VectorBlox MXP Quickstart for Intel

Contents

1 Before You Begin

If you do not have a MXP license, or are only intererested in software development, skip over the hardware specific sections.

2 Installation

This guide describes installing VectorBlox MXP to run with Qsys and the Intel Nios II Embedded Design Suite (EDS).

2.1 Prerequisites

Before you begin, make sure you have:

- Intel Quartus Prime, version 17.1. Download it from Intel. The VectorBlox MXP will likely work with older or newer versions of Quartus

- **For releases containing the SDK and hardware IP:** The VectorBlox MXP Software Development Kit (SDK) and encrypted MXP IP core , provided as a `.zip` file.

- **For releases containing only the SDK:** The VectorBlox MXP Software Development Kit (SDK), provided as a `.zip` file.

- One of the supported development boards, if you want to use one of the pre-built FPGA bitstreams included in the SDK.

2.2 Installing

- Install Intel Quartus Prime software according to Intel's instructions.

  http://www.altera.com/download/dnl-index.jsp

  Note the pathname where these tools are installed. For example, under Windows, the default install location for version 17.1 is `C:\intelFPGA\17.1\quartus` and the environment variable `QUARTUS_ROOTDIR` will be set to this location.

- Follow Altera's instructions for installing the USB Blaster device driver.

  http://www.altera.com/download/drivers/dri-index.html

- Copy vectorblox/repository/altera/mxp into `QUARTUS_ROOTDIR/../ip/vectorblox/`. You may have to create that directory.

To install the license file (provided separately) for the hardware IP, please refer to the Intel FPGA Software Installation and Licensing.

3 Hardware

A VectorBlox MXP system always uses a host CPU. This host CPU provides the MXP with instructions, and run the unaccelerated portions of your C/C++ application.

This section describes how to instantiate the VectorBlox MXP processor into a Qsys system. Some familiarity with Qsys is assumed; please refer to the *System Design with Qsys* section of the Quartus II Handbook for further details on Qsys.

You must ensure that the VectorBlox components are in the Qsys IP Search Path. If you extracted the Vectorblox MXP distribution into `QUARTUS_ROOTDIR/../ip` or into the `ip` subdirectory of your Quartus project, Qsys will be able find the VectorBlox components automatically. Otherwise, you can add a directory to the IP Search Path in Qsys; from the **Tools** menu, open the **Options** dialog box.
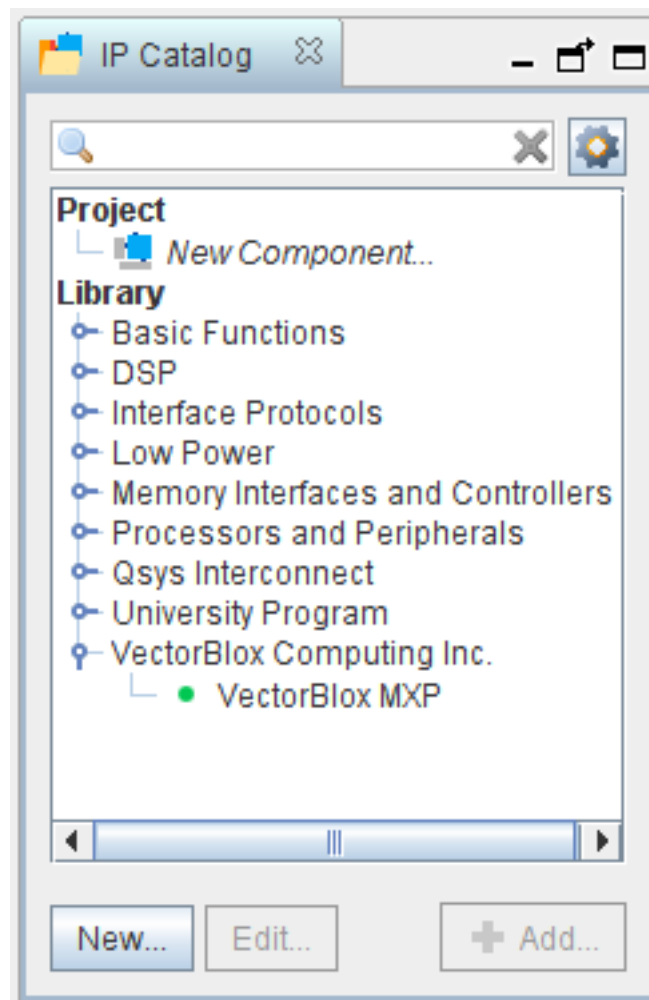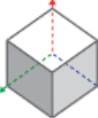

Figure 1: Qsys Component Library

The figure above shows the VectorBlox components in the Qsys IP Catalog.

