Rate This Document

CoreMBX v2.0

Handbook

Actel
POWER MATTERS

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200138-0

Release: October 2008

Trademarks

Table of Contents

Introduction

Core Overview

At the basic level, CoreMBX (Mail Box) allows data messages (mail) to pass back and forth from one processing element to another. At pre-synthesis configuration time, the user is able to choose whether this message passing capability is implemented with one or more instances of dual-port SRAM hard macros, or with one or more instances of FIFO hard macros. Implementation is limited to those device families that employ true dual-port SRAM macros: IGLOO®, Fusion®, and ProASIC®3 families of devices.

The first bus uses a set of AMBA advanced high-performance bus (AHB) slave connections. This first bus is used by the first processing element, such as Cortex™-M1, to initialize and pass messages back and forth to a second processing element, such as CoreABC or Core8051s, that uses a set of AMBA 2 or AMBA 3 advanced peripheral bus (APB) slave connections. Note that the first bus may be asynchronous to the second bus; therefore synchronization logic is included internally.

A set of optional output connections (Init/Config interface) are provided. These output connections can initialize the SRAM instances within a connected instance of CoreABC, if CoreABC is configured to operate in "soft" mode. This initialization interface is initiated by the first processor on the “A” bus connections for writes, and the writes are synchronized to the clock of the second processor on the “B” bus connections. For example, if an instance of CoreABC is to be initialized via its INIT* inputs, it should be connected to the APB “B” bus. Another processor, such as Cortex-M1 via an AHB or AHB-Lite bridge, should be connected to the “A” bus.

Several aspects of CoreMBX can be configured using top-level parameters (Verilog) or generics (VHDL). For a detailed description of the parameters/generics, refer to “Parameters/Generics” on page 15. The CoreMBX block diagram is shown in Figure 1 on page 6. A typical application using CoreMBX is shown in Figure 2 on page 6.

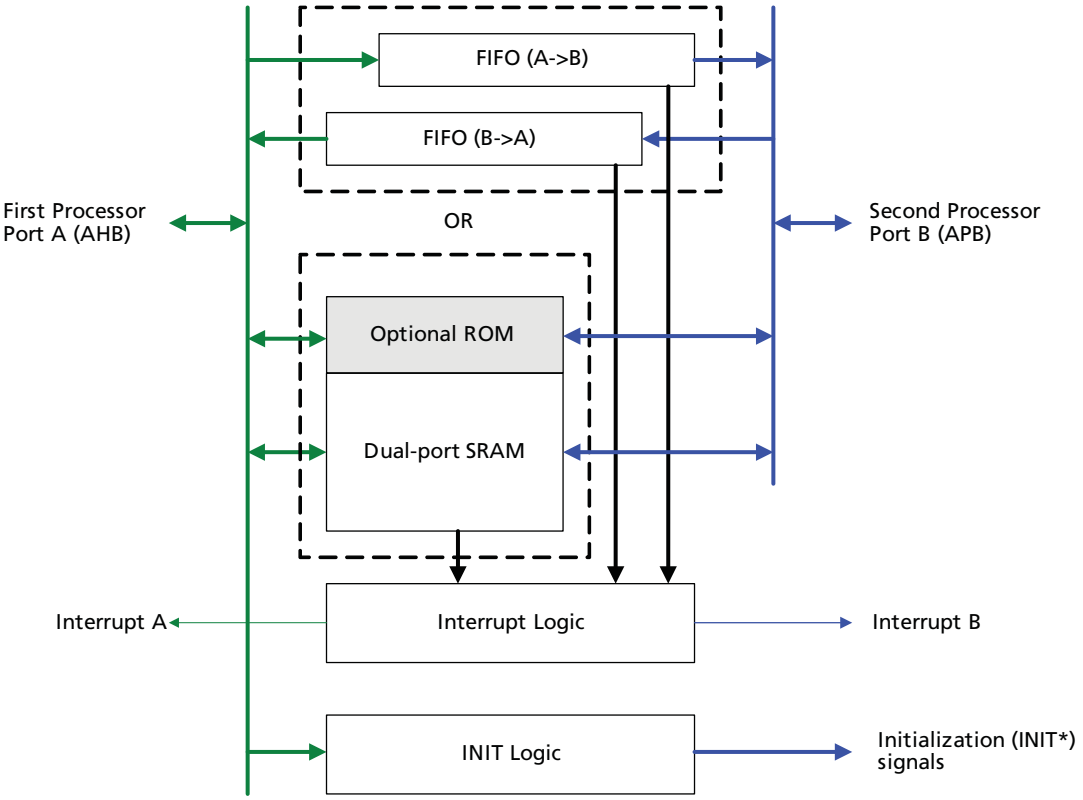

Figure 1 · CoreMBX Block Diagram

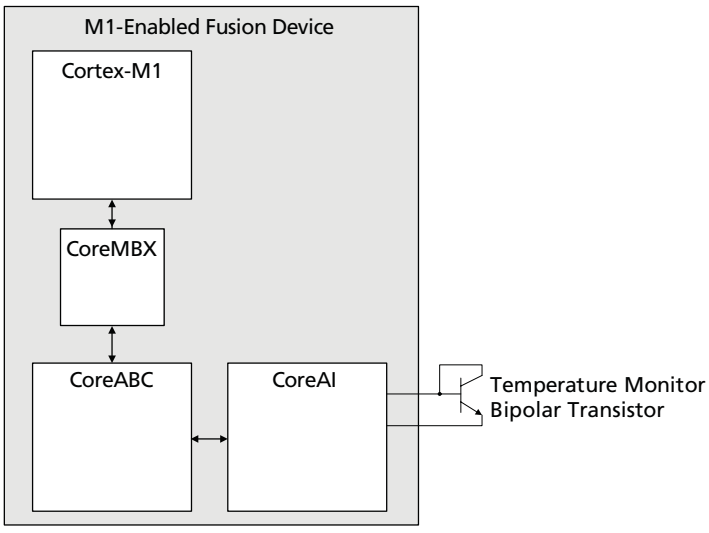

Figure 2 · CoreMBX Typical Application

Key Features

CoreMBX has the following features:

- Mailbox simultaneously accessible from an AMBA AHB/AHB-Lite master and an AMBA 2 APB master
- Mailbox memory storage elements made of either dual-port SRAM or FIFO blocks
- Init/Config master interface suitable for initializing CoreABC via AHB/AHB-Lite master
- Configurable number of interrupt flags between two processors
- Optional ROM built of FPGA tiles for up to 40 16-bit ROM words

Core Version

This handbook applies to CoreMBX v2.0.

Supported Interfaces

CoreMBX is available with one AHB slave interface that must be connected to an AMBA AHB or AMBA AHB-Lite master, and one AMBA 2 APB slave interface that must be connected to an AMBA 2 APB or AMBA 3 APB master.

Actel recommends using the CoreConsole IP Deployment Platform (IDP) or SmartDesign within Libero IDE to connect and configure CoreMBX in a multi processor-based system, such as Cortex-M1, Core8051s or CoreABC.

Utilization and Performance

CoreMBX has been implemented in the Actel IGLOO, Fusion and ProASIC3 families of devices. A summary of the data for CoreMBX is listed in Table 1 and Table 2 on page 8. CoreMBX can be used with any devices in the IGLOO, Fusion and ProASIC3 families of devices.

Table 1 · CoreMBX Device Utilization and Performance (Minimum Configuration)

Family	FPGA Resources				Utilization		Performance
	Sequential	Combinatorial	Total	RAM	Device	%	
IGLOO	25	79	104	1	AGLE600	1%	> 60 MHz
Fusion					AFS600		
ProASIC3					A3PE600		

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics that differ from the default values were set as follows: CFG_INTERRUPT_A=2, CFG_INTERRUPT_B=2, MBX_WIDTH=8, PORT_B_WIDTH=8.

