UG0445
User Guide
SmartFusion2 SoC FPGA and IGLOO2 FPGA Fabric

a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email:FPGA_marketing@microchip.com
www.microchip.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

50200445 . 11.0 6/25

Contents

Figures

Tables

1 Revision History

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

1.1 Revision 11.0

Updated Figure 63, page 108 by changing the MSIO Pads voltage to 3.3V.

1.2 Revision 10.0

Updated Low Voltage CMOS (LVCMOS), page 89.

1.3 Revision 9.0

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

- Removed Sub-LVDS related information.
- Updated to highlight that Mini-LVDS is only supported for MSIOD 2.5V, see Table 57, page 88.
- Added a note on how to set the ODT value when multiple ODT values are supported for an I/O standard, see Receiver ODT Configuration, page 96.

1.4 Revision 8.0

The following is a summary of the changes in revision 8.0 of this document.

- Updated information in the note read failure violation of the Block SRAM timing. For more information see, Table 14, page 25.

1.5 Revision 7.0

The following is a summary of the changes in revision 7.0 of this document.

- Updated information about RADDR Synchronizer circuit. For more information see,Figure 14, page 26 and Operating Modes, page 24.
- Updated Table 64, page 98. For more information see, Receiver ODT Configuration, page 96.
- Updated recommendation for A_BLK[1:0],B_BLK [1:0],and C_BLK signals. For more information see, A_BLK[1:0], B_BLK [1:0], and C_BLK [1:0], page 43.
- Updated Synchronous Read Mode, page 45. For more information, see Operating Modes, page 45.

1.6 Revision 6.0

The following is a summary of the changes in revision 6.0 of this document.

- Updated SSO Guidelines to Simultaneous Switching Noise. For more information, see Simultaneous Switching Noise, page 82.
- Added a table to provide the status of the V_{REF} pin when assigned to P-side of the pair. For more information, see Table 59, page 90.

1.7 Revision 5.0

The following is a summary of the changes in revision 5.0 of this document.

- Added 060 device information.
- Updated Figure 2, page 6. For more information, see Fabric Architecture, page 4.
- Updated Logic Element, page 6. For more information, see Fabric Architecture, page 4.
- Added note to Two-Port Mode, page 22. For more information, see LSRAM, page 14.
- Updated A_DOUT[17:0] and B_DOUT[17:0], page 19 with the unconnected information. For more information, see LSRAM, page 14.
- Updated Table 7, page 17. For more information, see LSRAM, page 14.
- Updated Table 44, page 63. For more information, see Math Blocks, page 57.

- Added Input Reference Voltage, page 89. For more information, see I/Os, page 78.
- Added 3.3V Input Tolerance in 2.5V MSIOD/DDRIO Banks, page 105. For more information, see I/Os, page 78.
- Added Simultaneous Switching Noise, page 82. For more information, see I/Os, page 78.
- Updated table note Table 64, page 98. For more information, see I/Os, page 78.

1.8 Revision 4.0

The following is a summary of the changes in revision 4.0 of this document.

- Updated Supported I/O Standards, page 88. For more information, see I/Os, page 78.
- Updated I/O Programmable Features, page 91 with ODT, Driver impedance, and other features. For more information, see I/Os, page 78.
- Updated Figure 47, page 81 for DDRIO. For more information, see I/Os, page 78.
- Added Internal Clamp Diode, page 103. For more information, see I/Os, page 78.

1.9 Revision 3.0

The following is a summary of the changes in revision 3.0 of this document.

- Merged the SmartFusion2 SoC and IGLOO2 FPGA Fabric user guide.
- Removed all instances of and references to M2GL100 device from Table 1, page 5 and Table 3, page 13. For more information, see Fabric Architecture, page 4.
- Removed all instances of and references to M2GL100 device from Table 4, page 14 and Table 3, page 13. For more information, see LSRAM, page 14.
- Removed all instances of and references to M2GL100 device from Table 24, page 38. For more information, see Micro SRAM (µSRAM), page 38.
- Removed all instances of and references to M2GL100 device from Table 42, page 57. For more information, see Math Blocks, page 57.
- Updated Table 18, page 30. For more information, see LSRAM, page 14.
- Updated Micro SRAM (µSRAM), page 38.
- Updated Math Blocks, page 57.
- Updated Introduction, page 78 and Functional Description, page 78. For more information, see I/Os, page 78.
- Updated Figure 46, page 79. For more information, see I/Os, page 78.
- Updated Table 57, page 88 and Table 60, page 91. For more information, see I/Os, page 78.
- Updated Programmable Slew-Rate Control, page 92 and Table 63, page 95. For more information, see I/Os, page 78.
- Updated Receiver ODT Configuration, page 96. For more information, see I/Os, page 78.
- Updated 5V Input Tolerance and Output Driving Compatibility (only MSIO), page 106. For more information, see I/Os, page 78.
- Updated I/O Banks, page 82. For more information, see I/Os, page 78.
- Updated the Receive Buffer, page 80 for DDR support in low power devices. For more information, see I/Os, page 78.
- Added the Sub-LVDS information.
- Added Solution 3, page 107 for 5 V input tolerance section. For more information, see I/Os, page 78.

1.10 Revision 2.0

The following is a summary of the changes in revision 2.0 of this document.

- Updated Introduction, page 4, Architecture Overview, page 6, and Table 3, page 13. For more information, see Fabric Architecture, page 4.
- Updated Figure 36, page 58 and Coding Style Examples, page 73. For more information, see Math Blocks, page 57.
- Updated Introduction, page 78, I/O Banks, page 82, Low-Power Signature Mode and Activity Mode, page 104, Table 60, page 91, and Table 75, page 109. For more information, see I/Os, page 78.

1.11 Revision 1.0

The following is a summary of the changes in revision 1.0 of this document.

- Updated Figure 46, page 79, Figure 51, page 92, Figure 52, page 93. For more information, see I/Os, page 78.
- Updated B-LVDS/M-LVDS, page 91. For more information, see I/Os, page 78.
- Updated 5V Input Tolerance and Output Driving Compatibility (only MSIO), page 106. For more information, see I/Os, page 78.
- Updated SerDes I/O Pins, page 112. For more information, see I/Os, page 78.

1.12 Revision 0.0

Revision 0.0 was the first publication of this document.

2 Fabric Architecture

2.1 Introduction

SmartFusion®2 SoC FPGA and IGLOO2 FPGA fabric comprises an array of logic blocks and embedded hard blocks such as large static random access memory (LSRAM), micro SRAM (µSRAM), and math blocks for digital signal processing (DSP) capability. These elements are arranged as several rows inside the fabric, interconnected by the clustered routing architecture of the SmartFusion2 and IGLOO2 device. Each element in the fabric has a distinct logical coordinate value assigned to it. Figure 1, page 5 shows the simple layout of the SmartFusion2 and IGLOO2 fabric architecture.

Three types of resources constitute the major part of the fabric logic blocks:

- Logic elements
- Interface logic elements
- I/O modules

The logic element is the basic element used for implementing the combinatorial circuits, arithmetic functions, and sequential circuits inside the fabric. Each logic module consists of a 4-input LUT, a D-flip-flop, and a dedicated carry chain.

The interface logic is the logic element that interfaces the embedded hard blocks to the fabric routing. The interface logic enables the accessibility of the embedded hard block through the fabric routing. The interface logic is structurally similar to the logic element except that it does not contain the dedicated carry chain. The interface logic can also be used to implement the combinatorial and sequential circuits, if the associated embedded hard block is not being used by the design.

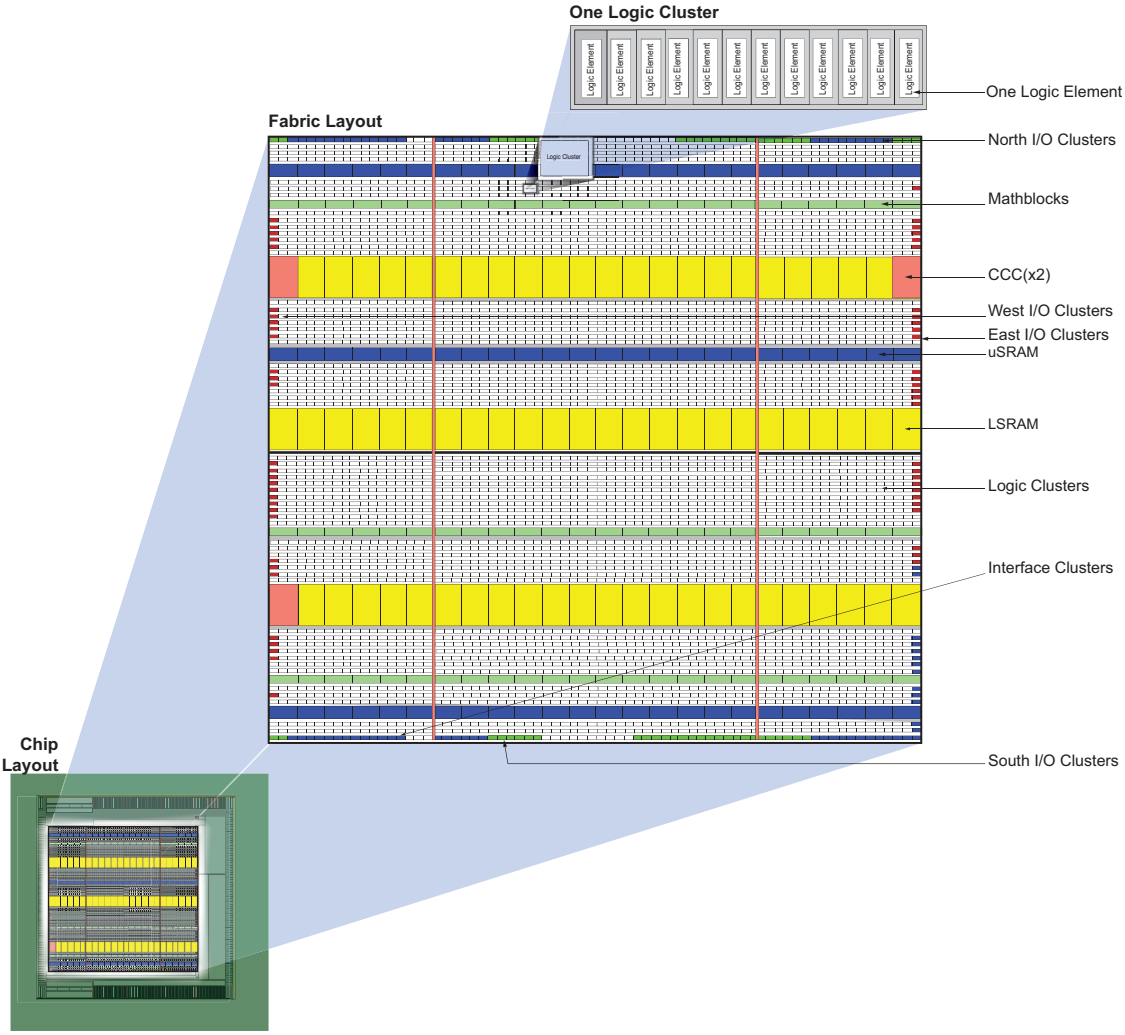
The I/O module forms the digital part of the fabric user I/Os, also called as multi-standard inputs/outputs (MSIOs). The I/O module enables the user I/Os to be connected to the fabric routing.

The SmartFusion2 and IGLOO2 fabric use a clustered routing architecture to interconnect the various elements of the fabric. In clustered architecture, various logic elements are grouped together to form the clusters. The SmartFusion2 and IGLOO2 fabric has three types of clusters:

- Logic clusters
- Interface clusters
- I/O clusters

The logic cluster is composed of 12 logic elements; the interface cluster is composed of 12 interface logic elements. I/O clusters are composed of 3 to 4 I/O modules, which are distributed on four sides of the device, as shown in the following figurer (north, south, east, and west I/O clusters).

Figure 1 • **SmartFusion2/IGLOO2 Fabric Architecture for M2S050/M2GL050**



2.2 Fabric Resources

The following tables list the fabric resources available on SmartFusion2 and IGLOO2 devices.

Table 1 • **Fabric Resources for SmartFusion2 Devices**

Fabric Resource	M2S005	M2S010	M2S025	M2S050	M2S060	M2S090	M2S150
Logic elements (4-input LUT + Flip-Flop)	6,060	12,084	27,696	56,340	56,520	86,316	146,124
LSRAM 18K blocks	10	21	31	69	69	109	236
µSRAM 1K blocks	11	22	34	72	72	112	240
Math blocks	11	22	34	72	72	84	240
PLLs and CCCs	2	2	6	6	6	6	8

Table 2 • **Fabric Resources for IGLOO2 Devices**

Fabric Resource	M2GL005	M2GL010	M2GL025	M2GL050	M2GL060	M2GL090	M2GL150
Logic elements (4-input LUT + Flip-Flop)	6,060	12,084	27,696	56,340	56,520	86,316	146,124
LSRAM 18K blocks	10	21	31	69	69	109	236
µSRAM 1K blocks	11	22	34	72	72	112	240
Math blocks	11	22	34	72	72	84	240
PLLs and CCCs	2	2	6	6	6	6	8

2.3 Architecture Overview

The following sections of this chapter describe the SmartFusion2 and IGLOO2 fabric architecture in detail.

- Logic Element
- Interface Logic Element
- I/O Module
- FPGA Routing Architecture

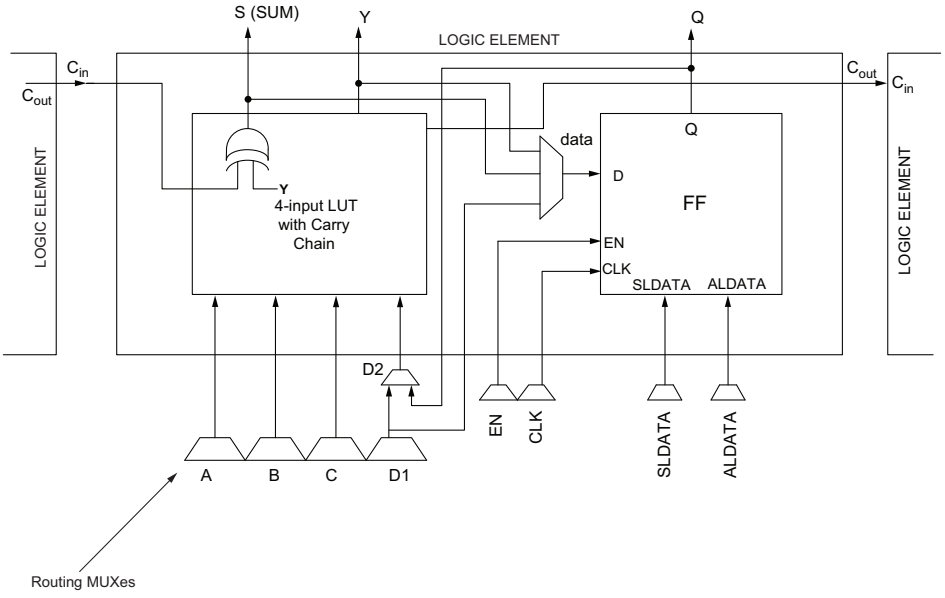
2.3.1 Logic Element

The logic elements can be used as a combinational logic element (CLE), and/or sequential logic element (SLE) in the design. Each logic element consists of:

- A 4-input LUT
- A dedicated carry chain based on the carry look-ahead technique
- A separate flip-flop which can be used independently from the LUT

The following illustration shows the functional block diagram of the logic element with carry chain.

Figure 2 • **Functional Block Diagram of Logic Element**



The 4-input LUT can be configured to implement any 4-input combinatorial function or to implement an arithmetic function, where the LUT output is XORed with carry input (Cin) to generate the sum (S) output. The sum output, S, is typically used as an output for arithmetic functions but can also be used as an output for logical functions along with the other output, Y, when the LUT is used to implement combinatorial functions.

Each logic element has a dedicated 3-bit look-ahead carry implementation, which is used to implement a dedicated carry chain between the logic elements when the LUT is used to implement arithmetic operations.

The carry chain has hardwired routing nets running between the logic elements, which reduces the carry propagation delay through the carry chain, thus giving better performance. The logic element also contains a dedicated flip-flop, which can be used in conjunction with or independently from the LUT. The flip-flop can be configured as a register or latch. It has asynchronous and synchronous load and clock enable inputs. Asynchronous load signal (ALDATA) can be used as asynchronous set or reset signal of each fabric D flip-flops. It sets or resets the register depending on configuration. Synchronous load signal (SLDATA) can be used as synchronous set or reset signal of each fabric D flip-flop. It sets or resets the register depending on configuration. The data input of the flip-flop can be fed from the direct input (D1) or from the outputs of the 4-input LUT inside the logic element.

2.3.2 Interface Logic Element

Embedded hard blocks (LSRAM blocks, µSRAM blocks, and math blocks) contain a dedicated interface logic. The embedded hard blocks are connected to the fabric routing structure through LUTs and flip-flops on their inputs and outputs, and these together form the interface logic element.

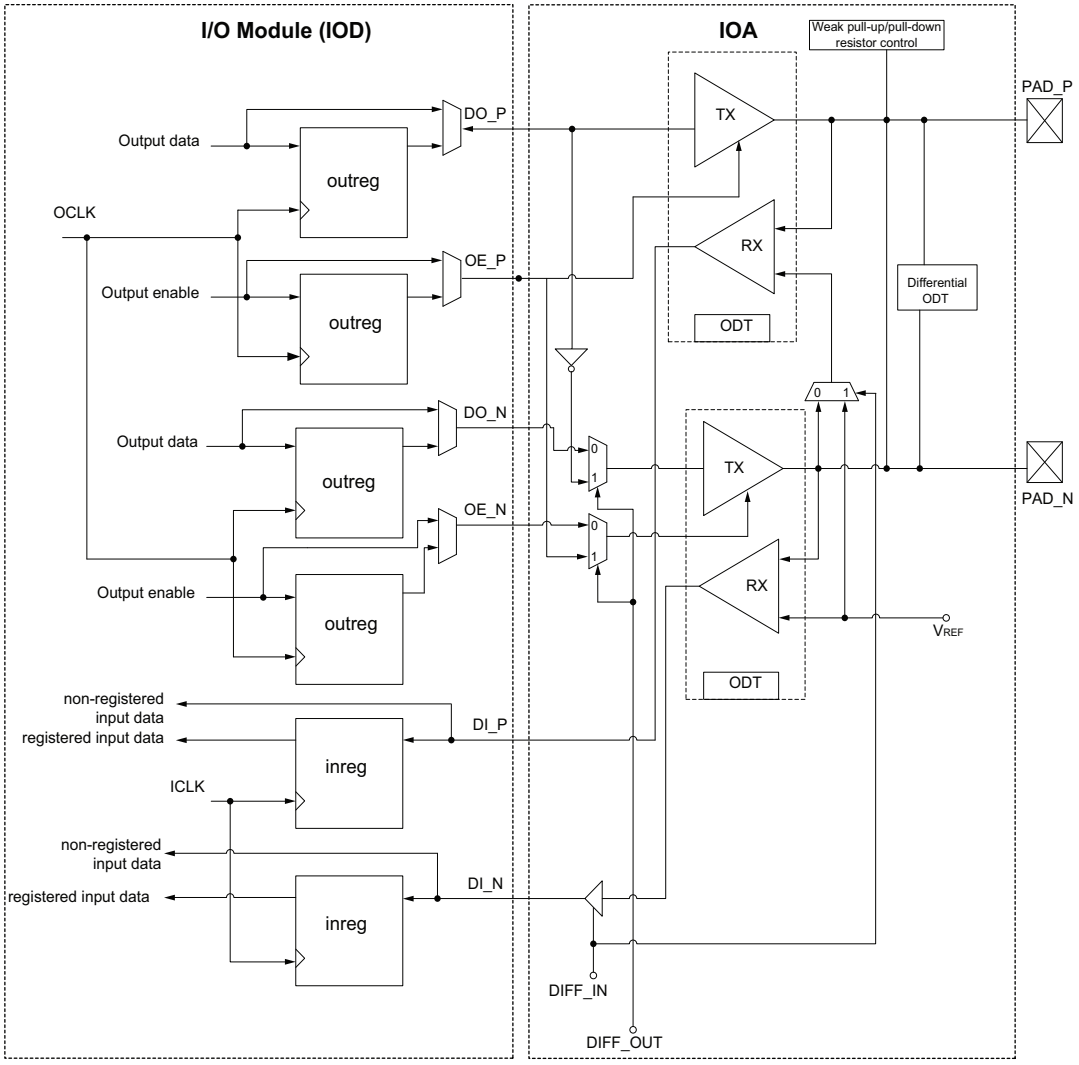
Each embedded hard block is associated with 36 interface logic elements. This interface logic element is structurally equivalent to a logic element but does not have a dedicated carry chain. When a given embedded hard block is used by the target design, the interface logic is used to connect the embedded hard block's I/Os to the fabric routing. If an embedded hard block is not used by the design, the interface logic element is available for use as a normal logic elements for implementing combinatorial and sequential circuits. These are in addition to the logic elements available in the fabric.

2.3.3 I/O Module

The I/O module includes the I/O digital (IOD) circuitry and the associated routing interface. Each user I/O pad is connected to its own dedicated I/O module. The I/O module interfaces the user I/Os with the fabric routing and enables the routing of external signals coming in through the I/Os to reach all the logic elements. The I/O modules also enable the internal signals to reach the I/Os.

The following illustration shows the functional diagram of the complete MSIO with the IOD and I/O analog (IOA) sections. The IOD consists of the input registers, output registers, output enable registers, and routing multiplexers (MUXes). The output register provides the registered version of the output signals to the I/Os. In the same way, the input registers are used to register the inputs received from the I/Os. The output enable acts as a control signal for the output, if the I/O is configured as a tristated or bidirectional I/O. These registers in the I/O modules are similar to the D-flip-flops available in the logic element. The usage of the output registers in the I/O modules for registering the output signals at I/Os enables better design performance. Also, in the case of a signal bus, these registers ensure that all the bits of the signal bus are synchronized to the clock signal when being sent out through the I/Os. At the input side, the input registers allow capturing the input signals and synchronizing them to the design clock.

Figure 3 • **Functional Block Diagram of MSIO**



2.3.4 FPGA Routing Architecture

The SmartFusion2 and IGLOO2 fabric has a clustered routing architecture. Clustering is a hierarchical grouping of fabric resources that allows a more area-efficient implementation of designs while maintaining optimal performance. It also helps in reducing the run-time of the place-and-route software.

The SmartFusion2 and IGLOO2 fabric routing architecture is composed of three types of clusters:

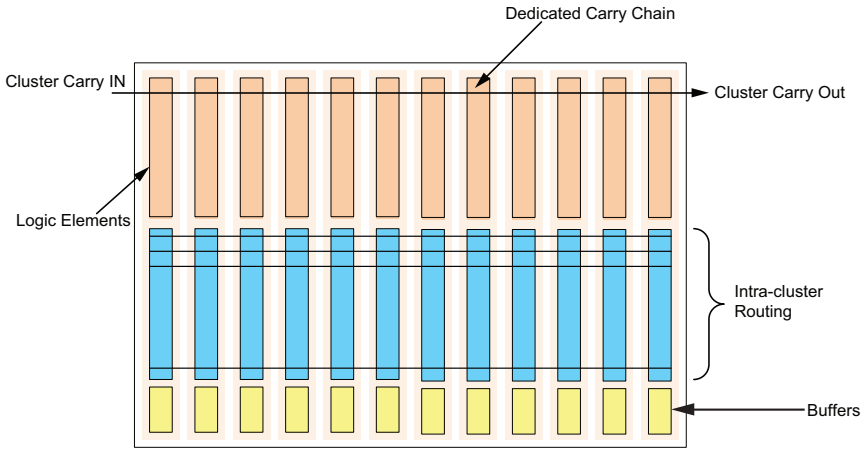
- Logic Cluster
- Interface Cluster
- I/O Cluster

2.3.4.1 Logic Cluster

The logic cluster is a combination of 12 logic elements with a dedicated hardwired carry chain implemented for all 12 logic elements. The logic clusters contain routing MUXes. Each routed signal is driven by a unique logic element output or routing MUX. All the logic elements are interconnected with feedback from outputs to inputs. The intra-routing inside the logic clusters has a very low propagation delay as compared to the routing outside the logic clusters.

Each LUT, D-flip-flop, and the carry-circuit in the logic cluster have an individual X-Y logical coordinate assigned, and this makes them independently addressable. The following illustration shows the top-level logic cluster layout diagram.

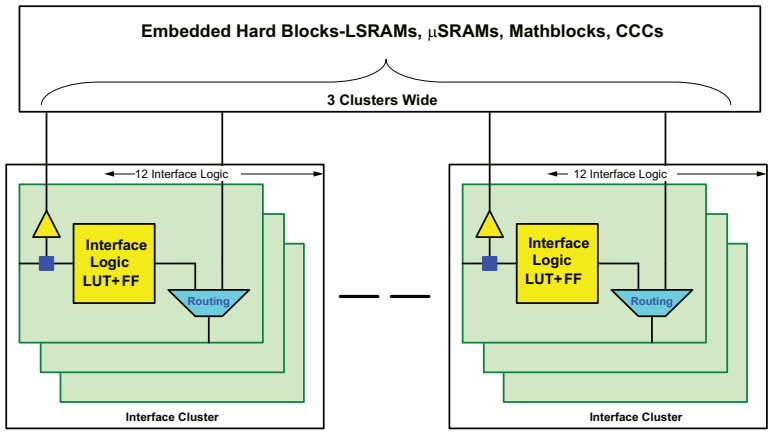
Figure 4 • **Logic Cluster Top-Level Layout**



2.3.4.2 Interface Cluster

The interface cluster is similar to the logic cluster except that it is a combination of 12 interface logic elements. These clusters are used to interface the inputs and outputs of the embedded hard blocks (LSRAM, µSRAM, math blocks, and CCCs) to fabric routing. Each embedded hard block is spanned by 3 interface clusters, as shown in the following figure. The interface logic can be used as a logic elements (without carry chain) when the associated embedded hard block is not used by the design.

Figure 5 • **Interface Cluster**



2.3.4.3 I/O Cluster

I/O clusters are combinations of I/O modules and the associated routing interfaces. The north and south I/O clusters each contain four I/O modules. The east and west I/O clusters, each contain three I/O modules. Each I/O pad is associated with its own dedicated I/O module.

2.3.4.4 Routing Structure

The routing of any design is completed automatically by the software, thus, the utilization of the routing resources is completely transparent to the user. The selection among various routing resources by the placement-and-routing software is impacted by the design constraints provided. For more details on how to use the constraints using Libero SoC software, see *SmartTime User Guide*, *I/O Editor User Guide*, and *ChipPlanner User Guide* on the *Libero SoC Documentation* page.

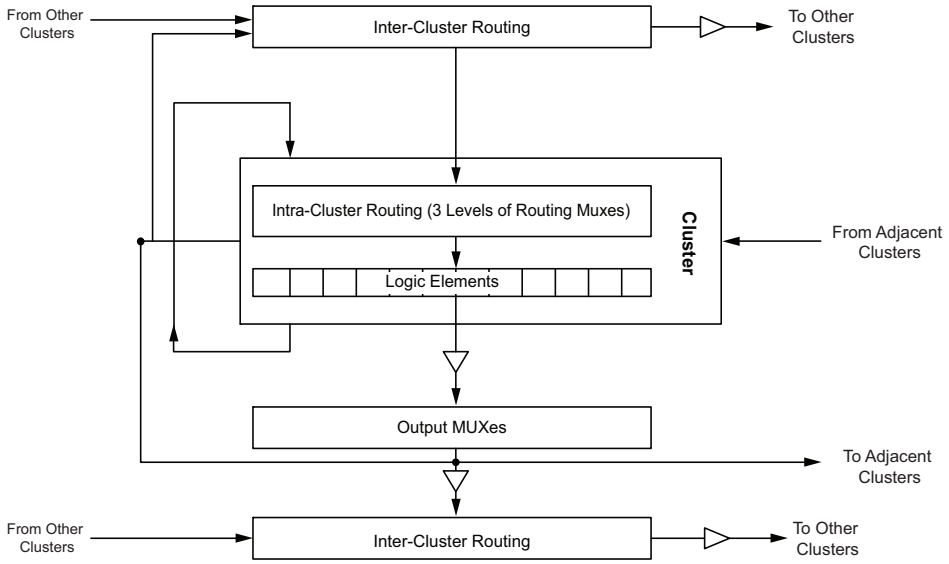
Knowledge of the routing architecture and functional modules can be useful in providing effective design constraints to the software, so that it can be guided to do an optimal design implementation on the SmartFusion2 and IGLOO2 fabric.

In the SmartFusion2 and IGLOO2 device, the fabric routing is segregated into two parts:

- Inter-cluster routing
- Intra-cluster routing

The following illustration shows the fabric routing structure for the SmartFusion2 and IGLOO2 device.

Figure 6 • **Fabric Routing Structure**



Inter-cluster routing spans the clusters and connects them together. The inter-cluster routing resource is common to all the clusters inside the fabric and is universal across the clusters.

Intra-cluster routing spans the modules that constitute a cluster. Intra-cluster routing is not unique and varies from cluster to cluster, depending upon the functionality of the cluster. For example, the intra-cluster routing for an interface cluster is different from that of a logic cluster. There are differences in the routing of the various interface clusters, depending upon the embedded hard block to which they interface.

Inter-cluster routing and intra-cluster routing are completely separate. Inter-cluster routing never drives the inputs of the functional modules (logic elements, interface logic elements, or I/O modules) directly and the outputs of the functional modules do not drive the inter-cluster routing directly. Inter-cluster routing has to pass through the intra-cluster routing to reach the functional modules. That makes SmartFusion2 and IGLOO2 routing a fully clustered routing architecture.

The global network can also drive intra-cluster routing through special routing MUXes. These global routing MUXes bring in flip-flop control signals such as clock, enable, and sets/resets.

There are a few short routing lines between the adjacent clusters and between the inter-cluster and intra-cluster routing MUXes. These short paths are provided to provide better performance to the signals routed through these lines.

2.4 Fabric Array Coordinate System

Every element in the SmartFusion2 and IGLOO2 fabric has individual logical X-Y coordinates associated with the fabric array coordinate system. These logical coordinates are used by the place-and-route software while implementing the design using the fabric elements. The place-and-route software can be constrained to occupy the design components in specific locations inside the fabric using this coordinate system. Regions can be created inside the fabric and a particular part of the design can be assigned to that region using the Libero SoC floor-planner software.

The boundaries of these regions can be specified using the array coordinates. Similarly, the embedded hard block is also addressable through the fabric coordinate system.

The array coordinates are measured from the bottom-left corner to the top-right corner of the FPGA fabric. Table 3, page 13 provides the array coordinates of logical modules and embedded hard blocks of SmartFusion2 and IGLOO2 devices. Figure 7, page 11, Figure 8, page 12, and Figure 9, page 12 show the array coordinates of an M2S050/M2GL050, M2S060/M2GL060, M2S025/M2GL025, and M2S010/M2GL010 devices. For more information on how to use array coordinates for region/placement constraints, see *Libero SoC Design Flow User Guide* or online help (available in the software) for SmartFusion2 and IGLOO2 Libero SoC tools.

