HB0919

Handbook

CoreVectorBlox

November 2020

Microsemi

a Microchip company

Contents

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 2.0

Revision 2.0 was published in November 2020. The following is the list of changes in revision 2.0.

- The *Overview* section was updated
- *Device Utilization and Performance* table was updated
- Added *Interrupt Signals* section
- Updated *Smart Design* diagram

1.2 Revision 1.0

Revision 1.0 was published in June 2020. It is the first publication of this document.

2 Introduction

This handbook provides details about Microsemi® CoreVectorBlox Environment and how to use it.

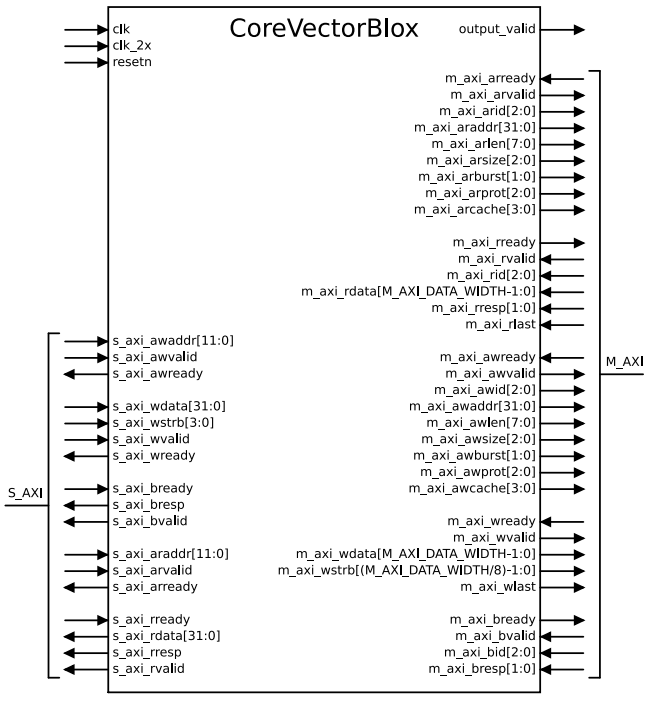
This document is used by Microsemi FPGA designers using Libero® System-on-Chip (SoC).

2.1 Overview

CoreVectorBlox provides a flexible neural network accelerator. It uses an overlay approach, where one instantiation can run different networks without needing to be resynthesized. A software toolkit called VectorBlox Accelerator Software Development Kit (SDK) compiles a neural network description from a supported framework (for example, TensorFlow, Caffe, and so on) into a Binary Large Object (BLOB) that is loaded into memory accessible by the CoreVectorBlox memory-mapped master. The CoreVectorBlox reads this BLOB and the network inputs from memory, processes the network, and places the result into an output buffer in memory. It can switch between multiple networks dynamically because of its overlay design. The overlay features a vector processor which can handle general vector layers and a convolutional accelerator, further accelerates Convolutional Neural Networks (CNNs). CoreVectorBlox efficiently supports most convolutional neural networks available today, such as ResNet, MobileNet, YOLO, and many more. Different configurations are available, allowing the user to scale the resource utilization to the required network performance.

The top-level interface diagram is shown in the following figure.

Figure 1 • I/O Signal Diagram



2.2 Features

The following are the key features of CoreVectorBlox.

- Multiple size configuration to trade-off performance for resource utilization.
- Overlay design allows multiple networks to run on the same core and even switch dynamically.
- Configurable width (64-bit to 256-bit) AXI4 memory master for data access.
- AXI4-Lite slave for control and status.
- Memory-based; reads inputs from and writes outputs to memory-mapped master.
- Internal vector processor which can process general neural-network layers.
- CNN accelerator for convolutional layers.

2.3 Core Versions

This handbook applies to CoreVectorBlox v1.0. The release notes provided with the core list known discrepancies between this handbook and the core release associated with the release notes.

2.4 Supported Families

- PolarFire®

2.5 Device Utilization and Performance

A summary of the implementation data is listed in the following table.