Table 2 · CoreMBX Device Utilization and Performance (Maximum Configuration)

Family	FPGA Resources				Utilization		Performance
	Sequential	Combinatorial	Total	RAM	Device	%	
IGLOO	314	864	1178	16	AGLE600	9%	> 60 MHz
Fusion					AFS600		
ProASIC3					A3PE600		

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level parameters/generics that differ from the default values were set as follows: MBX_DEPTH=2048, MBX_WIDTH=32, PORT_B_WIDTH=32, USE_INIT=1, MSG_A2B_FLAGS=32, MSG_B2A_FLAGS=32, CFG_ROM=40, ROM_ADDR_1 to ROM_ADDR_40 = 1 to 40, respectively, ROM_DATA_1 to ROM_DATA_40 = 1 to 40, respectively.

1 Functional Block Descriptions

CoreMBX, shown in Figure 1 on page 6, consists of memory macros (either one or more dual-port SRAM macros, or two or more FIFO macros), an interrupt logic block that contains optional flag logic for sending interrupts from processor A to processor B, or from processor B to processor A; and a logic block that interfaces to an Init/Config slave interface, such as that used by the CoreABC processor.

Memory Usage

FIFO Usage

When CoreMBX is configured to use FIFO blocks, one set of FIFOs is instantiated to send messages from the first processor (processor A on the left side of Figure 1) to the second processor (processor B on the right side of Figure 1). Another set of FIFOs is instantiated to send messages from the second processor to the first processor.

The empty and full flags from each set of FIFOs are synchronized to the rising edge of the first and second clock domains to serve as interrupt sources.

The content and structure of the data messages that get passed on from one processor to the other are left to be created by the developer of the application. CoreMBX is merely the receptacle of the messages and assumes no knowledge of the actual content of the messages.

Dual-Port SRAM Usage

When CoreMBX is configured to use dual-port SRAM blocks and the INIT* ports are being used to initialize an instance of CoreABC that is connected to the APB slave bus, the INITDONE signal may be used by CoreABC to qualify reading the contents of the messages (content) sent from the first processor. The first processor may wish to initialize the messages (content) of all of the memory locations of the dual-port SRAM blocks prior to initializing the INIT* ports of CoreABC. The optional message flags, via the interrupt outputs, may be used to indicate this functionality as well, or instead of, the INIT* outputs from CoreMBX. In other words, the CoreABC processor may check the status of one or more of the message flags and proceed with its own initialization routines, including eventual clearing of the message flags. Refer to "Flag Logic" on page 10 for more information.

When using dual-port SRAM blocks, an interrupt can be generated whenever a write occurs to either port (this is a configurable option). For example, if a write occurs to the A-port (initiated by first processor), an interrupt will be generated to the second processor. If a write occurs to the B-port (initiated by second processor), an interrupt will be generated to the first processor. If this behavior is not desired, each interrupt may be disabled permanently at configuration time or temporarily via software control during operation. Disabling each interrupt via software control will not clear a pending interrupt, and it will merely mask the true interrupt source. Reading each interrupt status register will clear each interrupt, except for the interrupt contribution from the optional flag logic, which can only be cleared by the processor to which the interrupt is destined. Refer to "Flag Logic" on page 10 for more details on operation of the optional flag logic.

As with the FIFO configuration, the actual content and meaning of the data messages that are written into each memory location are dependent on the application that the developer has in mind.

Optional ROM Usage

CoreMBX can be configured to include an optional block of ROM that will be built from FPGA tiles. This optional ROM can have up to 40 unique address/data pairs defined via the CoreConsole or SmartDesign configurator GUI. The default value of unaddressed locations can be set to any non-negative integer from 0 to (2**16)-1, or alternatively, to a "don't care" value that will be most efficient in logic synthesis optimization. The ROM can be read from either the AHB slave interface (port A) or the APB slave interface (port B), when either of the individual ROM select lines (ROMSEL_A, ROMSEL_B) are logic 1 and a read from the DPRAMA or DPRAMB address space occurs. Refer to "Register Maps: Programmer's View" on page 21 for more information.

Flag Logic

Within CoreMBX, flag logic that contributes to the interrupt outputs is optionally included. The flag logic can be configured to allow processor A to set message flag bits destined for processor B, by way of the INTERRUPT_B output. The flag logic can also be configured to allow processor B to set message flag bits destined for processor A, by way of the INTERRUPT_A output. If used, the message flag bits are set by writing logic 1 values to any or all of the bits; logic 0 values written to any of the message flag bits are ignored. Only the processor to which the interrupt is destined can clear each of the message flag bits, also by writing logic 1 values to any or all of the bits. Synchronization flip-flops are used to ensure that the message flag bits are synchronous to each of the processor clock domains.

2 Tool Flows

Licenses

CoreMBX is licensed in two ways. Depending on your license type, tool flow functionality may be limited.

Obfuscated

Complete RTL code is provided for the core, allowing the core to be instantiated within CoreConsole or SmartDesign. Simulation, synthesis and layout can be performed within the Libero® Integrated Design Environment (IDE). The RTL code for the core is obfuscated and some of the testbench source files are not provided; they are pre-compiled into the compiled simulation library instead.

RTL

Complete RTL source code is provided for the core and testbenches.

CoreConsole

CoreMBX is preinstalled in CoreConsole. To use the core, simply drag it from the IP core list into the main window. The CoreConsole project can be exported to Libero IDE at this point, providing access just to CoreMBX. Other IP blocks can be interconnected, allowing the complete system to be exported from CoreConsole to Libero IDE.

The core can then be configured using the configuration GUI within CoreConsole, as shown in Figure 2-1 and Figure 2-2 on page 13. Parameters can be configured within the CoreConsole GUI and are fully described in the "Parameters/Generics" on page 15. Cross references to the corresponding parameters in the GUI configuration screen are shown in boxes in Figure 2-1 on page 12 and Figure 2-2 on page 13.