Figure 7 • **M2S050/M2GL050 and M2S060/M2GL060 Fabric Logical Coordinates**

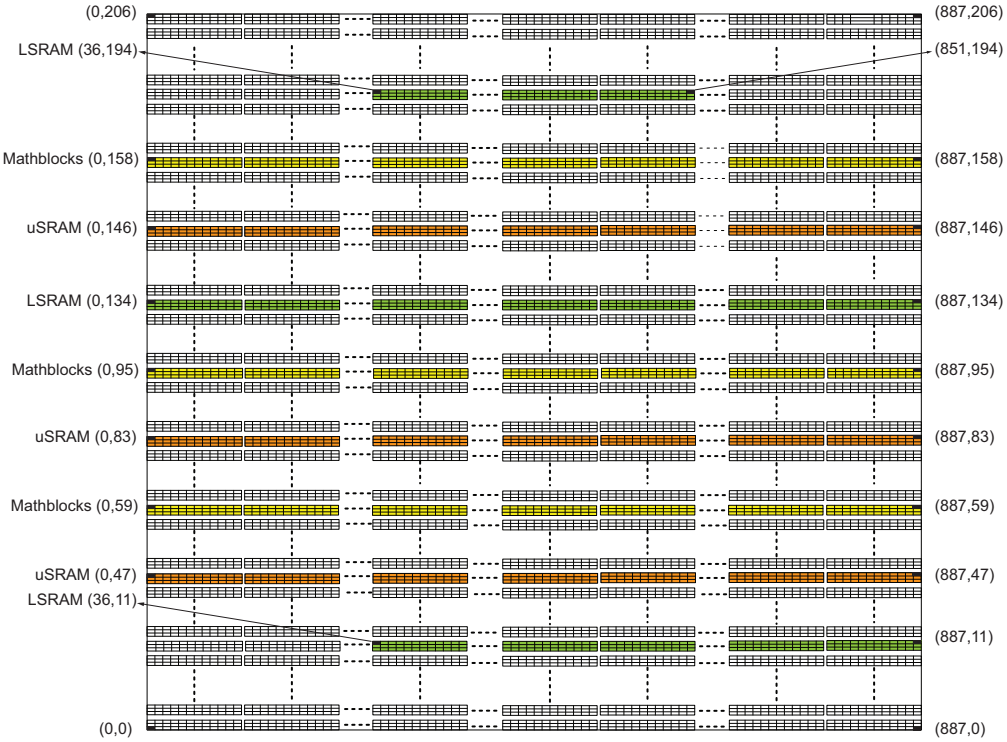


Figure 8 • **M2S025/M2GL025 Fabric Logical Coordinates**

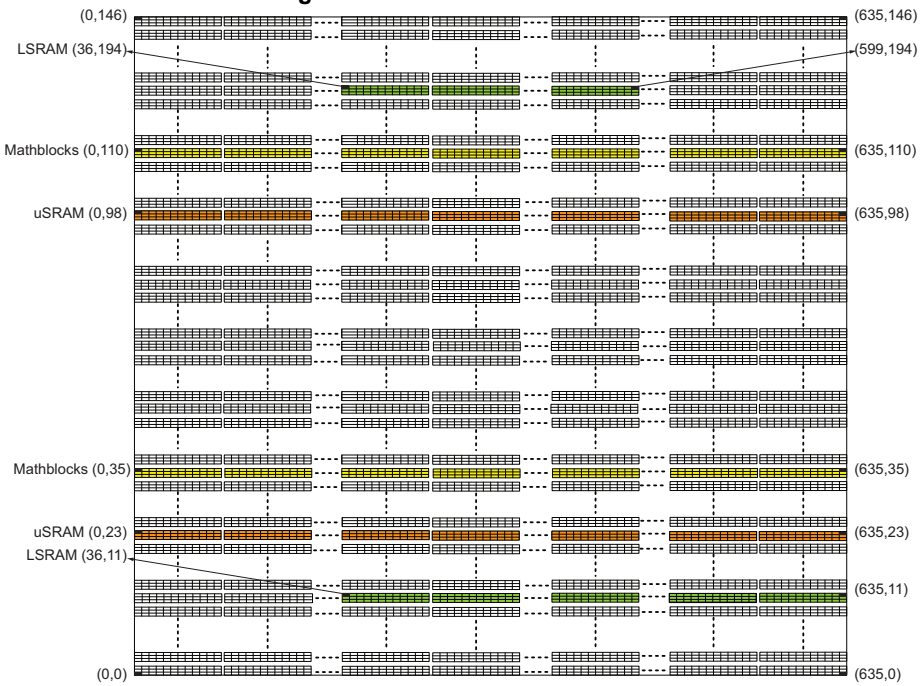


Figure 9 • **M2S010/M2GL010 Fabric Logical Coordinates**

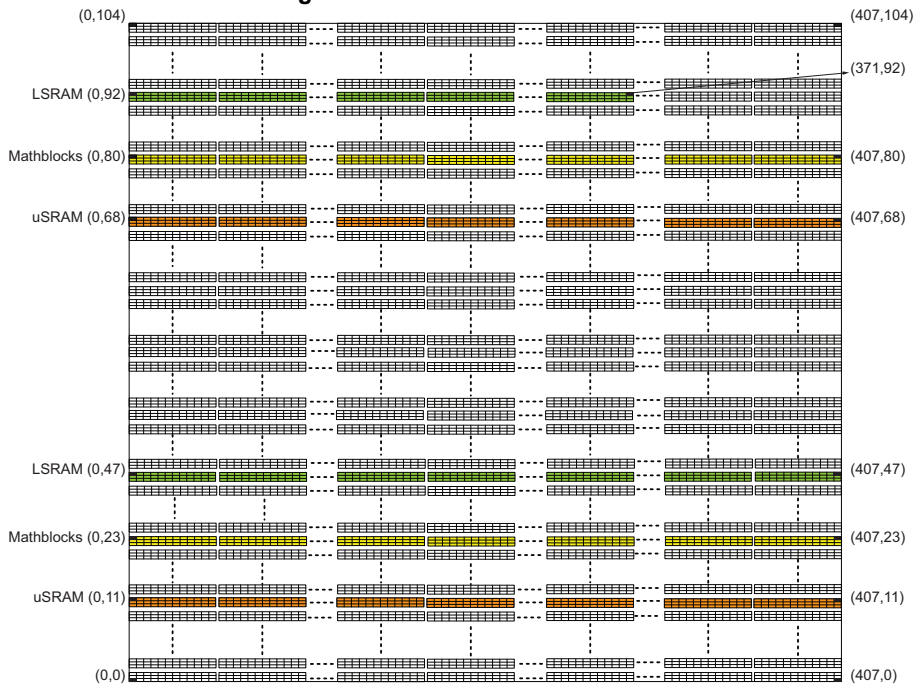


Table 3 • **Fabric Array Coordinate Systems**

	Logic Elements				µSRAM			LSRAM			Math Blocks		
	Min		Max		Bottom	Middle	Top	Bottom	Middle	Top	Bottom	Middle	Top
Device	**X**	**Y**	**X**	**Y**	**(X,Y)**	**(X,Y)**	**(X,Y)**	**(X,Y)**	**(X,Y)**	**(X,Y)**	**(X,Y)**	**(X,Y)**	**(X,Y)**
M2S005 M2GL005	0	0	407	56	NA	NA	(0,11)	NA	NA	(0,44)	NA	NA	(0,23)
M2S010 M2GL010	0	0	407	104	(0,11)	NA	(0,68)	(0,47)	NA	(0,92)	(0,23)	NA	(0,80)
M2S025 M2GL025	0	0	635	146	(0,23)	NA	(0,98)	(36,11)	NA	(36,134)	(0,35)	NA	(0,110)
M2S050 M2GL050	0	0	887	206	(0,47)	(0,83)	(0,146)	(36,11)	(0,134)	(36,194)	(0,59)	(0,95)	(0,158)
M2S060 M2GL060	0	0	887	206	(0,47)	(0,83)	(0,146)	(36,11)	(0,134)	(36,194)	(0,59)	(0,95)	(0,158)
M2S090 M2GL090	0	0	1031	266	(0, 23)	(0, 59) (0, 194)	(0, 242)	(36, 11)	(0, 119) (0, 170)	(36, 254)	(0, 35)	(0, 71)	(0, 206)
M2S150 M2GL150	0	0	1463	314	(0, 35) (0, 59)	(0, 107) (0, 182)	(0, 230) (0, 278)	(36, 11) (0, 95)	(0, 143) (0, 218)	(0, 266) (36, 302)	(0, 47) (0, 71)	(0, 119) (0, 194)	(0, 242) (0, 290)

3 LSRAM

3.1 Introduction

The SmartFusion2 and IGLOO2 fabric has embedded 18 Kbit SRAM blocks used for storing data. These large SRAM blocks (LSRAMs) are arranged in multiple rows within the FPGA fabric and can be accessed through the fabric routing architecture. The number of LSRAM blocks available depends upon the specific SmartFusion2 and IGLOO2 device, as shown in the following table. For example, in the M2S050 or M2GL050 device, there are 69 LSRAM blocks available, which are spread across three rows inside the fabric.

3.1.1 Features

The SmartFusion2 and IGLOO2 LSRAM blocks have the following features:

- Each LSRAM block can store up to 18,432 bits of data and can be configured in any of the following depth x width combinations: 512 x 36, 512 x 32, 1k x 18, 1k x 16, 2k x 9, 2k x 8, 4k x 4, 8k x 2, or 16k x 1.
- Each LSRAM block contains two independent data ports—Port A and Port B.
- The LSRAM is synchronous for both read and write operations. These operations are triggered on the rising edge of the clock.
- Supports maximum frequency up to 400 MHz.
- An optional pipeline register is available at the read data port to improve the clock-to-out delay.
- LSRAM supports two types of read operations:
  - Flow-through read (or non-pipelined)
  - Pipelined read
- LSRAM supports two types of write operations:
  - Simple write
  - Feed-through write (write-bypass write)
- LSRAM can be operated in two memory modes:
  - Dual-port mode
  - Two-port mode
- A write operation requires one clock cycle.
- A read operation requires one clock cycle in Non-pipelined mode. In Pipelined mode, the output data appears in the next cycle.
- Read from both ports at the same location is allowed.
- Read and write on the same location at the same time is not allowed. There is no built in collision prevention or detection circuit in LSRAM.

3.2 LSRAM Resources

The following table lists LSRAM rows and 18K blocks available in SmartFusion2 and IGLOO2 devices.

Table 4 • **SmartFusion2 and IGLOO2 LSRAM (18Kb Blocks) Resource Table**

Device	M2S005/ M2GL005	M2S010/ M2GL010	M2S025/ M2GL025	M2S050/ M2GL050	M2S060/ M2GL060	M2S090/ M2GL090	M2S150/ M2GL150
Rows	1	2	2	3	3	4	6
LSRAM 18 K Blocks	10	21	31	69	69	109	236

Note: All numbers given above are per device.

3.3 Functional Description

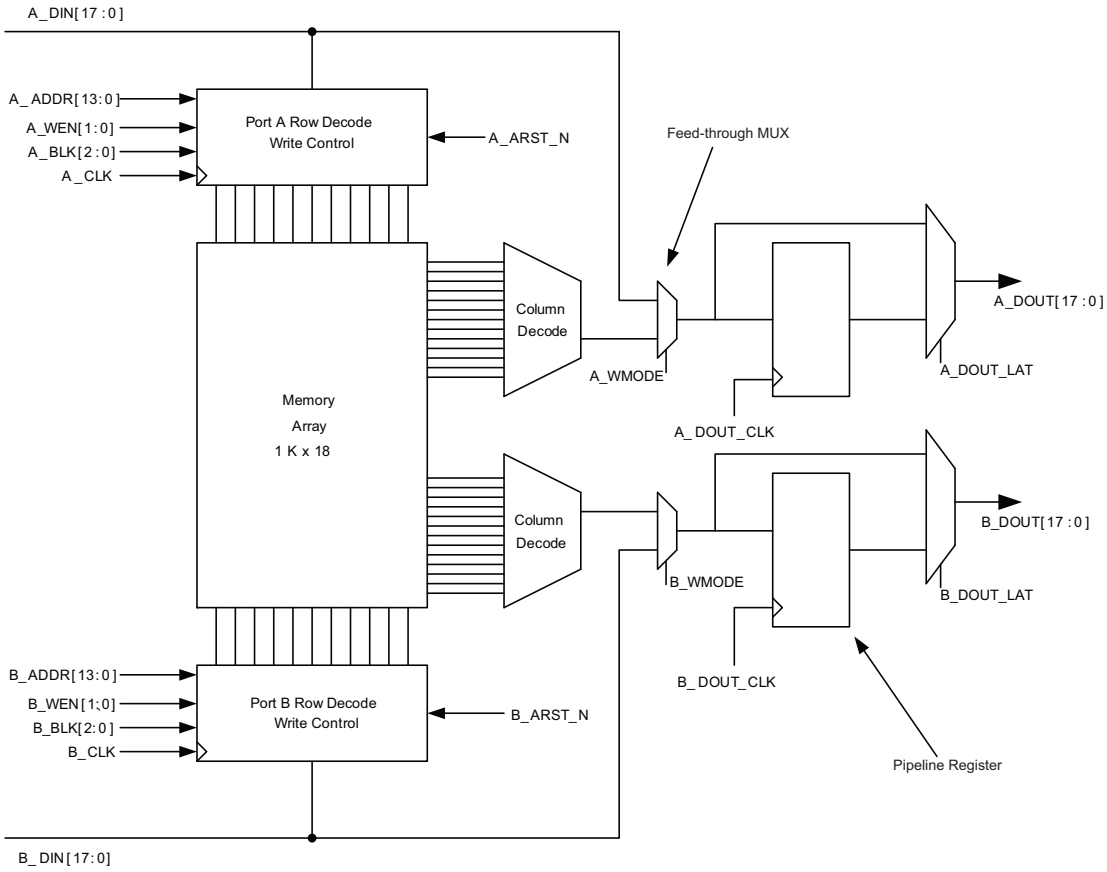
This section provides the detailed description of the following:

- Architecture Overview
- Port List
- Port Descriptions

Architecture Overview

SmartFusion2 and IGLOO2 LSRAM embedded memory includes the RAM1Kx18 macro. The following illustration shows a simplified block diagram of the LSRAM memory block and Table 5, page 16 provides the port descriptions. The following illustration shows two independent data ports, the pipeline registers for read data delay, and the feed-through multiplexers to enable immediate access to the write data.

Figure 10 • **Simplified Functional Block Diagram for LSRAM**



3.3.1 Port List

Table 5 • **Port List for LSRAM Macro (RAM1KX18)**

Port Name	Direction	Type[1]	Polarity	Description
PORT A				
A_WIDTH[2:0]	Input	Static		Port A Width/depth mode select
A_WEN[1:0]2	Input	Dynamic	High	Port A Write enable
A_ADDR[13:0]	Input	Dynamic		Port A Address input
A_DIN[17:0]	Input	Dynamic		Port A Data input
A_DOUT[17:0]	Output	Dynamic		Port A Data output
A_BLK[2:0]	Input	Dynamic	High	Port A Block select
A_WMODE	Input	Static	High	Port A Feed-through write select
A_CLK	Input	Dynamic	Rising	Port A Clock
A_ARST_N	Input	Dynamic	Low	Port A Asynchronous reset
A_DOUT_CLK	Input	Dynamic	Rising	Port A Pipeline register clock
A_DOUT_LAT	Input	Static	Low	Port A Pipeline register Select
A_DOUT_ARST_N	Input	Dynamic	Low	Port A Pipeline register asynchronous reset
A_DOUT_EN	Input	Dynamic	High	Port A Pipeline register enable
A_DOUT_SRST_N	Input	Dynamic	Low	Port A Pipeline register synchronous reset
PORT B				
B_WIDTH[2:0]	Input	Static		Port B Width/depth mode select
B_WEN[1:0][2]	Input	Dynamic	High	Port B Write enable
B_ADDR[13:0]	Input	Dynamic		Port B Address input
B_DIN[17:0]	Input	Dynamic		Port B Data input
B_DOUT[17:0]	Output	Dynamic		Port B Data output
B_BLK[2:0]	Input	Dynamic	High	Port B Block select
B_WMODE	Input	Static	High	Port B Feed-through write select
B_CLK	Input	Dynamic	Rising	Port B Clock
B_ARST_N	Input	Dynamic	Low	Port B Asynchronous reset
B_DOUT_CLK	Input	Dynamic	Rising	Port B Pipeline register clock
B_DOUT_LAT	Input	Static	Low	Port B Pipeline register select
B_DOUT_ARST_N	Input	Dynamic	Low	Port B Pipeline register asynchronous reset
B_DOUT_EN	Input	Dynamic	High	Port B Pipeline register enable
B_DOUT_SRST_N	Input	Dynamic	Low	Port B Pipeline register synchronous reset
Common Signals				
A_EN	Input	Static	Low	Port A power-down
B_EN	Input	Static	Low	Port B power-down
SII_LOCK	Input	Static	High	Lock access to SII
BUSY	Output	Dynamic	High	Busy signal from SII

1. Static inputs are defined at design time and can be or are controlled by flash configuration bits.

2. If LSRAM is configured in two-port mode with a write data width of x36/x32 and read data width of x36/x32, both the bits of A_WEN and B_WEN must be tied to logic 1 and should not be dynamically changed.

3.3.2 Port Descriptions

3.3.2.1 A_WIDTH[2:0] and B_WIDTH[2:0]

These signals represent the depth x width mode selections for each port. The following table shows the depth x width based on ports width selection.

Table 6 • **Depth/Width Mode Selection**

A_WIDTH/B_WIDTH	Depth/Width
000	16K x 1
001	8K x 2
010	4k x 4
011	2K x 9 2K x 8
100	1K x 18 1K x 16
101 110 111 (Two-port)	512 x 36 512 x 32

3.3.2.2 A_WEN[1:0] and B_WEN[1:0]

These signals represent the write enables for each port to select read/write operations. The following table shows the depth x width operations based on port write enable selection.

Table 7 • **Read/Write Operation Selection[1, 2]**

Depth x Width	A_WEN/B_WEN	Operation
16K x 1 8K x 2 4K x 4 2K x 8 2K x 9 1K x 16 1K x 18	00	Read operation
16K x 1 8K x 2 4K x 4 2K x 8 2K x 9 1K x 16 1K x 18	1	Write operation
512 x 32 (Two-port write-Port B)	A_WEN[1:0] = “11” B_WEN[1:0] = “11”	Write [31:0]
512 x 36 (Two-port write-Port B)	B_WEN[1:0] = “11” A_WEN[1:0] = “11”	Write [35:0]

1. In dual-port mode, every port reads when the corresponding write enable (A_WEN/B_WEN) is "00" and corresponding port select (A_BLK/B_BLK) is active.

2. In two-port mode, the read port (Port A) reads in every clock cycle if A_BLK is active.

3.3.2.3 A_ADDR[13:0] and B_ADDR[13:0]

These signals represent the address buses for the two ports. In x1 mode 14 bits are used to address the 16,384 independent locations. In wider modes (x2, x4, etc.) fewer address bits are used. The used address bits are the most significant bits (MSB). The unused bits are the least significant bits (LSBs) and they must be grounded. The following table shows the address bus used and unused bits for depth x width selections.

Table 8 • **Address Bus Used and Unused Bits**

	A_ADDR/B_ADDR	
Depth x Width	**Used Bits**	**Unused bits (to be grounded)**
16K x 1	[13:0]	None
8K x 2	[13:1]	[0]
4K x 4	[13:2]	[1:0]
2K x 9 2K x 8	[13:3]	[2:0]
1K x 18 1K x 16	[13:4]	[3:0]
512 x 36	[13:5]	[4:0]

3.3.2.4 A_DIN[17:0] and B_DIN[17:0]

These signals represent the data input buses for the two ports. In dual-port mode, the data width can range from 1 bit to 18 bits. In two-port mode, Port B becomes the write-only port. Giving a write data width of 36 bits, A_DIN[17:0] becomes write data[35:18] and B_DIN[17:0] becomes write data[17:0]. The used bits for any mode are LSB justified in the data bus and the unused MSB bits must be grounded. The following table shows the data input buses used and unused bits for depth x width selections.

Table 9 • **Data Input Buses Used and Unused Bits**

Depth x Width	**A_DIN/B_DIN**	
	Used Bits	**Unused bits (to be grounded)**
16K x 1	[0]	[17:1]
8K x 2	[1:0]	[17:2]
4K x 4	[3:0]	[17:4]
2K x 8	[7:0]	[17:8]
2K x 9	[8:0]	[17:9]
1K x 16	[16:9] is [15:8] [7:0] is [7:0]	[17] [8]
1K x 18	[17:0]	None
512 x 32	A_DIN[16:9] is [31:24] A_DIN[7:0] is [23:16] B_DIN[16:9] is [15:8] B_DIN[7:0] is [7:0]	A_DIN[17] A_DIN[8] B_DIN[17] B_DIN[8]
512 x 36	A_DIN[17:0] is [35:18] B_DIN[17:0] is [17:0]	None

3.3.2.5 A_DOUT[17:0] and B_DOUT[17:0]

These signals represent the data output buses for the two ports. In dual-port mode, the data width can range from 1 bit to 18 bits. In two-port mode, Port A becomes the read-only port. Giving a read data width of 36 bits, A_DOUT[17:0] becomes read data[35:18] and B_DOUT[17:0] becomes read data[17:0]. The used bits for any mode are LSB justified in the data bus and the unused MSB bits must be unconnected. The following table shows the data output buses used and unused bits for depth x width selections.

Table 10 • **Data Output Buses Used and Unused Bits**

	A_DOUT/B_DOUT	
Depth x Width	**Used Bits**	**Unused bits (unconnected)**
16K x 1	[0]	[17:1]
8K x 2	[1:0]	[17:2]
4K x 4	[3:0]	[17:4]
2K x 8	[7:0]	[17:8]
2K x 9	[8:0]	[17:9]
1K x 16	[16:9] is [15:8] [7:0] is [7:0]	[17] [8]
1K x 18	[17:0]	None
512 x 32	A_DOUT[16:9] is [31:24] A_DOUT[7:0] is [23:16] B_DOUT[16:9] is [15:8] B_DOUT[7:0] is [7:0]	A_DOUT[17] A_DOUT[8] B_DOUT[17] B_DOUT[8]
512 x 36	A_DOUT[17:0] is [35:18] B_DOUT[17:0] is [17:0]	None

3.3.2.6 A_BLK[2:0] and B_BLK[2:0]

These signals represent the port select control signals for each port. The following table shows operations (Read, Write, and No operation) based on selection of port select control signals.

Table 11 • **Port Select Control Signals**

Port Select Signal	Value	Result
A_BLK[2:0]	111	Perform read or write operation on Port A.
A_BLK[2:0]	000 001 010 011 100 101 110	No operation in memory from Port A. Port A output is forced to logic 0.
B_BLK[2:0]	111	Perform read or write operation on Port B.
B_BLK[2:0]	000 001 010 011 100 101 110	No operation in memory from Port B. Port B output is forced to logic 0.

3.3.2.7 A_WMODE and B_WMODE

These signals represent the Write mode control signals for Port A and Port B.

- Logic 0: Output data port holds the previous value.
- Logic 1: Feed-through; write data appears on the corresponding output data port. In two-port mode, feed-through write is not supported.

3.3.2.8 A_CLK and B_CLK

These signals represent the clock inputs for Port A and Port B. All inputs must be set up before the rising edge of the clock. The read or write operation begins with the rising edge.

3.3.2.9 A_ARST_N and B_ARST_N

These signals represent Active Low, asynchronous reset inputs for Port A and Port B. Assertion of these resets during read operation forces the data output lines to logic 0. Assertion of these resets during write operation results in garbage values written into the memory.

3.3.2.10 A_DOUT_ARST_N and B_DOUT_ARST_N

These signals represent Active Low, asynchronous reset inputs for the output pipeline registers for Port A and Port B. Assertion of these reset signals forces the data output to logic 0. In Non-pipelined mode, these inputs should be tied to logic 1.

3.3.2.11 A_DOUT_LAT and B_DOUT_LAT

These signals represent Latch mode inputs for the output pipeline registers for Port A and Port B.

- Logic 0: Register operation
- Logic 1: Latch operation

3.3.2.12 A_DOUT_EN and B_DOUT_EN

These signals represent Active High; enable inputs for the output pipeline registers for Port A and Port B.

- Logic 1: Normal register operation
- Logic 0: Register holds previous data

3.3.2.13 A_DOUT_SRST_N and B_DOUT_SRST_N

These signals represent Active Low, synchronous reset inputs for the output pipeline registers for Port A and Port B. Assertion of these reset signals forces the data output to logic 0. In Non-pipelined mode, these inputs should be tied to logic 1.

3.3.2.14 A_EN and B_EN

These are Active Low, power-down configuration bits for each port.

3.3.2.15 SII_LOCK

This control signal, when asserted to logic 1, locks the entire LSRAM memory for being accessed by the system controller interface bus (SII). The system controller can access the LSRAM for the following purposes:

- Testing the memory
- Moving data between LSRAM and embedded nonvolatile memory (eNVM) or external memories
- Moving data between various LSRAMs or between µSRAMs and LSRAMs
- LSRAMs cannot be accessed when the system controller is accessing them

3.3.2.16 BUSY

This signal acts as a Status signal when the system controller is accessing the particular LSRAM. Logic 1 on this signal indicates system controller access. This signal can be used to monitor the completion of LSRAM access.

3.4 Memory Modes

LSRAM can be configured as a dual-port SRAM or two-port SRAM.

3.4.1 Dual-Port Mode

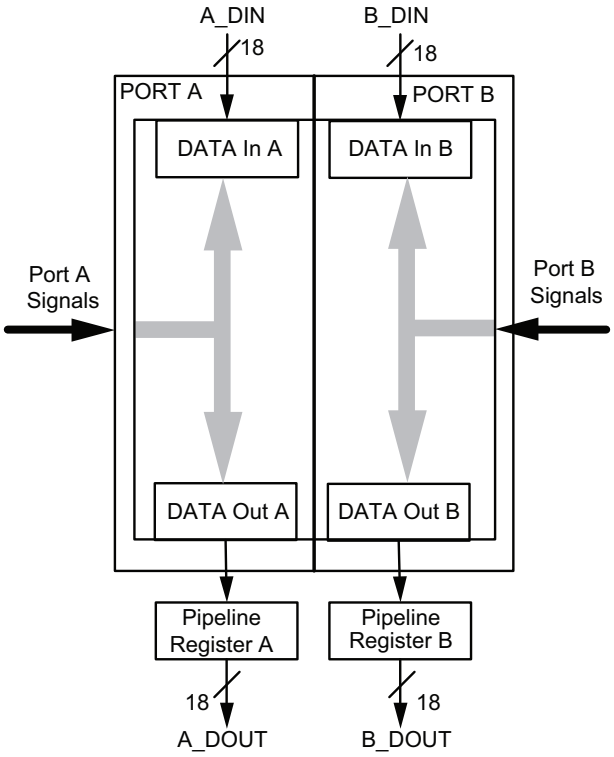
LSRAM configured as dual-port SRAM provides a data storage capability of 18 Kbits with two independent access ports: Port A and Port B, as shown in the following illustration. Read and write operations can be done from both the ports independently at any location as long as there is no collision.

In dual-port mode, the maximum data width can be x18 for either port. In dual-port mode, each port of the LSRAM can be configured in the following depth x width configurations:

- 1k x 18, 1k x 16
- 2k x 9, 2k x 8
- 4k x 4
- 8k x 2
- 16k x 1

The following illustration shows the data path for the dual-port SRAM (DPSRAM).

Figure 11 • **Data Path for Dual-Port Mode**



Data can be written to either or both ports and also can be read from either or both ports. Each port has its own address, data in, data out, clock, block select, and write enable. The read and write operations are synchronous and require a clock edge.

There is no collision detection or prevention circuit built into LSRAM. Simultaneous write operations from both the ports to the same address location result in data uncertainty. Simultaneous read and write operations from both the ports to the same address location results in correct data written into the memory but garbage values being read out.

The read operation requires one clock cycle in Non-pipelined mode. In Pipelined mode, the output data appears in the next cycle. The write operation requires one clock cycle.

When the read operation is configured with output pipeline registers, the input clock sourcing the pipeline registers has to be synchronized to the LSRAM's clock input; that is, A_DOUT_CLK should be synchronized to A_CLK and B_DOUT_CLK should be synchronized to B_CLK.

The following table describes the data width configurations that are supported by LSRAM configured in dual-port mode.

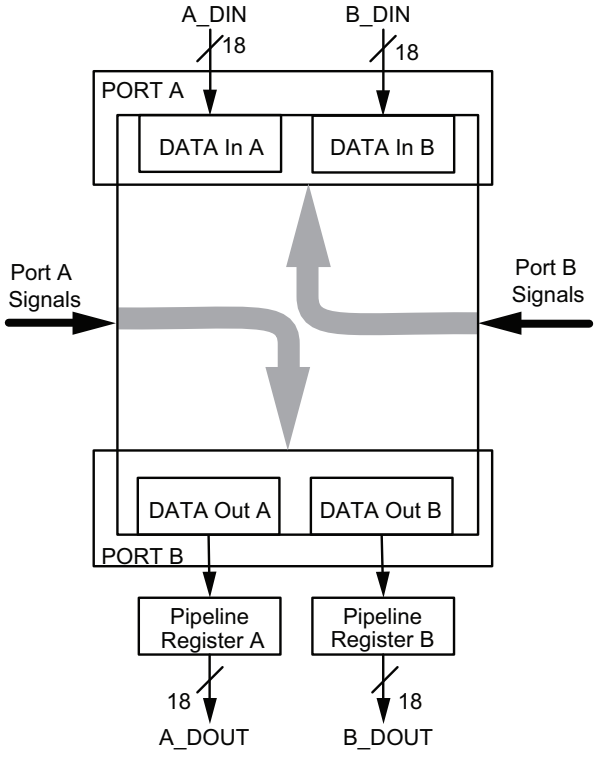
Table 12 • **Data Width Configurations for LSRAM in Dual-Port Mode**

Port A Data Width (represented as “x number of bits”)	Port B Data Width (represented as “x number of bits”)
x1	x1, x2, x4, x8, x16
x2	x1, x2, x4, x8, x16
x4	x1, x2, x4, x8, x16
x8	x1, x2, x4, x8, x16
x16	x1, x2, x4, x8, x16
x9	x9, x18
x18	x9, x18

3.4.2 Two-Port Mode

LSRAM configured as two-port SRAM provides a data storage capability of 18 Kbits, with Port A dedicated to read operations and Port B dedicated to write operations, as shown in the following figure. In two-port mode, the maximum data width for the read port (Port A) and the write port (Port B) is x36.

Figure 12 • **Data Path for Two-Port Mode**



In two-port mode, LSRAM can be configured in the following depth x width configurations:

- 512 x 36
- 512 x 32
- 1k x 18, 1k x 16
- 2k x 9, 2k x 8
- 4k x 4
- 8k x 2
- 16k x 1

There is no collision detection or prevention circuit built into LSRAM. Simultaneous read operations from Port A and write operations from Port B for the same address location should be avoided. This situation results in correct values being written into the memory, but garbage values will be read out from the memory.

When the read port data width is configured as x36/x32:

- Output data pins are borrowed from Port B, with Port A forming the MSB and Port B forming the LSB.
- Input data pins are borrowed from Port A, with Port A forming the MSB and Port B forming the LSB.

The read operation requires one clock cycle in Non-pipelined mode. In Pipelined mode, the output data appears in the next cycle. The write operation requires one clock cycle.

When the read operation is configured with output pipeline registers, the input clock sourcing the pipeline registers has to be synchronized to the LSRAM's clock input. When the read data width is x18 or less, A_DOUT_CLK has to be synchronized to A_CLK. When the read data width is x36/x32, both A_DOUT_CLK and B_DOUT_CLK have to be synchronized to A_CLK.

Note: Enable pipeline mode to achieve high performance. This will pipeline the output data bus before the data bus is delivered to the FPGA fabric.

The following table describes the data width configurations supported by LSRAM configured in two-port mode.

Table 13 • **Data Width Configurations for LSRAM in Two-Port Mode**

Read Port – Port A (represented as “x number of bits”)	Write Port – Port B (represented as “x number of bits”)
x1	x1, x2, x4, x8, x16
x2	x1, x2, x4, x8, x16
x4	x1, x2, x4, x8, x16
x8	x1, x2, x4, x8, x16
x9	x9, x18
x16	x1, x2, x4, x8, x16
x18	x9, x18
x32	x1, x2, x4, x8, x16, x32
x36	x9, x18, x36

In two-port mode, if the write data width is x36/x32 and read data width is x36/x32, both the bits of A_WEN and B_WEN have to be tied to logic 1 and should not be dynamically changed.

3.5 Operating Modes

3.5.1 Read Operation

3.5.1.1 Flow-Through Read

Flow-through mode indicates a non-pipelined read operation where the pipeline registers are bypassed and the data is displayed on the corresponding output in the same clock cycle. During flow-through read operation, the LSRAM can generate glitches on the data output buses. Therefore, Microchip recommends using LSRAM with pipeline registers to avoid these read glitches.

3.5.1.2 Pipelined Read

In a pipelined read operation, the output data is registered at the pipeline registers, so the data is displayed on the corresponding output in the next clock cycle. In Pipeline mode, pipeline clock input and LSRAM's clock input should be synchronized and fed with a single clock source.

Timing Diagram: Flow-Through Read and Pipeline Read

- The addresses (A_ADDR, B_ADDR), BLK enables (A_BLK, B_BLK), and read enables (A_WEN, B_WEN = '0') should be set up before the rising edge of the clock (A_CLK, B_CLK).
- For non-pipeline read operations, data comes on the output bus (A_DOUT, B_DOUT) after a delay of T_{CLK2Q} (read access time without pipeline register) in the same cycle.
- For pipeline read operations, the data is displayed on the output in the next clock cycle.

The following illustration shows the timing diagram for a read operation performed on LSRAM.

Figure 13 • **Read Operation Timing Waveforms**

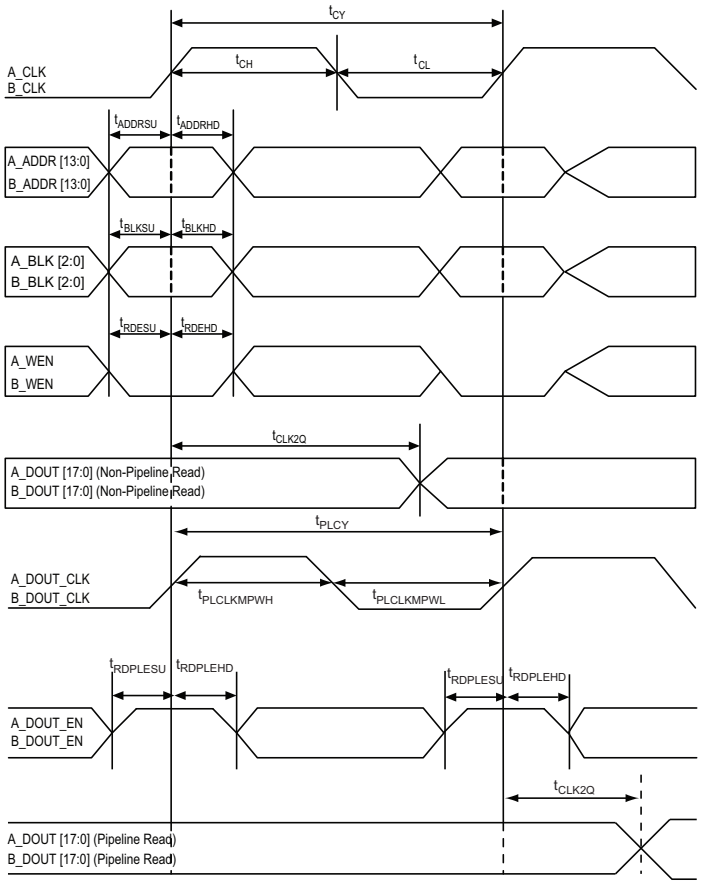


Table 14 • **Read Operation Timing Parameters**

Parameters	Description
T_{CY}	Clock period
T_{CH}	Clock minimum pulse width High
T_{CL}	Clock minimum pulse width Low
T_{ADDRSU}	Address setup time
T_{ADDRHD}	Address hold time
T_{BLKSU}	Block select setup time (With pipeline register enabled)
T_{BLKHD}	Block select hold time (With pipeline register enabled)
T_{RDESU}	Read enable setup time (A_WEN, B_WEN =0)
T_{RDEHD}	Read enable hold time (A_WEN, B_WEN =0)
T_{CLK2Q}	Read access time with pipeline register
	Read access time without pipeline register
T_{PLCY}	Pipelined clock period
$T_{PLCLKMPWH}$	Pipelined clock minimum pulse width High
$T_{PLCLKMPWHL}$	Pipelined clock minimum pulse width Low

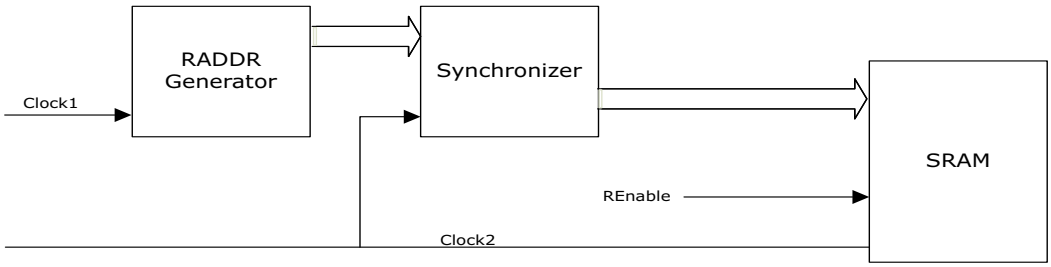
Table 14 • **Read Operation Timing Parameters *(continued)***

Parameters	Description
$T_{RDPLESU}$	Pipelined read enable setup time (A_DOUT_EN, B_DOUT_EN)
$T_{RDPLEHD}$	Pipelined read enable hold time (A_DOUT_EN, B_DOUT_EN)

Note: Data in the SRAM can be corrupted during a read operation when the read address does not meet setup and hold requirements with respect to the clock for that port.The data corruption occurs when the read address (RADDR) changes almost simultaneously with read clock (RCLK) i.e RADDR violates the timing requirement of the memory. Users must always meet the setup and hold time requirements on the RAM inputs to have reliable and predictable results for reads and writes.

To avoid data corruption due to Asynchronous clocking for read address and read clock of the RAM, users must implement a proper synchronizer circuit. The following figure illustrates a sample Synchronizer mechanism.

Figure 14 • **RADDR Synchronizer**



3.5.2 Write Operation

3.5.2.1 Feed-Through Write (write-bypass write)

During this write operation, the data written into the memory array is displayed immediately on the corresponding data output for non-pipeline operation. For pipeline operation data output displays in next clock. The feed-through write option is not supported when the LSRAM is configured in two-port mode.

3.5.2.2 Simple Write

In simple write, the data written into the memory array is not displayed on the corresponding data output until it is read out. The data output retains the last read data value.

3.5.2.3 Timing Diagram: Feed-Through Write and Simple Write

- The addresses (A_ADDR, B_ADDR), BLK enables (A_BLK, B_BLK), and write enables (A_WEN, B_WEN = '1') should be set up before the rising edge of the clock (A_CLK, B_CLK).
- For a feed-through write, the written data is displayed on the output (A_DOUT, B_DOUT) after a delay of T_{CLK2Q} in the same clock cycle.
- For a simple write, the written data is displayed on the output only when a read operation is performed on the same address.