Table 1 • CoreVectorBlox Device Utilization and Performance

Configuration	Resource					Utilization		Performance	Peak GOPs
	LUT4	DFF	Math Blocks	uSRAM	LSRAM	Device	LUT4 %		
V250	35595	32127	104	103	47	MPF100T-1	33%	154 MHz	78
V500	61574	55630	204	380	84		57%	148 MHz	152
V1000	79338	77662	332	604	148	MPF200T-1	41%	138 MHz	284

Note:

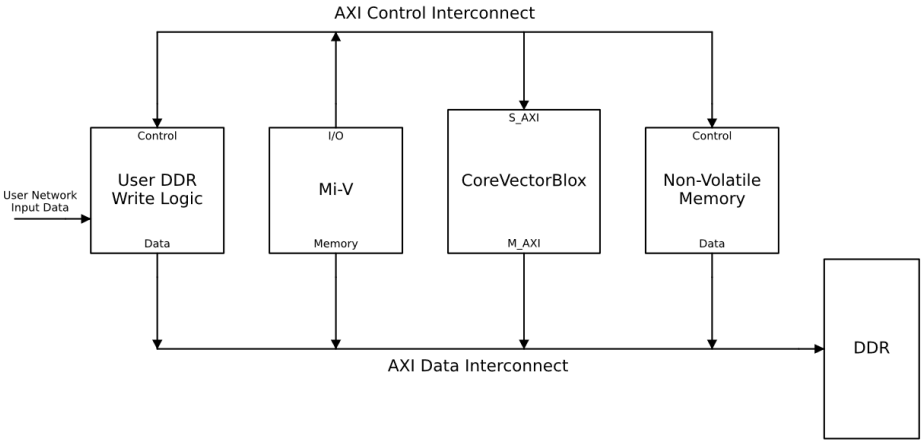
AXI master data width was set to widest possible for the configuration (128-bit for V250, 256-bit for V500/V1000). All performance data was obtained under commercial (COM) conditions.

3 Functional Description

3.1 System Level Overview

CoreVectorBlox processes neural networks residing in external memory. Its interfaces are an AXI4-Lite slave for setup and control and an AXI4 master for instruction and data memory. An example system of CoreVectorBlox usage is shown in the following figure.

Figure 2 • Example of System Level Block Diagram



The Mi-V soft processor is used to control the flow of data between components. At boot up, the firmware and network BLOBs (the network BLOBs were processed by the SDK and stored in non-volatile memory) are copied from non-volatile memory to DDR memory. Incoming data (for example, video frames from an image sensor) is written into DDR memory by user logic under the control of the Mi-V. When new data is available, the Mi-V instructs CoreVectorBlox to begin processing. CoreVectorBlox reads the weights and layer types from the Network BLOB and data from the network inputs from DDR memory, processes the network, and then writes the results to DDR memory. Finally, the Mi-V either directly reads the results or signals another module to consume them.

3.2 Memory Components

CoreVectorBlox requires the following components to be placed in memory accessible to its AXI4 master interface:

1. Firmware BLOB—Binary Large OBject (BLOB) common to all networks.
2. Network BLOB(s)—BLOBs produced by the VectorBlox Accelerator SDK for each network to run.
3. Network Inputs/Outputs—Network specific I/O for each run of a network.

The firmware BLOB is provided by Microchip and is common to all configurations of CoreVectorBlox. The firmware BLOB is included as part of the VectorBlox Accelerator SDK and contains instructions to parse Network BLOBs. Updated firmware can be released to target new network features. The firmware BLOB is loaded into external memory during bootup.

The Network BLOBs are compiled by the user to target their own networks. Microchip provides examples and tutorials in the VectorBlox Accelerator SDK. A Network BLOB contains information about the specific layer types in the network as well as weights and buffer space for activations. Each network has its own BLOB; multiple BLOBs can be present in memory at the same time and each time the CoreVectorBlox is