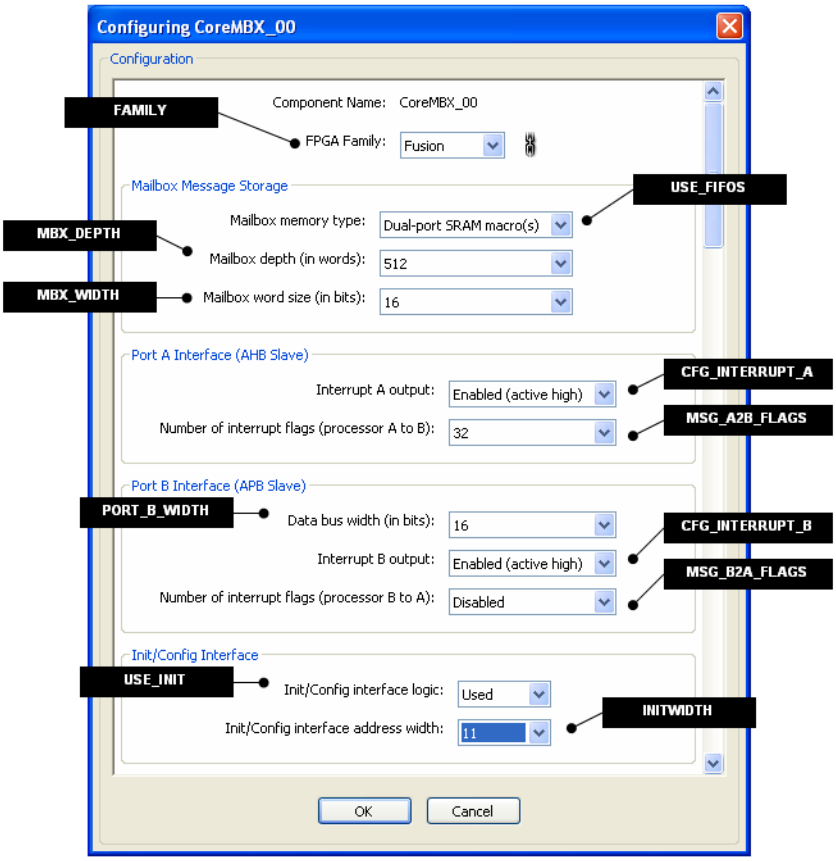

Figure 2-1 · CoreMBX Configuration Within CoreConsole

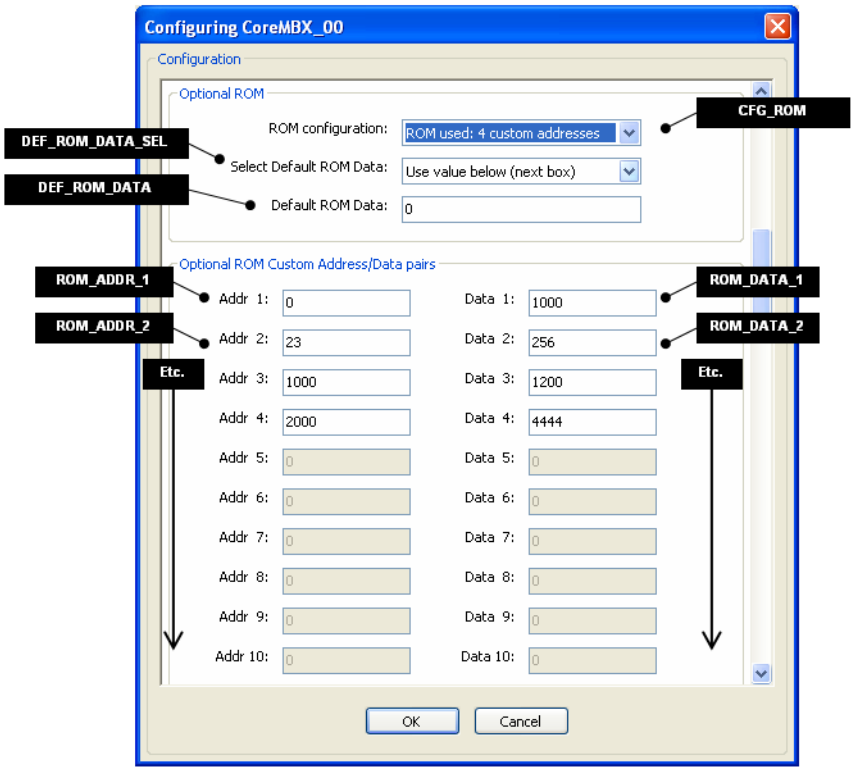

Figure 2-2 · CoreMBX Configuration Within CoreConsole (Continued)

Importing into Libero IDE

After generating and exporting the core from CoreConsole, the core may be imported into Libero IDE. Create a new project in Libero IDE and import the CoreConsole project from the *LiberoExport* directory within the CoreConsole installation directory tree. Libero IDE will then install the core and the selected testbenches into its project, along with constraints and documentation. Refer to the Libero IDE online help for details on how to create a CoreConsole subsystem within Libero IDE.

Note: If two or more DirectCores are required, they can both be included in the same CoreConsole project and imported into Libero IDE at the same time.

SmartDesign

As an alternative to the CoreConsole IDP, you can use SmartDesign to create an FPGA system using CoreMBX. CoreMBX is available for download to the SmartDesign IP Catalog via the Libero IDE web repository. Note that the CoreMBX IP configuration GUI in SmartDesign will very closely resemble the CoreConsole configuration shown in Figure 2-1 on page 12 and Figure 2-2 on page 13. For information on using SmartDesign to instantiate, configure, connect and generate cores, refer to the Libero IDE online help. For a detailed tutorial on DirectCore IP flow using SmartDesign, refer to *Using DirectCore in Libero IDE*.

Simulation Flows

To run simulations, select the user testbench within CoreConsole or SmartDesign through the CoreMBX configuration GUI. Run **Save & Generate** from the Generate pane.

When CoreConsole or SmartDesign generates the Libero IDE project, it will install the appropriate test bench files.

To run the testbenches, set the design root to the CoreMBX instantiation in the Libero IDE File Manager and click the **Simulation** icon in the Libero IDE Design Flow window. This will invoke Model*Sim*® and automatically run the simulation.

Synthesis in Libero IDE

To run Synthesis on the core with parameters set in CoreConsole or SmartDesign, set the design root appropriately and click the **Synthesis** icon in Libero IDE. The Synthesis window appears, displaying the Synplicity project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To perform synthesis, click the **Run** icon.

Place-and-Route in Libero IDE

After setting the design root appropriately and running Synthesis, click the **Layout** icon in Libero IDE to invoke Designer. CoreMBX requires no special place-and-route settings.

3 Interface Descriptions

CoreMBX is available with one AHB slave interface and one APB slave interface and can easily connected to an AHB bus and an APB bus within the CoreConsole IDP or SmartDesign canvas.

Parameters/Generics

CoreMBX has parameters (Verilog) and generics (VHDL), described in Table 3-1. All parameters and generics are integer types.

Table 3-1 · CoreMBX Parameter/Generic Descriptions

Name	Valid Range	Description
FAMILY	15, 16, 17, 20, 21, 22	Technology-specific device family: 15 - ProASIC3 16 - ProASIC3E 17 - Fusion (default) 20 - IGLOO 21 - IGLOOe 22 - ProASIC3L
CFG_INTERRUPT_A	0 to 3	Interrupt A output configuration: 0 - Interrupt A active-high (default) 1 - Interrupt A active-low 2 - Interrupt A permanently disabled (statically held at logic 0) 3 - Interrupt A permanently disabled (statically held at logic 1)
CFG_INTERRUPT_B	0 to 3	Interrupt B output configuration: 0 - Interrupt B active-high (default) 1 - Interrupt B active-low 2 - Interrupt B permanently disabled (statically held at logic 0) 3 - Interrupt B permanently disabled (statically held at logic 1)
USE_FIFOS	0 or 1	This value determines whether FIFO hard macros or dual-port SRAM hard macros are used as the storage elements for messages. 0 - dual-port SRAM macros used (default) 1 - FIFOs are used
MBX_DEPTH	512, 1024, or 2048	This value sets the word-depth of the memory elements (FIFOs or dual-port SRAMs) used for message storage. The default value is 512.
MBX_WIDTH	8, 16, or 32	This value sets the bit-width* of the memory elements (FIFOs or dual-port SRAMs) used for message storage. The default value is 16.

Table 3-1 · CoreMBX Parameter/Generic Descriptions (continued)

Name	Valid Range	Description
PORT_B_WIDTH	8, 16, or 32	This value specifies the width of the processor connected to the APB slave "B" port. Note that this value affects the manner in which the internal registers are accessed, refer to "Port B (AMBA 2 or AMBA 3 APB Slave) Register Interface" on page 26 for more information. The default value is 16. Note that PORT_B_WIDTH should be greater than or equal to MBX_WIDTH, otherwise MSB bits of the memory element(s) will not be accessible from the B processor.
USE_INIT	0 or 1	This value determines whether or not the Init/Config outputs INIT* are used to initialize a second processor (such as CoreABC). The default value of 0 disables the INIT* outputs and removes an internal loadable counter that is connected to the INITADDR outputs.
INITWIDTH	1 to 16	This value specifies the width of the INITADDR outputs that are used to initialize the second processor (such as CoreABC). If using CoreABC, the value of INITWIDTH in CoreMBX must match the value of INITWIDTH in CoreABC. The default value is 16. This parameter also is used to set the width of an internal loadable counter that is used as the INITADDR outputs that increments after each write to the INITWDATA register.
MSG_A2B_FLAGS	0, 8, 16, or 32	This setting determines how many message flags (A_FLAGS_A2B[31:0] and B_FLAGS_A2B[31:0] register) are used to send an interrupt from processor A to processor B. The default setting of 0 disables message flag logic from processor A to processor B. If set to 8, the upper 24 bits of the internal flags (A_FLAGS_A2B[31:0] and B_FLAGS_A2B[31:0] register) will return static logic 0 values when read, and writes to the upper 24 bits will be ignored. Similarly, if set to 16, the upper 16 bits of the internal flags register will return static logic 0 values when read, and writes to the upper 16 bits will be ignored.
MSG_B2A_FLAGS	0, 8, 16, or 32	This setting determines how many message flags (A_FLAGS_B2A[31:0] and B_FLAGS_B2A[31:0] register) are used to send an interrupt from processor B to processor A. The default setting of 0 disables message flag logic from processor B to processor A. If set to 8, the upper 24 bits of the internal flags (A_FLAGS_B2A[31:0] and B_FLAGS_B2A[31:0] register) will return static logic 0 values when read, and writes to the upper 24 bits will be ignored. Similarly, if set to 16, the upper 16 bits of the internal flags register will return static logic 0 values when read, and writes to the upper 16 bits will be ignored.
CFG_ROM	0, 4, 8, 16, 32, 40	Optional ROM configuration: 0 - No ROM used (default) 4 - ROM used with 4 custom addresses 8 - ROM used with 8 custom addresses 16 - ROM used with 16 custom addresses 32 - ROM used with 32 custom addresses 40 - ROM used with 40 custom addresses The default value of 0 disables the optional ROM. If set to values greater than 0, the ROMSEL_A and ROMSEL_B inputs are used to qualify reads from the internal ROM values using the DPRAMA and DPRAMB address decodes, respectively, for reads from the A or B processors.