3.1 VectorBlox MXP Qsys Parameters

To add a VectorBlox MXP instance to your Qsys system, double-click on **VectorBlox MXP**. Qsys will open the parameter editor shown in the figure. The parameters are described below.

System: vblox1   Path: vbx1

VectorBlox MXP
vectorblox_mxp

Details

Parameters

Number of Vector Lanes:	16 ▼
Number of Memory Lanes (DMA Bus Width):	1 ▼ Words
Number of Slave Memory Lanes:	1 ▼ Words
Maximum Burst Size in Beats:	8 ▼ Beats
Scratchpad Size:	64 kBytes
Instruction Port Type:	Nios Custom Instruction ▼
Multiplier Performance:	Byte ▼
Maximum # of waves for masked instructions:	256 ▼

Vector Custom Instruction (VCI) Options

Vector Custom Instruction Ports:	1 ▼

VCI 0

Unique ID:	0xffffffff
Lanes:	1
Opcode Start:	0 ▼
Internal Functions (Opcodes):	1 ▼
Function 0 Pipeline Depth:	0
☐ Modifies Destination Address	

Fixed-Point Multiply Format

☑ Fixed Point Support

Number of fractional bits in a word:	16 bits
Number of fractional bits in a halfword:	15 bits
Number of fractional bits in a byte:	4 bits

32-bit fixed-point format:	Q16.16
16-bit fixed-point format:	Q1.15
8-bit fixed-point format:	Q4.4

Derived Avalon Parameters

DMA Master Data Bus Width:	32 bits
DMA Master Burst Size:	32 bytes
DMA Master Burstcount Width:	4 bits
Scratchpad Slave Address Width:	14 bits

Figure 2: MXP Configuration Window

Number of Vector Lanes

The number of 32-bit vector lanes. This must be a power of 2. Each vector lane requires about the same amount of resources as the Nios II/f host CPU.

Number of Memory Lanes

The data bus width of the MXP DMA Engine's Avalon-MM master interface expressed in terms of 32-bit lanes. The number of memory lanes must be a power of two and no larger than the number of vector lanes. For best performance, the width of this interface should match the width of the Avalon slave port of the external memory controller.

Maximum Burst Size in Beats

The maximum number of beats per burst issued by the DMA Engine's Avalon-MM master interface. A beat is a clock cycle in which data is transferred between a master and slave interface. Using a larger value can improve the bandwidth between the MXP and slave devices. However, a very large value could potentially starve other masters (such as a Nios II/f host) from accessing a slave. Likewise, very small values will divide long transfers into very short bursts, losing potential bandwidth during the gaps.

Scratchpad Size

The Scratchpad RAM size in kilobytes. VectorBlox recommends 4KB per vector lane for most applications.

Instruction Port Type

The MXP can be controlled using several different instruction ports. To connect an ARM host processor to the MXP use the AXI Memory Mapped Port. To connect A NIOS II host processor to the MXP use the Nios Custom Instruction Port. The Vector Coprocessor Port is for connecting the MXP to the VectoBlox ORCA, a RISC-V based processor.

Multiplier Performance

This sets the minimum multiplier size. This can be used to reduce the number of hard multipliers used, at the cost of performance. If set to *Byte*, then byte, halfword, and word multipliers are instantiated and multiplication of any element size runs at full speed. If set to *Halfword*, only word and halfword multipliers are instantiated; byte-width multiplication will be executed with the halfword multiplier and run at half speed. If set to *Word*, only word multipliers are instantiated; halfword-width multiplication will run at half speed and byte-width multiplication will run at quarter speed.

Maximum number of waves for masked instructions

The maximum vector length that a masked instuction can be is calculated from this value. The Number of Vector Lanes multiplied by this value gives the maximum vector length for a word instruction. Multiply that

product by 2 and 4 to get the max vector length for halfword and byte instructions respectively.

Fixed-Point Multiply Format

These parameters affect the fixed-point multiply operation. They specify the number of least-significant bits that will be used to represent the fractional part of 32-bit, 16-bit, and 8-bit fixed-point numbers.

The fixed-point formats are also displayed in Q notation, where the first number specifies the number of integer bits and the second number specifies the number of fractional bits.

Fixed-point formats with no integer bits (Q0.X) are not allowed. Multiply high (VMULHI) is equivalent to fixed-point multiply for these formats and should be used instead.

Below the editable parameters are the Derived Avalon Parameters, which show the inferred widths of various Avalon ports:

DMA Master Data Bus Width

The data bus width, in bits, of the DMA Engine's Avalon-MM master interface. This is derived from the number of memory lanes.