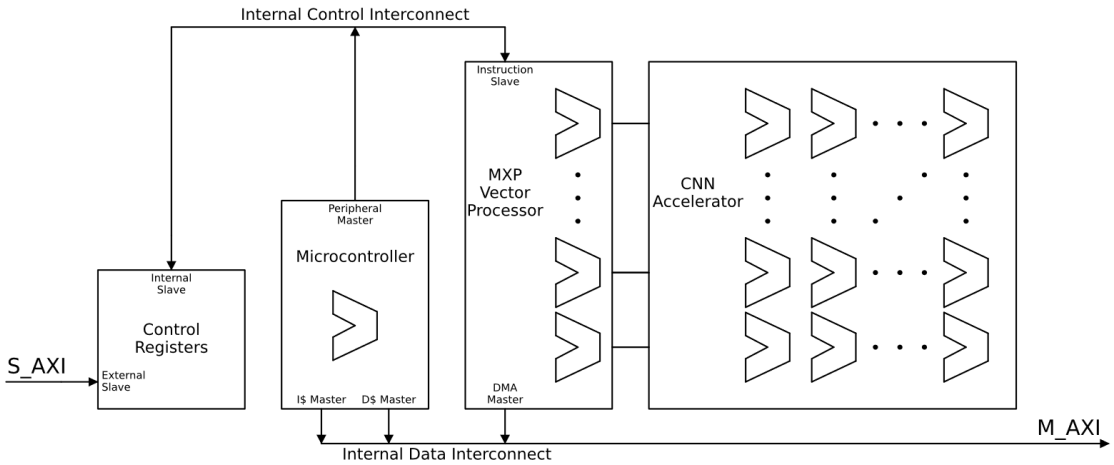
started, it can target a different BLOB. The user loads the Network BLOB(s) into external memory during bootup. The network inputs and outputs are specific to each network. For more information, see the *VectorBlox Accelerator SDK*.

3.3 Hardware Architecture

The following figure shows the primary blocks of CoreVectorBlox.

- Control Registers—Control, status, and error registers as well as addresses for BLOBs and I/O.
- Microcontroller—Parses network structures from BLOBs and controls the vector processor.
- MXP Vector Processor—Processes general neural network layers.
- CNN Accelerator—Processes convolutional network layers.

Figure 3 • CoreVectorBlox Block Diagram



The control registers have an AXI4-Lite slave interface for external logic (for example, a Mi-V soft processor) to control the state of CoreVectorBlox. They can detect and report simple errors, such as invalid BLOB addresses, but mostly are used to initialize and communicate with the microcontroller. Once initialized, the microcontroller can set status and error conditions in the control registers.

The microcontroller is a simple in-order RISC-V soft processor (internal to CoreVectorBlox and different from the Mi-V processor mentioned in the system description). It is used to read/set control registers, parse network BLOBs, and issue instructions to the vector processor for processing network layers. Its instruction memory and working data memory are contained in the Firmware BLOB, and it has instruction and data caches to reduce traffic over the data memory master interface.

The MXP Vector Processor is a soft vector processor, which can perform data parallel operations on long vectors of data. It uses an internal scratchpad as working memory and a variable-width DMA controller for transferring data between the scratchpad and external memory. The CoreVectorBlox size configuration (see *Configuration Options*) configures the number of parallel ALUs and scratchpad memory size of the MXP. The microcontroller issues instructions to the MXP to read network data into its scratchpad, perform operations such as addition and multiplication on the data, and store the data back to memory.

The CNN Accelerator is an array of processing elements (PEs) that consists of a multiply-accumulate unit and small accumulation RAM. Data is passed from the MXP scratchpad to the PE array, where it is processed over multiple cycles before being written back to the MXP scratchpad. The PEs are laid out in a 2D grid, which can take advantage of the multiple levels of parallelism in many neural network layers. This achieves a higher level of performance than the MXP alone, especially in convolutional neural networks and achieves a higher degree of parallelism than the MXP, when processing the convolutional layers.

3.4 Configuration Options

CoreVectorBlox can be configured to one of the various size configurations, based on size configuration performance. Detailed resource usage and device utilization by size configuration can be found in *Device Utilization and Performance*. The following table lists the details of the configuration changes the vector processor and CNN accelerator. More benchmarks along with power and memory bandwidth usage are available in the CoreVectorBlox SDK.