Table 3-1 · CoreMBX Parameter/Generic Descriptions (continued)

Name	Valid Range	Description
DEF_ROM_DATA_SEL	0 or 1	Select Default ROM Data: If set to 0, CoreMBX uses the DEF_ROM_DATA parameter to determine the default value (0 to (2**16)-1) of the ROM data for addresses that are not specified. If set to 1, the default ROM data will be set to a “don't care” value for optimal synthesis results. The default value of this parameter is 0.
DEF_ROM_DATA	0 to (2**16)-1	If CFG_ROM is not 0 and DEF_ROM_DATA_SEL is 0, this parameter determines the default value of ROM data for address/data pairs that are not configured with the ROM_ADDR_xx/ROM_DATA_xx parameter pairs. If CFG_ROM has the default value of 0, this parameter is ignored. The default value of this parameter is 0.
ROM_ADDR_1, ROM_ADDR_2, …, ROM_ADDR_40	0 to (2**11)-1	Custom ROM Addresses: there are 40 possible custom ROM addresses. If CFG_ROM is not 0, these ROM addresses are paired with the ROM_DATA_xx parameters to return custom data when the ROM is read from either processor A or processor B. Note that the number set in the CFG_ROM parameter must have all custom address/data pairs specified. For example, if CFG_ROM=4, then ROM_ADDR_1, ROM_ADDR_2, ROM_ADDR_3, and ROM_ADDR_4 must all be specified (as well as the corresponding ROM_DATA_1 to ROM_DATA_4 parameters); also note that in this example, ROM addresses that fall outside the values specified by ROM_ADDR_1 to ROM_ADDR_4 will return data whose value is determined by the DEF_ROM_DATA parameter. If CFG_ROM is the default value of 0, these parameters are ignored. The default collective value of these parameters is 0, therefore, it is imperative that all used addresses configured in the CFG_ROM parameter are set appropriately.
ROM_DATA_1, ROM_DATA_2, …, ROM_DATA_40	0 to (2**16)-1	Custom ROM Data: there are 40 possible custom ROM data values; if CFG_ROM is not 0, these ROM data values are paired with the ROM_ADDR_xx parameters to return custom data when the ROM is read from either processor A or processor B. Note that the number set in the CFG_ROM parameter must have all custom address/data pairs specified. For example, if CFG_ROM=4, then ROM_DATA_1, ROM_DATA_2, ROM_DATA_3, and ROM_DATA_4 must all be specified (as well as the corresponding ROM_ADDR_1 to ROM_ADDR_4 parameters). If CFG_ROM is the default value of 0, these parameters are ignored. The default collective value of these parameters is 0.

* *Note that more than 1 dual-port SRAM will be instantiated for 16 or 32-bit widths.*

Ports

The port signals for the CoreMBX macro are defined in Table 3-2 on page 18 and illustrated in Figure 3-1 on page 18. CoreMBX has from 139 (minimum configuration) to 202 (maximum configuration) I/O signals.

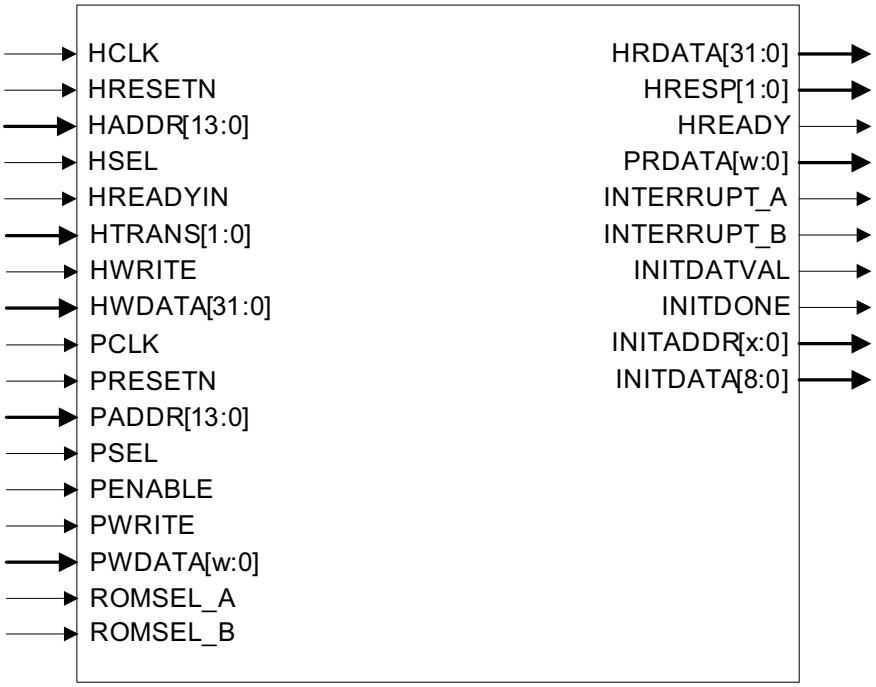

Figure 3-1 · CoreMBX I/O Signal Diagram

Table 3-2 · CoreMBX Signal Descriptions

Name	Type	Description
Port A (AHB Slave) Signals		
HCLK	Input	AHB clock: All AHB signals are synchronous to the rising edge of this clock signal.
HRESETN	Input	AHB active-low asynchronous reset.
HADDR[13:0]	Input	AHB address bus: This port is used to address registers and message data within CoreMBX.
HSEL	Input	AHB slave select: This signal selects CoreMBX for reads or writes via the first processor.
HREADYIN	Input	AHB ready input: This signal indicates that all other slaves are ready when high.
HTRANS[1:0]	Input	AHB transfer type: Indicates the type of current transfer: 00 - Idle 01 - Busy 10 - Non-sequential 11 - Sequential

Table 3-2 · CoreMBX Signal Descriptions (continued)

Name	Type	Description
HWRITE	Input	AHB write/read: If high, a write will occur when an AHB transfer to CoreMBX takes place; if low, a read from CoreMBX will take place.
HWDATA[31:0]	Input	AHB write data from AHB master.
HRDATA[31:0]	Output	AHB read data to AHB master.
HRESP[1:0]	Output	AHB transfer response: 00 - Okay 01 - Error 10 - Retry 11 - Split These pins are fixed to always return 00 (Okay response).
HREADY	Output	AHB ready signal: used to insert wait states on writes to the Init/Config interface in order to synchronize to the "B" clock domain PCLK.
Port B (APB Slave) Signals		
PCLK	Input	APB clock: All APB signals are synchronous to the rising edge of this clock signal.
PRESETN	Input	APB active-low asynchronous reset.
PADDR[13:0]	Input	APB address bus: This port is used to address registers and message data within CoreMBX.
PSEL	Input	APB slave select: This signal selects CoreMBX for reads or writes via the second processor.
PENABLE	Input	APB strobe: This signal indicates the second cycle of an APB transfer.
PWRITE	Input	APB write/read: If high, a write will occur when an APB transfer to CoreMBX takes place; if low, a read from CoreMBX will take place.
PWDATA[w:0]	Input	APB write data: w = PORT_B_WIDTH-1.
PRDATA[w:0]	Output	APB read data: w = PORT_B_WIDTH-1.
ROM Select Signals		
ROMSEL_A	Input	ROM select A: This input is used to select the optional ROM that can be read from processor A. If high (logic 1), processor A can read the optional ROM contents when addressing the DPRAMA register space (refer to Table 4-11 on page 26). If low (logic 0), processor A can read the DPRAMA contents when addressing the DPRAMA register space. If the CFG_ROM parameter is 0, this input is ignored and should be permanently tied low.

