MicroBlaze Processor Embedded Design User Guide

UG1579 (v2022.1) June 1, 2022

Xilinx is creating an environment where employees, customers, and partners feel welcome and included. To that end, we're removing non-inclusive language from our products and related collateral. We've launched an internal initiative to remove language that could exclude people or reinforce historical biases, including terms embedded in our software and IPs. You may still find examples of non-inclusive language in our older products as we work to make these changes and align with evolving industry standards. Follow this link for more information.

AMD
XILINX

AMD
XILINX

Table of Contents

AMD XILINX

Chapter 1

Introduction

Device Tools Flow Overview

The Vivado® tools provide specific flows for programming, based on the processor. The Vivado IDE uses the IP integrator with graphic connectivity screens to specify the device, select peripherals, and configure hardware settings.

You can use the Vivado IP integrator to capture hardware platform information in XML format applications, along with other data files to develop designs for Xilinx processors. Software design tools use the XML to do the following:

- Create and configure board support package (BSP) libraries
- Infer compiler options
- Program the processor logic (PL)
- Define JTAG settings
- Automate other operations that require information about the hardware

The MicroBlaze™ embedded processor is a Reduced Instruction Set Computer (RISC) core, optimized for implementation in Xilinx field programmable gate arrays (FPGAs). Use Chapter 2: Using a MicroBlaze Processor in an Embedded Design to understand how to use IP integrator and other Xilinx tools to create an embedded MicroBlaze processor design. See the *MicroBlaze Processor Reference Guide* (UG984) for more processor information.

Xilinx provides design tools for developing and debugging software applications for Xilinx processors, including, but not limited to, the following:

- Software IDE
- GNU-based compiler tool-chain
- Debugging tools

These tools let you develop both bare-metal applications that do not require an operating system, and applications for an open-source Linux-based operating system.

Xilinx provides integration between a hardware design and the software development with an integrated flow down to Vitis™ software platform: a standalone product that is available for download from the Xilinx website. See the *Vitis Unified Software Platform Documentation* for more information about how to use the tool.

General Steps for Creating an Embedded Processor Design

To complete an embedded processor design, you typically perform the following steps:

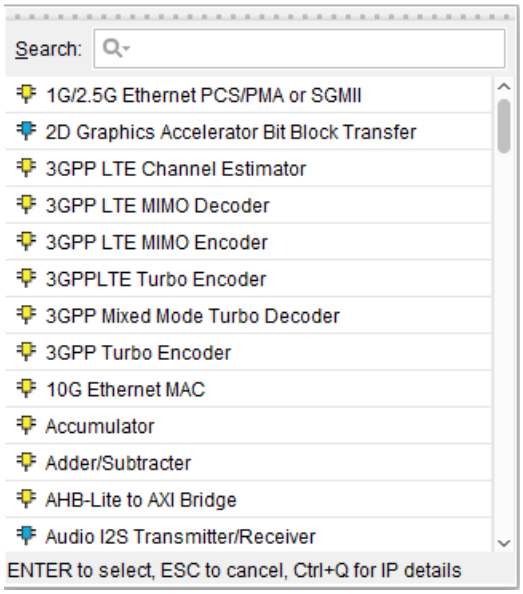
1. Create a new Vivado Design Suite project.
2. Create a block design in the IP integrator tool and instantiate a Xilinx processor, along with any other Xilinx IP or your custom IP.
3. Generate Output Products of the IP in the block design with the correct synthesis mode option.
4. Create a top-level wrapper and instantiate the block design into a top-level RTL design.
5. Run the top-level design through synthesis and implementation, and then export the hardware to the Vitis software platform.
6. Create your software application. In the Vitis software platform, associate the Executable Linkable File (ELF) file with the hardware design.
7. Program into the target board.

Embedded IP Catalog

The Vivado Design Suite IP catalog is a unified repository that lets you search, review detailed information, and view associated documentation for the IP.

After you add the third-party or customer IP to the Vivado Design Suite IP catalog, you can access the IP through the Vivado Design Suite flows. The following figure shows a portion of the Vivado IDE IP integrator IP catalog.

Figure 1: **IP Integrator IP Catalog**



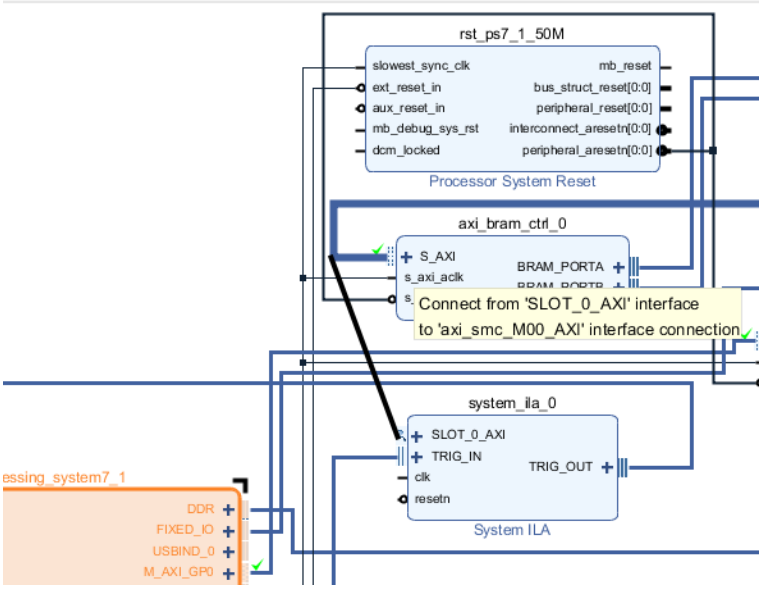
Making Manual Connections in a Design

The following figure shows how you can connect the ILA `SLOT_0_AXI` or the `clk` pin to the clock and the AXI interface that needs to be monitored in the design. You can do this manually.

As you move the cursor near an interface or pin connector on an IP block, the cursor changes to a pencil. Click an interface or pin connector on an IP block, and drag the connection to the destination block.

The following figure illustrates the use of manual connections.

Figure 2: **Manually Connecting Ports**



Manually Creating and Connecting to I/O Ports

You can manually create external I/O ports in the Vivado IP integrator by connecting signals or interfaces to external I/O ports then selecting a pin, a bus, or an interface connection.

To manually create/connect to an I/O port, right-click the port in the block diagram, and then select one of the following from the right-click menu:

- **Make External:** Use the Ctrl+Click keyboard combination to select multiple pins and invoke the Make External connection. This command ties a pin on an IP to an I/O port on the block design.
- **Create Port:** Creates non-interface signals, such as a `clock`, `reset`, or `uart_txd`. The Create Port option gives more control in terms of specifying the input and output, the bit-width and the type (`clk`, `reset`, or `data`). In case of a clock, you can also specify the input frequency.
- **Create Interface Port:** Creates ports on the interface for groupings of signals that share a common function. For example, the `S_AXI` is an interface port on several Xilinx IP. The command gives more control in terms of specifying the interface type and the mode (master or slave).

Platform Board Flow in IP Integrator

The Vivado Design Suite is a *board-aware*. The tools know the various components present on the target board and can customize an IP to be instantiated and configured to connect to the components of a particular board.

The IP integrator shows all the components present on the board in a separate tab called the Board tab.

When you use this tab to select components and the designer assistance offered by IP integrator, you can easily connect your design to the components of your choice. I/O constraints are automatically generated as a part of using this flow.

See section Using the Board Flow in the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information.

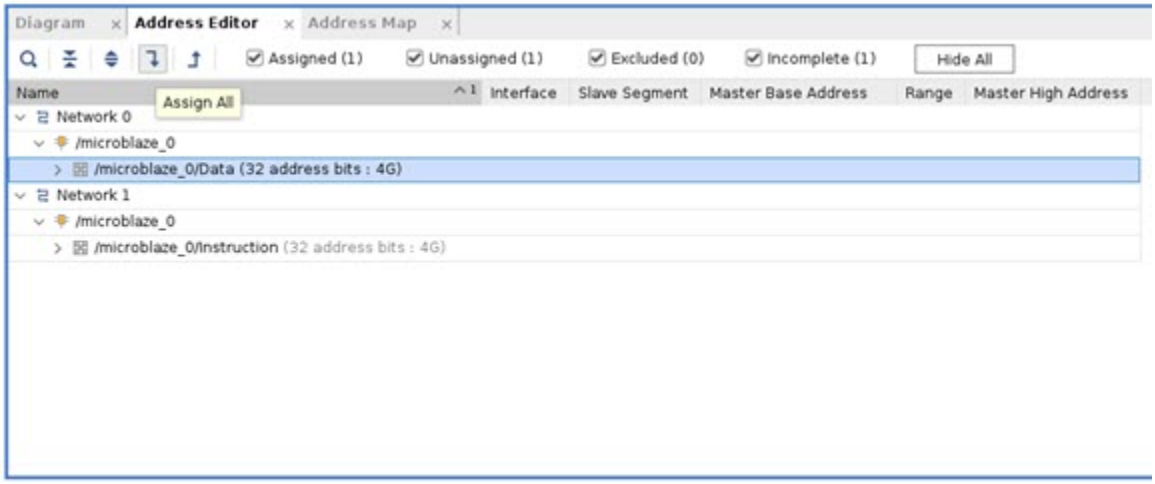
Memory-Mapping in the Address Editor

While memory-mapping of the peripherals (slaves) instantiated in the block design are automatically assigned, you can manually assign the addresses also. To generate the address map for this design, do the following:

1. Click the **Address Editor** tab above the diagram.
2. Click the **Assign All**  button.

You can manually set addresses by entering values in the Offset Address and Range columns. See section Creating a Memory Map in the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information.

Figure 3: **Memory-Mapping Peripherals**



TIP: *The Address Editor tab only opens if the diagram contains an IP such as the MicroBlaze device that functions as a bus master in the design.*

Running Design Rule Checks

The Vivado IP integrator runs basic DRCs in real time as you put the design together. However, errors can occur during design creation. For example, the frequency on a clock pin might not be set correctly.

To run a comprehensive DRC, click the **Validate Design** button .

If no warnings or errors occur in the design, a validation dialog box displays to confirm that there are no errors or critical warnings in your design.

Integrating a Block Design in the Top-Level Design

After you complete the block design and validate the design, there are two more steps required to complete the design:

- Generate the output products
- Create a HDL wrapper

Generating output products makes the source files and the appropriate constraints for the IP available in the Vivado IDE **Sources** window.

Depending upon what you selected as the target language during project creation, the IP integrator tool generates the appropriate files. If the Vivado IDE cannot generate the source files for a particular IP in the specified target language, a message displays in the console.

Generating Output Products

To generate output products, perform one of the following step.

- In the Block Design panel, expand the Design Sources hierarchy and select **Generate Output Products**.
- In the Flow Navigator panel, under IP integrator, click **Generate Block Design**.

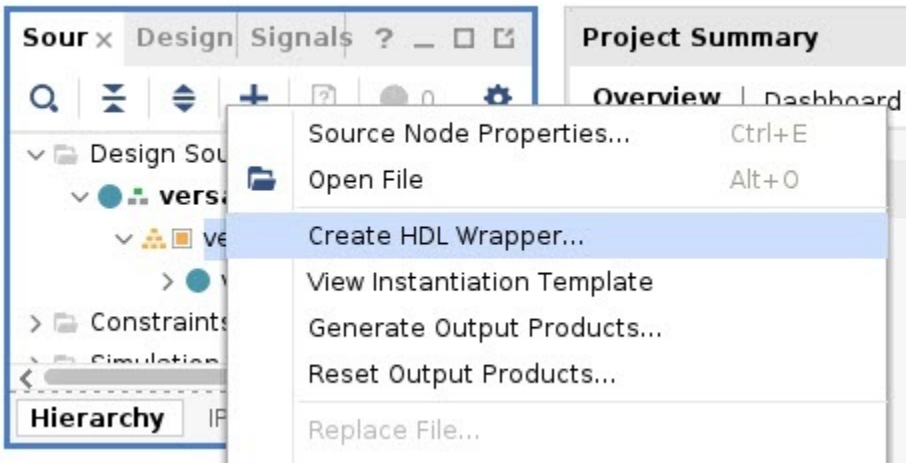
The Vivado Design Suite generates the HDL source files and the appropriate constraints for all the IP used in the block design. The source files are generated based upon the Target Language that you select during project creation, or in the Settings dialog box. See *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994), for more information on generating output products.

Creating an HDL Wrapper

You can integrate an IP integrator block design into a higher-level design by instantiating the design in a higher-level HDL file.

To instantiate at a higher level, in the Design Sources hierarchy of the Block Design panel, right-click the design and select **Create HDL Wrapper**, as shown in the following figure.

Figure 4: **Creating an HDL Wrapper**

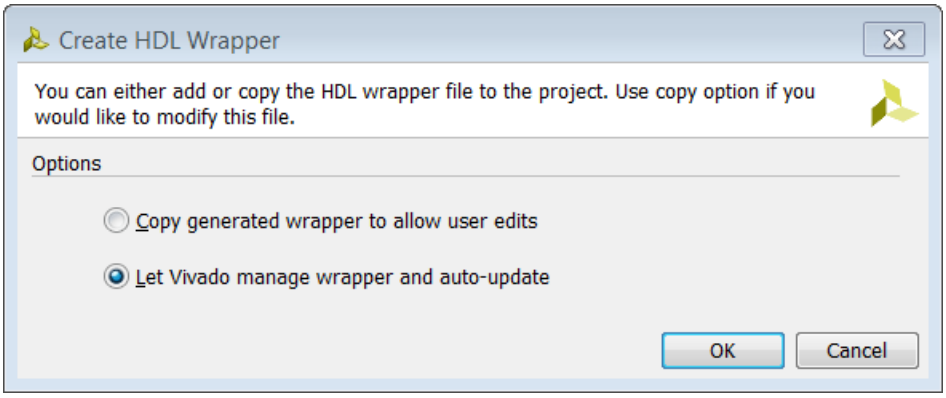


See section Integrating the Block Design into a Top-Level Design in *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information on generating output products.

Vivado offers two choices for creating an HDL wrapper, as shown in the following figure:

- You can let Vivado create and automatically update the wrapper, which is the default option.
- Create a user-modifiable script, which you can edit and maintain. Choosing this option requires that you update the wrapper every time you make port-level changes in the block design.

Figure 5: **Create HDL Wrapper Dialog Box**



This generates a top-level HDL file for the IP integrator subsystem. You can now take your design through the other design flows: elaboration, synthesis, and implementation.

Using the Vitis Software Platform

The Vitis software platform IDE provides a complete environment for creating software applications targeted for Xilinx embedded processors. It includes:

- A GNU-based compiler toolchain (GCC compiler, TCF System debugger, utilities, and libraries)
- A JTAG debugger
- A flash programmer
- Drivers for Xilinx IP and bare-metal board support packages
- Middleware libraries for application-specific functions
- An IDE for C/C++ bare-metal and Linux application development and debugging

Based upon the open source Eclipse platform, the Vitis software platform incorporates the C/C++ Development Toolkit (CDT).

Features of the Vitis software platform include:

- C/C++ code editor and compilation environment
- Project management
- Application build configuration and automatic make file generation
- Error navigation
- Integrated environment for debugging and profiling embedded targets
- Additional functionality available using third-party plug-ins, including source code version control

Exporting a Hardware Description

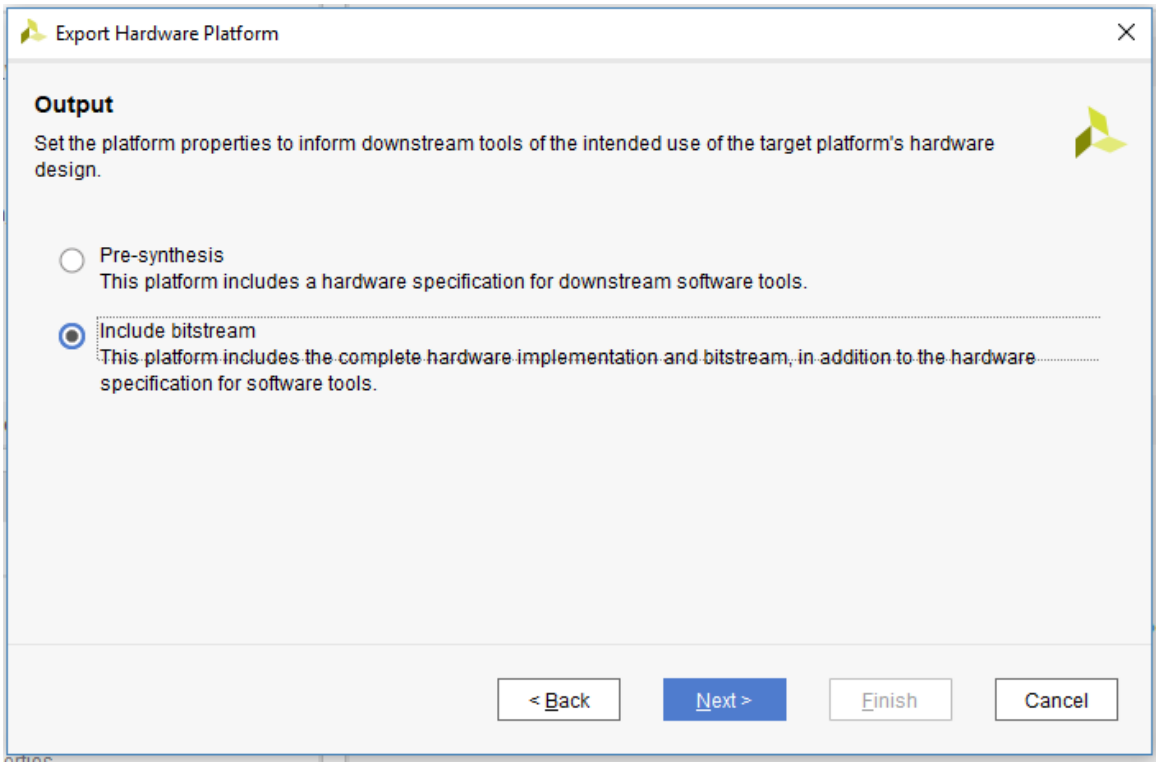
Once a design has been implemented and the bitstream generated, you can export the design to the Vitis software platform for software application development. In rare cases where the Processing Logic does not contain any logic at all, you can also export the design without implementing or generating the bitstream.

To export your design, perform the following:

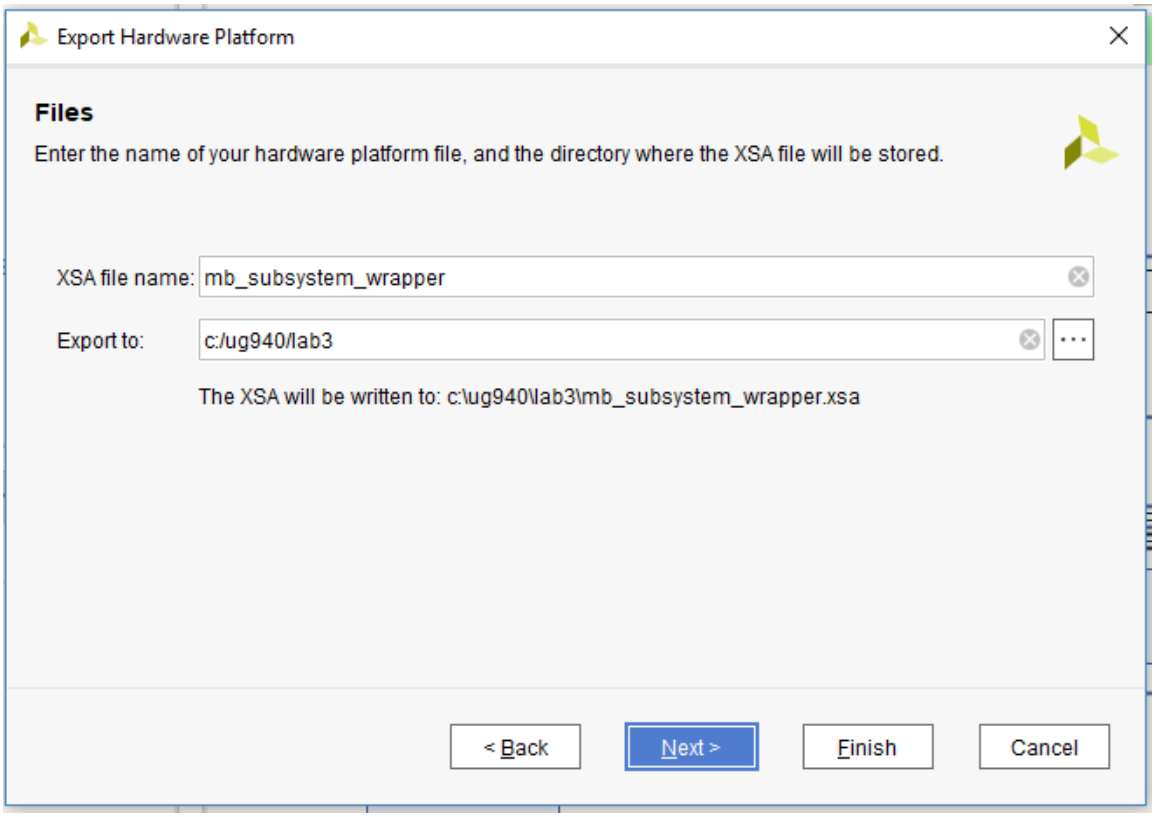
1. From the Vivado File menu, select **File → Export → Export Hardware**.

   The Export Hardware dialog box opens.

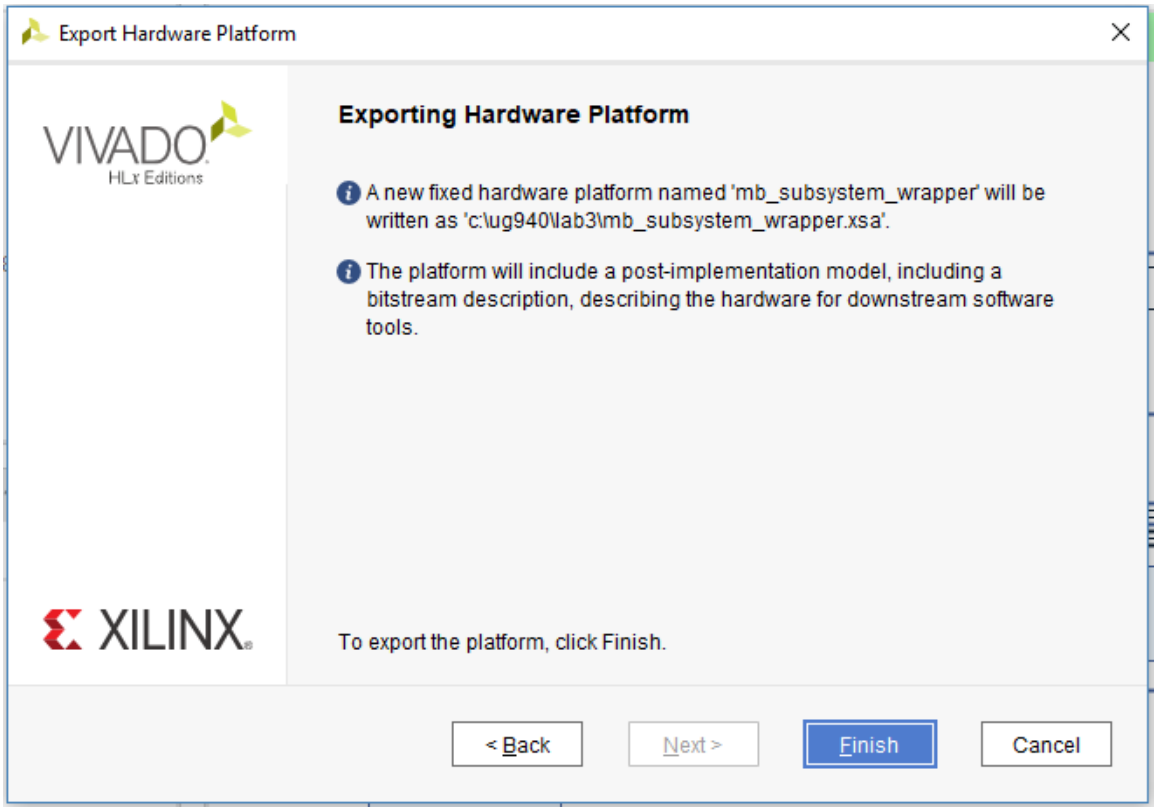
2. Select the **Include bitstream** option using the radio button in the Output view and click **Next**.



3. Leave the XSA file name field at its default value and click **Next**. (The following figure shows Windows-specific settings.)

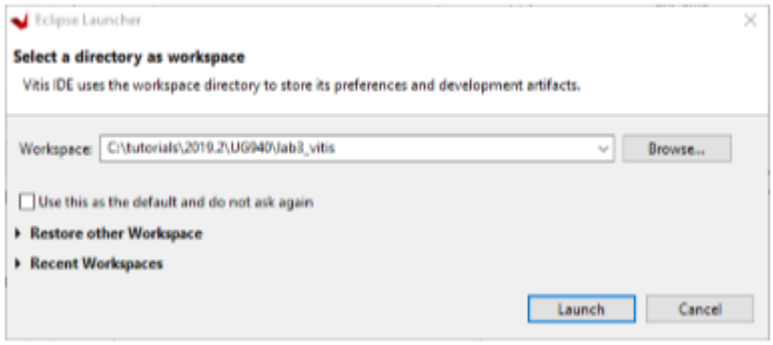


4. Click **Finish**. This will export the hardware XSA File in the project directory.



5. After the hardware definition has been exported, select **Tools → Launch Vitis** to launch the Vitis software platform from Vivado.

   The Eclipse Launcher dialog box opens, as shown in the following figure.



The Workspace field should be populated with the name of the directory where the software application project is to be created.

After you export the hardware definition to the Vitis software platform, and launch, you can start writing your software application.

You can perform further debug and software download from the Vitis software platform.

Alternatively, you can import the ELF file for the software back into the Vivado tools, and integrate it with the FPGA bitstream for further download and testing.

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal® ACAP design process Design Hubs and the Design Flow Assistant materials can be found on the Xilinx.com website. This document covers the following design processes:

- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
  - Chapter 2: Using a MicroBlaze Processor in an Embedded Design
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Chapter 3: Designing with the Memory IP Core
  - Chapter 4: Reset and Clock Topologies in IP Integrator

Chapter 2

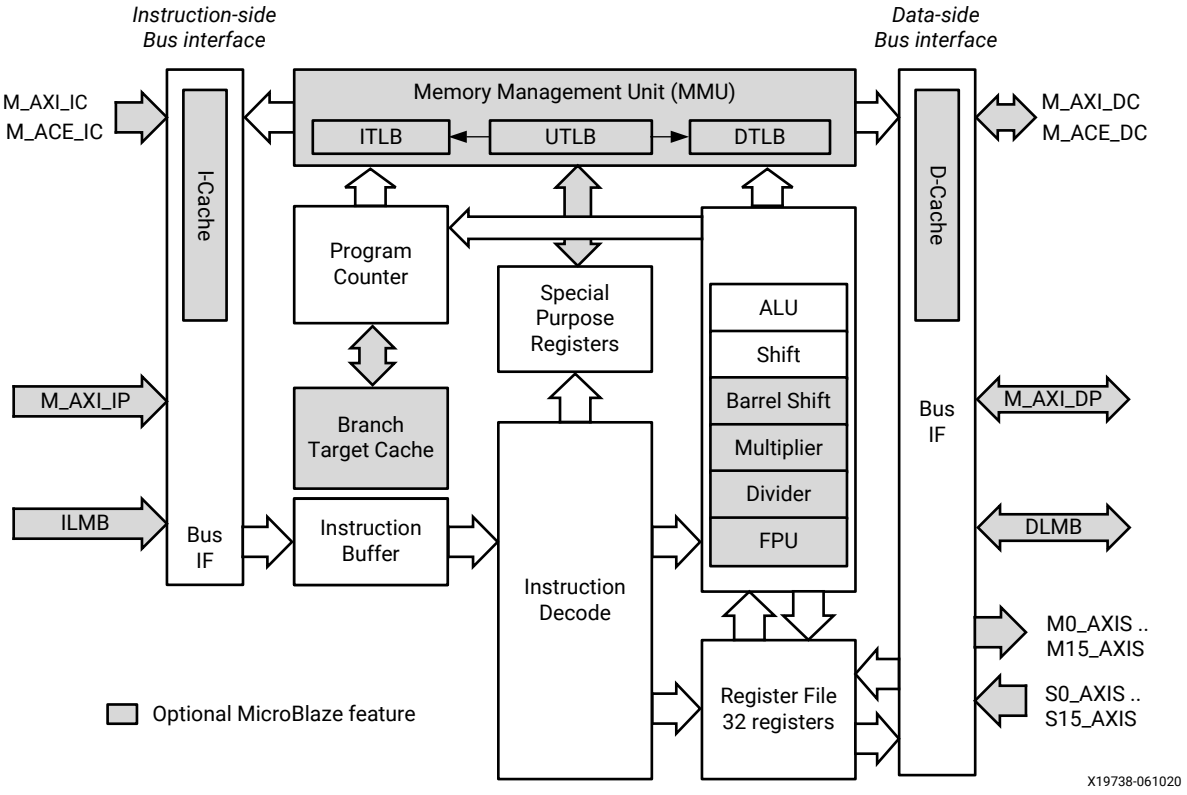
Using a MicroBlaze Processor in an Embedded Design

The Vivado® IDE IP integrator is a powerful tool that lets you stitch together a processor-based system.