Table 2 • Processor Size Configuration Details

Size Configuration	Vector Processor Width	Vector Scratchpad	CNN Accelerator Array Size	Peak CNN Throughput	Resnet50 Performance
V250	128-bit	64 kB	16x16	78 GOPs	4.2 FPS
V500	256-bit	128 kB	16x32	146 GOPs	7.8 FPS
V1000	256-bit	256 kB	32x32	264 GOPs	13.6 FPS

Note:

Network latency is 1/FPS; networks are run with a batch size of 1.

4 Operation

4.1 Memory Map

The following table lists the control slave memory map. All registers are 32-bit in width. Register access is specified as a combination of readable (R), writable (W), write-to-set (WS), and write-to-clear (WC). Write-to-set (WS) bits are set to '1' by a write to the specified register that has a '1' in the WS bit, but can only be cleared by an internal logic (a write to the specified register with a '0' in the WS bit has no effect). Write-to-clear (WC) bits are cleared to '0' by a write to the specified register with a '1' in the WC bit (a write to the specified register with a '0' in the WC bit has no effect). Write-to-clear (WC) registers are cleared by any write to the specified register.

Table 3 • Control Slave Memory Map

Address	Name	Description			Access
		Bit 0 = LSB	Function	Reset Value	
0x00	Control Register	0 Soft Reset	Write 1 to soft reset. All other control register bits are ignored when soft reset is activated. The error register is cleared on soft reset as well. Soft reset expects to fulfill memory transactions on all interfaces. If the modules connected to S_AXI or M_AXI are placed into reset and may not have correctly fulfilled outstanding AXI transactions, then a hard reset (through the resetn pin) must be performed to ensure that the S_AXI and M_AXI interfaces come up correctly. 1→0 transition triggers the load of instruction memory; the Instruction BLOB Address must be set before clearing this bit. Furthermore, the instruction BLOB must be reloaded every time a soft or hard reset is performed; failure to do so issues an error.	1	R/W
		1 Start	Write 1 to set. Setting this bit causes the CoreVectorBlox to start processing a network. It begins fetching the network BLOB from the location specified in the Network Model Address register and decode the BLOB to determine how to proceed. The Network Model Address and Network I/O Address registers must be set before setting this bit. Once cleared, the hardware starts processing the current network (concurrently with setting the Running bit). While set, it is an error to Start again or to write to the Network Model Address or Network I/O Address registers.	0	R/WS
		2 Running	Set during processing (concurrently with clearing the Start bit).	0	R

Address	Name	Description			Access
		Bit 0 = LSB	Function	Reset Value	
			Cleared concurrently to setting the 'Output Valid' bit.		
		3 Output Valid	Write 1 to clear. Set once the network outputs are valid. It must be cleared once per network invocation. An invocation is started by writing the Start bit and is ended by clearing the Output Valid bit. The Output Valid bit is not cleared before writing the Start bit for the next network invocation, but until it is cleared, the subsequent network does not finish (the Running bit stays set). The Output Valid bit sets once and must be cleared once per setting of the Start bit. If the control slave does not clear this bit, subsequent network invocations will not finish. While this bit is clear, it is an error to write to this bit.	0	R/WC
		4 Error	Indicates the contents of the Error Register are valid and must be examined.	0	R
		31:5	Reserved	0	R
0x04	Error Register	Write any value to clear to zero, which also clears the Error bit of the Control Register. Errors will cause a soft reset, which is identical to setting the Soft Reset bit of the Control Register except that the Error bit and this register are not cleared. See *Table 5 • Error Codes* for more information.			R/WC
0x08	Firmware BLOB Address	Address pointing to firmware BLOB to which contains instruction memory. Must be be aligned to a 2 MB boundary and be greater than or equal to 0x0020_0000 when clearing the Soft Reset bit or an error will be raised. Writing while not in Soft Reset raises an error.			R/W
0x10	Network Model Address	Address Pointing to Model Blob. Must be aligned to an 8 byte boundary and be greater than or equal to 0x0020_0000 when setting the 'Start' bit or an error will be raised. Writing while the Start bit is set raises an error.			R/W
0x18	Network I/O Address	Address pointing to the I/O data structure for the network. Must be aligned to an 8 byte boundary and be greater than or equal to 0x0020_0000 when setting the 'Start' bit or an error will be raised. Writing while the Start bit is set raises an error.			R/W
0x28	Version	See the following table for version information.			R

Table 4 • Version

Bits	Name	Function
7:0	Product ID	Reserved; Reads 0 for CoreVectorBlox.
15:8	Size Configuration	Size configuration:

Bits	Name	Function
		0=>V250, 1=>V500 , 2=>V1000
19:16	Reserved	Reserved core version information
27:20	Minor Version	Minor version number (For example, 0x05 for CoreVectorBlox 1.5).
31:28	Major Version	Major version number (For example, 0x01 for CoreVectorBlox 1.5).