DMA Master Burst Size

The Avalon-MM master interface's maximum burst size in bytes, as determined by the memory bus width and the maximum number of beats per burst.

DMA Master Burstcount Width

The width of the Avalon-MM master interface's `burstcount` output, as determined by the maximum burst size and the memory bus width.
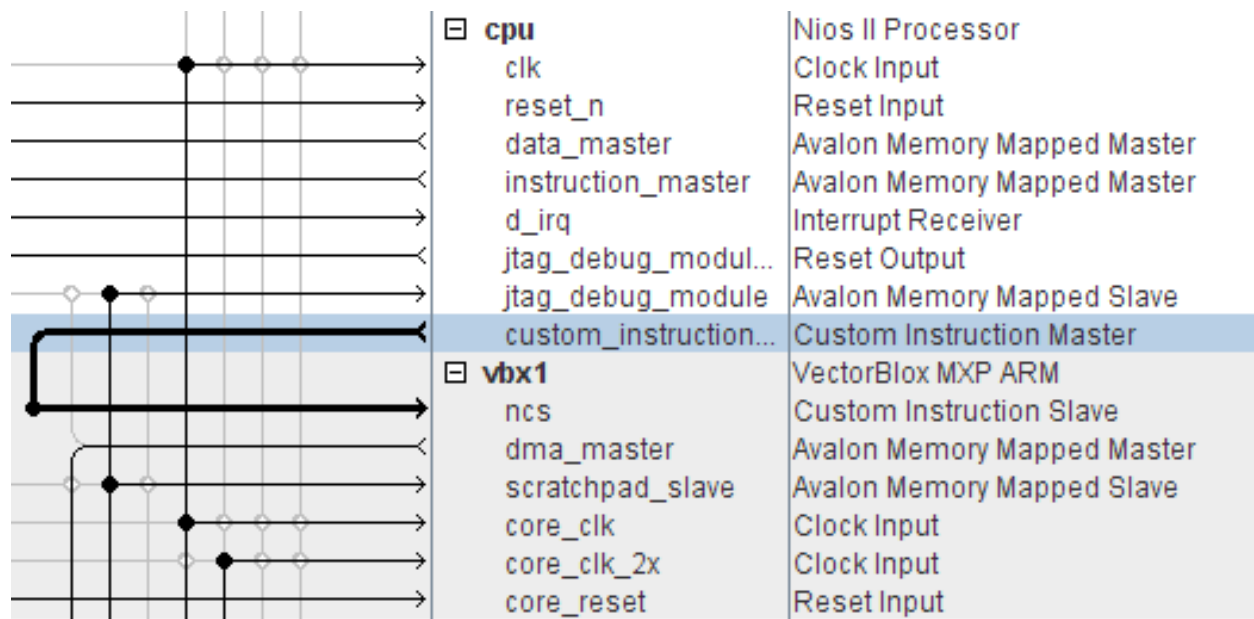
Slave Address Width

The MXP's Avalon-MM slave address width as determined by the scratchpad size.

3.2   Qsys Connections

This section describes the MXP processor's interfaces, shown in the figure below.

ncs

The MXP connects to the Nios II CPU through through a custom instruction port Make sure the Opcode base is set to 0.

⊟ **cpu**		Nios II Processor
	clk	Clock Input
	reset_n	Reset Input
	data_master	Avalon Memory Mapped Master
	instruction_master	Avalon Memory Mapped Master
	d_irq	Interrupt Receiver
	jtag_debug_modul...	Reset Output
	jtag_debug_module	Avalon Memory Mapped Slave
	custom_instruction...	Custom Instruction Master
⊟ **vbx1**		VectorBlox MXP ARM
	ncs	Custom Instruction Slave
	dma_master	Avalon Memory Mapped Master
	scratchpad_slave	Avalon Memory Mapped Slave
	core_clk	Clock Input
	core_clk_2x	Clock Input
	core_reset	Reset Input

axi_instr_slave

When Using an ARM as a host processor, connect the MXP to the processor via the axi memory mapped slave.

dma_master

This is the MXP DMA Engine's interface to external memory. The data bus width is determined by the number of memory lanes selected in the MXP Parameter Editor.

scratchpad_slave

This slave interface provides external access to the MXP's scratchpad memory. The address width is determined by the scratchpad size.

core_clk

This is the main MXP clock. The MXP's Avalon Memory Mapped Master and Slave interfaces run synchronously to **core_clk**. **core_clk** has to be synchronous to the Nios II CPU clock.

core_clk_2x

This clock must be double the frequency of **core_clk** and must be synchronous to **core_clk**. It should be generated from the same PLL that provides **core_clk**.

core_reset

> This is the active high reset input. Assertion and deassertion must be synchronous to **core_clk** and **core_clk_2x**.