The MicroBlaze™ embedded processor is a reduced instruction set computer (RISC) core, optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs).

The following figure shows a functional block design of the MicroBlaze core.

Figure 6: **Block Design of MicroBlaze Core**



The MicroBlaze processor is highly configurable: you can select a specific set of features required by your design. The fixed feature set of the processor includes:

- Thirty-two 32-bit or 64-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus, extensible to 64-bits
- Single issue pipeline

In addition to these fixed features, the MicroBlaze processor has parameterized values that allow selective enabling of additional functionality.

RECOMMENDED: *Older (deprecated) versions of MicroBlaze support a subset of the optional features described in this manual. Only the latest (preferred) version of MicroBlaze (v11.0) supports all options. Xilinx recommends that new designs use the latest preferred version of the MicroBlaze processor.*

See the *MicroBlaze Processor Reference Guide* (UG984) for more information.

MicroBlaze can be implemented either as a 32-bit processor or a 64-bit processor, depending on user requirements. In general, Xilinx recommends that you select the 32-bit processor implementation unless specific requirements cannot be met. The 64-bit processor extends general-purpose registers to 64 bits, provides additional instructions to handle 64-bit data, and can transparently address instructions and data using up to a 64-bit address. In addition, the floating point unit (FPU) is extended to support double precision.

Another useful document reference is the *Triple Modular Redundancy (TMR) LogiCORE IP Product Guide* (PG268), which provides soft error detection, correction and recovery for Xilinx devices. The guide describes the IP cores that are part of the solution, and explains typical use cases.

Creating a MicroBlaze Processor Design

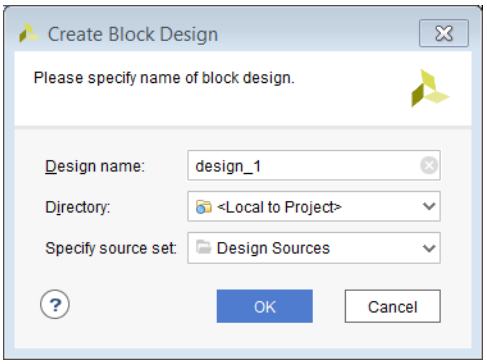
Designing with a MicroBlaze processor in the Xilinx Vivado IP integrator is different than it was in the legacy ISE® Design Suite and the Embedded Development Kit (EDK).

The Vivado IDE uses the IP integrator tool for embedded development. The IP integrator is a GUI-based interface that lets you stitch together complex IP subsystems.

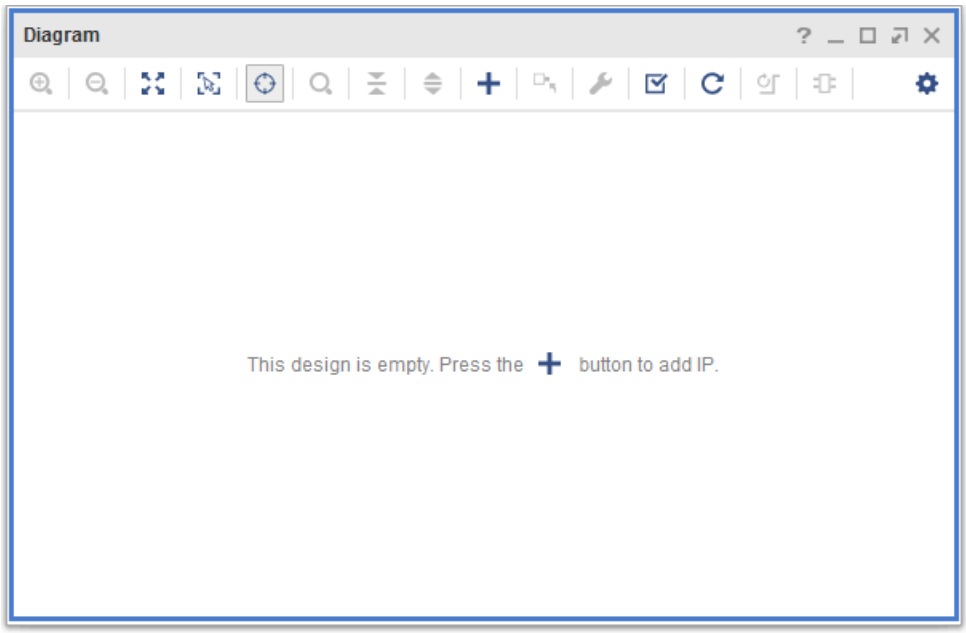
A variety of IP are available in the Vivado IDE IP catalog to meet the needs of complex designs. You can also add custom IP to the IP catalog.

Designing with the MicroBlaze Processor

1. In the Flow navigator panel, under IP integrator, click the **Create Block Design** button to open the Create Block Design dialog box.
2. Type the Design Name, as shown in the following figure.

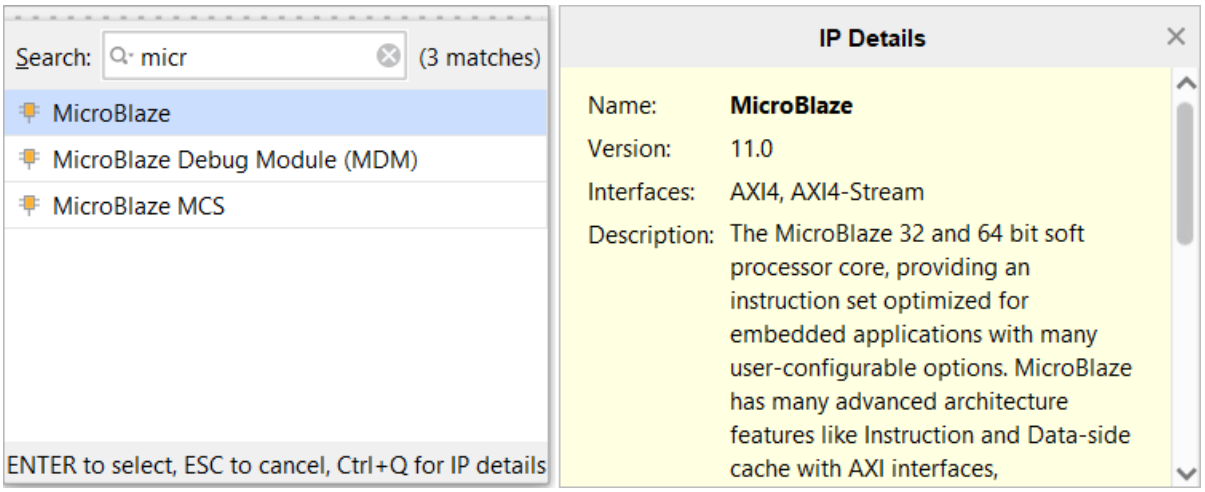


The Block Design window opens, as shown in the following figure.



3. Within the empty design, use either the **Add IP** button  on the design canvas, or right-click in the canvas, and select **Add IP**.

   A Search box opens to let you search for and select the MicroBlaze processor, as shown in the following figure.



When you select the MicroBlaze IP, the Vivado IP integrator adds the IP to the design, and a graphical representation of the processing system displays, as shown in the following figure.

***Note*:** The Tcl command is as follows:

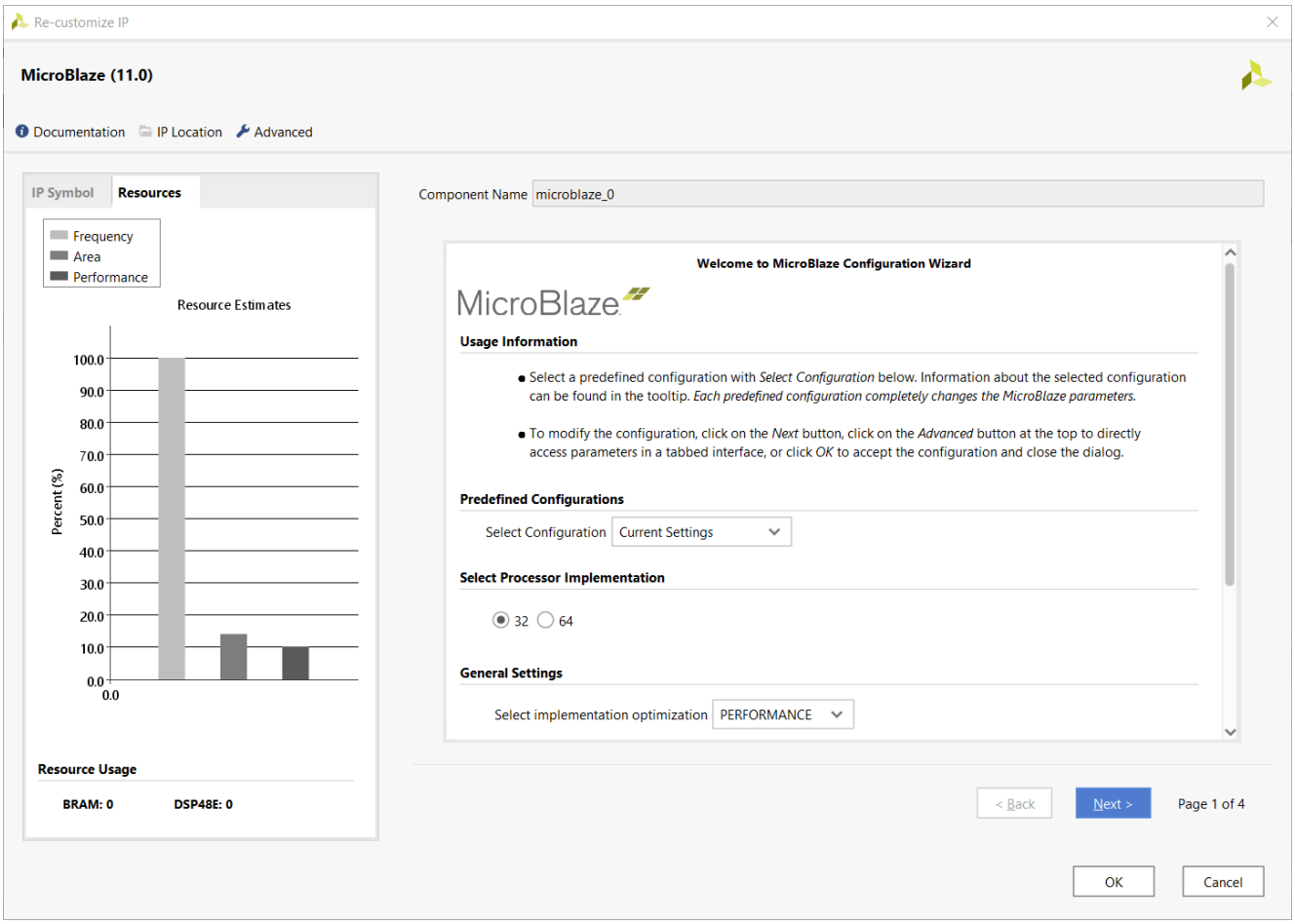
```
create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze:11.0 microblaze_0
```

4. Double-click the MicroBlaze IP in the canvas to invoke the Re-customize IP process, which displays the Re-customize IP configuration page for the MicroBlaze processor, shown in the following figure.

Using the MicroBlaze Configuration Window

The following figure shows the Welcome page of the MicroBlaze configuration wizard.

Figure 7: **MicroBlaze Configuration Wizard**



The MicroBlaze Configuration wizard provides the following:

- Predefined configuration templates for one-click configuration.
- Estimates of MicroBlaze relative frequency, area, and performance, giving immediate feedback based on selected configuration options.
- Page by page guidance through the configuration process.
- Tool tips for all configuration options to understand the effect of each option.
- An **Advanced** button that provides a tabbed interface for direct access to all of the configuration options, see MicroBlaze Configuration Wizard: Advanced Mode.

> **IMPORTANT!** *Interrupt & Reset and PVR options are only accessible through the Advanced mode.*

The MicroBlaze Configuration wizard includes the following pages which are shown depending on the options selected on the Welcome page:

- **Welcome Page**: Shows the **Predefined Configurations and General Settings**. See the MicroBlaze Configuration Wizard: Welcome Page for more information.
- **General**: Shows the selection of execution units and optimization settings (this General information is persistent). See the MicroBlaze Configuration Wizard: General Page for more information.
- **Exceptions**: Shows the Exceptions page when you select **Enable Selections** that option on the Welcome Page. See the MicroBlaze Configuration Wizard: MMU Page for more information.
- **Cache**: Cache settings page is shown when you select **Use Instructions and Data Caches**. See the MicroBlaze Configuration Wizard: Cache Page for more information.
- **MMU**: Shows the MMU settings page when you select **Use Memory Management** on the Welcome Page. See the MicroBlaze Configuration Wizard: MMU Page for more information.
- **Debug**: Shows the number of breakpoints and watchpoints when you select **Enable MicroBlaze Debug Module Interface**. See the MicroBlaze Configuration Wizard: Debug Page for more information.
- **Buses**: Shows the Bus settings, which are persistent, as the last page of the configuration wizard. See the MicroBlaze Configuration Wizard: Buses Page for more information.

The left portion of the dialog box shows the relative values of the frequency, area, and performance for the current settings, block RAM, and DSP numbers:

- **Frequency**: Estimated frequency percentage relative to the maximum achievable frequency with this architecture and speed grade, which gives an indication of the relative frequency that can be achieved with the current settings.

  Note: This is an estimate based on a set of predefined benchmarks, which can deviate up to 30% from the actual value. Do not take this estimation as a guarantee that the system can reach a corresponding frequency.

- **Area**: Estimated area percentage in LUTs relative to the maximum area using this architecture, which gives an indication of the relative MicroBlaze area achievable with the current settings.

  Note: This is an estimate, which can deviate up to 5% from the actual value. Do not take this estimation as a guarantee that the implemented area matches this value.

- **Performance**: Indicates the relative MicroBlaze processor performance achievable with the current settings, relative to the maximum possible performance.

  Note: This is an estimate based on a set of benchmarks, and actual performance can vary significantly depending on the user application.

- **BRAMs**: Total number of block RAMs used by the MicroBlaze processor. The instruction and data caches, and the branch target cache use block RAMs, as well as the memory management unit (MMU), which uses one block RAM in virtual or protected mode with 32-bit mode, and two with 64-bit mode.
- **DSP48**: Total number of DSP48 used by the MicroBlaze processor. The integer multiplier, and the floating point unit (FPU) use this total value to implement float multiplication.

MicroBlaze Configuration Wizard: Welcome Page

The simplest way to use the MicroBlaze Configuration wizard is to select one of the ten predefined templates, each defining a complete MicroBlaze configuration. You can use a predefined template as a starting point for a specific application, using the wizard to refine the configuration, by adapting performance, frequency, or area.

When you modify an option, you received direct feedback that shows the estimated relative change in performance, frequency, and area in the information display.

The three presets are:

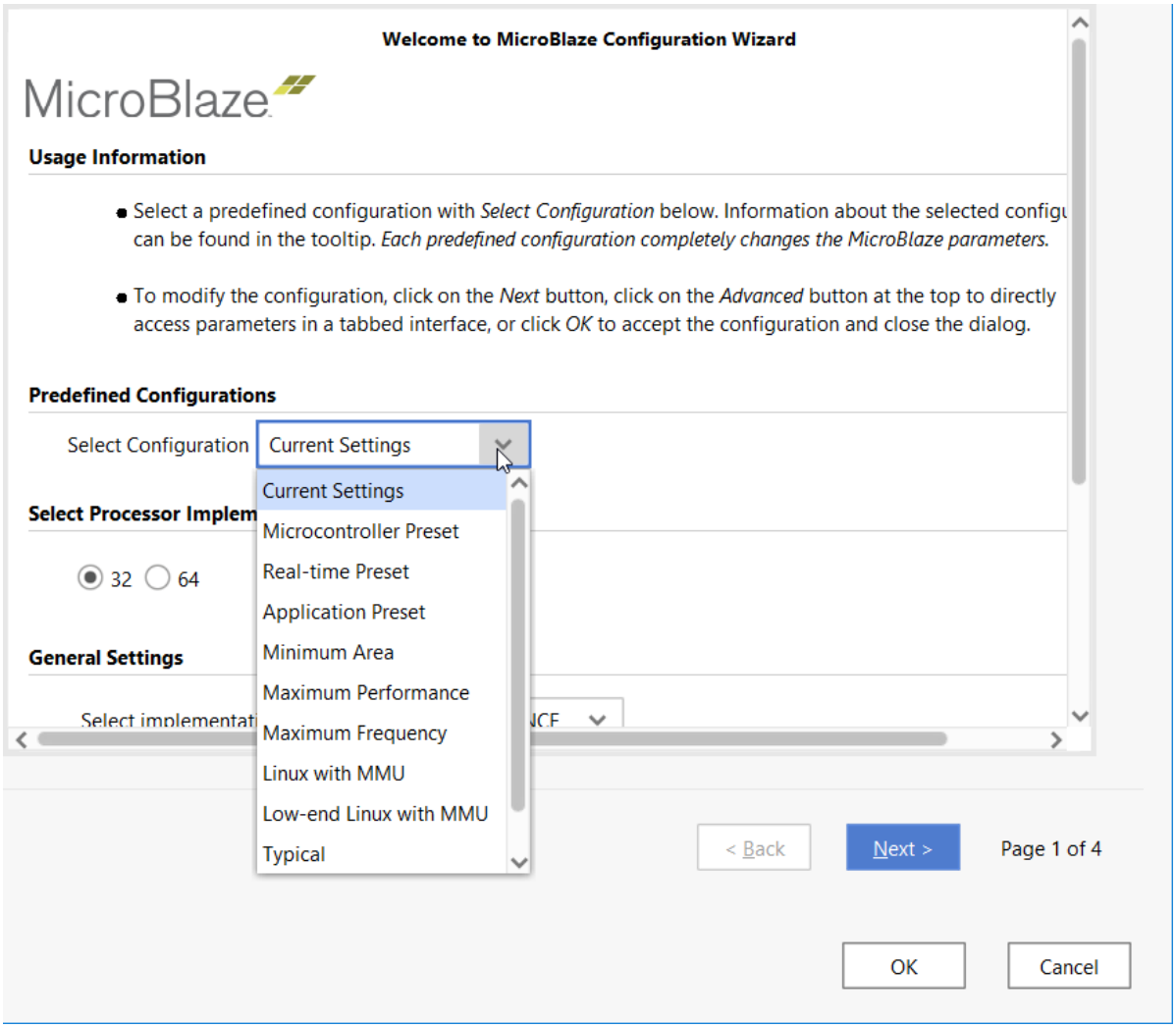
- **Microcontroller preset**: Microcontroller preset suitable for microcontroller designs. Area optimized, with no caches and debug enabled.
- **Real-time preset**: Real-time preset geared towards real-time control. Performance optimized, small caches and debug enabled, most execution units.
- **Application preset**: Application preset design for high performance applications. Performance optimized, large caches and debug enabled, and all execution units including floating-point.

The other options are:

- **Minimum Area**: The smallest possible MicroBlaze core. No caches or debug.
- **Maximum Performance**: Maximum possible performance. Large caches and debug, as well as all execution units.
- **Maximum Frequency**: Maximum achievable frequency. Small caches and no debug, with few execution units.
- **Linux with MMU**: Settings suitable to get high performance when running Linux with MMU. Memory Management enabled, large caches and debug, and all execution units.
- **Low-end Linux with MMU**: Settings corresponding to the MicroBlaze Embedded Reference System. Provides suitable settings for Linux development on low-end systems. Memory Management enabled, small caches and debug.
- **Typical**: Settings giving a reasonable compromise between performance, area, and frequency. Suitable for standalone programs, and low-overhead kernels. Caches and debug enabled.
- **Frequency Optimized**: Designed to provide all MicroBlaze features, including MMU, while still achieving high frequency by utilizing the frequency optimized 8-stage pipeline.

The following figure shows the **Predefined Configurations** in the Configuration wizard.

Figure 8: **MicroBlaze Predefined Configuration Settings**



Select Processor Implementation

Select 32-bit or 64-bit processor implementation. The 64-bit processor extends all registers to 64 bits, provides additional instructions to handle 64-bit data, and can address up to 4 EB instructions and data using up to a 64-bit address. The extended addressing is selected on the General tab.

The compiler automatically generates a 64-bit executable when 64-bit mode is selected.

General Settings

If a pre-defined template is not used, you can select the options from the pages, which are available for fine-tuning the MicroBlaze processor, based on your design needs. As you position the mouse over these different options, a tooltip informs you what the particular option means. The following bullets detail these options.

- **Select implementation optimization**: When set to:
  - **PERFORMANCE**: Implementation is selected to optimize computational performance, using a five-stage pipeline.
  - **AREA**: Implementation is selected to optimize area, using a three-stage pipeline with lower instruction throughput.
  - **FREQUENCY**: Implementation is selected to optimize MicroBlaze frequency, using an eight-stage pipeline.

  RECOMMENDED: *It is recommended to select AREA optimization on architectures with limited resources such as Artix 7 or Spartan 7 devices. Selecting FREQUENCY optimization is recommended in order to reach system frequency targets, particularly with cache-based external memory, MMU, and/or large LMB memory. However, if performance is critical, AREA or FREQUENCY optimization should not be selected, because some instructions require additional clock cycles to execute.*

  ***Note*:** You cannot use the Memory Management Unit (MMU), Branch Target Cache, Instruction Cache Streams, Instruction Cache Victims, Data Cache Victims, and AXI Coherency Extension (ACE) with area optimization.
- **Enable MicroBlaze Debug Module Interface**: Enable debug to be able to download and debug programs using Xilinx System Debugger (XSDB).

  RECOMMENDED: *Unless area resources are very critical, it is recommended that debugging always is enabled.*
- **Use Instruction and Data Caches**: You can use MicroBlaze with optional instruction and data caches for improved performance when executing code that resides outside the LMB address range.

  The caches have the following features:
  - Direct mapped (1-way associative)
  - User selectable cacheable memory address range
  - Configurable cache size
  - Caching over AXI4 interfaces (M_AXI_IC and M_AXI_DC)
  - Option to use 4, 8, or 16 word cache line
  - Cache on and off controlled using bits in the MSR
  - Optional WIC and WDC instructions to invalidate, clear or flush instruction cache lines

- Optional instruction cache stream buffers to improve performance by speculatively prefetching instructions
- Optional victim cache to improve performance by saving evicted cache lines
- Optional parity protection; invalidates cache lines if Block RAM bit error is detected
- Optional data width selection to either use 32 bits, an entire cache line, or 512 bits

Activating caches significantly improves performance when using external memory, even if you must select small cache sizes to reduce resource usage.

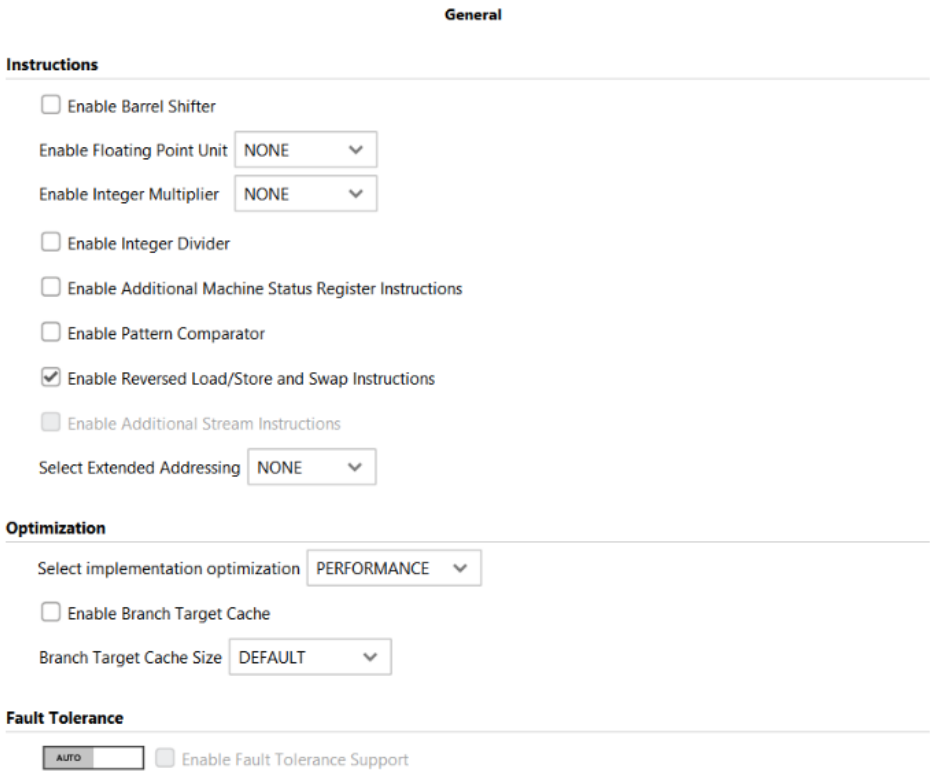
- **Enable Exceptions**: Enables exceptions when using an operating system with exception support, or when explicitly adding exception handlers in a standalone program.
- **Use Memory Management**: Enables Memory Management if planning to use an operating system - such as Linux -with support for virtual memory of memory protection.

  Note: When you enable area optimized MicroBlaze or stack protection, the Memory Management Unit is not available.
- **Enable Discrete Ports**: Enables discrete ports on the MicroBlaze instance, which is useful for:
  - Generating software breaks (`Ext_BRK`, `Ext_NM_BRK`)
  - Managing processor sleep and wakeup (`Sleep`, `Hibernate`, `Suspend`, `Wakeup`, `Dbg_Wakeup`)
  - Handling debug events (`Debug_Stop`, `MB_Halted`)
  - Signaling error when using fault tolerance (`MB_Error`)
  - Pausing the processor (`Pause`, `Pause_Ack`, `Dbg_Continue`)
  - Setting reset mode (`Reset_Mode`)

MicroBlaze Configuration Wizard: General Page

The following figure shows the General page of the MicroBlaze Configuration wizard.