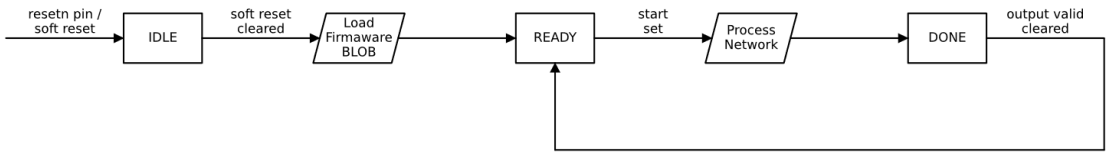
Table 5 • Error Codes

Value	Code	Description
1	INVALID_INSTRUCTION_ADDRESS	The instruction address is within the reserved memory range (first 2 MB).
2	INSTRUCTION_ADDRESS_NOT_READY	The Instruction BLOB Address was written while not in reset.
3	START_NOT_CLEAR	The Start bit of the Control Register, the Network Model Address, or the Network I/O Address were written while the Start bit of the Control Register was still set.
4	OUTPUT_VALID_NOT_SET	The Output Valid bit of the Control register was cleared when it was not set.
5	INSTRUCTION_BLOB_VERSION_MISMATCH	The Instruction BLOB was built for the wrong version of the core.
6	INVALID_NETWORK_ADDRESS	The network model or I/O addresses are within the reserved memory range (first 2 MB).
7	NETWORK_BLOB_INVALID	The network BLOB has an invalid format.
8	NETWORK_BLOB_VERSION_MISMATCH	The network BLOB was built for the wrong version of the Instruction BLOB.
9	NETWORK_BLOB_PRESET_MISMATCH	The network BLOB was built for a different preset configuration than the one in use.
10	INSTRUCTION_BLOB_STALE	The Instruction BLOB was not reloaded between resets of the accelerator.

4.2 Network Processing

The following figure shows the flow of processing networks.

Figure 4 • Network Processing Flow



A C API is provided in the CoreVectorBlox SDK.

When coming out of reset, CoreVectorBlox is held in an IDLE state. Once the Firmware BLOB is present in memory, the Firmware BLOB address can be set, and then the core taken out of soft reset. The core starts executing the firmware instructions and initializes, at which point it is in the READY state and can process networks.

Processing a network is started by setting the Network BLOB address and then setting the ‘start’ bit of the control register. CoreVectorBlox will then begin parsing the Network BLOB and read in the inputs as it starts processing. Exact steps of processing depend on the Network BLOB. During processing, the memory master may also access temporary buffer memory. The temporary buffers are pre-allocated inside the Network BLOB; CoreVectorBlox only accesses memory inside the Firmware BLOB, Network BLOB, and Network Input and Output buffers.

When network processing completes, the ‘Output Valid’ bit of the control register is set and the network outputs can be read. The next network run might start as soon as the current run is finished.

At any point, CoreVectorBlox can be put back into the Soft Reset state by setting the ‘soft reset’ bit of the control register. Soft reset attempts to finish any outstanding AXI transactions on the master and slave interfaces, as opposed to a hard reset using the `resetn` pin, which will immediately cease all transactions and reset those interfaces. After either kind of reset, the Firmware BLOB must be reloaded as it contains volatile data that is modified during processing. For more details on signals, see *Memory Map*.

5 CoreVectorBlox

5.1 Generics

Customers can define the generics listed in the following table as required in the source code.