3.3   Qsys System Generation

Once all of the components in your Qsys system have their interfaces connected correctly, you can proceed to Qsys generation by clicking on the *Generation* tab, setting options, then clicking *Generate*.

To generate synthesizable HDL, check ***Create HDL design files for synthesis*** in the Synthesis subsection.

To generate a simulation model, you must have a VectorBlox license that supports IP Functional Simulation (IPFS) model generation in either Verilog or VHDL (or both). If your license supports it, select either Verilog or VHDL from the ***Create Simulation Model*** drop-down menu in the Simulation subsection.

4 Software

This section describes how to download one of the provided pre-built FPGA bitstreams to a development board and how to compile and run a test program on it.

The string **EXAMPLES** refers to the `examples` subdirectory of the extracted release.

4.1 Prerequisites

Before you begin, make sure you have:

- Installed IntelFPGA Quartus Prime 17.1

  Quartus Prime 17.1 is the currently recommended version, as this is the version that was used to create the pre-built bitstreams and BSPs, but the the VectorBlox MXP IP will likely work with other versions of Quartus

- Installed and tested the USB Blaster device driver.

- A supported development board. See the contents of `EXAMPLES/boards/`.

- Connected your development board to your computer via USB cable.

4.2 Configuring the board

1. **Start a Nios II Command Shell**

   This sets up the proper environment variables for using the Nios II compiler system. Under Windows, this is a bash-shell under the Cygwin environment.

2. **Connect to the board**

   Run `jtagconfig` which finds the USB Blaster and connects to the development board. If the response is "No JTAG hardware available," be sure the USB cable is connected and the board is turned on.

   If more than one board is connected, the target cable will have to be specified in all the commands below, adding `-c CABLE_NUM` to each command.

3. **Configure the FPGA**

   Navigate to one of the prebuilt VectorBlox MXP systems for your development board, located in `EXAMPLES/boards/<board_name>/prebuilt_*`.

   For example, for the DE2-115 board, change directory to `EXAMPLES/boards/de2_115/prebuilt_de2_115_v16` for a 16-lane MXP system.

   From here, download the `.sof` file to the FPGA board using the command line:

   ```
   quartus_pgm -m JTAG -o P\;vblox1.sof
   ```

   Alternatively, you may start the GUI using `quartus_pgmw`, press **Add File...** and choose the `.sof` file, press **Hardware Setup...** and choose **USB Blaster** in the pulldown list, and finally press the **Start** button.

   Now, assuming you are still in the directory containing the SOF, store the absolute path of the pre-built system's Board Support Package in a shell variable. This will save typing later. In Linux, run

   ```
   PROJ_ROOT=`pwd`
   ```

   In Windows, Windows-style paths are expected by Altera's software Makefiles, so instead use

```
PROJ_ROOT=`cygpath -wa .`
```

4.3   Running the software

1. **Make the executable**

   Navigate to a test application such as `vbw_vec_add_t`, located in `EXAMPLES/software/bmark/vbw_vec_add_t`.

   To compile the program for the FPGA bitstream you previously selected, you need to pass the absolute path of the BSP to `make`. Assuming you have already saved this path in the `BSP_ROOT_DIR` shell variable as described earlier, you can run

   ```
   make clean_all all BSP_ROOT_DIR=$BSP_ROOT_DIR
   ```

   After a long list of messages and a few seconds, the file `test.elf` should have been created. If you receive the error "../vblox1.sopcinfo has been modified since the BSP was generated.", run the command `touch $BSP_ROOT_DIR/public.mk` to resolve it.

   Note that we used the `clean_all` target first to ensure that any libraries that might have been compiled against a different BSP in the past were cleaned and re-compiled. If you know that the libraries were already compiled for the selected BSP, you can omit the `clean_all` target.

2. **Download the executable**

   Download and run the executable from the command line:

   ```
   nios2-download -g -r test.elf
   ```

   After the download completes, the flag `-r` resets the Nios II computer system, and the flag `-g` tells the computer to 'go' or execute the program.

3. **View execution output**

   The executable may use stdin or stdout for `printf()`, `scanf()`, and similar functions. A terminal emulator can connect with the Nios II computer system to provide this I/O behaviour by running

   ```
   nios2-terminal
   ```

   If the Nios II program wants to disconnect with the terminal, it can print an EOT (end of transmission) character with

   ```
   putchar(4);
   ```

   To disconnect manually host-side, type `C-c`. Note that Control-C does not actually stop the application from running. However, with the `nios2-terminal` disconnected, the application may eventually block if it fills its `printf()` buffer and cannot flush the output device.