Figure 9: **General Page of the MicroBlaze Configuration Wizard**



Instructions

- **Enable Barrel Shifter**: Enables a hardware barrel shifter in MicroBlaze. This parameter enables the instructions `bsrl`, `bsra`, `bsll`, `bsrli`, `bsrai`, `bslli`, `bsifi`, and `bsefi`. With the 64-bit processor implementation the corresponding long instructions are also enabled. Enabling the barrel shifter can dramatically improve the performance of an application, but increases the size of the processor. The compiler uses the barrel shift instructions automatically if this parameter is enabled.
- **Enable Floating Point Unit**: Enables a floating point unit (FPU) based on the IEEE-754 standard. Single-precision is available with the 32-bit processor implementation, and double-precision is added with the 64-bit implementation. Using the FPU significantly improves the floating point performance of the application and significantly increases the size of MicroBlaze.

  Setting this parameter to BASIC enables `add`, `subtract`, `multiply`, `divide` and `compare` instructions. Setting it to EXTENDED also enables `convert` and `square-root` instructions. The compiler automatically uses the FPU instructions corresponding to setting of this parameter.
- **Enable Integer Multiplier**: Enables a hardware integer multiplier in MicroBlaze. This parameter enables the instructions `mul` and `muli` when set to MUL32.

When set to MUL64, this enables the additional instructions: `mulh`, `mulhu`, and `mulhsu` for 64-bit multiplication. This parameter can be set to NONE to free up `DSP48` primitives in the device for other uses. Setting this parameter to NONE has a minor effect on the area of the MicroBlaze processor. When this parameter is enabled, the compiler uses the `mul` instructions automatically.

- **Enable Integer Divider**: Enables a hardware integer divider in MicroBlaze. This parameter enables the instructions, `idiv` and `idivu`. Enabling this parameter can improve the performance of an application that performs integer division, but increases the size of the processor. When this parameter is enabled, the compiler uses the `idiv` instructions automatically.
- **Enable Additional Machine Status Register Instructions**: Enables additional machine status register (MSR) instructions for setting and clearing bits in the MSR. This parameter enables the instructions `msrset` and `msrclr`. Enabling this parameter improves the performance of changing bits in the MSR.
- **Enable Pattern Comparator**: Enables pattern compare instructions `pcmpbf`, `pcmpeq`, and `pcmpne`.

  The pattern compare bytes find (`pcmpbf`) instructions return the position of the first byte that matches between two words and improves the performance of string and pattern matching operations. The Vitis™ libraries use the `pcmpbf` instructions automatically when this parameter is enabled.

  The `pcmpeq` and `pcmpne` instructions return 1 or 0 based on the equality of the two words. These instructions improve the performance of setting flags and the compiler uses them automatically. With the 64-bit processor implementation, the corresponding long instructions are also enabled.

  Selecting this option also enables count leading zeroes instruction, `clz`. The `clz` instruction can improve performance of priority decoding, and normalization.
- **Enable Reversed Load/Store and Swap Instructions**: Enables reversed load/store and swap instructions `lbur`, `lhur`, `lwr`, `sbr`, `shr`, `swr`, `swapb`, and `swaph`. With the 64-bit processor implementation, the long reversed load/store instructions `llr` and `slr` are also enabled. The reversed load/store instructions read or write data with opposite endianness, and the swap instructions allow swapping bytes or half-words in registers. These instructions are mainly useful to improve performance when dealing with big-endian network access with a little-endian MicroBlaze.
- **Enable Additional Stream Instructions**: Provides additional functionality when using AXI4-Stream links, including dynamic access instruction `getd` and `putd` that use registers to select the interface.

  The instructions are also extended with variants that provide:

  - Atomic `get`, `getd`, `put`, and `putd` instructions
  - Test-only `get` and `getd` instructions

  - `get` and `getd` instructions that generate a stream exception if the control bit is not set

> **IMPORTANT!** *The extended stream instructions must be enabled to use these additional instructions, and at least one stream link must be selected. The stream exception must be enabled to use instructions that generate stream exceptions.*

- **Select Extended Addressing**: Set the memory addressing capability. With the 32-bit processor implementation, this enables additional load/store instructions to be able to access a larger address space than 4GB (32-bit address). With the 64-bit processor implementation, the extended address is handled by normal load/store instructions. The data side LMB and AXI bus addresses are extended to the number of address bits corresponding to the selected memory size. The available choices are:
  - **NONE** (32-bit address, no additional instructions)
  - **64GB** (36-bit address)
  - **1TB** (40-bit address)
  - **16TB** (44-bit address)
  - **256TB** (48-bit address)
  - **16EB** (64-bit address)
  - **4PB** (52-bit address)

  For more information, including software usage and limitations, see the *MicroBlaze Processor Reference Guide* (UG984).

Optimization

- **Select implementation optimization**: This option is the same as in the General Settings options.
- **Enable Branch Target Cache**: When set, implements the branch target, which improves branch performance by predicting conditional branches and caching branch targets.

> **TIP:** *The Enable Branch Target Cache option is not enabled when Select implementation optimization is set to AREA on the MicroBlaze Configuration Wizard: Welcome Page . Conversely, enabling Branch Target Cache disables the Area option in Select implementation optimization.*

- **Branch Target Cache Size**: Specify the size of the cache for branch targets.

Fault Tolerance

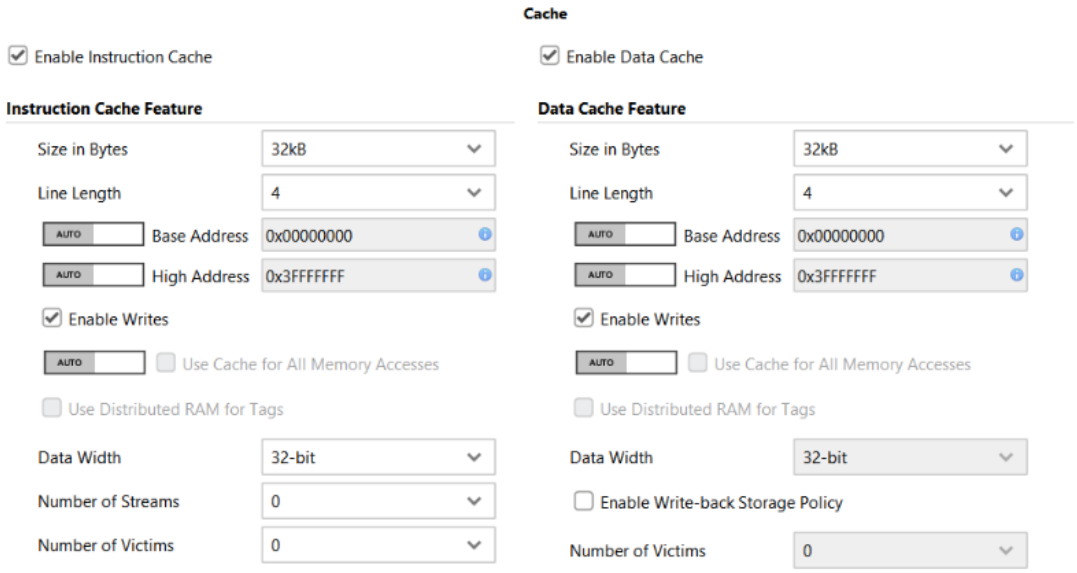
- **Auto/Manual**: Determines if the Vivado tool will automatically enable fault tolerance, or if you will specify it manually.

- **Enable Fault Tolerance Support**: When enabled, MicroBlaze protects internal block RAM with parity, and supports error correcting codes (ECC) in LMB block RAM, including exception handling of ECC errors. This prevents a bit flip in block RAM from affecting the processor function.
  - If this value is auto-computed (by not overriding it), fault tolerance is automatically enabled in MicroBlaze when ECC is enabled in connected LMB BRAM controllers.
  - If fault tolerance is explicitly enabled, the IP integrator tool enables ECC automatically in connected LMB blokRAM Controllers.
  - If fault tolerance is explicitly disabled, ECC in connected LMB block RAM controllers is not affected.

MicroBlaze Configuration Wizard: Cache Page

The following figure shows the Cache options page for the MicroBlaze configuration.

Figure 10: **Cache Options Page of the MicroBlaze Configuration Wizard**



- **Enable Instruction Cache**: Uses this cache only when it is also enabled in software by setting the instruction cache enable (ICE) bit in the machine status register (MSR).

  The Instruction Cache configurable options are:

  - **Size in Bytes**: Specifies the size of the instruction cache if C_USE_ICACHE is enabled. Not all architectures permit all sizes.
  - **Line Length**: Select between 4, 8, or 16 word cache line length for cache miss-transfers from external instruction memory.
  - **Base Address**: Specifies the base address of the instruction cache. This parameter is used only if C_USE_ICACHE is enabled.

- **High Address**: Specifies the high address of the instruction cache. This parameter is used only if C_USE_ICACHE is enabled.
- **Enable Writes**: When enabled, one can invalidate instruction cache lines with the `wic` instruction. This parameter is used only if C_USE_ICACHE is enabled.
- **Use Cache for All Memory Accesses**: When enabled, uses the dedicated cache interface on MicroBlaze is for all accesses within the cacheable range to external instruction memory, even when the instruction cache is disabled.

  Otherwise, the instruction cache uses the peripheral AXI for these accesses when the instruction cache is disabled.

  When enabled, an external memory controller must provide only a cache interface MicroBlaze instruction memory. Enable this parameter when using AXI Coherency Extension (ACE).
- **Use Distributed RAM for Tags**: Uses the instruction cache tags to hold the address and a valid bit for each cache line. When enabled, the instruction cache tags are stored in Distributed RAM instead of block RAM. This saves block RAM, and can increase the maximum frequency.
- **Data Width**: Specifies the instruction cache bus width when using AXI Interconnect. The width can be set to:
  - **32-bit**: Bursts are used to transfer cache lines for 32-bit words depending on the cache line length,
  - **Full Cache line**: A single transfer is performed for each cache line, with data width 128, 256, or 512 bits depending on cache line length
  - **512-bit**: Performs a single transfer, but uses only 128 or 256 bits, with 4 or 8 word cache line lengths.

  The two wide settings require that the cache size is at least 8 KB, 16KB, or 32KB depending upon cache line length. To reduce the AXI interconnect size, this setting must match the interconnect data width. In most cases, you can obtain the best performance with the wide settings.

  This setting is not available with area optimization, AXI Coherency Extension (ACE), or when you enable fault tolerance.
- **Number of Streams**: Specifies the number of stream buffers used by the instruction cache. A stream buffer is used to speculatively pre-fetch instructions, before the processor requests them. This often improves performance, because the processor spends less time waiting for instruction to be fetched from memory.

  To be able to use instruction cache streams, do not enable area optimization or AXI Coherency Extension (ACE).

- **Number of Victims**: Specifies the number of instruction cache victims to save. A victim is a cache line that is evicted from the cache. If no victims are saved, all evicted lines must be read from memory again, when they are needed. By saving the most recent lines, they can be fetched much faster, thus improving performance.

> **RECOMMENDED:** *It is possible to save 2, 4, or 8 cache lines. The more cache lines that are saved, the better performance becomes. The recommended value is 8 lines.*

***Note*:** To be able to use instruction cache victims, do not enable area optimization or AXI Coherency Extension (ACE).

- **Enable Data Cache**: Uses this cache only when it is also enabled in software by setting the data cache enable (DCE) bit in the machine status register (MSR).

  Data Cache Features:

  - **Size in Bytes**: Specifies the size of the data cache if **C_USE_DCACHE** is enabled. Not all architectures permit all sizes.
  - **Line Length**: Select between **4**, **8**, or **16** word cache line length for cache miss-transfers from external memory.
  - **Base Address**: Specifies the base address of the data cache. This parameter is used only if **C_USE_DCACHE** is enabled.
  - **High Address**: Specifies the high address of the data cache. This parameter is used only if **C_USE_DCACHE** is enabled.
  - **Enable Writes**: When enabled, one can invalidate data cache lines with the `wdc` instruction. This parameter is used only if **C_USE_DCACHE** is enabled.
  - **Use Cache for All Memory Accesses**: When enabled, uses the dedicated cache interface on MicroBlaze is for all accesses within the cacheable range to external memory, even when the data cache is disabled.

    Otherwise, the data cache uses the peripheral AXI for these accesses when the data cache is disabled.

    When enabled, an external memory controller must provide only a cache interface MicroBlaze data memory. Enable this parameter when using AXI Coherency Extension (ACE).
  - **Use Distributed RAM for Tags**: Uses the data cache tags to hold the address and a valid bit for each cache line. When enabled, the data cache tags are stored in Distributed RAM instead of block RAM. This saves block RAM, and can increase the maximum frequency.
  - **Data Width**: Specifies the data cache bus width when using AXI Interconnect. The width can be set to:
    - **32-bit**: Bursts are used to transfer cache lines for 32-bit words depending on the cache line length
    - **Full Cache line**: A single transfer is performed for each cache line, with data width 128, 256, or 512 bits depending on cache line length

- **512-bit**: Performs a single transfer, but uses only 128 or 256 bits, with 4 or 8 word cache line lengths

The two wide settings require that the cache size is at least 8 KB, 16KB, or 32KB depending upon cache line length. To reduce the AXI Interconnect size, this setting must match the interconnect data width. In most cases, you can obtain the best performance with the wide settings.

***Note*:** This setting is not available with area optimization, AXI Coherency Extension (ACE), or when you enable fault tolerance.

- **Number of Streams**: Specifies the number of stream buffers used by the instruction cache. A stream buffer is used to speculatively pre-fetch instructions, before the processor requests them. This often improves performance, because the processor spends less time waiting for instuctions to be fetched from memory.

  To be able to use instruction cache streams, do not enable area optimization for AXI Coherency Extension (ACE).

- **Enable Write-back Storage Policy**: This parameter enables use of a write-back data storage policy. When this policy is in effect, the data cache only writes data to memory when necessary, which improves performance in most cases. With write-back enabled, data is stored by writing an entire cache line. Using write-back also requires that the cache is flushed by software when appropriate, to ensure that data is available in memory; for example, when using direct memory access (DMA). When not enabled, a write-through policy is used, which always writes data to memory immediately.

TIP: *When the MMU is enabled, setting this parameter allows individual selection of storage policy for each TLB entry.*

- **Number of Victims**: Specifies the number of data cache victims to save. A victim is a cache line that is evicted from the cache. If no victims are saved, all evicted lines must be read from memory again, when they are needed. By saving the most recent lines, they can be fetched much faster, thus improving performance.

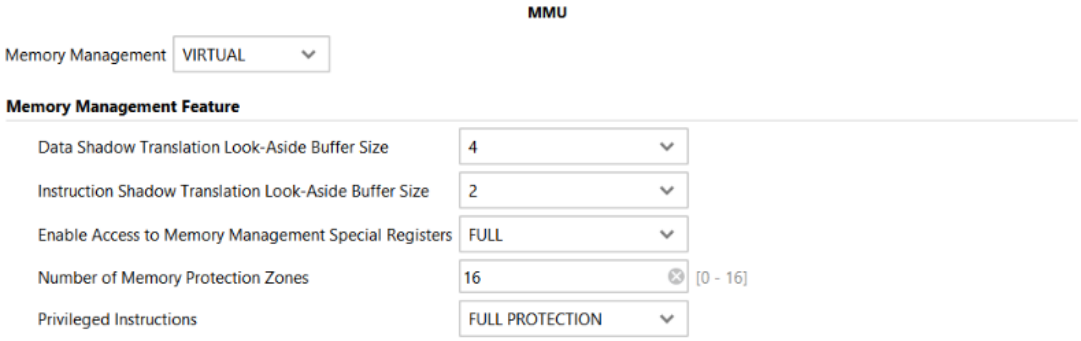
RECOMMENDED: *It is possible to save 2, 4, or 8 cache lines. The more cache lines that are saved, the better performance becomes. The recommended value is 8 lines.*

***Note*:** To be able to use data cache victims, do not enable area optimization or AXI Coherency Extension (ACE).

MicroBlaze Configuration Wizard: MMU Page

The following figure shows the MMU page of the MicroBlaze Configuration.

Figure 11: **MicroBlaze Configuration Wizard MMU Page**



Memory Management

The Memory Management field specifies the implementation of the memory management unit (MMU).

- To disable the MMU, set this parameter to None (0), which is the default.
- To enable only the User Mode and Privileged Mode instructions, set this parameter to USERMODE (1). To enable Memory Protection, set the parameter to PROTECTION (2).
- To enable full MMU functionality, including virtual memory address translation, set this parameter to VIRTUAL (3).

When USERMODE is set, it enables the Privileged Instruction exception. When PROTECTION or VIRTUAL is set, it enables the Privileged Instruction exception and the four MMU exceptions (Data Storage, Instruction Storage, Data TLB Miss, and Instruction TLB Miss).

Memory Management Features

- **Data Shadow Translation Look-Aside Buffer Size**: Defines the size of the instruction shadow translation look-aside buffer (TLB). This TLB caches data address translation information, to improve performance of the translation. The selection is a trade-off between smaller size and better performance: the default value is 4.
- **Instruction Shadow Translation Look-Aside Buffer Size**: Defines the size of the instruction shadow translation look-aside buffer (TLB). This TLB caches instruction address translation information to improve performance of the translation. The selection is a trade-off between smaller size and better performance: the default value is 2.
- **Enable Access to Memory Management Special Registers**: Enables access to the memory management special register using the `MFS` and `MTS` instructions:
  - Minimal (0) only allows writing `TLBLO`, `TLBHI`, and `TLBX`.
  - Read (1) adds reading to `TLBLO`, `TLBHI`, `TLBX`, `PID`, and `ZPR`.
  - Write (2) allows writing all registers, and reading `TLBX`.

- Full (3) adds reading of `TLBLO`, `TLBHI`, `TLBX`, `PID`, and `ZPR`.

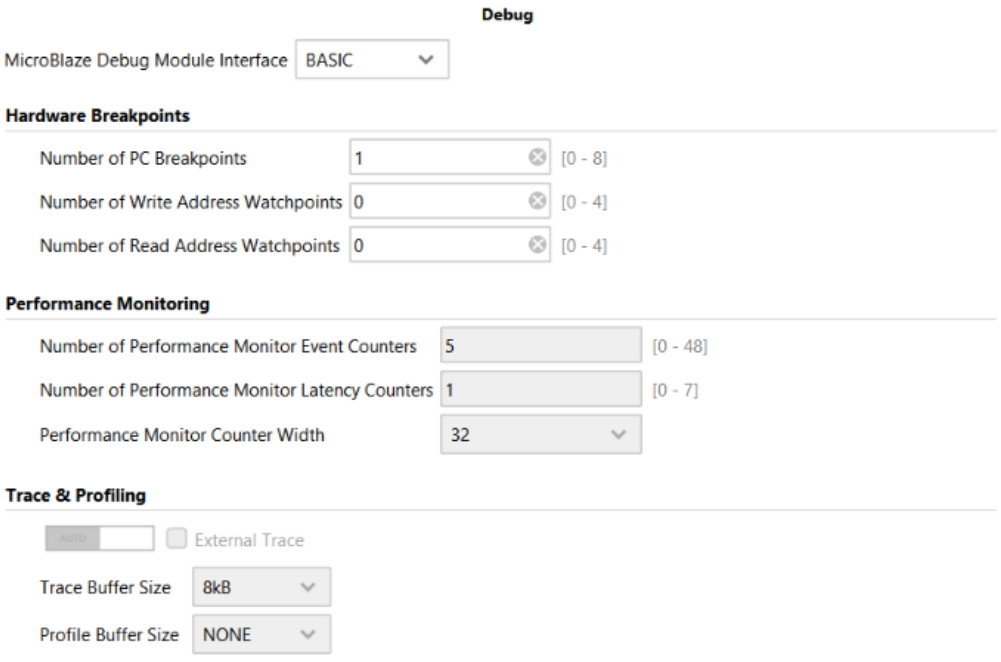
In many cases, it is not necessary for the software to have full read access. For example, this is the case for Linux memory management code. It is then safe to set access to **Write**, to save area. When using static memory protection, access can be set to **Minimal**, because the software then has no need to use `TLBSX`, `PID`, and `ZPR`.

- **Number of Memory Protection Zones**: Specifies the number of memory protection zones to implement. In many cases memory management software does not use all available zones. For example, the Linux memory management code only uses two zones. In this case, it is safe to reduce the number of implemented zones, to save resources.
- **Privileged Instructions**: Specifies which instructions to allow in **User Mode**.
  - **Full Protection** (0): Ensures full protection between processes.
  - **Allow Stream Instructions** (1): Makes it possible to use AXI4-Stream instructions in **User Mode**.
  - **Allow Extended Address Instruction** (2): Makes it possible to use extended load/store instructions when available.
  - **Allow Both** (3): Allows both types of instructions.

> **CAUTION!** *It is strongly discouraged to change this setting from Full Protection, unless it is necessary for performance reasons.*

MicroBlaze Configuration Wizard: Debug Page

Figure 12: **MicroBlaze Configuration Wizard Debug Page**



Debug Options

MicroBlaze Debug Module Interface

- **BASIC**: Enables the MicroBlaze Debug Module (MDM) interface to MicroBlaze processor for debugging. With this option, you can use Xilinx System Debugger (XSDB) to debug the processor over the Joint Test Action Group (JTAG) boundary-scan interface.
- **EXTENDED**: Enables enhanced debug features of MicroBlaze such as Cross-Trigger, Trace, and Profiling.
- **NONE**: Disables this option after you finish debugging to reduce the size of the MicroBlaze processor.

Hardware Breakpoints

IMPORTANT! *The following options are only applied if C_DEBUG_ENABLED is on. The MicroBlaze processor takes a noticeable frequency hit as the numbers are increased.*

- **Number of PC Breakpoints:** Specifies the number of program counter (PC) hardware breakpoints Xilinx System Debugger (XSDB) can set.
- **Number of Write Address Watchpoints:** Specifies the number of write address watchpoints XSDB can set.
- **Number of Read Address Watchpoints:** Specifies the number of read address watchpoints XSDB can set.

RECOMMENDED: *For **Number of PC Breakpoints** and **Number of Read Address Watchpoints**, it is recommended that these two options be set to 0 if you are not using watchpoints for debugging.*

Interface

This option is only available when using Advanced Mode.

- **MicroBlaze Debug Connection**: Select the type of interface for connecting the MicroBlaze Debug Module (MDM).
  - **SERIAL** is the default JTAG interface, which is generally recommended and uses the least amount of resources.
  - **PARALLEL** provides synchronous parallel access to MicroBlaze debug registers, with better performance and timing.
  - **AXI** is a subset of PARALLEL, providing an AXI4-Lite interface that can be connected through AXI register slices or AXI clock converters to further improve timing.

Performance Monitoring

With extended debugging, MicroBlaze provides the following performance monitoring counters to count various events and to measure latency during program execution:

- C_DEBUG_EVENT_COUNTERS: Configures the event counters.
- C_DEBUG_LATENCY_COUNTERS: Configures the latency counters.
- C_DEBUG_COUNTER_WIDTH: Sets the counter width to 32, 48, or 64 bits.

With the default configuration, the counter width is set to 32 bits and there are five event counters and one latency counter.

Trace and Profiling

With extended debugging, MicroBlaze provides program trace, storing information in the embedded trace buffer (ETB) to enable program execution tracing. Users can also toggle the **Auto** switch and select the **External Trace** check box, if desired.

Use the parameter C_DEBUG_TRACE_SIZE to configure the size of the embedded trace buffer from 8KB to 128KB, or the external trace buffer from 32B to 8 KB.

RECOMMENDED: *It is recommended to always keep the external trace buffer set to 8KB, to avoid buffer overflow.*

By setting C_DEBUG_TRACE_SIZE to 0 (None), program trace is disabled.

Extended debugging also provides non-intrusive profiling, storing program execution statistics in a profiling buffer. The buffer is divided into a number of bins, each counting the number of executed instructions or clock cycles within a certain address range.

Use the parameter C_DEBUG_PROFILE_SIZE to configure the size of the profiling buffer from 4K to 128K. By setting the parameter to 0 (None), profiling is disabled.

MicroBlaze Configuration Wizard: Buses Page

Local Memory Bus Interfaces

- **Enable Local Memory Bus Instruction Interface**: Enables LMB instruction interface. When this instruction is set as shown in Other Interfaces, the Local Memory Bus (LMB) instruction interface is available.

  A typical MicroBlaze system uses this interface to provide fast local memory for instructions. Normally, it connects to an LMB bus using an LMB Bus Interface Controller to access a common block RAM.

- **Enable Local Memory Bus Data Interface**: Enables LMB data interface. When this parameter is set, the local memory bus (LMB) data interface is available. A typical MicroBlaze system uses this interface to provide fast local memory for data and vectors. Normally, it connects to an LMB bus using an LMB Bus Interface Controller to access a common block RAM. With 64-bit MicroBlaze, there is also an option to set the Data Width to 32-bit or 64-bit.

AXI and ACE Interfaces

- **Select Bus Interface**: When this parameter is set to **AXI**, then AXI is selected for both peripheral and cache access. When this parameter is set to **ACE**, then AXI is selected for peripheral access and ACE is selected for cache access, providing cache coherency support.

  Note: To be able to use ACE, area optimization, write-back data cache, instruction cache streams, or victims cache data widths other than 32-bit must not be set. You must set **Use Cache for All Memory Accesses** for both caches.

- **Enable Peripheral AXI Interface Instruction Interface**: When this parameter is set, the peripheral AXI4-Lite instruction interface is available. In many cases, this interface is not needed, in particular if the Instruction Cache is enabled and `C_ICACHE_ALWAYS_USED` is set.
- **Enable Peripheral AXI Data Interface**: When this parameter is set, the peripheral AXI data interface is available. This interface usually connects to peripheral I/O using AXI4-Lite, but it can be connected to memory also. If you enable exclusive access, the AXI4 protocol is used. With 64-bit MicroBlaze, there is also an option to set the Data Width to 32-bit or 64-bit.

Stream Interfaces

- **Number of Stream Links**: Specifies the number of pairs of AXI4-Stream link interfaces. Each pair contains a master and a slave interface. The interface provides a unidirectional, point-to-point communication channel between MicroBlaze and a hardware accelerator or co-processor. This is a low-latency interface, which provides access between the MicroBlaze register file and the FPGA fabric.

Other Interfaces

- **Enable Trace Bus Interface**: When this parameter is set, the Trace bus interface is available. This interface is useful for debugging, execution statistics and performance analysis. In particular, connecting interface to a ChipScope Logic Analyzer (ILA) allows tracing program execution with clock cycle accuracy.

  The MicroBlaze Trace interface can be used to view the processor software execution in simulation and in hardware. It is sufficient to enable the interface without actually connecting it, to get access to the signals in simulation, and to add them to an ILA in hardware.

  The waveform can be related to the assembler and source code by looking at the executable object dump. In the Vitis software platform this can be viewed by double-clicking on the generated ELF file. It is also possible to generate an object dump from the ELF file with interspersed source code using the `mb-objdump` command. The `Trace_PC` and `Trace_Instruction` signals correspond to the address and instruction in the object dump. Note that these, and most other signals, are only valid when `Trace_Valid_Instr` is set.

Memory access addresses are shown using the `Trace_Data_Address` signal, which is valid when either `Trace_Data_Read` or `Trace_Data_Write` is set. Instruction results are written to a MicroBlaze destination register indicated by `Trace_Reg_Addr` when the `Trace_Reg_Write` signal is set, with the value shown by the `Trace_New_Reg_Value` signal.

The `Trace_Exception_Kind` signal, valid when `Trace_Exception_Taken` is set, indicates interrupts, breaks and exceptions. This can be useful to find error conditions or interrupt related issues.

