	1	$P[17:0] = CARRYIN + \{ B[8:0] \times A[8:0] \}$ $P[47:18] = C[47:18] + E[47:18] + \{ (B[17:9] - D[17:9]) \times A[17:9] \}$
1	0	1	0	$P[17:0] = CARRYIN + \{ B[8:0] \times A[8:0] \}$ $P[47:18] = C[47:18] + E[47:18] - \{ (B[17:9] + D[17:9]) \times A[17:9] \}$
1	0	1	1	$P[17:0] = CARRYIN + \{ B[8:0] \times A[8:0] \}$ $P[47:18] = C[47:18] + E[47:18] - \{ (B[17:9] - D[17:9]) \times A[17:9] \}$

3.3.5. 16 x 18 Coefficient ROM and B2 Register [\(Ask a Question\)](#)

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 3-1](#). 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 μ PROM.

3.4. Cascading Math Blocks [\(Ask a Question\)](#)

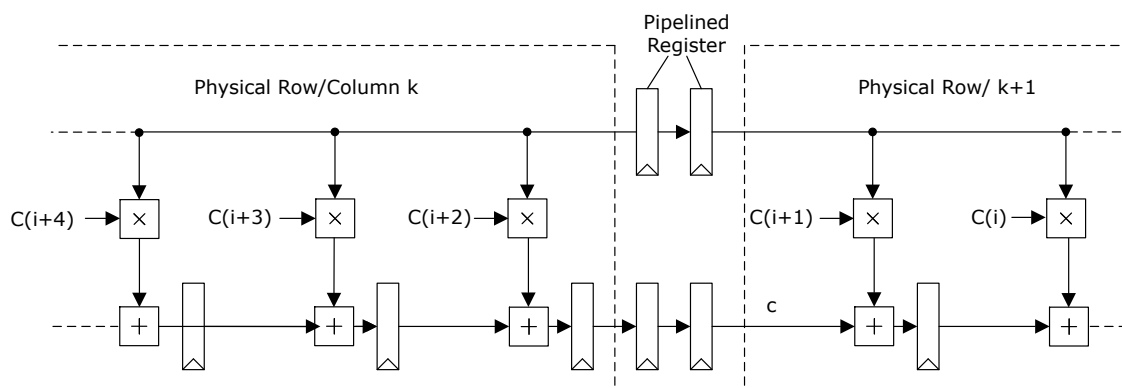
The Math blocks in each row cascade in a chain from left to right and terminate at the end of each row. Fabric routing is required to extend the cascading chain from one row to another. The following are the two options to connect cascading chains across multiple rows:

- [Wrap-Around to the Start of the Cascade Chain](#)
- [Wrap-Around to the Inside of the Cascade Chain](#)

3.4.1. Wrap-Around to the Start of the Cascade Chain [\(Ask a Question\)](#)

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. The following figure 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 3-2. Wrap-Around of Cascade Chain Using Pipeline Registers in Fabric

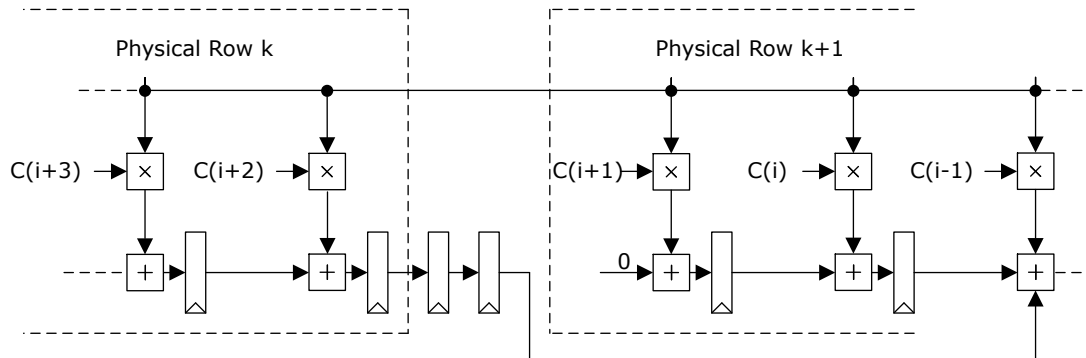


3.4.2. Wrap-Around to the Inside of the Cascade Chain [\(Ask a Question\)](#)

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.

The following figure shows an example implementation of a wrap-around to the inside of a cascade chain.

Figure 3-3. Wrap-Around to Inside of Cascade Chain

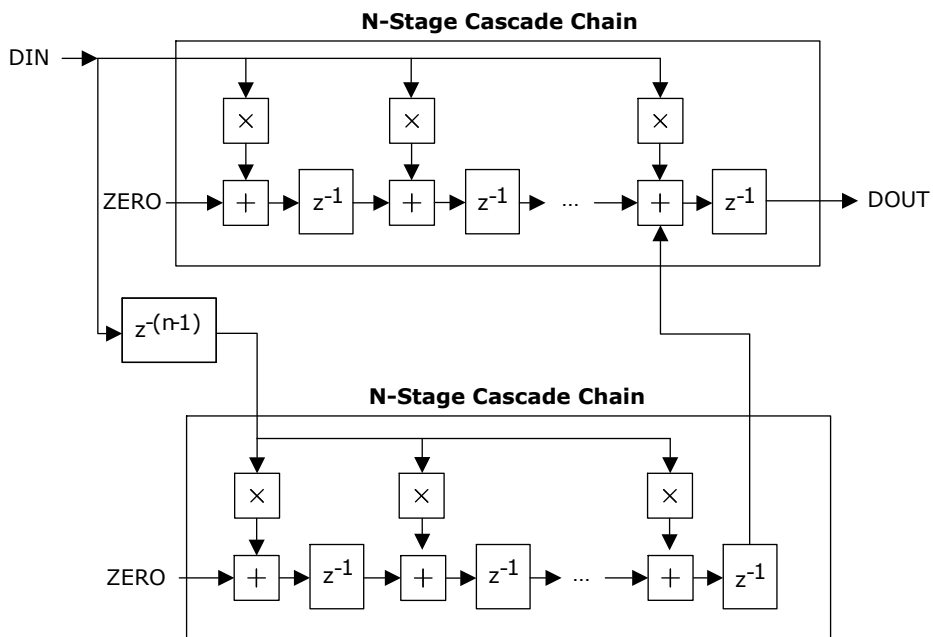


3.4.2.1. Merging Multiple Cascade Chains [\(Ask a Question\)](#)

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.

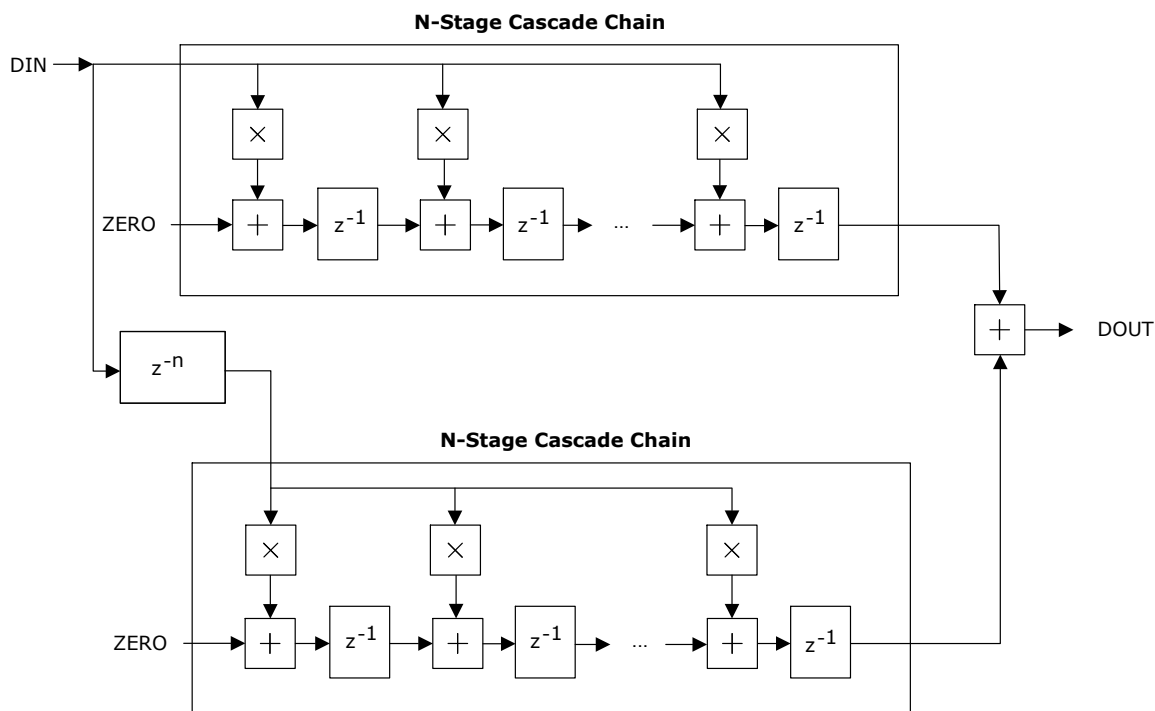
The following figure shows multiple cascaded chains merged using a math block C input.

Figure 3-4. Merging Cascade Chains Using C Input



The following figure shows multiple cascaded chains merged with fabric adders.

Figure 3-5. Merging Cascade Chains Using Fabric Adders



3.5. Operational Modes [\(Ask a Question\)](#)

The Math block supports the following operational modes:

- [Normal Mode](#)
- [DOTP Mode](#)
- [SIMD Mode](#)

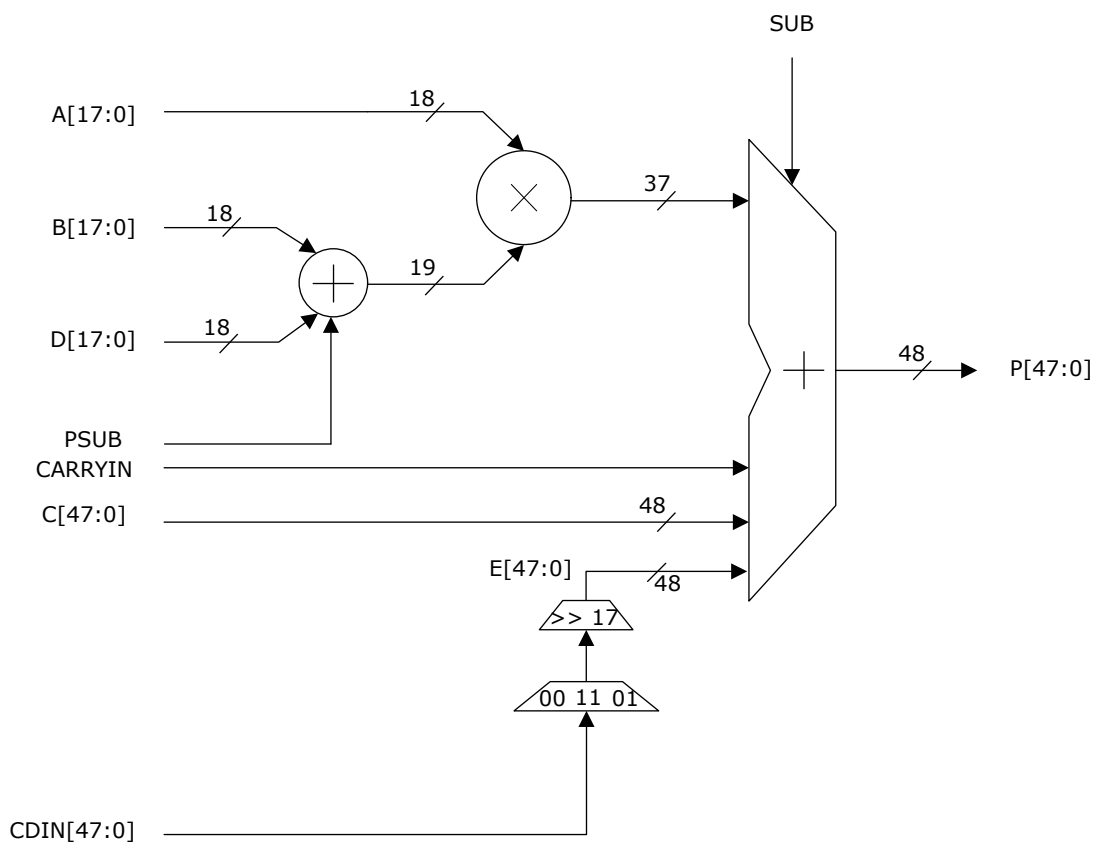
3.5.1. Normal Mode [\(Ask a Question\)](#)

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

$$P[47:0] = ((B[17:0] \pm D[17:0]) \times A[17:0]) + C[47:0] + E[47:0] + \text{CARRYIN}$$

The following figure shows the functional block diagram of the Math block in normal mode.

Figure 3-6. Functional Block Diagram of the Math Block in Normal Mode



3.5.2. DOTP Mode [\(Ask a Question\)](#)

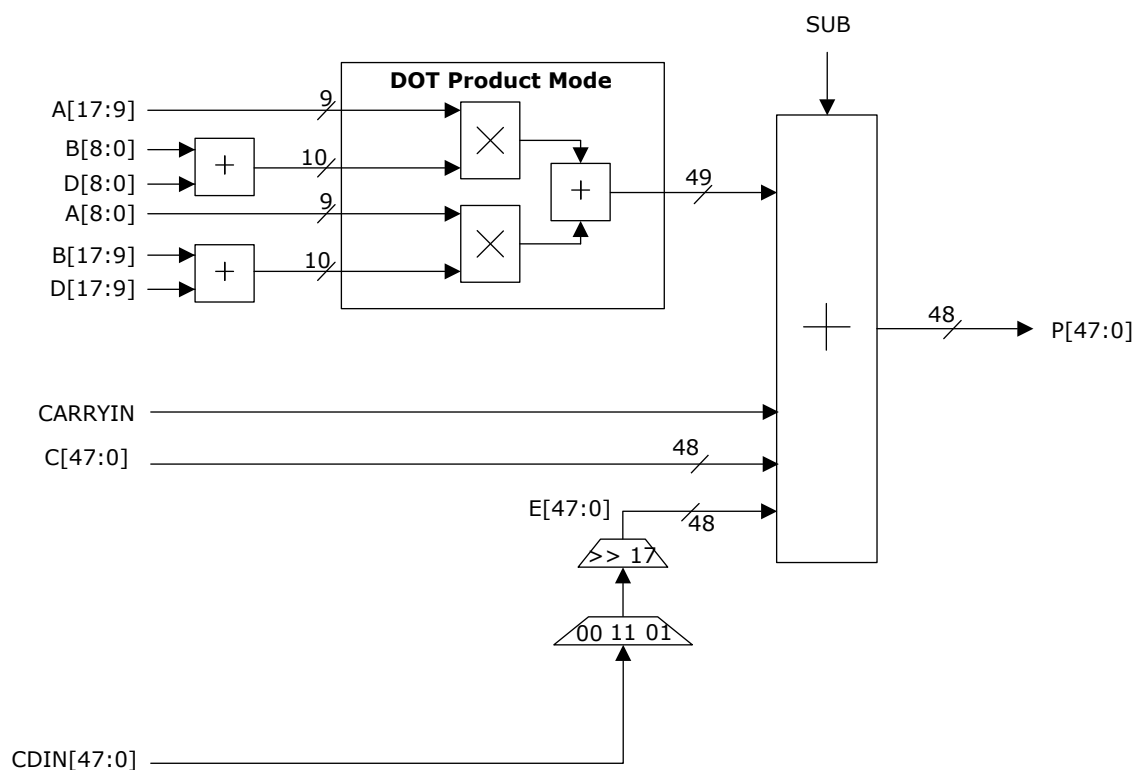
DOTP mode computes the sum of two 10 x 9-bit multiplications. In this mode, the pre-adder is split into two 9-bit adders. The two 9-bit pre-adders must be in the same mode, either addition or subtraction.

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

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

The following figure shows the functional block diagram of the Math block in DOTP mode.

Figure 3-7. Functional Block Diagram of the Math Block in DOTP Mode



3.5.3. SIMD Mode [\(Ask a Question\)](#)

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.

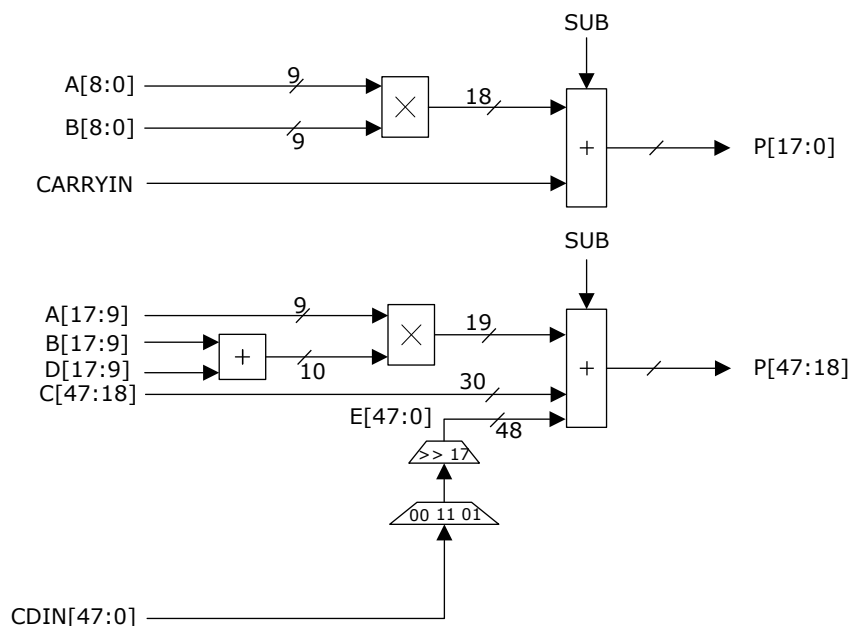
SIMD mode implements the following output equations:

$$P[17:0] = (B[8:0] \times A[8:0]) + \text{CARRYIN}$$

$$P[47:18] = ((B[17:9] \pm D[17:9]) \times A[17:9]) + C[47:18] + E[47:18]$$

The following figure shows the functional block diagram of the Math block in SIMD mode.

Figure 3-8. Functional Block Diagram of the Math Block in SIMD Mode



3.6. Implementation [\(Ask a Question\)](#)

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

- [RTL Inference](#)
- [Math Block Macro](#)
- [Use Models](#)

3.6.1. RTL Inference [\(Ask a Question\)](#)

Synplify Pro 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 ties unused dynamic input signals to ground and provides default values to unused static signals.

Synplify Pro automatically maps any multiplication functions with input widths of three or more to Math blocks. For input widths less than three, Math blocks 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 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, see [Inferring PolarFire MACC Blocks Application Note](#).

3.6.2. Math Block Macro [\(Ask a Question\)](#)

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](#).

3.6.3. Use Models [\(Ask a Question\)](#)

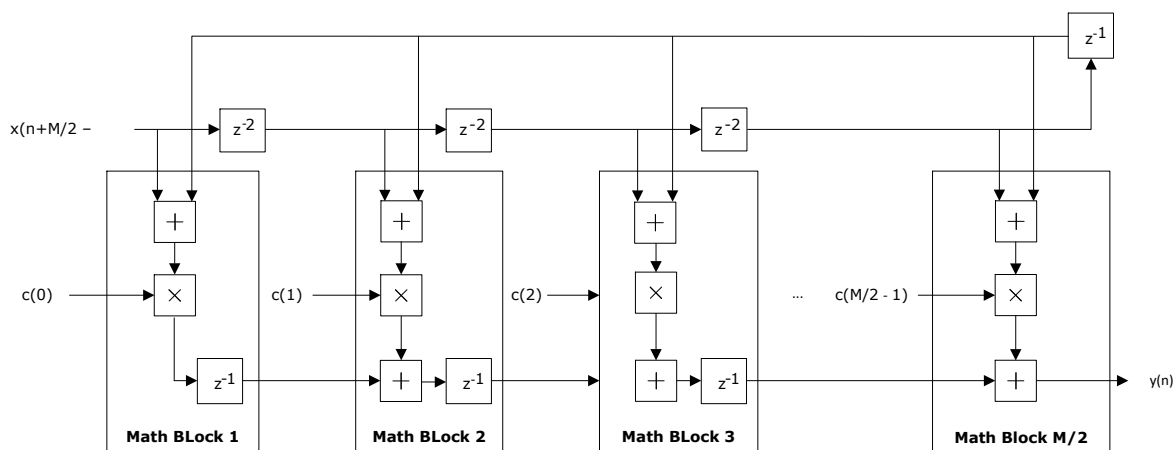
Math blocks can be used in a wide variety of DSP applications and this section describes the following use models:

- [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](#)

3.6.3.1. Symmetric FIR Filter [\(Ask a Question\)](#)

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 the following figure.

Figure 3-9. Symmetric FIR Filter Implementation

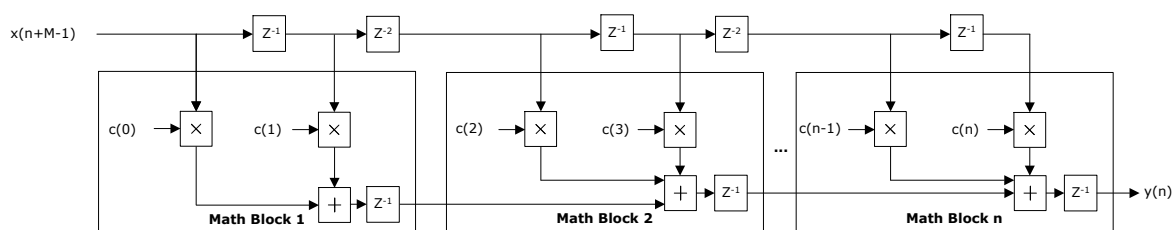


Anti-symmetric FIR filters can also be implemented in the same way, but with the pre-adder configured to subtract the input signals feeding the anti-symmetric filter coefficients, as shown in Figure 3-10.

3.6.3.2. 9 x 9 Systolic FIR Filter [\(Ask a Question\)](#)

A 9 x 9 systolic FIR filter can be implemented using the 9 x 9 DOTP mode of the Math block, see Figure 3-10. 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.

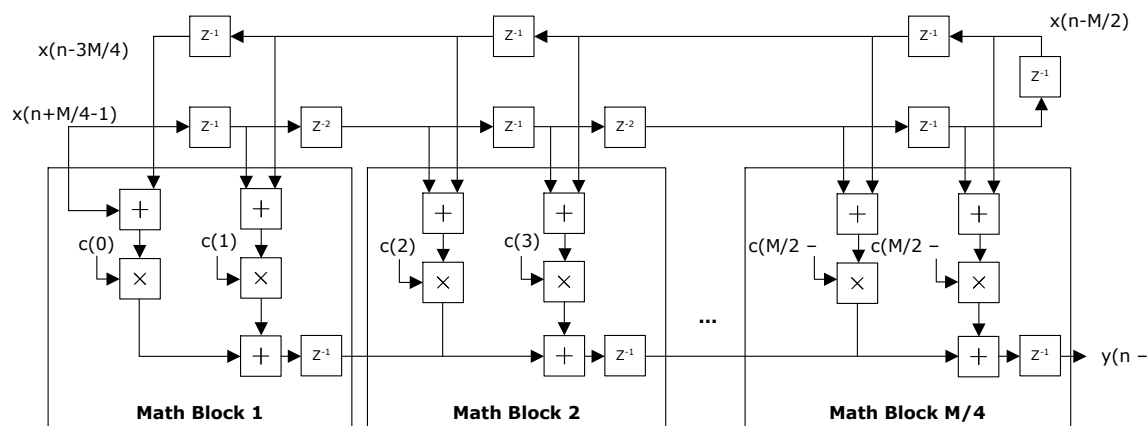
Figure 3-10. 9 x 9 Systolic FIR Filter



3.6.3.3. 9 x 9 Symmetric FIR Filter [\(Ask a Question\)](#)

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. The following figure shows an example of a 9 x 9 symmetric FIR filter for M number (divisible by 4) of taps.

Figure 3-11. 9 x 9 Symmetric FIR Filter



3.6.3.4. 9 x 9 Complex Multiplier [\(Ask a Question\)](#)

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 3-12, and this logic consumes minimal fabric resources.

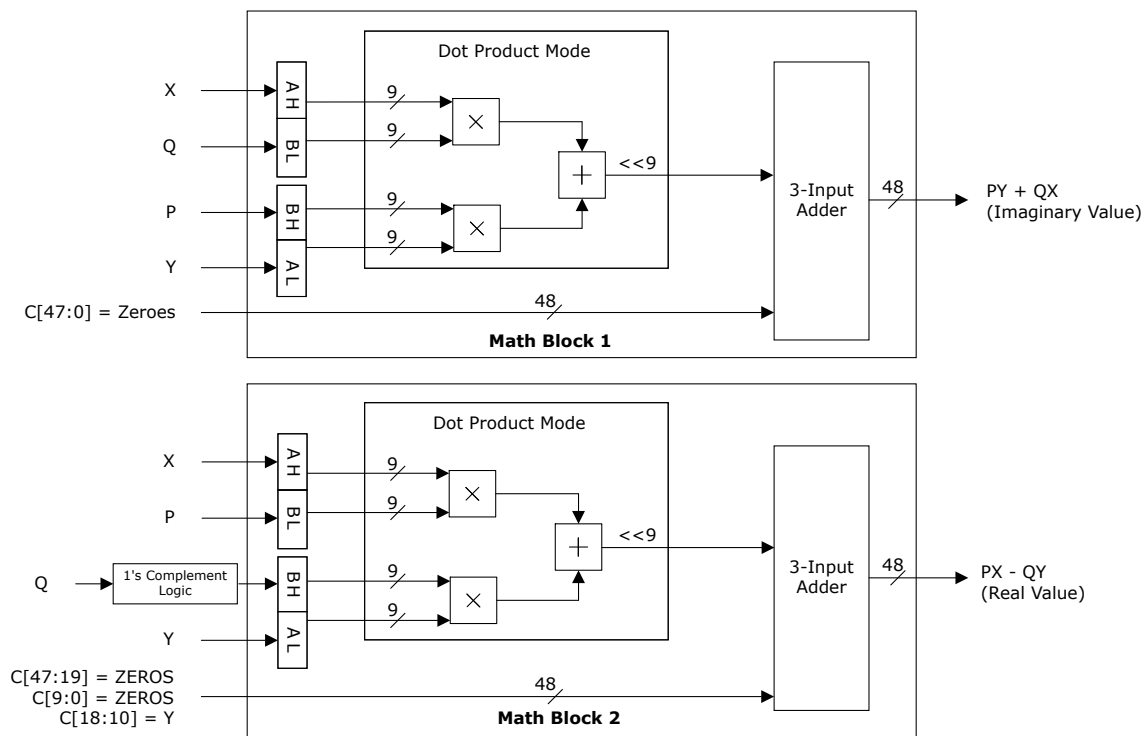
For two complex numbers, $X + jY$ and $P + jQ$, the complex multiplication is:

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 ($-Q = \sim Q + 1$).

$$\text{Real part} = P \times X + (\sim Q) \times Y + Y.$$

Figure 3-12. 9-Bit Complex Multiplication Using DOTP Mode

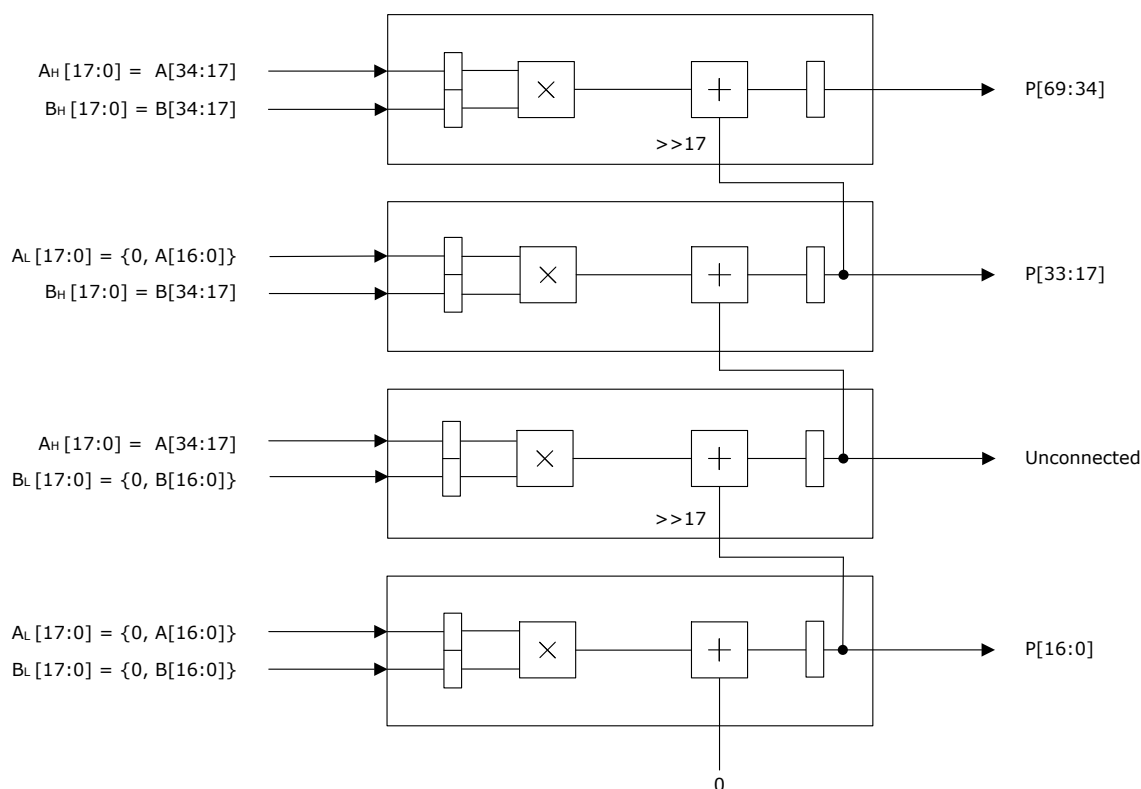


3.6.3.5. Non-Pipelined 35 x 35 Multiplier Using Cascade Chain [\(Ask a Question\)](#)

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 3-13](#) 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 3-13. Non-Pipelined Implementation Using Cascade Chain

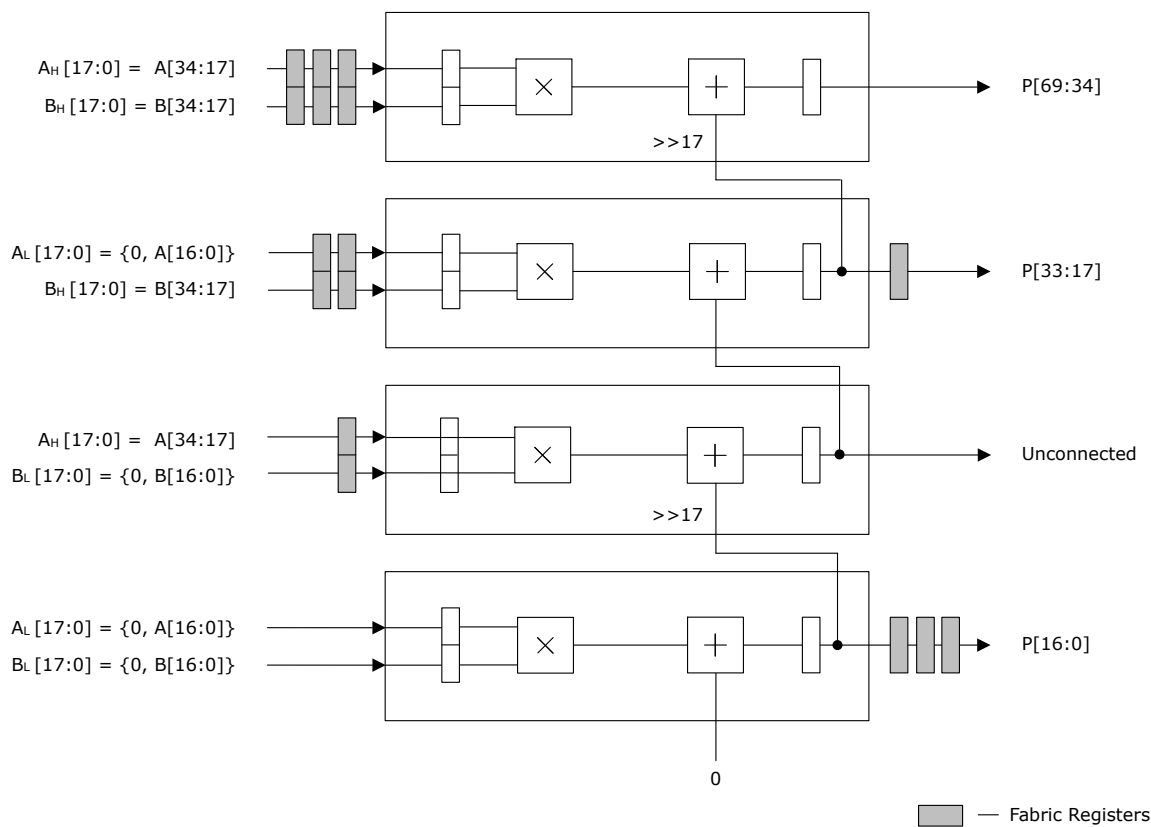


3.6.3.6. Pipelined 35 x 35 Multiplier Using Cascade Chain [\(Ask a Question\)](#)

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.

The following figure shows a typical 35 x 35 multiplier implementation with fabric pipeline registers.

Figure 3-14. Pipelined Implementation Using Cascade Chain

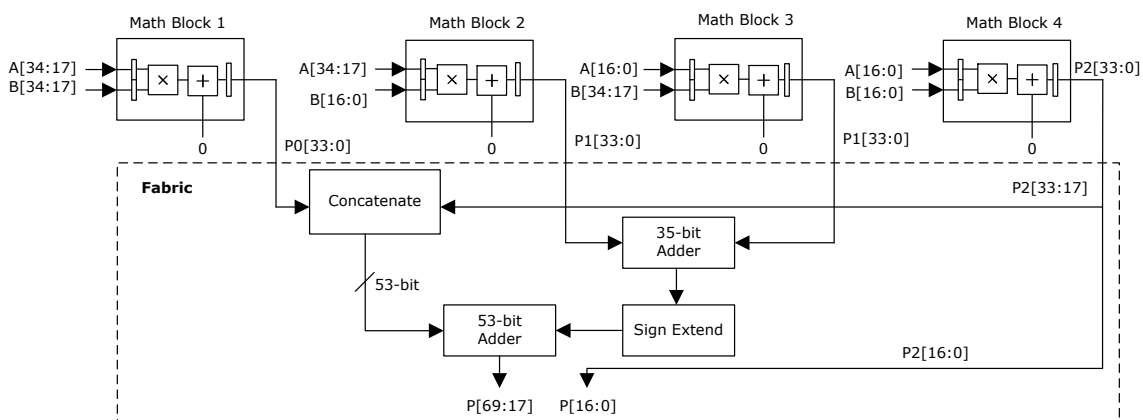


3.6.3.7. Non-Pipelined 35 x 35 Multiplier Using Fabric Adders [\(Ask a Question\)](#)

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.

The following figure shows the block diagram of the 35 x 35 multiply using fabric adders.

Figure 3-15. 35 x 35 Multiply Using Fabric Adders

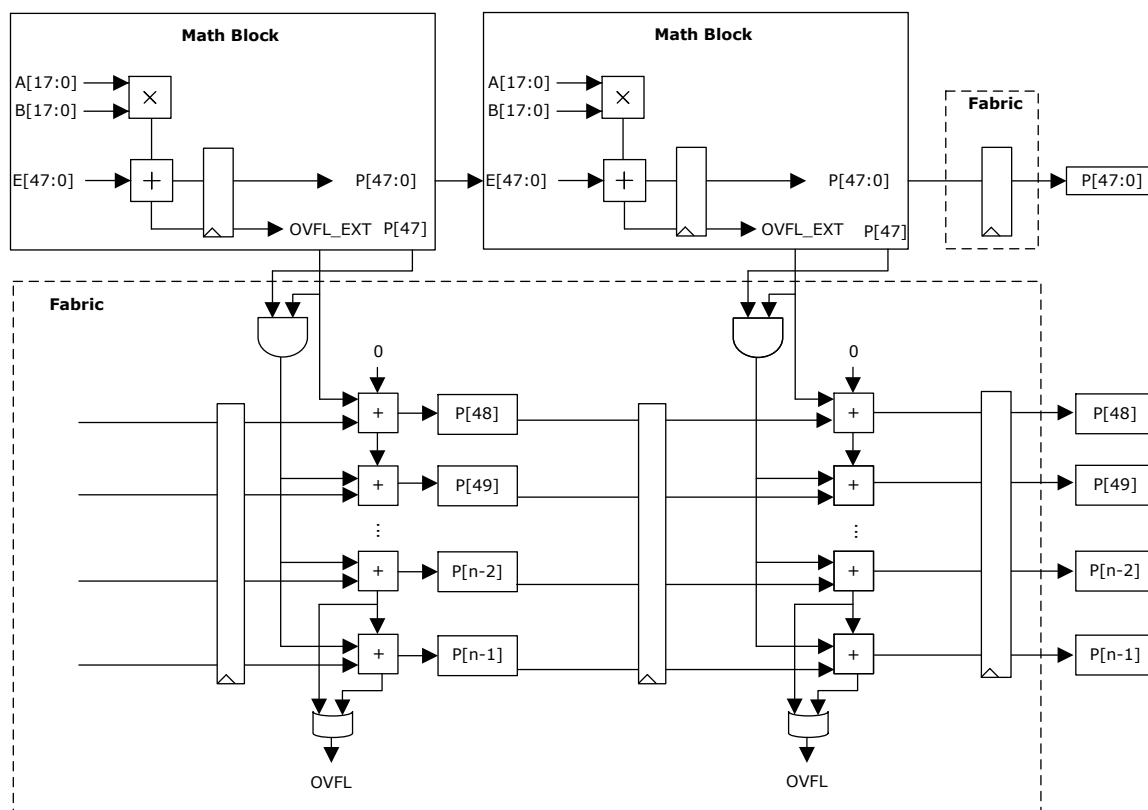


3.6.3.8. Extension Adder [\(Ask a Question\)](#)

A Math block's adder/accumulator width can be extended using fabric logic. [Figure 3-16](#) and [Figure 3-17](#) 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.

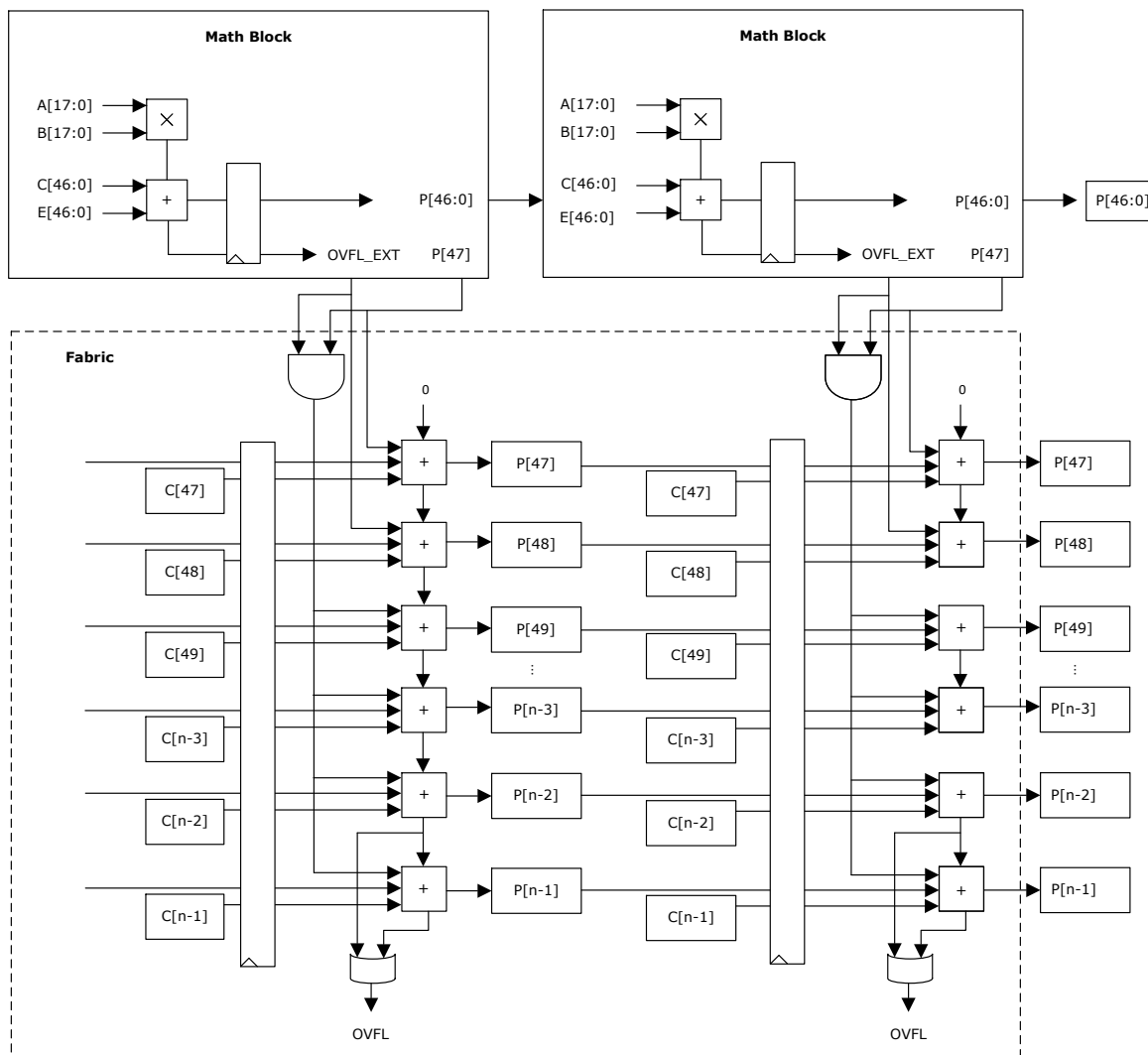
The following figure shows a case where input C = 0, and input E is used for the cascade chain.

Figure 3-16. Extension of 2-Input Adder in Fabric



The following figure shows a case where both inputs C and E are used.

Figure 3-17. Extension of 3-Input Adder in Fabric



4. Appendix: SRAM Optimization for High-Speed Versus Low-Power [\(Ask a](#)

Question)

The PolarFire μ SRAM and LSRAM blocks can be optimized for high-speed or low-power depending upon the option selected in the configurator. The **High Speed** or **Low Power** optimization setting affects the resulting initialization data size for custom values and might cause the initialization data to not fit in sNVM. For example, suppose a design uses two LSRAMs (LSRAM_1 and LSRAM_2) as follows:

- LSRAM_1 uses **High Speed** optimization setting with each individual RAM block aspect ratio configured for a higher memory depth and a lower memory width (x1). These x1 bit RAMs are cascaded side-by-side to form the user's desired memory word width. For example, to create a 32-bit word, 32 of the x1 width RAM blocks are cascaded widthwise.
- LSRAM_2 uses **Low Speed** optimization setting with each individual RAM block aspect ratio configured for a lower memory depth and a higher memory width (x20 or x40 width, as needed depending on the RAM mode and desired word-width).

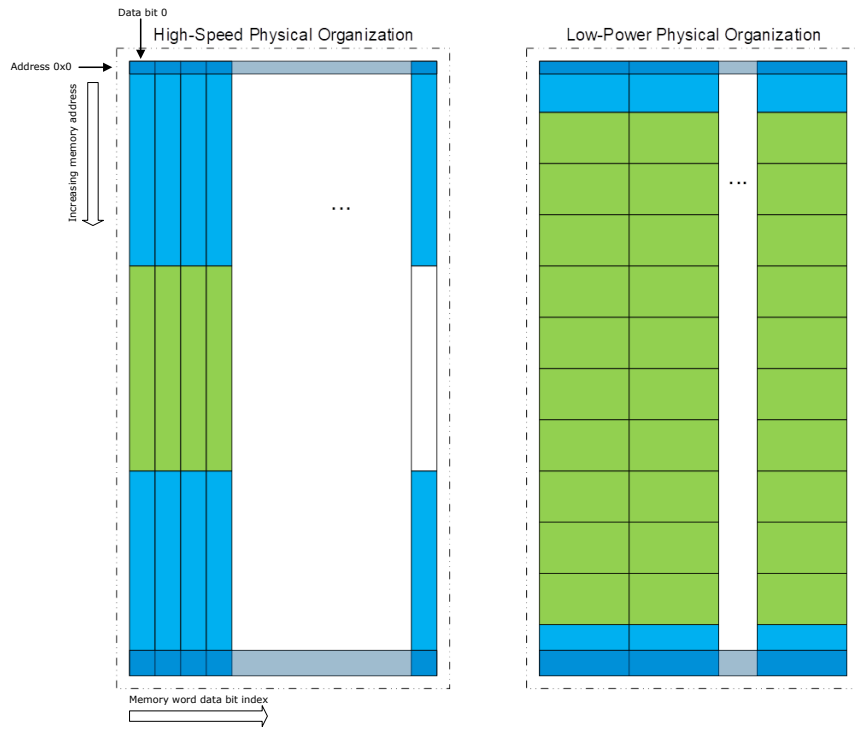
Figure 4-1 illustrates the physical organization for high-speed and low-power optimization of LSRAM_1 and LSRAM_2. In Figure 4-1, the blue blocks represent the custom user data blocks that need to be initialized and the darker blue portion of these blocks is the actual data that is non-zero in these physical blocks. The green blocks represent the physical RAM blocks configured to initialize to zero.

During PolarFire System Controller power-up initialization of LSRAMs, all RAMs that are configured for initialization are first initialized to zero using a broadcast method that requires very little space in the Non-Volatile Memory (NVM), such as sNVM. This process covers all the green blocks in Figure 4-1. However, for the darker blue blocks that have non-zero data, each block must be initialized independently, which could require a lot of non-volatile storage space to hold the custom data. Depending on the physical RAM block topology used, such as high-speed versus low-power mode, it defines how many non-zero RAM blocks must be initialized. Even in the situation shown in Figure 4-1, where there is a relatively small amount of non-zero data in the darker blue areas, the high-speed topology requires that each x1 RAM block cascaded widthwise be individually initialized from data stored in NVM, such as 32 blocks, multiplied by how many depth-cascaded rows are also initialized, out of the total number of rows required to achieve the user desired memory depth. In contrast, for the low-power topology, far fewer individual RAM blocks need to be initialized, resulting in much less NVM storage required to hold the initialization data.

The same concept of high-speed versus low-power SRAM optimization applies to RAM blocks inferred by Synplify Pro. Therefore, when inferring RAM blocks with custom initialized content, the amount of non-volatile storage required depends on the optimization style used. For information about methods to control the RAM optimization style for inferred RAMs, See the "RAM Inference in Low Power Mode" section of the [Inferring Microchip PolarFire RAM Blocks Application Note](#).

In summary, for the same set of custom initialization data, physical organization is defined by the optimization and port setting defines how many non-zero blocks must be initialized. In this example, the initialization data of LSRAM_1 might not fit in the sNVM as there are many more non-zero blocks.

Figure 4-1. Physical Organization for High-Speed and Low-Power RAM Optimization Setting



5. Appendix: Supported Memory File Formats for LSRAM and μ SRAM (Ask a Question)

a Question)

Microchip supports Intel-Hex, Motorola-S, Simple-Hex, and Binary file formats.

Note: Implementation of these formats interpret data sets in bytes. If the memory width is 7 bits, then every eighth 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. For more information, see [Write Port Width Alignment](#)

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 Microchip SoC tools).
 - 04—extended linear address record.
 - 05—start linear address record (ignored by Microchip 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
```

Note: Configurator organizes data according to big Endian sequence. Intel-Hex format requires byte-aligned port widths. For more information, see [Write Port Width Alignment](#).

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:

`S1laaaa[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
```

Note: Configurator organizes data according to big Endian sequence. Motorola-S format requires byte-aligned port widths. For more information, see [Write Port Width Alignment](#).

Simple-Hex

Simple-Hex is a format created by Autodesk. SHX files are used for environment enabling designs of 2D and 3D projects for engineering purposes. These files end with an SHX extension (for example, `file2.shx` or `file3.shx`).

The Simple-Hex record is arranged as follows:

```
0:2000D9B8
```

Where, the colon (:) separates the address from the data.

Note: Configurator organizes data according to little Endian sequence. Simple-Hex format requires byte-aligned port widths. For more information, see [Write Port Width Alignment](#).

Binary

Binary files consist of a sequence of bytes, with the binary digits (bits) grouped in eights. Files consist of 0s and 1s and end with a BIN extension. For example, `file2.bin` or `file3.bin`.

The Binary record is composed as follows:

```
001100101010101010010010
```

Note: Configurator organizes data according to little Endian sequence. Binary format requires byte-aligned port widths. For more information, see [Write Port Width Alignment](#).

MEMFILE (RAM Content Manager output file)

Transfer of RAM data (from the RAM Content Manager) to test equipment is accomplished through MEM files.

The contents of RAM is first organized into the logical layer and then reorganized to fit the hardware layer. Then it is stored in MEM files that are read by other systems and used for testing. The MEM files are named according to the logical structure of the RAM elements created by the configurator. In this scheme, the highest order RAM blocks are named `CORE_R0C0.mem`, where R stands for

row and C stands for column. For multiple RAM blocks, the naming continues with CORE_R0C1, CORE_R0C2, CORE_R1C0, and so on.

The data intended for the RAM is stored as ASCII 1s and 0s within the file. Each memory address occupies one line. Words from the logical layer blocks are concatenated or split in order to make them fit efficiently within the hardware blocks. If the logical layer width is less than the hardware layer, two or more logical layer words are concatenated to form one hardware layer word. In this case, the lowest bits of the hardware word are made up of the lower address data bits from the logical layer. If the logical layer width is more than the hardware layer, the words are split, placing the lower bits in lower addresses.

If the logical layer words do not fit cleanly into the hardware layer words, the most significant bit of the hardware layer words is not used and defaulted to zero. This is also done when the logical layer width is 1 in order to avoid having leftover memory at the end of the hardware block.

5.1. Write Port Width Alignment [\(Ask a Question\)](#)

The Microchip implementation of these formats interprets data sets in bytes. The implementation used is the same for all memory formats. The following examples show how data in a memory file is interpreted for different write port widths of memory.

The following figure shows data in Intel HEX memory file format. The same data is assumed to be used to initialize RAM in all examples.

Figure 5-1. Memory File Data—Intel HEX Memory File Format

:04000000	FF11EE22	DC
:04000400	DD33CC44	D8
:04000800	BB550012	D2

The Hex data from the memory file is converted into binary and read by the tool as a stream of bits. Based on the memory port width, the *required number of bits* for each address location in RAM is taken from the stream of bits.



Important: The *required number of bits* taken from the stream of bits for each address location is always a multiple of 8 (byte).

5.1.1. Write Port Widths Aligned on Byte Boundary [\(Ask a Question\)](#)

The following examples show how data from a memory file is stored in RAM when memory port widths are a multiple of a byte (aligned on a byte boundary).

5.1.1.1. 32-Bit Write Port Width [\(Ask a Question\)](#)

When the memory write port width is 32 bits, which is aligned on a byte boundary, the tool uses 32 bits from the binary stream for each 32-bit word in the RAM. The resulting data stored in RAM is shown in the following figure.

Figure 5-2. Data in RAM—32-Bit Write Port Width Aligned on Byte Boundary

Memory File Data	Address	Data Stored in RAM
:04000000FF11EE22DC	0	0x22EE11FF
:04000400DD33CC44D8	1	0x44CC33DD
:04000800BB550012D2	2	0x120055BB

5.1.1.2. 16-bit Write Port Width [\(Ask a Question\)](#)

When the memory write port width is 16 bits, which is aligned on a byte boundary, the tool uses 16 bits from the binary stream for each 16-bit word in the RAM. The resulting data stored in RAM is shown in the following figure.

Figure 5-3. Data in RAM - 16-bit Write Port Width Aligned on Byte Boundary

Memory File Data	Address	Data Stored in RAM
:04000000FF11EE22DC	0	0x11FF
:04000400DD33CC44D8	1	0x22EE
:04000800BB550012D2	2	0x33DD
	3	0x44CC
	4	0x55BB
	5	0x1200

5.1.1.3. 8-bit Write Port Width [\(Ask a Question\)](#)

When the memory write port width is 8 bits, which is aligned on a byte boundary, the tool uses 8 bits from the binary stream for each 8-bit word in the RAM. The resulting data stored in RAM is shown in the following figure.

Figure 5-4. Data in RAM - 8-bit Write Port Width Aligned on Byte Boundary

Memory File Data	Address	Data Stored in RAM
:04000000FF11EE22DC	0	0xFF
:04000400DD33CC44D8	1	0x11
:04000800BB550012D2	2	0xEE
	3	0x22
	4	0xDD
	5	0x33
	6	0xCC
	7	0x44
	8	0xBB
	9	0x55
	A	0x00
	B	0x12

5.1.2. Write Port Widths Not Aligned on Byte Boundary [\(Ask a Question\)](#)

The following examples show how data from a memory file is stored in RAM when memory port widths are not a multiple of a byte.

5.1.2.1. 9-bit Write Port Width [\(Ask a Question\)](#)

When the memory write port width is 9 bits, which is not aligned on a byte boundary, the tool uses 16 bits from the binary stream for each 9-bit word. The tool ignores the upper 7 bits of each 16 bits when processing the memory file data. The resulting data stored in RAM is shown in the following figure.

Figure 5-5. Data in RAM - 9-bit Write Port Width Not Aligned on Byte Boundary

Memory File Data	Address	Data Stored in RAM
:04000000FF11EE22DC	0	0x1FF
:04000400DD33CC44D8	1	0x0EE
:04000800BB550012D2	2	0x1DD
	3	0x0CC
	4	0x1BB
	5	0x000

5.1.2.2. 4-Bit Write Port Width [\(Ask a Question\)](#)

When the memory write port width is 4 bits, which is not aligned on a byte boundary, the tool uses 8 bits from the binary stream for each 4-bit word. The tool ignores the upper 4 bits of each 8 bits when processing the memory file data. The resulting data stored in RAM is shown in the following figure.

Figure 5-6. Data in RAM—4-Bit Write Port Width Not Aligned on Byte Boundary

Memory File Data	
:04000000	FF11EE22DC
:04000400	DD33CC44D8
:04000800	BB550012D2

Address	Data Stored in RAM
0	0xF
1	0x1
2	0xE
3	0x2
4	0xD
5	0x3
6	0xC
7	0x4
8	0xB
9	0x5
A	0x0
B	0x2

5.1.3. Specifying Data in Memory File [\(Ask a Question\)](#)

If all the bits in the memory file are relevant, take steps to avoid bits from being ignored when the write port width is not aligned on a byte boundary. The following examples describe how to specify data in memory file so that the tool does not ignore bits. To avoid bits from being ignored, convert the hexadecimal data intended to initialize RAM into binary stream of bits, and then pad (insert) zeros based on the port width so that resulting data is byte-aligned, as described in the following sections.

5.1.3.1. 9-bit Write Port Width [\(Ask a Question\)](#)

Consider the following Intel HEX memory file.

Figure 5-7. Memory File Data - Intel HEX Memory File

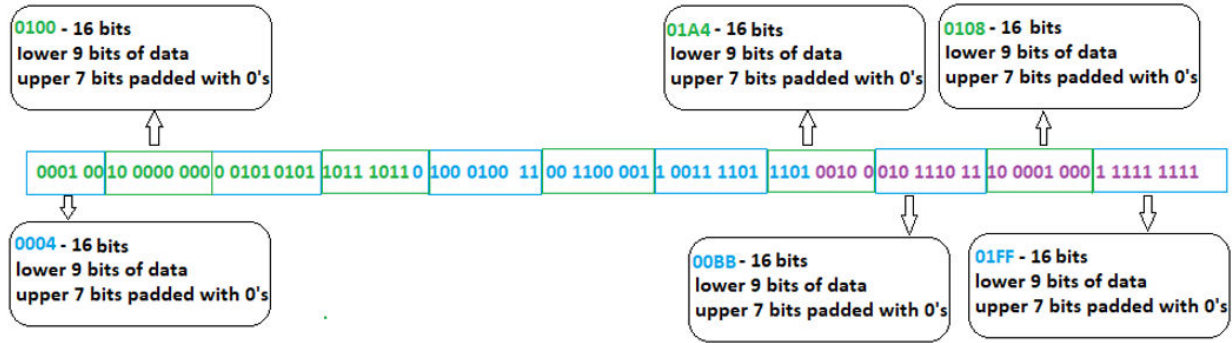
:04000000	FF11EE22DC
:04000400	DD33CC44D8
:04000800	BB550012D2

The binary stream of bits for above memory file data is:

0001 0010 0000 0000 0101 0101 1011 1011 0100 0100 1100 1100 0011 0011 1101 1101 0010 0010 1110 1110 0001 0001 1111 1111

If the memory port width is 9 bits, you must pad 7 zeros to every 9 bits of data from the binary stream to create 16 bits (byte-aligned), as shown in the following figure.

Figure 5-8. Padding Zeros to Create 16 Bits



The following figure shows the equivalent memory file data padded with zeros to achieve a 9-bit write port width.

Figure 5-9. Equivalent Memory File Data Padded with Zeros (9-bit Write Port Width)

: 04000000	FF010801F3
: 04000400	BB00A40198
: 04000800	3D01610055
: 04000c00	1301760165
: 04001000	5500000196
: 04001400	04000000E4

When the tool parses the above memory file data (padded with zeros), the tool converts the data to binary and reads it as a stream of bits. If the port width is 9 bits, the tool reads 16 bits (byte-aligned), ignores the upper 7 bits, and stores the lower 9 bits of actual data in RAM, as shown in the following table.

Table 5-1. 16-bit Write Port Width

Address	Data
0	0x1FF
1	0x108
2	0x0BB
3	0x1A4
4	0x13D
5	0x061
6	0x113
7	0x176
8	0x055
9	0x100
A	0x004
B	0x000

5.1.3.2. 4-Bit Write Port Width [\(Ask a Question\)](#)

Enter a short description of your concept here (optional).

Consider the following Intel HEX memory file.

Figure 5-10. Memory File Data—Intel HEX Memory File

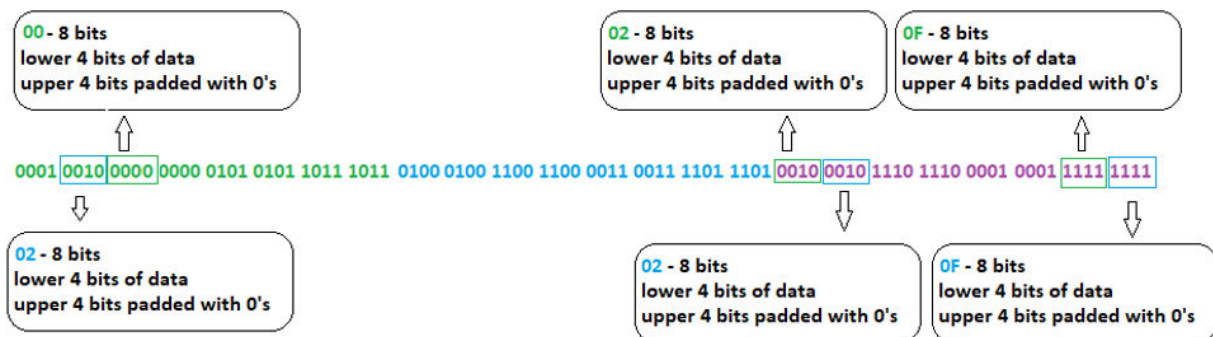
:04000000	FF11EE22DC
:04000400	DD33CC44D8
:04000800	BB550012D2

The binary stream of bits for above memory file data is:

0001 0010 0000 0000 0101 0101 1011 1011 0100 0100 1100 1100 0011 0011 1101 1101 0010 0010 1110 1110 0001 0001 1111 1111

If the memory port width is 4 bits, the tool reads 8 bits at a time from the binary stream above. For the 8 bits, you must pad zeros for the upper 4 bits and specify the actual data in the lower 4 bits, as shown in the following figure.

Figure 5-11. Padding Zeros for the Upper 4 Bits and Specifying Data in the Lower 4 Bits



The following figure shows the equivalent memory file data padded with zeros to achieve a 4-bit write port width.

Figure 5-12. Equivalent Memory File Data Padded with Zeros (4-bit Write Port Width)

:04000000	0F0F0101DC
:04000400	0E0E0202D8
:04000800	D0D0303D4
:04000c00	c0c0404D0
:04001000	B0B0505CC
:04001400	000201E5

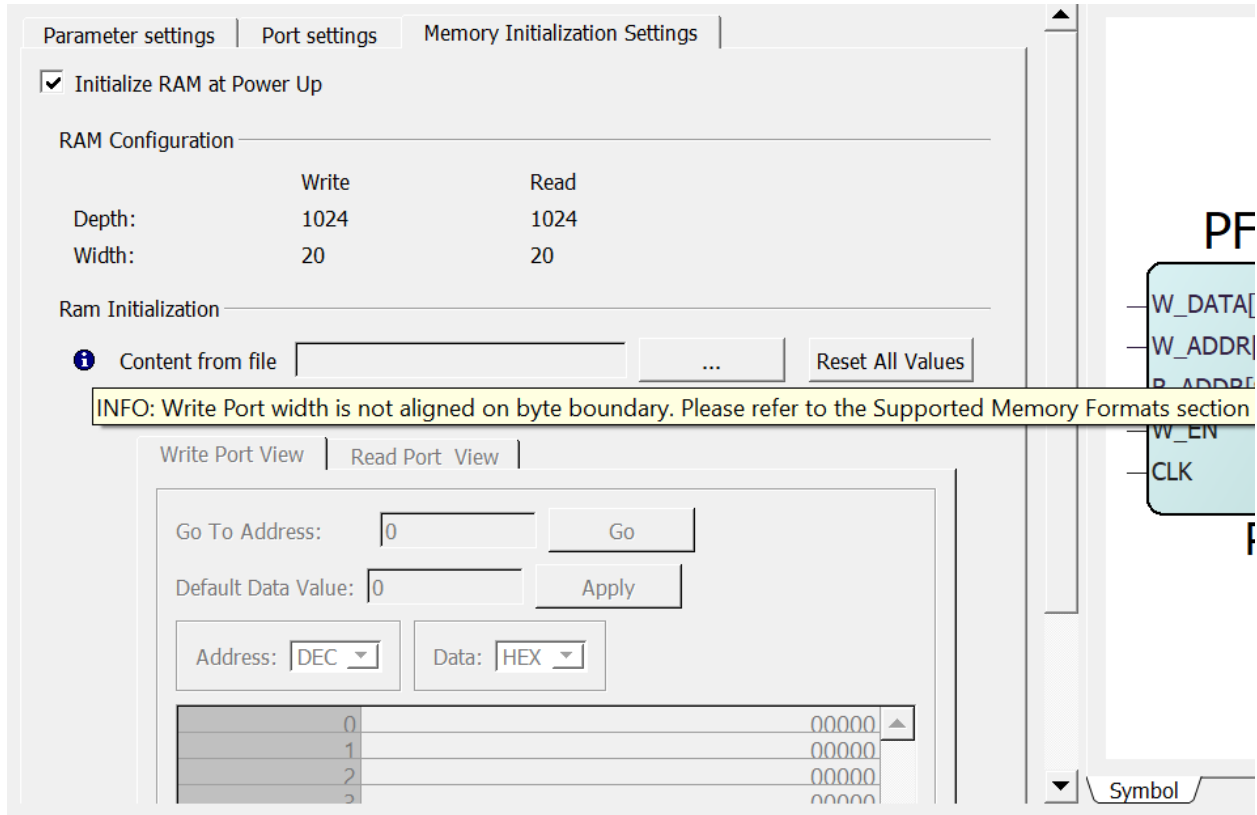
When the tool parses the above memory file data (padded with zeros), it converts the data to binary and reads it as a stream of bits. If the port width is 4 bits, the tool reads 8 bits (byte-aligned), ignores the upper 4 bits of actual data, and stores the lower 4 bits of actual data in RAM, as listed in the following table.

Table 5-2. 16-bit Write Port Width

Address	Data
0	0xF
1	0xF
2	0x1
3	0x1
4	0xE
5	0xE
6	0x2
7	0x2
8	0xD
9	0xD
A	0x3
B	0x3
C	0xC
D	0xC
E	0x4
F	0x4
10	0xB
11	0xB
12	0x5
13	0x5
14	0x0
15	0x0
16	0x2
17	0x1

Note: x1 and x2 port widths are handled using the same technique of padding zeros. You always zero pad to the next 8-bit-width increment (8, 16, 24, and so on). If a write port width is not aligned on a byte boundary, the following message appears.

Figure 5-13. Message when a Write Port Width is Not Aligned on a Byte Boundary



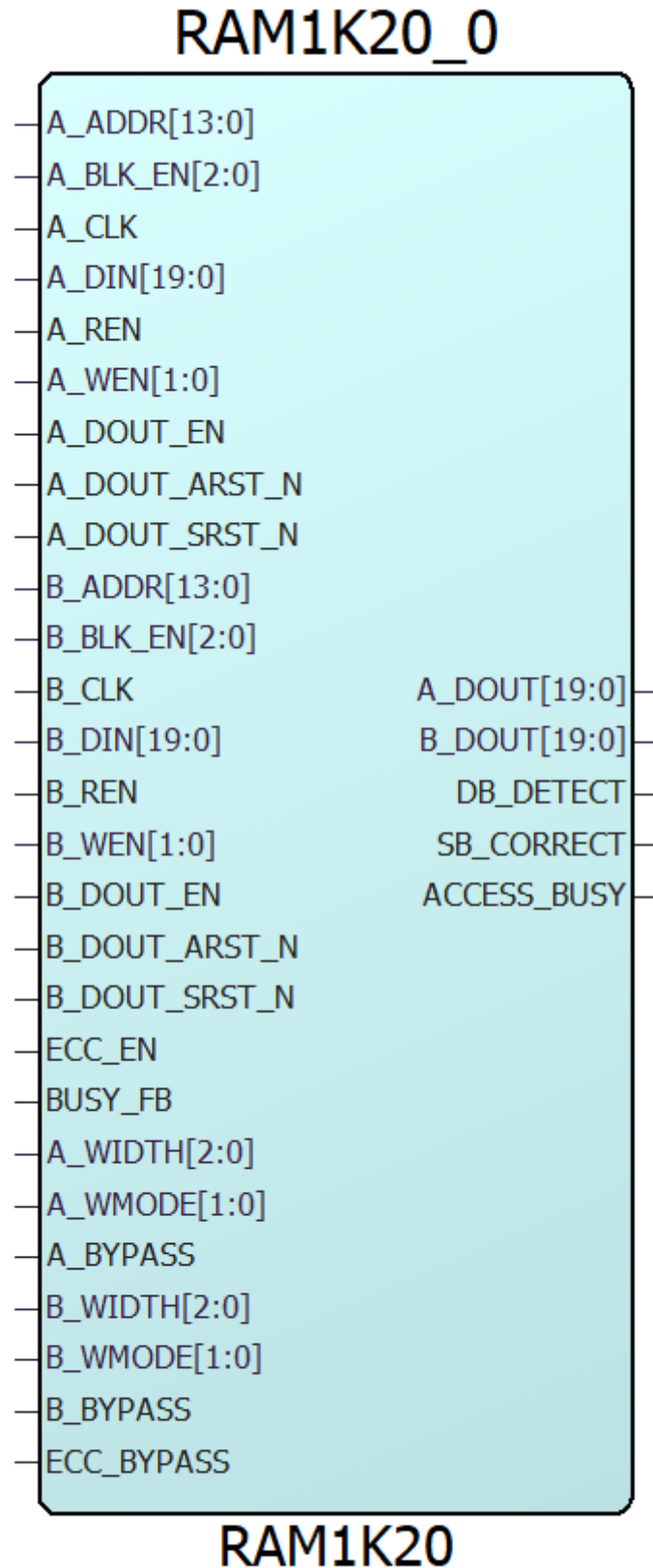
6. Appendix: Macro Configuration [\(Ask a Question\)](#)

This section describes the configuration of LSRAM, μ SRAM, and Math block macros.

6.1. LSRAM Macro [\(Ask a Question\)](#)

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](#). The following figure shows the LSRAM macro (RAM1K20).

Figure 6-1. RAM1K20 Macro



The following table lists the ports of RAM1K20.

Table 6-1. 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 High	Port A pipeline register bypassed when High
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 High	Port B pipeline register bypassed when High
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 High	ECC pipeline register bypassed when High
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

Note:

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

6.1.1. A_WIDTH and B_WIDTH [\(Ask a Question\)](#)

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. The following table lists the width/depth mode selections for each port.

Table 6-2. Width/Depth Mode Selection

Depth x Width	A_WIDTH/B_WIDTH
16K x 1	000

Table 6-2. Width/Depth Mode Selection (continued)

Depth x Width	A_WIDTH/B_WIDTH
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

6.1.2. A_WEN and B_WEN [\(Ask a Question\)](#)

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. The following table lists the write/read control signals for each port.

Table 6-3. 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	Write B_DIN[16:0]
	A_WEN[1:0] = 11	Write A_DIN[15:0]

6.1.3. A_ADDR and B_ADDR [\(Ask a Question\)](#)

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. The following table lists the address buses for the two ports.

Table 6-4. Write/Read Operation Select

Depth x Width	A_ADDR/B_ADDR	
	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]

6.1.4. A_DIN and B_DIN [\(Ask a Question\)](#)

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. The following table lists the data input buses for the two ports.

Table 6-5. Data Input Buses Used and Unused Bits

Depth x Width	A_DIN/B_DIN	
	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_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]

6.1.5. A_DOUT and B_DOUT [\(Ask a Question\)](#)

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. The following table lists the data output buses for the two ports.

Table 6-6. Data Output Buses Used and Unused Bits

Depth x Width	A_ADDR/B_ADDR	
	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
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]

6.1.6. A_BLK_EN and B_BLK_EN [\(Ask a Question\)](#)

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. The following table lists the block-port select control signals for the two ports.

Table 6-7. 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 is forced to 0. If the width is greater than 20 bits, the read-data from both ports A and B is 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 is forced to 0, unless the width is greater than 20 bits and write operation to both ports A and B is gated.

6.1.7. A_WMODE and B_WMODE [\(Ask a Question\)](#)

In Dual-Port write mode, each port has a feed-through write or read-before-write option. When A_WMODE or 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.

6.1.8. A_CLK and B_CLK [\(Ask a Question\)](#)

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.

6.1.9. A_REN and B_REN [\(Ask a Question\)](#)

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.

6.1.9.1. Read-Data Pipeline Register Control Signals [\(Ask a Question\)](#)

- 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.

The following table lists the functionality of the control signals on the A_DOUT and B_DOUT pipeline registers.

Table 6-8. 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	Qn+1
0	X	X	X	X	X	0
1	0	Not rising	X	X	X	Qn
1	0	↑	0	X	X	Qn
1	0	↑	1	0	X	0
1	0	↑	1	1	D	D
1	1	X	X	X	D	D

6.1.10. ECC_EN and ECC_BYPASS [\(Ask a Question\)](#)

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.

6.1.11. SB_CORRECT and DB_DETECT [\(Ask a Question\)](#)

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

Table 6-9. 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.

6.1.12. BUSY_FB [\(Ask a Question\)](#)

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

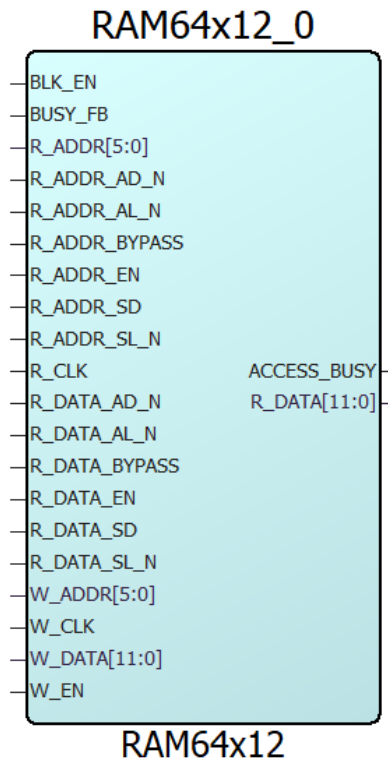
6.1.13. ACCESS_BUSY [\(Ask a Question\)](#)

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

6.2. μ SRAM Macro [\(Ask a Question\)](#)

The μ SRAM macro (RAM64x12) in Libero SoC can be used directly to instantiate μ SRAM in the design. μ SRAM must be configured correctly with appropriate values provided to the static signals before instantiating in the design. Instantiating μ SRAM primitives in a design is not recommended. For the recommended methods of instantiating memory into a user design, see [\$\mu\$ SRAM Memory Macro](#). The following figure shows the μ SRAM macro (RAM64x12) available in the Libero SoC macro library.

Figure 6-2. RAM64x12 Macro



The following table lists the ports of RAM64x12.

Table 6-10. 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 is 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
R_ADDR_BYPASS	Input	Static	Active high	Read-address and BLK_EN register bypassed when high
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

Table 6-10. Port List for RAM64x12 (continued)

Pin Name	Direction	Type ¹	Polarity	Description
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 high	Read-data pipeline register bypassed when high
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

Note:

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

6.2.1. Read-Address and Read-Data Pipeline Register Control Signals [\(Ask a Question\)](#)

The following table lists the functionality of the control signals on the R_ADDR and R_DATA registers.

Table 6-11. 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	Qn+1
0	ADn	X	X	X	X	X	X	!ADn
1	X	0	Not rising	X	X	X	X	Qn
1	X	0	↑	0	X	X	X	Qn
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

6.3. Math Block Macro [\(Ask a Question\)](#)

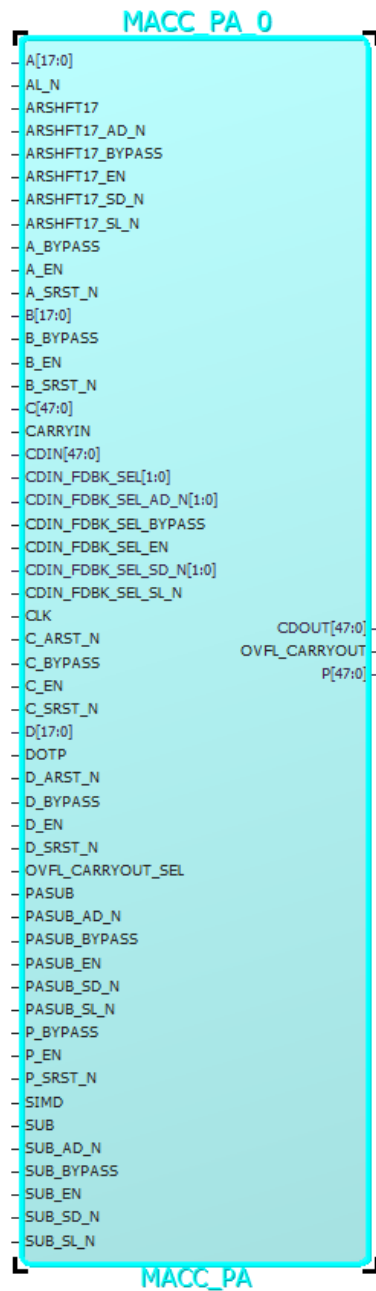
Two math block macros, MACC_PA and MACC_PA_BC_ROM, are available in Libero SoC IP catalog macro library. These macros can be used in designs created with SmartDesign or by directly instantiating the macro wrapper in an HDL file as a component. Instantiating math block primitives in a design is not recommended. For the recommended methods of instantiating math blocks into a user design, see [Implementation](#). 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.

6.3.1. MACC_PA (MACC with Pre-Adder) [\(Ask a Question\)](#)

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.

The following figure shows the MACC_PA available in the macro library.

Figure 6-3. MACC_PA Macro



6.3.1.1. Port List [\(Ask a Question\)](#)

The following table lists the MACC_PA ports.

Table 6-12. 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 3-3 .
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 6-14 .
A_SRST_N	Input	Dynamic	Active low	Synchronous reset for data A registers. Connect to 1 if not registered. For more information, see Table 6-14 .
A_EN	Input	Dynamic	Active high	Enable for data A registers. Connect to 1 if not registered. For more information, see Table 6-14 .
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 6-14 .
B_SRST_N	Input	Dynamic	Active low	Synchronous reset for data B registers. Connect to 1 if not registered. For more information, see Table 6-14 .
B_EN	Input	Dynamic	Active high	Enable for data B registers. Connect to 1 if not registered. For more information, see Table 6-14 .
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 6-15 .
D_ARST_N	Input	Dynamic	Active low	Asynchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 6-15 .
D_SRST_N	Input	Dynamic	Active low	Synchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 6-15 .
D_EN	Input	Dynamic	Active high	Enable for data D registers. Connect to 1 if not registered. For more information, see Table 6-15 .
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 6-15 .

Table 6-12. MACC_PA Pin Descriptions (continued)

Port Name	Direction	Type ¹	Polarity	Description
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 6-15 .
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 6-15 .
C_EN	Input	Dynamic	Active high	Enable for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 6-15 .
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 3-3 .
P[47:0]	Output		Active high	Result data. For more information, see Table 3-4 .
OVFL_CARRYOUT	Output		Active high	OVERFLOW or CARRYOUT. For more information, see Table 3-4 .
P_BYPASS	Input	Static	Active high	Bypass P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 6-14 .
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 6-14 .
P_EN	Input	Dynamic	Active high	Enable for P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 6-14 .
CDOUT[47:0]	Output	Cascade	Active high	Cascade output of result P. For more information, see Table 3-4 . 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 6-13 .
PASUB_AD_N	Input	Static	Active low	Asynchronous load data for PASUB register. For more information, see Table 6-13 .
PASUB_SL_N	Input	Dynamic	Active low	Synchronous load for PASUB register. Connect to 1 if not registered. For more information, see Table 6-13 .
PASUB_SD_N	Input	Static	Active low	Synchronous load data for PASUB register. For more information, see Table 6-13 .
PASUB_EN	Input	Dynamic	Active high	Enable for PASUB register. Connect to 1 if not registered. For more information, see Table 6-13 .
CDIN_FDBK_SEL[1:0]	Input	Dynamic	Active high	Select CDIN, P or 0 for operand E. For more information, see Table 3-3 .
CDIN_FDBK_SEL_BYPASS	Input	Static	Active high	Bypass CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 6-13 .
CDIN_FDBK_SEL_AD_N [1:0]	Input	Static	Active low	Asynchronous load data for CDIN_FDBK_SEL register. For more information, see Table 6-13 .
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 6-13 .
CDIN_FDBK_SEL_SD_N [1:0]	Input	Static	Active low	Synchronous load data for CDIN_FDBK_SEL register. For more information, see Table 6-13 .
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 6-13 .
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 3-3 . 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 6-13 .
ARSHFT17_AD_N	Input	Static	Active low	Asynchronous load data for ARSHFT17 register. For more information, see Table 6-13 .

Table 6-12. MACC_PA 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 6-13 .
ARSHFT17_SD_N	Input	Static	Active low	Synchronous load data for ARSHFT17 register. For more information, see Table 6-13 .
ARSHFT17_EN	Input	Dynamic	Active high	Enable for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 6-13 .
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 6-13 .
SUB_AD_N	Input	Static	Active low	Asynchronous load data for SUB register. For more information, see Table 6-13 .
SUB_SL_N	Input	Dynamic	Active low	Synchronous load for SUB register. Connect to 1 if not registered. For more information, see Table 6-13 .
SUB_SD_N	Input	Static	Active low	Synchronous load data for SUB register. For more information, see Table 6-13 .
SUB_EN	Input	Dynamic	Active high	Enable for SUB register. Connect to 1 if not registered. For more information, see Table 6-13 .

Note:

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 3-4](#), 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 6-13. Truth Table for Control Registers ARSHFT17, CDIN_FDBK_SEL, PASUB, and SUB

AL_N	_AD_N	_BYPASS	CLK	_EN	_SL_N	_SD_N	D	Qn+1
0	AD_N	0	X	X	X	X	X	!AD_N
1	X	0	Not rising	X	X	X	X	Qn
1	X	0	↑	0	X	X	X	Qn
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	Qn
X	X	1	X	1	0	SD_N	X	!SD_N
X	X	1	X	1	1	X	D	D

Table 6-14. Truth Table for Data Registers A, B, P, and OVFL_CARRYOUT

AL_N	_BYPASS	CLK	_EN	_SRST_N	D	Qn+1
0	0	X	X	X	X	0
1	0	Not rising	X	X	X	Qn
1	0	↑	0	X	X	Qn
1	0	↑	1	0	X	0
1	0	↑	1	1	D	D
X	1	X	0	X	X	Qn
X	1	X	1	0	X	0
X	1	X	1	1	D	D

Table 6-15. Truth Table for Data Registers C, CARRYIN, and D¹

_ARST_N	_BYPASS	CLK	_EN	_SRST_N	D	Qn+1
0	0	X	X	X	X	0
1	0	Not rising	X	X	X	Qn
1	0	↑	0	X	X	Qn

Table 6-15. Truth Table for Data Registers C, CARRYIN, and D¹ (continued)

_ARST_N	_BYPASS	CLK	_EN	_SRST_N	D	Qn+1
1	0	↑	1	0	X	0
1	0	↑	1	1	D	D
X	1	X	0	X	X	Qn
X	1	X	1	0	X	0
X	1	X	1	1	D	D

Note:

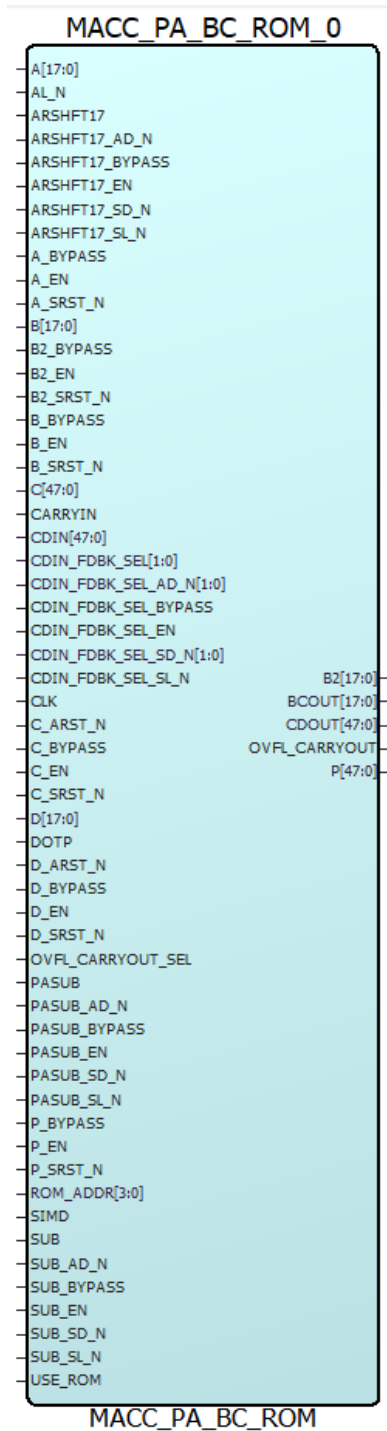
1. For more information, see [Math Blocks](#).

6.3.2. MACC_PA_BC_ROM (MACC with Pre-Adder, BCOU Register, and Coefficient ROM) [\(Ask a](#)

[Question\)](#)

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 6-4. MACC_PA_BC_ROM Macro



6.3.2.1. Parameters [\(Ask a Question\)](#)

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. The following table lists the INIT declaration for loading coefficients.

Table 6-16. 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, };	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"& B"00_0000_0000_0000_0000"& B"00_0000_0000_0000_0000"))	16 x 18 ROM content specified in VHDL.

6.3.2.2. Port List ([Ask a Question](#))**Table 6-17.** 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 3-3 .
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 6-17. 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 6-14 .
A_SRST_N	Input	Dynamic	Active low	Synchronous reset for data A registers. Connect to 1 if not registered. For more information, see Table 6-14 .
A_EN	Input	Dynamic	Active high	Enable for data A registers. Connect to 1 if not registered. For more information, see Table 6-14 .
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 6-14 .
B_SRST_N	Input	Dynamic	Active low	Synchronous reset for data B registers. Connect to 1 if not registered. For more information, see Table 6-14 .
B_EN	Input	Dynamic	Active high	Enable for data B registers. Connect to 1 if not registered. For more information, see Table 6-14 .
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 6-14 .
B2_SRST_N	Input	Dynamic	Active low	Synchronous reset for data B2 registers. Connect to 1 if not registered. For more information, see Table 6-14 .
B2_EN	Input	Dynamic	Active high	Enable for data B2 registers. Connect to 1 if not registered. For more information, see Table 6-14 .

Table 6-17. 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 6-15 .
D_ARST_N	Input	Dynamic	Active low	Asynchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 6-15 .
D_SRST_N	Input	Dynamic	Active low	Synchronous reset for data D registers. Connect to 1 if not registered. For more information, see Table 6-15 .
D_EN	Input	Dynamic	Active high	Enable for data D registers. Connect to 1 if not registered. For more information, see Table 6-15 .
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 6-15 .
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 6-15 .
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 6-15 .
C_EN	Input	Dynamic	Active high	Enable for CARRYIN and C registers. Connect to 1 if not registered. For more information, see Table 6-15 .
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 3-3 .
P[47:0]	Output	—	Active high	Result data. For more information, see Table 3-3 . B2 output must be floating when P is used.
OVFL_CARRYOUT	Output	—	Active high	OVERFLOW or CARRYOUT. For more information, see Table 3-2 .

Table 6-17. MACC_PA_BC_ROM Pin Descriptions (continued)

Port Name	Direction	Type ¹	Polarity	Description
P_BYPASS	Input	Static	Active high	Bypass P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 6-14 .
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 6-14 .
P_EN	Input	Dynamic	Active high	Enable for P and OVFL_CARRYOUT registers. Connect to 1 if not registered. For more information, see Table 6-14 .
CDOUT[47:0]	Output	Cascade	Active high	Cascade output of result P. For more information, see Table 3-4 . 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 6-13 .
PASUB_AD_N	Input	Static	Active low	Asynchronous load data for PASUB register. For more information, see Table 6-13 .
PASUB_SL_N	Input	Dynamic	Active low	Synchronous load for PASUB register. Connect to 1 if not registered. For more information, see Table 6-13 .
PASUB_SD_N	Input	Static	Active low	Synchronous load data for PASUB register. For more information, see Table 6-13 .
PASUB_EN	Input	Dynamic	Active high	Enable for PASUB register. Connect to 1 if not registered. For more information, see Table 6-13 .
CDIN_FDBK_SEL[1:0]	Input	Dynamic	Active high	Select CDIN, P or 0 for operand E. For more information, see Table 3-3 .
CDIN_FDBK_SEL_BYPASS	Input	Static	Active high	Bypass CDIN_FDBK_SEL register. Connect to 1 if not registered. For more information, see Table 6-13 .
CDIN_FDBK_SEL_AD_N [1:0]	Input	Static	Active low	Asynchronous load data for CDIN_FDBK_SEL register. For more information, see Table 6-13 .
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 6-13 .
CDIN_FDBK_SEL_SD_N [1:0]	Input	Static	Active low	Synchronous load data for CDIN_FDBK_SEL register. For more information, see Table 6-13 .
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 6-13 .

Table 6-17. MACC_PA_BC_ROM Pin Descriptions (continued)

Port Name	Direction	Type ¹	Polarity	Description
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 3-3 . 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 6-13 .
ARSHFT17_AD_N	Input	Static	Active low	Asynchronous load data for ARSHFT17 register. For more information, see Table 6-13 .
ARSHFT17_SL_N	Input	Dynamic	Active low	Synchronous load for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 6-13 .
ARSHFT17_SD_N	Input	Static	Active low	Synchronous load data for ARSHFT17 register. For more information, see Table 6-13 .
ARSHFT17_EN	Input	Dynamic	Active high	Enable for ARSHFT17 register. Connect to 1 if not registered. For more information, see Table 6-13 .
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 6-13 .
SUB_AD_N	Input	Static	Active low	Asynchronous load data for SUB register. For more information, see Table 6-13 .
SUB_SL_N	Input	Dynamic	Active low	Synchronous load for SUB register. Connect to 1 if not registered. For more information, see Table 6-13 .
SUB_SD_N	Input	Static	Active low	Synchronous load data for SUB register. For more information, see Table 6-13 .
SUB_EN	Input	Dynamic	Active high	Enable for SUB register. Connect to 1 if not registered. For more information, see Table 6-13 .

Note:

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

6.4. Libero SoC Compile Report [\(Ask a Question\)](#)

Libero SoC Design Suite offers high productivity with its comprehensive, easy-to-learn, easy-to-adopt development tools for designing with both the device families. The compile report contains fabric resource utilization and the total number of resources available. This report provides the number of 4LUTs, DFFs, μ SRAMs, 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 the following figure.

Figure 6-5. 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

For a sample compile report on MPFS250, see the following figure.

Figure 6-6. Sample Compile Report for MPFS250

Resource Usage

Type	Used	Total	Percentage
4LUT	15654	254196	6.16
DFF	12205	254196	4.80
I/O Register	0	432	0.00
User I/O	51	144	35.42
-- Single-ended I/O	51	144	35.42
-- Differential I/O Pairs	0	72	0.00
uSRAM	102	2352	4.34
LSRAM	49	812	6.03
Math	0	784	0.00
H-Chip Global	12	48	25.00
PLL	1	8	12.50
DLL	0	8	0.00
CRN_INT	1	24	4.17
INIT	1	1	100.00
OSC_RC160MHZ	1	1	100.00
DRI	1	1	100.00
Transceiver Lanes	4	4	100.00
Transceiver PCIe	1	2	50.00
TX_PLL	1	11	9.09
XCVR_REF_CLK	1	3	33.33
PCIE_COMMON	1	1	100.00
ICB_CLKDIV	2	24	8.33
ICB_CLKINT	7	72	9.72
NGMUX	1	12	8.33
ICB_INT	2	12	16.67
MSS	1	1	100.00
MSS I/O	101	136	74.26

Detailed Logic Resource Usage

Type	4LUT	DFF
Fabric Logic	12666	9217
uSRAM Interface Logic	1224	1224
LSRAM Interface Logic	1764	1764
Math Interface Logic	0	0
Total Used	15654	12205

For a sample compile report on RTPF500, see the following figure.

Figure 6-7. Sample Compile Report for RTPF500

Resource Usage

Type	Used	Total	Percentage
4LUT	178359	481272	37.06
DFF	217621	481272	45.22
User I/O	18	632	2.85
-- Single-ended I/O	18	632	2.85
-- Differential I/O Pairs	0	316	0.00
-- I/Os using I/O Registers	0	632	0.00
I/O Register Flip-Flops	0	1896	0.00
-- Input I/O Flip-Flops	0	632	0.00
-- Output I/O Flip-Flops	0	632	0.00
-- Enable I/O Flip-Flops	0	632	0.00
uSRAM	500	4440	11.26
LSRAM	451	1520	29.67
Math	500	1480	33.78
H-Chip Global	9	48	18.75
Local Global	4	1440	0.28
PLL	2	8	25.00
DLL	0	8	0.00
BANKEN	1	7	14.29
CRN_INT	2	24	8.33
INIT	1	1	100.00
OSC_RC160MHZ	1	1	100.00
DRI	1	1	100.00
Transceiver Lanes	2	24	8.33
Transceiver PCIe	0	2	0.00
TX_PLL	1	15	6.67
XCVR_REF_CLK	1	15	6.67
ICB_CLKDIV	1	24	4.17
ICB_CLKINT	7	72	9.72
ICB_INT	1	12	8.33

Detailed Logic Resource Usage

Type	4LUT	DFF
Fabric Logic	138123	177385
uSRAM Interface Logic	6000	6000
LSRAM Interface Logic	16236	16236
Math Interface Logic	18000	18000
Total Used	178359	217621

For a sample compile report on RTPFS460, see the following figure.

Figure 6-8. Sample Compile Report for RTPFS460

Resource Usage

Type	Used	Total	Percentage
4LUT	52922	461556	11.47
DFF	52494	461556	11.37
User I/O	157	516	30.43
-- Single-ended I/O	119	516	23.06
-- Differential I/O Pairs	19	258	7.36
-- I/Os using I/O Registers	0	516	0.00
I/O Register Flip-Flops	0	1548	0.00
-- Input I/O Flip-Flops	0	516	0.00
-- Output I/O Flip-Flops	0	516	0.00
-- Enable I/O Flip-Flops	0	516	0.00
uSRAM	78	4260	1.83
LSRAM	185	1460	12.67
Math	27	1420	1.90
H-Chip Global	16	48	33.33
Local Global	4	1440	0.28
PLL	3	8	37.50
DLL	1	8	12.50
CRN_INT	3	24	12.50
INIT	1	1	100.00
OSC_RC2MHZ	1	1	100.00
Transceiver Lanes	4	24	16.67
Transceiver PCIe	0	2	0.00
TX_PLL	1	15	6.67
XCVR_REF_CLK	1	15	6.67
ICB_CLKDIV	5	24	20.83
ICB_CLKINT	6	72	8.33
MSS	1	1	100.00
MSS I/O	2	136	1.47

Detailed Logic Resource Usage

Type	4LUT	DFF
Fabric Logic	44354	43926
uSRAM Interface Logic	936	936
LSRAM Interface Logic	6660	6660
Math Interface Logic	972	972
Total Used	52922	52494

7. Revision History (Ask a Question)

The revision history table describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Table 7-1. Revision History

Revision	Date	Description
J	05/2026	<p>The following is the summary of changes in this revision of the document:</p> <ul style="list-style-type: none"> Added the following new sections: <ul style="list-style-type: none"> Block Select Operation in Two-Port Mode Write Operation
I	02/2026	<p>The following is the summary of changes in this revision of the document:</p> <ul style="list-style-type: none"> Updated RTL Inference During Synthesis (LSRAM) and RTL Inference During Synthesis (μSRAM) by adding the guidance for providing RAM initialization content for Inferred RAM blocks. Updated the μPROM memory units to 513 Kb. Added Appendix: SRAM Optimization for High-Speed Versus Low-Power to describe the details about how high-speed versus low-power SRAM topology impacts the amount of NVM storage required to perform power-up LSRAM/μSRAM initialization using custom data. Added a footnote under Table 6-15. Added Figure 6-7 and Figure 6-8.
H	05/2025	<p>The following is the summary of changes in this revision of the document:</p> <ul style="list-style-type: none"> Updated the document for RT PolarFire SoC support. The changes are made throughout the document. Added the Simulating ECC Errors in LSRAM section and the following sub-sections: <ul style="list-style-type: none"> Probability-Based ECC Flag Assertion Verilog Task-Based Flag Assertion Simulation Flow Example LSRAM Waveforms Updated Embedded Memory Blocks as follows: <ul style="list-style-type: none"> Added information related to the sNVM size due to the addition of built-in SECCED ECC. Added a note related to eNVM in the Embedded Memory Blocks. Updated the sNVM and eNVM for the ECC feature in Table 2-1.
G	09/2023	<p>The following is the summary of changes in this revision of the document:</p> <ul style="list-style-type: none"> Added a line on how to enable the Reset All Values option for RAM initialization. See Dual-Port Large SRAM Configurator, Two-Port LSRAM Configurator, and Memory Initialization at Power-Up. Updated the polarity and description of A_BYPASS, B_BYPASS, and ECC_BYPASS signals in Table 2-2 and Table 6-1. Updated the polarity and description of R_ADDR_BYPASS and R_DATA_BYPASS signals in Table 2-17 and Table 6-10.
F	03/2023	<p>The following is the summary of changes in this revision of the document:</p> <ul style="list-style-type: none"> Updated the document title and added RT PolarFire information. In μPROM, modified the sentence to mention that fabric logic has read-only access to μPROM. Updated ECC Mode (For x33 Two-Port Mode Only) to include information about Multi-bit errors.

Table 7-1. Revision History (continued)

Revision	Date	Description
E	07/2022	The following is the summary of changes in this revision of the document: <ul style="list-style-type: none"> Added the μPROM port BUSY, see Figure 2-36 and Table 2-19. Added sNVM storage space requirements for fabric RAM initialization, see Implementation Added MEMFILE (RAM Content Manager output file)
D	03/2022	The following is the summary of changes in this revision of the document: <ul style="list-style-type: none"> In Content from File, added the supported memory file formats Corrected the real value PX+QY to PX-QY in Figure 3-12 Updated Dual-Port Large SRAM Configurator and Two-Port LSRAM Configurator for describing the LSRAM behavior when the Write Byte Enables option is selected. For more information, see CN20020.
C	12/2021	The following is the summary of changes in this revision of the document: <ul style="list-style-type: none"> Added information about Simple-Hex and Binary memory file formats, see Appendix: Supported Memory File Formats for LSRAM and μSRAM Added Write Port Width Alignment
B	08/2021	Added MPF for PolarFire FPGA and MPFS for PolarFire SoC FPGA in Introduction .
A	08/2021	The first publication of this document. This user guide was created by merging the following documents: <ul style="list-style-type: none"> UG0680: PolarFire FPGA Fabric User Guide UG0912: PolarFire SoC FPGA Fabric User Guide For more information, see Table 7-2 and Table 7-3 respectively.

The following revision history table describes the changes that were implemented in the UG0680: PolarFire FPGA Fabric User Guide document. The changes are listed by revision.

Note: UG0680: PolarFire FPGA Fabric User Guide document is now obsolete and the information in the document has been migrated to PolarFire® FPGA and PolarFire SoC FPGA Fabric User Guide.

Table 7-2. Revision History of UG0680: PolarFire FPGA Fabric User Guide

Revision	Date	Description
Revision 7.0	04/2021	<ul style="list-style-type: none"> Updated the Read Operation in Dual-Port Mode figure to correct the output data values in the Pipeline Mode. Removed collision prevention from the LSRAM, μSRAM, μPROM, and sNVM Features table. Updated the Byte Write Enables Settings for Dual-Port Mode table 8 for 1K x 16 mode. Added information about how RAM blocks are cascaded when Write byte Enables option is selected. Updated μPROM Operation to mention that μPROM memory file supports only the plain text file. Updated the Simplified Functional Block Diagram of LSRAM in Dual-Port Mode figure and Simplified Functional Block Diagram for LSRAM in Two-Port Mode figure 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.
Revision 6.0	04/2020	Updated information for for x33 Two-Port Mode Only in ECC mode.

Table 7-2. Revision History of UG0680: PolarFire FPGA Fabric User Guide (continued)

Revision	Date	Description
Revision 5.0	04/2019	<ul style="list-style-type: none"> Structural changes were made throughout the document. Information about PolarFire LSRAM, μSRAM, μPROM, and sNVM Features were updated in the LSRAM, μSRAM, μPROM, and sNVM Features table. Math Block Features were updated. Libero SoC PolarFire Compile Report is moved to appendix.
Revision 4.0	03/2018	Updated the Math Blocks Resources in the Fabric Resources in PolarFire Family table.
Revision 3.0	11/2017	Revision 3.0 of this document is updated to include features and enhancements introduced in Libero SoC PolarFire v2.0.
Revision 2.0	06/2017	<ul style="list-style-type: none"> Added reference to the ChipPlanner user guide. Added reference to the Synplify Pro RAM block application note for LSRAM. Added reference to the Synplify Pro RAM block application note for μSRAM. Added reference to the Synplify Pro MACC block application note for MACC.
Revision 1.0	02/2017	The first publication of the document.

The following revision history table describes the changes that were implemented in the UG0912: PolarFire SoC FPGA Fabric User Guide document. The changes are listed by revision.

Note: UG0912: PolarFire SoC FPGA Fabric User Guide document is now obsolete and the information in the document has been migrated to PolarFire® FPGA and PolarFire SoC FPGA Fabric User Guide.

Table 7-3. Revision History of UG0912: PolarFire SoC FPGA Fabric User Guide

Revision	Date	Description
Revision 2.0	04/2021	<ul style="list-style-type: none"> Updated the Read Operation in Dual-Port Mode figure to correct the output data values in the Pipeline Mode. Removed collision prevention from the Memory Blocks table. Updated the Byte Write Enables Settings for Dual-Port Mode table for 1K x 16 mode. Added information about how RAM blocks are cascaded when Write byte Enables option is selected. Updated μPROM Operation to mention that μPROM memory file supports only the plain text file. Updated the Simplified Functional Block Diagram of LSRAM in Dual-Port Mode figure and Simplified Functional Block Diagram for LSRAM in Two-Port Mode the figure 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.
Revision 1.0	04/2020	The first publication of the document.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at <https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.