Table 3-2 · CoreMBX Signal Descriptions (continued)

Name	Type	Description
ROMSEL_B	Input	ROM select B: This input is used to select the optional ROM that can be read from processor B. If high (logic 1), processor B can read the optional ROM contents when addressing the DPRAMB register space (refer to Table 4-19 on page 31). If low (logic 0), processor B can read the DPRAMB contents when addressing the DPRAMB register space. If the CFG_ROM parameter is 0, this input is ignored and should be permanently tied low.
Interrupt Signals		
INTERRUPT_A	Output	Processor A interrupt: This interrupt output has different meanings depending on whether FIFO or dual-port SRAM elements are used. If FIFO elements are used (USE_FIFOS=1), this signal indicates that the B->A FIFO is not empty or that the A->B FIFO is full. If dual-port SRAM elements are used (USE_FIFOS=0), this signal indicates that a write from the B processor into the dual-port SRAM macros has occurred. This signal also contains contributions from optional flag logic from processor B to processor A.
INTERRUPT_B	Output	Processor B interrupt: This interrupt output has different meanings depending on whether FIFO or dual-port SRAM elements are used. If FIFO elements are used (USE_FIFOS=1), this signal indicates that the A->B FIFO is not empty or that the B->A FIFO is full. If dual-port SRAM elements are used (USE_FIFOS=0), this signal indicates that a write from the A processor into the dual-port SRAM macros has occurred. This signal also contains contributions from optional flag logic from processor A to processor B.
INITCFG Signals		
INITDATVAL	Output	This output can be connected to the INITDATVAL input of CoreABC to indicate that the INITADDR[x:0] and INITDATA[8:0] outputs are valid. It can be left unconnected if unused. If USE_INIT=0, this output is permanently tied low.
INITDONE	Output	This output can be connected to the INITDONE input of CoreABC to indicate that initialization of CoreABC's internal RAM blocks has completed. It can be left unconnected if unused. If USE_INIT=0, this output is permanently tied low.
INITADDR[x:0]	Output	These outputs can be connected to the INITADDR[x:0] inputs of CoreABC to configure the RAM blocks within CoreABC, when using it in soft mode (x=INITWIDTH-1). These outputs can be left unconnected if unused. If USE_INIT=0, these outputs are permanently tied low.
INITDATA[8:0]	Output	These outputs can be connected to the INITDATA[8:0] inputs of CoreABC to configure the RAM blocks within CoreABC, when using it in soft mode. These outputs can be left unconnected if unused. If USE_INIT=0, these outputs are permanently tied low.

Note: All signals are active-high (logic 1) unless otherwise noted.

4

Register Maps: Programmer's View

Port A (AHB Slave) Register Interface

Processor A has access to the register address map listed in Table 4-1. Individual register descriptions are listed in Table 4-1 to Table 4-11 on page 26. All registers are read/write unless otherwise noted. Any bits that are not listed in the descriptions of the registers will return logic 0 values when read by processor A, and will be ignored on writes from processor A.

Table 4-1 · Port A Register Address Map

HADDR[13:0]	Type	Register Name	Reset Value	Description
0x0000	R/W	INTCTRLA	0x00000000	Interrupt A control register.
0x0004	R	INTSTATA	0x00000000	Interrupt A status register (read-only).
0x0010	R/W	A_FLAGS_B2A	0x00000000	32 possible flag logic bits that can be used to send interrupts from processor B to processor A (readable and clear-able from processor A).
0x0020	R/W	A_FLAGS_A2B	0x00000000	32 possible flag logic bits that can be used to send interrupts from processor A to processor B (readable and set-able from processor A).
0x0030	R/W	INITWADDR	0x00000000	INITCFG indirect address register: the outputs of this register are connected to the INITADDR outputs. The contents are incremented after each write to the INITWDATA register.
0x0040	W	INITWDATA	Undefined	Write-only access to INITCFG initialization address space (used to write to the INITDATA[8:0] inputs of CoreABC).
0x0048	R/W	INITCTRL	0x00000000	Control register to set or clear the INITDONE output signal.
0x0050	W	FIFOWDA2B	N/A	Write-only access to send message from processor A to processor B. This address is only used if USE_FIFOS=1.
0x0054	R	FIFORDB2A	Undefined	Read-only access to read message from processor B to processor A. This address is only used if USE_FIFOS=1.
0x2000-0x3FFC	R/W	DPRAMA	Undefined	Processor A access to port A of internal dual-port SRAM instance(s) or optional ROM. This address space is only used if USE_FIFOS=0.

Note: Type designations: "R" = read-only, "R/W" = read/write, "W" = write-only

Table 4-2 · Interrupt A Control (INTCTRLA) Register Bit Description

Bit	Name	Description
3	EN_A_FLAGS_B2A_OR	Enable A_FLAGS_B2A_OR bit: This bit controls whether or not the ORed version of the flag bits from processor B to processor A (A_FLAGS_B2A[31:0]) contributes to the INTERRUPT_A output or if it is masked. If set to 1, the ORed flag contribution is enabled; otherwise, if set to 0, the ORed flag contribution is masked. The default value of 0 disables (masks) the contribution of the ORed flags to the INTERRUPT_A output.
2	EN_DPWRB	Enable DPWRB bit: This bit controls whether or not the DPWRB bit of the INTSTATA register contributes to the INTERRUPT_A output or if it is masked. If set to 1, the DPWRB bit contribution is enabled; otherwise, if set to 0, the DPWRB bit is masked. The default value of 0 disables (masks) the contribution of the DPWRB bit to the INTERRUPT_A output.
1	EN_FULL_A2B	Enable FULL_A2B bit: This bit controls whether or not the FULL_A2B bit of the INTSTATA register contributes to the INTERRUPT_A output or if it is masked. If set to 1, the FULL_A2B bit contribution is enabled; otherwise, if set to 0, the FULL_A2B bit is masked. The default value of 0 disables (masks) the contribution of the FULL_A2B bit to the INTERRUPT_A output.
0	EN_MSG_B2A	Enable MSG_B2A bit: This bit controls whether or not the MSG_B2A bit of the INTSTATA register contributes to the INTERRUPT_A output or if it is masked. If set to 1, the MSB_B2A bit contribution is enabled; otherwise, if set to 0, the MSG_B2A bit is masked. The default value of 0 disables (masks) the contribution of the MSG_B2A bit to the INTERRUPT_A output.

Table 4-3 · Interrupt A Status (INTSTATA) Register Bit Description

Bit	Name	Description
3	A_FLAGS_B2A_OR	ORed flag bits from processor B to processor A: If set to 1, this bit indicates that processor B has set one or more general purpose flags in the A_FLAGS_B2A[31:0] register to indicate to processor A that it must take some action, followed by clearing of these general purpose flags. This action is entirely application specific. This bit will not be cleared when reading this register, but rather when all bits of the A_FLAGS_B2A[31:0] register are cleared. If MSG_B2A_FLAGS=0, this bit will be permanently tied to logic 0.
2	DPWRB	Processor B has written to dual-port SRAM: If set to 1, this bit indicates that processor B has written data into the B port of the dual-port SRAM. This bit will be cleared when reading this register. If USE_FIFOS=1, this bit will be permanently tied to logic 0.
1	FULL_A2B	FIFO A->B is full: If set to 1, this bit indicates that the A->B FIFO (from processor A to processor B) is full. This bit will be cleared when at least one message has been read from the A->B FIFO by the B processor. If USE_FIFOS=0, this bit will be permanently tied to logic 0.
0	MSG_B2A	Message(s) available in FIFO B->A: If set to 1, this bit indicates that at least one message is available in the B->A FIFO (from processor B to processor A). This bit will be cleared when all messages have been read from the B->A FIFO. If USE_FIFOS=0, this bit will be permanently tied to logic 0.

Table 4-4 · Processor B to Processor A Flags (A_FLAGS_B2A) Register Bit Description

Bit	Name	Description
31:0	FLAGS	Message flag bits from processor B to processor A: These 32 possible flag logic bits can be used to send interrupts from processor B to processor A. Processor A can clear any of the 32 bits by writing a logic 1 in that bit position. For example, if the A_FLAGS_B2A[31:0] register value contains 0xFFAA0000 and processor A writes the value 0xFF000000 to this register, the value will be changed to 0x00AA0000 on subsequent reads, since writing logic 0 values to any of the bits is ignored.

Table 4-5 · Processor A to Processor B Flags (A_FLAGS_A2B) Register Bit Description

Bit	Name	Description
31:0	FLAGS	Message flag bits from processor A to processor B: These 32 possible flag logic bits can be used to send interrupts from processor A to processor B. Processor A can set any of the 32 bits by writing a logic 1 in that bit position. For example, if the A_FLAGS_A2B[31:0] register value contains 0x00550000 and processor A writes the value 0xAA000000 to this register, the value will be changed to 0xAA550000 on subsequent reads. Writing logic 0 values to any of the bits is ignored.

Table 4-6 · Init/Config Write Address (INITWADDR) Register Bit Description

Bit	Name	Description
n:0	INITADDR	Init/Config write address (n = INITWIDTH-1): If the parameter USE_INIT=1, this register is connected to the INITADDR[n:0] outputs. Note that the actual width of this register will be determined by the INITWIDTH parameter and can range from 1 to 16 bits; unused bits will return 0 when read, and will be ignored if processor A writes to them. If the Init/Config interface is used (USE_INIT=1), processor A should first write the starting address for the INITADDR[n:0] outputs to this register, followed by subsequent writes to the INITWDATA register, after which the contents of this register will be automatically incremented by 1. The Init/Config interface can be initialized in a connected instance of CoreABC operating in soft mode. If the parameter USE_INIT=0, these bits are permanently tied to logic 0 and writes to it have no effect and the INITADDR outputs will be tied to logic 0).

Table 4-7 · Init/Config Write Data (INITWDATA) Register Bit Description

Bit	Name	Description
8:0	INITDATA	Init/Config write data: If the parameter USE_INIT=1, these write-only bits are used to send initialization data out of the INITDATA[8:0] and INITDATAVAL ports that may be connected to an instance of CoreABC configured to operate in soft mode. If the parameter USE_INIT=0, these bits are permanently tied to logic 0 and writes to it have no effect.

Table 4-8 · Init/Config Control (INITCTRL) Register Bit Description

Bit	Name	Description
0	INITDONE	Init/Config done bit: If the parameter USE_INIT=1, this bit controls the state of the INITDONE output pin to indicate that the initialization/configuration process is completed. This signal should only be set to logic 1 by processor A after it has completed writing all initialization data to the INIT* outputs that may be connected to an instance of CoreABC configured to operate in soft mode; this is accomplished by processor A writing to the INITWADDR and INITWDATA registers. If the INITWIDTH parameter is 0, this register bit is permanently tied to logic 0 and writes to it have no effect.

Table 4-9 · FIFO Write Data A->B (FIFOWDA2B) Register Bit Description

Bit	Name	Description
n:0	DATA	FIFO write data (n = MBX_WIDTH-1): During a write cycle, this data will be written from processor A into the A->B FIFO (for sending message data from processor A to processor B) if USE_FIFOS=1. If USE_FIFOS=0, writing to this address has no effect. This register is write-only.

Table 4-10 · FIFO Read Data B->A (FIFORDB2A) Register Bit Description

Bit	Name	Description
n:0	DATA	FIFO read data (n = MBX_WIDTH-1): During a read cycle, this data will be read by processor A from the B->A FIFO, for sending message data from processor B to processor A, if USE_FIFOS=1. If USE_FIFOS=1, reading from this address has no effect, unknown logic values are returned and should be ignored. This register is read-only.

Table 4-11 · Dual-Port SRAM A (DPRAMA) Register Bit Description

Bit	Name	Description
n:0	DATA	Dual-port SRAM data (n = MBX_WIDTH-1): During a write cycle, this data will be written from processor A into the dual-port SRAM, if USE_FIFOS=0. Note that only every fourth 32-bit word aligned address is used: 0x2000, 0x2004, 0x2008, ..., 0x3FFC. If USE_FIFOS=1, writing to these addresses has no effect. During a read cycle, data from the dual-port SRAM will be read by processor A, if USE_FIFOS=0. Note that if the CFG_ROM parameter is greater than 0 and the ROMSEL_A input is high (logic 1), then reads from this address space will return data from the optional ROM; however, writes to this address space will still write to the dual-port RAM. If the CFG_ROM parameter is greater than 0 and the ROMSEL_A input is low (logic 0), then reads from this address space will return data read from the dual-port RAM. Note that each address in this range is 32-bit word aligned, therefore when reading from the ROM, keep in mind that the values specified in the ROM_ADDR_1 to ROM_ADDR_40 parameters will actually need to be multiplied by 4 (shifted to left by 2 bits) to access the desired ROM contents. For example, for processor A to read the contents of ROM address 255 (0xFF), it will need to set the ROMSEL_A input high and read from address 0x2000 + (0xFF<<2) = 0x23FC. If USE_FIFOS=1 and CFG_ROM=0, reading from these addresses has no effect (unknown logic values are returned and should be ignored).

Port B (AMBA 2 or AMBA 3 APB Slave) Register Interface

Processor B has access to the register address map listed in Table 4-12. Individual register descriptions are listed in Table 4-13 on page 27 to Table 4-19 on page 31. All registers are read/write unless otherwise noted. Any bits that are not listed in the descriptions of the registers will return logic 0 values when read by processor B, and will be ignored on writes from processor B.

Table 4-12 · Port B Register Address Map

PADDR[13:0]	Type	Register Name	Reset Value	Description
0x0000	R/W	INTCTRLB	0x00000000	Interrupt B control register.
0x0004	R	INTSTATB	0x00000000	Interrupt B status register (read-only).
0x0010 - 0x001C	R/W	B_FLAGS_B2A	0x00000000	32 possible flag logic bits that can be used to send interrupts from processor B to processor A, readable and setable from processor B.

Table 4-12 · Port B Register Address Map (continued)

PADDR[13:0]	Type	Register Name	Reset Value	Description
0x0020 - 0x002C	R/W	B_FLAGS_A2B	0x00000000	32 possible flag logic bits that can be used to send interrupts from processor A to processor B, readable and clearable from processor B.
0x0050	R	FIFORDA2B	Undefined	Read-only access to read message from processor A to processor B. This address is only used if USE_FIFOS=1.
0x0054	W	FIFOWDB2A	N/A	Write-only access to send message from processor B to processor A. This address is only used if USE_FIFOS=1.
0x2000 - 0x3FFC	R/W	DPRAMB	Undefined	Processor B access to port B of internal dual-port SRAM instance(s) or optional ROM. This address space is only used if USE_FIFOS=0.

Note: Type designations: "R" = read-only, "R/W" = read/write, "W" = write-only

Table 4-13 · Interrupt B Control (INTCTRLB) Register Bit Description

Bit	Name	Description
3	EN_B_FLAGS_A2B_OR	Enable B_FLAGS_A2B_OR bit: This bit controls whether the ORed version of the flag bits from processor A to processor B (B_FLAGS_A2B[31:0]) contributes to the INTERRUPT_B output or it is masked. If set to 1, the ORed flag contribution is enabled; otherwise, if 0, the ORed flag contribution is masked. The default value of 0 disables (masks) the contribution of the ORed flags to the INTERRUPT_B output.
2	EN_DPWRA	Enable DPWRA bit: This bit controls whether the DPWRA bit of the INTSTATB register contributes to the INTERRUPT_B output or it is masked. If set to 1, the DPWRA bit contribution is enabled; otherwise, if 0, the DPWRA bit is masked. The default value of 0 disables (masks) the contribution of the DPWRA bit to the INTERRUPT_B output.

Table 4-13 · Interrupt B Control (INTCTRLB) Register Bit Description (continued)

Bit	Name	Description
1	EN_FULL_B2A	Enable FULL_B2A bit: This bit controls whether the FULL_B2A bit of the INTSTATB register contributes to the INTERRUPT_B output or it is masked. If set to 1, the FULL_B2A bit contribution is enabled; otherwise, if 0, the FULL_B2A bit is masked. The default value of 0 disables (masks) the contribution of the FULL_B2A bit to the INTERRUPT_B output.
0	EN_MSG_A2B	Enable MSG_A2B bit: This bit controls whether the MSG_A2B bit of the INTSTATB register contributes to the INTERRUPT_B output or it is masked. If set to 1, the MSB_A2B bit contribution is enabled; otherwise, if 0, the MSG_A2B bit is masked. The default value of 0 disables (masks) the contribution of the MSG_A2B bit to the INTERRUPT_B output.

Table 4-14 · Interrupt B Status (INTSTATB) Register Bit Description

Bit	Name	Description
3	B_FLAGS_A2B_OR	ORed flag bits from processor A to processor B: If set to 1, this bit indicates that processor A has set one or more general purpose flags in the B_FLAGS_A2B[31:0] register to indicate to processor B that it must take some action, followed by clearing of these general purpose flags. This action is entirely application specific. This bit will not be cleared when reading this register, but rather when all bits of the B_FLAGS_A2B[31:0] register are cleared. If MSG_A2B_FLAGS=0, this bit will be permanently tied to logic 0.
2	DPWRA	Processor A has written to dual-port SRAM: If set to 1, this bit indicates that processor A has written data into the A port of the dual-port SRAM. This bit will be cleared when reading this register. If USE_FIFOS=1, this bit will be permanently tied to logic 0.
1	FULL_B2A	FIFO B->A is full: If set to 1, this bit indicates that the B->A FIFO (from processor B to processor A) is full. This bit will be cleared when at least one message has been read from the B->A FIFO by the A processor. If USE_FIFOS=0, this bit will be permanently tied to logic 0.
0	MSG_A2B	Message(s) available in FIFO A->B: If set to 1, this bit indicates that at least one message is available in the A->B FIFO (from processor A to processor B). This bit will be cleared when all messages have been read from the A->B FIFO. If USE_FIFOS=0, this bit will be permanently tied to logic 0.

Table 4-15 · Processor B to Processor A Flags (B_FLAGS_B2A) Register Bit Description

Bit	Name	Description
31:0	FLAGS	These 32 possible flag logic bits can be used to send interrupts from processor B to processor A. Processor B can set any of the 32 bits by writing a logic 1 in that bit position. For example, if the B_FLAGS_B2A[31:0] register value contains 0xFFAA0000 and processor B writes the value 0x00000055 to this register, the value will be changed to 0xFFAA0055 on subsequent reads, since writing logic 0 values to any of the bits is ignored. Note that the organization of this 32-bit register will differ depending on the value of the PORT_B_WIDTH parameter. If the PORT_B_WIDTH parameter is 8, the internal 32-bit register B_FLAGS_B2A[31:0] is read/write accessible by processor B in 4 groups of 8 bits, via the 4 word-aligned addresses from 0x0010 to 0x001C: B_FLAGS_B2A[7:0] accessible at address 0x0010, B_FLAGS_B2A[15:8] accessible at address 0x0014, B_FLAGS_B2A[23:16] accessible at address 0x0018, and B_FLAGS_B2A[31:24] accessible at address 0x001C. If the PORT_B_WIDTH parameter is 16, the internal 32-bit register B_FLAGS_B2A[31:0] is read/write accessible in 2 groups of 16 bits, via the 4 word-aligned addresses from 0x0010 to 0x001C: B_FLAGS_B2A[15:0] accessible at addresses 0x0010 and 0x0014, andB_FLAGS_B2A[31:16] accessible at addresses 0x0018 and 0x001C, If the PORT_B_WIDTH parameter is 32, the internal 32-bit register B_FLAGS_B2A[31:0] is read/write accessible as a single group of 32 bits at any of the 4 word-aligned addresses from 0x0010 to 0x001C.

Table 4-16 · Processor A to Processor B Flags (B_FLAGS_A2B) Register Bit Description

Bit	Name	Description
31:0	FLAGS	Message flag bits from processor A to processor B: These 32 possible flag logic bits can be used to send interrupts from processor A to processor B. Processor B can clear any of the 32 bits by writing a logic 1 in that bit position. For example, if the B_FLAGS_A2B[31:0] register value contains 0x00550000 and processor B writes the value 0xFF55FFFF to this register, the value will be changed to 0x00000000 on subsequent reads. Writing logic 0 values to any of the bits is ignored. Note that the organization of this 32-bit register will differ depending on the value of the PORT_B_WIDTH parameter. If the PORT_B_WIDTH parameter is 8, the internal 32-bit register B_FLAGS_A2B[31:0] is read/write accessible by processor B in 4 groups of 8 bits, via the 4 word-aligned addresses from 0x0020 to 0x002C: B_FLAGS_A2B[7:0] accessible at address 0x0020, B_FLAGS_A2B[15:8] accessible at address 0x0024, B_FLAGS_A2B[23:16] accessible at address 0x0028, and B_FLAGS_A2B[31:24] accessible at address 0x002C. If the PORT_B_WIDTH parameter is 16, the internal 32-bit register B_FLAGS_A2B[31:0] is read/write accessible in 2 groups of 16 bits, via the 4 word-aligned addresses from 0x0020 to 0x002C: B_FLAGS_A2B[15:0] accessible at addresses 0x0020 and 0x0024, and B_FLAGS_A2B[31:16] accessible at addresses 0x0028 and 0x002C, If the PORT_B_WIDTH parameter is 32, the internal 32-bit register B_FLAGS_A2B[31:0] is read/write accessible as a single group of 32 bits at any of the 4 word-aligned addresses from 0x0020 to 0x002C.

Table 4-17 · FIFO Read Data A->B (FIFORDA2B) Register Bit Description

Bit	Name	Description
n:0	DATA	FIFO read data (n = MBX_WIDTH-1): During a read cycle, this data will be read by processor B from the A->B FIFO (for sending message data from processor A to processor B) if USE_FIFOS=1. If USE_FIFOS=1, reading from this address has no effect, as unknown logic values are returned and should be ignored. This register is read-only. Note that if PORT_B_WIDTH is less than MBX_WIDTH, then the number of data bits accessible from processor B will be limited to the value of PORT_B_WIDTH rather than the value of MBX_WIDTH.

Table 4-18 · FIFO Write Data B->A (FIFOWDB2A) Register Bit Description

Bit	Name	Description
n:0	DATA	FIFO write data (n = MBX_WIDTH-1): During a write cycle, this data will be written from processor B into the B->A FIFO (for sending message data from processor B to processor A) if USE_FIFOS=1. If USE_FIFOS=0, writing to this address has no effect. This register is write-only. Note that if PORT_B_WIDTH is less than MBX_WIDTH, then the number of data bits accessible from processor B will be limited to the value of PORT_B_WIDTH rather than the value of MBX_WIDTH.

Table 4-19 · Dual-port SRAM B (DPRAMB) Register Bit Description

Bit	Name	Description
n:0	DATA	Dual-port SRAM data (n = MBX_WIDTH-1): During a write cycle, this data will be written from processor B into the dual-port SRAM if USE_FIFOS=0. Note that only every fourth 32-bit word aligned address is used: 0x2000, 0x2004, 0x2008, ..., 0x3FFC. If USE_FIFOS=1, writing to these addresses has no effect. During a read cycle, data from the dual-port SRAM will be read by processor B if USE_FIFOS=0. Note that if the CFG_ROM parameter is greater than 0 and the ROMSEL_B input is high (logic 1), then reads from this address space will return data from the optional ROM; however, writes to this address space will still write to the dual-port RAM. If the CFG_ROM parameter is greater than 0 and the ROMSEL_B input is low (logic 0), then reads from this address space will return data read from the dual-port RAM. Note that each address in this range is 32-bit word aligned, therefore when reading from the ROM, keep in mind that the values specified in the ROM_ADDR_1 to ROM_ADDR_40 parameters will actually need to be multiplied by 4 (shifted to left by 2 bits) to access the desired ROM contents. For example, for processor B to read the contents of ROM address 255 (0xFF), it will need to set the ROMSEL_B input high and read from address 0x2000 + (0xFF<<2) = 0x23FC. If USE_FIFOS=1 and CFG_ROM=0, reading from these addresses has no effect, as unknown logic values are returned and should be ignored. Note that if PORT_B_WIDTH is less than MBX_WIDTH, then the number of data bits accessible from processor B will be limited to the value of PORT_B_WIDTH rather than the value of MBX_WIDTH.

5 Testbench Operation and Modification

User Testbench

Included with the releases of CoreMBX is a user testbench that gives an example of how to use the core with two processors. The user testbench (Figure 5-1) instantiates two behavioral Actel DirectCore AMBA BFM modules to emulate using an AHB-Liter master on the AHB port of CoreMBX and an APB master on the APB port of CoreMBX.

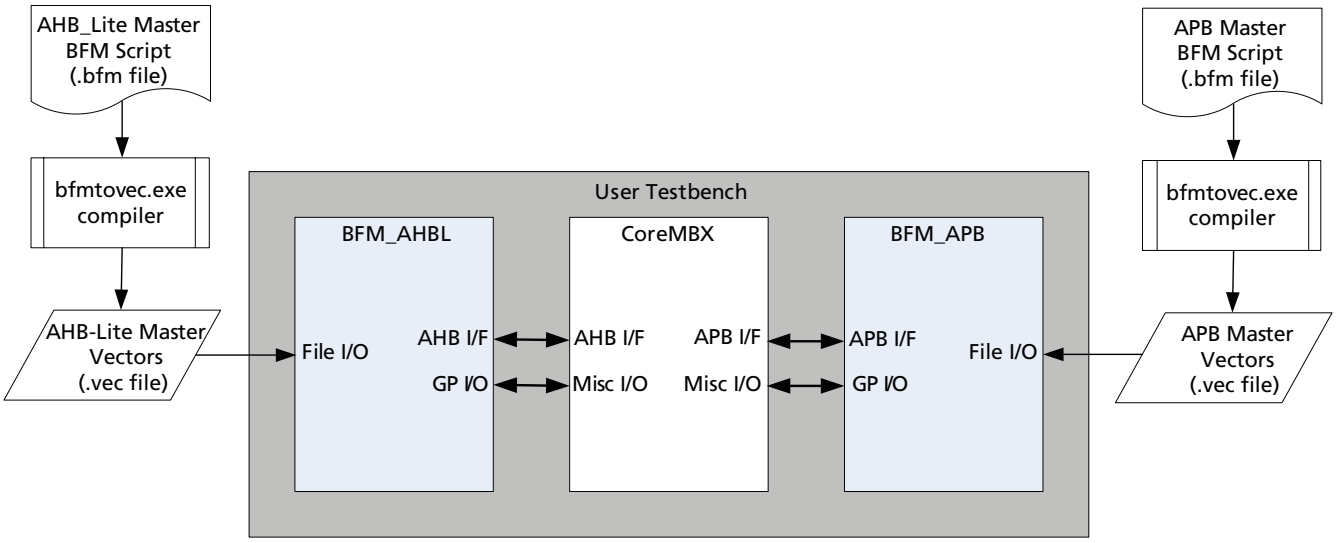

Figure 5-1 · CoreMBX User Testbench

Two BFM ASCII script source files (**.bfm*), with comments, are included in the directory *YourLiberoProjectDirectory/simulation*; where *YourLiberoProjectDirectory* represents the path to your Libero IDE project. The BFM source files (*corembx_usertb_ahb_master.bfm* and *corembx_usertb_apb_master.bfm*) are for controlling the AHB-Lite master and APB master processors respectively. These two BFM source files are automatically recompiled each time the simulation is invoked from Libero IDE by the *bfmtovec.exe* executable, if running on a Windows® platform, or by the *bfmtovec.lin* executable, if running on a Linux platform. The *output.vec* vector files, created by the bfmtovec executable, are read in by the BFM modules for simulation in Model*Sim*.

The source code for the user testbench and BFM scripts is available with the CoreMBX obfuscated and RTL releases. A compiled Model*Sim* simulation library containing the BFM modules is available with the CoreMBX obfuscated release. Obfuscated RTL versions of the BFM modules are available with the CoreMBX RTL release.

6 System Operation

This chapter provides various hints to ease the process of implementation and integration of CoreMBX into your own design.

Usage with Cortex-M1 and CoreABC

CoreMBX can be used with Cortex-M1, Actel's soft IP version of the popular ARM® microprocessor that has been optimized for the M1 Fusion flash-based FPGA devices, and with CoreABC. To create a design using Cortex-M1, CoreMBX, and CoreABC, you should use CoreConsole or SmartDesign. An example CoreConsole sub-system using Cortex-M1, CoreMBX, and CoreABC is shown in Figure 6-1. Please refer to the CoreConsole online help on how to create your Cortex-M1-based design using the CoreConsole IDP. Please refer to the Libero IDE online help for information on creating a Cortex-M1-based design using SmartDesign.

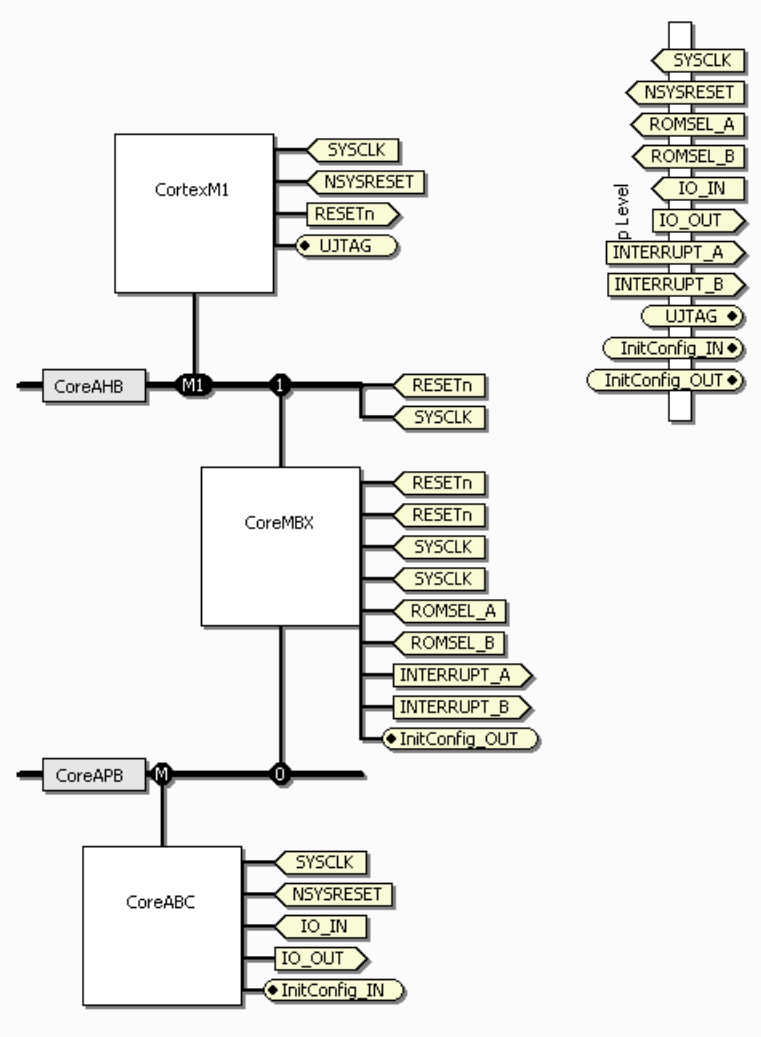

Figure 6-1 · Example System using CoreMP7 and CoreMBX

7 Ordering Information

Ordering Codes

CoreMBX can be ordered through your local Actel sales representative. It should be ordered using the following number scheme: CoreMBX-XX, where XX is listed in Table 7-1.

Table 7-1 · Ordering Codes

XX	Description
OM	RTL for Obfuscated RTL – multiple-use license
RM	RTL for RTL source – multiple-use license

Note: CoreMBX-OM is included free with a Libero IDE license

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

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

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at
www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 • USA*
Phone 650.318.4200 • Fax 650.318.4600 • Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Building 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • http://jp.actel.com

Actel Hong Kong *• Room 2107, China Resources Building • 26 Harbour Road • Wanchai • Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200138-0/10.08