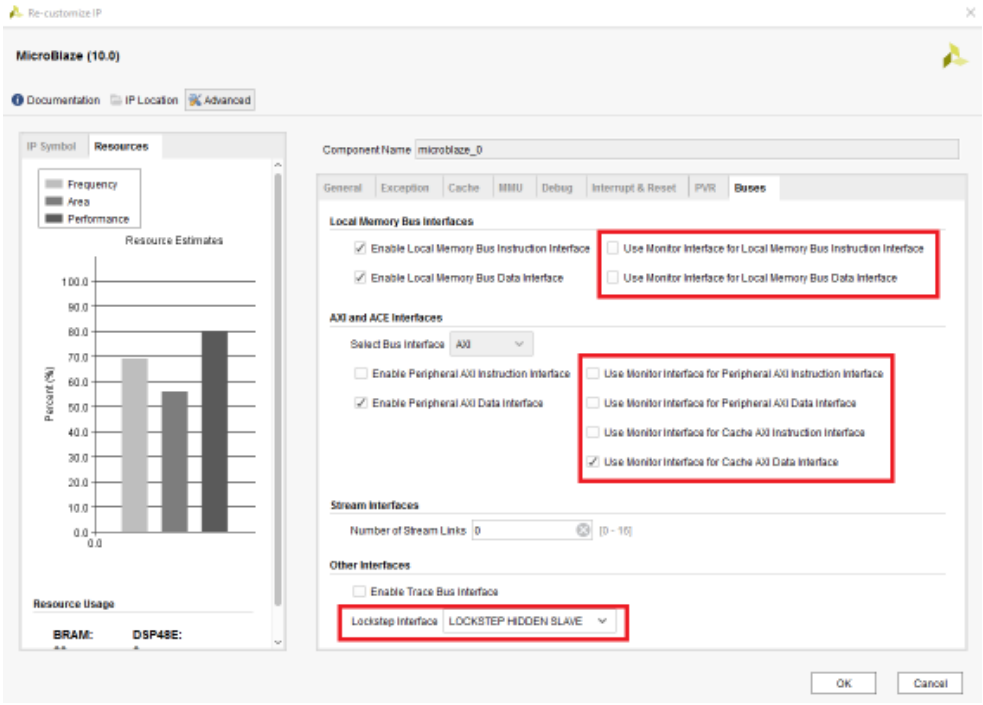
For a complete description of all the Trace bus interface signals, see Chapter 3, "Trace Interface Description" in *MicroBlaze Processor Reference Guide* (UG984).

- **Lockstep Interface**: When you enable lockstep support, two MicroBlaze cores run the same program in lockstep, and you can compare their outputs to detect errors.
  - When set to NONE, no lockstep interfaces are enabled.
  - When set to LOCKSTEP_MASTER, it enables the `Lockstep_Master_Out` and `Lockstep_Out` output ports.
  - When set to LOCKSTEP_SLAVE, it does the following:
    - Enables the `Lockstep_Slave_In` input port and `Lockstep_Out` output ports.
    - Sets the `C_LOCKSTEP_SLAVE` parameter to 1.

    The slave processor is visible as a CPU, and can have private LMB memory.
  - LOCKSTEP_HIDDEN_SLAVE behaves the same way as LOCKSTEP_SLAVE, except that the slave processor is not visible as a CPU. This setting is recommended, *except* when using private LMB memory.

  When this option is enabled, additional options become available under the Local Memory Bus Interfaces and AXI and ACE Interfaces section as shown in Other Interfaces. These options are explained below.

Figure 13: **MicroBlaze Configuration Wizard Buses Page**



- **Use Monitor Interface for Local Memory Bus Instruction Interface**: Select Monitor Interface for LMB instruction interface. This can be used to simplify connection of LMB for a lockstep slave processor when private LMB memory is not used.
- **Use Monitor Interface for Local Memory Bus Data Interface**: Select Monitor Interface for LMB data interface. This can be used to simplify connection of LMB for a lockstep slave processor when private LMB memory is not used.
- **Use Monitor Interface for Peripheral AXI Instruction Interface**: Select Monitor Interface for AXI peripheral data interface. This can be used to simplify connection of AXI for a lockstep slave processor.
- **Use Monitor Interface for Peripheral AXI Data Interface**: Select Monitor Interface for AXI peripheral data interface. This can be used to simplify connection of AXI for a lockstep slave processor.
- **Use Monitor Interface for Cache AXI Instruction Interface**: Select Monitor Interface for AXI cache instruction interface. This can be used to simplify connection of AXI for a lockstep slave processor.
- **Use Monitor Interface for Cache AXI Data Interface**: Select Monitor Interface for AXI cache data interface. This can be used to simplify connection of AXI for a lockstep slave processor.
- **Temporal Depth**: Allow configuration of temporal lockstep clock cycle delay of the slave processor.

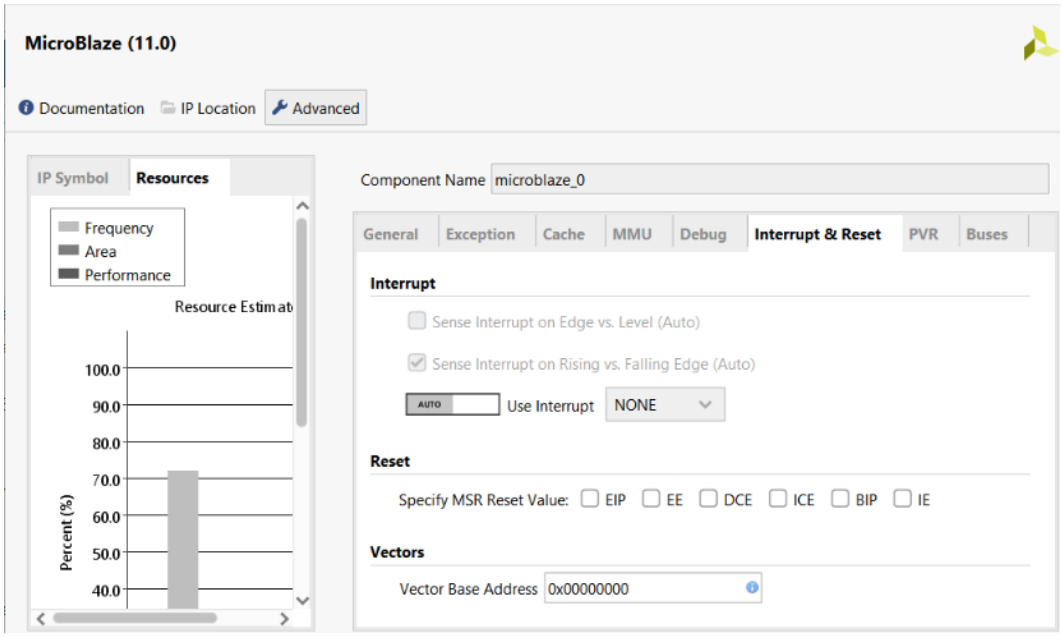
There is also a monitor option for interrupt on the **Interrupt & Reset** tab:

Use Monitor Interface for Interrupt: Select Monitor Interface for the interrupt interface. This can be used to simplify connection of interrupt for a lockstep slave processor when a common interrupt source is used.

MicroBlaze Configuration Wizard: Advanced Mode

Accessible through the Advanced button on the Welcome page of the MicroBlaze Configuration wizard, the Advanced mode provides a tabbed interface that lets you interact directly with the various configuration options. The following figure shows the Advance Mode Interrupt and Reset options.

Figure 14: **Advanced Mode: Interrupt and Reset Tab**



The tabbed interface of the Advanced mode provides access to each of the pages of the MicroBlaze Configuration wizard as follows:

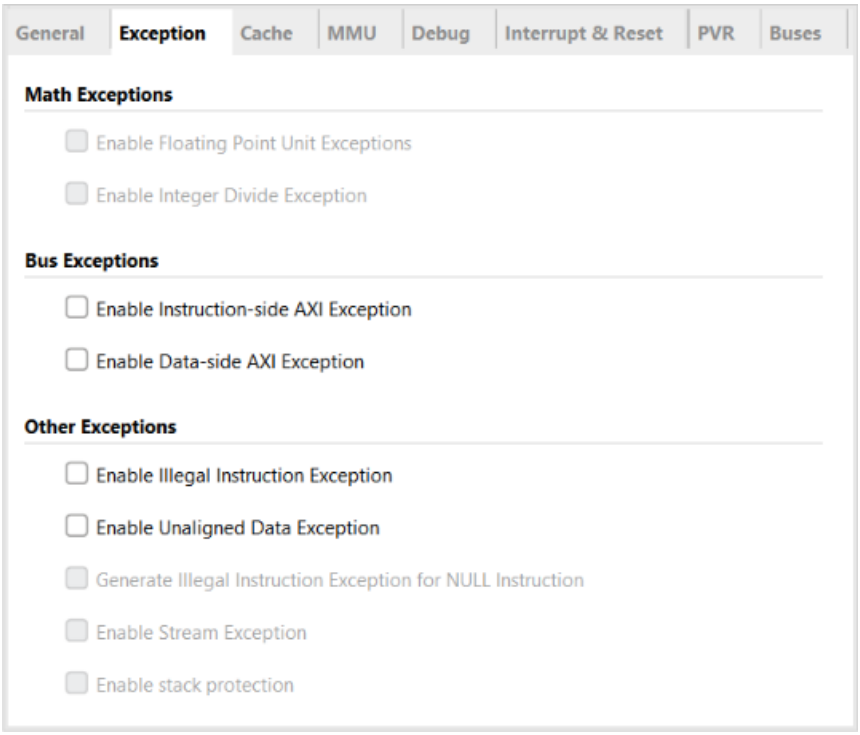
- MicroBlaze Configuration Wizard: General Page
- MicroBlaze Configuration Wizard: Cache Page
- MicroBlaze Configuration Wizard: MMU Page
- MicroBlaze Configuration Wizard: Debug Page
- MicroBlaze Configuration Wizard: Buses Page

In addition, the **Exception**, **Interrupt & Reset** and **PVR** tabs are only available through the Advanced mode interface.

MicroBlaze Advanced Mode Exception Tab

The following figure shows the MicroBlaze Exception options page.

Figure 15: **Exception Options in the MicroBlaze Configuration Wizard**



IMPORTANT! *You must provide your own exception handler.*

Math Exceptions

- **Enable Floating Point Unit Exceptions**: Enables exceptions generated by the floating point unit (FPU). The FPU throws exceptions for all of the IEEE standard conditions: underflow, overflow, divide-by-zero, and illegal operations. In addition, the MicroBlaze FPU throws a de-normalized operand exception.

  IMPORTANT! *To select this option, ensure to enable* ***Enable Floating Point Unit*** *in MicroBlaze Configuration Wizard: General Page.*

- **Enable Integer Divide Exception**: Causes an exception if the divisor (rA) provided to the `idiv` or `idivu` instruction is zero, or if an overflow occurs for `idiv`.

  IMPORTANT! *To select this option, ensure to enable* ***Enable Integer Multiplier*** *in MicroBlaze Configuration Wizard: General Page.*

Bus Exceptions

- **Enable Instruction-side AXI Exception**: Causes an exception if there is an error on the instruction-side AXI bus.
- **Enable Data-side AXI Exception**: Causes an exception if there is an error on the data-side AXI bus.

Other Exceptions

- **Enable Illegal Instruction Exception**: Causes an exception if the major opcode is invalid.
- **Enable Unaligned Data Exception**: When enabled, the tools automatically insert software to handle unaligned accesses.
- **Generated Illegal Instruction Exception for NULL Instructions**: MicroBlaze compiler does not generate, nor do Vitis libraries use the NULL instruction code (`0x00000000`). This code can only exist legally if it is hand-assembled. Executing a NULL instruction normally means that the processor has jumped outside the initialized instruction memory.

  If C_OPCODE_0x0_ILLEGAL is set, MicroBlaze traps this condition; otherwise, it treats the command as a `NOP`. This setting is only available if you have enabled Illegal Instruction Exception.
- **Enable Stream Exception**: Enables stream exception handling for Advanced eXtensible Interface (AXI) read accesses.

  IMPORTANT! *You must enable additional stream instructions to use stream exception handling.*
- **Enable Stack Protection**: Ensures that memory accesses using the stack pointer (R1) to ensure they are within the limits set by the stack low register (SLR) and stack high register (SHR). If the check fails with exceptions enabled, a stack protection violation exception occurs. The Xilinx System Debugger (XSDB) also reports if the check fails.

MicroBlaze Advanced Mode Interrupt & Reset Tab

MicroBlaze Configuration Wizard: Advanced Mode shows the Interrupt & Reset tab of the MicroBlaze Configuration wizard.

Interrupt

- **Sense Interrupt on Edge vs. Level (Auto)**: Specifies whether the MicroBlaze processor senses interrupts on edge or level. This option is automatically set based on the connected interrupt controller.
  - If this parameter is enabled, then MicroBlaze only detects an interrupt on the edge specified by `C_EDGE_IS_POSITIVE`. This option is automatically set based on the connected interrupt controller.
  - If this parameter is disabled, whenever the interrupt is high an interrupt will be triggered.

If an interrupt is generated and handled while the interrupt input remains high, another interrupt will be generated.

- **Sense Interrupt on Rising vs. Falling Edge (Auto)**: Specifies whether the MicroBlaze processor detects interrupts on the rising or falling edges if `C_INTERRUPT_IS_EDGE` is set to 1.
- **Use Interrupt**: Specifies whether the MicroBlaze processor interrupt input is enabled. Selecting **NORMAL** enables interrupts. Selecting **FAST** also enables low-latency interrupt handling.

Reset

- **Specify MSR Reset Value**: Specify reset value for select MSR bits.
  - Setting ICE (`=0x0020`) enables instruction cache at reset.
  - Setting DCE (`=0x0080`) enables data cache at reset.
  - Setting EIP (`=0x0200`) sets exception in progress at reset.
  - Setting EE (`=0x0100`) enables exceptions at reset.
  - Setting BIP (`=0x0008`) sets break in progress at reset.
  - Setting IE (`=0x0002`) enables interrupts at reset.

TIP: *Enabling caches at reset will allow execution to start immediately from external memory and can thus be used to reduce or eliminate the need for LMB memory.*

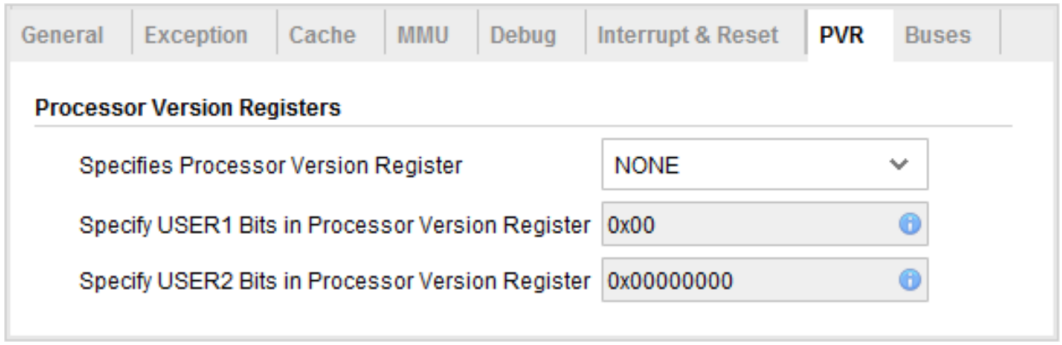
Vectors

- **Vector Base Address**: Change the base address used for MicroBlaze vectors. This affects the vectors for Reset, User Vector, Interrupt, Break, and Hardware Exception. See the *MicroBlaze Processor Reference Guide* (UG984) for more information. Normally the base address is `0x00000000` in Local Memory, but if this address is used for other purposes, this parameter allows the vectors to be moved to another address. The 7 least significant bits (LSBs) in the address must be zero.

MicroBlaze Advanced Mode PVR Tab

The following figure shows the PVR tab of the MicroBlaze Configuration wizard. See the *MicroBlaze Processor Reference Guide* (UG984) for more information on Processor Version Register (PVR).

Figure 16: **MicroBlaze Configuration Wizard PVR Page**



Processor Version Registers

- **Specifies Processor Version Register**: PVR options are, as follows:
  - **None** (0): The default, disables the PVR.
  - **Basic** (1): Enables only the first PVR.
  - **Full** (2): Enables all PVRs.
- **Specifies USER1 Bits in Processor Version Register**: This parameter specifies the `USER1` bits, 24 through 31, in the PVR. This parameter is only used if **C_PVR** is set to **Basic** (1) or **Full** (2).
- **Specifies USER2 Bits in Processor Version Register**: This parameter specifies the value of the second processor version register (USER2). This parameter is only used if **C_PVR** is set to **Full** (2).

Cross-Trigger Feature of MicroBlaze Processors

With basic debugging, cross trigger support is provided by two signals: `DBG_STOP` and `MB_Halted`.

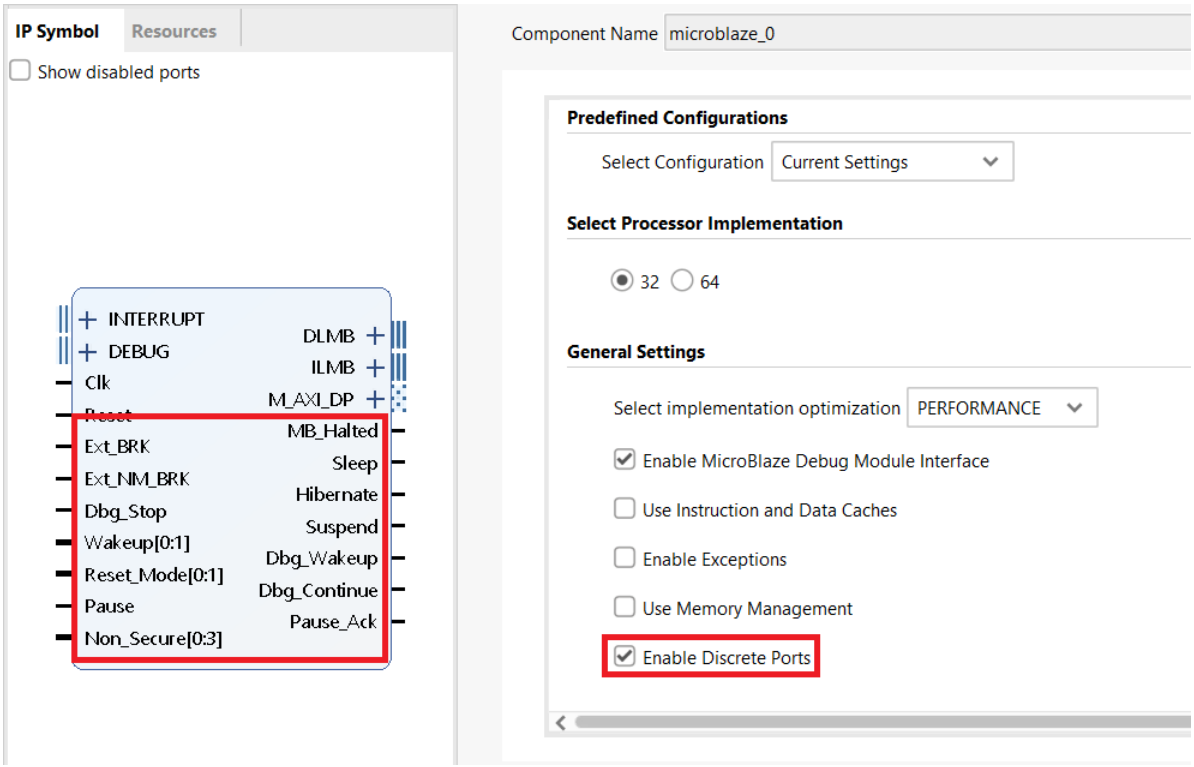
- When the `DBG_STOP` input is set to 1, MicroBlaze halts after a few instructions. XSDB detects that MicroBlaze has halted, and indicates where the halt occurred. The signal can be used to halt MicroBlaze processors at any external event, such as when an Integrated Logic Analyzer (ILA) is triggered.
- The `MB_Halted` output signal is set to 1 whenever the MicroBlaze processor is halted, such as after a breakpoint or watchpoint is hit, after a stop XSDB command, or when the `DBG_STOP` input is set.

  The output is cleared when MicroBlaze execution is resumed by an XSDB command.

IMPORTANT! *The `DBG_STOP` and `MB_Halted` pins are hidden. To see those pins, you must enable the **Enable Discrete Ports** option on the Welcome page of the MicroBlaze Configuration dialog box, as shown in the following figure.*

You can use the `MB_Halted` signal to trigger an Integrated Logic Analyzer (ILA), or halt other MicroBlaze cores in a multiprocessor system by connecting the signal to their `DBG_STOP` inputs. The following figure shows the discrete port and the Enable Discrete Ports check box.

Figure 17: **Enable Discrete Ports Option**

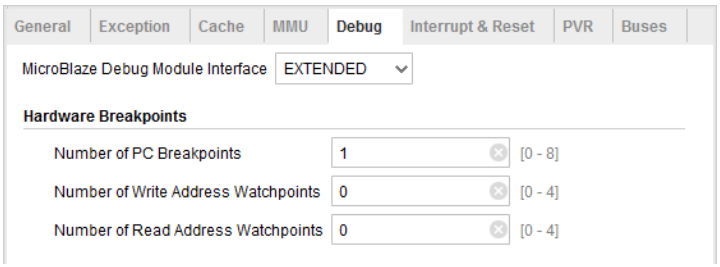


With extended debugging, cross trigger support is available in conjunction with the MDM. The MDM provides programmable cross triggering between all connected processors, as well as external trigger inputs and outputs. For details, see the *MicroBlaze Debug Module (MDM) LogiCORE IP Product Guide* (PG115).

To enable extended debug, set the MicroBlaze Debug Module Interface to EXTENDED in the Debug Page of the MicroBlaze Configuration Wizard as shown in the following figure.

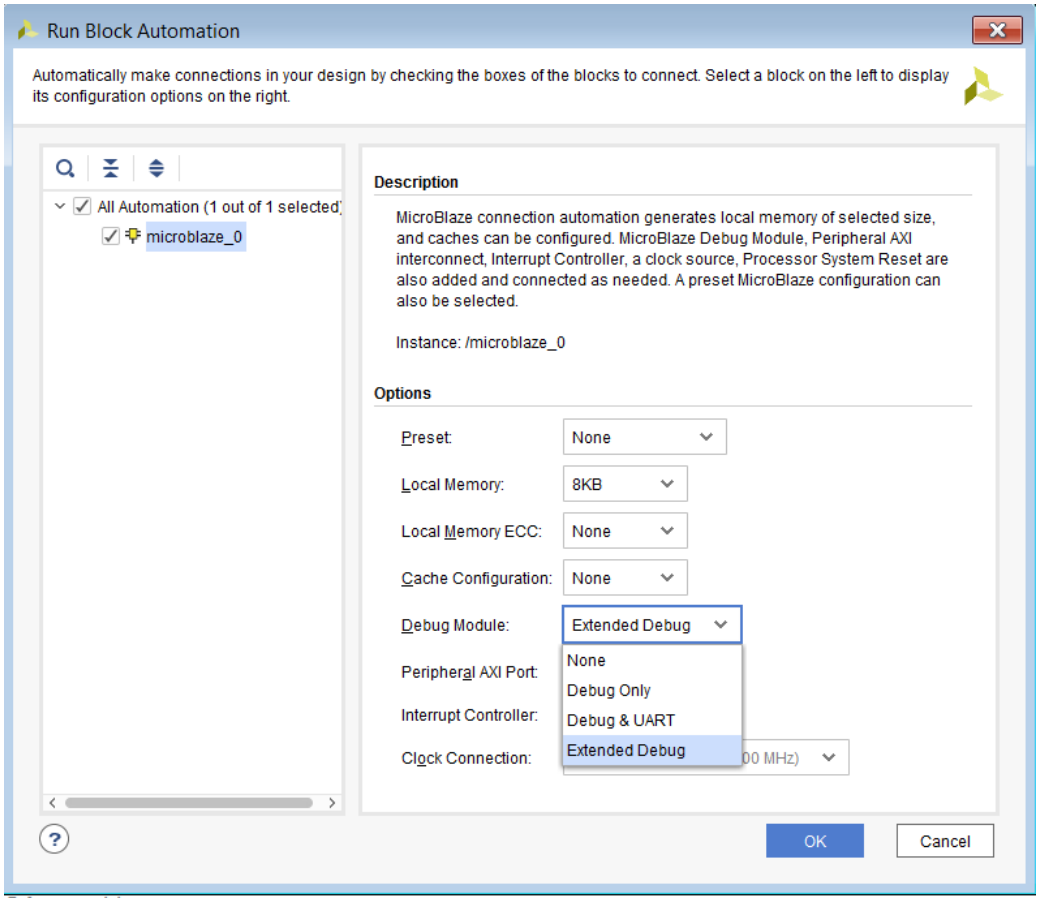
MicroBlaze can handle up to eight cross trigger actions. Cross trigger actions are generated by the corresponding MDM cross trigger outputs, connected using the Debug bus, shown in the following figure.

Figure 18: **Enable EXTENDED Debug for MicroBlaze**



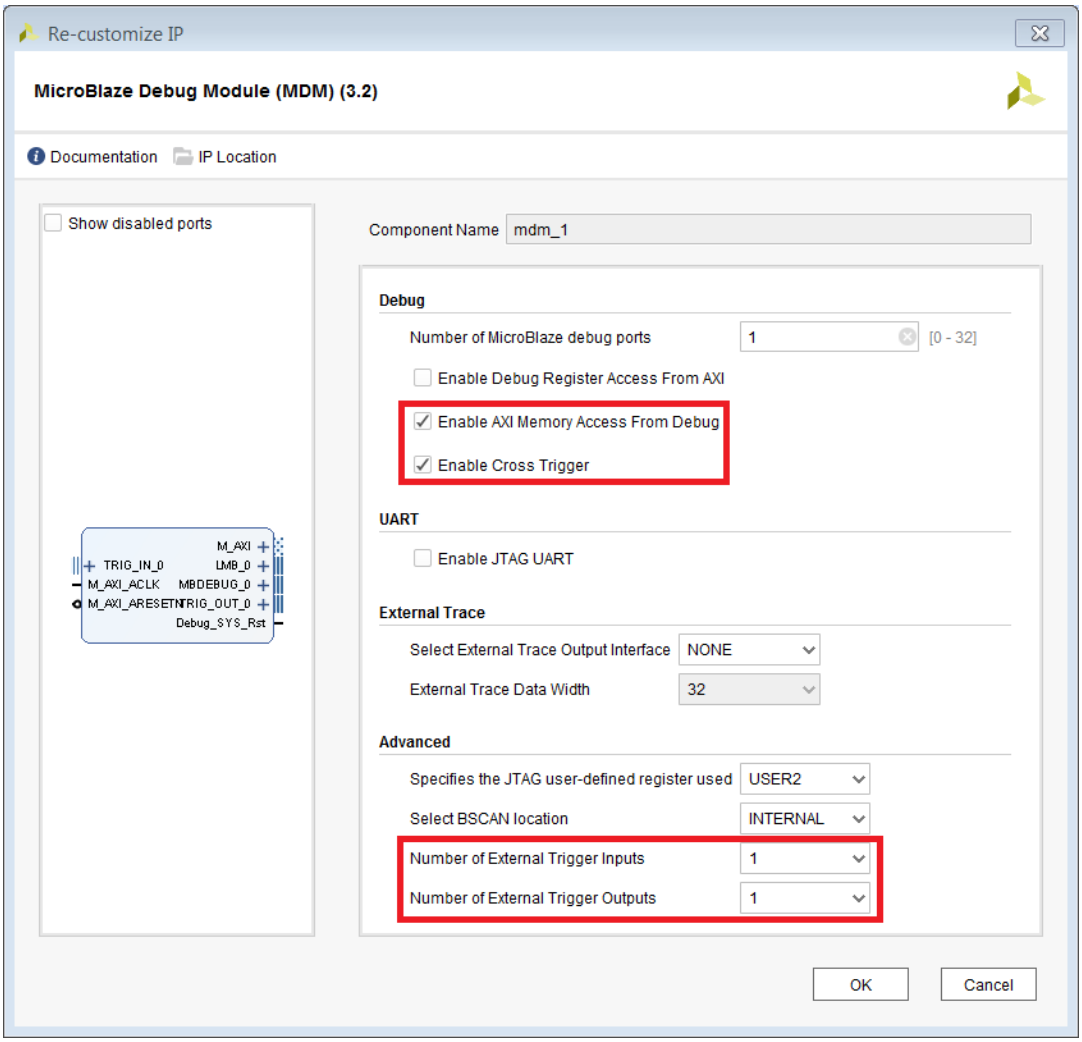
You can also set the Extended Debug option when running Block Automation for the MicroBlaze processor, as shown in the following figure.