The following illustration shows the timing diagram for a write operation performed on LSRAM.

Figure 15 • **Write Operation Timing Waveforms**

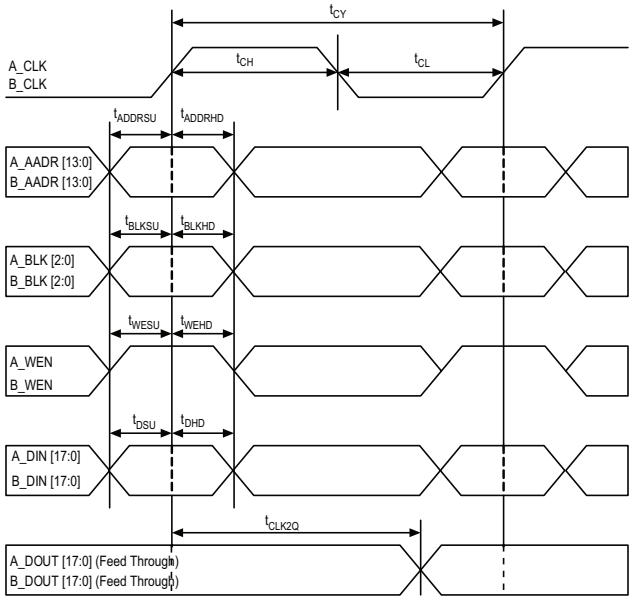


Table 15 • **Write Operation Timing Parameters**

Parameters	Description
T_{CY}	Clock period
T_{CH}	Clock minimum pulse width High
T_{CL}	Clock minimum pulse width Low
T_{ADDRSU}	Address setup time
T_{ADDRHD}	Address hold time
T_{BLKSU}	Block select setup time (With pipeline register enabled)
T_{BLKHD}	Block select hold time (With pipeline register enabled)
T_{WESU}	Write enable setup time (A_WEN, B_WEN =1)
T_{WEHD}	Write enable hold time (A_WEN, B_WEN =1)
T_{DSU}	Data setup time
T_{DHD}	Data setup time
T_{CLK2Q}	Read access time with Feed-through write timing

3.5.3 Reset Operation

The reset signals (A_ARST_N and B_ARST_N) are asynchronous Active Low signals. For any normal operation of LSRAM, these reset signals should be kept High. To reset the LSRAM, the reset signals must be Low.

When reset is asserted (A_ARST_N or B_ARST_N forced Low), the LSRAM behaves as follows during read and write operations:

1. **Read operation**: If reset is asserted when the read operation is in process, the data output port is forced Low after a certain amount of delay. If the clock is High and the reset signal is asserted and then deasserted in the same High clock phase or Low clock phase, the data output stays Low until the next cycle. The data output changes its state only if a read operation or write operation in Bypass mode is performed on the LSRAM. In a simple write operation, the data output stays Low.
2. **Write operation**: The corrupted data is written into the memory. Therefore, Microchip recommends to avoid asserting reset during write operation.

3.5.3.1 Timing Diagram: Asynchronous Reset Operation

Figure 16 • **Asynchronous Reset Operation**

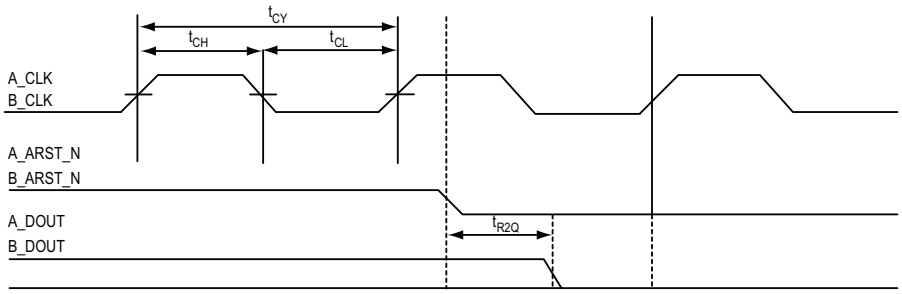


Table 16 • **Asynchronous Reset Timing Parameters**

Parameters	Description
T_{CY}	Clock period
T_{CH}	Clock minimum pulse width High
T_{CL}	Clock minimum pulse width Low
T_{R2Q}	Asynchronous reset to output propagation delay

3.5.4 Block Select Operation

The block select in LSRAM works like a chip select. When the block select (A_BLK and B_BLK) is High, the LSRAM is active and read and write operations can be performed.

If the block select is Low, the LSRAM does not perform any read or write operations. It drives logic 0 on the data output pins until the next read cycle or write operation in Bypass mode. When the pipeline registers are used, the block select effect at the output is delayed by one pipeline clock cycle (the pipeline registers are independent of block select).

The following illustration shows the timing diagram for block select inputs for LSRAM.

Figure 17 • **Block Select Timings**

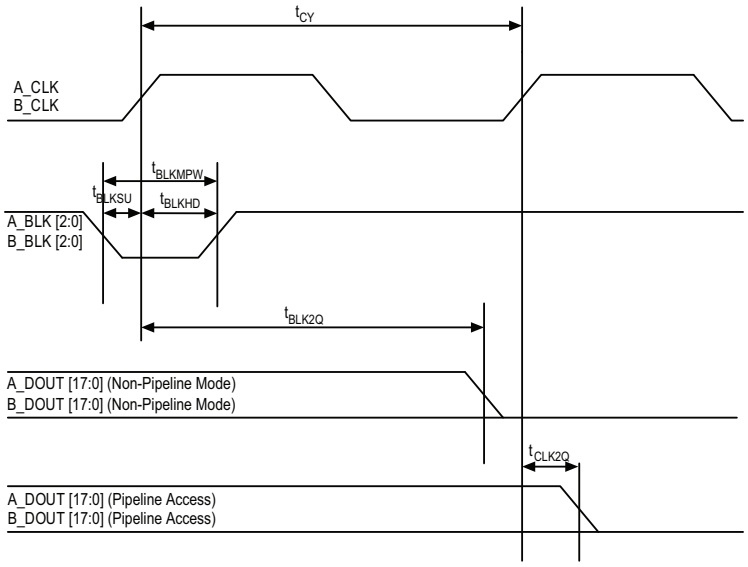


Table 17 • **Block Selection Timing Parameters**

Parameters	Description
T_{CY}	Clock period
T_{CH}	Clock minimum pulse width High
T_{CL}	Clock minimum pulse width Low
T_{BLKSU}	Block select setup time (with pipeline register enabled)
T_{BLKHD}	Block select hold time (with pipeline register enabled)
T_{BLKMPW}	Block select minimum pulse width
T_{BLK2Q}	Block select to out disable time (when pipeline registers are disabled)
T_{CLK2Q}	Read access time without pipeline register

Figure 16, page 28 shows the timing diagram for asynchronous reset operation performed on LSRAM.

3.5.5 Collision

Collision scenarios arise between both ports of the LSRAM when a read operation is requested from one port and a write operation from the other port simultaneously on the same address location, or when a write operation occurs at the same location at the same time from both the ports. The following table describes the behavior of the LSRAM during the various cases of collisions.

Table 18 • **Collision Operation Description**

Operation	Description
Simultaneous read from Port A and Port B at the same location	Operation is allowed without any restrictions and data is available on the output ports after the specified time, as described in the read timing diagrams in Figure 13, page 25.
Simultaneous read from Port A and write from Port B at the same location	Not allowed. If the user does this, the new data will be written, but the output data will be corrupted.
Simultaneous read from Port B and write from Port A at the same location	Not allowed. If the user does this, the new data will be written, but the output data will be corrupted.
Simultaneous write from Port A and Port B at the same location	Not allowed. If the data to be written is same on both the ports, then data is successfully written. If the data is different, then the LSRAM cell has an undetermined state.

There are no collision prevention or detection techniques available in LSRAM. The last 3 scenarios mentioned in the preceding table are not allowed on LSRAM and should be avoided.

3.6 How to Use LSRAM

The following sections describe how to use LSRAM in an application:

- Design Flow
- LSRAM Use Model

3.6.1 Design Flow

Libero SoC software provides separate configuration tools for dual-port mode and two-port mode. Using these configuration tools, LSRAM blocks can be configured in the required operating modes. These configuration tools generate the required HDL wrapper files for LSRAM with appropriate values assigned to the static signals. The generated LSRAM wrapper HDL files can be used in the design hierarchy by connecting the ports to the rest of the design.

3.6.1.1 LSRAM Dual-Port Mode

The following illustration shows the ports of the DPSRAM IP macro available in Libero SoC. See *SmartFusion2 Dual-Port Large SRAM Configuration* for detailed software configuration information on dual-port LSRAM.

Figure 18 • **Ports of the LSRAM Configured as Dual-Port SRAM - DPSRAM Macro in Libero SoC**

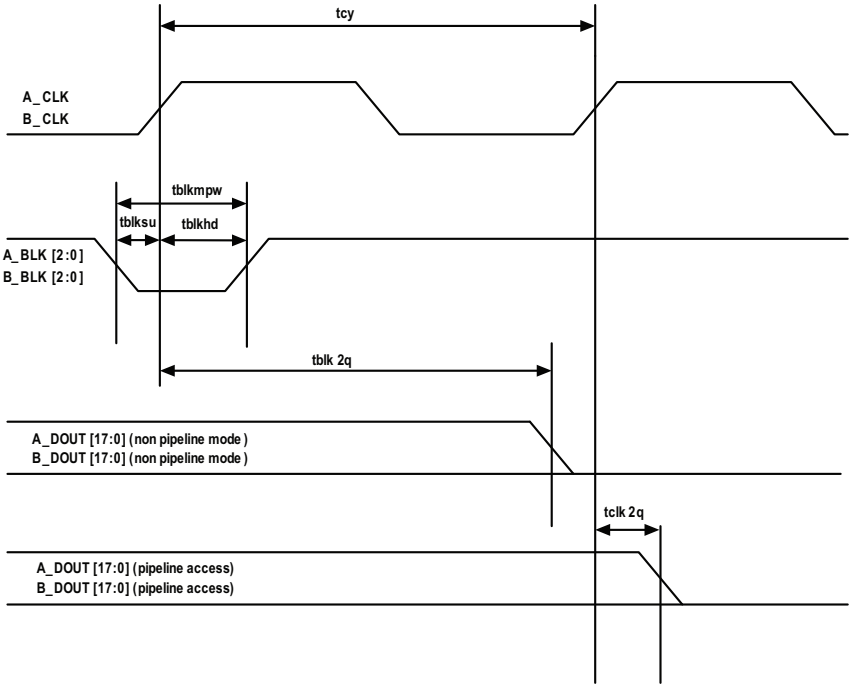


Table 19 • **Port Description for the DPSRAM Macro**

Port Name	Direction	Description
A_CLK, B_CLK	Input	These signals represent the clock inputs for Port A and Port B. The same clock inputs also act as clock inputs for the output pipeline registers if configured as registers. All inputs must be set up before the rising edge of the clock. The read or write operation begins with the rising edge.
A_ADDR, B_ADDR	Input	These signals represent the address inputs for Port A and Port B.
A_BLK, B_BLK	Input	These signals represent the block-select inputs for Port A and Port B.
A_DIN, B_DIN	Input	These signals represent the data inputs for Port A and Port B.
A_WEN, B_WEN	Input	These signals represent the write enables for Port A and Port B.
A_DOUT, B_DOUT	Output	These signals represent the data outputs for Port A and Port B.
A_DOUT_EN, B_DOUT_EN	Input	These signals represent the Read data register Enable for Port A and Port B
A_DOUT_SRST_N, B_DOUT_SRST_N	Input	These signals represent the Read data register Synchronous reset for Port A and Port B
A_DOUT_ARST_N, B_DOUT_ARST_N	Input	These signals represent the Read data register Asynchronous reset for Port A and Port B

3.6.1.2 LSRAM Two-Port Mode

The following figure shows the ports of the TPSRAM IP macro available in Libero SoC. See *SmartFusion2 Two-Port Large SRAM Configuration* document for detailed software configuration information for two-port LSRAM.

Figure 19 • **Ports of the LSRAM Configured as Two-Port SRAM - TPSRAM Macro in Libero SoC**

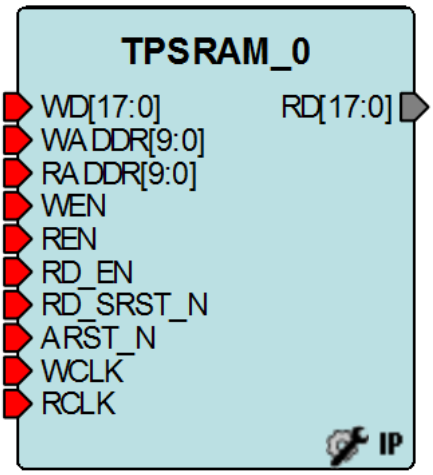


Table 20 • **Port Description for the TPSRAM Macro**

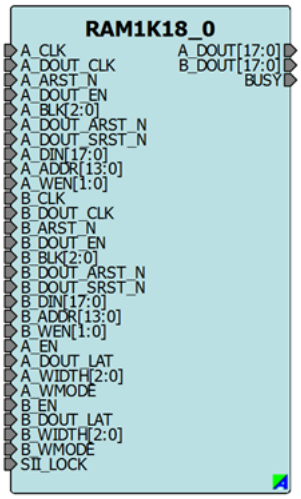
Port Name	Direction	Description
WCLK	Input	This signal represents the clock input for the write port (Port B). All write inputs must be set up before the rising edge of the clock. The write operation begins with the rising edge.
RCLK	Input	This signal represents the clock input for the read port (Port A). The same clock inputs also act as clock inputs for the output pipeline registers if configured as registers. All read inputs must be set up before the rising edge of the clock. The read operation begins with the rising edge.
ARST_N	Input	This signal represents Active Low, asynchronous reset inputs for Port A and Port B. Assertion of this reset during a read operation forces the data output lines to logic '0'. Assertion of these resets during a write operation results in garbage values written into the memory.
WADDR	Input	This signal represents the address input for write Port B.
RADDR	Input	This signal represents the address input for read Port A.
WEN	Input	This signal represents the write enable for write Port B.
WD	Input	This signal represents the data input for write Port B.
REN	Input	This signal represents the read enable for read Port A.
RD	Output	This signal represents the data output for read Port A.
RD_EN	Input	This signal represents the Read data register enable.
RD_SRST_N	Input	This signal represents the Read data register Synchronous reset.

3.6.1.3 LSRAM Macro (RAM 1Kx18)

Libero SoC can be used to instantiate the LSRAM macro (RAM1Kx18) in the design. When using the RAM1Kx18 macro, care should be taken to provide appropriate values to the static signals to configure the LSRAM correctly before instantiating it in the design.

The following figure shows the LSRAM macro RAM1Kx18 available in Libero SoC.

Figure 20 • **RAM1Kx18 Macro**



3.6.1.4 Associated LSRAM IP Cores

In addition to LSRAM macros, Libero SoC also has IP cores available to access the LSRAM through AHB and APB slave interfaces through which configuration parameters such as bus (AHB/APB) data width, RAM selection (LSRAM and µSRAM), and depth of the memory can be set. Figure 21, page 33 and Figure 22, page 34 show CoreAHBLSRAM and CoreAPBLSRAM, available in the Libero SoC catalog.

3.6.1.4.1 CoreAHBLSRAM

The following image shows CoreAHBLSRAM IP (LSRAM with AHB slave Interface), available in Libero SoC. See *CoreAHBLSRAM Handbook* for detailed software configuration information for dual-port LSRAM.

Figure 21 • **CoreAHBLSRAM IP in Libero SoC**

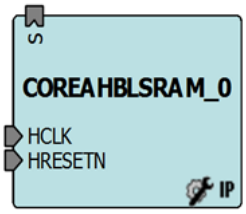


Table 21 • **Port Description for the CoreAHBLSRAM IP**

Port Name	Direction	Description
HCLK	Input	AHB clock. All AHB signals inside the block are clocked on the rising edge.
HRESETn	Input	AHB Reset. The signal is Active Low. Asynchronous assertion and synchronous deassertion. Used to reset AHB registers in the block.
S	Input/Output	AHB slave interface signals include: HSEL: AHBL slave select HADDR: AHBL address HWRITE: AHBL write HREADYIN: AHBL ready input

3.6.1.4.2 CoreAPBLSRAM

The following figure shows CoreAPBLSRAM IP (LSRAM with APB slave interface), available in Libero SoC. See *CoreAPBLSRAM Handbook* for detailed software configuration information.

Figure 22 • **CoreAPBLSRAM IP in Libero SoC**

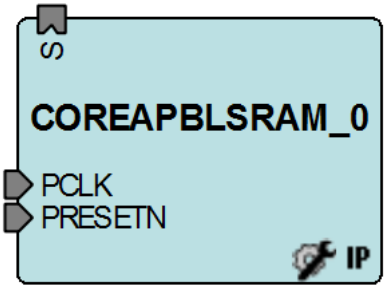


Table 22 • **Port Description for the CoreAPBLSRAM IP**

Port Name	Direction	Description
PCLK	Input	APB clock. All APB signals inside the block are clocked on the rising edge.
PRESETn	Input	APB Active Low asynchronous reset.
S	Input/Output	APB Slave interface signals include: PSEL: APB slave select PADDR: APB Address PWDATA: APB write data PRDATA: APB read data PENABLE: APB strobe. Indicates the second cycle of an APB transfer. PWRITE: APB write PREADY: APB3 ready signal for future APB3 compliance. Used to extend APB transfer. PSLVERR: APB slave error. Indicates transfer failure. It is tied to Low.

3.6.1.4.3 CoreFIFO IP

Libero SoC IP catalog has a CoreFIFO IP, which can be configured as a soft FIFO for generation of FIFO control logic. Memory configuration can be selected as LSRAM, µSRAM, or external memory as per the design requirements. For detailed software configuration information, see the *CoreFIFO Handbook*. This handbook can be downloaded from the Libero SoC Catalog.

3.6.2 LSRAM Use Model

3.6.2.1 Use Model 1: Two-Port SRAM with a Write Data Width of x36 and Read Data Width of x18

LSRAM does not support any two-port configurations with a write port (Port B) data width of x36/x32 and a read port (Port B) data width of x18/x9/x8/x4/x2/x1. If such a configuration is required for the design, two LSRAM blocks must be used to implement these configurations.

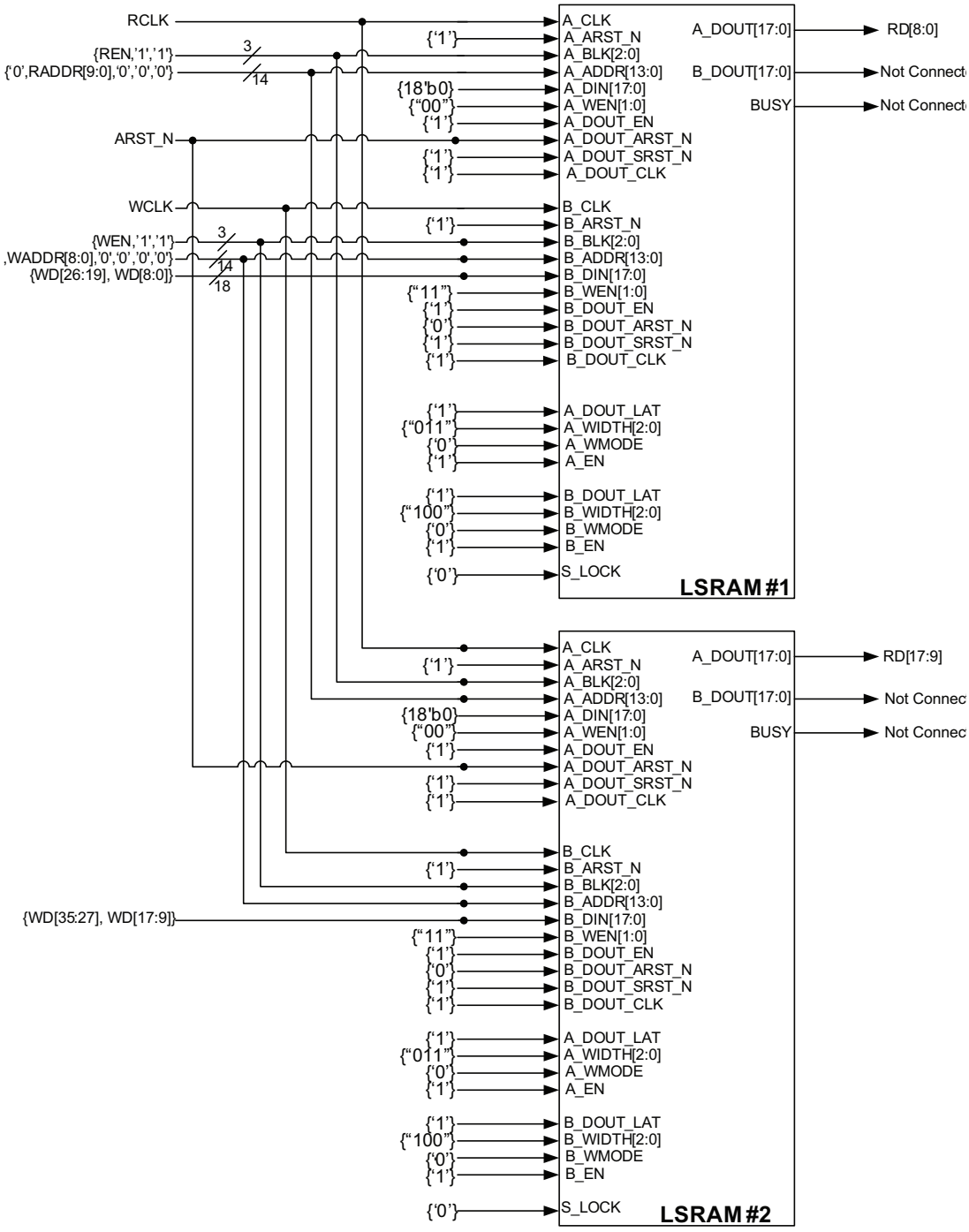
Following use model explains how to implement a two-port SRAM (using RAM1Kx18 macros) with a write data width of x36 and a read data width of x18.

The implementation has the following configurations:

- Write port: 512 x 36
- Read port: 1024 x 18
- Read and write input clock: Two different clock sources
- Pipelined read mode: Disabled

The following illustration shows the two-port SRAM with a write data width of x36 and read data width of x18.

Figure 23 • **Two-Port SRAM With W36 and R18**



The above implementation can be configured automatically using two-port LSRAM (TPSRAM) macro available in Libero SoC. The following table shows the TPSRAM data width configurations that require two LSRAM blocks.

Table 23 • **Two-Port Configurations Requiring Two LSRAM Blocks**

Write Data Width	Read Data width
x36	x18
x32	x16
x36	x9
x32	x8
x32	x4
x32	x2
x32	x1

4 Micro SRAM (µSRAM)

4.1 Introduction

The SmartFusion2 SoC and IGLOO2 FPGA fabrics have embedded 1 Kbit micro SRAM (µSRAM) blocks used for storing data. These µSRAMs are arranged in multiple rows within the FPGA fabric can be accessed through the fabric routing architecture. The number of µSRAM blocks available varies among SmartFusion2 and IGLOO2 devices, as shown in the following figure. For example, in the M2GL050 device there are 72 µSRAM blocks available, spread across three rows inside the fabric.

4.1.1 Features

The SmartFusion2 and IGLOO2 µSRAM blocks have the following features:

- Each µSRAM block stores up to 1 Kbits (1,152 bits) of data and can be configured in any of the following depth × width combinations: 64 × 18, 64 × 16, 128 × 9, 128 × 8, 256 × 4, 512 × 2 and 1,024 × 1.
- Each µSRAM has 2 read data ports (Port A and Port B) and 1 write data port (Port C).
- Read operations can be performed in both Synchronous and Asynchronous modes. The write operation is always synchronous.
- The 2 read ports have address/block select registers for enabling Synchronous mode operation. These registers can also be configured as transparent latches for Asynchronous mode operations.
- In Pipelined mode, the 2 read ports have output registers with independent clocks. These Output pipeline registers can also be configured as transparent latches for Asynchronous mode operation.
- Due to the availability of separate input address and output pipeline registers, read operations through Port A and Port B in µSRAM can be performed in 6 different modes:
  - Synchronous read mode without pipeline registers (Synchronous-Asynchronous mode)
  - Synchronous read mode with pipeline registers (Synchronous-Synchronous mode)
  - Asynchronous read mode without pipeline registers (Asynchronous-Asynchronous mode)
  - Asynchronous read mode with pipeline registers (Asynchronous-Synchronous mode)
  - Synchronous read mode with pipeline registers configured as latches
  - Asynchronous read mode with pipeline registers configured as latches
- Separate synchronous and asynchronous resets are provided for the input address/block select registers. These resets can be used to initialize the read ports.
- The output pipeline registers have separate synchronous and asynchronous resets which provide independent control to these registers.
- µSRAM can operate up to 400 MHz in Synchronous-Synchronous read mode through Port A and Port B, including a write operation at 400 MHz through Port C.
- The two read ports are independent of each other and simultaneous read operations can be performed from both ports at the same address location.
- Simultaneous read and write operations at the same location are not allowed.

4.2 µSRAM Resource Table

The following table lists µSRAM blocks available for SmartFusion2 and IGLOO2 devices.

Table 24 • **SmartFusion2 and IGLOO2 µSRAM (1Kb Blocks) Resource Table**

	Number of µSRAM		
SmartFusion2/IGLOO2	**Rows**	**Number per Row**	**Total**
M2S005/M2GL005	1	11	11
M2S010/M2GL010	2	11	22
M2S025/M2GL025	2	17	34
M2S050/M2GL050	3	24	72
M2S060/M2GL060	3	24	72

Table 24 • **SmartFusion2 and IGLOO2 µSRAM (1Kb Blocks) Resource Table *(continued)***

	Number of µSRAM		
SmartFusion2/IGLOO2	**Rows**	**Number per Row**	**Total**
M2S090/M2GL090	4	28	112
M2S150/M2GL150	6	40	240

4.3 Functional Description

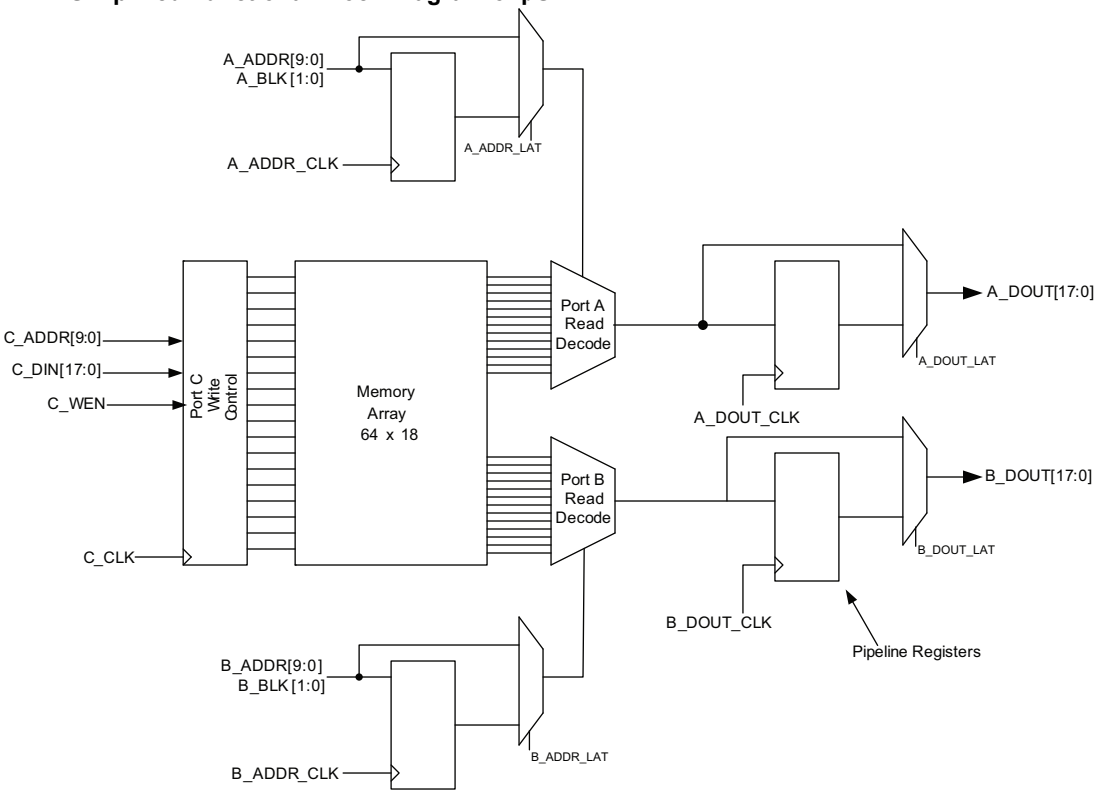
The following sections provide the detailed description of the following:

- Architecture Overview
- Port List
- Port Description

4.3.1 Architecture Overview

SmartFusion2 and IGLOO2 µSRAM embedded memory includes the RAM64X18 macro, available in Libero SoC software. The following illustration shows a simplified block diagram of the µSRAM memory block with two read data ports, one write data port and pipeline registers at read port. Table 25, page 40 provides the port descriptions.

Figure 24 • **Simplified Functional Block Diagram of µSRAM**



4.3.2 Port List

Table 25 • **Port List for µSRAM**

Port Name	Direction	Type[1]	Polarity	Descriptions
Port A				
A_ADDR[9:0]	Input	Dynamic		Address input
A_BLK[1:0]	Input	Dynamic	Active High	Block select
A_WIDTH[2:0]	Input	Static		Depth x width mode selection
A_DOUT[17:0]	Output	Dynamic		Data output
A_DOUT_ARST_N	Input	Dynamic	Active Low	Pipeline register asynchronous reset
A_DOUT_CLK	Input	Dynamic	Rising	Pipeline register clock input
A_DOUT_EN	Input	Dynamic	Active High	Pipeline register enable
A_DOUT_LAT	Input	Static	Active High	Pipeline Latch mode input
A_DOUT_SRST_N	Input	Dynamic	Active Low	Pipeline register synchronous reset
A_ADDR_CLK	Input	Dynamic	Rising	Address register clock
A_ADDR_EN	Input	Dynamic	Active High	Address register enable
A_ADDR_LAT	Input	Static	Active High	Address register Latch mode input
A_ADDR_SRST_N	Input	Dynamic	Active Low	Address register synchronous reset
A_ADDR_ARST_N	Input	Dynamic	Active Low	Address register asynchronous reset
Port B				
B_ADDR[9:0]	Input	Dynamic		Address input
B_BLK[1:0]	Input	Dynamic	Active High	Block select
B_WIDTH[2:0]	Input	Static		Depth x width mode selection
B_DOUT[17:0]	Output	Dynamic		Data output
B_DOUT_ARST_N	Input	Dynamic	Active Low	Pipeline register Asynchronous reset
B_DOUT_CLK	Input	Dynamic	Rising	Pipeline register clock input
B_DOUT_EN	Input	Dynamic	Active High	Pipeline register enable
B_DOUT_LAT	Input	Static	Active High	Pipeline Latch mode input
B_DOUT_SRST_N	Input	Dynamic	Active Low	Pipeline register synchronous reset
B_ADDR_CLK	Input	Dynamic	Rising	Address register clock
B_ADDR_EN	Input	Dynamic	Active High	Address register enable
B_ADDR_LAT	Input	Static	Active High	Address register Latch mode input
B_ADDR_SRST_N	Input	Dynamic	Active Low	Address register synchronous reset
B_ADDR_ARST_N	Input	Dynamic	Active Low	Address register asynchronous reset
Port C				
C_ADDR[9:0]	Input	Dynamic		Address input
C_BLK[1:0]	Input	Dynamic	Active High	Block select
C_WIDTH[2:0]	Input	Static		Depth x width mode selection
C_DIN[17:0]	Output	Dynamic		Data output
C_CLK	Input	Dynamic	Rising	Clock input

Table 25 • **Port List for µSRAM *(continued)***

Port Name	Direction	Type[1]	Polarity	Descriptions
C_WEN	Input	Dynamic	Active High	Write enable
Common Signals				
A_EN	Input	Static	Low	Port A power-down
B_EN	Input	Static	Low	Port B power-down
C_EN	Input	Static	Low	Port C power-down
SII_LOCK	Input	Static	High	Lock access to SII
Busy	Output	Dynamic	High	Busy signal while SII access

1. Static inputs are defined at design time and are controlled by flash configuration bits.

4.3.3 Port Description

4.3.3.1 A_WIDTH[2:0], B_WIDTH [2:0], and C_WIDTH [2:0]

These signals represent the depth x width mode selections for each port. The following table shows the depth x width based on ports width selection.

Table 26 • **Width/Depth Mode Selection**

A_WIDTH / B_WIDTH / C_WIDTH	Depth x Width
000	1K x 1
001	512 x 2
010	256 x 4
011	128 x 9 128 x 8
100 101 110 111	64 x 18 64 x 16

4.3.3.2 A_ADDR[9:0], B_ADDR [9:0], and C_ADDR [9:0]

These signals represent the address buses for the three ports (two read and one write). In ×1 mode, 10 bits are used to address the 1,152 independent locations. In wider modes such as ×2 and ×4, fewer address bits are used. The used address bits are the most significant bits (MSB). The unused bits are the least significant bits (LSBs) and they must be grounded. The following table shows the address bus used and unused bits for depth × width selections.

Table 27 • **Address Bus Used and Unused Bits**

	A_ADDR / B_ADDR / C_ADDR	
Depth x Width	**Used Bits**	**Unused Bits (to be grounded)**
1K x 1	[9:0]	None
512 x 2	[9:1]	[0]
256 x 4	[9:2]	[1:0]
128 x 9 128 x 8	[9:3]	[2:0]

Table 27 • **Address Bus Used and Unused Bits *(continued)***

	A_ADDR / B_ADDR / C_ADDR	
Depth x Width	**Used Bits**	**Unused Bits (to be grounded)**
64 x 18 64 x 16	[9:4]	[3:0]

4.3.3.3 C_DIN[17:0]

This signal represents the data input bus for the write Port C. The used bits for any mode are LSB justified in the data bus and the unused MSB bits must be grounded. The following table shows the data input bus used and unused bits for depth × width selections.

Table 28 • **Data Input Buses Used and Unused Bits**

	C_DIN	
Depth x Width	**Used Bits**	**Unused Bits (to be grounded)**
1K x 1	[0]	[17:1]
512 x 2	[1:0]	[17:2]
256 x 4	[3:0]	[17:4]
128 x 8	[7:0]	[17:8]
128 x 9	[8:0]	[17:9]
64 x 16	[16:9] [7:0]	[17] [8]
64 x 18	[17:0]	None

4.3.3.4 A_DOUT[17:0] and B_DOUT[17:0]

These signals represent the data output buses for the two ports (Port A and Port B). The used bits for any mode are LSB justified in the data bus and the unused MSB bits must be grounded. The following table shows the data output bus used and unused bits for different depth x width selections.

Table 29 • **Data Output Buses Used and Unused Bits**

	A_DOUT/B_DOUT	
Depth x Width	**Used Bits**	**Unused Bits**
1K x 1	[0]	[17:1]
512 x 2	[1:0]	[17:2]
256 x 4	[3:0]	[17:4]
128 x 8	[7:0]	[17:8]
128 x 9	[8:0]	[17:9]
64 x 16	[16:9] [7:0]	[17] [8]
64 x 18	[17:0]	

4.3.3.5 A_BLK[1:0], B_BLK [1:0], and C_BLK [1:0]

These signals represent the port select control signal for each port. The following table shows the operations (Read, write and no operation) based on selection of port select control signals.

Table 30 • **Port Select Control Signals**

Port Select Signal	Value	Operation
A_BLK[1:0]	11	Perform read operation on Port A.
	00	
	01	Port A is not selected and its read data will be logic 0.
	10	
B_BLK[1:0]	11	Perform read operation on Port B.
	00	
	01	Port B is not selected and its read data will be logic 0.
	10	
C_BLK[1:0]	11	Perform write operation on Port C.
	00	
	01	Port C is not selected.
	10	

Note: A_BLK[1:0],B_BLK [1:0],and C_BLK signals are synchronous/registered with respect to port clock.

- Asserting A_BLK reads the RAM at the address given by the output of the A_ADDR register onto the input of the A_DOUT register.De-asserting A_BLK forces A_DOUT to zero.
- Asserting B_BLK reads the RAM at the address given by the output of the B_ADDR register onto the input of the B_DOUT register. De-asserting B_BLK forces B_DOUT to zero.
- Asserting C_BLK when C_WEN is high will write the data C_DIN into the RAM at the address C_ADDR on the next rising edge of C_CLK.

4.3.3.6 C_CLK

This signal represents the clock signal for Port C. Ensure all inputs are set up before the first rising clock edge. The write operation starts at the rising edge of this clock signal.

4.3.3.7 C_WEN

This signal represents the write enable for Port C.

4.3.3.8 A_ADDR_CLK and B_ADDR_CLK

These signals represent the clock inputs for the input address/block select registers for Port A and Port B. In Synchronous read mode, set up the address and block select inputs before the rising edge of these clocks. In Asynchronous mode, tie these clocks to logic 1.

4.3.3.9 A_DOUT_CLK and B_DOUT_CLK

These signals represent the clock inputs for the output pipeline registers for Port A and Port B. In Pipelined mode, the output data appears in the next clock cycle. In Latch mode operation, the output data appears in the same clock cycle. When the registers are configured as transparent, tie these inputs to logic 1.

4.3.3.10 A_ADDR_LAT and B_ADDR_LAT

These signals represent Latch mode inputs for the input address/block select registers for Port A and Port B.

- Logic 0: Register operation
- Logic 1: Transparent operation

4.3.3.11 A_DOUT_LAT and B_DOUT_LAT

These signals represent Latch mode inputs for the output pipeline registers for Port A and Port B.

- Logic 0: Register operation
- Logic 1: Latch/Transparent operation

4.3.3.12 A_ADDR_ARST_N and B_ADDR_ARST_N

These signals represent Active Low, asynchronous reset inputs for the input address/block select registers for Port A and Port B.

The assertion of these reset signals forces the address and block select input registers to logic 0, which in turn forces the data output to logic 0. When these registers are configured as transparent, tie these inputs to logic 1.

4.3.3.13 A_DOUT_ARST_N and B_DOUT_ARST_N

These signals represent Active Low, asynchronous reset inputs for the output pipeline registers for Port A and Port B. Assertion of these reset signals forces the data output to logic 0. When these registers are configured as transparent, tie these inputs to logic 1.

4.3.3.14 A_ADDR_SRST_N and B_ADDR_SRST_N

These signals represent Active Low, synchronous reset inputs for the input address/block select registers for Port A and Port B. The assertions of these reset signals forces the address input registers and block select registers to logic 0, which in turn forces the data output to logic 0. When the registers are configured as transparent, these inputs should be tied to logic 1.

4.3.3.15 A_DOUT_SRST_N and B_DOUT_SRST_N

These signals represent Active Low, synchronous reset inputs for the output pipeline registers for Port A and Port B. Assertion of these reset signals forces the data output to logic 0. In non-pipelined mode of operation, tie these inputs to logic 1.

4.3.3.16 A_ADDR_EN and B_ADDR_EN

These signals represent Active High enable inputs for the input address/block select registers for Port A and Port B. When logic 0 is applied on these inputs, the input registers hold the previous input address. When logic 1 is applied on these inputs, the input registers behave as normal D flip-flops. When the registers are configured as transparent, these inputs should be tied to logic 1.

4.3.3.17 A_DOUT_EN and B_DOUT_EN

These signals represent Active High enable inputs for the output pipeline registers for Port A and Port B. When logic 0 is applied on these inputs, the pipeline registers hold the previously read data out. In non-pipelined mode, tie these inputs to logic 1.

4.3.3.18 A_EN, B_EN, and C_EN

These are Active Low, power-down configuration bits for each port.

4.3.3.19 SII_LOCK

This control signal, when asserted to logic 1, locks the entire µSRAM memory from being accessed by the system controller interface bus (SII). The system controller can access the µSRAM for the following reasons:

- Testing the memory
- Moving data between µSRAM and eNVM or external memories
- Moving data between various µSRAMs

- Moving data between µSRAMs and LSRAMs

µSRAMs cannot be accessed while the system controller is accessing them.

4.3.3.20 BUSY

This signal acts as a status signal when the system controller is accessing a particular µSRAM. Logic 1 on this signal indicates system controller access. This signal can be used to monitor the completion of µSRAM access.

4.4 Operating Modes

4.4.1 Read Operation

µSRAM blocks are read through two ports: Port A and Port B. There are six modes for read operations:

- Synchronous read mode without pipeline registers (Synchronous-Asynchronous mode)
- Synchronous read mode with pipeline registers (Synchronous-Synchronous mode)
- Synchronous read mode with pipeline registers configured as latches
- Asynchronous read mode without pipeline registers (Asynchronous-Asynchronous mode)
- Asynchronous read mode with pipeline registers (Asynchronous-Synchronous mode)
- Asynchronous read mode with pipeline registers configured as latches

4.4.1.1 Synchronous Read Mode

Synchronous read mode requires that the input registers for the address and block select inputs are configured in Flip-flop mode (A_ADDR_LAT or B_ADDR_LAT = 0). Similarly, on the output side, the pipeline registers can be configured as registers, latches, or transparent, providing read data as registered, latched, or asynchronous.

When the pipeline registers are configured as normal registers, the clock inputs of both the input and output registers should be synchronous to each other and should be fed with a single clock source. If these registers are configured as a transparent latch, the Latch inputs should be tied to High. In Latch mode, both the input and output clocks should be in opposite phase. Microchip recommends configuring the pipeline registers, in either the register or Latch mode during read operation to avoid glitches on the read output data lines.

In Synchronous read mode, the address (A_ADDR or B_ADDR) and block-select (A_BLK or B_BLK) inputs must satisfy the setup and hold timings with respect to the input clocks (A_ADDR_CLK or B_ADDR_CLK).

4.4.1.2 Synchronous Read Mode without Pipeline Registers (Synchronous-Asynchronous Read Mode)

- The input registers are configured in Synchronous read mode.
- The output pipeline registers are configured as transparent.
- This mode is achieved by setting A_DOUT_LAT or B_DOUT_LAT = 1, A_DOUT_CLK or B_DOUT_CLK = 1, A_DOUT_ARST_N or B_DOUT_ARST_N = 1, A_DOUT_SRST_N = 1 or B_DOUT_SRST_N = 1, A_DOUT_EN or B_DOUT_EN = 1, A_BLK = 1, B_BLK = 1.
- The following illustration shows the synchronous asynchronous operation with data output behavior when block select inputs are deasserted (any bit forced to logic 0).
- The output data is displayed immediately-in the same clock cycle in which the address and block select inputs were registered.
- The µSRAM can generate glitches on the output buses when used without the pipeline registers.

The following illustration and table describe the timing parameter values for Synchronous read mode without pipeline registers, with reference to timing waveforms.

Figure 25 • **Timing Waveforms for Synchronous-Asynchronous Read Operation**

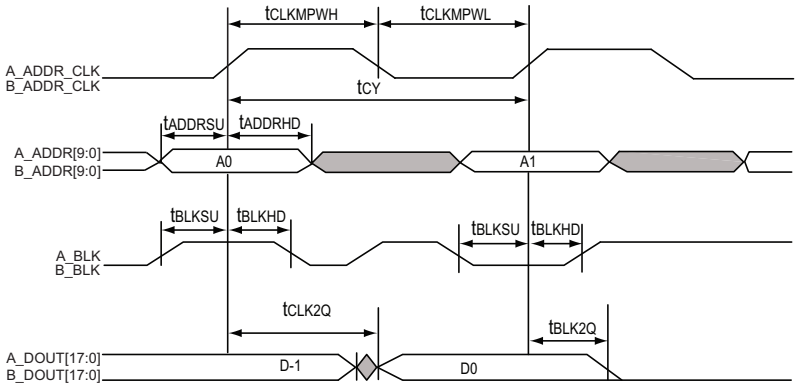


Table 31 • **Timing Parameters for Synchronous-Asynchronous Read Operation**

Parameter	Description
T_{CY}	Read clock period
$T_{CLKMPWH}$	Read clock minimum pulse width High time
$T_{CLKMPWL}$	Read clock minimum pulse width Low time
T_{ADDRSU}	Read address setup time in Synchronous mode
T_{ADDRHD}	Read address hold time in Synchronous mode
T_{BLKSU}	Read block select setup time (when pipeline registers enabled)
T_{BLKHD}	Read block select hold time (when pipeline registers enabled)
T_{CLK2Q}	Read access time without pipeline registers
T_{BLK2Q}	Read block select to out disable/enable time

4.4.1.3 Synchronous Read Mode with Pipeline Registers (Synchronous-Synchronous Read Mode)

- The input registers are configured in Synchronous read mode.
- The output pipeline registers are configured as edge-triggered registers (Pipelined mode).
- Pipelined mode is achieved by setting A_DOUT_LAT or B_DOUT_LAT = 0, A_DOUT_CLK or B_DOUT_CLK = rising edge clock, A_DOUT_ARST_N or B_DOUT_ARST_N = 1, A_DOUT_SRST_N = 1 or B_DOUT_SRST_N = 1, A_DOUT_EN or B_DOUT_EN = 1, A_BLK = 1, B_BLK = 1.
- The input register clock and pipeline register clock must be synchronous to each other; hence they should be sourced from the same clock input.
- The output data appears on the output bus in the next clock cycle.

The following illustration and table describe the timing parameter values for Synchronous read mode with pipeline registers.

Figure 26 • **Timing Waveforms for Synchronous-Synchronous Read Operation**

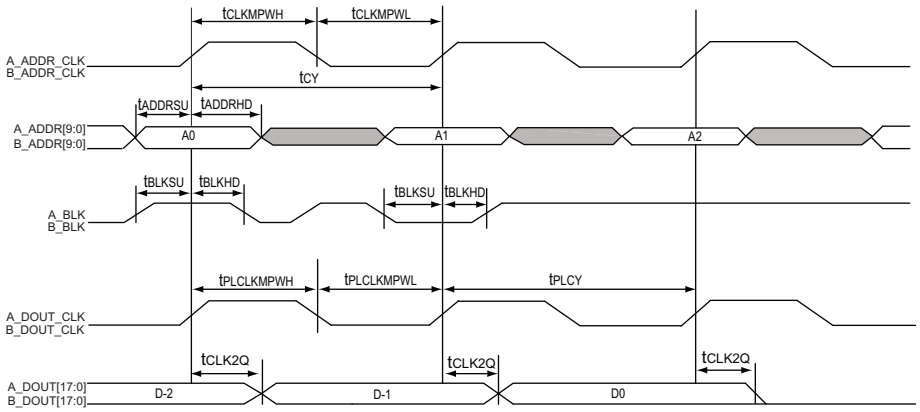


Table 32 • **Timing Parameters for Synchronous-Synchronous Read Operation**

Parameter	Description
T_{CY}	Read clock period
$T_{CLKMPWH}$	Read clock minimum pulse width High time
$T_{CLKMPWL}$	Read clock minimum pulse width Low time
T_{ADDRSU}	Read address setup time in Synchronous mode
T_{ADDRHD}	Read address hold time in Synchronous mode
T_{BLKSU}	Read block select setup time (when pipeline registers enabled)
T_{BLKHD}	Read block select hold time (when pipeline registers enabled)
T_{CLK2Q}	Read access time with pipeline registers
T_{PLCY}	Read pipeline clock period
$T_{PLCLKMPWH}$	Read pipeline clock minimum pulse width High
$T_{PLCLKMPWL}$	Read pipeline clock minimum pulse width Low

4.4.1.4 Synchronous Read Mode with Pipeline Registers Configured as Latches

- The input registers are configured in Synchronous read mode.
- The output pipeline registers are configured as level-sensitive latches with A_DOUT_CLK or B_DOUT_CLK acting as latch enables.
- The pipeline registers are configured as latches by setting A_DOUT_LAT or B_DOUT_LAT = 1.
- The pipeline latches are enabled by the pipeline register clock (A_DOUT_CLK or B_DOUT_CLK) with opposite phase with respect to the input register clock (A_ADDR_CLK or B_ADDR_CLK). During the low phase of the pipeline clocks, the pipeline latches hold the previous data until the latch inputs become stable.
- In this case, the read access time is related to the negative edge of the address input clock (A_ADDR_CLK or B_ADDR_CLK)-the positive edge of the pipeline clock (A_DOUT_CLK or B_DOUT_CLK).
- This mode is used to moderate the effect of glitches that can appear on the µSRAM's data output bus when used without the pipeline registers (when µSRAM is configured in Synchronous-Asynchronous read mode).

The following illustration and table describe the timing parameter values for Synchronous read mode with Latched mode.

Figure 27 • **Timing Waveforms for Synchronous Latched Read Operation**

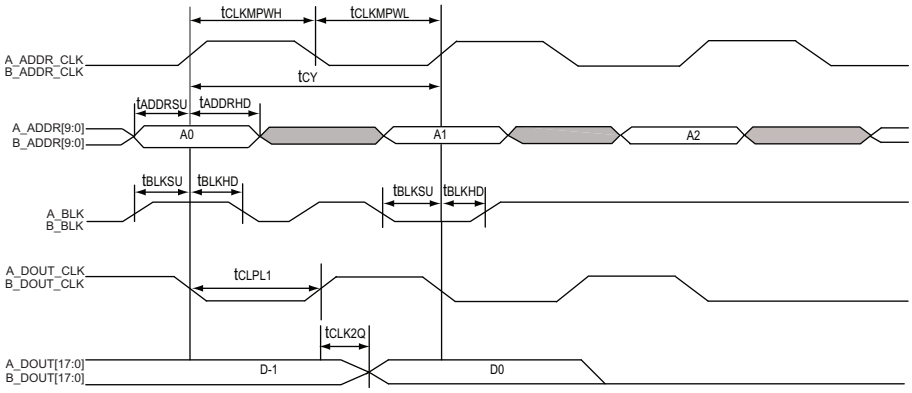


Table 33 • **Timing Parameters for Synchronous Latched Read Operation**

Parameter	Description
T_{CY}	Read clock period
$T_{CLKMPWH}$	Read clock minimum pulse width High time
$T_{CLKMPWL}$	Read clock minimum pulse width Low time
T_{ADDRSU}	Read address setup time in Synchronous mode
T_{ADDRHD}	Read address hold time in Synchronous mode
T_{BLKSU}	Read block select setup time (when pipeline registers enabled)
T_{BLKHD}	Read block select hold time (when pipeline registers enabled)
T_{CLK2Q}	Read access time with pipeline registers in Latch mode
T_{CLPL1}	Minimum pipeline clock low phase in order to prevent glitches with pipeline register in Latch mode.

4.4.1.5 Asynchronous Read Mode

Asynchronous read mode requires that the input registers for the address and block-select inputs are configured as transparent (A_ADDR_LAT or B_ADDR_LAT = 1, A_ADDR_CLK or B_ADDR_CLK = 1, A_ADDR_EN or B_ADDR_EN = 1, A_ADDR_ARST_N or B_ADDR_ARST_N = 1, A_ADDR_SRST_N or B_ADDR_SRST_N = 1, A_BLK = 1, B_BLK = 1).

4.4.1.6 Asynchronous Read Mode Without Pipeline Registers (Asynchronous-Asynchronous Mode)

- The input registers are configured in Asynchronous read mode.
- The output pipeline registers are configured as transparent (non-pipelined operation).
- The pipeline registers can be made transparent by setting A_DOUT_LAT or B_DOUT_LAT = 1, A_DOUT_CLK or B_DOUT_CLK = 1, A_DOUT_ARST_N or B_DOUT_ARST_N = 1, A_DOUT_SRST_N = 1 or B_DOUT_SRST_N = 1, A_DOUT_EN or B_DOUT_EN = 1.
- After the input address is provided, the output data is displayed on the output data bus after a T_{A2QR} delay, as shown in the following figure.
- The µSRAM can generate glitches on the data output bus when used without the pipeline register.

The following illustration and table describe a timing diagram for Asynchronous-Asynchronous read mode for µSRAM and various timing parameters.

Figure 28 • **Timing Waveforms for Read Operations with Asynchronous Inputs Without Pipeline Registers**

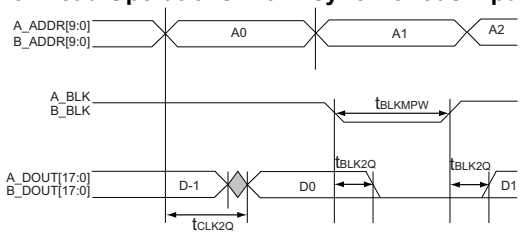


Table 34 • **Timing Parameters of the Asynchronous Read Mode Without Pipeline Registers**

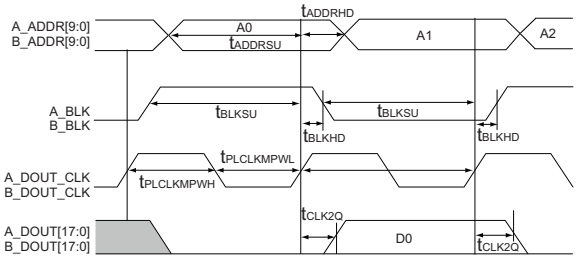
Parameter	Description
T_{CLK2Q}	Read access time without pipeline register
T_{BLK2Q}	Read block select to out disable/enable time
T_{BLKMPW}	Read block select minimum pulse width

4.4.1.7 Asynchronous Read Mode with Pipeline Registers (Asynchronous-Synchronous Mode)

- The input registers are configured in Asynchronous read mode.
- The output pipeline registers are configured as registers (Pipelined mode).
- Pipelined mode is achieved with A_DOUT_LAT or B_DOUT_LAT = 0, A_DOUT_CLK or B_DOUT_CLK = rising edge clock, A_DOUT_ARST_N or B_DOUT_ARST_N = 1, A_DOUT_SRST_N = 1 or B_DOUT_SRST_N = 1, A_DOUT_EN or B_DOUT_EN = 1, A_BLK = 1, B_BLK = 1.
- After the input address is provided, the output data is displayed on the output data bus after the next rising edge of the pipeline register input clock.

The following illustration describe the timing diagrams for Asynchronous-Synchronous read mode for µSRAM.

Figure 29 • **Timing Waveforms for Read Operations with Asynchronous Inputs with Pipeline Registers**



The following table lists the timing parameters.

Table 35 • **Timing Parameters of the Asynchronous Read Mode with Pipeline Registers**

Parameter	Description
T_{PLCY}	Read pipeline clock period
$T_{PLCLKMPWH}$	Read pipeline clock minimum pulse width High
$T_{PLCLKMPWL}$	Read pipeline clock minimum pulse width Low
T_{ADDRSU}	Read address setup time in Synchronous mode
T_{ADDRHD}	Read address hold time in Synchronous mode

Table 35 • **Timing Parameters of the Asynchronous Read Mode with Pipeline Registers *(continued)***

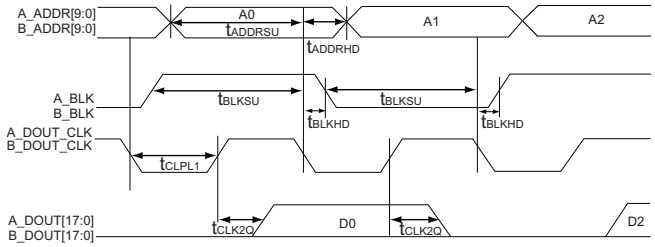
Parameter	Description
T_{BLKSU}	Read block select setup time (when pipeline registers enabled)
T_{BLKHD}	Read block select hold time (when pipeline registers enabled)
T_{CLK2Q}	Read access time with pipeline register

4.4.1.8 Asynchronous Read Mode with Pipeline Registers Configured as Latches

- The input registers are configured in Asynchronous read mode.
- The output pipeline registers are configured as level-sensitive latches with A_DOUT_CLK or B_DOUT_CLK acting as latch enables.
- The pipeline registers can be configured as latches by setting A_DOUT_LAT or B_DOUT_LAT =1.
- After the input address is provided, the output data is displayed on the output data bus when the next high level comes on the latch enable inputs-A_DOUT_CLK or B_DOUT CLK.
- This mode is provided to moderate the effect of the glitches which can occur on µSRAM's data output buses when used without the pipeline registers.

The following illustration shows the timing diagrams for Asynchronous read mode with latched outputs-pipeline registers configured as latches.

Figure 30 • **Timing Waveforms for Read Operations with Asynchronous Inputs with Latched Outputs**



The following table describes the timing parameters.

Table 36 • **Timing Parameters of the Asynchronous Read Mode with Latched Outputs**

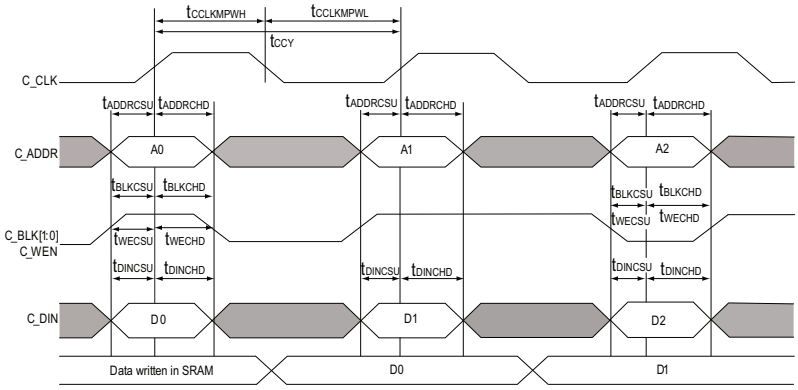
Parameter	Description
T_{CLPL1}	Minimum pipeline clock low phase in order to prevent glitches with pipeline register in Latch mode
T_{ADDRSU}	Read address setup time in Synchronous mode
T_{ADDRHD}	Read address hold time in Synchronous mode
T_{BLKSU}	Read block select setup time (when pipeline registers enabled)
T_{BLKHD}	Read block select hold time (when pipeline registers enabled)
T_{CLK2Q}	Read access time with pipeline register

4.4.2 Write Operation

- Port C is the only port through which a write operation can be performed on µSRAM.
- The write operation is purely synchronous and all operations are synchronized to the rising edge of the Port C clock input (C_CLK).
- The write inputs-C_ADDR, C_BLK, C_WEN, and C_DIN-have to satisfy the setup and hold timings with respect to the rising edge of the C_CLK input for a successful write operation.
- If all the inputs meet the required timing parameters, the input data is written into µSRAM in one clock cycle.

The following illustration shows the timing waveforms for a Port C write operation.

Figure 31 • **Timing Waveforms for the Write Operation**



The following table describes the timing parameters.

Table 37 • **Timing Parameters of the Write Operation**

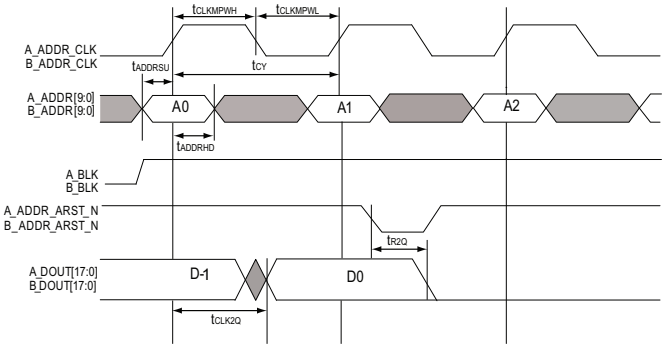
Parameter	Description
T_{CCY}	Write clock period
$T_{CCLKCMPWH}$	Write clock minimum pulse width High
$T_{CCLKCMPWL}$	Write clock minimum pulse width Low
$T_{ADDRCSU}$	Write address setup time
$T_{ADDRCHD}$	Write address hold time
T_{BLKCSU}	Write block setup time
T_{BLKCHD}	Write block hold time
T_{WECSU}	Write enable setup time
T_{WECHD}	Write enable hold time
T_{DINCSU}	Write input data setup time
T_{DINCHD}	Write input data hold time

4.5 Reset Operation

The reset signals (A_ADDR_ARST_N, B_ADDR_ARST_N) are asynchronous Active Low signals for the address and block select input registers for Port A and Port B. The assertion of these reset signals forces the address and block select input registers to logic 0, which in turn forces the data output to logic 0. When the registers are configured as transparent, tie these inputs to logic 1.

The following illustration shows the timing waveforms for these asynchronous reset signals.

Figure 32 • **Timing Waveforms for Asynchronous Reset**



The following table lists the Timing parameters for the asynchronous reset.

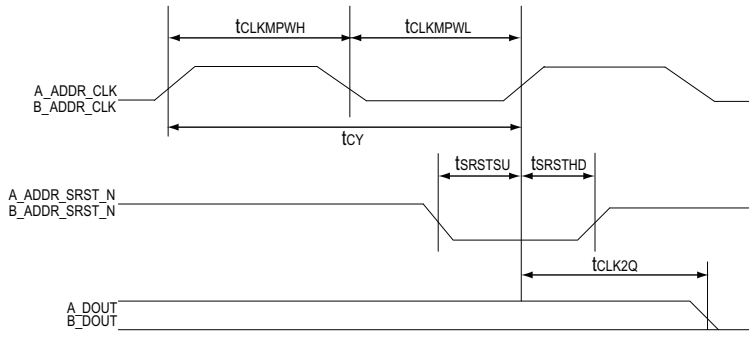
Table 38 • **Timing Parameters of the Asynchronous Reset**

Parameter	Description
T_{CY}	Read clock period
$T_{CLKMPWH}$	Read clock minimum pulse width High
$T_{CLKMPWL}$	Read clock minimum pulse width Low
T_{ADDRSU}	Read address setup time
T_{ADDRHD}	Read address hold time
T_{R2Q}	Read asynchronous reset to output propagation delay
T_{CLK2Q}	Read access time without pipeline register

The reset signals (A_ADDR_SRST_N, B_ADDR_SRST_N) are synchronous Active Low signals for the address and block select input registers for Port A and Port B. The assertion of these reset signals forces the address and block select input registers to logic 0, which in turn forces the data output to logic 0.

The following illustration shows the timing waveform for synchronous reset.

Figure 33 • **Timing Waveforms for Synchronous Reset**



The following table lists the timing parameters of the synchronous reset.

Table 39 • **Timing Parameters of the Synchronous Reset**

Parameter	Description
T_{CY}	Read clock period
$T_{CLKMPWH}$	Read clock minimum pulse width High
$T_{CLKMPWL}$	Read clock minimum pulse width Low
T_{SRSTSU}	Read synchronous reset setup time
T_{SRSTHD}	Read synchronous reset hold time
T_{CLK2Q}	Read synchronous reset to output propagation delay

4.5.1 Collision

Collision between ports occurs when the read and write operations are requested from two or all three ports at the same time at the same address location. The following table lists the different scenarios for collision.

Table 40 • **Collision Scenarios**

Operation	Comments
Simultaneous read from Port A and read from Port B to the same address location	Allowed since the read ports are independent of each other. Both read ports deliver correct read data.
Simultaneous read from Port A and write to Port C to the same address location	Collision occurs. The write operation works correctly but the read operation from Port A generates ambiguous data output unless the clock cycle is long enough to allow the newly written data to be read.
Simultaneous read from Port B and write to Port C to the same address location	Collision occurs. The write operation works correctly but the read operation from Port B generates ambiguous data output unless the clock cycle is long enough to allow the newly written data to be read.
Simultaneous read form Port A, read from Port B, and write to Port C to the same address location	Collision occurs. The write operation works correctly but the read operation from both the ports generates ambiguous data output unless the clock cycle is long enough to allow the newly written data to be read.

There is no collision prevention or detection implemented in the µSRAM architecture, so the designer must take measures to avoid the last three scenarios in designs.

4.6 How to Use µSRAM

The following section describes the Design Flow of µSRAM.

4.6.1 Design Flow

Libero SoC software provides a tool for configuring µSRAM blocks in the required operating modes. The required HDL wrapper files for µSRAM are generated with appropriate values assigned to the static signals. The generated µSRAM wrapper HDL files can be used in the design hierarchy by connecting the ports to the rest of the design.

4.6.1.1 µSRAM - IP

The following figure shows the ports of the µSRAM IP macro available in Libero SoC. See the *SmartFusion2/IGLOO2 Micro SRAM Configuration User Guide* for detailed information about software configuration for SRAM.

Figure 34 • **µSRAM IP Macro in Libero SoC**

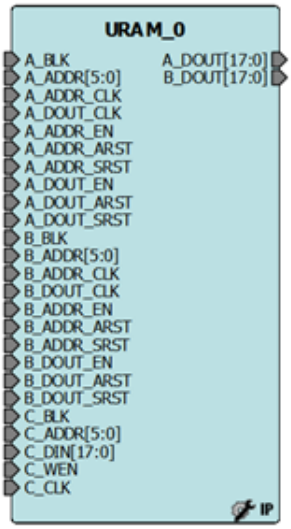


Table 41 • **Port Description for the µSRAM-IP Macro**

Port Name	Direction	Polarity	Description
A_ADDR[]	Input		Port A address input
A_BLK	Input	Active High	Port A block select
A_ADDR_CLK	Input	Rising edge	Port A clock for A_ADDR
A_DOUT_CLK	Input	Rising edge	Port A clock for A_DOUT
A_DOUT[]	Output		Port A data output
A_DOUT_ARST	Input	Active Low	Port A pipeline register asynchronous reset
A_DOUT_EN	Input	Active High	Port A pipeline register enable
A_DOUT_SRST	Input	Active Low	Port A pipeline register synchronous reset
A_ADDR_EN	Input	Active High	Port A address register enable
A_ADDR_SRST	Input	Active Low	Port A address register synchronous reset
A_ADDR_ARST	Input	Active Low	Port A address register asynchronous reset
B_ADDR[]	Input		Port B address input
B_BLK	Input	Active High	Port B block select

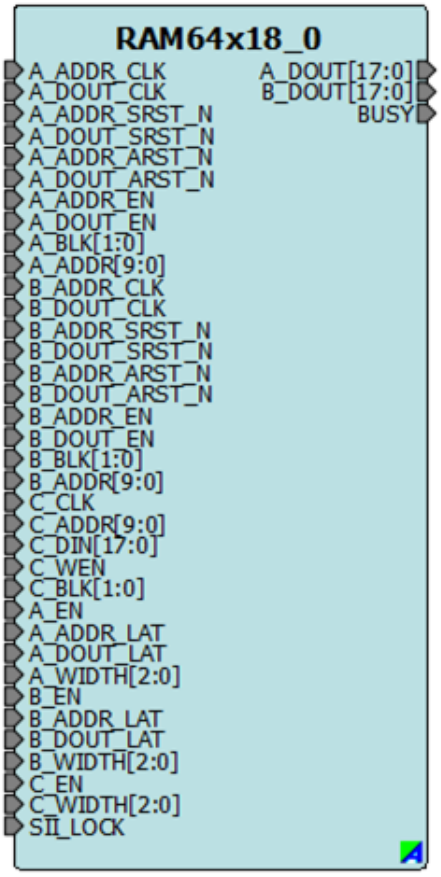
Table 41 • **Port Description for the µSRAM-IP Macro *(continued)***

Port Name	Direction	Polarity	Description
B_ADDR_CLK	Input	Rising edge	Port B clock for B_ADDR
B_DOUT_CLK	Input	Rising edge	Port B clock for B_DOUT
B_DOUT[]	Output		Port B data output
B_DOUT_ARST	Input	Active Low	Port B pipeline register asynchronous reset
B_DOUT_EN	Input	Active High	Port B pipeline register enable
B_DOUT_SRST	Input	Active Low	Port B pipeline register synchronous reset
B_ADDR_EN	Input	Active High	Port B address register enable
B_ADDR_SRST	Input	Active Low	Port B address register synchronous reset
B_ADDR_ARST	Input	Active Low	Port B address register asynchronous reset
C_ADDR[]	Input		Port C address input
C_CLK	Input	Rising edge	Port C clock for C_ADDR and C_DIN
C_DIN[]	Input		Port C write data
C_WEN	Input	Active High	Port C write enable
C_BLK	Input	Active High	Port C block select

4.6.1.2 µSRAM Macro (RAM64X18)

The µSRAM macro (RAM64x18) in Libero SoC can be used directly to instantiate the µSRAM in the design. The µSRAM must be configured correctly with the appropriate values provided to the static signals before instantiating it in the design. The following figure shows the µSRAM macro (RAM64x18) available in Libero SoC.

Figure 35 • **RAM64x18 Macro**



4.6.1.3 Associated µSRAM IP Cores

4.6.1.3.1 CoreAHBLSRAM and CoreAPBLSRAM IP Cores

In addition to µSRAM macros, Libero SoC also has CoreAHBLSRAM and CoreAPBLSRAM IP cores available to access the µSRAM through AHB and APB slave interfaces. Configuration parameters such as bus (AHB/APB) data width, RAM selection (LSRAM, µSRAM), and depth of the memory can be set as per design requirement.

See *CoreAHBLSRAM Handbook* for µSRAM with AHB slave interface detailed software configuration information.

See *CoreAPBLSRAM Handbook* for µSRAM with APB slave interface detailed software configuration information.

4.6.1.3.2 CoreFIFO IP

Libero SoC IP catalog has a CoreFIFO IP, which can be configured as a soft FIFO for generation of FIFO control logic. Memory configuration can be selected as LSRAM, µSRAM or external memory as per the design requirements. For detailed software configuration information, see *CoreFIFO Handbook*. This handbook can be downloaded from the Libero SoC Catalog.

5 Math Blocks

5.1 Introduction

The SmartFusion2 SoC and IGLOO2 FPGA devices have embedded math blocks, which are optimized for digital signal processing (DSP) applications such as finite impulse response (FIR) filters, infinite impulse response (IIR) filters, fast fourier transform (FFT) functions, and encoders that require high data throughput.

The SmartFusion2 and IGLOO2 math blocks have a built-in multiplier and adder, which minimizes the external logic required to implement multiplication, multiply-add, and multiply-accumulate (MACC) functions. Implementation of these arithmetic functions 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 (µSRAM and LSRAM) to implement complex DSP algorithms efficiently. The number of math blocks varies depending on the size of the device, as shown in the following table.

5.1.1 Features

Each math block has the following features:

- High-performance and power optimized multiplications operations
- Supports 18 x 18 signed multiplication natively
- Supports 17 x 17 unsigned multiplications
- Supports dot product: the multiplier computes(A[8:0] x B[17:9] + A[17:9] x B[8:0]) x 29
- Built-in addition, subtraction, and accumulation units to combine multiplication results efficiently
- Independent third input C with data width 44 bits completely registered.
- Supports both registered and unregistered inputs and outputs
- Supports signed and unsigned operations
- Internal cascade signals (44-bit CDIN and CDOUT) enable cascading of the math blocks to support larger accumulator, adder, and subtractor without extra logic
- Supports loopback capability
- Adder support: (A x B) + C or (A x B) + D or (A x B) + C + D
- Clock-gated input and output registers for power optimizations
- Width of adder and accumulator can be extended by implementing extra adders in the FPGA fabric.

5.2 Math Block Resource Table

The following table lists the math blocks available for SmartFusion2 and IGLOO2 devices.

Table 42 • **SmartFusion2 and IGLOO2 Math Blocks Resource**

Device	Number of Math Blocks		
SmartFusion2/IGLOO2	**Rows**	**Number per Row**	**Total**
M2S005/M2GL005	1	11	11
M2S010/M2GL010	2	11	22
M2S025/M2GL025	2	17	34
M2S050/M2GL050	3	24	72
M2S060/M2GL060	3	24	72
M2S090/M2GL090	3	28	84
M2S150/M2GL150	6	40	240

5.3 Functional Description

This section provides the detailed description of the architecture of math block.

5.3.1 Architecture Overview

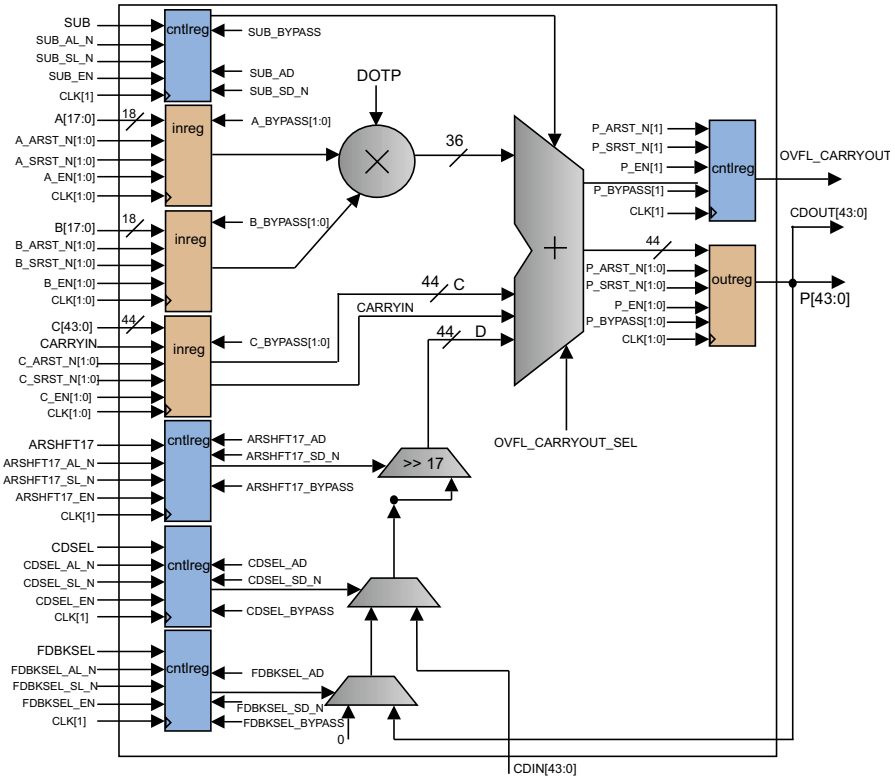
The SmartFusion2 and IGLOO2 devices can have one to three rows of math blocks in the FPGA fabric, as listed in Table 42, page 57. Math blocks can be accessed through the FPGA routing architecture and cascaded in a chain, starting from the left-most block to the right-most block.

Each math block consists of the following:

- Multiplier
- Adder or Subtractor
- I/O and Control Registers

The following illustration shows the functional block diagram of the math block

Figure 36 • **Functional Block Diagram of the Math Block**



5.3.1.1 Multiplier

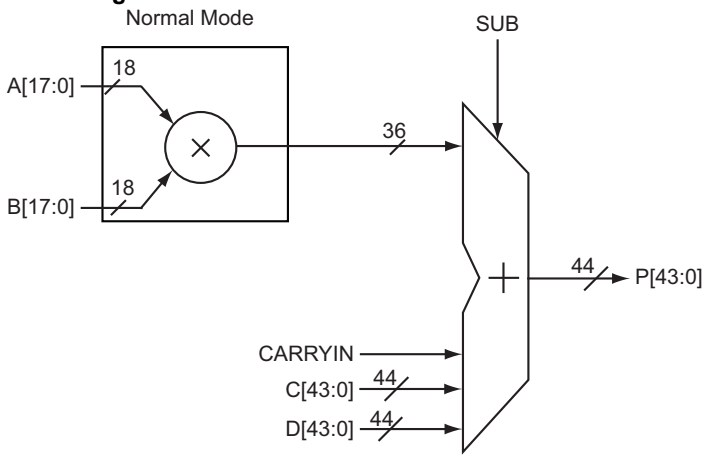
A SmartFusion2 and IGLOO2 math block can be used as a multiplier, which accepts two 18-bit inputs (A and B), and generates a 36-bit output. The math block multiplier can be configured in two different operating modes:

- Normal Mode
- DOTP Mode

5.3.1.1.1 Normal Mode

In Normal mode, the math block implements a single 18 x18 signed multiplier. The math block accepts the inputs, A [17:0] and B [17:0], and generates A × B with a 36-bit wide result. The following illustration shows the functional block diagram of the math block in Normal mode.

Figure 37 • **Functional Block Diagram of the Math Block in Normal Mode**



5.3.1.1.2 DOTP Mode

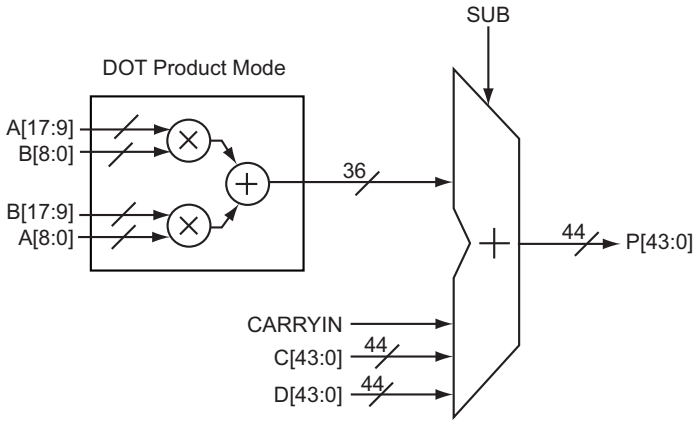
DOTP mode has two independent 9-bit x 9-bit multipliers with adder and the product sum is stored in Upper 36 bits of 44-bit register. In Dot Product (DOTP) mode, the math block implements the following equation:

(A [8:0] x B [17:9] + A[17:9] x B[8:0]) x 2^9

DOTP mode can be used to implement 9 x 9 complex multiplications.

The following illustration shows the functional block diagram of the math block in DOTP mode.

Figure 38 • **Functional Block Diagram of the Math Block in DOTP Mode**



5.3.1.2 Adder or Subtractor

The adder sums the output from the multiplier, C input, CARRYIN, or D input. The final output (P) of the adder is ((A [17:0] x B [17:0]) + C [43:0] + D [43:0] + CARRYIN).

The math block can be configured as a 2-input or 3-input adder.

- As a 2-input adder, the math block computes A x B + C or A x B + D.
- As a 3-Input adder, the math block computes A x B + C + D.

If the adder is configured as a subtractor, the adder output is ((C [43:0] + D [43:0] + CARRYIN) – (A[17:0] x B[17:0])).

5.3.1.3 I/O and Control Registers

Math blocks have built-in registers on data inputs (A, B, C), data output (P), and control signals. If required, these registers can be bypassed. All the registers in the math block have clock gating capability to reduce power consumption.

Math blocks do not have a pipeline register at the cascade input (CDIN), so pipeline registers can be added from the fabric when multiple math blocks are cascaded to implement higher bit-width multiplications.

5.3.1.3.1 C Input

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

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

5.3.1.3.2 Cascaded Input, Output, and Selection

Higher level DSP functions are supported by cascading individual math blocks in a row. The two data signals, CDIN [43:0] and CDOUT [43:0], provide the cascading capability with a cascade select input (CDSEL). Table 43, page 60 shows the selection of CDSEL for propagating CDIN to the D input of the adder. To cascade math blocks, the CDOUT of one block must feed the CDIN of another block. CDOUT to CDIN is a hardwired connection between the blocks within a row.

Two different rows can be cascaded using the fabric routing between the two rows. Extra pipeline registers may be needed to compensate for the extra delays added due to the fabric routing, which in turn will increase the latency of the chain.

The ability to cascade math blocks is useful in filter designs. For example, an FIR filter design can use cascading inputs to arrange a series of input data samples and cascading outputs to arrange a series of partial output results. The ability to cascade provides a high-performance and low power implementation of DSP filter functions because the general routing in the fabric is not used.

5.3.1.3.3 Overflow Output

Each math block has an overflow signal, OVFL_CARRYOUT. This signal indicates any overflow from the additional operation performed by the adder. This signal is also used to extend the adder data widths from the existing 44 bits using fabric. The overflow signal is also used for the implementation of 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. In SmartFusion2 and IGLOO2 math blocks, this capability is implemented using the adder's output sign bit (MSB [43] bit of the P output) and the overflow signal.

5.3.1.3.4 Shift Input

For multi-precision arithmetic, math blocks provide a right-wire-shift by 17 which is controlled by the ARSHFT17 input. Thus, 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 bigger multipliers.

5.3.1.3.5 Feedback Select Input

For accumulation operations, math block output needs to loopback to the D input of the adder block. Selection of the D input is controlled by the feedback select (FDBKSEL) input. The following table lists the selection of FDBKSEL for loopback.

Table 43 • **Truth Table for Propagating Operand D of the Adder or Accumulator**

CDSEL	FDBKSEL	ARSHFT17	Operand D
0	0	0	0
0	0	1	0
1	X	0	CDIN[43:0]
1	X	1	{{17{CDIN[43]}}, CDIN[43:18]}
0	1	0	P[43:0]

Table 43 • **Truth Table for Propagating Operand D of the Adder or Accumulator *(continued)***

CDSEL	FDBKSEL	ARSHFT17	Operand D
0	1	1	{{17{P[43]}}, P[43:18]}

5.3.1.3.6 Math Block Interface to Fabric Routing

Math blocks can access the fabric routing through interface logic routing clusters. These clusters are composed of 12 flip-flops and 12 4-input (look-up tables) LUTs. When math blocks are used, these flip-flops and LUTs act as an interface to fabric routing. When math blocks are not used, these flip-flops and LUTs can be utilized as normal flip-flops and LUTs. The interface logic clusters do not have carry chain support.

5.4 How to Use Math Blocks

The following sections describe how to use math block in an application:

- Design Flow
- Math block Use Models
- Coding Style Examples

5.4.1 Design Flow

Math blocks can be used in two ways: through inference or by using the math block primitive. Inference is done during the synthesis stage of an RTL design. Alternately, the math block primitive is available in the Libero SoC IP catalog as a component that can be used directly in the HDL file or instantiated in SmartDesign.

5.4.1.1 Using a Math Block Through Inference

Synplify Pro can infer math blocks and can configure them into appropriate modes automatically, if the RTL contains any specific multiply, multiply-accumulate, multiply-add, or multiply-subtract functions. In this case, the synthesis tool takes care of all the signal connections of the math block to the rest of the design and provides the correct values for the static signals to configure the appropriate operational mode. The tool ties unused dynamic input signals to ground and provides default values to unused static signals.

The synthesis tool maps any multiplication function with input widths of 3 or greater to math blocks. However, the mapping of multiplication functions with input widths less than 3, which are implemented in FPGA logic by default, can be controlled by the synthesis attribute (syn_multstyle). The tool also has the capability to cascade multiple math blocks, if the function crosses the limits of a single math block. For example, if an RTL function has a 35 x 35 multiplication, the synthesis tool implements this using four math blocks cascaded in a chain. It also has the capability to place the input and output registers inside the math block boundary, provided they are driven by same clock. If the registers have different clocks, the clock that drives the output register has priority, and all registers driven by that clock are placed into the math block. If the outputs are unregistered and the inputs are registered with different clocks, the input registers with the larger input have priority and are placed into the math block.

The synthesis tool supports inference of math block components across hierarchical boundaries, which means even if the multipliers, input registers, output registers, and subtracter/adders are present in different hierarchies, they can be placed into the same math block.

For more information on math block inference by Synplify Pro, see Synopsys application note on inferring *Microchip IGLOO2 MACC Blocks*.

5.4.1.2 Using the Math Block Primitive

The math block primitive available in the Libero SoC IP Catalog is called MACC. Figure 45, page 71 shows the MACC primitive with input/output port and the bit width of each port. The port list and definitions are given in the following table.

The MACC primitive can be used in designs by SmartDesign for schematic-based design entry or by directly instantiating the MACC wrapper in an HDL file as a component. For the MACC primitive, the inputs and outputs must be connected manually to the design signals. Proper values to the static signals must 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.

Unused active high dynamic signals should be connected to ground, unused active low dynamic signals should be connected to high, and unused static signals should be in default state.

Figure 39 • **Math Block Macro**

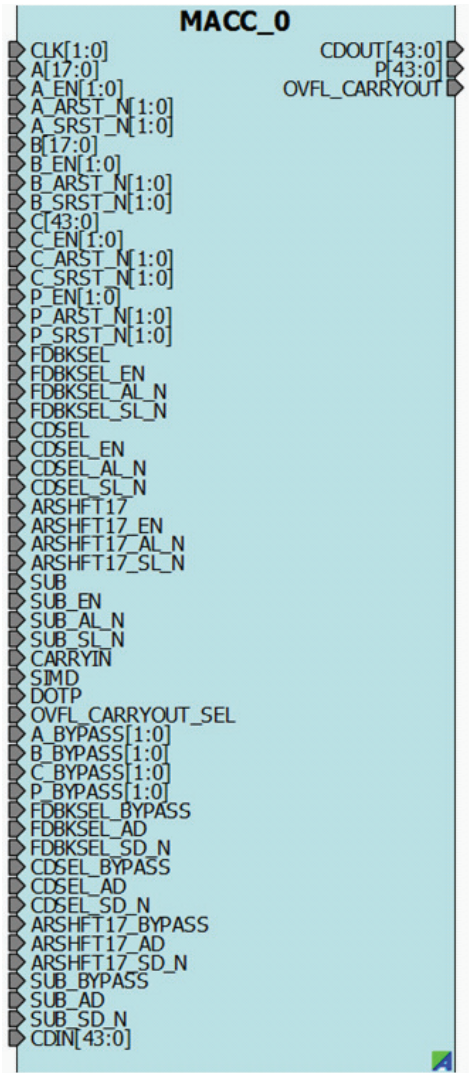


Table 44 • **Math Block Pin Descriptions**

Pin Name	Direction	Type	Polarity	Description
CLK[1:0]	Input	Dynamic	Rising Edge	Input clocks CLK[1] is the clock for A[17:9], B[17:9], P[40:18], OVFL, SHFTSEL, CDSEL, FDBKSEL, and SUB registers CLK[0] is the clock for A[8:0], B[8:0], and P[17:0] In Normal mode, ensure CLK[1] = CLK[0].
Port A (to Multiplier)				
A[17:0]	Input	Dynamic		Input Data
A_ARST_N[1:0]	Input	Dynamic	Low	Asynchronous reset A_ARST_N[1] is for A[17:9] A_ARST_N[0] is for A[8:0] When not registered, connect A_ARST_N[1:0] to logic 1. In Normal mode, ensure A_ARST_N[1] = A_ARST_N[0].
A_SRST_N[1:0]	Input	Dynamic	Low	Synchronous reset A_SRST_N[1] is for A[17:9] A_SRST_N[0] is for A[8:0] When not registered, connect A_SRST_N[1:0] to logic 1. In Normal mode, ensure A_SRST_N[1] = A_SRST_N[0].
A_EN[1:0]	Input	Dynamic	High	Enable for data registers A_EN[1] is for A[17:9] A_EN[0] is for A[8:0] When not registered, connect A_EN[1:0] to logic 1. In Normal mode, ensure A_EN[1] = A_EN[0].
A_BYPASS[1:0]	Input	Static	High	Latch input to bypass data registers A_BYPASS[1] is for A[17:9] A_BYPASS[0] is for A[8:0] When not registered, connect A_BYPASS [1:0] to logic 1. In Normal mode, ensure A_BYPASS [1] = A_BYPASS [0].
Port B (to Multiplier)				
B[17:0]	Input	Dynamic		Input Data
B_ARST_N[1:0]	Input	Dynamic	Low	Asynchronous reset B_ARST_N[1] is for B[17:9] B_ARST_N[0] is for B[8:0] When not registered, connect B_ARST_N [1:0] to logic 1. In Normal mode, ensure B_ARST_N [1] = B_ARST_N [0].

Table 44 • **Math Block Pin Descriptions *(continued)***

Pin Name	Direction	Type	Polarity	Description
B_SRST_N[1:0]	Input	Dynamic	Low	Synchronous reset B_SRST_N[1] is for B[17:9] B_SRST_N[0] is for B[8:0] When not registered, connect B_SRST_N [1:0] to logic 1. In Normal mode, ensure B_SRST_N [1] = B_SRST_N [0].
B_EN[1:0]	Input	Dynamic	High	Enable for data registers B_EN[1] is for B[17:9] B_EN[0] is for B[8:0] When not registered, connect B_EN [1:0] to logic 1. In Normal mode, ensure B_EN [1] = B_EN [0].
B_BYPASS[1:0]	Input	Static	High	Latch input to bypass data registers B_BYPASS[1] is for B[17:9] B_BYPASS[0] is for B[8:0] When not registered, connect B_BYPASS [1:0] to logic 1. In Normal mode, ensure B_BYPASS [1] = B_BYPASS[0].
Port C (to Adder)				
C[43:0]	Input	Dynamic		Input Data
CARRYIN	Input	Dynamic		Adder/accumulator's carry input
C_ARST_N[1:0]	Input	Dynamic	Low	Asynchronous reset C_ARST_N[1] is for C[43:18] C_ARST_N[0] is for C[17:0] When not registered, connect C_ARST_N[1:0] to logic 1. In Normal mode, ensure C_ARST_N[1] = C_ARST_N[0].
C_SRST_N[1:0]	Input	Dynamic	Low	Synchronous reset C_SRST_N[1] is for C[43:18] C_SRST_N[0] is for C[17:0] When not registered, connect C_SRST_N[1:0] to logic 1. In Normal mode, ensure C_SRST_N[1] = C_SRST_N[0].
C_EN[1:0]	Input	Dynamic	High	Enable for data registers C_EN[1] is for C[43:18] C_EN[0] is for C[17:0] When not registered, connect C_EN[1:0] to logic 1. In Normal mode, ensure C_EN[1] = C_EN[0].
C_BYPASS[1:0]	Input	Static	High	Latch input to bypass data registers C_BYPASS[1] is for C[43:18] C_BYPASS[0] is for C[17:0] When not registered, connect C_BYPASS[1:0] to logic 1. In Normal mode, ensure C_BYPASS[1] = C_BYPASS[0].

Table 44 • **Math Block Pin Descriptions *(continued)***

Pin Name	Direction	Type	Polarity	Description
Other Inputs				
CDIN[43:0]	Input	Cascade		Cascaded input for operand D of the adder/accumulator. The entire CDIN will be driven by another math block's CDOUT.
DOTP	Input	Static	High	Dot product mode When DOTP = 1, math block performs (A[8:0] x B[17:9] + A[17:9] x B[8:0]) x 2^9 When DOTP = 0, math block performs normal 18 x 18 multiplication operations.
SUB	Input	Dynamic	High	Subtract operation When SUB = 1, perform 2's complement subtraction to get P = C + D + CARRYIN - (A x B). When SUB = 0, perform 2's complement addition to get P = C + D + CARRYIN + (A x B).
SUB_AL_N	Input	Dynamic	Low	Asynchronous reset input for SUB input's control register.
SUB_SL_N	Input	Dynamic	Low	Synchronous reset input for SUB input's control register.
SUB_EN	Input	Dynamic	High	Enable input for SUB input's control register.
SUB_BYPASS	Input	Static	High	Latch input to bypass SUB input's data register. When logic 1, SUB is not registered.
SUB_AD	Input	Static	High	Asynchronous load data for the SUB input's control register.
SUB_SD_N	Input	Static	Low	Synchronous load data for the SUB input's control register.
ARSHFT17	Input	Dynamic	High	Arithmetic right-shift for operand D. When asserted, a 17-bit arithmetic right-shift is performed on operand D of the adder/accumulator.
ARSHFT17_AL_N	Input	Dynamic	Low	Asynchronous reset input for ARSHFT17 input's control register.
ARSHFT17_SL_N	Input	Dynamic	Low	Synchronous reset input for ARSHFT17 input's control register.
ARSHFT17_EN	Input	Dynamic	High	Enable input for ARSHFT17 input's control register.
ARSHFT17_BYPASS	Input	Static	High	Latch input to bypass ARSHFT17 input's data register. When logic '1', ARSHFT17 is not registered.
ARSHFT17_AD	Input	Static	High	Asynchronous load data for the ARSHFT17 input's control register.
ARSHFT17_SD_N	Input	Static	Low	Synchronous load data for the ARSHFT17 input's control register.

Table 44 • **Math Block Pin Descriptions *(continued)***

Pin Name	Direction	Type	Polarity	Description
CDSEL	Input	Dynamic	High	Selects CDIN for operand D of the adder/accumulator input. When CDSEL = 1, CDIN is propagated to the operand D. When CDSEL = 0, either logic 0 or feedback from output P is routed to the operand D depending upon the FDBKSEL.
CDSEL_AL_N	Input	Dynamic	Low	Asynchronous reset input for CDSEL input's control register.
CDSEL_SL_N	Input	Dynamic	Low	Synchronous reset input for CDSEL input's control register.
CDSEL_EN	Input	Dynamic	High	Enable input for CDSEL input's control register.
CDSEL_BYPASS	Input	Static	High	Latch Input to bypass CDSEL input's data register. When logic 1, CDSEL is not registered.
CDSEL_AD	Input	Static	High	Asynchronous load data for the CDSEL input's control register.
CDSEL_SD_N	Input	Static	Low	Synchronous load data for the CDSEL input's control register.
FDBKSEL	Input	Dynamic	High	Select the feedback from P for operand D of the adder or accumulator. When FDBKSEL = 1, propagate the current value of result P register. Ensure P_BYPASS[1] = 0 and CDSEL = 0. When FDBKSEL = 0, logic 0 is propagated. Ensure CDSEL = 0.
FDBKSEL_AL_N	Input	Dynamic	Low	Asynchronous reset input for FDBKSEL input's control register.
FDBKSEL_SL_N	Input	Dynamic	Low	Synchronous reset input for FDBKSEL input's control register.
FDBKSEL_EN	Input	Dynamic	High	Enable input for FDBKSEL input's control register.
FDBKSEL_BYPASS	Input	Static	High	Latch input to bypass FDBKSEL input's data register. When logic 1, FDBKSEL is not registered.
FDBKSEL_AD	Input	Static	High	Asynchronous load data for the FDBKSEL input's control register.
FDBKSEL_SD_N	Input	Static	Low	Synchronous load data for the FDBKSEL input's control register.

Table 44 • **Math Block Pin Descriptions *(continued)***

Pin Name	Direction	Type	Polarity	Description
Output Port				
P[43:0]	Output			Result data out Normal mode P = C + D + CARRYIN + (A x B) when SUB = 0 P = C + D + CARRYIN - (A x B) when SUB = 1 DOTP mode P = C + D + CARRYIN + ((A[8:0] x B[17:9] + A[17:9] x B[8:0]) x 29) when SUB = 0 P = C + D + CARRYIN - ((A[8:0] x B[17:9] + A[17:9] x B[8:0]) x 29) when SUB = 1
OVFL_CARRYOUT	Output			Overflow output Normal mode if C + D + CARRYIN +/- (A x B) > (243 - 1), then OVFL_CARRYOUT = 1 if C + D + CARRYIN +/- (A x B) < - (243), then OVFL_CARRYOUT = 1 else OVFL_CARRYOUT = 0. DOTP mode if C + D + CARRYIN +/- ((A[8:0] x B[17:9] + A[17:9] x B[8:0]) x 29) > (243- 1), then OVFL_CARRYOUT = 1 if C + D + CARRYIN +/- ((A[8:0] x B[17:9] + A[17:9] x B[8:0]) x 29) < - (243), then OVFL_CARRYOUT = 1 else OVFL_CARRYOUT = 0.
OVFL_CARRYOUT_SEL	Input	Static	High	Input to the adder for generating the overflow bit or an external bit, which finally comes as an output on the OVFL_CARRYOUT port. The overflow bit indicates the overflow generated in the addition process. The external bit is generated to extend the adder into the fabric. In this case, P[43], C[43], and D[43] are basically not representing the sign bit. When OVFL_CARRYOUT_SEL = 1, OVFL_CARRYOUT is the external bit for fabric extension. Otherwise, OVFL_CARRYOUT is the overflow output.
CDOUT[43:0]	Output			Cascade output of result P. CDOUT is the same as P. It is used to drive the CDIN of another math block.

Table 44 • **Math Block Pin Descriptions *(continued)***

Pin Name	Direction	Type	Polarity	Description
P_ARST_N[1:0]	Input	Dynamic	Low	Asynchronous reset input for P and OVFL_CARRYOUT control registers P_ARST_N [1] is for OVFL_CARRYOUT and P[43:18] P_ARST_N [0] is for P[17:0] When not registered, connect P_ARST_N [1:0] to logic 1. In Normal mode, ensure P_ARST_N [1] = P_ARST_N [0].
P_SRST_N[1:0]	Input	Dynamic	Low	Synchronous reset input for P and OVFL_CARRYOUT control registers P_SRST_N [1] is for OVFL_CARRYOUT and P[43:18] P_SRST_N [0] is for P[17:0] When not registered, connect P_SRST_N [1:0] to logic 1. In Normal mode, ensure P_SRST_N [1] = P_SRST_N [0].
P_EN[1:0]	Input	Dynamic	High	Enable input for P and OVFL_CARRYOUT control registers P_EN[1] is for OVFL_CARRYOUT and P[43:18] P_EN[0] is for P[17:0] When not registered, connect P_EN[1:0] to logic 1. In Normal mode, ensure P_EN[1] = P_EN[0].
P_BYPASS[1:0]	Input	Static	High	Latch input for P and OVFL_CARRYOUT control registers P_BYPASS[1] is for OVFL_CARRYOUT and P[43:18] P_BYPASS[0] is for P[17:0] When not registered, connect P_BYPASS[1:0] to logic 1. In Normal mode, ensure P_BYPASS[1] = P_BYPASS[0].

Note: The asynchronous reset has priority over the synchronous reset and enable signal of the Input/Output registers.

Note: Asynchronous load input has higher priority than the synchronous load input.

5.4.2 Math Block Use Models

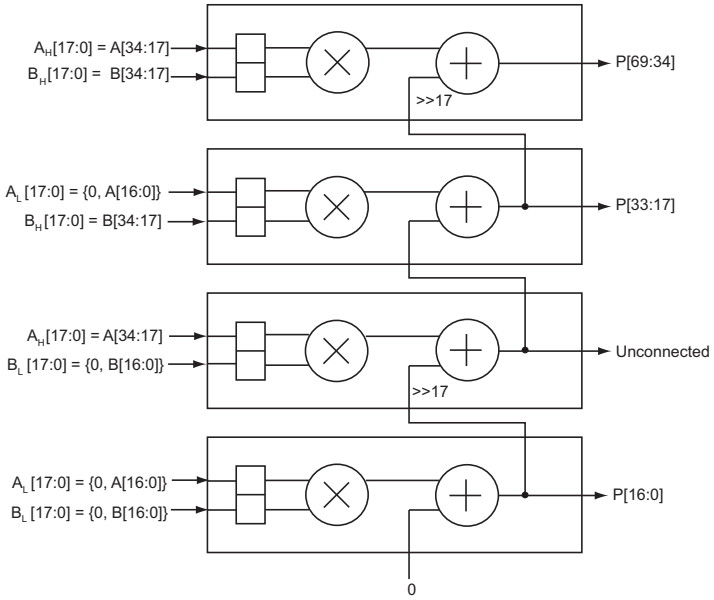
This section describes a few use models for SmartFusion2 and IGLOO2 math blocks.

5.4.2.1 Use Model 1: Non-Pipelined Implementation of the 35 x 35 Multiplier

35 x 35 multipliers are useful for applications which require more than 18-bit precision. Non-pipelined implementation is typically used for low speed applications. A 35 x 35 multiplier can be constructed using 4 math blocks in a single row, connected in a cascade. The following illustration shows a typical implementation of a non-pipelined 35 x 35 multiplier.

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

Figure 40 • **Non-Pipelined 35 x 35 Multiplier**

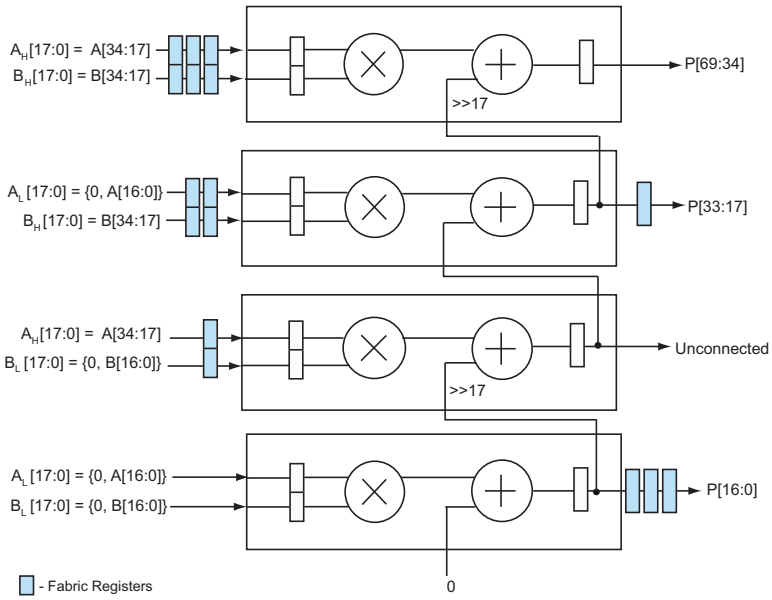


5.4.2.2 Use Model 2: Pipelined Implementation of the 35 x 35 Multiplier

The SmartFusion2 and IGLOO2 math blocks have built-in registers on all input and output ports. To implement high-speed multipliers, extra registers are added to the input or output side of the math blocks to balance the pipeline latency. These extra registers are implemented in the fabric.

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

Figure 41 • **Pipeline 35 x 35 Multiplier**



5.4.2.3 Use Model 3: Implementation of 9-Bit Complex Multiplication

Complex multiplication implemented using a math block in DOTP mode requires additional 2's complement logic in the fabric for negating the Q input. The DOTP implementation in the following illustration shows the optimized way of implementing the 2's complement with minimal logic in the fabric.

For two complex numbers X + jY, P + jQ, the complex multiplication is shown in the following equation:

Multiplication Result = Real part + Imaginary Part = (PX - QY) + j (PY + QX)

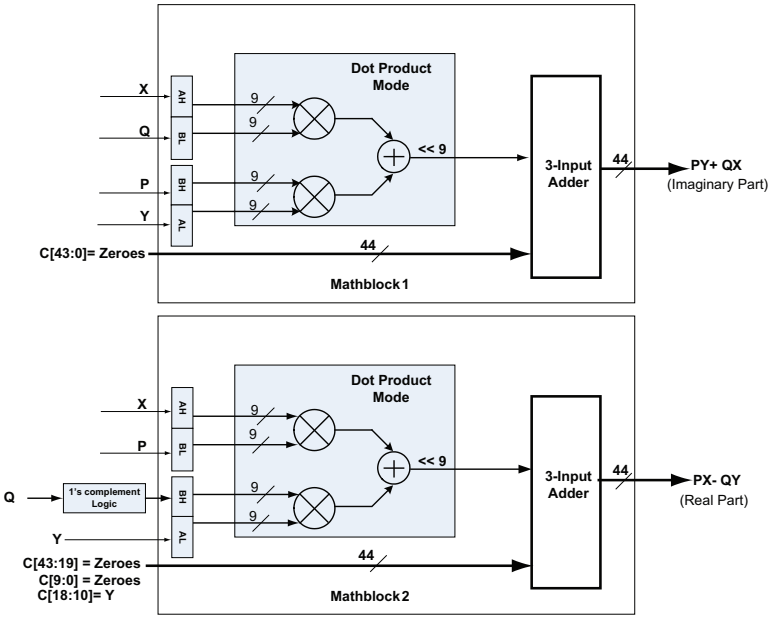
In the preceding equation, real part (PX-QY) requires that '-Q' for the multiplication result. This can be compute using the one's complement of Q and add the Y using the c input (since -Q = ~Q+1).

Imaginary part = P × Y + Q × X

Real part = P × X + (~Q) × Y + Y

The following illustration shows the implementation of 9 × 9 complex multiplication using a math block configured in DOTP mode.

Figure 42 • **9-Bit Complex Multiplication Using DOTP Mode**



5.4.2.4 Use Model 4: Multi-Threading and Multi-Channeling

Math blocks support a multi-threading option where the same math block can be used for performing more than one computation by time multiplexing. Time multiplexing can be done easily for designs with low sample rates.

The multi-threading capability, if implemented for a chain of math blocks, is called multi-channeling.

Multi-channeling can be used to implement multi-channel FIR filters where the same math block chain can be used to process multiple input channels by time multiplexing the math block chain. Multi-channel filtering is used in applications such as wireless communications, image processing, and multimedia applications. The math block uses its C input for multi-threading and multi-channeling, but fabric registers are also required for implementation.

5.4.2.5 Use Model 5 - Rounding and Trimming

5.4.2.5.1 Rounding

Rounding can be computed by adding a fixed term and a variable term to the input value to be rounded, and then truncating. The fixed term can be feed using the C-Input of the math block and the value depends on the number of decimal points required after rounding. The variable term is always a single bit in the least-significant position whose value may be determined from the input value based on the type of rounding.

Types of rounding are:

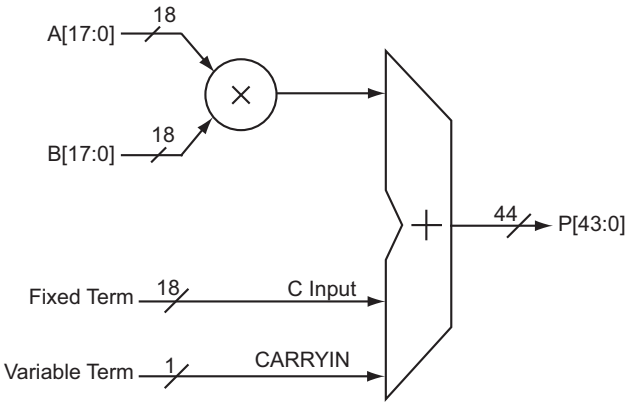
- Round to the adjacent even integer: The variable term is determined from the 20 bit of the input value.
- Round towards zero: The variable term is determined from the sign bit of the input value. For example, 1.5 rounds to 1 and -1.5 rounds to -1.

The following table lists the examples for 6-bit values including three fraction bits.

Table 45 • **Rounding Examples**

Input Value		Fixed Term	Round To Even				Round Toward Zero			
Decimal	Binary	C-Input	Variable Term	Sum	Truncated Sum	Decimal	Variable Term	Sum	Truncated Sum	Decimal
2.5	010.100	0.011	000.000	010.111	010	2	000.000	010.111	010	2
1.5	001.100	0.011	000.001	010.000	010	2	000.000	001.111	001	1
-1.5	110.100	0.011	000.000	110.111	110	-2	000.001	111.000	111	-1
-2.5	101.100	0.011	000.001	110.000	110	-2	000.001	110.000	110	-2

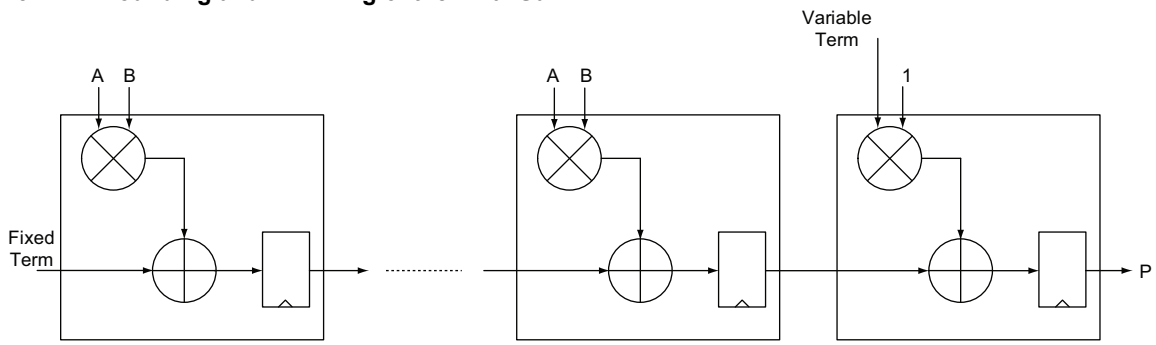
Figure 43 • **Rounding Using C-Input and CARRYIN**



5.4.2.5.2 Trimming

Trimming of the Final Sum: Applications like IIR and FFT often requires the rounding and trimming of the final result (for example, last output of a cascade chain or the final value read from an accumulator). The addition of the rounding terms can be done as shown in the The following illustration and final result can be trimmed in fabric.

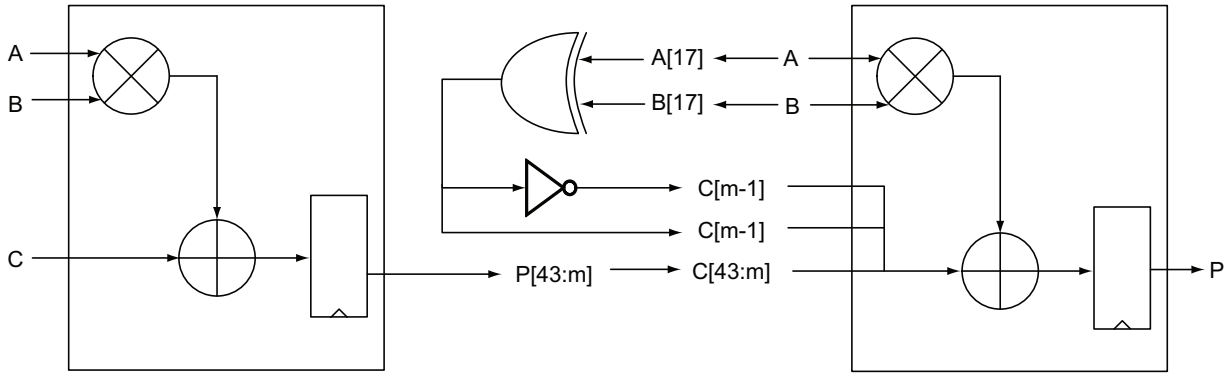
Figure 44 • **Rounding and Trimming of the Final Sum**



Trimming of Grouped Sums: When computing very large dot products (for example, a large, fully-enumerated FIR) it is good to avoid overflow by breaking the sum into a few groups, trimming the sum for each group, and only then combining the groups' sums into a final result. The rounding of each group's sum can be done as shown in the following illustration. The trimming of each group's sum and summation of the final result can be done in the fabric. Trimming can be done between the output of each cascade and the final fabric adder.

Trimming of Products: The following illustration shows the implementation of rounding all products towards zero and then trimming the least significant m bits of the product. As long as there are no additive terms other than the products, it is possible to equivalently trim the partial sums instead of the products. Round towards zero can be done using sign bit of the product (A × B) from the sign bits of the incoming factors A and B using an EXOR.

Figure 45 • **Rounding and Trimming of the Final Sum**



5.4.3 Coding Style Examples

The following code examples illustrate coding styles from which the synthesis tool can infer and implement SmartFusion2 and IGLOO2 math blocks.

Note: Examples provided are only in Verilog. VHDL examples are provided on request.

Example 1: 18 x 18 Signed Multiplication – Non-Registered

The following code is for an 18 x 18-bit signed multiplier. The input and output registers are configured in Transparent mode. The synthesis tool maps the code into one math block.

```
module sign18x18_mult ( in1, in2, out1 );
input signed [17:0] in1, in2;
output signed [40:0] out1;
wire signed [40:0] out1;
assign out1 = in1 × in2;
endmodule
```

Example 2: 18 x 18 Signed Multiplication – Registered

The following code is for an 18 x 18 signed multiplier. The inputs and outputs are registered, with a synchronous active low reset signal. The synthesis tool maps the code into one math block.

```
module sign18x18_mult_reg ( in1, in2, clock, reset, out1 );
input signed [17:0] in1, in2;
input clock;
input reset;
output signed [40:0] out1;
reg signed [40:0] out1;
reg signed [17:0] in1_reg, in2_reg;
always @ ( posedge clock )
begin
if ( ~reset )
begin
in1_reg <= 18'b0;
in2_reg <= 18'b0;
out1 <= 41'b0;
end
else
begin
in1_reg <= in1;
n2_reg <= in2;
out1 <= in1_reg × in2_reg;
end
end
endmodule
```

Example 3: 17 x 17-Bit Unsigned Multiplier with Different Resets

The following code is for a 17 x 17-bit unsigned multiplier, which has input and output registers with different asynchronous resets. The synthesis tool maps the code into one SmartFusion2 or IGLOO2 math block.

```
module mult_17x17unsign( in1, in2, clock, reset1, reset2, out1 );
input [16:0] in1, in2;
input clock, reset1, reset2;
output [33:0] out1;
reg [33:0] out1;
reg [16:0] in1_reg, in2_reg;
always @(posedge clock or negedge reset1)
begin
if (~reset1 )
begin
in1_reg <= 17'b0;
in2_reg <= 17'b0;
end
else
begin
in1_reg <= in1;
in2_reg <= in2;
end
end
always @(posedge clock or negedge reset2)
begin
if (~reset2 )
begin
out1 <= 34'b0;
end
else
begin
out1 <= in1_reg × in2_reg;
end
end
endmodule
```

Example 4: 17 x 17-Bit Unsigned Multiplier with Different Clocks

This example shows an unsigned multiplier with inputs and outputs that are registered with different clocks: clock1 and clock2. In this case, the synthesis tool places only the output registers and the multiplier into the SmartFusion2 or IGLOO2 math block. The input registers are implemented in FPGA logic outside the math block.

```
module mult_17x17unsign ( in1, in2, clock1, clock2, outl );
input [16:0] in1, in2;
input clock1,clock2;
output [33:0] outl;
reg [33:0] outl;
reg [16:0] in1_reg, in2_reg;
always @ ( posedge clock1 )
begin
in1_reg <= in1;
in2_reg <= in2;
end
always @ ( posedge clock2 )
begin
outl <= in1_reg × in2_reg;
end
endmodule
```

Example 5: Multiplier-Adder

In the code below. the output of a multiplier is added with another input. Inputs and outputs are registered and have enables and synchronous resets. The synthesis tool maps the code into one SmartFusion2 or IGLOO2 math block.

```
module mult_add_v1( in1, in2, in3, clock, reset, en, out1);
input [16:0] in1, in2;
input [33:0] in3;
input clock, reset, en;
output [34:0] out1;
reg [34:0] out1;
reg [16:0] in1_reg, in2_reg;
reg [33:0] in3_reg;
wire [33:0] mult_out;
always @(posedge clock)
begin
if (~reset)
begin
in1_reg <= 17'b0;
in2_reg <= 17'b0;
in3_reg <= 34'b0;
end
else
begin
if (en == 1'b1)
begin
in1_reg <= in1;
in2_reg <= in2;
in3_reg <= in3;
end
end
end
always @(posedge clock)
begin
if (~reset)
begin
out1 <= 35'b0;
end
else
begin
if (en == 1'b1)
begin
out1 <= {1'b0, mult_out} + {1'b0, in3_reg};
end
end
end
assign mult_out = in1_reg × in2_reg;
endmodule
```

Example 6: Multiplier-Subtractor

There are two ways to implement multiplier and subtract logic. The synthesis tool places the logic differently, depending on how it is implemented.

- Subtract the result of multiplier from an input value (P = Cin – mult_out). The synthesis tool places all logic in the math block.
- Subtract a value from the result of the multiplier (P = mult_out – Cin). The synthesis tool places only the multiplier in the math block. The subtractor is implemented in FPGA logic outside the math block.
    - Unsigned MultSub Example (P = Cin – Mult_out) - Implemented in single math block.

```
module mult_sub ( in1, in2, in3, clk, rst, out1 );
input [16:0] in1, in2;
input [36:0] in3;
input clk;
input rst;
output [39:0] out1;
reg [39:0] out1;
reg [16:0] in1_reg, in2_reg;
always @ ( posedge clk )
begin
if (~rst)
begin
in1_reg <= 17'b0;
in2_reg <= 17'b0;
out1 <= 40'b0;
end
else
begin
in1_reg <= in1;
in2_reg <= in2;
out1 <= in3 - (in1_reg × in2_reg);
end
end
endmodule
```

    - Unsigned MultSub Example (P = Mult - Cin) - Multiplier is implemented in math block and subtractor in FPGA logic

```
module mult_sub_v2 ( in1, in2, in3, clk, rst, out1 );
input [16:0] in1, in2;
input [36:0] in3;
input clk;
input rst;
output [39:0] out1;
reg [39:0] out1;
reg [16:0] in1_reg, in2_reg;
always @ ( posedge clk )
begin
if ( ~rst )
begin
in1_reg <= 17'b0;
in2_reg <= 17'b0;
out1 <= 40'b0;
end
else
begin
in1_reg <= in1;
in2_reg <= in2;
out1 <= (in1_reg × in2_reg) - in3;
end
end
endmodule
```

Example 7: Signed 35 x 35 Multiplication

The code below implements a signed 35 x 35 multiplication function. The synthesis tool uses 4 cascaded math blocks to implement this multiplication function.

```
module sign35x35_mult ( in1, in2, out1);
input signed [34:0] in1;
input signed [34:0] in2;
output signed [69:0] out1;
wire signed [69:0] out1;
assign out1 = in1 × in2;
endmodule
```

Example 8: Signed 35 x 35 Multiplication with Two Pipelined Register Stages

The code below implements a signed 35 x 35 multiplication function with two pipelined register stages. The synthesis tool uses four cascaded math blocks to implement this multiplication function. The synthesis tool first infers pipeline registers at the input, output of the SmartFusion2 or IGLOO2 math block and controls pipeline latency by balancing the number of register stages. To balance the stages, the tool adds additional registers at the input or output of the math block as required, implemented in the fabric logic.

```
module sign35x35_mult ( in1, in2, clk, rst, out1 );
input signed [34:0] in1, in2;
input clk;
input rst;
output signed [69:0] out1;
reg signed [69:0] out1;
reg signed [34:0] in1_reg, in2_reg;
always @ ( posedge clk or negedge rst)
begin
if ( ~rst )
begin
in1_reg <= 35'b0;
in2_reg <= 35'b0;
out1 <= 70'b0;
end
else
begin
ini_reg <= in1;
in2_reg <= in2;
out1 <= ini_reg × in2_reg;
end
end
endmodule
```

6 I/Os

6.1 Introduction

SmartFusion2 and IGLOO2 devices have different types of inputs/outputs (I/Os), such as multi-standard I/Os (MSIO and MSIOD), double-data-rate I/Os (DDRIO), and dedicated I/Os based on functional usage.

MSIO, MSIOD, and DDRIO provide programmable I/O features such as drive strength, slew rate, input delay, weak pull-up, and weak pull-down for several voltages. These programmable I/O features are explained in detail in the I/O Programmable Features, page 91.

DDRIO is an MSIO optimized for LPDDR/DDR2/DDR3 performance. In SmartFusion2 and IGLOO2 devices, there are two DDR subsystems: the fabric DDR and memory subsystem (MDDR) controllers, which control external DDR memory. DDRIOs can be connected to the respective DDR subsystem PHYs or used directly as user I/Os. For more information about DDR subsystem, see the *UG0446: SmartFusion2 and IGLOO2 FPGA High Speed DDR Interfaces User Guide*.

MSIO, MSIOD, and DDRIO can be configured as MSS, HPMS, or fabric I/Os, whereas dedicated I/Os can be used for a single purpose, serializer/deserializer (SerDes), device reset, and clock functions.

The MSIO, MSIOD, and DDRIO are configured at power-up through the flash bits used to initialize the fabric register blocks. This is automatically done using the Libero SoC software.

6.2 Functional Description

SmartFusion2 and IGLOO2 I/Os are classified into the following three categories depending on their functional usage:

- MSIO, MSIOD, and DDRIO
- JTAG I/O
- Dedicated I/Os

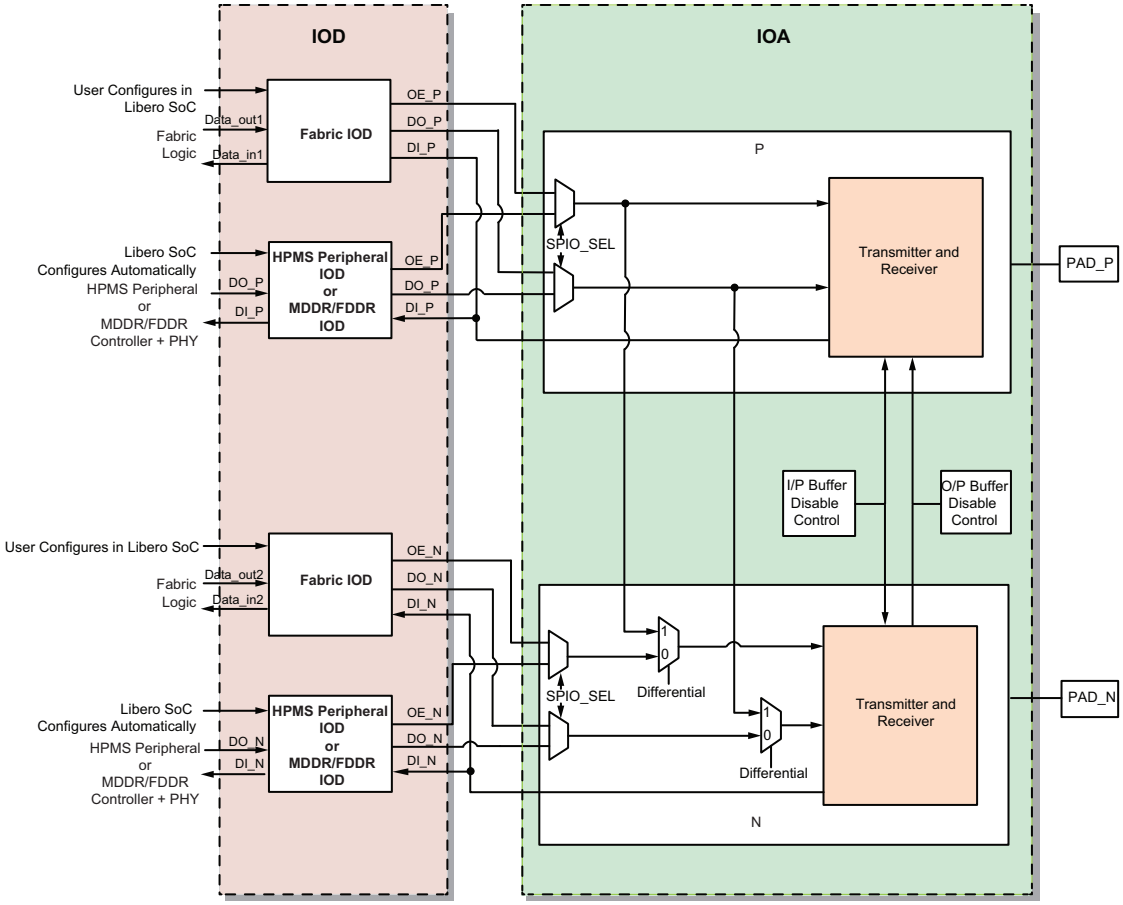
The following illustration shows the top-level view of I/O interconnection between the fabric logic and the FDDR.

The DDRIOs are shared among the fabric logic and MDDR/FDDR. When the MDDR/FDDR controller is used, the Libero SoC software automatically assigns and configures the controller signals to the respective DDRIOs. The SPIO_SEL signal (as shown in the following illustration) determines whether the fabric logic or MDDR/FDDR/MSS peripheral is connected to the corresponding I/Os. This selection is set automatically by the Libero SoC software during programming. When the MDDR/FDDR controller is not used, the respective DDRIOs are available to the fabric logic, as shown in the following illustration.

Similarly, when the MSS or HPMS peripheral is used, Libero SoC automatically assigns and configures the MSS or HPMS peripheral's signals to the MSIOs and the MSIOD.The SPIO_SEL signal (as shown in Figure 46, page 79) determines whether the fabric logic, MSS, or HPMS peripheral is connected to the corresponding I/Os.This selection is set automatically by the Libero SoC software during programming. When the MSS or HPMS peripherals are not used, the respective I/Os are available to the fabric logic, as shown in the following illustration.

For the fabric logic, each I/O port of the design must be individually assigned to I/Os in Libero SoC.

Figure 46 • **I/O Interconnection**



MSIO, MSIOD, and DDRIO can be configured as one differential I/O or two single-ended I/Os. Single-ended I/Os are composed of two separate I/Os named P and N, as shown in the preceding figure.

The differential I/O is implemented by pairing up P and N. The differential standards are implemented as true differential outputs and not complementary single-ended outputs.

An I/O consists of a bidirectional I/O buffer. The I/O is divided into two main sections, as shown in the preceding illustration:

- Digital: IOD (fabric and MDDR/FDDR/HPMS peripherals)
- Analog: IOA

The digital section (IOD) generates output enable (OE), data out (DO), and data in (DIN) signals for both P and N. See Fabric Architecture, page 4 for more details on IOD.

As shown in Figure 47, page 81, analog blocks (IOA) together form a differential pair, which supports differential and pseudo differential modes of operation. The differential pair is composed of a true IOA and a complement IOA. The true IOA is called IOP (with positive polarity relative to the DO/DIN data signals of the P cell). The complement IOA is called ION (with negative polarity relative to the DO/DIN data signals of the N cell). The IOA blocks form a ring around the periphery of the device (excluding the SerDes channel edge).

The top and bottom edge of the IOA order of the device starts with P on the left and N on the right. The left and right edges use N on the top and P on the bottom. There is one IOD for each pair of IOAs.

To support different differential standards, SmartFusion2 and IGLOO2 use a pair of regular I/O cells: P and N. These two I/O cells of MSIO, MSIOD, and DDRIO can be configured as separate single-ended I/Os or configured as one differential I/O pair. In differential output mode, the output data signal is driven

out on both the P cell and N cell as a differential pair, where the true signal is on pad P and the complement signal is on N pad.

The P and N output signals are complementary as required by the DDR1/DDR2/DDR3 standards for the CK and DQS signals. The P and N cells have to be placed next to each other, as a pair, to minimize skew between the two output signals of the differential pair.

IOA has transmitter and receiver buffers for the P and N pair as shown in Figure 47, page 81). The transmit and receive buffers support various I/O standards and contain the following modules:

- Transmit Buffer
- Receive Buffer
- Low-Power Exit
- On-Die Termination

6.2.1 Transmit Buffer

Transmit and receive buffers transfer signals between the FPGA fabric and the IOA. They also transfer signals between the MDDR, FDDR, MSS peripherals, HPMS peripherals, and the IOA.

The OE_P and OE_N control the direction of I/O buffers, as shown in Figure 47, page 81. When an I/O is operated as a single-ended I/O, OE_P and OE_N individually control the P and N I/O buffers. When an I/O is operated as a differential I/O, OE_P controls both the P and N I/O buffers. The dynamic OE disables or enables the output buffer for all the standards.

6.2.2 Receive Buffer

The enabling and disabling of the input buffer is controlled automatically by Libero SoC.

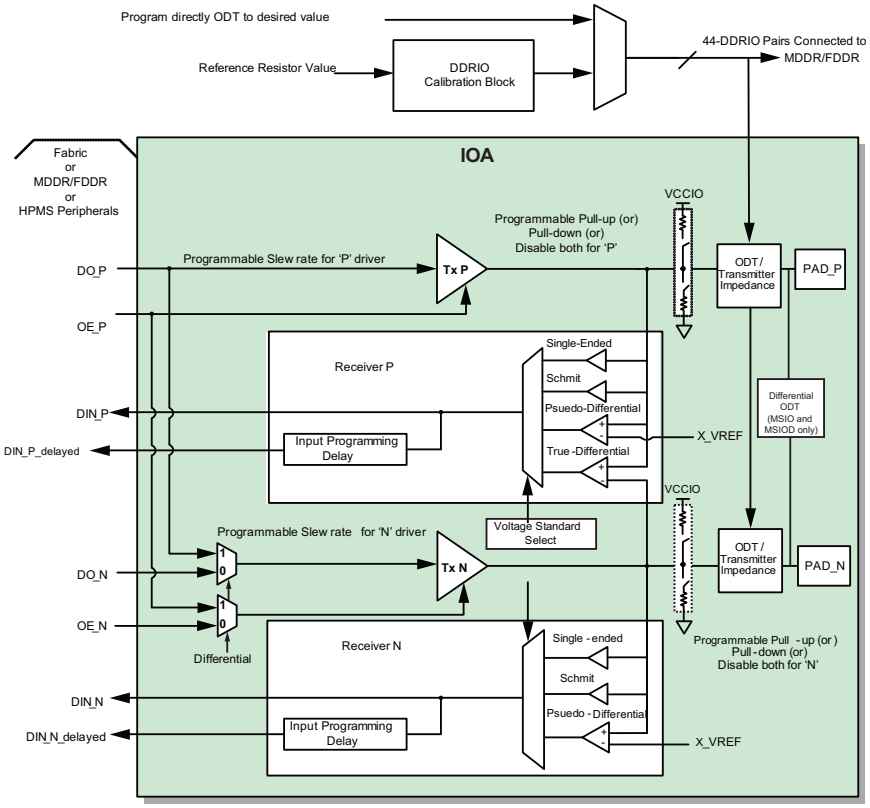
The I/O receiver can be made to operate in four different modes, as shown in Figure 47, page 81. These modes are selected based on flash configuration bits, which are configured during programming, after power-on. Following are the four modes of operation of the receiver:

- True differential
- Pseudo-differential
- Single-ended
- Schmitt trigger

In true differential mode, P and N pad inputs are fed to the comparator, whereas in pseudo-differential mode, each pad input is compared to reference with an external reference voltage. Figure 47, page 81 shows the detailed IOA structure.

The I/O input can be configured as a schmitt trigger receiver or a single-ended receiver. When schmitt trigger receiver is selected, the input buffer has hysteresis that filters noise at the receiver and prevents double glitching caused by the noisy input edges.

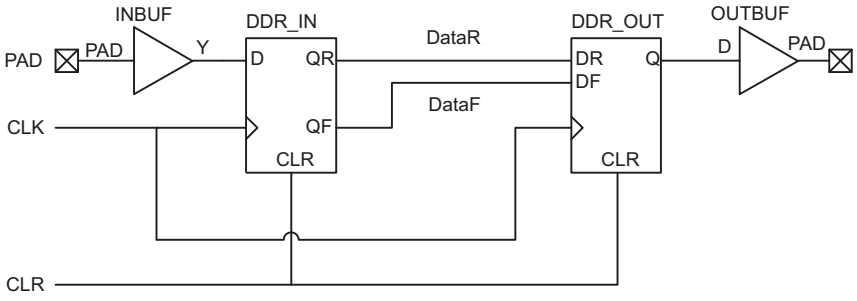
Figure 47 • **IOA Architecture**



The MSIO and MSIOD in SmartFusion2/IGLOO2 devices support DDR mode. In DDR mode, the new data is present on every transition (or clock edge) of the clock signal. DDR mode doubles the data transfer rate as compared to single data rate (SDR) mode where new data is present on one transition (or clock edge) of the clock signal. Low-power flash devices have DDR circuitry built in to the I/O tiles. I/Os are configured in DDR mode by instantiating the DDR macros (DDR_OUT or DDR_IN) in the RTL design and buffers, as shown in the following illustration. See the *DS0128: SmartFusion2 and IGLOO2 Datasheet* for more information.

Note: DDRIOs are different from the DDR macro (DDR_IN and DDR_OUT).

Figure 48 • **DDR Support in Low Power Flash Devices**



6.2.3 Low-Power Exit

Low-power exit logic indicates to the system controller that the I/Os have either matched the pre-defined signature bit or have detected activity on the selected I/O after the chip entered low-power mode. For details on signature and activity modes, see Signature Mode, page 105 and Activity Mode, page 105.

6.2.4 On-Die Termination

On-die termination (ODT) improves the signaling environment by reducing the electrical discontinuities and enables reliable operation at higher signaling rates.

For more information on the programmed ODT values for DDRIO, MSIO, and MSIOD, see I/O Programmable Features, page 91.

6.3 I/O Banks

I/Os are grouped on the basis of I/O voltage standards. The grouped I/Os of each voltage standards form an I/O bank. Each I/O bank has dedicated I/O supply and ground voltages; therefore, only I/Os with compatible standards can be assigned to the same I/O voltage bank.

Every I/O bank has input and output buffers to support a wide range of standards, each requiring a different VDDI voltage, and where applicable, a different reference voltage (V_{REF}). These voltages are externally supplied and connected to supply pins, which serve banks of I/Os.

For I/O pin name and bank assignments for different device packages, see the *DS0115: SmartFusion2 Pin Descriptions Datasheet* and *DS0124: IGLOO2 Pin Descriptions Datasheet* documents.

6.4 Simultaneous Switching Noise

6.4.1 GND Bounce and V_{DDI} Bounce

When multiple output drivers switch simultaneously, they induce a voltage drop in the chip/package power distribution. The simultaneous switching momentarily raises the ground voltage within the device relative to the system ground. This apparent shift in the ground potential to a non-zero value is known as simultaneous switching noise (SSN) or, more commonly, ground bounce.

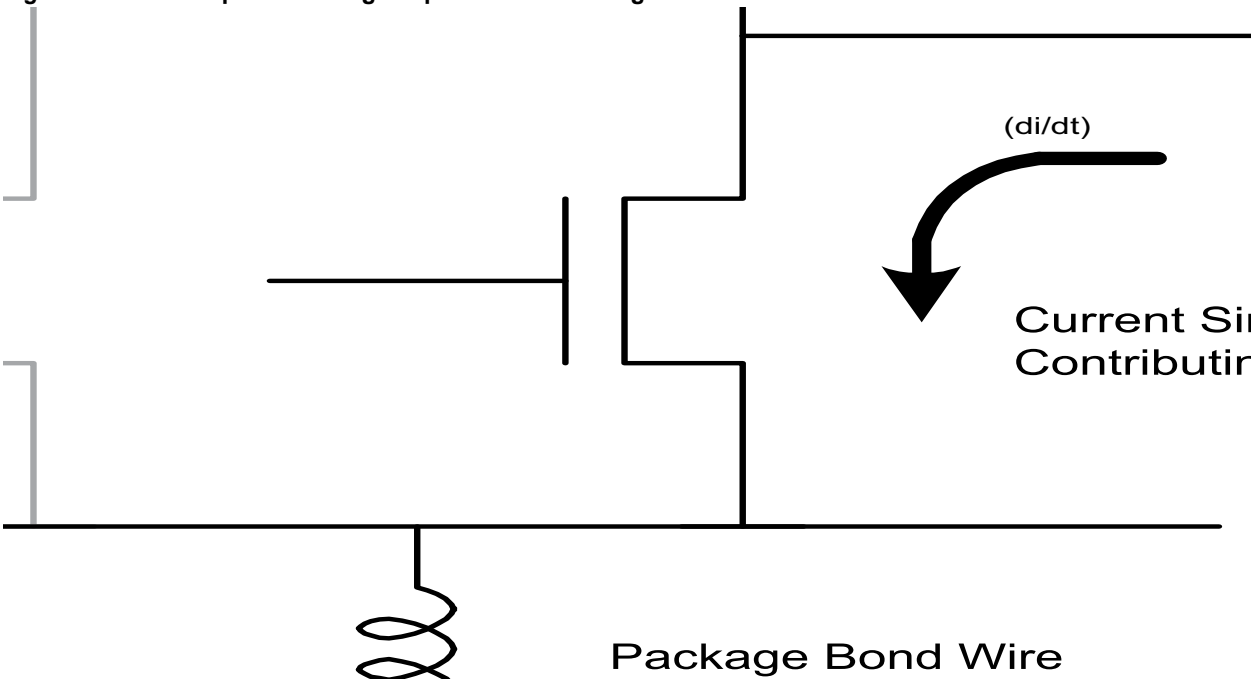
The ground bounce voltage is related to the inductance present between the device ground and the system ground, and the amount of current sunk by each output. It is given in the following equation:

$V = L \times di/dt$

An I/O switching from high to low or low to high is actually discharging or charging the capacitor that loads the I/O. The resulting value of di/dt is cumulative and increases with the number of simultaneously switching outputs (SSOs). Therefore, the higher di/dt, the higher the ground bounce amplitude.

Where does the inductance come from? The device ground is connected to the system ground (PCB ground) through a series of inductors, comprised of package bond wire, package trace, and board inductance as shown in the following figure.

Figure 49 • **A Sample Switching Output Buffer Showing Parasitic Inductance**



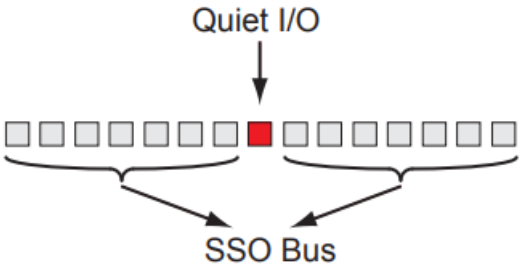
As a result, the higher Leff, the higher the amplitude will be. Problems may arise when this ground bounce gets transferred to the outside through output buffers driving low. If the bounce is higher than the VIL threshold of the input being driven, there is a possibility that the glitch will be recognized as a legal logic '1'.

The same phenomenon applies to VCC and is called VCC bounce. Both ground bounce and VCC bounce are important noise parameters, but devices usually tend to have more noise margin near the high level ('1') than near the low level ('0'). Therefore, ground bounce is considered more often.

6.4.1.1 SSO Effects

The total number of SSOs for each bus is determined by identifying the outputs that are synchronous to a single clock domain, have their clock-to-out times within ±300 ps of each other, and are placed next to each other on die pads that are on both sides of a sensitive I/O, as shown in the following figure.

Figure 50 • **Basic Block Diagram of Quiet I/O Surrounded by SSO Bus**



The sensitive I/O affected by SSO is sometimes referred to as the victim I/O or quiet I/O. SSOs may affect the victim I/O if the total number of SSOs on both sides of the victim I/O exceeds the SmartFusion2 / IGLOO2 device SSO recommendation. It is important to note that the SSOs must be referenced to the die pads and not package pins.

Note: SSO trace load for MSIO and MSIOD is 500 R in parallel with 50 pF load. SSO trace load for DDRIO is 17 pF.

Table 46 • **MSIO SSO Guidelines for M2S010 - FG484 Device**

LVTTL (SSOs Causing)			LVCMOS25 (SSOs Causing)			LVCMOS18 (SSOs Causing)			LVCMOS15 (SSOs Causing)			LVCMOS12 (SSOs Causing)		
Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce
20	28	18	16	28	20	12	28	28	8	28	28	4	28	28
16	28	22	12	28	28	10	28	28	6	28	28	2	28	28
12	28	28	8	28	28	8	28	28	4	28	28			
8	28	28	6	28	28	6	28	28	2	28	28			
4	28	28	4	28	28	4	28	28						
2	28	28	2	28	28	2	28	28						

Table 47 • **MSIOD SSO Guidelines for M2S010 - FG484 Device**

LVCMOS25 (SSOs Causing)			LVCMOS18 (SSOs Causing)			LVCMOS15 (SSOs Causing)			LVCMOS12 (SSOs Causing)		
Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce
12	28	18	10	28	28	6	28	28	4	28	28
8	28	24	8	28	28	4	28	28	2	28	28
6	28	28	6	28	28	2	28	28			
4	28	28	4	28	28						
2	28	28	2	28	28						

Table 48 • **DDRIO SSO Guidelines for M2S010 - FG484 Device**

LVCMOS25 (SSOs Causing)			LVCMOS18 (SSOs Causing)			LVCMOS15 (SSOs Causing)			LVCMOS12 (SSOs Causing)		
Drive Strength (mA)	GND Bounce	V_{CCI} Bounce	Drive Strength (mA)	GND Bounce	V_{CCI} Bounce	Drive Strength (mA)	GND Bounce	V_{CCI} Bounce	Drive Strength (mA)	GND Bounce	V_{CCI} Bounce
16	28	28	16	28	28	12	28	28	8	28	28
12	28	28	12	28	28	10	28	28	4	28	28
8	28	28	10	28	28	8	28	28	2	28	28
6	28	28	8	28	28	6	28	28			
4	28	28	6	28	28	4	28	28			
2	28	28	4	28	28	2	28	28			
			2	28	28						

Table 49 • MSIO, MSIOD, and DDRIO SSO Guidelines for M2S025 - FG484 Device

MSIO (SSOs Causing)				MSIOD (SSOs Causing)				DDRIO (SSOs Causing)			
I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce
LVTTL	20	28	12								
LVCMOS25	16	28	14	LVCMOS25	12	28	18	LVCMOS25	16	18	28
LVCMOS18	12	28	28	LVCMOS18	10	28	28	LVCMOS18	16	28	28
LVCMOS15	8	28	28	LVCMOS15	6	28	28	LVCMOS15	12	28	28
LVCMOS12	4	28	28	LVCMOS12	4	28	28	LVCMOS12	8	28	28

Table 50 • MSIO SSO Guidelines for M2S050 - FG896 Device

I/O Standards	LVTTL (SSOs Causing)		LVCMOS25 (SSOs Causing)			LVCMOS18 (SSOs Causing)			LVCMOS15 (SSOs Causing)			LVCMOS12 (SSOs Causing)		
Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce
20	40	6	16	28	8	12	60	10	8	60	14	4	60	60
16	60	10	12	60	10	10	60	12	6	60	20	2	60	60
12	60	12	8	60	14	8	60	16	4	60	60			
8	60	20	6	60	20	6	60	22	2	60	60			
4	60	60	4	60	60	4	60	60						
2	60	60	2	60	60	2	60	60						

Table 51 • MSIOD SSO Guidelines for M2S050 - FG896 Device

	LVCMOS25 (SSOs Causing)		LVCMOS18 (SSOs Causing)			LVCMOS15 (SSOs Causing)			LVCMOS12 (SSOs Causing)		
Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	Drive strength (mA)	GND Bounce	V_{DDI} Bounce
12	60	14	10	60	28	6	60	60	4	60	60
8	60	20	8	60	60	4	60	60	2	60	60
6	60	60	6	60	60	2	60	60			
4	60	60	4	60	60						
2	60	60	2	60	60						

Table 52 • DDRIO SSO Guidelines for M2S050 - FG896 Device

	LVCMOS25 (SSOs Causing)		LVCMOS18 (SSOs Causing)			LVCMOS15 (SSOs Causing)			LVCMOS12 (SSOs Causing)		
Drive Strength (mA)	GND Bounce	V_{CCI} Bounce	Drive Strength (mA)	GND Bounce	V_{CCI} Bounce	Drive Strength (mA)	GND Bounce	V_{CCI} Bounce	Drive Strength (mA)	GND Bounce	V_{CCI} Bounce
16	8	16	16	20	52	12	60	54	8	60	60
12	18	28	12	60	60	10	60	60	4	60	60
8	60	58	10	60	60	8	60	60	2	60	60
6	60	60	8	60	60	6	60	60			
4	60	60	6	60	60	4	60	60			
2	60	60	4	60	60	2	60	60			
			2	60	60						

Table 53 • MSIO, MSIOD, and DDRIO SSO Guidelines for M2S060 - FG676 Device

MSIO (SSOs Causing)				MSIOD (SSOs Causing)				DDRIO (SSOs Causing)			
I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce
LVTTL	20	32	28								
LVCMOS25	16	32	32	LVCMOS25	12	32	16	LVCMOS25	16	14	32
LVCMOS18	12	32	32	LVCMOS18	10	32	30	LVCMOS18	16	28	32
LVCMOS15	8	32	32	LVCMOS15	6	32	32	LVCMOS15	12	32	32
LVCMOS12	4	32	32	LVCMOS12	4	32	32	LVCMOS12	8	32	32

Table 54 • MSIO, MSIOD, and DDRIO SSO Guidelines for M2S090 - FG676 Device

MSIO (SSOs Causing)				MSIOD (SSOs Causing)				DDRIO (SSOs Causing)			
I/O Standards	Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce	I/O Standards	I/O Drive Strength (mA)	GND Bounce	V_{DDI} Bounce
LVTTL	20	40	26								
LVCMOS25	16	40	40	LVCMOS25	12	40	22	LVCMOS25	16	40	40
LVCMOS18	12	40	40	LVCMOS18	10	40	40	LVCMOS18	16	40	40
LVCMOS15	8	40	40	LVCMOS15	6	40	40	LVCMOS15	12	40	40

Table 54 • **MSIO, MSIOD, and DDRIO SSO Guidelines for M2S090 - FG676 Device *(continued)***

MSIO (SSOs Causing)				MSIOD (SSOs Causing)				DDRIO (SSOs Causing)			
LVCMOS12	4	40	40	LVCMOS12	4	40	40	LVCMOS12	8	40	40

Table 55 • **MSIO, MSIOD, and DDRIO SSO Guidelines for M2S090 - FCS325 Device**

MSIO (SSOs Causing)				MSIOD (SSOs Causing)				DDRIO (SSOs Causing)			
I/O Standards	**Drive Strength (mA)**	**GND Bounce**	**V_{DDI} Bounce**	**I/O Standards**	**Drive Strength (mA)**	**GND Bounce**	**V_{DDI} Bounce**	**I/O Standards**	**Drive Strength (mA)**	**GND Bounce**	**V_{DDI} Bounce**
LVTTL	20	40	40								
LVCMOS25	16	40	40	LVCMOS25	12	30	30	LVCMOS25	16	40	40
LVCMOS18	12	40	40	LVCMOS18	10	30	30	LVCMOS18	16	40	40
LVCMOS15	8	40	40	LVCMOS15	6	30	30	LVCMOS15	12	40	40
LVCMOS12	4	40	40	LVCMOS12	4	30	30	LVCMOS12	8	40	40

Table 56 • **MSIO, MSIOD, and DDRIO SSO Guidelines for M2S150 - FC1152 Device**

MSIO (SSOs Causing)				MSIOD (SSOs Causing)				DDRIO (SSOs Causing)			
I/O Standards	**Drive Strength (mA)**	**GND Bounce**	**V_{DDI} Bounce**	**I/O Standards**	**Drive Strength (mA)**	**GND Bounce**	**V_{DDI} Bounce**	**I/O Standards**	**Drive Strength (mA)**	**GND Bounce**	**V_{DDI} Bounce**
LVTTL	20	60	60								
LVCMOS25	16	60	60	LVCMOS25	12	60	60	LVCMOS25	16	60	60
LVCMOS18	12	60	60	LVCMOS18	10	60	60	LVCMOS18	16	60	60
LVCMOS15	8	60	60	LVCMOS15	6	60	60	LVCMOS15	12	60	60
LVCMOS12	4	60	60	LVCMOS12	4	60	60	LVCMOS12	8	60	60

6.5 Supported I/O Standards

SmartFusion2/IGLOO2 devices support the different I/O standards, as listed in the following table. These I/O standards can be configured using Libero SoC. See *I/O Editor User Guide* for more details.

The following table lists all the I/O standards supported for single-ended and differential I/Os:

Table 57 • **Supported I/O Standards**

I/O Standards	Single-Ended	Differential	MSIO (Max 3.3V)	MSIOD (Max 2.5V)	DDRIO (Max 2.5V)
LVTTL	Yes		Yes		
PCI	Yes		Yes		
LVPECL (input only)		Yes	Yes		
LVDS33		Yes	Yes		
LVCMOS33	Yes		Yes		
LVCMOS25	Yes		Yes	Yes	Yes
LVCMOS18	Yes		Yes	Yes	Yes
LVCMOS15	Yes		Yes	Yes	Yes
LVCMOS12	Yes		Yes	Yes	Yes
SSTL2I	Yes	Yes	Yes	Yes	Yes (DDR1)
SSTL2II	Yes	Yes	Yes		Yes (DDR1)
SSTL18I	Yes	Yes			Yes (DDR2)
SSTL18II	Yes	Yes			Yes (DDR2)
SSTL15I (only for I/Os used by MDDR/FDDR)	Yes	Yes			Yes (DDR3)
SSTL15II (only for I/Os used by MDDR/FDDR)	Yes	Yes			Yes (DDR3)
HSTLI	Yes	Yes			Yes
HSTLII	Yes	Yes			Yes
LVDS		Yes	Yes	Yes	
RSDS		Yes	Yes	Yes	
Mini LVDS		Yes		Yes	
BUSLVDS		Yes	Yes	Yes (input only)	
MLVDS		Yes	Yes	Yes (input only)	

Note: Mini-LVDS is only supported for MSIOD voltage 2.5V. For I/O pin names and bank assignments for different device packages, see *DS0115: SmartFusion2 Pin Descriptions Datasheet* and *DS0124: IGLOO2 Pin Descriptions Datasheet* documents.

6.5.1 Single-Ended Standards

Single-ended I/O standards use a push-pull CMOS output stage with a voltage referenced to system ground. The input buffer configuration, output drive, and I/O supply voltage (VCCI) vary among the I/O standards. The advantage of these standards is that a common ground can be used for multiple I/Os. This simplifies board layout and reduces system cost. The reduced slew rate of these I/O standards causes less electromagnetic interference (EMI) on the board. However, these I/Os are not suitable for high frequency (>200 MHz) switching due to noise and higher power consumption.

6.5.1.1 Low Voltage TTL (LVTTL)

This is a general purpose standard (EIA/JESD8-B) for 3.3V applications. It uses an LVTTL input buffer and a push-pull output buffer. The LVTTL output buffer can have up to eight different programmable drive strengths.

6.5.1.2 Low Voltage CMOS (LVCMOS)

SmartFusion2 and IGLOO2 devices provide five different kinds of LVCMOS: LVCMOS 3.3V, LVCMOS 2.5V, LVCMOS 1.8V, LVCMOS 1.5V, and LVCOMS1.2V. LVCMOS 3.3V (only in MSIO) is an extension of the LVCMOS standard (JESD8-B compliant) used for general purpose 3.3V applications. LVCMOS 2.5V is an extension of the LVCMOS standard (JESD8-5-compliant) used for general purpose 2.5V applications.

LVCMOS 1.8V is an extension of the LVCMOS standard (JESD8-7-compliant) used for general purpose 1.8V applications. The LVCMOS 1.5V is an extension of the LVCMOS standard (JESD8-11-compliant) used for general purpose 1.5V applications.

The VCCI values for these standards are 3.3V, 2.5V, 1.8V, 1.5V, and 1.2V, respectively. All these versions use a 3.3V-tolerant CMOS input buffer and a push-pull output buffer, except MSIOD and DDRIO banks as they do not have 3.3V tolerant I/Os. Similar to LVTTL, the output buffer has up to eight different programmable drive strengths.

6.5.1.3 3.3V Peripheral Component Interface (PCI)

This standard specifies support for both 33 MHz and 66 MHz PCI bus applications. It uses an LVTTL input buffer and a push-pull output buffer. With the aid of an external resistor, this I/O standard can be 5V-compliant.

6.5.2 Voltage-Referenced Standards

I/Os using these standards are referenced to an external reference voltage (V_{REF}).

6.5.2.1 Input Reference Voltage

Each I/O bank supports reference voltage (V_{REF}). Any I/O in a bank can be configured as the input reference voltage pin to use with voltage reference input and bidirectional buffers. A V_{REF} pin is a regular MSIO/MSIOD that is configured as a reference voltage input in the design. To support SSTL and HSTL inputs, the reference voltage is typically powered with a voltage of one-half that of the bank's VDDI level.

In general, mixing of a single-ended voltage referenced I/O with a non-referenced I/O is permitted in MSIO and MSIOD banks. The mixing of signals allows the combinations of LVCMOS, HSTL, and SSTL I/O types considering they share the same VDDI level. However, any I/O type mixing within a bank must follow placement I/O pair restrictions between the positive differential IO pin (IOP) and negative differential pin (ION) within an IOA block.

The following table lists the valid and invalid pairs that can be created in IOA block pins.

Table 58 • **IOA Pair Design Rules[1]**

IOA Block Pins		
IOP	**ION**	**Valid/Invalid**
HSTL/SSTL	Unused	Valid
HSTL/SSTL	HSTL/SSTL	Valid
HSTL/SSTL	LVCMOS/LVTTL/PCI	Invalid

1. Applicable only for MSIO/MSIOD I/O types.

Note: The rules apply to all HSTL/SSTL Class I or Class II input, output, or bidirectional I/O types.

Note: According to JEDEC standards, HSTL/SSTL outputs and bidirectional pins must be terminated to VTT, and inputs referenced to VTT.

Note: Lack of SSTL/HSTL termination or use of a non-HSTL/SSTL combination results in excessive V_{REF} leakage. This leakage can reduce the V_{REF} voltage level on the board and affect reliability of the device.

Note: Input V_{REF} leakage is specified in the device datasheet.

The following table provides the assignment of ION signal, when IOP signal is assigned for V_{REF} pin of MSIO/MSIOD banks.

Table 59 • **Status of the V_{REF} Pin Assigned Rule for IOA**

IOP	ION	Status
V_{REF}	Output	Invalid
	Tristate	Invalid
	bidirectional	Invalid
	Input	Valid

6.5.2.2 High-Speed Transceiver Logic (HSTL) Class I

These are general purpose, high-speed 1.5V bus standards (EIA/JESD8-6) for signaling between integrated circuits. The signaling range is 0V to 1.5V, and signals can be either single-ended or differential. HSTL requires a differential amplifier input buffer and a push-pull output buffer. These standards are used in the memory bus interface with data switching capability of up to 400 MHz. The other advantages of these standards are low power and fewer EMI concerns. HSTL has four classes, of which SmartFusion2 and IGLOO2 devices support Class I. The reference voltage (V_{REF}) is 0.75V.

6.5.2.3 Stub Series Terminated Logic 2.5V (SSTL2) Class I and II

These are general purpose 2.5V memory bus standards (JESD8-9) for driving transmission lines, designed specifically for driving the DDR SDRAM modules used in computer memory. The SSTL2 requires a differential amplifier input buffer and a push-pull output buffer. The reference voltage (V_{REF}) is 1.25V.

6.5.2.4 Stub Series Terminated Logic 1.8V (SSTL18) Class I and II

These are general purpose 1.8V memory bus standards (JESD8-15) for driving transmission lines, designed specifically for driving the DDR2 SDRAM modules used in computer memory. SSTL18 requires a differential amplifier input buffer and a push-pull output buffer. The V_{REF} is 0.9V.

6.5.3 Differential Standards

These standards require two I/Os per signal (called a signal pair). Logic values are determined by the potential difference between the lines, not with respect to ground. This is why differential drivers and receivers have much better noise immunity than single-ended standards. The differential interface

standards offer higher performance and lower power consumption than their single-ended counterparts. Two I/O pins are used for each data transfer channel. Differential standards require resistor termination on both I/Os.

6.5.3.1 Low Voltage Positive Emitter Coupled Logic

Low voltage positive emitter coupled logic (LVPECL) requires that one data bit is carried through two signal lines; therefore, two pins are needed per input or output. It also requires external resistor termination. The voltage swing between the two signal lines is approximately 850 mV. When the power supply is +3.3V, it is commonly referred to as LVPECL.

6.5.3.2 Low Voltage Differential Signal

Low voltage differential signal (LVDS) is a differential I/O standard. As with all differential signaling standards, LVDS requires that one data bit is carried through two signal lines, and it has inherent noise immunity over single-ended I/O standards. The voltage swing between two signal lines is approximately 350 mV. The external V_{REF} or board termination voltage (V_{TT}) is not required. LVDS requires the use of two pins per input or output.

6.5.3.3 Reduced Swing Differential Signaling

Reduced swing differential signaling (RSDS) is a signaling standard that defines the output characteristics of a transmitter and inputs of a receiver along with the protocol for a chip-to-chip interface between flat-panel timing controllers and column drivers.

6.5.3.4 B-LVDS/M-LVDS

Bus LVDS (B-LVDS) refers to bus interface circuits based on LVDS technology. Multi-point LVDS (M-LVDS) specifications extend the LVDS standard to high-performance multi-point bus applications. Multi-drop and multi-point bus configurations may contain any combination of drivers, receivers, and transceivers. The LVDS drivers provide the higher drive current required by B-LVDS and M-LVDS to accommodate the bus loading.

The driver requires series terminations for better signal quality and to control voltage swing. Termination is also required at both ends of the bus, since the driver can be located anywhere on the bus. The SmartFusion2 and IGLOO2 MSIOD has an internal circuit isolation, and the bus isolation should be taken care of in the design external to the device when using M-LVDS.

6.5.3.5 Mini-LVDS

A serial, intra-flat panel solution that serves as an interface between the timing control function and an LCD source driver.

6.6 I/O Programmable Features

SmartFusion2 and IGLOO2 devices support different I/O programmable features for MSIO, MSIOD, and DDRIO user I/Os. Users cannot modify some of these features, if the software rules are locked in the I/O attribute editor.

The following table lists the supported I/O programmable features that can be configured through the I/O attribute editor or pdc file.

Table 60 • **SmartFusion2 and IGLOO2 I/O Features**

I/O Features	MSIO	MSIOD	DDRIO
Programmable slew rate control			Yes
Programmable input delay	Yes	Yes	Yes
Programmable weak pull-up/down	Yes	Yes	Yes
Programmable Schmitt trigger receiver	Yes	Yes	Yes
Pre-emphasis		Yes	
Bus keeping	Yes	Yes	Yes

Table 60 • **SmartFusion2 and IGLOO2 I/O Features**

I/O Features	MSIO	MSIOD	DDRIO
Receiver ODT configuration	Yes	Yes	Yes
Driver impedance configuration	Yes	Yes	Yes
Hot insertion	Yes		
IO state control in low power mode	Yes	Yes	Yes

6.6.1 Programmable Slew-Rate Control

The output buffer has a programmable slew-rate control for high-speed and low-noise performance. A faster slew-rate provides the high-speed transition and slow slew-rate reduces system noise with nominal delay in raising and falling transitions.

There are four slew-rate controls configured through the I/O attribute editor or the pdc file for a particular I/O standard of DDRIO.

Note: MSIOs and MSIODs do not support programmable slew-rate control.

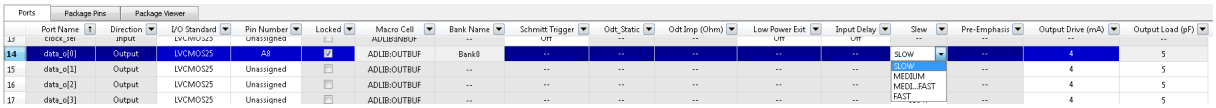
The following table lists the programmable slew-rate control options that can be set through the I/O attribute editor.

Table 61 • **Programmable Slew Rate Control**

User I/O	I/O Standard	Slew-Rate	Options
DDRIO	LVCMOS12	0	Slow
	LVCMOS15	1	Medium
	LVCMOS18	2	Medium-Fast
	LVCMOS25	3	Fast

The following figure shows an example slew-rate using the I/O attribute editor.

Figure 51 • **Programmable Slew-Rate**



Following is the example script to set slew-rate using io.pdc:

```
set_io signal name     \
    -pinname A8         \
    -fixed yes          \
    -SLEW MEDIUM_FAST \
    -DIRECTION OUTPUT
```

Signal name is the user I\O name where the designer is needed to set slew-rate.

6.6.2 Programmable Input Delay

Each I/O, when configured as an input, can be programmed with different input delays.

The input delay is calculated using:

Delay = D + N x 0.1 ns (N ranges from 0 to 63)

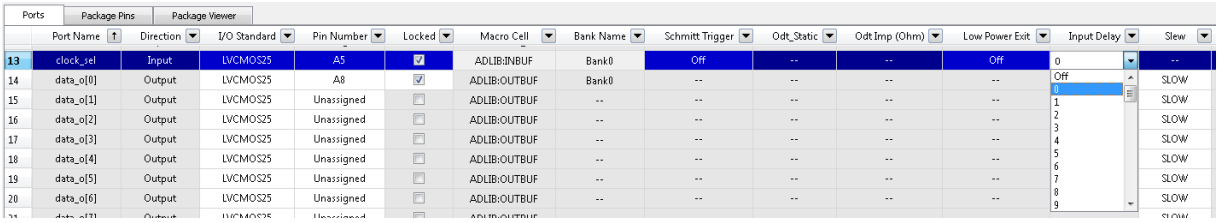
D is the intrinsic delay or circuit delay of an input without additional delay, when N is 0. The total delay range is between D ns to D + 6.3 ns.

There are 64 input delay values, which can be chosen and configured using the I/O attribute editor or the pdc file of Libero SoC for MSIO, MSIOD, and DDRIO.

Table 63, page 95 lists the programmable input delay options available for different I/O standards, and can be set from 0 to 63 through the I/O attribute editor or the pdc file.

The following figure shows an example to set input delay using the I/O attribute editor.

Figure 52 • **Programmable Input Delay**



Following is the example script to set input delay using io.pdc:

```
set_io signal name     \
    -pinname A5        \
    -fixed yes         \
    -IN_DELAY 0        \
    -DIRECTION INPUT
```

Signal name is the user I/O name that the designer can set for input delay.

Note: Input delays can be used for hold time improvement of the input register by increasing input pin to input register delay.

6.6.3 Programmable Weak Pull-Up and Pull-Down

All user I/Os can be programmed to optional weak pull-up and pull-down, which are mutually exclusive and weakly hold the output to either VDDI or GND, respectively.

The following table shows the three settings for weak pull-up and pull-down provided by Libero SoC and can be set through the I/O attribute editor or the pdc file.

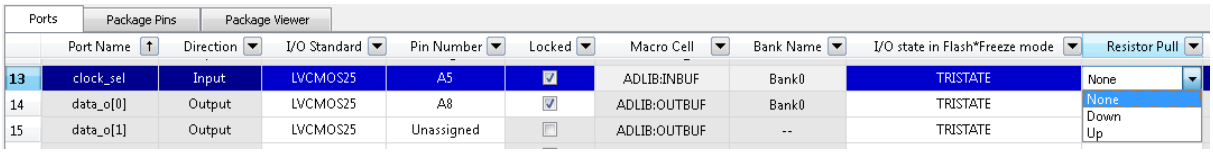
Table 62 • **Programmable Weak Pull-up and Pull-down**

Weak Pull-Up/Pull-Down	Options
None	Disable pull-up or pull-down
Up	Enable pull-up
Down	Enable pull-down

Table 63, page 95 lists the weak pull-up and pull-down options available for different I/O standards, and can be set as Up, Down, or None through the I/O attribute editor or the pdc file.

The following figure shows an example to set weak pull-up and pull-down using the I/O attribute editor.

Figure 53 • **Programmable Weak Pull-Up and Pull-Down**



Following is the example script to set weak pull-up and pull-down using io.pdc:

```
set_io signal name       \
    -pinname A5          \
    -fixed yes           \
    -RES_PULL Up         \
    -DIRECTION INPUT
```

Signal name is the user I/O name where the designer is required to set weak pull-up and pull-down options.

6.6.4 Programmable Schmitt Trigger Receiver

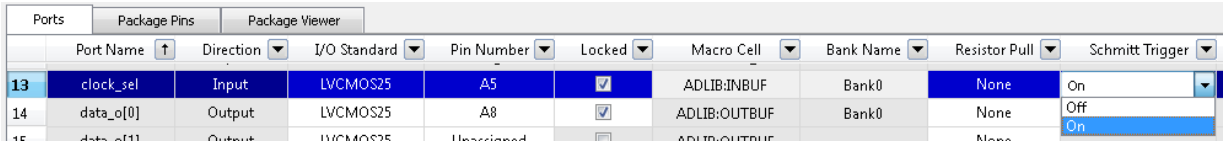
The input buffer of an I/O can be configured as a schmitt trigger or single-ended receiver with the support of different I/O standards. To improve noise immunity for signals with slow edge rate, a schmitt trigger feature introduces hysteresis to the input signal.

See Table 63, page 95 for the I/O standards, which support the schmitt trigger option.

The schmitt trigger feature can be enabled or disabled by using the I/O attribute editor or the pdc file in Libero SoC, but it is disabled by default.

The following figure shows an example to enable schmitt trigger using the I/O attribute editor.

Figure 54 • **Programmable Schmitt Trigger Receiver**



Following is the example script to enable schmitt trigger using io.pdc:

```
set_io signal name     \
   -pinname A5       \
   -fixed yes    \
   -SCHMITT_TRIGGER On    \
   -DIRECTION INPUT
```

Signal name is the user I/O name that the designer decides on enabling the schmitt trigger feature.

6.6.5 Programmable Pre-emphasis

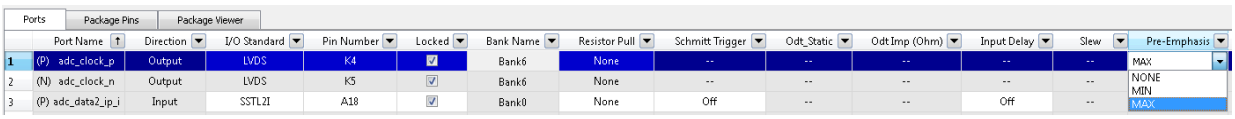
The differential swing and output impedance of the driver set the output current limit of a high-speed signal. For high frequency differential signals, slew-rate might not be sufficient to reach the peak before the next edge comes, this produces jitter. With pre-emphasis enabled, the output current is boosted momentarily during transition to increase the slew-rate. MSIOD buffers only support pre-emphasis feature.

Pre-emphasis option is NONE by default. The pre-emphasis value can be changed to MIN (dB) or MAX (dB) through the I/O attribute editor or the pdc file.

See Table 63, page 95 for the I/O standards, which support programmable pre-emphasis option.

The following figure shows an example to set pre-emphasis using the I/O attribute editor.

Figure 55 • **Programmable Pre-emphasis**



Following is the example script to set pre-emphasis using io.pdc

```
set_io signal name      \
    -pinname K5         \
    -fixed yes          \
    -iostd LVDS         \
    -PRE_EMPHASIS MIN   \
    -DIRECTION OUTPUT
```

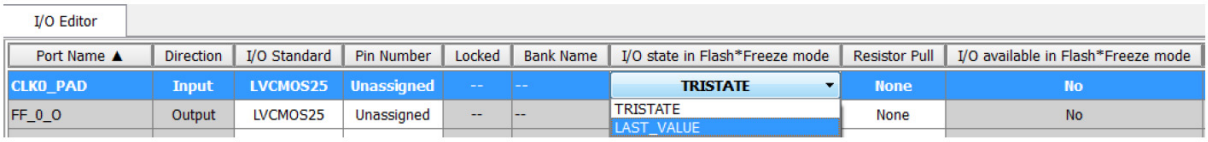
Signal name is the user I/O name that the designer needs to set for pre-emphasis.

6.6.6 Bus Keeper

The main function is to weakly hold the signal on an I/O pin at its last driven state, holding it at a valid level with minimal power dissipation. The bus keeper circuitry also pulls undriven pins away from the input threshold voltage where noise can cause unintended oscillation. Bus Keeper is only available in Flash*Freeze mode (not during normal operation). This feature is activated by setting LAST_VALUE option for the selected I/O pad under I/O state in Flash*Freeze mode column in the I/O attribute editor.

The following figure shows the configuration in I/O Editor.

Figure 56 • **Bus Keeper Configuration in I/O Editor**



When regular User I/Os (MSIO, MSIOD, DDRIO) are not used, Libero configures the I/O as input buffer disabled, output buffer tristated with weak pull-up. Unused dedicated global I/Os behave similar to unused regular user I/Os.

The following table lists the supported I/O programmable features and their support for different standards.

Table 63 • **I/O Programmable Features and Standards**

I/O Standards	Input Delay (Off/0-63)			Hot-Swap	Pre-Emphasis	Programmable Slew-Rate	Schmitt Trigger Input			Resistor Pull		
	MSIO	MSIOD	DDRIO	MSIO	MSIOD	DDRIO	MSIO	MSIOD	DDRIO	MSIO	MSIOD	DDRIO
LVTTL	Yes			Yes			Yes			Yes		
PCI	Yes						Yes			Yes		
LVPECL (Input only)	Yes			Yes						Yes		
LVDS33	Yes			Yes						Yes		
LVCMOS12	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes
LVCMOS15	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes
LVCMOS18	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes
LVCMOS25	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes	Yes	Yes
LVCMOS33	Yes			Yes			Yes			Yes		
SSTL2I	Yes	Yes	Yes	Yes						Yes	Yes	Yes
SSTL2II	Yes		Yes	Yes						Yes		Yes
SSTL18I			Yes									Yes

Table 63 • **I/O Programmable Features and Standards *(continued)***

I/O Standards	Input Delay (Off/0-63)			Hot-Swap	Pre-Emphasis	Programmable Slew-Rate	Schmitt Trigger Input			Resistor Pull		
	MSIO	MSIOD	DDRIO	MSIO	MSIOD	DDRIO	MSIO	MSIOD	DDRIO	MSIO	MSIOD	DDRIO
SSTL18II			Yes									Yes
SSTL15I (only for I/Os used by MDDR/FDDR)			Yes									Yes
SSTL15II (only for I/Os used by MDDR/FDDR)			Yes									Yes
HSTLI			Yes									Yes
HSTLII			Yes									Yes
LVDS	Yes	Yes		Yes	Yes					Yes	Yes	
RSDS	Yes	Yes		Yes	Yes					Yes	Yes	
Mini LVDS	Yes	Yes		Yes	Yes					Yes	Yes	
BUSLVDS	Yes	Yes		Yes						Yes	Yes	
MLVDS	Yes	Yes		Yes						Yes	Yes	

6.7 Receiver ODT Configuration

SmartFusion2 and IGLOO2 user I/Os support ODT features available on I/O, which is configured as input or bidirectional buffers. The ODT termination provides a good signal integrity, saves board space, and reduces external components on PCB.

Note: ODT value depends on the I/O standard. Some I/O standards support more than one ODT value as listed in Table 64, page 98. When there is more than one ODT value, the I/O Attribute Editor only displays the default ODT value. In this case, set the required ODT value in the I/O Attribute Editor or PDC.

6.7.1 Receiver ODT Configuration for MSIO and MSIOD Banks

Libero SoC has the following settings for receiver ODT configuration.

- ODT Static
- ODT Impedance

6.7.1.1 ODT Static

There are two types of ODT static: ON and OFF.

- **ON**: The termination resistor for impedance matching is enabled in the chip. The value set to ODT impedance is configured as parallel termination for the input or bidirectional buffers.
- **OFF**: The termination resistors for impedance matching are located on the printed circuit board.

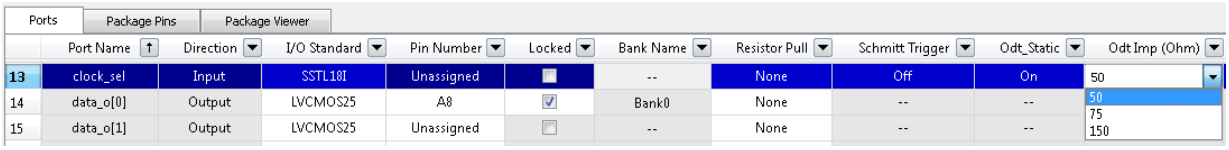
6.7.1.2 ODT Impedance

When ODT static is ON, the valid ODT impedance values for the input or bidirectional buffers are chosen from the I/O attribute editor. See Table 64, page 98 for ODT impedance values of different I/O standards.

The ODT static and ODT impedance values can be set through the I/O attribute editor or the pdc file for all the different I/O standards.

The following figure shows an example to set ODT static and ODT impedance values using the I/O attribute editor.

Figure 57 • **Receiver ODT Configuration**



Following is the sample constraint to set ODT static and ODT impedance values using io.pdc:

```
set_io signal name       \
   -iostd SSTL18I     \
   -ODT_IMP 75        \
   -ODT_STATIC On     \
   -DIRECTION INPUT
```

Signal name is the user I/O name that the designer has to set ODT static and impedance values.

The following table lists ODT impedance values for MSIO and MSIOD.

Table 64 • **ODT Impedance Values**

	ODT Impedance	
I/O Standards	**ODT Static Enabled**	**ODT Static Disabled**
Single-ended		
LVCMOS18	50, 75, 150	External on-board terminations are required as per simulation results
LVCMOS15		
LVCMOS12		
Differential		
LVPECL (differential input only)	100	
LVDS		
RSDS		
Mini LVDS		
BUSLVDS (inputs only)		
MLVDS (inputs only)		

Note: Due to electromagnetic concerned, ODT is not allowed for 2.5V or higher single ended signals. It is allowed for differential signals.

6.7.2 Receiver ODT Configuration for DDRIO Banks

SmartFusion2 and IGLOO2 DDRIOs have an in-built I/O calibration engine for impedance calibration. The I/O calibration engine can be enabled or disabled by using System Builder during MDDR or FDDR configuration. The I/O calibration engine is enabled to achieve the impedance control by calibrating the I/O drivers to an external on-board resistor connected between the DDR_IMP_CALIB and VSS pins. If the I/O calibration engine is disabled, ODT impedance is not supported.

In Libero SoC (System Builder Configurator), the DDR Configurator has an option to set the calibration engine ON or OFF for the LPDDR memory only. For DDR2 and DDR3 memories, by default the calibration engine is set to ON internally, and user does not have access to disable it. It is recommended to have the ODT option enabled for higher data rates.

Libero SoC has the following settings for receiver ODT configuration:

- ODT Static
- ODT Impedance
- I/O Calibration Engine

The ODT static and ODT impedance values can be set through the I/O attribute editor or PDC file. The I/O calibration engine can be set through System Builder during MDDR/FDDR Configurator.

6.7.2.1 ODT Static

- **ON**: The termination resistor for impedance matching is enabled in the chip. The value set to ODT impedance is configured as parallel termination for input or bidirectional buffers.
- **OFF**: The termination resistors for impedance matching are disabled. Changing the ODT impedance value has no effect on the impedance calibration and termination resistors are located on the printed circuit board.

6.7.2.2 ODT Impedance

During ODT static ON, the valid ODT impedance values for input or bidirectional buffer are chosen from the I/O attribute editor. Table 66, page 100 and Table 67, page 100 for ODT impedance values of different I/O standards.

If an I/O is connected to a memory controller, ODT static has no effect and it is overridden by the memory controller. If ODT is not desired, it can be disabled from the memory controller and the option is available in Libero System Builder.

6.7.2.3 I/O Calibration Engine

The calibration engine is part of the MDDR/FDDR memory controller IP. The engine calibrates ODT and driver impedance to an external calibration resistor. Calibration occurs during system power up and optionally during DDR refresh. Table 65, page 99 lists the ODT configuration options for MSIO, MSIOD, and DDRIOs.

DDRIO bank may have fabric I/Os muxed with DDR controlled I/Os and can be calibrated for I/O impedance calibration. Calibrate I/Os only during power-up and re-calibrate if fabric I/Os are fully tristated. If designers re-calibrate I/Os during DDR PHY self-refresh mode, the fabric controller I/Os can possibly glitch. There are two possible solutions to avoid glitch on user I/Os.

1. If re-calibration is required for DDR controlled I/Os, designer must not use user I/Os from the same bank for any application.
2. DDR PHY self-refresh mode must not be enabled if the user wants to reuse some of the DDRIOs in a bank for general purpose.

Apart from the ODT static and ODT impedance settings, the DDRIOs have an in-built I/O calibration engine to configure ODT.

The following table lists the ODT configuration options for MSIO, MSIOD, and DDRIOs.

Table 65 • **ODT Configuration Options for MSIO, MSIOD, and DDRIOs**

User I/Os	MDDR/FDDR Controller	I/O Calibration Engine	ODT Configuration
DDRIO	Enable	Enable	Configure ODT with calibration engine and calibrated to external resistor
		Disable	ODT not available
DDRIO	Disable	Disable	ODT not available
MSIO MSIOD			Configure ODT using fixed value through I/O attribute editor.

ODT can be enabled for any I/O in the DDRIO bank, even if the I/O is not associated with DDR controller. To enable ODT, a designer needs the DDR controller in the design with the I/O calibration enabled. All I/Os in the DDRIO bank (including the I/O that is not used by the controller) can be calibrated. When DDR controller is used in a bank, it takes over ODT_STATIC of its own I/Os only. ODT for other I/Os in the same bank can still be controlled using ODT_STATIC option for that particular I/O.

The following table lists the settings available for the designer to configure DDRIO ODT impedance using I/O calibration engine, I/O attribute editor and external on-board resistor.

Table 66 • **DDRIO ODT Configuration- for I/O Connected to Fabric**

		Set Through System Builder[1]	Set Through I/O Attribute Editor		Resistor to be Mounted on PCB	
Memory	**I/O Standards**	**I/O Calibration Engine**	**ODT Static**	**ODT Impedance**	**On-board Resistor for I/O Calibration[2]**	**On-board External Terminations[3]**
LPDDR	LVCMOS 18[4]		ON- Not supported			
		Off	Off			Optional
		On	On	50, 75, 150	150 Ω ± 1%	Optional
DDR2	SSTL18	ON (by default) User does not have access to disable.	On	50, 75, 150	150 Ω ± 1%	Optional
			Off			Required
DDR3	SSTL15		On	20, 30, 40, 60, 120	240 Ω ± 1%	Optional
			Off			Required
DDRIO (non-memory)	LVCMOS12 LVCMOS15	Off	Off			Required
	LVCMOS18	Off	Off			Required
	HSTLI HSTLII	Off	Off			Required
	LVCMOS12 LVCMOS15	On	On	50, 75, 150		Optional
	LVCMOS18	On	On	50, 75, 150		Optional
	HSTLI HSTLII	On	On	47.8		Optional

1. I/O calibration engine and drive strength can be selected through System Builder during external memory MDDR/FDDR controller configuration. The I/O calibration engine is available only for DDRIOs.
2. Resistor should be mounted between DDR_IMP_CALIB and VSS.
3. Values and location of on-board external terminations are based on the signal integrity analysis.
4. LVCMOS18 is a non-terminated standard and usually does not require on-board external terminations.

Table 67 • **DDRIO ODT Configuration- for I/O Connected to DDR Controller**

		Set Through System Builder	Set Through I/O Attribute Editor		Resistor to be Mounted on PCB	
Memory	**I/O Standards**	**I/O Calibration Engine**	**Local ODT**	**ODT Impedance**	**On-board Resistor for I/O Calibration**	**On-board External Terminations**
LPDDR	LVCMOS 18	Off	On- Not Supported			
		Off	Off			Optional
		On	On	50, 75, 150	150 Ω ± 1%	Optional

Table 67 • **DDRIO ODT Configuration- for I/O Connected to DDR Controller *(continued)***

Memory	I/O Standards	Set Through System Builder I/O Calibration Engine	Set Through I/O Attribute Editor Local ODT	 ODT Impedance	Resistor to be Mounted on PCB On-board Resistor for I/O Calibration	 On-board External Terminations
DDR2	SSTL18	On (by default) User does not have access to disable.	On	50, 75, 150	150 Ω ± 1%	Optional
			Off			Required
DDR3	SSTL15		On	20, 30, 40, 60, 120	240 Ω ± 1%	Optional
			Off			Required

6.8 Driver Impedance Configuration

SmartFusion2/IGLOO2 I/Os support driver impedance configuration only for the output or bidirectional buffers.The driver impedance internal series termination provides a good signal integrity, saves board space, and reduces external components on the PCB.

The Libero SoC tool has output drive settings for driver impedance configuration.

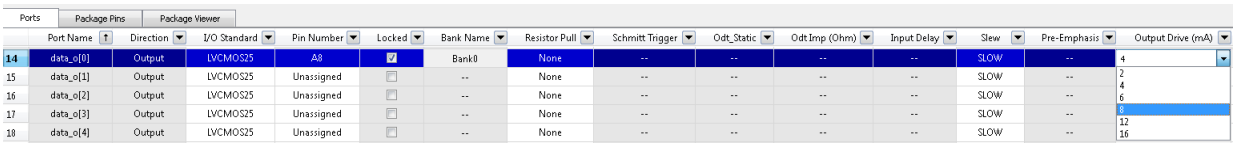
The following table shows the options of driver impedance configurations.

Table 68 • **Driver Impedance Configurations**

User I/Os	MDDR/FDDR Controller	I/O Calibration Block	Driver impedance
DDRIO	Enable	Enable	Configure driver impedance with I/O calibration engine and calibrated to external on-board reference resistor. The target impedance must be set through the I/O attribute editor.
		Disable	Driver impedance disabled.
DDRIO	Disable	Disable	Driver impedance disabled.
MSIO/MSIOD			Configure driver impedance using fixed values depending on the drive strength set through the I/O attribute editor.

The following figure shows an example to set output drive using the I/O attribute editor.

Figure 58 • **Output Drive Impedance**



Following is the example script to set output drive using io.pdc

```
set_io signal name      \
   -pinname A8          \
   -fixed yes           \
   -OUT_DRIVE 8         \
   -DIRECTION OUTPUT
```

Signal name is the user I/O name for which the designer wants to set driver impedance.

6.8.1 Driver Impedance Configuration for MSIO/MSIODs

SmartFusion2/IGLOO2 device output or bidirectional buffers have a programmable drive-strength control for certain I/O standards to mitigate the effects of high signal attenuation due to the long transmission line.

The following table lists the programmable drive strengths and these can be set through the I/O attribute editor:

Table 69 • **Driver Impedance Configurations for MSIO/MSIODs**

	Output Drive Strength (mA)	
I/O Standards	**MSIO**	**MSIOD**
LVTTL	2, 4, 8, 12, 16, 20	
LVCMOS12	2, 4	2, 4
LVCMOS15	2, 4, 6, 8	2, 4, 6
LVCMOS18	2, 4, 6, 8, 10, 12	2, 4, 6, 8, 10
LVCMOS25	2, 4, 6, 8, 12, 16	2, 4, 6, 8, 12
LVCMOS33	2, 4, 8, 12, 16, 20	

For other supporting I/O standards, output drive strength is fixed and different for each bank type and the I/O standard combination. For example, PCI standard output drive strength is 20 mA and HSTL is 8 mA.

6.8.2 Driver Impedance Configuration for DDRIOs

The calibration engine is enabled to achieve the impedance control by calibrating the I/O drivers to an external I/O calibration on-board resistor. If I/O calibration engine is disabled, driver impedance is disabled.

In Libero SoC, option is available in the DDR System Builder Configurator to set the calibration engine ON or OFF for the LPDDR memory alone. For DDR2/DDR3 memories, the calibration engine is set to ON internally by default.

The output drive strength for DDR2/DDR3 memory interfaces can be selected through System Builder during an external memory MDDR/FDDR controller configuration.

The following table lists the driver impedance configuration for DDRIOs with the DDR controller enabled.

Table 70 • **Driver Impedance Configurations for DDRIOs**

MDDR/FDDR Controller	Memory Type	I/O Standards	Output Drive Strength (mA)
Enable	LPDDR	LVCMOS18	2, 4, 6, 8, 10, 12, and 16
	DDR2	SSTL18	Half/Full
	DDR3	SSTL15	Half/Full

The driver impedance depends on the value of drive strength (mA) set through the I/O attribute editor. The maximum performance is achieved by setting the highest output drive strength of the device.

IBIS simulation can show the effects of different drive strengths, termination resistors, and capacitive loads on the system.

The DDRIO bank has more I/Os than required for the memory controller. These I/Os can be used for general FPGA I/Os with some caveats. The memory controller calibration engine affects all DDRIO bank I/Os. A re-calibration event results in glitches on these DDRIOs when configured as outputs and also configured as inputs with ODT enabled. DRRIO configured as inputs with ODT OFF does not experience calibration related glitches.

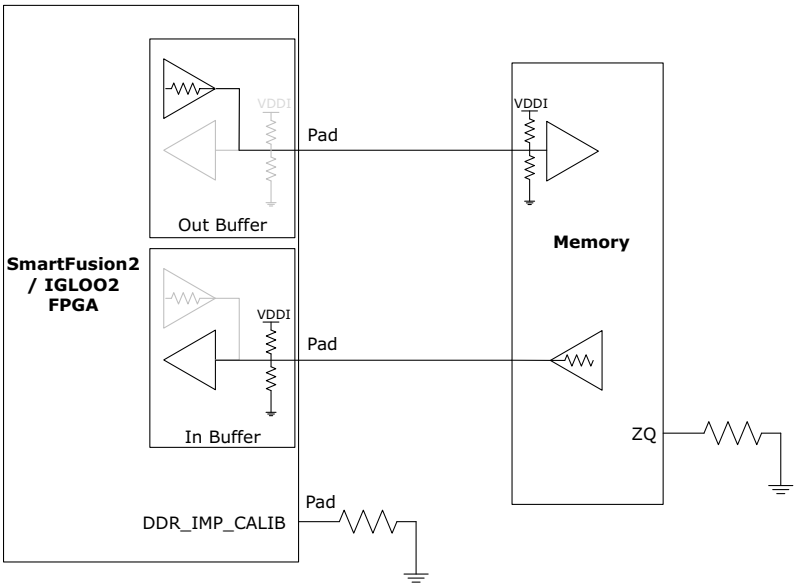
The following table lists the driver impedance configuration for DDRIOs without the DDR controller enabled.

Table 71 • **Driver Impedance Configurations for DDRIOs without DDR Controller**

DDRIOs	Drive Strength Setting Through I/O Attribute Editor (mA)	Corresponding Driver Impedance (Ω)
LVCMOS25	2, 4, 6, 8, 12,16	75, 60, 50, 33, 25, 20
LVCMOS18	2, 4, 6, 8, 10, 12,16	75, 60, 50, 33, 25, 20
LVCMOS15	2, 4, 6, 8, 10, 12	75, 60, 50, 40
LVCMOS12	2, 4, 6	75, 60, 50, 40
SSTL2I	8.1	42
SSTL2II	16.2	20
SSTL18I	6.5	42
SSTL18II	13.4	20
SSTL15I		40
SSTL15II		34
HSTLI	8	47.8
HSTLII	16	25.5

6.9 I/O Buffer Structure

Figure 59 • **Driver Impedance Configurations for MSIO/MSIODs**



6.10 Internal Clamp Diode

System controller keeps the user I/Os in tristate mode during power-up. The user I/Os have internal clamp diodes to protect the device I/Os.

All MSIOs are cold separable as the internal clamp diodes are always disabled, except if MSIOs are configured in the PCI I/O standards, which are not cold separable. For more information, see the *AC396: SmartFusion2 and IGLOO2 in Hot Swapping and Cold Sparing Application Note*.

6.11 Low-Power Signature Mode and Activity Mode

SmartFusion2 and IGLOO2 devices support Flash*Freeze mode, where several device resources are put into a low-power state using various power management hooks available for each resource.

The following two methods can be used for SmartFusion2 and IGLOO2 devices wake-up from Flash*Freeze mode:

- Real time counter (RTC) timeout
- I/O cell wake-up

In the RTC timeout method, the timeout value is set in the RTC before entering Flash*Freeze mode. In the I/O cell wake-up method, any activity on a specified input or matching a user-defined pattern value (signature) on a number of inputs wakes up the device.

There are two modes for exiting low-power mode (Flash*Freeze mode): signature mode and activity mode. Each I/O can be configured in Libero SoC to be in either of these modes. Each DDRIO has four options for configuring and controlling low-power exit:

- I/O is not designated for low-power exit monitoring
- I/O is designated for low-power activity monitoring
- I/O is designated for low-power signature, look for 0
- I/O is designated for low-power signature, look for 1

6.11.1 Signature Mode

After entering low-power mode, every I/O designated as a signature I/O becomes input-only. All other I/Os are tristated, held by bus hold, or weakly pulled-up or pulled-down. The signature I/O is

pre-configured to check the signal (0 or 1) on each pin in the Libero SoC I/O Editor. A group of pins are configured in the I/O Editor to form a signature pattern. In low-power mode when the correct pattern exits on these pins, the device exits low-power mode.

6.11.2 Activity Mode

In activity mode, the value at I/O pin is latched before the device goes to low-power mode. The I/O is configured for wake-on change input logic 0 or 1 in the Libero SoC I/O Editor. When an I/O activity is detected on the configured pin, the device exits low-power mode.

6.12 3.3V Input Tolerance in 2.5V MSIOD/DDRIO Banks

Uses two external resistor termination (Rext)—Rext 1: 33-200 W and Rext 2: 180-560 W. Resistor values are taken as available standard value resistor. The ± 5% tolerance is considered for Rext1 and Rext2. Supply variation is ± 5%. Rise time/fall time is taken as 10-90% value.

The following illustration describes the simulation setup.

Figure 60 • **Simulation Setup**

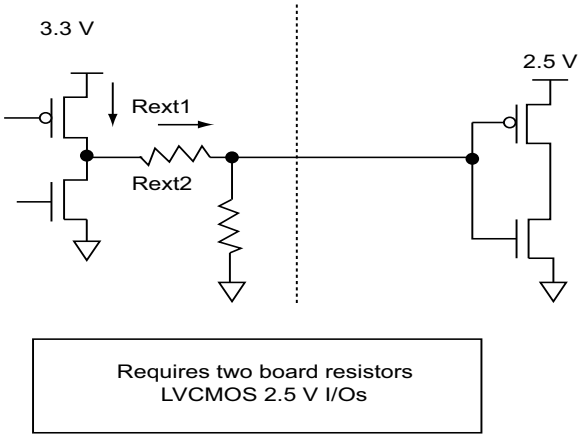


Table 72 • **F_{MAX}, I_{RMS}, and Max DC Voltage and Current of MSIOD**

Rext 1 (W)	Rext 2 (W)	I_{RMS} (mA)	Rise Time (ns)	Fall Time (ns)	F_{MAX} (MHz) = 1/4 × (Rise Time + Fall Time)	F_{MAX} (MHz) = 1/3 × (Rise Time + Fall Time)
33	180	2.72	2.32	2.3	54	72
51	220	2.54	2.61	2.62	47	63
68	270	2.41	2.97	2.96	42	56
100	330	2.18	3.54	3.51	35	47
150	430	1.9	4.54	4.4	27	37
200	560	1.7	5.56	5.41	22	30

Table 73 • F_{MAX}, I_{RMS}, and Max DC Voltage and Current of DDRIO

Rext 1 (W)	Rext 2 (W)	I_{RMS} (mA)	Rise Time (ns)	Fall Time (ns)	F_{MAX} (MHz) = 1/4 × (Rise Time + Fall Time)	F_{MAX} (MHz) = 1/3 × (Rise Time + Fall Time)
33	180	2.57	2.24	2.13	57	76
51	220	2.4	2.48	2.39	51	68
68	270	2.28	2.8	2.72	45	60
100	330	2.06	3.4	3.24	37	50
150	430	1.81	4.18	4.1	30	40
200	560	1.62	4.84	4.79	25	34

6.13 5V Input Tolerance and Output Driving Compatibility (only MSIO)

6.13.1 5V Input Tolerance

I/Os can support 5V input tolerance when LVTTL 3.3V, LVCMOS 3.3V, or LVCMOS 2.5V configurations are used. There are three recommended solutions for achieving 5V receiver tolerance. All the solutions meet the requirement of limiting the voltage at the input to 3.45V or less. In fact, the absolute maximum I/O voltage rating is 3.45V, and any voltage above this can cause long-term gate oxide failure.

6.13.1.1 Solution 1

The board design must ensure that the reflected waveform at the pad does not exceed the limits provided in the recommended operating conditions in the datasheet. Adhering to the recommended operating conditions is a requirement to put in place to ensure long-term reliability of input tolerance.

This solution also works for a 3.3V PCI configuration, but the internal diode must not be used for clamping, and the voltage must be limited by two external resistors. Relying on diode clamping creates an excessive pad DC voltage of 3.3V + 0.7V = 4V.

This solution requires two on-board resistors. Here are some examples of possible resistor values based on a simplified simulation model with no line effects and with a 10 Ω transmitter output impedance, where

Rtx_out_high = [VCCI – VOH] / IOH and Rtx_out_low = VOL / IOL).

Example 1 (high speed, high current):

Rtx_out_high = Rtx_out_low = 10 Ω

R1 = 36 Ω (±5%), P(r1)min = 0.069 Ω

R2 = 82 Ω (±5%), P(r2)min = 0.158 Ω

Imax_tx = 5.5V / (82 × 0.95 + 36 × 0.95 + 10) = 45.04 mA

T_{RISE} = T_{FALL} = 0.85 ns at C_pad_load = 10 pF (includes up to 25% safety margin)

T_{RISE} = T_{FALL} = 4 ns at C_pad_load = 50 pF (includes up to 25% safety margin)

Example 2 (low-medium speed, medium current):

Rtx_out_high = Rtx_out_low = 10 Ω

R1 = 220 Ω (±5%), P(r1)min = 0.018 Ω

R2 = 390 Ω (±5%), P(r2)min = 0.032 Ω

Imax_tx = 5.5V / (220 × 0.95 + 390 × 0.95 + 10) = 9.17 mA

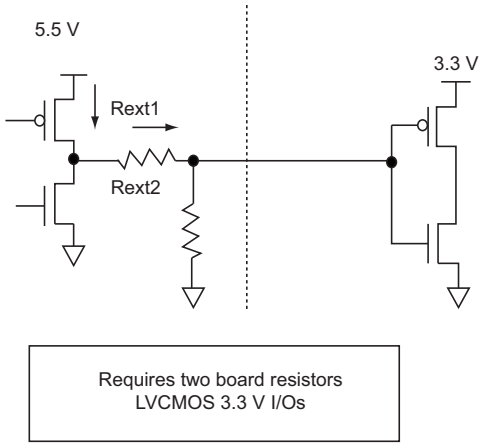
T_{RISE} = T_{FALL} = 4 ns at C_pad_load = 10 pF (includes up to 25% safety margin)

TRISE = T_{FALL} = 20 ns at C_pad_load = 50 pf (includes up to 25% safety margin)

Other values of resistors are also allowed, as long as the resistors are sized appropriately to limit the voltage at the receiving end to 2.5V < V_{IN} (rx) < 3.6V when the transmitter sends a logic 1.

This range of V_{IN_DC} (rx) must be ensured for any combination of the transmitter supply (5V ± 0.5V), transmitter output resistance, and board resistor tolerance.

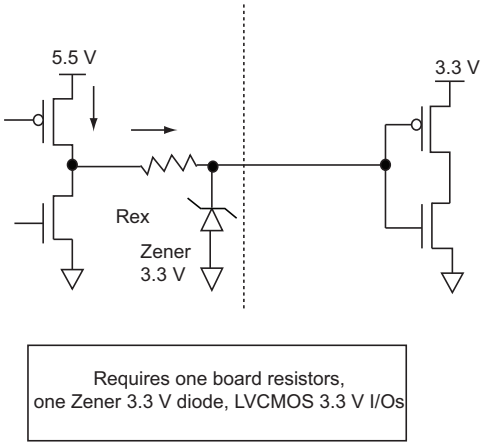
Figure 61 • **5V-Input Tolerance Solution 1**



6.13.1.2 Solution 2

The board design must ensure that the reflected waveform at the pad does not exceed the voltage overshoot/undershoot limits provided in the datasheet. This is a requirement to ensure long-term reliability. This solution also works for a 3.3V PCI configuration, but the internal diode must not be used for clamping, and the voltage must be limited by the external resistors and Zener. Adhering to the recommended operating conditions on the diode clamping creates an excessive pad DC voltage of 4V (3.3V + 0.7V = 4V).

Figure 62 • **5V Input Tolerance Solution 2**



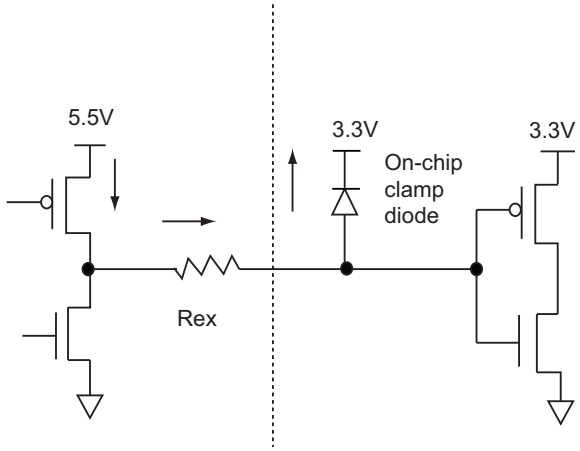
6.13.1.3 Solution 3

The board-level design must ensure that the reflected waveform at the pad does not exceed limits provided in the overshoot and undershoot limits. This is a long-term reliability requirement. Bus switch provides high-speed switching without adding propagation delay or generating additional ground bounce noise. These are ideal for voltage translation interfaces between buses, and in applications that require isolation and protection. Well-implemented bus switch designs maximize the bus speed.

This solution also works for a 3.3V PCI/PCIX configuration, but the internal diode must not be used for clamping, and the voltage must be limited by the bus switch. Adhering to the recommended operating conditions on the diode clamping might create an excessive pad DC voltage of 4V (3.3V + 0.7V = 4V).

Solution 3 requires a bus switch on the board and LVTTL/LVCMOS 3.3V I/Os.

Figure 63 • **5V Input Tolerance Solution 3**



The following table shows the slew rate control for all the three solutions.

Table 74 • **Slew Rate Control**

Solutions	Board Components	Speed	Limitations
1	Two resistors	Low to high1	Limited to transmitter's drive strength
2	Resistor and Zener 3.3V	Medium	Limited to transmitter's drive strength
3	Bus switch	High	N/A

Speed and current consumption increase as board resistance values decrease.

6.13.2 5V Output Driving Compatibility

SmartFusion2 and IGLOO2 I/Os must be set to 3.3V LVTTL mode or 3.3V LVCMOS mode to reliably drive 5 V TTL receivers. It is also critical that there is no external I/O pull-up resistor to 5V, since this pulls the I/O pad voltage beyond the 3.6V absolute maximum value and, consequently, cause damage to the I/O. When set to 3.3V LVTTL mode or 3.3V LVCMOS mode, the I/Os can directly drive signals into 5 VTTL receivers. In fact, noise margin levels V_{OL} = 0.4V and VOH = 2.4V in both 3.3V LVTTL and 3.3V LVCMOS modes exceed the V_{IL} = 0.8 V and V_{IH} = 2V level requirements of 5V TTL receivers. Therefore, level 1 and level 0 are recognized correctly by 5V TTL receivers.

6.14 I/Os in Conjunction with Fabric, MDDR/FDDR, and MSS/HPMS Peripherals

6.14.1 DDRIOs with MDDR/FDDR

If MDDR/FDDR is selected, Libero SoC automatically connects MDDR/FDDR signals to the DDRIOs. Depending on the memory configuration, the required DDRIOs are used by Libero SoC. The unused DDRIO are available to connect to the FPGA fabric.

6.14.2 DDRIOs with Fabric

If MDDR/FDDR is not selected, DDRIOs are available to the FPGA fabric. DDRIOs must be manually configured in Libero SoC.

6.14.3 MSIOs/MSIODs with MSS or HPMS Peripherals

If MSS or HPMS peripherals are selected, Libero SoC automatically connects MSS or HPMS peripheral signals to either MSIOs or to the MSIODs. The unused MSIOs or MSIODs are available to connect to the FPGA fabric.

6.14.4 MSIOs/MSIODs with Fabric

If HPMS peripherals are not selected, MSIOs/MSIODs are available to the FPGA fabric. MSIOs and MSIOD must be manually configured in Libero SoC.

6.15 JTAG I/O

The system controller implements the functionality of a JTAG slave with IEEE 1532 support, which also implies IEEE 1149.1 compliance. JTAG communicates with the system controller using a command register that conveys the JTAG instruction to be executed and a 128-bit data I/O buffer that transfers any associated data. The TAP controller uses 8-bit instructions consistent with previous Microchip families.

The JTAG pin standards are in accordance with MSIO standards. The JTAG pins can be run at any voltage from 1.5V to 3.3V (nominal). The I/O voltage of JTAG interface is set by powering the VDDI(JTAG) power pin with the desired I/O voltage. Core voltage must also be powered for the JTAG state machine to operate, even if the device is in bypass mode. Isolating the JTAG power supply in a separate I/O bank gives greater flexibility with supply selection and simplifies power supply and board design. If the JTAG interface is not used, with no plan to use it even in the future, the **VDDI** (JTAG) pin together with the TRSTB pin should be tied to GND. For mandatory bank supplies, see the *AN4153: Board Design Guidelines for SmartFusion2 SoC and IGLOO2 FPGAs Application Note*.

Table 75 • **JTAG Pin Description**

Name	Type	Description
JTAGSEL	Input	JTAG controller selection Depending on the state of the JTAGSEL pin, an external JTAG controller detects the FPGA fabric TAP/auxiliary TAP. The JTAGSEL pin should be connected to an external pull-up resistor such that the default configuration selects the FPGA fabric TAP. Logic 1: FPGA fabric TAP selected Logic 0: AUX TAP selected
TCK	Input	Test clock Serial input for JTAG boundary scan, ISP, and UJTAG. The TCK pin does not have an internal pull-up/-down resistor. If JTAG is not used, Microchip recommends tying it off TCK to GND or **VDDI** (JTAG) through a resistor placed close to the FPGA pin. This prevents JTAG operation in case TMS enters an undesired state. To operate at all **VDDI** (JTAG) voltages, the resistor values mentioned in Table 76, page 110 are recommended.
TDI	Input	Test data Serial input for JTAG boundary scan, ISP, and UJTAG usage. There is an internal weak pull-up resistor on the TDI pin.
TDO	Output	Test data Serial output for JTAG boundary scan, ISP, and UJTAG usage. TDS does not have an internal pull-up/pull-down resistor. Signal drive strength depends on the operating voltage: 3.3V (16 mA), 2.5V (16 mA), 1.8V (12 mA), or 1.5V (8 mA).
TMS		Test mode select The TMS pin controls the use of the IEEE1532 boundary scan pins (TCK, TDI, TDO, and TRSTB). There is an internal weak pull-up resistor on the TMS pin. The signal drive strength depends on the operating voltage: 3.3V (16 mA), 2.5V (16 mA), 1.8V (12 mA), or 1.5V (8 mA).

Table 75 • **JTAG Pin Description** ***(continued)***

Name	Type	Description
TRSTB		Boundary scan reset pin. The TRSTB pin functions as an active-low input to asynchronously initialize (or reset) the boundary scan circuitry. There is an internal weak pull-up resistor on the TRSTB pin. If JTAG is not used, an external pull-down resistor must be included to ensure the TAP is held in reset mode. The resistor values must be chosen from Table 76, page 110 and must satisfy the parallel resistance value requirement (multiple devices connected via JTAG chain). The values in Table 76, page 110 correspond to the resistor recommended when a single device is used. In critical applications, a fault in the JTAG circuit allows the device entering an undesired JTAG state. In such cases, Microchip recommends tying off TRSTB to GND through a resistor placed close to the FPGA pin. The TRSTB pin also resets the serial wire JTAG debug port (SWJ-DP) circuitry within the Cortex-M3 processor.

Table 76 • **Recommended Tie-Off Values for the TCK and TRST Pins**

V_{DDI}(JTAG)	Tie-Off Resistance[1, 2]
V_{DDI}(JTAG) at 3.3V	200 Ω to 1 KΩ
V_{DDI}(JTAG) at 2.5V	200 Ω to 1 KΩ
V_{DDI}(JTAG) at 1.8V	500 Ω to 1 KΩ
V_{DDI}(JTAG) at 1.5V	500 Ω to 1 KΩ

1. The TCK pin can be pulled up/down.
2. The TRSTB pin can only be pulled down.

6.16 Dedicated I/O

SmartFusion2 and IGLOO2 devices have following dedicated I/Os:

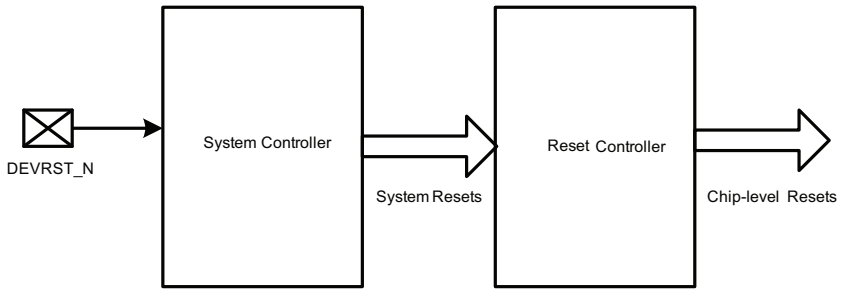
- Device Reset I/O
- Crystal Oscillator I/O
- SerDes I/O

6.16.1 Device Reset I/O

SmartFusion2 and IGLOO2 devices have a dedicated input reset, which, when asserted, resets the device (chip) as a whole. The device reset feeds the system controller, which generates the system reset for the reset controller to reset the entire device.

The following figure shows the full chip reset flow from device reset.

Figure 64 • **Chip Level Resets From Device Reset**



Asserting device reset causes a SmartFusion2 or IGLOO2 device to exit Flash*Freeze mode; this is very useful in recovering from a situation where the device enters Flash*Freeze mode without the Flash*Freeze exit mechanism being correctly configured in the I/O cells or in the real-time clock (RTC). This can be considered a cold reset, as it resets all parts of the device. For more information on how different reset signals are generated, see *UG0331: SmartFusion2 Microcontroller Subsystem User Guide* and *UG0448: IGLOO2 High Performance Memory Subsystem User Guide*.

6.16.1.1 Port List and I/O Pins

Table 77 • **Device Reset I/O Pin**

Pin	Type	I/O	Description
DEVRST_N	Analog	Input	Device reset is an asynchronous input, and powered by V_{PP} (active low).

6.16.2 Crystal Oscillator I/O

SmartFusion2 and IGLOO2 devices have two dedicated I/O pins (EXTLOSC and XTLOSC) connected to each on-chip crystal oscillator. These I/O pins can be connected to a crystal, (ceramic) resonator, or an RC circuit.

6.16.2.1 Crystal Oscillator I/O Pins

Table 78 • **Crystal Oscillator I/O Pins**

Pin	Type	I/O	Description
EXTLOSC	Analog	Input	Dedicated pin for a crystal external RC network connection
XTLOSC	Analog	Input	Dedicated pin to be used only for crystal connection

For detailed information on the configuration of these pins and operational modes, see the *UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide*.

6.16.3 SerDes I/O

The SerDes I/Os available in SmartFusion2 and IGLOO2 devices are dedicated to high-speed serial communication protocols. The SerDes I/O supports any user-defined high-speed serial protocol implementation in fabric. Supported protocols include PCI Express 2.0, XAUI, serial gigabit media independent interface (SGMII), and serial rapid I/O (SRIO). These protocols access the SerDes lanes through the physical media attachment (PMA) and physical coding sub-layer (PCS) within SerDes interface. For more information, see the *UG0447: SmartFusion2 and IGLOO2 FPGA High Speed Serial Interfaces User Guide*.

This section describes the SerDes I/O pins, SerDes I/O banks, SerDes I/O standards, and board-level design considerations available.

6.16.3.1 SerDes I/O Banks

The SerDes I/Os reside in dedicated I/O banks. The number of SerDes I/Os depends on the device size and pin count. For example, the M2GL050 device has two SERDES_IFs (SERDES_IF0 and SERDES_IF1), which reside in two out of ten I/O banks (bank #6 and bank #9). The M2GL010 device, on the other hand, has only one SERDES_IF (SERDES_IF0), which resides in bank #5.

For details on I/O bank locations and I/O electrical specifications, see the *DS0128: SmartFusion2 and IGLOO2 FPGA Datasheet*.

6.16.3.2 SerDes I/O Pins

Each SerDes interface in SmartFusion2 and IGLOO2 devices has four SerDes I/O data lanes or sixteen SerDes I/Os available for accessing the SerDes interface (SERDESIF block). Each data lane has two pairs of differential signals: one for transmit data (TxDP, TxDN) and the other for receive data (RxDP, RxDN). Data lanes are multiplexed to support different serial protocols and are scalable to various link widths such as x1, x2, and x4. These settings can be configured in the SERDES_IF macro using the Libero SoC software. Each SERDES_IF has two sets of dedicated power, clock, and reference signals. One set for data lanes 0 and 1 and another for data lanes 2 and 3. For SerDes I/O pin names and descriptions, see the *DS0115: SmartFusion2 Pin Descriptions Datasheet* and *DS0124: IGLOO2 Pin Descriptions Datasheet* documents.

7 Glossary

7.1 Acronyms

CCC
Clock conditioning circuits

DDRIO
Double data rate input output

ECC
Error correction code

ESD
Electrostatic discharge protection

FDDR
Controller for external DDR memory

HPMS
High-performance memory subsystem

HSTL
High-speed transceiver logic

IOA
Input output analog

IOD
Input output digital

LPDDR
Low power double data rate memory

LPE
Low power exit

LSB
Least significant bit

LSRAM
Large static random access memory

LVDS
Bus LVDS

LVPECL
Low-voltage positive emitter coupled logic

LVTTL
Low voltage transistor transistor logic

MDDR
Microcontroller subsystem double data rate

MLVDS
Multipoint LVDS

MSB

Most significant bit

MSIO

Multi-standard I/O

MVN

MultiView Navigator

ODT

On-die termination

RSDS

Reduced swing differential signaling

SerDes

Serializer/deserializer

SSTL

Stub series terminated logic

µSRAM

Micro static random access memory

7.2 Terminology

Bus Keeper

Holds the signal on an I/O pin at its last driven state.

Clusters

Clusters are formed by grouping a certain number of logic elements and interconnecting them. This is related to the clustered routing architecture of the SmartFusion2 SoC and IGLOO2 FPGA fabric.

Dual-Port Mode

SRAM with two independent ports through which both read and write operation can be done.

Feed-Through Write (Write-Bypass Write)

A write operation in which the data written appears on the SRAM output ports immediately for non-pipeline mode and next clock cycle for pipeline mode.

Flow-Through Read

A read operation performed with the output not being registered by the output pipeline registers.

Hot Insertion

Capability to connect I/O to external circuitry even after power-up.

I/O Cluster

I/O cluster is formed by grouping either three or four I/O modules.

I/O Module

The logic element consists of flip-flops and routing multiplexers. This logic element interfaces the user I/Os to fabric routing.

Inference

Using RTL to infer math blocks

Inter-Cluster Routing

Inter-cluster routing refers to routing resources between various types of clusters.

Interface Cluster

An interface cluster is formed by grouping 12 interface logic elements.

Interface Logic

The logic element consists of a 4-input LUT and a D flip-flop. This logic element interfaces the hard macros (LSRAMs, µSRAMs, and math blocks) to fabric routing.

Intra-Cluster Routing

Intra-cluster routing refers to routing resources existing inside a specific cluster.

Logic Cluster

A logic cluster is formed by grouping 12 logic elements.

Logic Element

The basic logic element in the SmartFusion2 SoC and IGLOO2 FPGA fabric, consisting of a 4-input LUT, a D-flip-flop, and a dedicated carry chain.

Low-Power Exit

Logic for the chip to come out from low-power state.

Multi-Channeling

Multi-threading done for a chain of math blocks

Multi-Threading

Using a math block for performing more than one computation by time multiplexing it.

Pipelined Operation

The mode of operation where the math block output is registered at the pipeline registers.

Pipelined Read

A read operation performed with the output being registered by the output pipeline registers.

Read Before Write Mode

In read before write operation, the data output will be updated with the content of write address before write.

Simple Write

A write operation in which the data written does not appear on the SRAM output ports.

STMR

Self-corrected triple module redundancy

Transparent Mode

Non-registered/Non-pipelined mode

Two-Port Mode

SRAM with two ports, one dedicated to read operations and the other dedicated to write operations.