Table 6 • CoreVectorBlox Generics

Generic	Default Setting	Valid Values	Description
Size Configuration	V1000	[V250, V500, V1000]	Size Configuration; see *Configuration Options* table for details.
M_AXI_DATA_WIDTH	256	[64, 128, 256]	AXI4 Data Master data width in bits. V250 only supports 64 bits to 128 bits, while V500 and V1000 support 64 bits to 256 bits.

6 Interface Description

The port signals for CoreVectorBlox are defined in the following tables and are also described in *Overview*.

6.1 Clocks and Resets

The clocks and the resets are listed in the following table.

Table 7 • Clocks and Resets

Signal	Function	I/O	Description
clk	Clock	Input	System clock. The control slave and data master are synchronous to this clock.
clk_2x	2X Clock	Input	Double-frequency clock. Clocks MathBlocks and LSRAM resources at twice the frequency of the system clock. It must be synchronous to and in phase with clk. See the following notes.
resetn	Reset	Input	System reset (active low). Resets all core functions as well as the control slave and data master.

Note:

1. To minimize skew between clk and clk_2x, the clocks must be created from the same CCC on PolarFire devices. Additionally, the clock outputs are paired. Either the OUT0 and OUT1 or the OUT2 and OUT3 pair must be used, but not one output from each pair.
2. The Libero Place and Route tool must be configured with the 'Repair Minimum Delay Violations' option selected to repair any hold violations caused by skew between the two clocks.

6.2 Control Slave Signals

The Control Slave is an AXI4-Lite compliant interface with memory map described in *Memory Map*. It is synchronous to clk and reset by `resetn` pin. Its signals are listed in the following table.

Table 8 • Control Slave Signals

Signal	Function	I/O	Description
s_axi_awaddr	Write Address	Input	AXI4-Lite slave write address.
s_axi_awvalid	Write Address Valid	Input	AXI4-Lite slave write address valid.
s_axi_awready	Write Address Ready	Output	AXI4-Lite slave write address ready.
s_axi_wdata	Write Data	Input	AXI4-Lite slave write data.
s_axi_wstrb	Write Data Strobe	Input	AXI4-Lite slave write data strobe (byte enable). CoreVectorBlox expects all write strobe signals to be all high or all low; partial register

Signal	Function	I/O	Description
			writes results in undefined results.
s_axi_wvalid	Write Data Valid	Input	AXI4-Lite slave write data valid.
s_axi_wready	Write Data Ready	Output	AXI4-Lite slave write data ready.
s_axi_bready	Write Response Ready	Input	AXI4-Lite slave write response ready.
s_axi_bresp	Write Response	Output	AXI4-Lite slave write response code.
s_axi_bvalid	Write Response Valid	Output	AXI4-Lite slave write response valid.
s_axi_araddr	Read Address	Input	AXI4-Lite slave read address.
s_axi_arvalid	Read Address Valid	Input	AXI4-Lite slave read address valid.
s_axi_arready	Read Address Ready	Output	AXI4-Lite slave read address ready.
s_axi_rready	Read Data Ready	Input	AXI4-Lite slave read data ready.
s_axi_rdata	Read Data	Output	AXI4-Lite slave read data.
s_axi_rresp	Read Data Response	Output	AXI4-Lite slave read data response code.
s_axi_rvalid	Read Data Valid	Output	AXI4-Lite slave read data valid.

Note:

1. All control slave signals are synchronous to clk.
2. The control slave is reset by `resetn` pin.

6.3 Data Master Signals

The Data Master is an AXI4 compliant interface. It is synchronous to clk and reset by `resetn` pin. Its signals are listed in the following table.

Table 9 • Data Master Signals

Signal	Function	I/O	Description
m_axi_arready	Read Address Ready	Input	AXI4 master read address ready.
m_axi_arvalid	Read Address Valid	Output	AXI4 master read address valid.
m_axi_arid	Read Address ID	Output	AXI4 master read address ID.

Signal	Function	I/O	Description
			CoreVectorBlox uses multiple IDs; these must be propagated correctly through any interconnect attached to the Data Master.
m_axi_araddr	Read Address	Output	AXI4 master read address.
m_axi_arlen	Read Length	Output	AXI4 master read length (beats per burst minus 1).
m_axi_arsize	Read Size	Output	AXI4 master read size. CoreVectorBlox does not issue narrow reads; this will be fixed to the data width size.
m_axi_arburst	Read Burst Type	Output	AXI4 master read burst type. CoreVectorBlox only issues incrementing bursts.
m_axi_arprot	Read Protection	Output	AXI4 master read protection. CoreVectorBlox only issues unprivileged, secure data acceses.
m_axi_arcache	Read Transaction Attributes	Output	AXI4 master read transaction attributes. CoreVectorBlox issues modifiable and bufferable transactions. It does not set allocation bits and assumes that transactions can be read from memory in systems with caches.
m_axi_rready	Read Data Ready	Output	AXI4 master read data ready.
m_axi_rvalid	Read Data Valid	Input	AXI4 master read data
m_axi_rid	Read Data ID	Input	AXI4 master read data ID. CoreVectorBlox uses multiple IDs.
m_axi_rdata	Read Data	Input	AXI4 master read data.
m_axi_rresp	Read Data Response	Input	AXI4 master read data response code.
m_axi_rlast	Read Data Last	Input	AXI4 master read data last (end of burst).
m_axi_awready	Write Address Ready	Input	AXI4 master write address ready.
m_axi_awvalid	Write Address Valid	Output	AXI4 master write address valid.
m_axi_awid	Write Address ID	Output	AXI4 master write address ID. CoreVectorBlox uses multiple IDs; these must be propagated correctly through any interconnect attached to the Data Master.
m_axi_awaddr	Write Address	Output	AXI4 master write address.
m_axi_awlen	Write Length	Output	AXI4 master write length (beats per burst minus 1).
m_axi_awsize	Write Size	Output	AXI4 master write size. CoreVectorBlox does not issue narrow writes; this will be fixed to the data width size.

Signal	Function	I/O	Description
m_axi_awburst	Write Burst Type	Output	AXI4 master write burst type. CoreVectorBlox only issues incrementing bursts.
m_axi_awprot	Write Protection	Output	AXI4 master write protection. CoreVectorBlox only issues unprivileged, secure data acceses.
m_axi_awcache	Write Transaction Attributes	Output	AXI4 master write transaction attributes. CoreVectorBlox issues modifiable, bufferable transactions. It does not set allocation bits.
m_axi_wready	Write Data Ready	Input	AXI4 master write data ready.
m_axi_wvalid	Write Data Valid	Output	AXI4 master write data valid.
m_axi_wdata	Write Data	Output	AXI4 master write data.
m_axi_wstrb	Write Data Strobe	Output	AXI4 master write data strobe (byte enable).
m_axi_wlast	Write Data Last	Output	AXI4 master write last (end of burst).
m_axi_bready	Write Response Ready	Output	AXI4 master write response ready.
m_axi_bvalid	Write Response Valid	Input	AXI4 master write response valid.
m_axi_bid	Write Response ID	Input	AXI4 master write response ID.
m_axi_bresp	Write Response Code	Input	AXI4 master write response code.

Note:

1. All data master signals are synchronous to clk.
2. The data master is reset by `resetn` pin.

6.4 Interrupt Signals

The following table lists the interrupt signals.

Table 10 • Interrupt Signals

Signal	Function	I/O	Description
output_valid	Interrupt Output	Output	Mirrors the Output Valid bit in the Control Register (see *Memory Map*) for use as an interrupt for needed systems.

7 Tool Flows

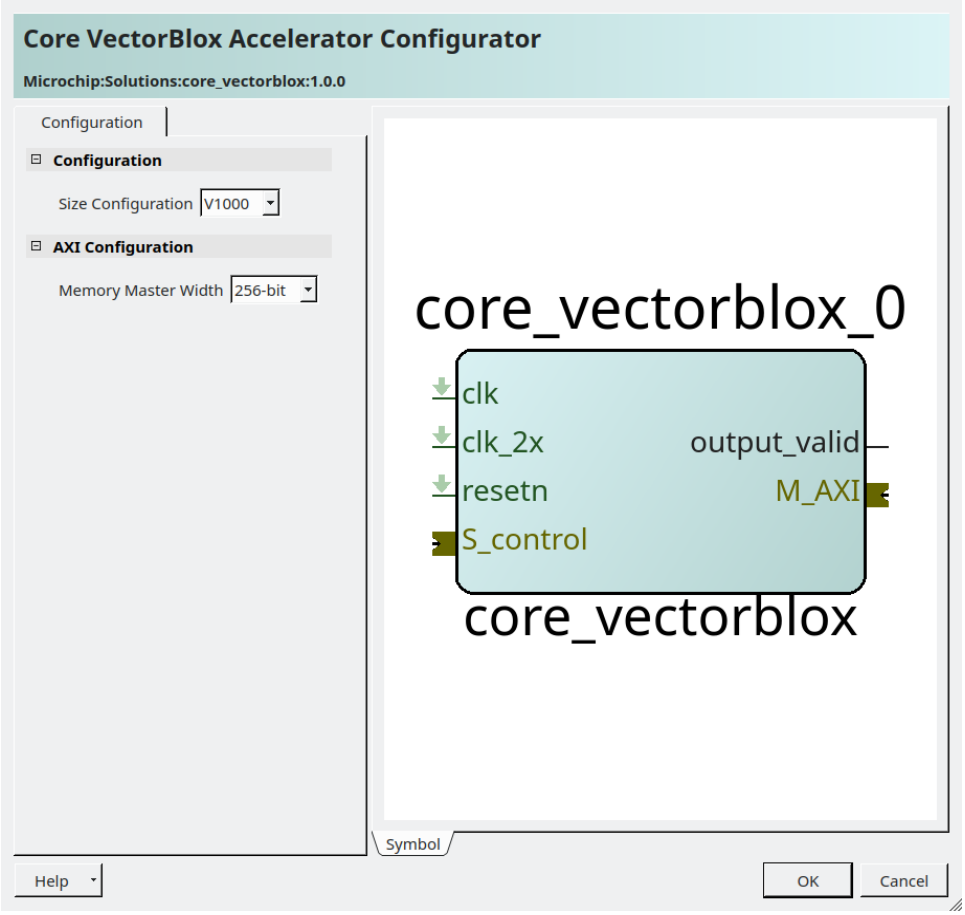
7.1 Licenses

CoreVectorBlox is licensed using encrypted RTL.

7.2 Smart Design

The following figure shows the core configured using the configuration GUI within SmartDesign.

Figure 5 • CoreVectorBlox Configurator within SmartDesign with V1000 Configuration



7.3 Simulation

CoreVectorBlox can be functionally simulated as well as simulated at the RTL level. For functional simulation, see the SDK. Functional simulation is the preferred way of verifying network functionality and accuracy.

The RTL can be simulated as part of a higher level design. The user has to load the Firmware and Instruction BLOBs into memory attached to the Data Master port. See the documentation for the specific memory interface used. The user must also write to the control registers, either using state machine logic or by instantiating a control processor, such as a Mi-V and loads a program for it to set the control registers.

7.4 Synthesis

Set the design root appropriately and click the Synthesis icon in Libero. To perform synthesis, right-click, and select Run. CoreVectorBlox requires no special synthesis settings.

7.5 Place and Route

The ‘Repair Minimum Delay Violations’ option must be turned on in the Libero Place and Route tool to fix any hold violations between clk and clk_2x caused by clock skew through the fabric (see *Clocks and Resets*).

Set the design route appropriately and run Synthesis. Click the Place and Route icon in Libero to invoke the Designer software.

Microsemi
2355 W. Chandler Blvd.
Chandler, AZ 85224 USA

Within the USA: +1 (480) 792-7200
Fax: +1 (480) 792-7277

Microsemi Corporation, a subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), and its corporate affiliates are leading providers of smart, connected and secure embedded control solutions. Their easy-to-use development tools and comprehensive product portfolio enable customers to create optimal designs which reduce risk while lowering total system cost and time to market. These solutions serve more than 120,000 customers across the industrial, automotive, consumer, aerospace and defense, communications and computing markets. Headquartered in Chandler, Arizona, the company offers outstanding technical support along with dependable delivery and quality. Learn more at **www.microsemi.com**.

50200919