Figure 19: **Extended Debug Option**



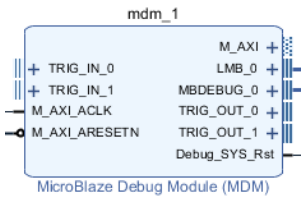
The MicroBlaze Debug Module (MDM) configuration dialog box can be opened by double-clicking on the MDM instance, as shown in the Figure 25: Using Connection Automation Feature of IP Integrator. In the MDM configuration dialog box, the **Enable Cross Trigger** check box is enabled, as highlighted in the following figure.

Figure 20: **Enable Cross Trigger Check Box in MDM**



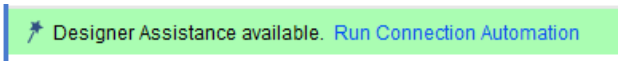
You can also select up to four external trigger inputs and external trigger outputs. When enabled, the block design updates to show the MDM details, as shown in the following figure.

Figure 21: **MDM in Block Design After Enabling Cross Trigger**



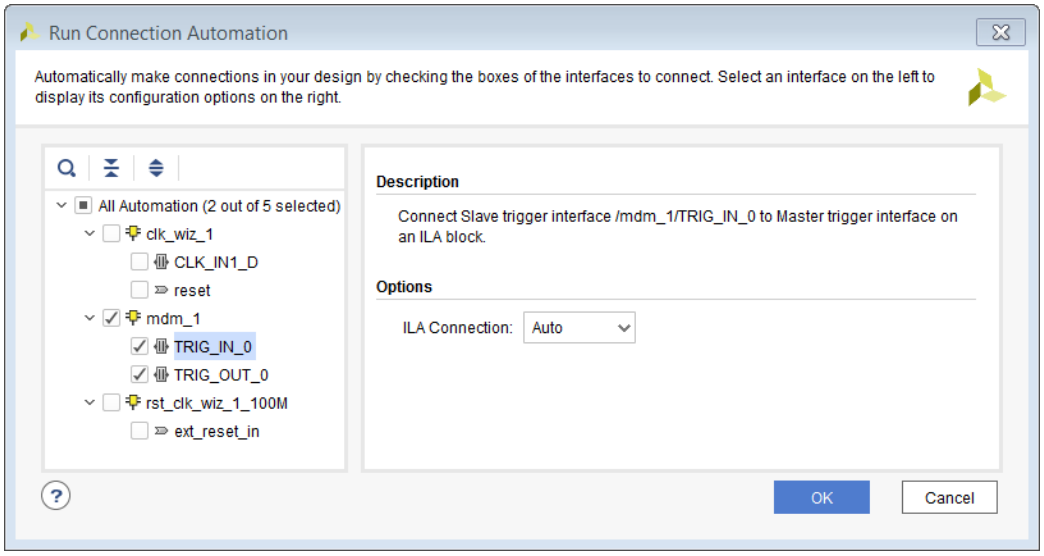
Next, run Connection Automation, shown in the following figure, to connect the cross trigger signals to an ILA.

Figure 22: **Connecting the TRIG_IN_0 Interface Pin to an ILA**



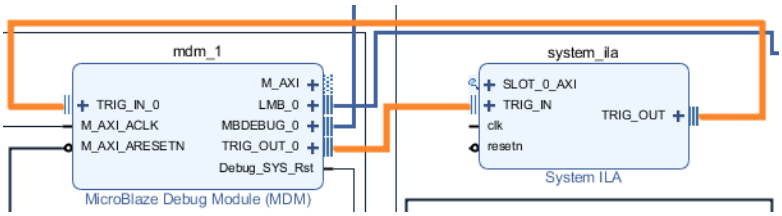
Leaving the settings of **Auto**, as shown in the following figure, on both `TRIG_IN_0/TRIG_OUT_0` in the Run Connection Automation dialog box, instantiates a new ILA and connects the `TRIG_IN_0/TRIG_OUT_0` signal of the MDM to the corresponding pin of the System ILA.

Figure 23: **Run Connection Automation Confirmation Dialog Box**



The following figure show the resulting block design.

Figure 24: **Block Design After Connecting Cross Trigger Pins to the ILA**



Completing Connections

After you have configured the MicroBlaze processor, you can start to instantiate other IP that constitutes your design.

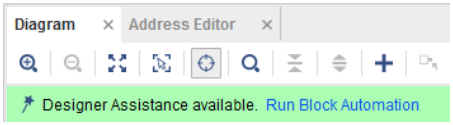
In the IP integrator canvas, right-click and select **Add IP**. You can use two built-in features of the IP integrator to complete the rest of the IP subsystem design: the Block Automation and Connection Automation features assist you with putting together a basic microprocessor system in the IP integrator tool and/or connecting ports to external I/O ports.

Block Automation

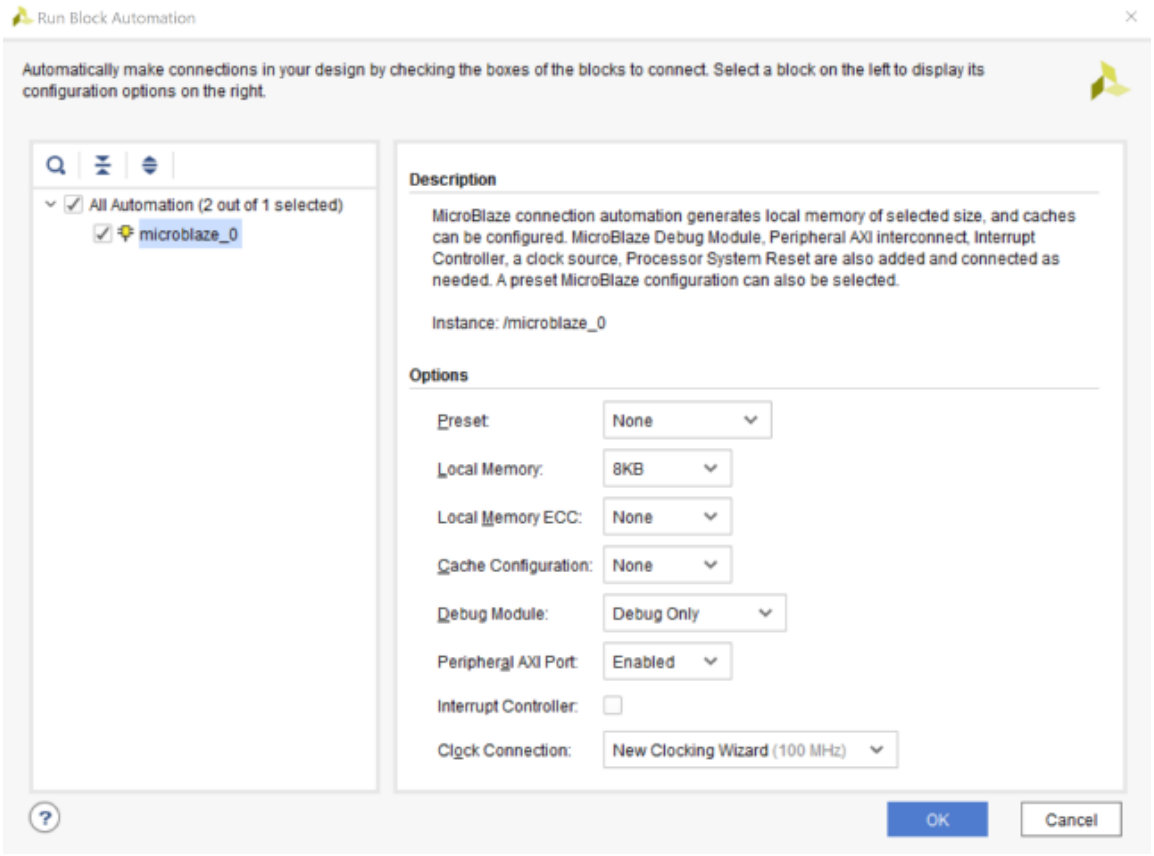
The Block Automation feature is available when you instantiate a microprocessor in the block design of the IP integrator tool.

***Note*:** The block design must specify a part or board that uses a specific processor to make that processor accessible through the IP catalog.

1. Click **Run Block Automation** to get assistance with putting together a simple MicroBlaze System.



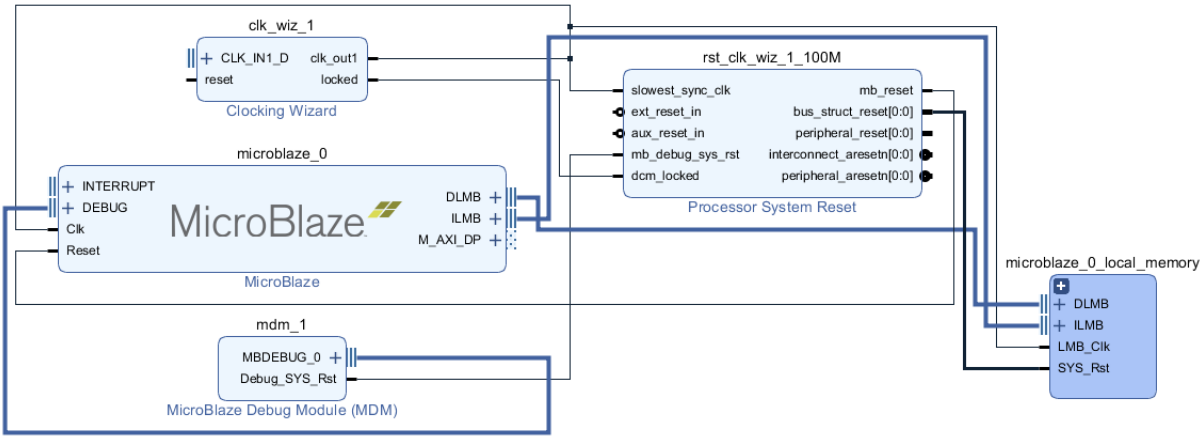
The Run Block Automation dialog box lets you provide input about basic features that the microprocessor system requires. The following figure shows the Block Automation dialog box.



The MicroBlaze Preset option provides a convenient way of configuring the processor settings according to the particular use case: microcontroller, real-time, or application. If necessary, further configuration can be done in the MicroBlaze Configuration wizard.

2. Select the required options and click **OK**.

   The Run Block Automation creates the following MicroBlaze system.

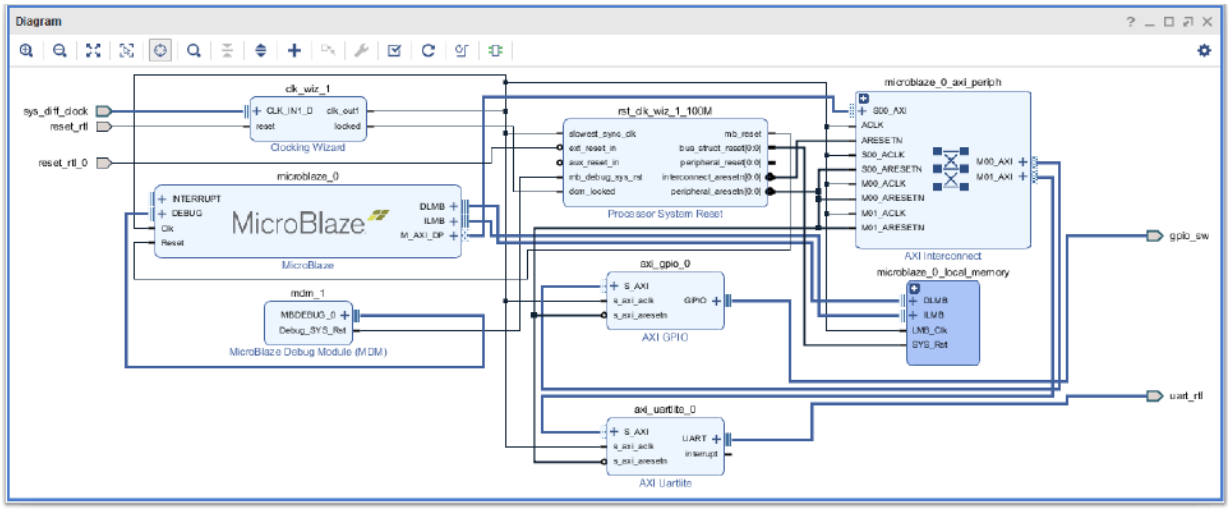


Using Connection Automation

When the IP integrator tool determines that a potential connection exists among the instantiated IP in the canvas, it opens the Connection Automation feature.

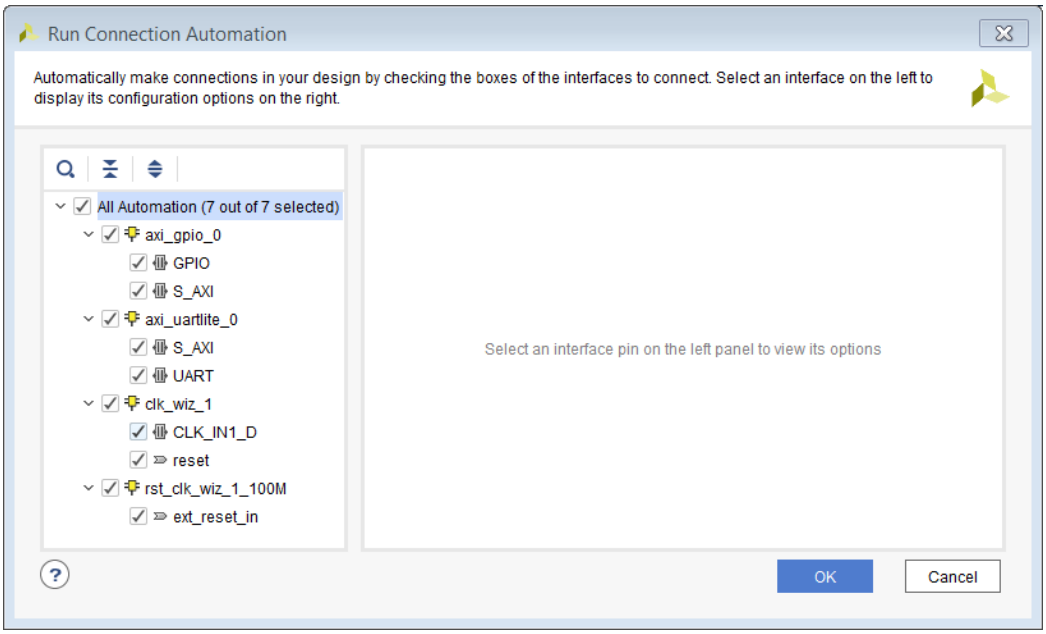
In the following figure, two IP, the GPIO and the UARTLite, are instantiated along with the MicroBlaze processor subsystem.

Figure 25: **Using Connection Automation Feature of IP Integrator**



When you click the **Run Connection Automation** link, the following dialog box, shown in the following figure, opens.

Figure 26: **The Run Connection Automation Dialog Box**

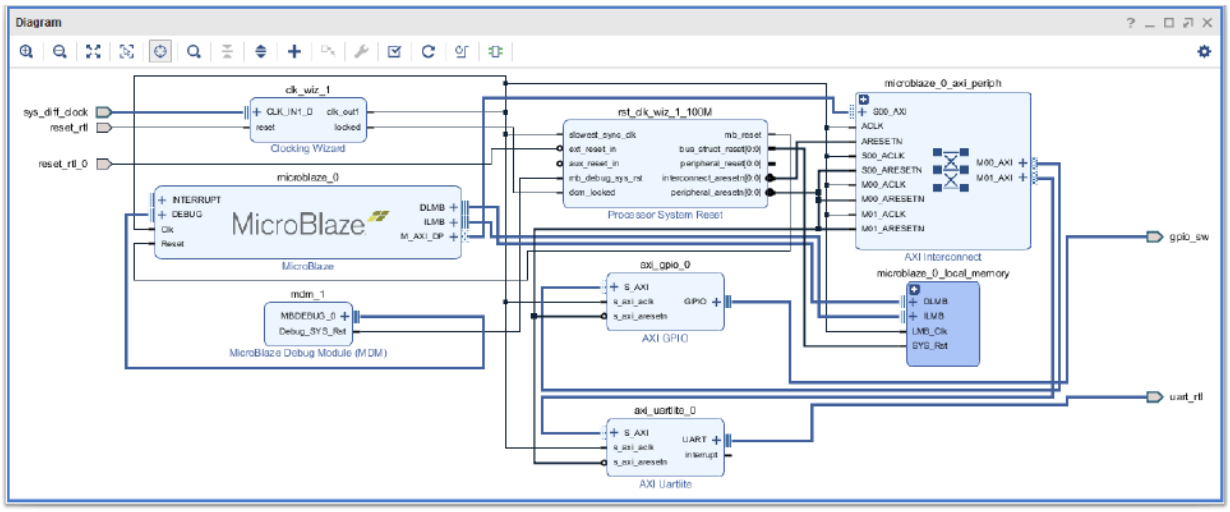


The IP integrator determines that there is a potential connection for the following objects:

- The Proc Sys Rst IP `ext_reset_in` pin must connect to a reset source, which can be either an internal reset source or an external input port.
- The Clocking Wizard `CLK_IN_1_D` pin must connect to either an internal clock source or an external input port.
- The AXI GPIO `s_axi` interface must connect to a master AXI interface.
- The AXI GPIO core `gpio` interface must connect to external I/Os.
- The Uartlite IP `s_axi` interface must connect to a master AXI interface.
- The Uartlite IP `uart` interface must connect to external I/Os.

When you run connection automation on each of those available options, the block design looks as shown in the following figure.

Figure 27: **Running Connection Automation for a Sample MicroBlaze Design**



Completing the Design

See the following sections for common considerations in an embedded design:

- Platform Board Flow in IP Integrator
- Making Manual Connections in a Design
- Manually Creating and Connecting to I/O Ports
- Memory-Mapping in the Address Editor
- Running Design Rule Checks
- Integrating a Block Design in the Top-Level Design

MicroBlaze Processor Constraints

The IP integrator generates constraints for IP generated within the tool during output products generation; however, you must generate constraints for any custom IP or higher-level code.

A constraint set is a set of XDC files that contain design constraints, which you can apply to your design. There are two types of design constraints:

- Physical constraints define pin placement, and absolute, or relative placement of cells such as: block RAMs, LUTs, Flip-Flops, and device configuration settings.
- Timing constraints, written in industry standard SDC, define the frequency requirements for the design. Without timing constraints, the Vivado Design Suite optimizes the design solely for wire length and routing congestion.

  ***Note*:** Without timing constraints, Vivado implementation makes no effort to assess or improve the performance of the design.

IMPORTANT! *The Vivado Design Suite does not support UCF format. For information on migrating UCF constraints to XDC commands see the ISE to Vivado Design Suite Migration Guide (UG911) for more information.*

The options on how to use constraint sets, are, as follows:

- Multiple constraints files within a constraint set.
- Constraint sets with separate physical and timing constraint files.
- A master constraints file, and direct design changes to a new constraints file.
- Multiple constraint sets for a project, and make different constraint sets active for different implementation runs to test different approaches.
- Separate constraint sets for synthesis and for implementation.
- Different constraint files to apply during synthesis, simulation, and implementation to help meet your design objectives.

Separating constraints by function into different constraint files can make your overall constraint strategy more clear, and facilitate being able to target timing and implementation changes.

Organizing design constraints into multiple constraint sets can help you do the following:

- Target different Xilinx FPGAs for the same project. Different physical and timing constraints could be necessary for different target parts.
- Perform "what-if" design exploration. Using constraint sets to explore different scenarios for floorplanning and over-constraining the design.
- Manage constraint changes. Override master constraints with local changes in a separate constraint file.

TIP: *A good way to validate the timing constraints is to run the* `report_timing_summary` *command on the synthesized design. Problematic constraints must be addressed before implementation.*

For more information on defining and working with constraints that affect placement and routing, see the *Vivado Design Suite User Guide: Using Constraints* (UG903).

Taking the Design through Synthesis, Implementation, and Bitstream Generation

After you complete the design and constrain it appropriately, you can run synthesis and implementation, and then you can generate a bitstream.

Exporting Hardware to the Vitis Integrated Design Environment

See Using the Vitis Software Platform for more information. In general, after you generate the bitstream for your design, you are ready to export your hardware definition to the Vitis software platform.

This action exports the necessary files needed for the Vitis software platform to understand the IP used in the design and also the memory mapping from the perspective of the processor. After a bitstream is generated and the design is exported, you can then download the device and run the software on the processor.

TIP: *If you want to start software development before a bitstream is created, you can export the hardware definition after generating the design.*

For detailed procedure, see Exporting a Hardware Description.

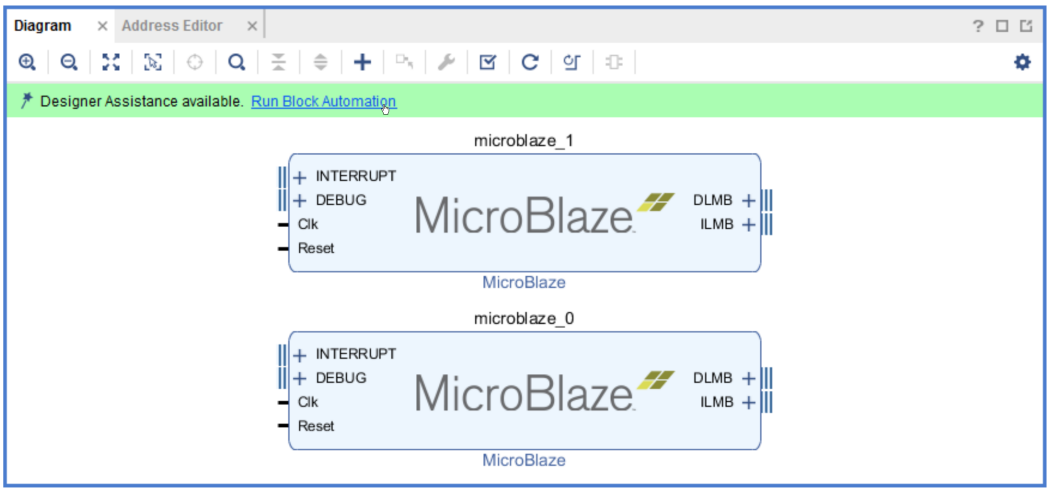
Multiple MicroBlaze Processor Designs

Multiple MicroBlaze processors can be included in a block design. The configurations of these multiple MicroBlaze designs may vary based on design needs. A simple dual MicroBlaze design is discussed in the following sections.

Instantiate MicroBlaze IP Cores

Create a block design and instantiate two instances of MicroBlaze IP as shown. Note that the Run Block Automation link becomes active in the banner.

Figure 28: **Multiple MicroBlaze Instances in a Block Design**



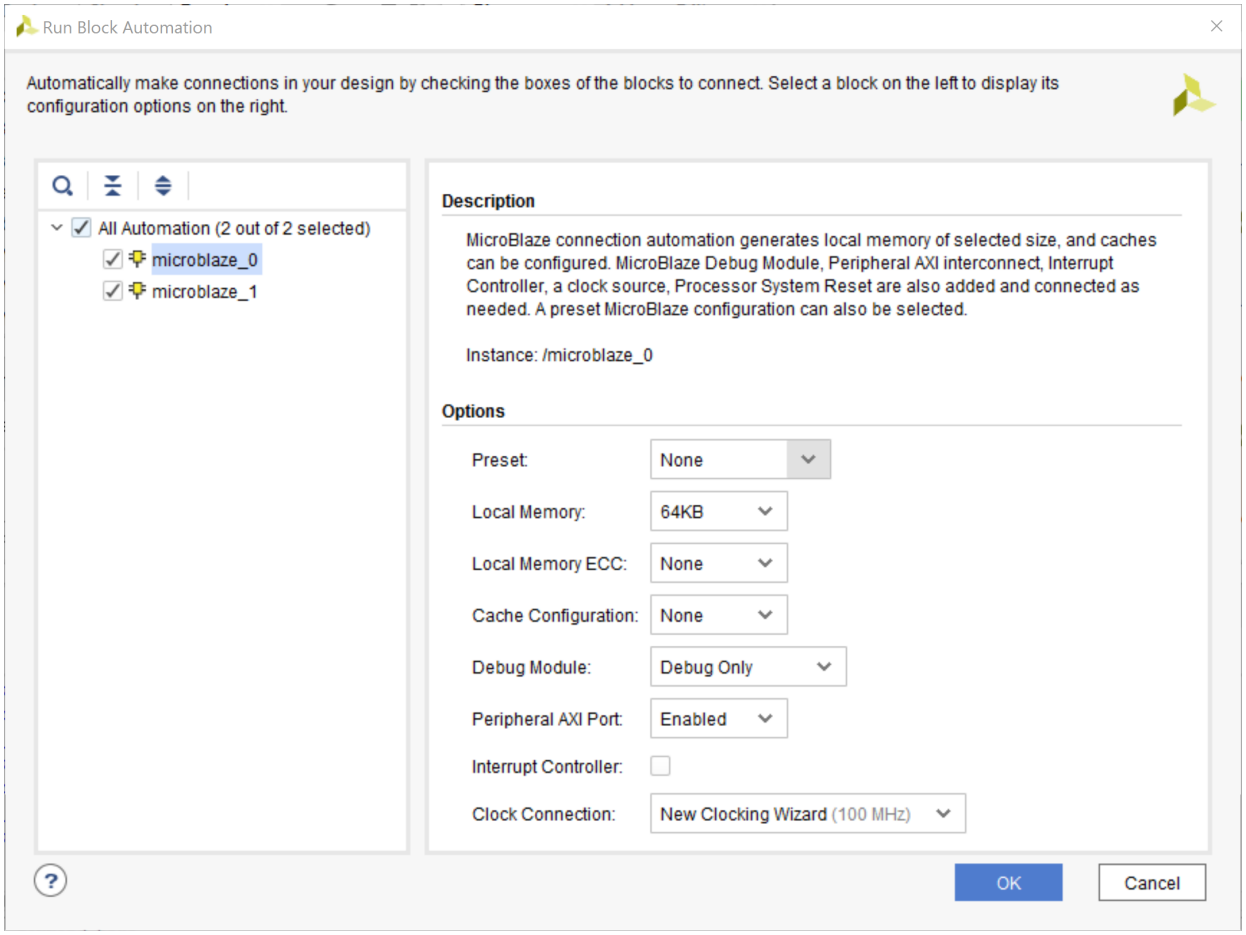
Click the **Run Block Automation** link to run block automation on both the MicroBlaze instances. Again, the options here varies on the design requirements.

For example:

- Both the MicroBlaze processors might run from a single system clock or they could be totally independent.
- They could share the Clocking Wizard IP or they could have independent Clocking Wizard IP.

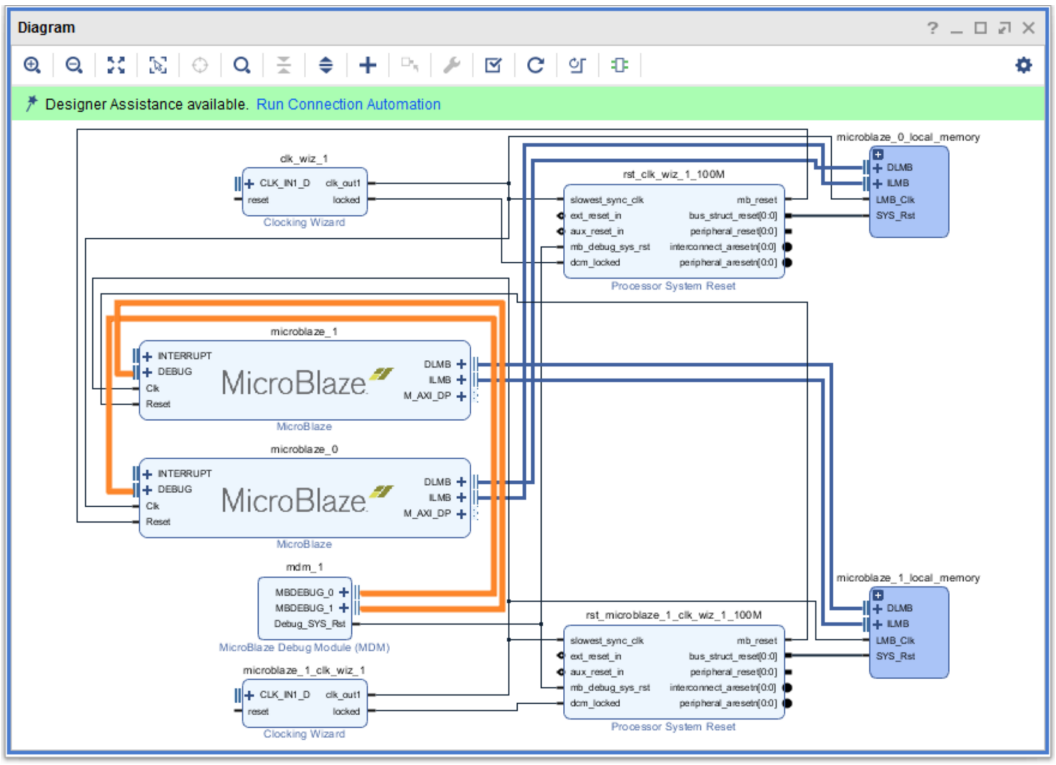
This topology shows two independent Clocking Wizard IP for each MicroBlaze processor as in the following figure.

Figure 29: **Block Automation Dialog Box for Dual MicroBlaze Design**



The block design looks as shown in the following figure:

Figure 30: **Block Design After Running Block Automation**



***Note*:** Both the MicroBlaze processors share the same MicroBlaze Debug Module that is automatically configured to support two debug interfaces.

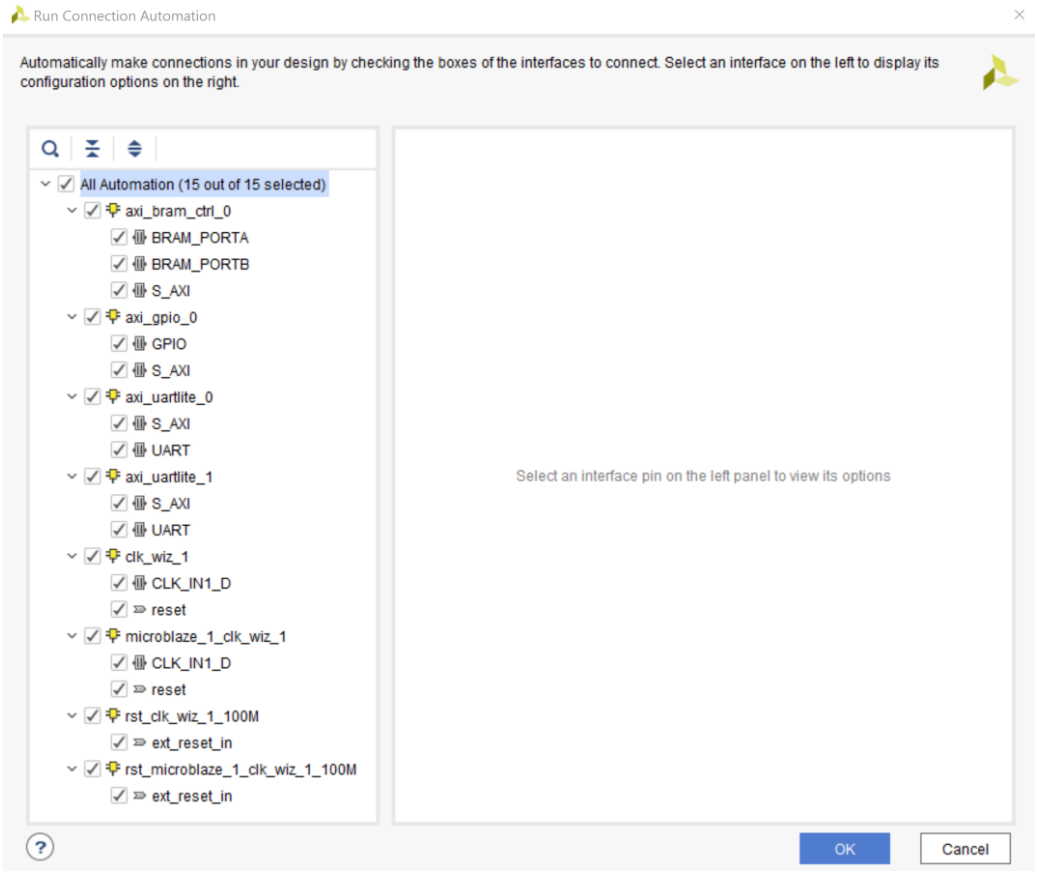
At this point you can add peripherals to your design as needed. In this case, two instances of AXI UART Lite, one GPIO and a AXI blcok RAM Controller were added.

- The AXI UART Lite IP is connected to each of the MicroBlaze processor instances.
- The GPIO is connected to one instance of the MicroBlaze IP.
- Finally, the AXI BRAM Controller controlling the Block Memory Generator is shared by both MicroBlaze processors.
- The input clock to one of the Clocking Wizard IP is the on-board System Differential Clock while the other Clocking Wizard is tied to the on-board PCIe Clock.

Run Connection Automation

***Note*:** The Run Connection Automation link is active at the top of the block design banner. Click **Run Connection Automation**. Check the **All Automation** check-box (15 out of 15 selected), as shown in the following figure.

Figure 31: **Connection Automation Dialog Box**



Make the selections listed in the following table for each automation.

Table 1: **Connection Automation Options**

Connection	More Information	Setting
axi_bram_ctrl_0 • BRAM_PORTA	The only option for this automation is to instantiate a new Block Memory Generator as shown under options.	Leave the `Blk_Mme_Gen` to its default option of Auto.
axi_bram_ctrl_0 • BRAM_PORTB	The Run Connection Automation dialog box gives you two choices: Instantiate a new BMG and connect the PORTB of the AXI BRAM Controller to the BMG IP. Use the previously instantiated BMG core and automatically configure it to be a true dual-ported memory and connected to PORTB of the AXI BRAM Controller.	Leave the `Blk_Mem_Gen` option to its default value of Auto.
axi_bram_ctrl_0 • S_AXI	The Master Field can be set to either `/microblaze_0` or `/microblaze_1`.	Leave it to the default value of `/microblaze_0`.

Table 1: **Connection Automation Options** *(cont'd)*

Connection	More Information	Setting
axi_gpio_0 • GPIO	The GPIO interface can be tied to several on-board interfaces.	Set the Selected Board Part Interface to led_8bits (LED).
axi_gpio_0 • S_AXI	The Master field is set to its default value of `/microblaze_0 (Periph)`. All other fields is set to its default value of Auto.	Keep the default settings.
axi_uartlite_0 • S_AXI	The Master field is set to its default value of `/microblaze_0 (Periph)`. All other fields is set to its default value of Auto.	Keep the default settings.
axi_uartlite_0 • UART	Set the Select Board Part Interface to the `rs232_uart` interface present on-board or tie it to a custom interface.	Keep the default setting of `rs232_uart (UART)`.
axi_uartlite_1 • S_AXI	The Master field is set to its default value of `/microblaze_1 (Periph)`. All other fields is set to its default value of Auto.	Keep the default settings.
axi_uartlite_0 • UART	The Select Board Part Interface can be set to the `rs232_uart` interface present on-board or can be tied to a custom interface.	Because you already used the `rs232_uart (UART)` interface on the board to connect to the `/uartlite_0` instance, set the Select Board Part Interface option to Custom.
clk_wiz_1 • CLK_IN1_D	The input clock source of the Clocking Wizard can be tied to the several on-board clock sources or it can be tied to a Custom input clock.	Leave the Select Board Part Interface field to `sys_diff_clock` (System differential clock).
clk_wiz_1 • reset	The reset pin of the Clocking Wizard can be tied to either the on-board reset source or to a custom input pin.	Leave the Select Board Part Interface to its default value of `reset` (FPGA Reset).
microblaze_1_clk_wiz_1 • CLK_IN1_D	The input clock source of the Clocking Wizard can be tied to the several on-board clock sources or it can be tied to a Custom input clock.	Leave the Select Board Part Interface field to New External Port (100 MHz).
microblaze_1_clk_wiz_1 • reset	The reset pin of the Clocking Wizard can be tied to either the on-board reset source or to a custom input pin.	Leave the Select Board Part Interface to its default value of `reset` (FPGA Reset).
rst_clk_wiz_1_100M • ext_reset_in	The reset pin of the Processor System Reset IP can be tied to either the on-board reset source or to a custom input pin.	Leave the Select Board Part Interface to its default value of `reset` (FPGA Reset).
rst_microblaze_1_clk_wiz_1_100M • ext_reset_in	The reset pin of the Processor System Reset IP can be tied to either the on-board reset source or to a custom input pin.	Leave the Select Board Part Interface to its default value of `reset` (FPGA Reset).

After running connection automation, one instance of the Microblaze (`microblaze_0`) is connected to three slaves AXI block RAM Controller (`axi_bram_ctrl_0`), AXI Uartlite (`axi_uartlite_0`) and AXI GPIO (`axi_gpio_0`). The other instance of MicroBlaze (`microblaze_1`) is connected to the AXI Uartlite (`axi_uartlite_1`).

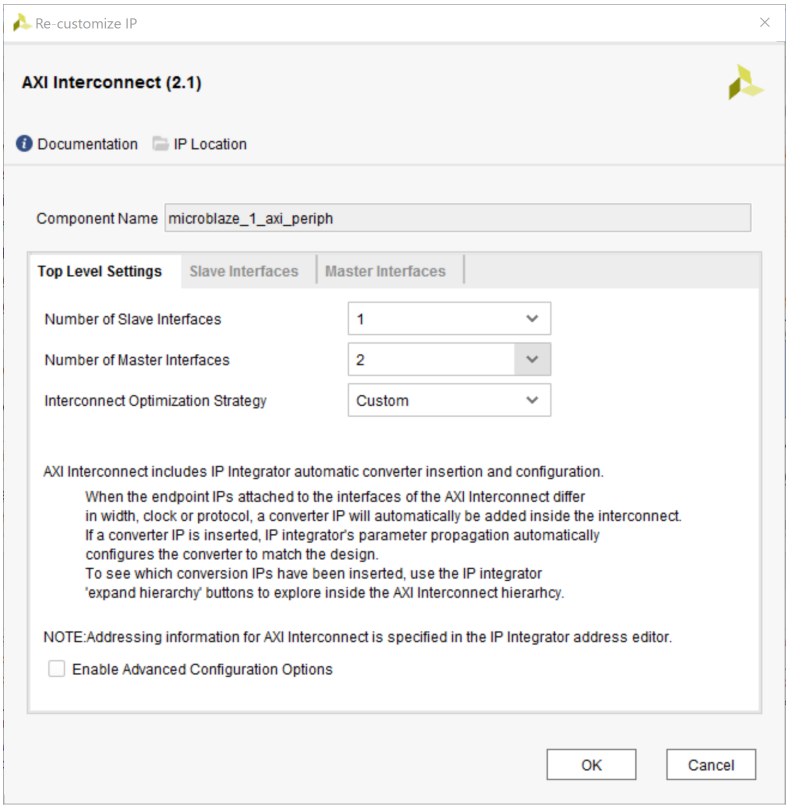
Re-Customizing AXI Interconnects

If you want the `microblaze_1` instance of MicroBlaze to access the AXI block RAM Controller (`axi_bram_ctrl_0`), then the two interconnects instances must be reconfigured.

1. Double-click the AXI Interconnect (`microblaze_1_axi_periph`).

   The Re-customize IP dialog box opens.

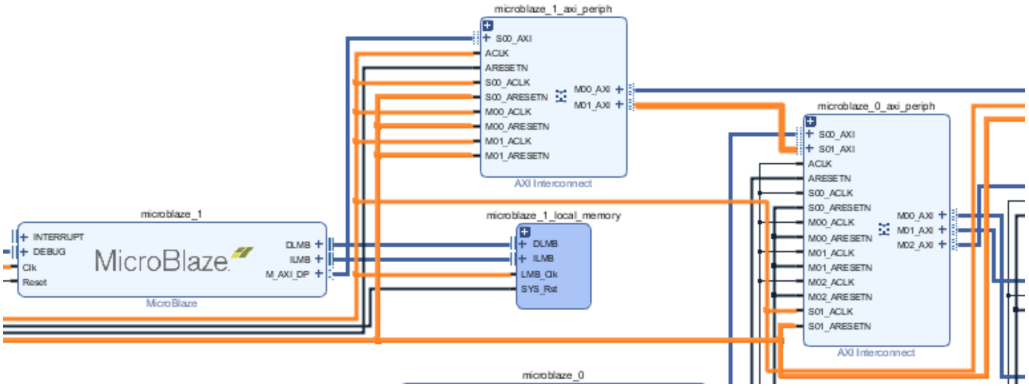
2. From the drop-down menu, set the **Number of Master Interfaces** field to **2**, as shown in the following figure.



   Similarly, re-customize the `microblaze_axi_periph` instance such that the Number of Slave Interfaces field is set to 2.

3. After that, you can connect the Master Interface `M01_AXI` of `microblaze_1_axi_periph` to the `S01_AXI` slave interface of `microblaze_0_axi_periph`.

4. Connect the clocks and resets accordingly as well, as shown in the following figure.

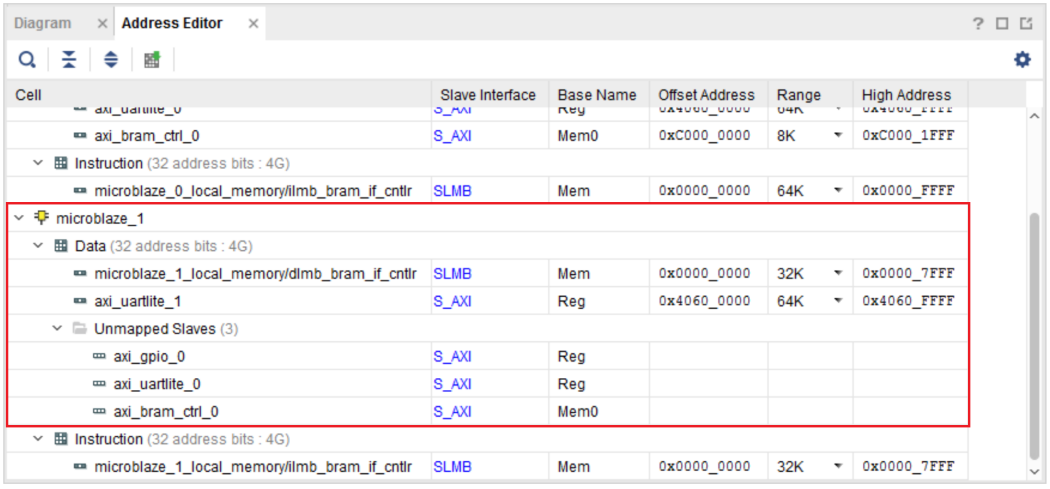


By doing this, you made the `microblaze_1` access all the slaves accessible by `microblaze_0`. This is an optional decision.

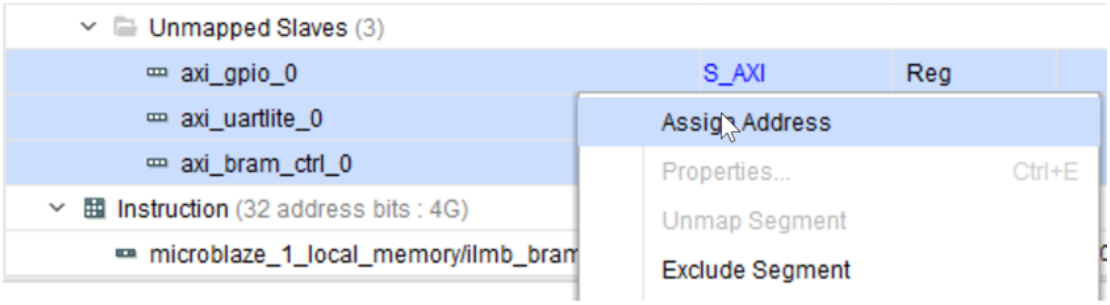
Mapping and Excluding Unwanted Slaves

If you want to access the AXI block RAM Controller from `microblaze_1`, you can exclude the other slaves from the `microblaze_1` memory space. This can be done in the address editor. As can be seen in the following figure, the `microblaze_1` memory space has three unmapped slaves.

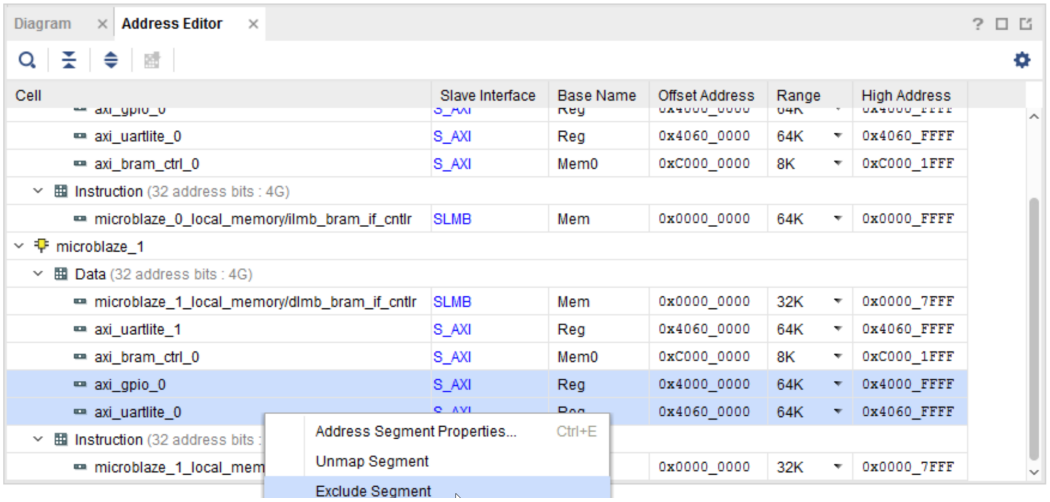
Figure 32: **microblaze_1 Memory Map**



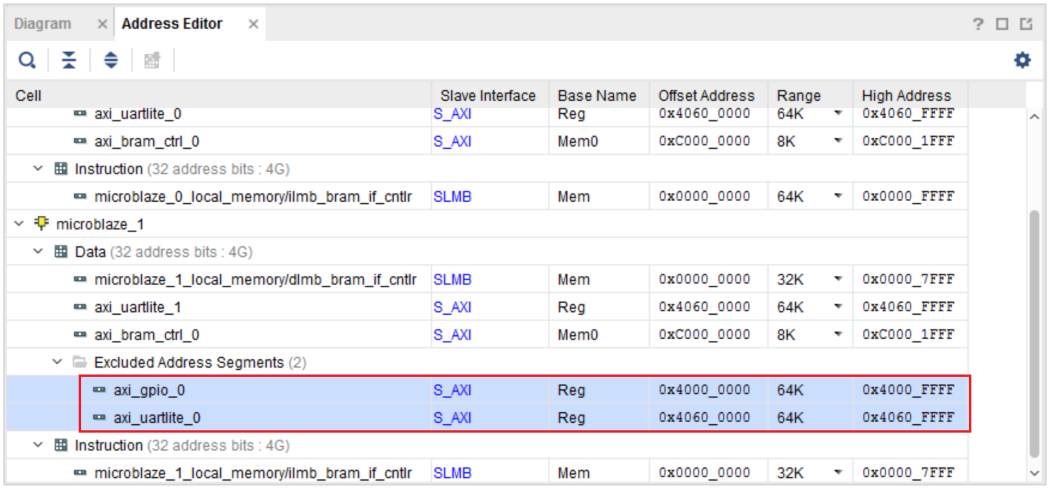
1. First, assign addresses to all the unmapped slaves by selecting them, right-clicking, and selecting **Assign Address** from the context menu.



2. Next, "exclude" the unwanted slaves from the memory map of `microblaze_1` by selecting them in the address editor, right-clicking, and selecting **Exclude Segment**, as shown in the following figure.



The address editor now looks as follows.



After you complete the design in this way, the rest of the design flow is the same as any other Block design flow.

AMD XILINX

Chapter 3

Designing with the Memory IP Core

The Xilinx® memory IP is a combined pre-engineered controller and physical layer (PHY) for interfacing UltraScale architecture and 7 series FPGA user designs with AMBA® advanced extensible interface (AXI4) slave interfaces to DDR2, DDR3, or DDR4 SDRAM, QDRII+ SRAM, or RLDRAM 3 devices.

For more information, see the following:

- *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150)

This chapter provides information about using, customizing, and simulating a LogiCORE IP DDR4, DDR3, or DDR2 SDRAM memory interface core in the Vivado IP integrator tool. This chapter describes the core architecture and provides details on customizing and interfacing to the core.

TIP: *Although the information in this chapter is tailored for the KC705, Kintex®-7 board, the differences for UltraScale devices, and the KCU105 board, are highlighted throughout this text. These guidelines can also be applied to Xilinx devices on custom boards.*

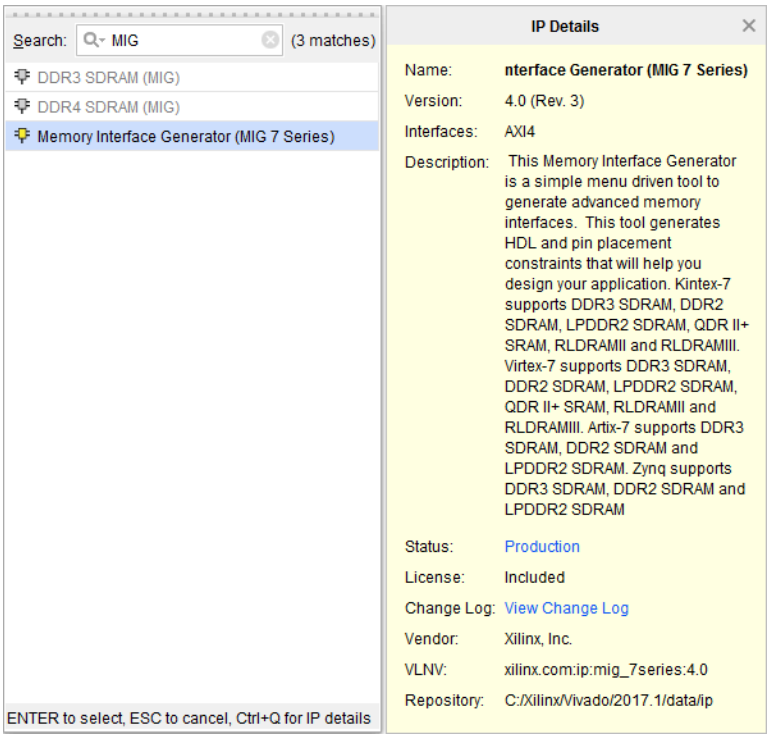
Adding the Memory IP

To add the Memory IP core to a block design, right-click in the IP integrator design canvas and select **Add IP**. A searchable IP catalog opens. When you type the first few letters of an IP name, in this case Memory IP, only the IP cores matching the name are listed.

Alternatively, you can click the **Add IP** button on the toolbar at the top of the canvas  .

Double-click to select the Memory Interface Generator IP and add it to your block design.

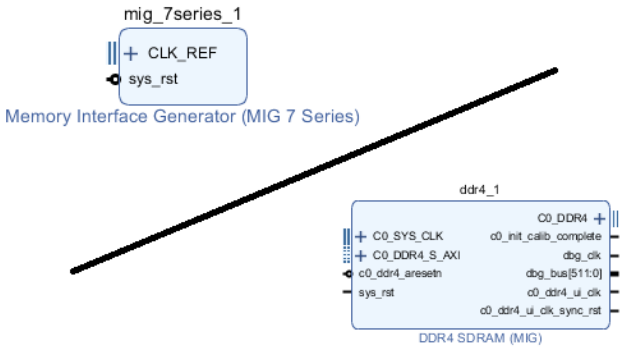
Figure 33: **Add the Memory IP by Searching in the IP Catalog**



This places the Memory IP core into the IP integrator block design.

1. To make changes to the Memory IP configuration, right-click the block to open the menu, and click **Customize Block**. You can also double-click the Memory IP block to open the Xilinx Memory Interface Generator dialog box.

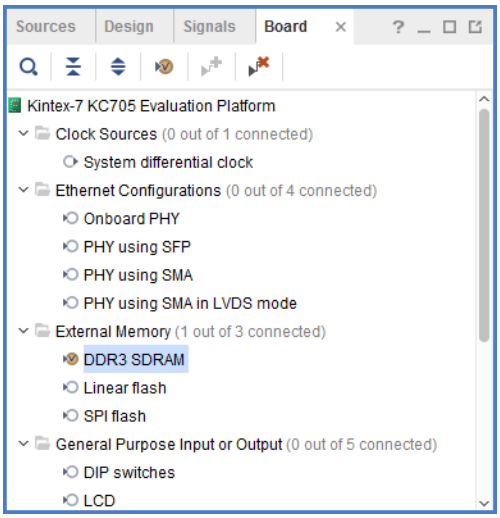
   The following figure shows both the Memory IP and the 7 series IP core in the upper-left, and the DDR4 Memory IP core for UltraScale devices in the lower-right. The Memory IP that is available in the IP catalog depends on the target part or platform board selected for your project. There are separate IP cores to support DDR3 and DDR4 memory controllers for UltraScale devices.



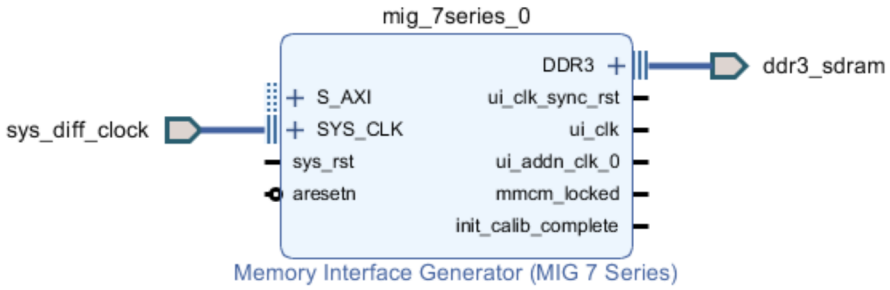
This example targets the KC705 board for the project. As shown in the following figure, the Board tab of the platform board flow is available to let you select components to interface to your design.

2. From the Board tab, drag and drop the DDR3 SDRAM component into the block design canvas.

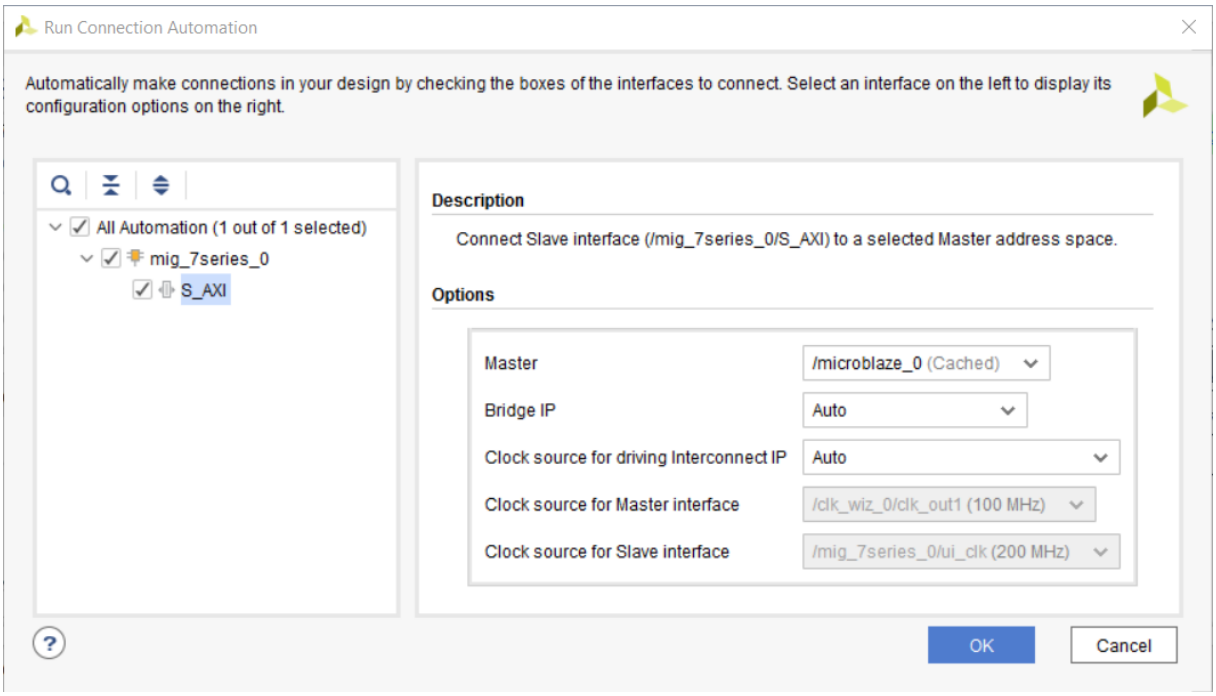
   Note: In the case of the UltraScale KCU105 board, you can also use the DDR4 SDRAM component.



   To connect the memory controller to the memory components on the target platform board, the Vivado IP integrator connects the `SYS_CLK` and DDR interfaces of the Memory IP to external interface ports, as seen in the following figure.



3. Select the **Run Connection Automation** link at the top of the design canvas, as seen in the following figure. This connects the Memory IP to the system FPGA reset on the platform board.

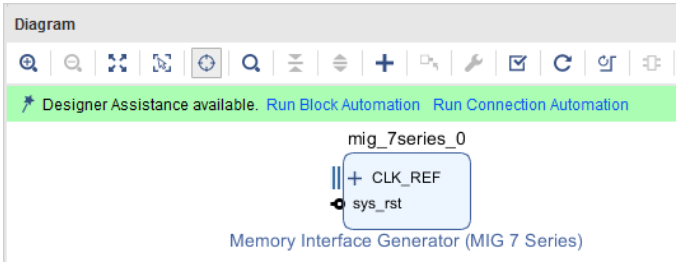


***Note*:** For the KCU105 board, the Run Connection Automation dialog box includes both the `CO_SYS_CLK` and the `sys_rst` interfaces for the Memory IP.

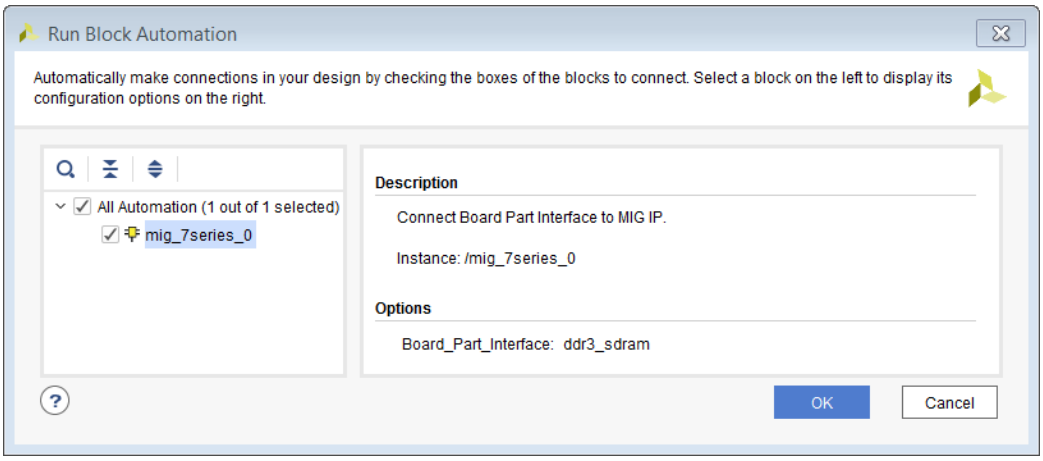
Making Connections with Block Automation

As an alternative to dragging and dropping the DDR SDRAM component from the Board tab, you could use the Block Automation feature of IP integrator to configure the Memory IP and tie its `SYS_CLK` and DDR3 interfaces to the board interfaces.

1. Because the Memory IP core provides the clocking for the entire KC705 board, you should **Run Block Automation**, shown in the following figure, for the Memory IP core prior to adding a clock controller.



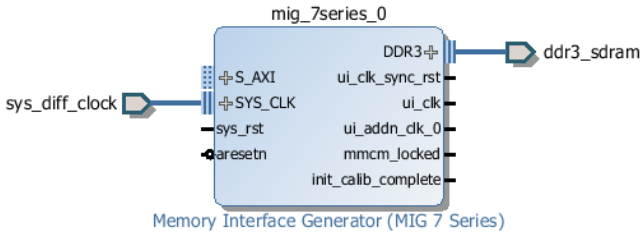
This opens the Run Block Automation dialog box as shown in the following figure.



The Run Block Automation dialog box shows the available IP. In this case, the block design only has the Memory IP you previously added.

2. Ensure the Memory IP is selected, and click **OK**.

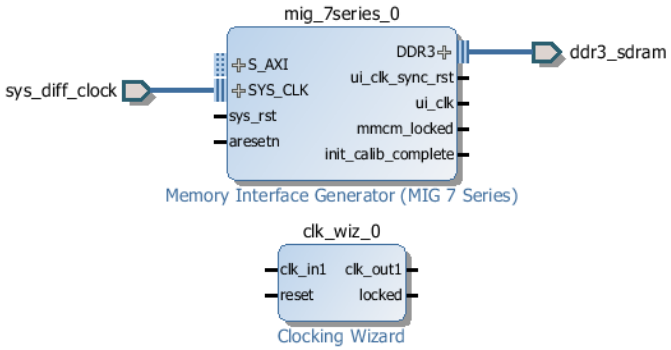
   The `SYS_CLK` and DDR interfaces of the Memory IP are connected to the DDR memory components on the platform board. The Memory IP core is configured for 400 MHz operation with the correct pins selected to interface to the KC705 board. The following figure shows the Memory IP core after running Block Automation.



Adding a Clocking Wizard

If the design requires clocking in addition to the clock generated by the Memory IP core, you need to add a Clocking wizard IP into the block design.

1. Select the **Add IP** command, type **Clock** into the search field, and select the **Clocking Wizard** IP. The following figure shows a Clock Wizard IP with a Memory IP core within a design.

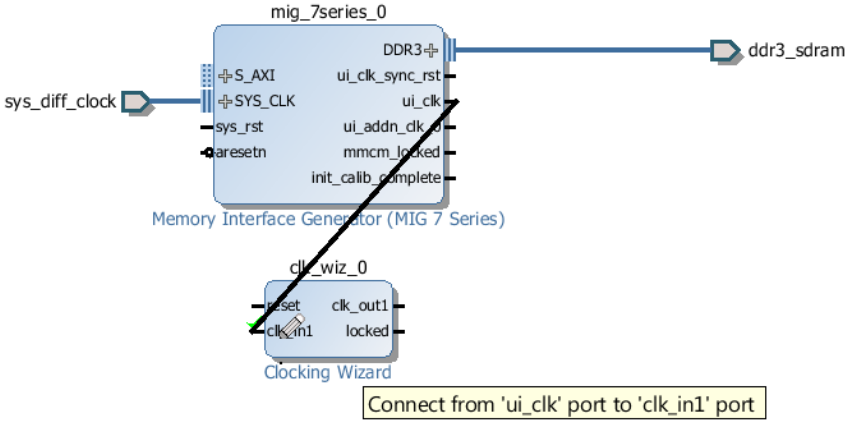


Follow these steps to connect the Clocking Wizard to the Memory IP core:

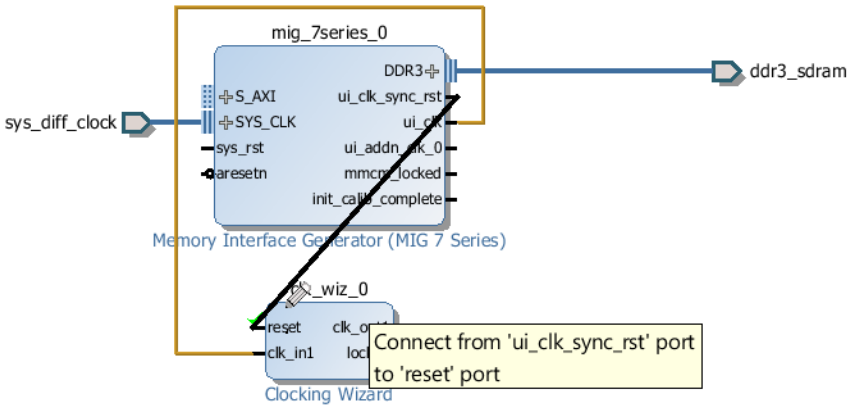
2. Connect the `ui_clk` or `ui_addn_clk_0` output of the Memory IP, as well as any other clocks generated, to the `clk_in1` input of the Clocking wizard, as shown in the following figure.

> **TIP:** *Make sure to use the appropriate output clock pin with the desired frequency.*

3. For the UltraScale Memory IP, connect the `c0_ddr4_ui_clk` pin to the Clocking Wizard, as shown in the following figure.



4. Connect the `ui_clk_sync_rst` pin of the Memory IP core to the `reset` pin of the Clocking wizard, as shown below.
5. For the UltraScale Memory IP, connect the `c0_ddr4_ui_clk_sync_rst` pin to the Clocking wizard, shown in the following figure.



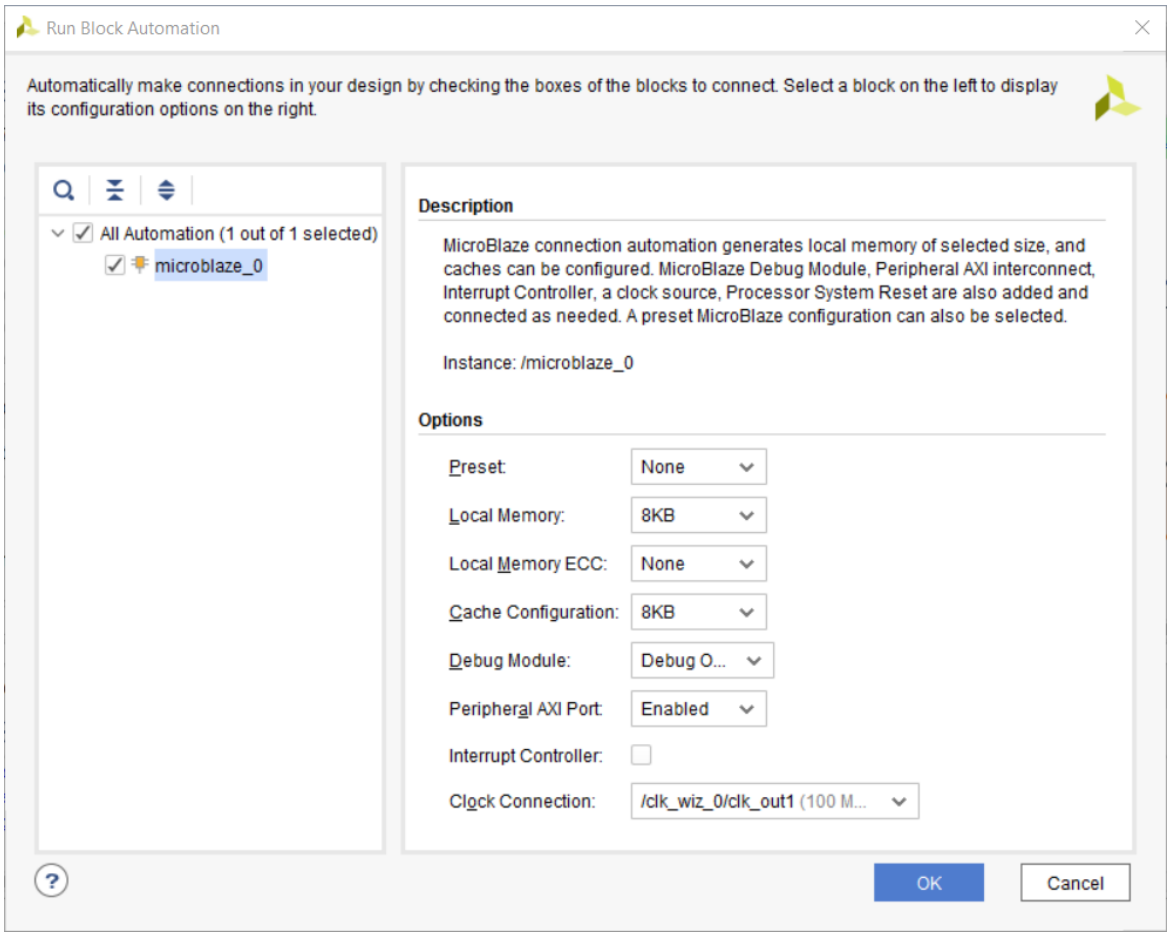
6. Configure the Clocking wizard to generate any required clocks for the design, by double-clicking the IP.

Adding an AXI Master

To complete the Memory IP design, an AXI master such as a MicroBlaze embedded processor, or an external processor is required. The following procedure lists the steps to instantiate a MicroBlaze processor into the block design.

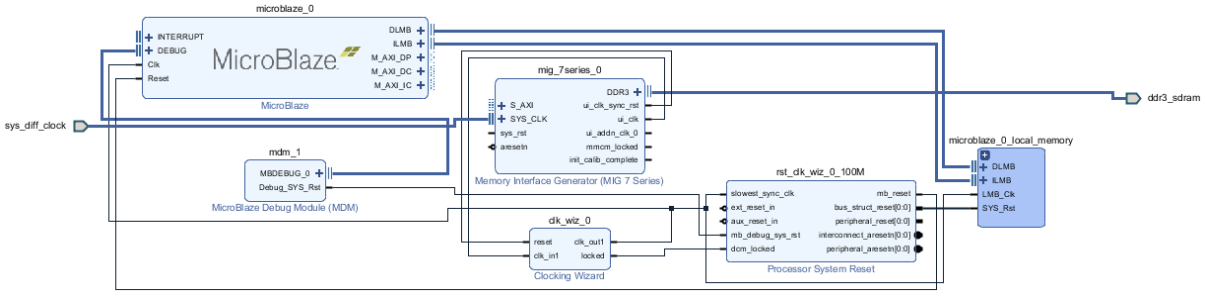
1. Select the **Add IP** command, type `Micro` into the search field, and select the MicroBlaze processor to add it to the design.
2. Click **Run Block Automation** to construct a basic MicroBlaze system, and configure the settings in the dialog box as follows:
   - **Preset: None (or the one that is desired)**
   - **Local Memory**: Select the required amount of local memory from pull-down menu.
   - **Local Memory ECC**: Turn on ECC if desired.
   - **Cache Configuration**: Select the required amount of Cache memory.
   - **Debug Module**: Specify the type of debug module from the pull-down menu.
   - **Peripheral AXI Interconnect**: This option must be enabled.
   - **Interrupt Controller**: Optional.
   - **Clock Connection**: Select the clock source from the pull-down menu.

     The following figure shows the Run Block Automation page.

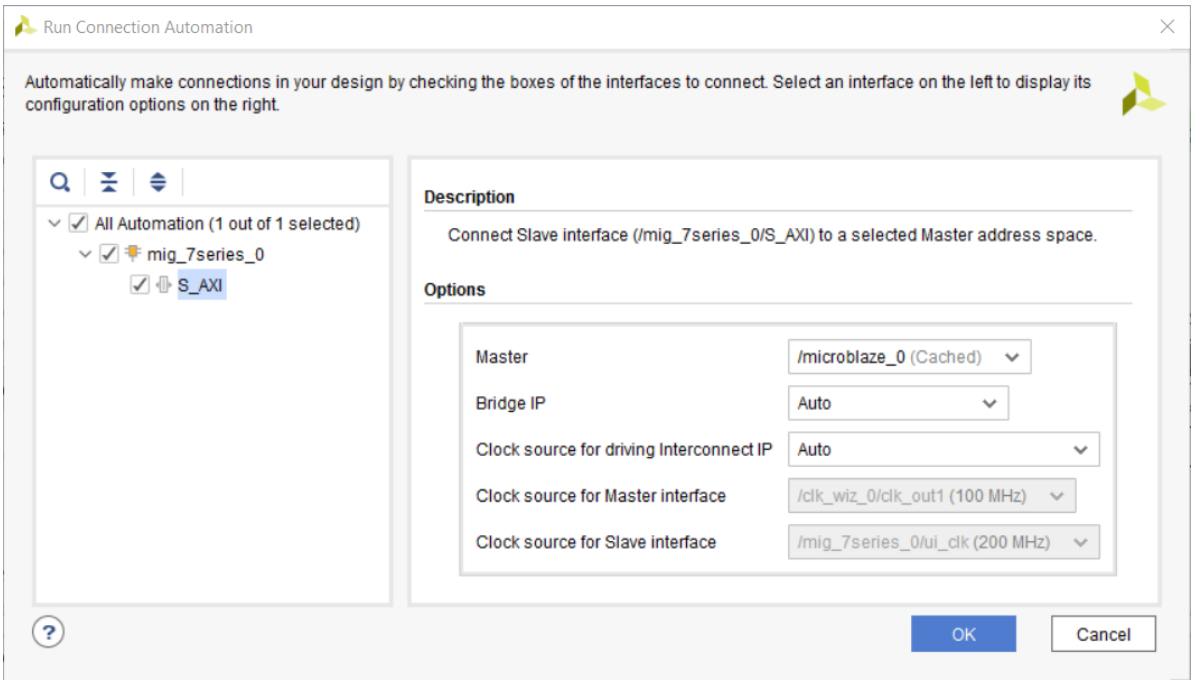


3. Click **OK**.

   The Run Block Automation adds and connects IP needed to support the MicroBlaze processor into the block design. The block design should look similar to the following figure; however, notice that the Memory IP core is not yet connected to the MicroBlaze processor.



4. At the top of the design canvas, click **Run Connection Automation** to connect the Memory IP core to the MicroBlaze processor. The Run Connection Automation dialog box opens, as shown in the following figure.



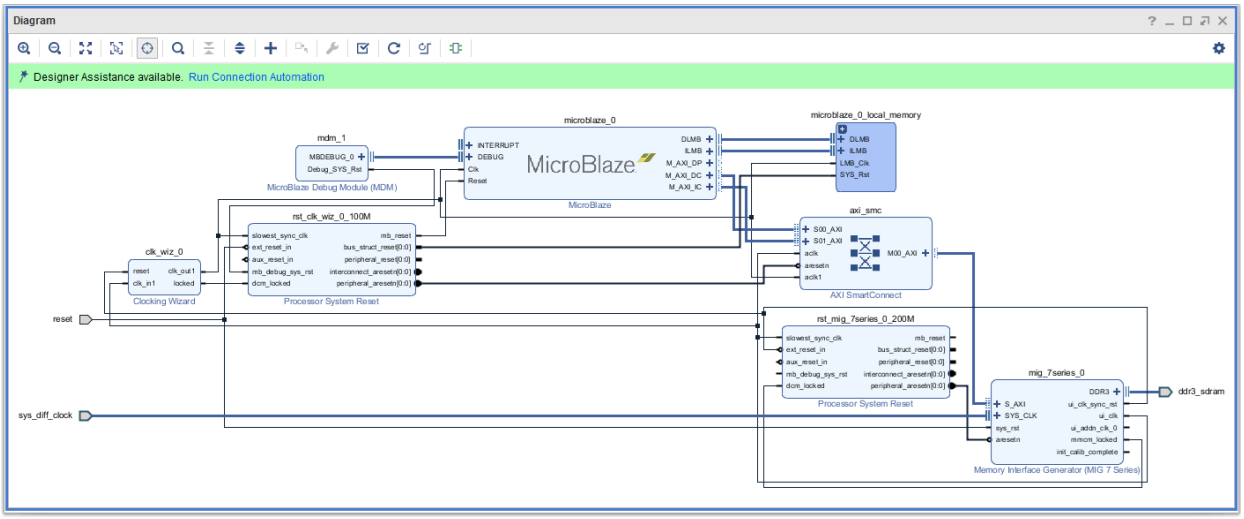
5. Select the `S_AXI` interface of the `mig_7series_0`.

   Note: For the UltraScale Memory IP, select the `C0_DDR4_S_AXI` interface of the `mig_0`.

   The `/microblaze_0 (Cached)` option should be selected by default.

6. You have a choice to select either the `AXI Interconnect` or the `AXI SmartConnect` for the Interconnect IP. For high bandwidth application (such as the Memory IP), the Auto option selects the AXI SmartConnect IP.
7. Leave the rest of the options to their default values.
8. Click **OK**.

This instantiates an AXI Interconnect and makes the required connection between the Memory IP core and the MicroBlaze processor, as shown in the following figure.

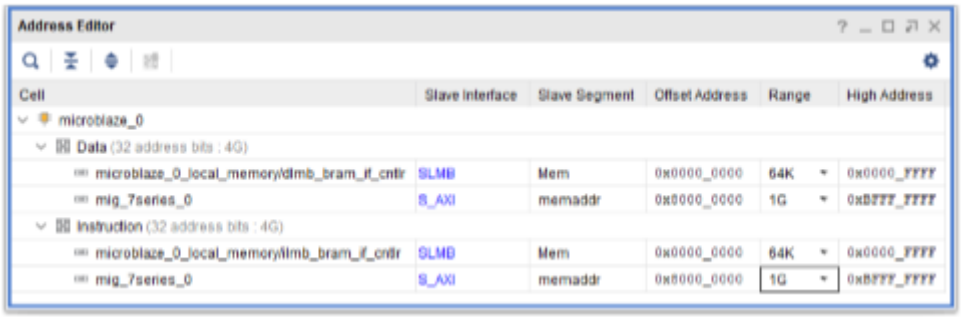


From here you can complete any remaining connections to the design, such as connecting to an external reset source, or connecting any interrupt sources through a concat IP to the MicroBlaze processor.

Creating a Memory Map

To generate the address map for this design, click the Address Editor tab above the diagram. The memory map is automatically created as IP, and added to the design. You can set the addresses manually by entering values in the Offset Address and Range columns. See this link in the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information. The following figure shows the Address Editor.

Figure 34: **Address Editor**



TIP: *The Address Editor tab only appears if the diagram contains an IP block that functions as a bus master, such as the MicroBlaze processor in the following diagram.*

Running Design Rule Checks

The Vivado IP integrator runs basic design rule checks in real time as you create the design. However, problems can occur during design creation. For example, the frequency on a clock pin might not be set correctly. To run a comprehensive design check, click the **Validate Design** button  .

If the design is free of warnings and errors, a successful validation dialog box displays.

Implementing the Design

Now you can implement the design, generate the bitstream, and create the software application in the Vitis™ software platform.

For more information, see *Vitis Unified Software Platform Documentation: Embedded Software Development* (UG1400).

AMD XILINX

Chapter 4

Reset and Clock Topologies in IP Integrator

To create designs with IP integrator that function correctly on the target hardware, you must understand reset and clocking considerations. This chapter provides information about clock and reset connectivity at the system level. In the Vivado® IP integrator, you can use the Xilinx® platform board flow, which enables you to configure IP in your design to connect to board components using signal interfaces in an automated manner. You can also make all the connections manually. The examples and overall flow described in this chapter use the platform board flow, but the considerations are valid for all block designs.

For designs using the Memory IP core, the core provides the clock source, and the primary clock from the board oscillator must be connected directly to the Memory IP core. For more information, see Chapter 3: Designing with the Memory IP Core.

The Memory IP core can generate up to five additional clocks (Memory IP core for UltraScale devices can generate only four additional clocks), which you can use for resetting the design as needed. For designs that contain a Memory IP core, ensure that the primary onboard clock is connected to memory controller, and then use the user clock (`ui_clock` or the `ui_addn_clk_x`) as additional clock sources for the rest of the design.

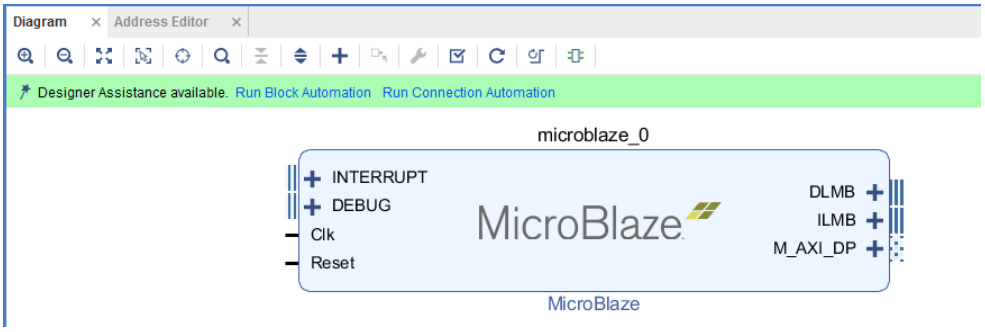
For IP integrator designs with platform board flow, specific IP (for example, Memory IP and Clocking Wizard) support board-level clock configuration. For the rest of the system, clocking can be derived from the supported IP. Similarly, for driving reset signals, board-level reset configuration is supported by a specific reset IP (for example, `proc_sys_reset`). You can use other IP that also require external reset but are not currently supported by the platform board flow.

The following sections describe the reset topologies for different types of designs.

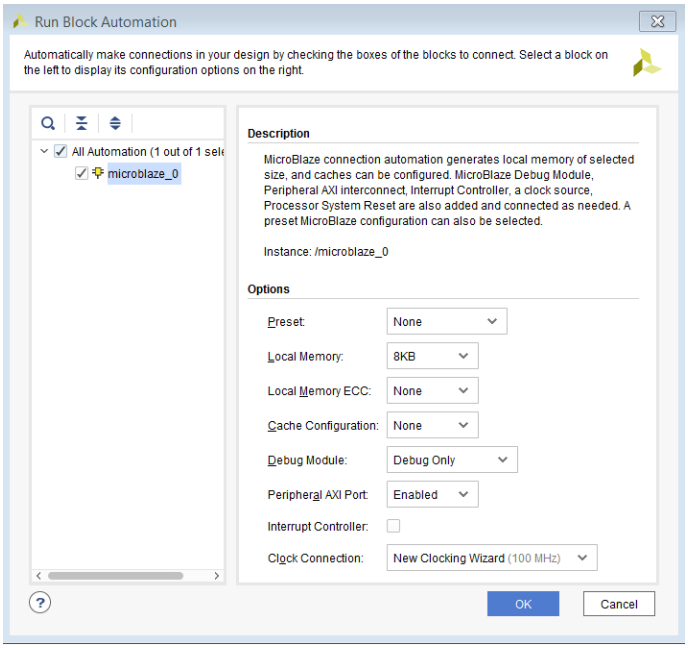
MicroBlaze Design without a Memory IP Core

For any design that uses a MicroBlaze processor without a Memory IP core, you can instantiate a Clocking Wizard IP to generate the clocks required. For the platform board flow, you can configure the connection as follows:

1. After instantiating a MicroBlaze processor in the design, click the **Run Block Automation** link. This creates the MicroBlaze subsystem, as shown in the following figure.

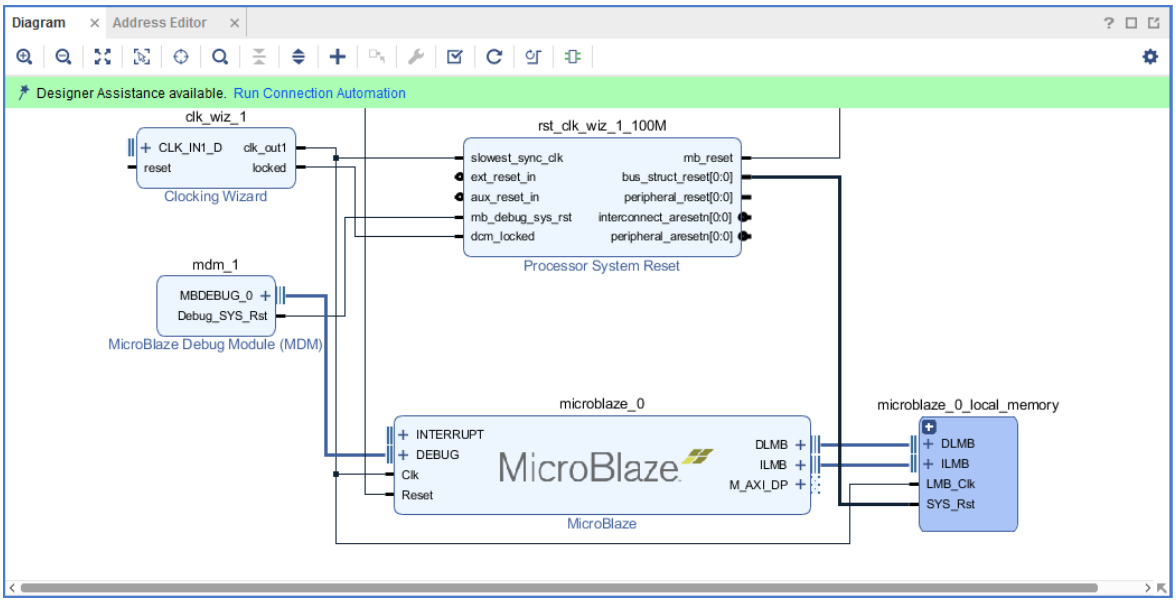


2. In the Run Block Automation dialog box, select the **New Clocking Wizard** option to instantiate the Clocking Wizard IP, and click **OK**, as shown in the following figure.



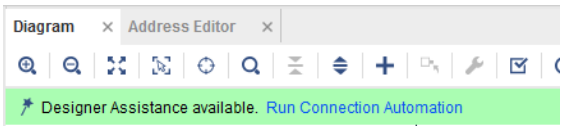
Running Block Automation also instantiates and connects the Proc Sys Reset IP to the various blocks in the design.

The IP integrator canvas looks like the following figure.

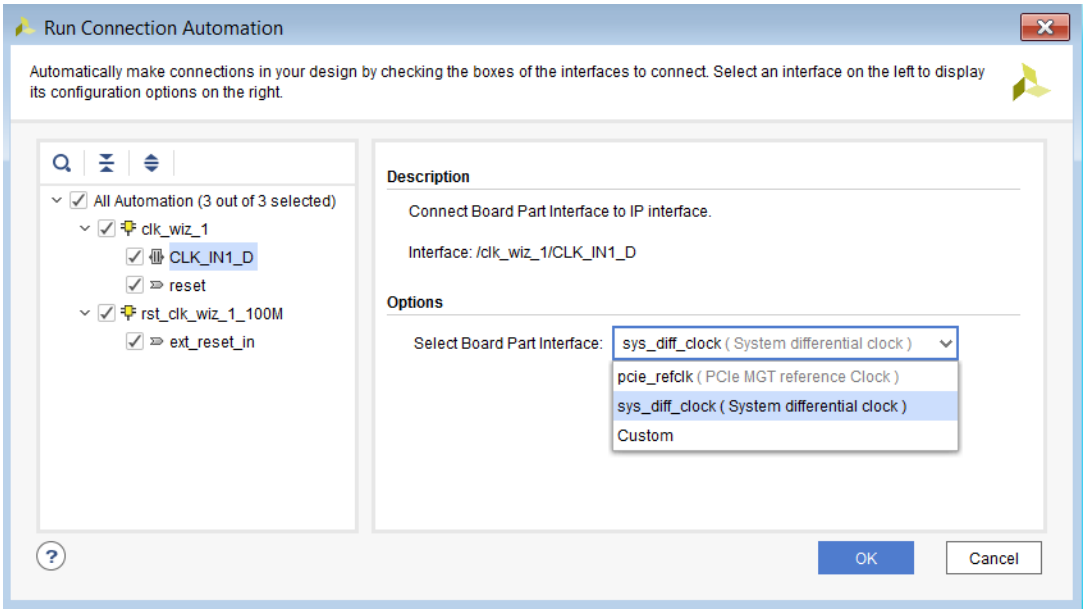


3. Click **Run Connection Automation** and select **/clk_wiz_1/CLK_IN1_D** to connect the on-board clock to the input of the Clocking Wizard IP, according to the board definition.

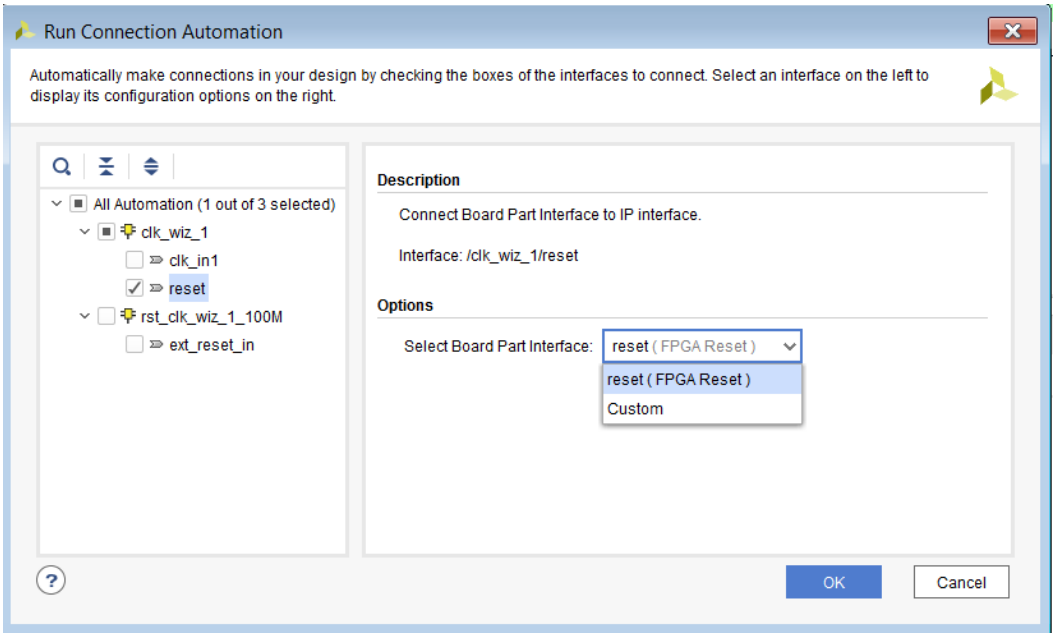
   ***Note*:** You can customize the Clocking Wizard to generate the various clocks required by the design.



4. In the Run Connection Automation dialog box, select **sys_diff_clock** to select the board interface for the target board, or select **Custom** to tie a different input clock source to the Clocking Wizard IP, then click **OK**.



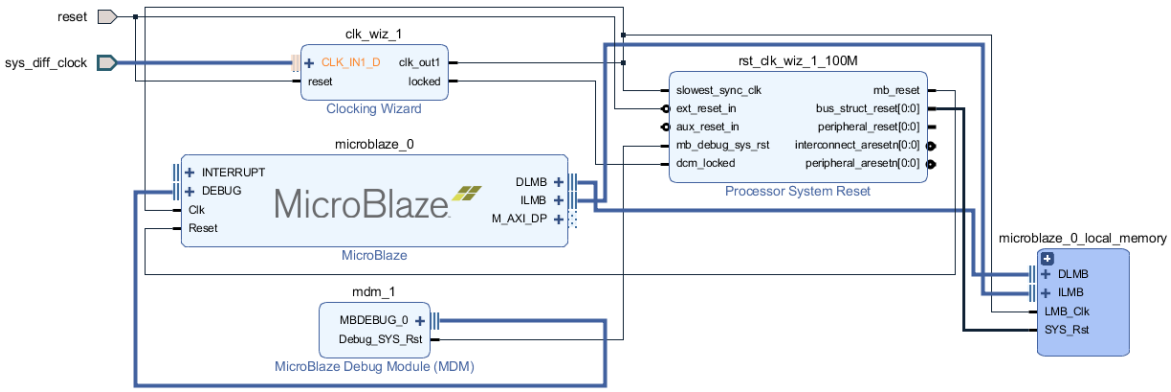
5. For the reset pin of the Clocking Wizard, select the dedicated reset interface on the target board or a Custom reset input source.



Note: Steps 4 and 5 above can also be done by dragging and dropping the System Differential Clock under the Clock Sources folder and FPGA Reset from the Reset folder in the Board tab.

6. For the `ext_reset_in` pin for the Processor System Reset block choose the same reset source as chosen for the Clocking Wizard in the step above or a Custom reset source.

   After you make your choice and click **OK**, the IP integrator canvas looks like the following figure.



CAUTION! *If the platform board flow is not used, ensure that the "locked" output of the Clocking Wizard is connected to the "dcm_locked" input of Proc_Sys_Reset.*

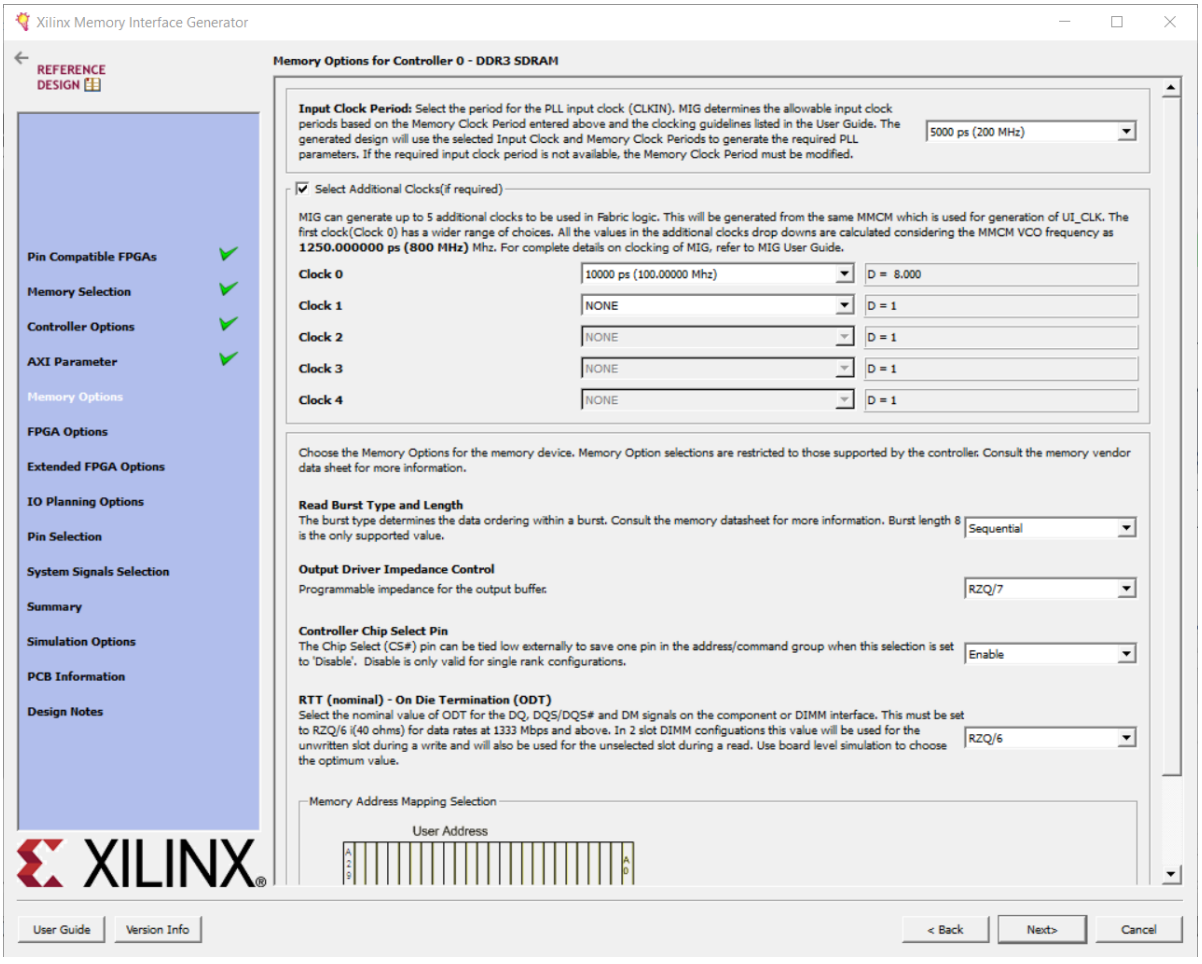
MicroBlaze Design with a Memory IP Core

RECOMMENDED: *As mentioned in the introduction, the Memory IP is a clock source, and Xilinx recommends that you connect the on-board clock directly to the Memory IP core.*

The Memory IP core provides a user clock (`ui_clock`) and up to five additional clocks (four in case of UltraScale Memory IP) that can be used in the rest of the design. You can configure the connection, as follows:

1. When using the platform board flow automation in a design that contains the Memory IP, add the Memory IP first (or drag and drop the DDR3 SDRAM/DDR4 SDRAM interface from the Board window which instantiates the Memory IP core and configures it for the board), and then run Block Automation. This connects the on-board clock to the Memory IP core.

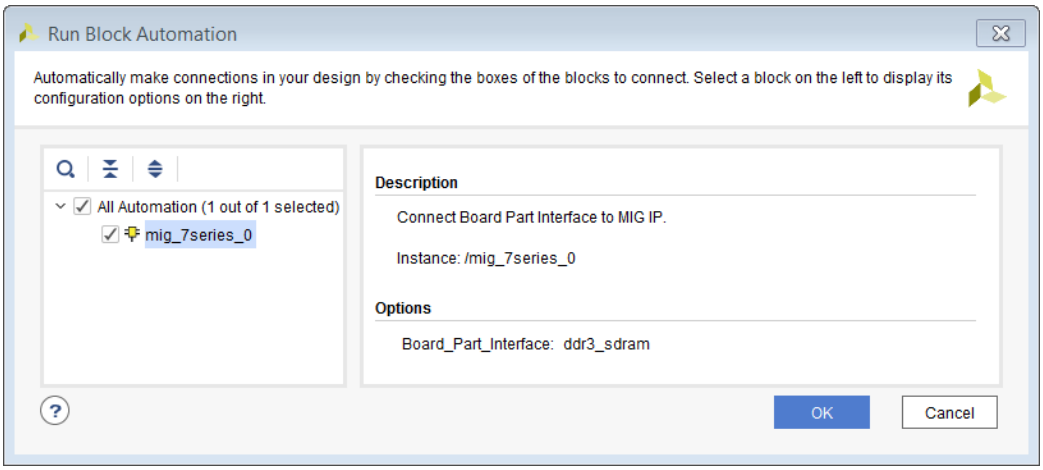
   You can then customize Memory IP to generate additional clocks, as shown in the following figure.



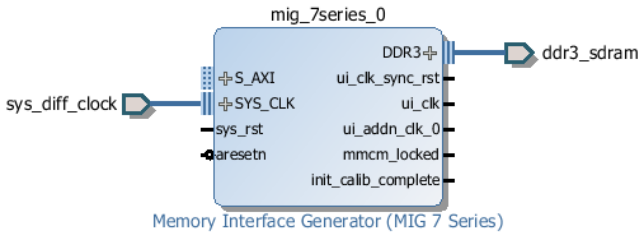
2. After configuring the MIG to generate additional clocks, click the **Run Connection Automation** link at the top of the banner.

The Run Connection Automation dialog box states that the `ddr3_sdram` interface is available, as shown in the following figure.

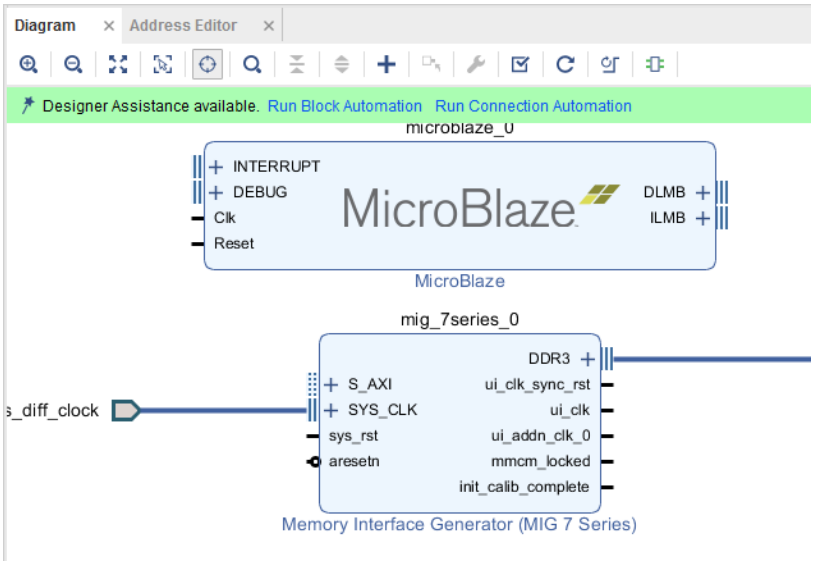
3. Click **OK**.



This connects the interface ports to the Memory IP, as shown in the following figure.

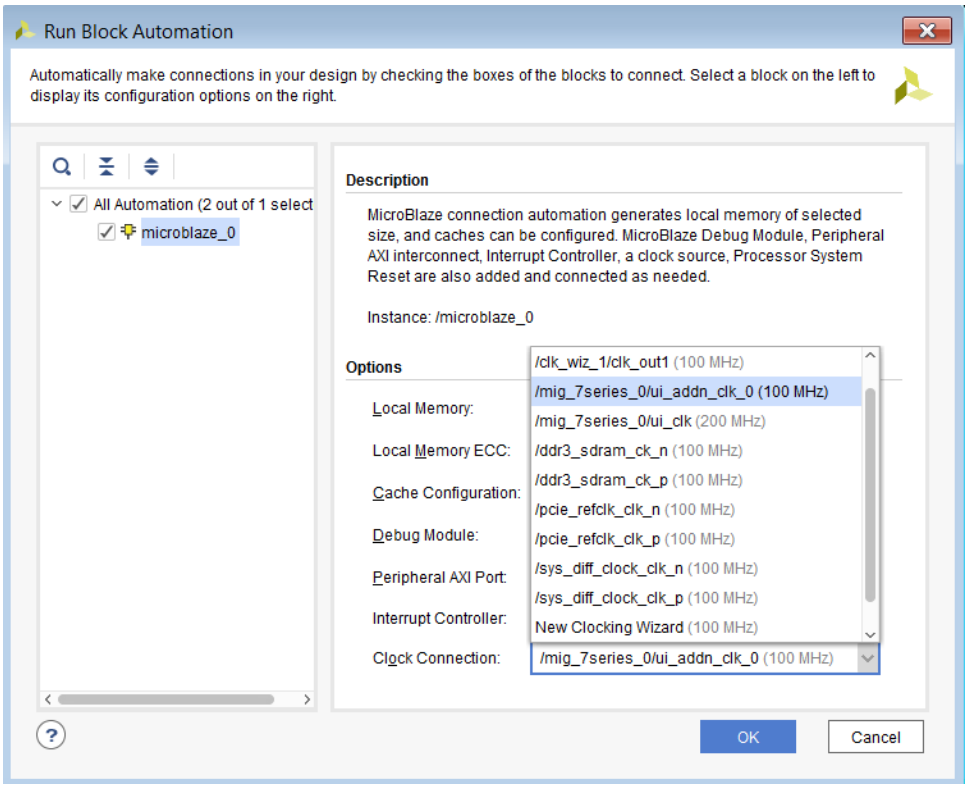


4. Add the MicroBlaze processor to the design and run Block Automation, as shown in the following figure.

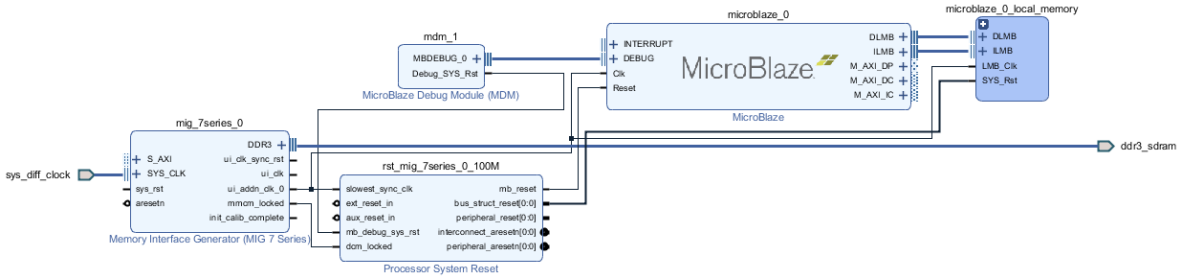


5. In the **Clock Connection** field of the Run Block Automation dialog box, select the Memory IP `ui_clk` (**/mig_7series_0/ui_clk** or **mig_7series/u_addn_clk_0**) as the clock source for the MicroBlaze processor, as shown in the following figure, and click **OK**.

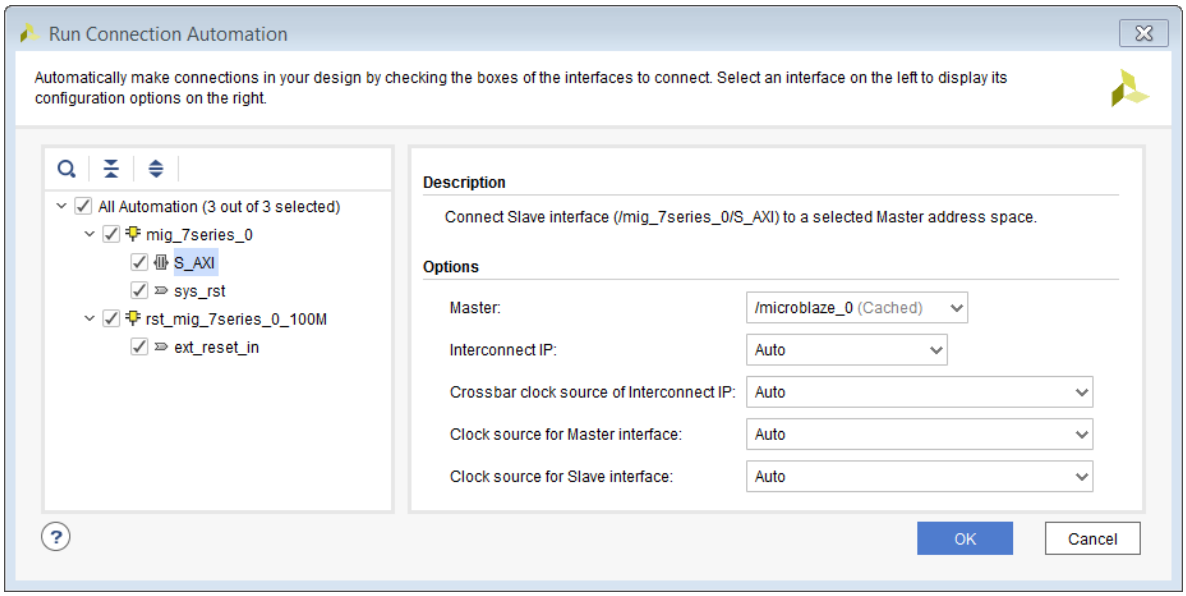
> **TIP:** *The **mig_7series_0/ui_addn_clk_0** is selected by default.*



This creates a MicroBlaze subsystem and connects the `ui_addn_clk_0` as the input source clock to the subsystem, as shown by the highlighted net in the following figure.



6. Make the following additional connections:
   a. Click **Connection Automation** and select `/mig_7series/S_AXI` to connect the Memory IP to MicroBlaze.
   b. In the Run Connection Automation dialog box select `/microblaze_0` (Cached) option for the `S_AXI` interface.
   c. Leave all other settings for `S_AXI` to their default value of **Auto**.

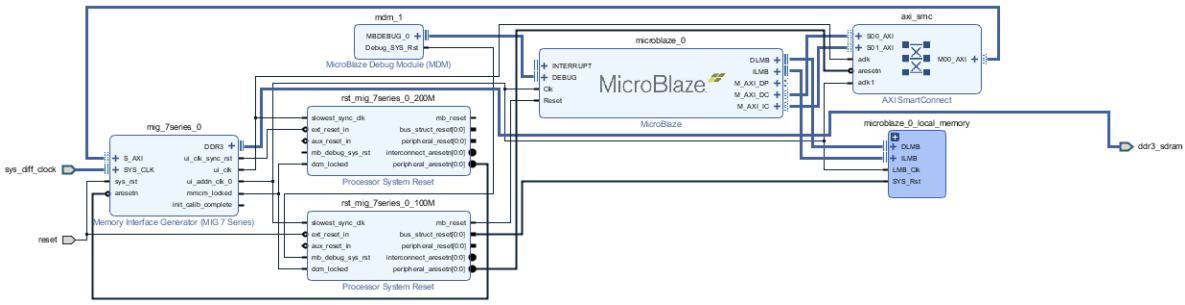


d. Connect the on-board reset to the sys_rst input of the Memory IP.

e. Connect the `ext_reset_in` of the `rst_mig_7_series_0_100M` Processor System Reset block to `reset` (FPGA Reset).

f. Click **OK**.

The following figure shows the completed connection for MB-Memory IP with Designer Assistance.



Designs with Memory IP and the Clocking Wizard

For designs that require specific clock frequencies not generated by the Memory IP core, you can instantiate a Clocking Wizard IP and use the `ui_clock` output of the Memory IP as the clock input for the IP Clocking wizard.

You also need to make the following additional connections:

1. Connect the onboard reset to the Clocking wizard reset input in addition to the Memory IP.
2. Connect the `mmcm_locked` pin of the Memory IP and locked pin of Clocking wizard to the `Util_Vector_Logic` IP configured to the `AND` operation. Then, connect the output of the `Util_Vector_Logic` to the `dcm_locked` input of `Proc_Sys_Reset`.

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

1. *Triple Modular Redundancy (TMR) LogiCORE IP Product Guide* (PG268)

2. *MicroBlaze Debug Module (MDM) LogiCORE IP Product Guide* (PG115)
3. *UltraScale Architecture-Based FPGAs Memory IP LogiCORE IP Product Guide* (PG150)
4. *Vitis Unified Software Platform Documentation*
5. *Vivado Design Suite Tcl Command Reference Guide* (UG835)
6. *Vivado Design Suite User Guide: Design Flows Overview* (UG892)
7. *Vivado Design Suite User Guide: Using the Vivado IDE* (UG893)
8. *Vivado Design Suite User Guide: System-Level Design Entry* (UG895)
9. *Vivado Design Suite User Guide: Synthesis* (UG901)
10. *Vivado Design Suite User Guide: Using Constraints* (UG903)
11. *Vivado Design Suite User Guide: Dynamic Function eXchange* (UG909)
12. *ISE to Vivado Design Suite Migration Guide* (UG911)
13. *Vivado Design Suite Tutorial: Embedded Processor Hardware Design* (UG940)
14. *UltraScale Architecture Libraries Guide* (UG974)
15. *MicroBlaze Processor Reference Guide* (UG984)
16. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
17. *Vivado Design Suite Tutorial: Designing IP Subsystems Using IP Integrator* (UG995)
18. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* (UG1118)

Training Resources

1. Designing FPGAs Using the Vivado Design Suite 1 Training Course
2. Embedded Systems Design Training Course
3. Advanced Features and Techniques of Embedded Systems Design Training Course
4. Vivado Design Suite QuickTake Video Tutorials

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/01/2022 Version 2022.1	
Initial release.	N/A

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright