See all versions of this document

MicroBlaze V Processor Embedded Design User Guide

UG1711 (v2025.2) December 3, 2025

AMD

Table of Contents

Chapter 1

Introduction

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the Design Hubs page. You can also use the Design Flow Assistant to better understand the design flows and find content that is specific to your intended design needs. 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 V 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 AMD 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

Device Tools Flow Overview

The AMD 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 IP integrator to capture hardware platform information and export in XML format applications with other data files to develop designs for AMD processors. Software design tools use the XML to perform the following tasks.

- 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 AMD MicroBlaze™ V embedded processor is a Reduced Instruction Set Computer (RISC) core, optimized for implementation in AMD field programmable gate arrays (FPGAs and adaptive SoCs). The core is based on the RISC-V open standard Instruction set architecture.

To create an embedded MicroBlaze V processor design, refer Chapter 2: Using a MicroBlaze V Processor in an Embedded Design to understand how to use IP integrator and other AMD tools. See the *MicroBlaze V Processor Reference Guide* (UG1629) for more processor information.

AMD provides design tools for developing and debugging software applications for AMD 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.

AMD provides integration between a hardware design and the software development with an integrated flow down to AMD Vitis™ software platform. Vitis is 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.

***Note*:** MicroBlaze V requires the use of the Vitis Unified IDE.

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 and instantiate an AMD processor, along with any other AMD 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.

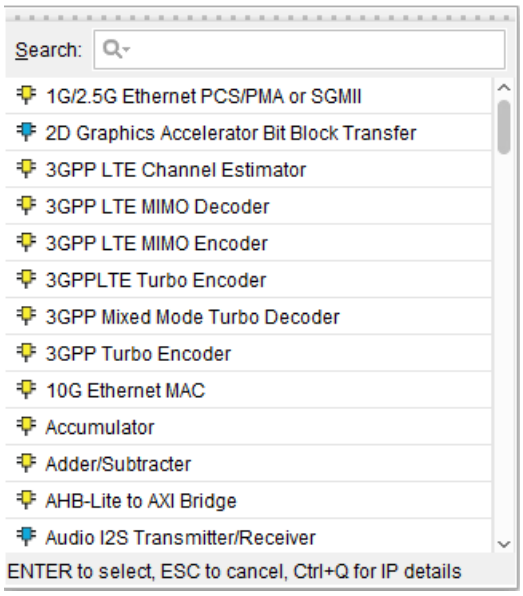
5. Run the top-level design through synthesis and implementation and export the hardware to the Vitis software platform.
6. Create your software application. In the Vitis software platform, associate the Executable Linkable File (ELF) file with the hardware design.
7. Program into the target board.

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**

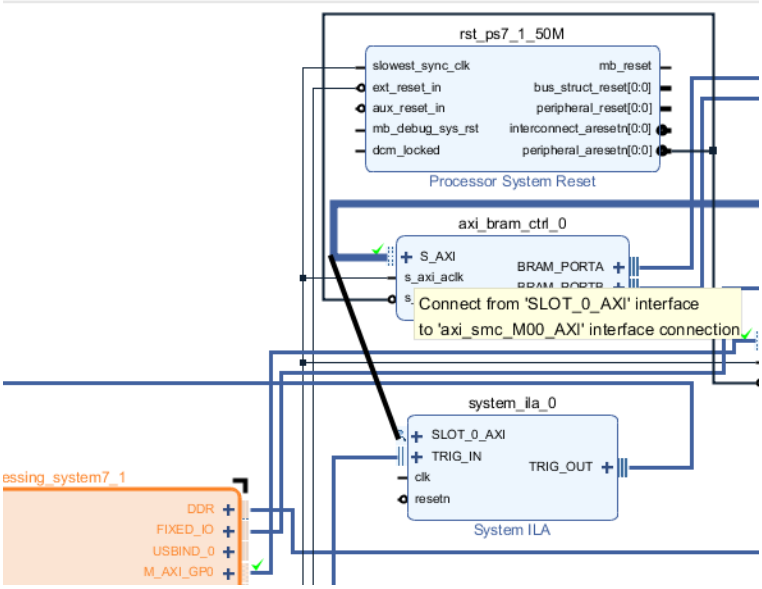


Making Manual Connections in a Design

The following figure shows how to connect the ILA `SLOT_0_AXI` or the `clk` pin to the clock and the AXI interface that requires monitoring 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 and 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 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 to specify 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 AMD IP. The command provides more control to specify the interface type and the mode (master or slave).

Platform Board Flow in IP Integrator

The Vivado Design Suite is a board-aware tool. The tool understands various components present on the target board and customizes an IP to be instantiated and configured to connect to the components of a particular board.

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

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.

TIP: *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.

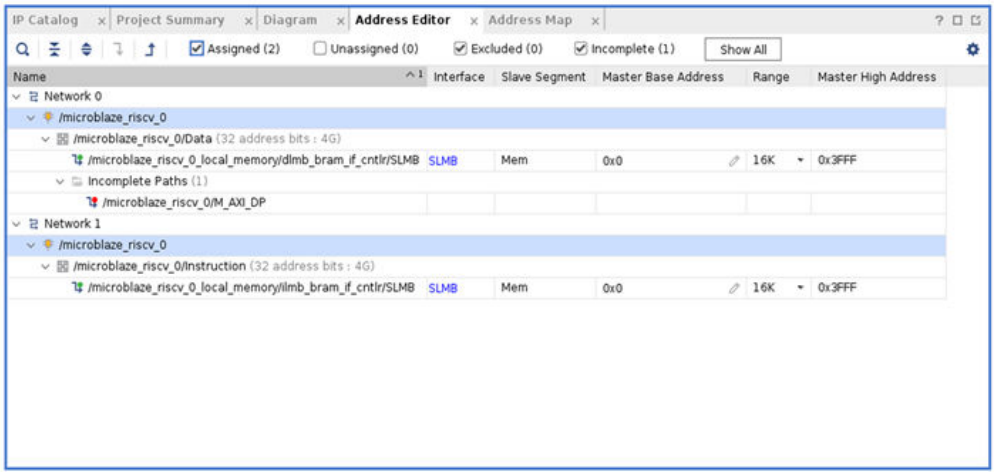
Memory-Mapping in the Address Editor

The memory-mapping of the peripherals (slaves) instantiated in the block design are automatically assigned, but you can also assign the addresses manually. Perform the following steps to generate the address map for this design.

1. Click the **Address Editor** tab above the diagram.
2. Click the **Assign All** button. Set addresses by entering values in the Offset Address and Range columns. See section Addressing for Block Designs in the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information.

   TIP: *The Address Editor tab only opens if the Diagram contains an IP. For example, a MicroBlaze V device that functions as a bus master in the design.*

Figure 3: **Memory-Mapping Peripherals**



Running Design Rule Checks

The Vivado IP integrator executes 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 is not 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, the following are two steps are required to complete the design.

- Generate the output products
- Create a HDL wrapper

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

Based on 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 Sources window, right-click the block design under the Design Sources folder, and click Generate Block Design.
- In the Flow Navigator panel, under IP Integrator, click **Generate Block Design**.

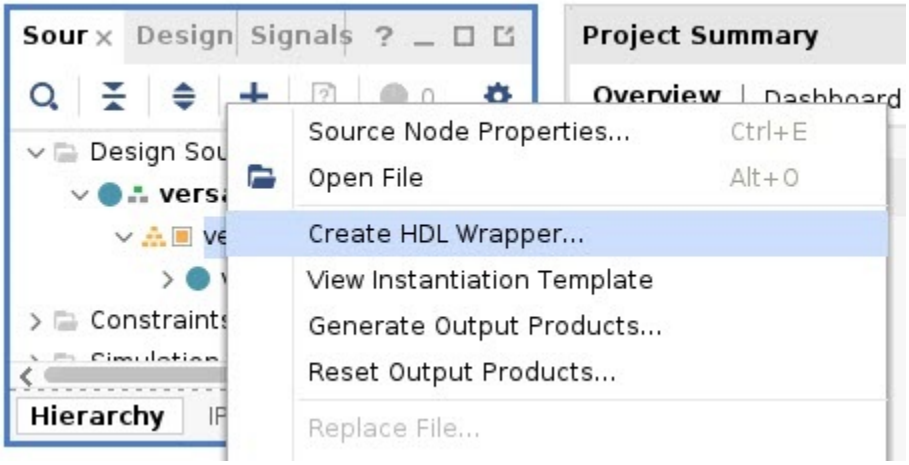
The Vivado Design Suite generates the HDL source files and appropriate constraints for all IPs used in the block design. The source files are generated based upon the Target Language selected during project creation, or in the Settings dialog box. See *Vivado Design Suite User Guide: Designing IP Subsystems 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, navigate to 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**

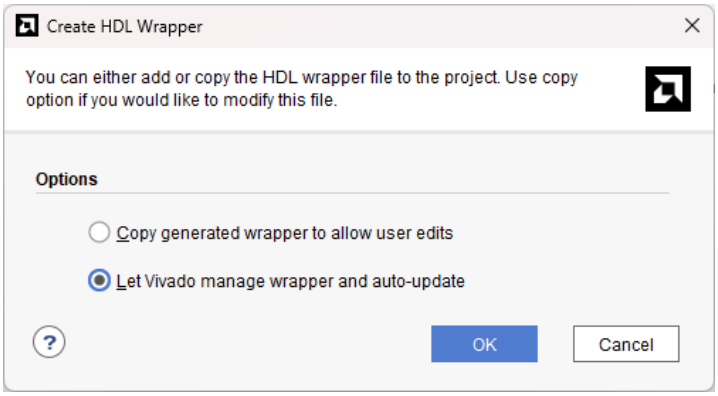


See section Integrating the Block Design into a Top-Level Design in *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information on generating output products.

Vivado offers the following two choices for creating an HDL wrapper.

- 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 which are elaboration, synthesis, and implementation.

Using the Vitis Software Platform

The Vitis unified software platform IDE provides a complete environment for creating software applications targeted for AMD 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 AMD 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

The Vitis unified software platform incorporates the C/C++ Development Toolkit (CDT).

Features of the Vitis unified software platform include:

- C/C++ code editor and compilation environment
- Project management

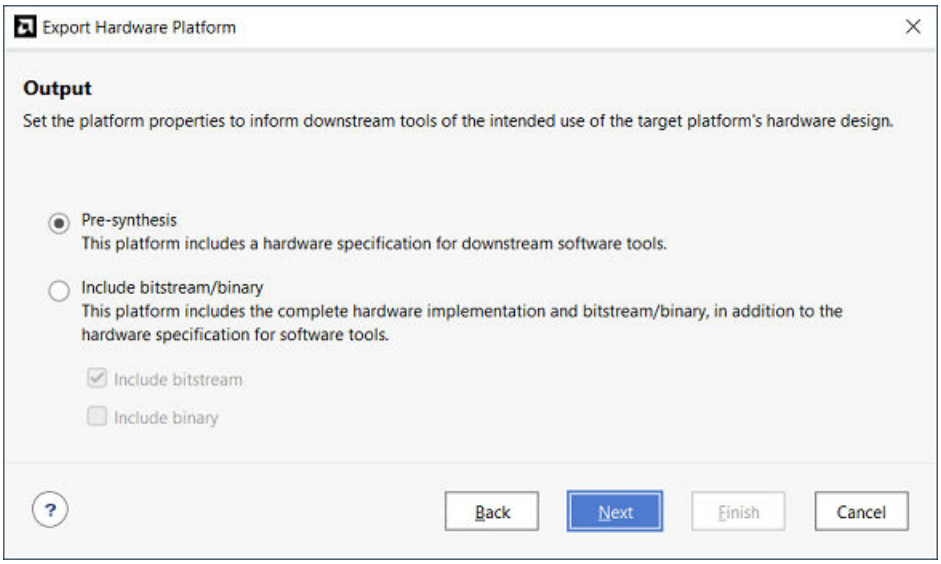
- Application build configuration and automatic make file generation
- Error navigation
- Integrated environment for debugging and profiling embedded targets
- Additional functionality available using third-party plug-ins, including source code version control

Exporting a Hardware Description

Once a design is implemented and the bitstream is generated, you can export the design to the Vitis software platform for software application development. If the processing logic(PL) does not contain any logic, you can export the design without implementing or generating the bitstream. Perform the following steps to export your design:

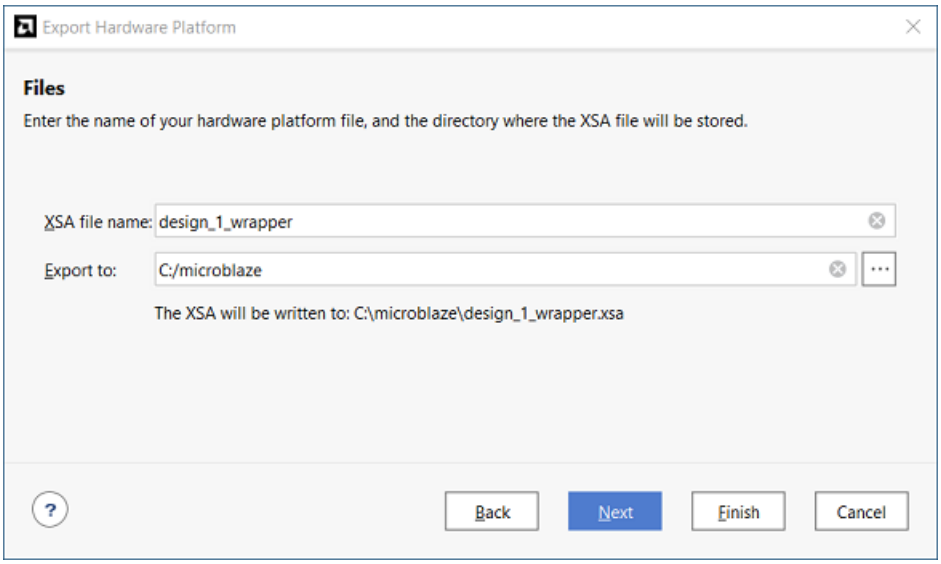
1. Navigate to the Vivado File menu.
2. Select **File → Export → Export Hardware**.

   The Export Hardware dialog box opens.
3. Select the **Include bitstream** option using the checkbox in the Platform State page and click **Next**.



4. In the Export Hardware Platform window, click **Next**.

   ***Note*:** Do not change the auto-populated default value in the XSA file name.



5. Click **Finish**. The exported hardware XSA file is generated in the project directory.
6. After the hardware definition is exported, select **Tools → Launch Vitis** to launch the Vitis software platform from Vivado.
7. Click **Open Workspace** from the Welcome to the Vitis Unified IDE screen.

   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.

Chapter 2

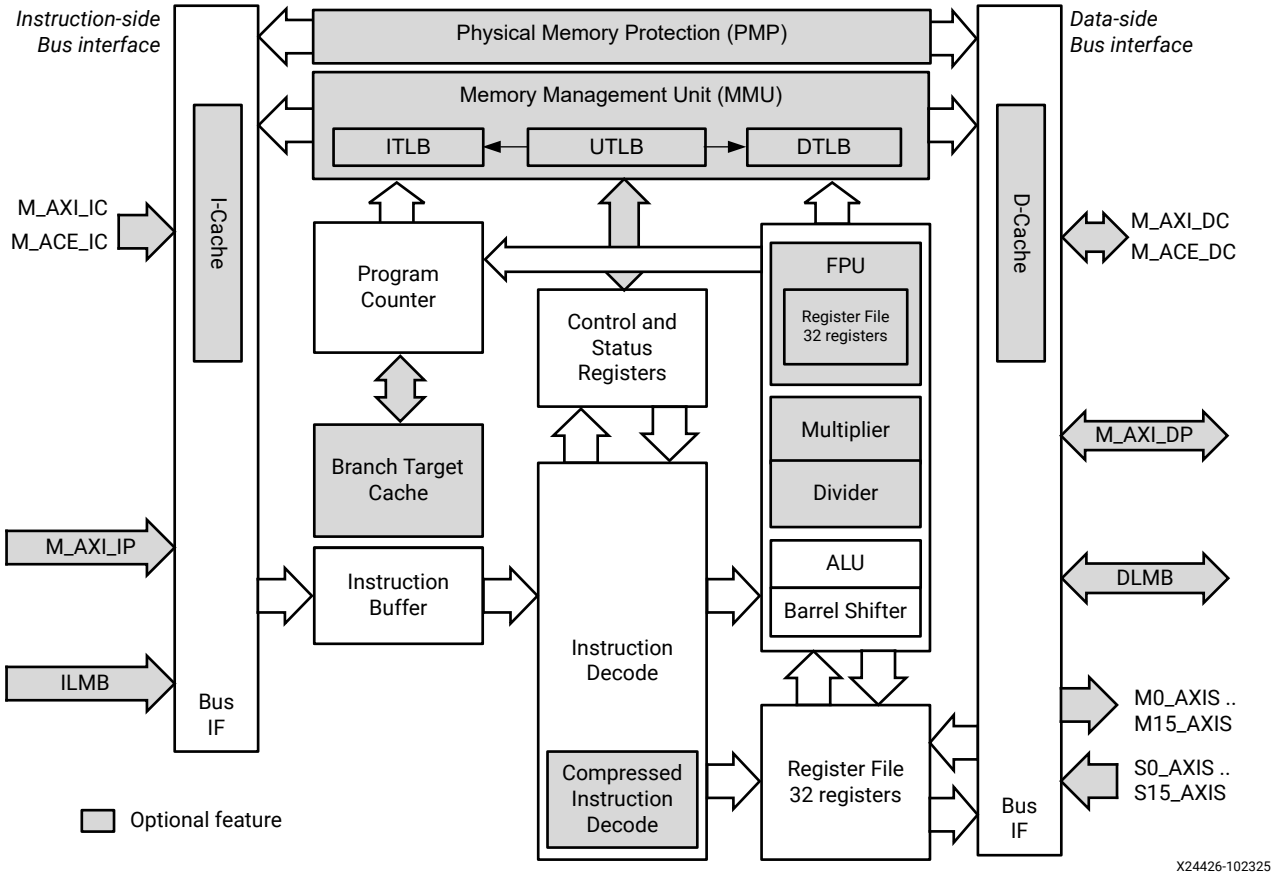
Using a MicroBlaze V Processor in an Embedded Design

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

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

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

Figure 6: **MicroBlaze V Processor Block Diagram**



The MicroBlaze V 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
- 32-bit address bus, extensible to 64-bit
- Single issue pipeline

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

See the *MicroBlaze V Processor Reference Guide* (UG1629) for more information.

MicroBlaze V can be implemented either as a 32-bit processor or a 64-bit processor, depending on user requirements. In general, AMD 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-bit, provides instructions to handle 64-bit data, and can transparently address instructions and data using up to a 64-bit address.

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

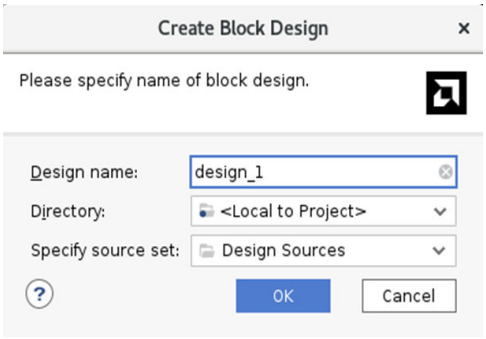
Creating a MicroBlaze V Processor Design

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

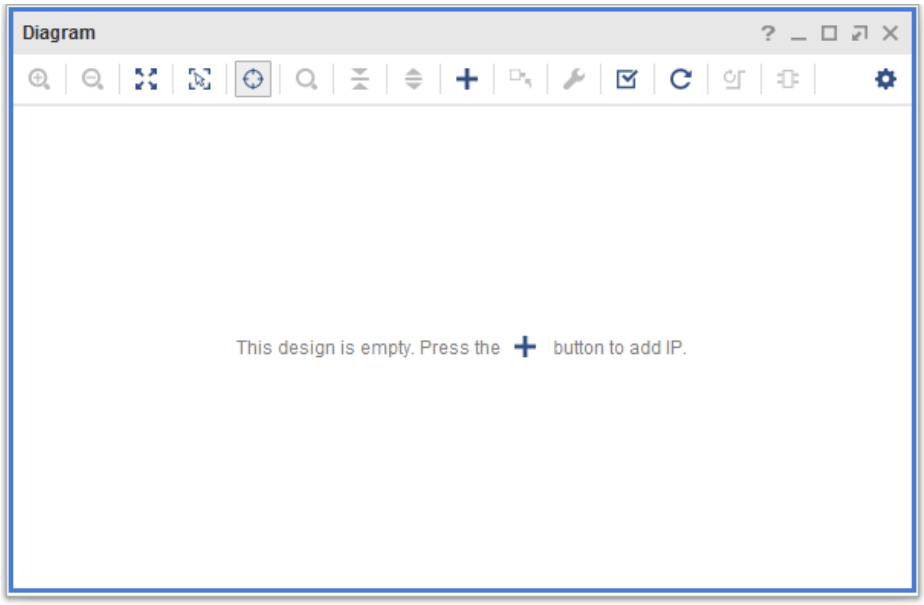
A variety of IP are available in the Vivado IDE IP catalog to meet the requirements of complex designs. You can also add custom IP to the IP catalog.

Designing with the MicroBlaze V 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.

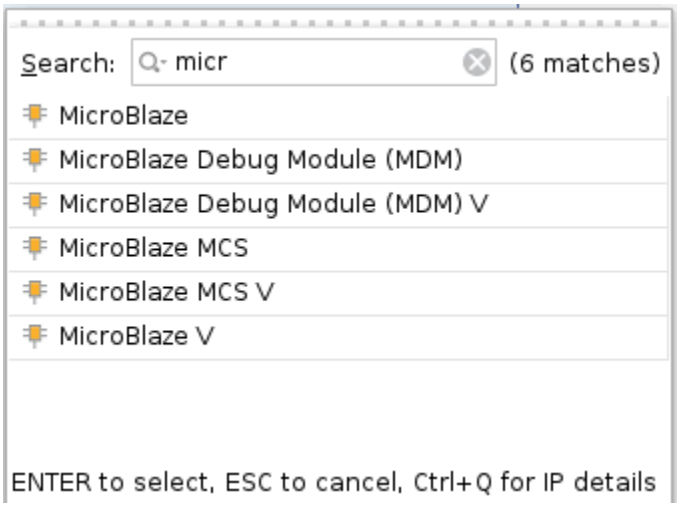


The Block Design window opens, as shown in the following figure.



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

   A Search box lets you search and select the MicroBlaze V processor as shown in the following figure.



When you double-click the MicroBlaze V 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_riscv:1.0
microblaze_riscv_0
```

4. Double-click the MicroBlaze V IP in the canvas to invoke the Re-customize IP process, which displays the Re-customize IP dialog box for the MicroBlaze V processor, shown in the Using the MicroBlaze V Configuration Wizard.

Converting from a Classic MicroBlaze Design

When opening a classic MicroBlaze design in IP integrator, a Block Automation feature is available to automatically convert the design to MicroBlaze V.

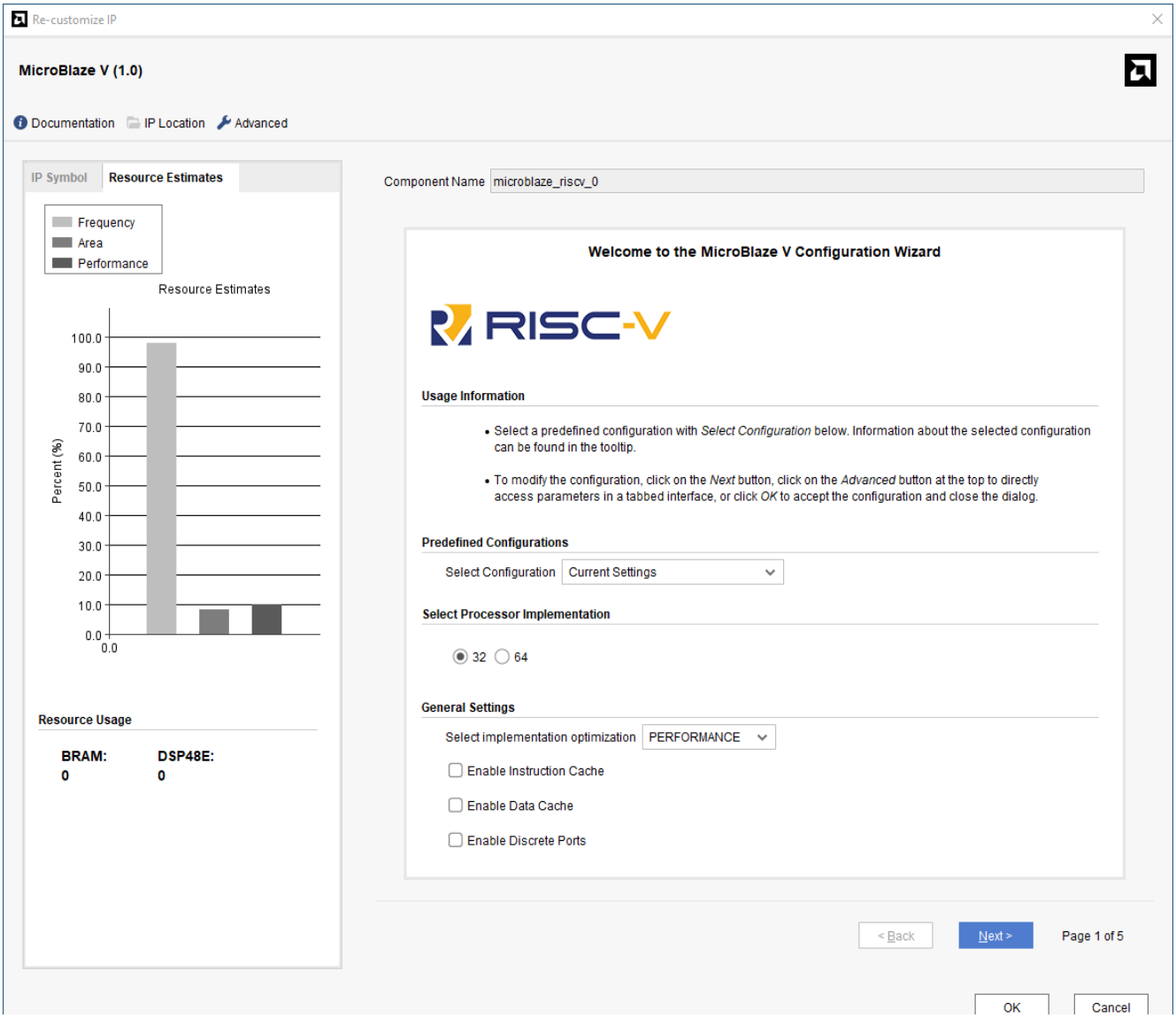
- **Keep Classic MicroBlaze**: This option is checked by default. To perform the conversion, it must be unchecked before clicking OK. If block automation is dismissed without performing the conversion, it can be reactivated by setting the MicroBlaze parameter `C_ENABLE_CONVERSION`.
- **Convert All Processors**: This option is only shown for designs with more than one MicroBlaze processor, and allows converting all simultaneously.

- **Enable Compressed Instructions**: This option enables the RISC-V Standard Extension for Compressed Instructions ("C").

Using the MicroBlaze V Configuration Wizard

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

Figure 7: **MicroBlaze V Configuration Wizard**



The Configuration wizard provides the following features.

- Predefined configuration templates for one-click configuration.

- Estimates of MicroBlaze V 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 V Configuration Wizard: Advanced Mode.

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

The MicroBlaze V 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 V Configuration Wizard: Welcome Page for more information.
- **General**: Shows the selection of execution units and optimization settings. See the MicroBlaze V Configuration Wizard: General Page for more information.
- **Exceptions**: Shows the Exceptions page when you select **Enable Selections** that option on the Welcome Page.
- **Cache**: Cache settings page is shown when you select **Use Instructions and Data Caches**. See the MicroBlaze V Configuration Wizard: Cache Page for more information.
- **Debug**: Shows the number of breakpoints and watchpoints when you select **Enable MicroBlaze Debug Module Interface**. See the MicroBlaze V 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 V 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 V 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 V 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 V processor. The instruction and data caches, and the branch target cache use block RAMs, and 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 V processor. The integer multiplier, and the floating point unit (FPU) use this total value to implement multiplication.

MicroBlaze V Configuration Wizard: Welcome Page

The simplest way to use the MicroBlaze V Configuration wizard is to select one of the six predefined templates, each defining a complete MicroBlaze V 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:

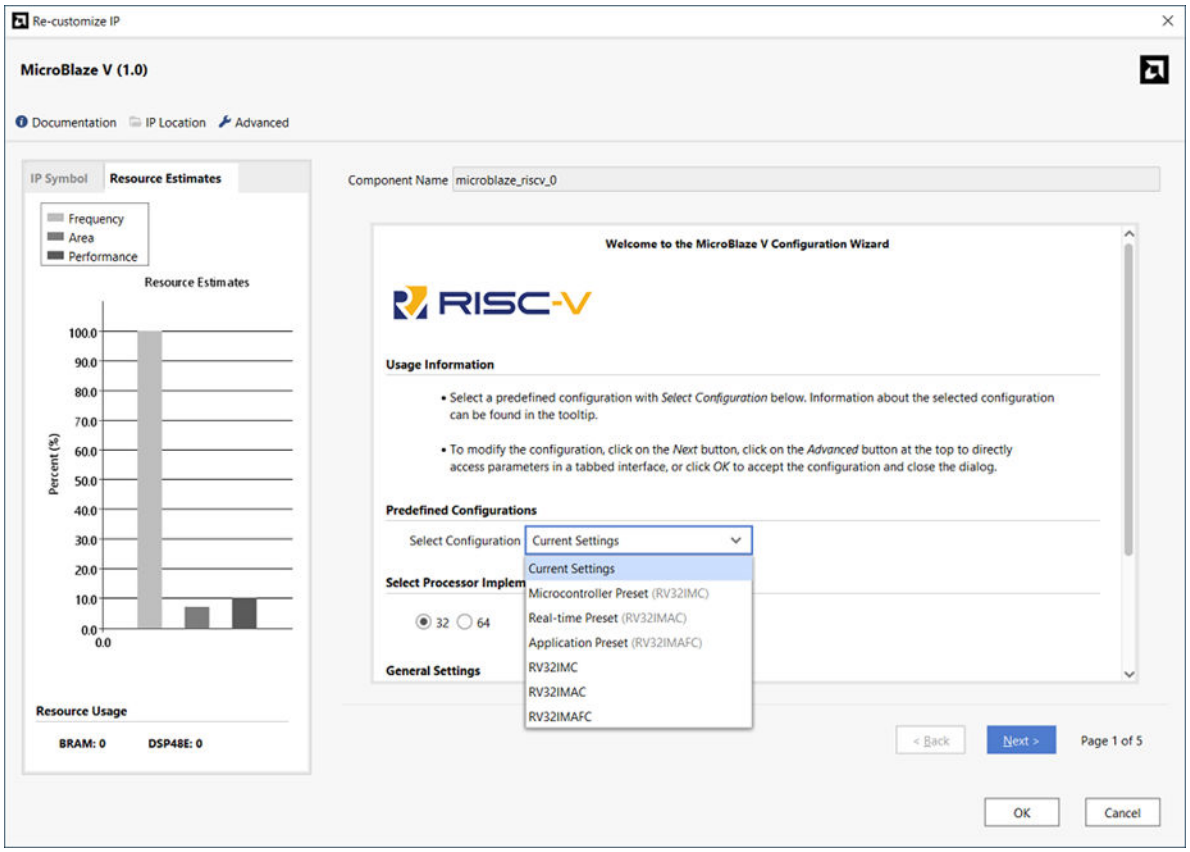
- **Microcontroller preset**: Microcontroller preset suitable for microcontroller designs. Area optimized, with no caches and debug enabled.
- **Real-time preset**: Real-time preset geared towards real-time control. Performance optimized, small caches and debug enabled, most execution units.
- **Application Preset**: Application preset design for high performance applications with floating-point, Supervisor Mode, and Page-based Virtual Memory.

The other options are:

- **RV32IMC**: A RISC-V configuration that includes the M-extension with multiplication and division, and the compressed instruction C-extension.
- **RV32IMAC**: A configuration that adds the A-extension with atomic instructions.
- **RV32IMAFC**: A configuration that adds the single-precision floating-point F-extension.

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

Figure 8: **MicroBlaze V 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™ V processor, based on the 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 balance computational performance and achievable frequency 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.
  - **THROUGHPUT**: Implementation selected to maximize computational performance, using a four-stage pipeline.

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

- **Enable Instruction Cache, Enable Data Cache**: You can use MicroBlaze V 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
  - RISC-V Cache Block Management instructions to invalidate, clear, or flush cache lines
  - Optional parity protection, invalidates cache lines if Block RAM bit error is detected
  - Optional data width selection to either use 32-bit, an entire cache line, or 512-bit

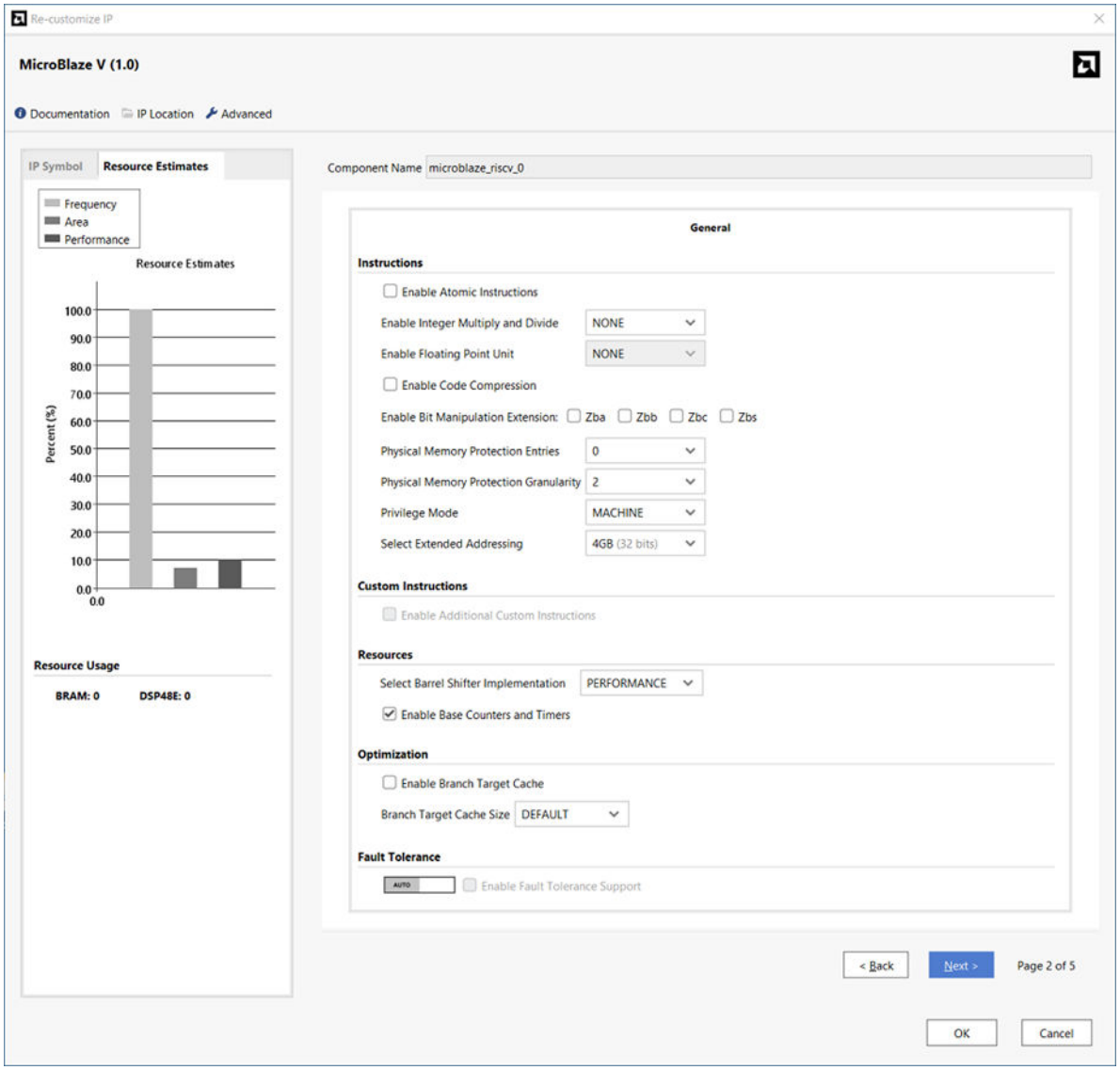
  ***Note*:** Activating caches significantly improves performance when using external memory. Ensure to select small cache sizes to reduce resource usage.

- **Enable Discrete Ports**: Enables discrete ports on the MicroBlaze V instance, which is useful for:
  - Generating custom external break interrupt or non-maskable interrupt. (`Ext_BRK`, `Ext_NM_BRK`)
  - Managing processor sleep and wakeup (`Sleep`, `Hibernate`, `Suspend`, `Wakeup`, `Dbg_Wakeup`)
  - Handling debug events (`Dbg_Stop`, `Halted`)
  - Pausing the processor (`Pause`, `Pause_Ack`, `Dbg_Continue`)
  - Setting reset mode (`Reset_Mode`)
- Manage non-secure and secure AXI transactions (`Non_Secure`)

MicroBlaze V Configuration Wizard: General Page

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

Figure 9: **MicroBlaze V Configuration Wizard: General Page**



Instructions

- **Enable Atomic Instructions**: Enables atomic instructions according to the RISC-V "A" Standard Extension for Atomic Instructions.
- **Enable Integer Multiply and Divide**: Enables hardware integer multiplication, division, and remainder instructions according to the RISC-V "M" Standard extension for Integer Multiplication and Division.

- **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 64-bit implementation. Using the FPU significantly improves the floating point performance of the application and significantly increases the size of MicroBlaze V.

  The FPU implements instructions according to the RISC-V 'F' and 'D' Standard Extensions for Floating Point.

  The FPU also requires that Integer Multiply and Divide is enabled.

  The compiler automatically uses the FPU instructions corresponding to setting of this parameter.
- **Enable Code Compression**: Enable compressed instructions according to the RISC-V "C" Standard Extension for Compressed Instructions.
- **Enable Bit Manipulation Extension**: Enable bit manipulation instructions according to RISC-V Bit-Manipulation extensions.
- **Privilege Mode**: Set the available RISC-V privilege modes to Machine, User or Supervisor mode.
- **Select Extended Addressing**: Set the memory addressing capability.
  - With the 32-bit processor implementation using privilege mode MACHINE or USER, this enables additional custom load/store instructions to be able to access a larger address space than 4 GB (32-bit address). With extended address the data side LMB and AXI bus addresses are extended to the number of address bits corresponding to the selected memory size.
  - With the 32-bit processor implementation with privilege mode SUPERVISOR, the extended address is fixed to Sv32 (32-bit virtual address space). In this case, the LMB and AXI bus addresses are 34 bits.
  - With the 64-bit processor implementation using privilege mode MACHINE or USER, 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.
  - With the 64-bit processor implementation using privilege mode SUPERVISOR, the extended address can be Sv39, Sv48, or Sv57 (39-bit, 48-bit, or 57-bit virtual address space). In this case, the LMB and AXI bus addresses are 56 bits.
- **Enable Additional Custom Instructions**: Provides additional functionality when using AXI4-Stream links.

  The instructions are also extended with the following variants.
  - 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.*

Optimization

- **Enable Branch Target Cache**: When set, implements the branch target, which improves branch performance by predicting conditional branches and caching branch targets.
- **Branch Target Cache Size**: Specify the size of the cache for branch targets.

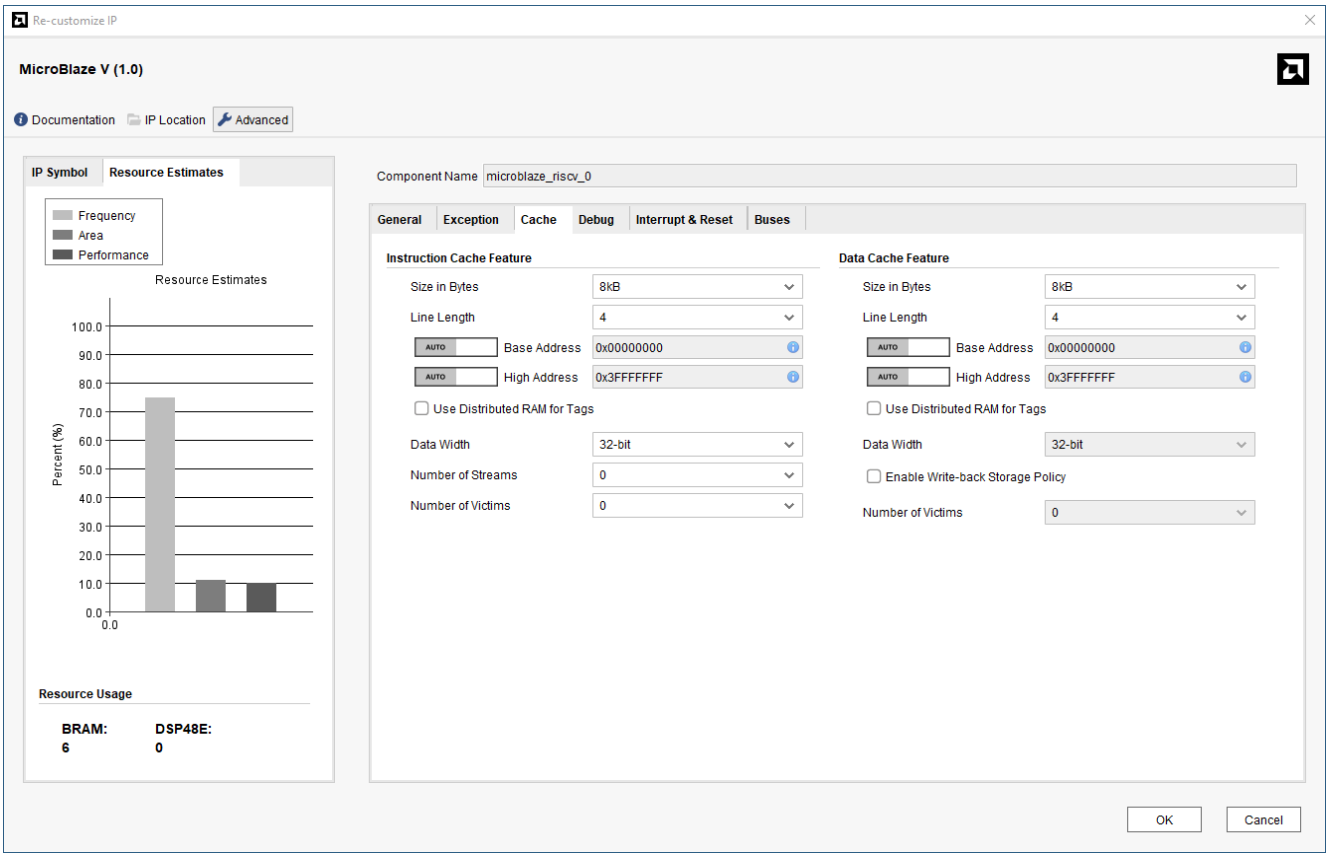
Fault Tolerance

- **Auto**/**Manual**: Determines if the Vivado tool automatically enables fault tolerance, or if you specify it manually.
- **Enable Fault Tolerance Support**: When enabled, MicroBlaze V 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 V when ECC is enabled in connected LMB block RAM controllers.
  - If fault tolerance is explicitly enabled, the IP integrator tool enables ECC automatically in connected LMB block RAM Controllers.
  - If fault tolerance is explicitly disabled, ECC in connected LMB block RAM controllers is not affected.

MicroBlaze V Configuration Wizard: Cache Page

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

Figure 10: **MicroBlaze V Configuration Wizard: Cache Options**



- **Enable Instruction Cache**: The Instruction Cache Feature 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.
  - **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, 16 KB, or 32 KB 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 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 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 AXI Coherency Extension (ACE).

- **Enable Data Cache**: The Data Cache Features are:
  - **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.
  - **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 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, 16 KB, or 32 KB 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 AXI Coherency Extension (ACE) or when you enable fault tolerance.

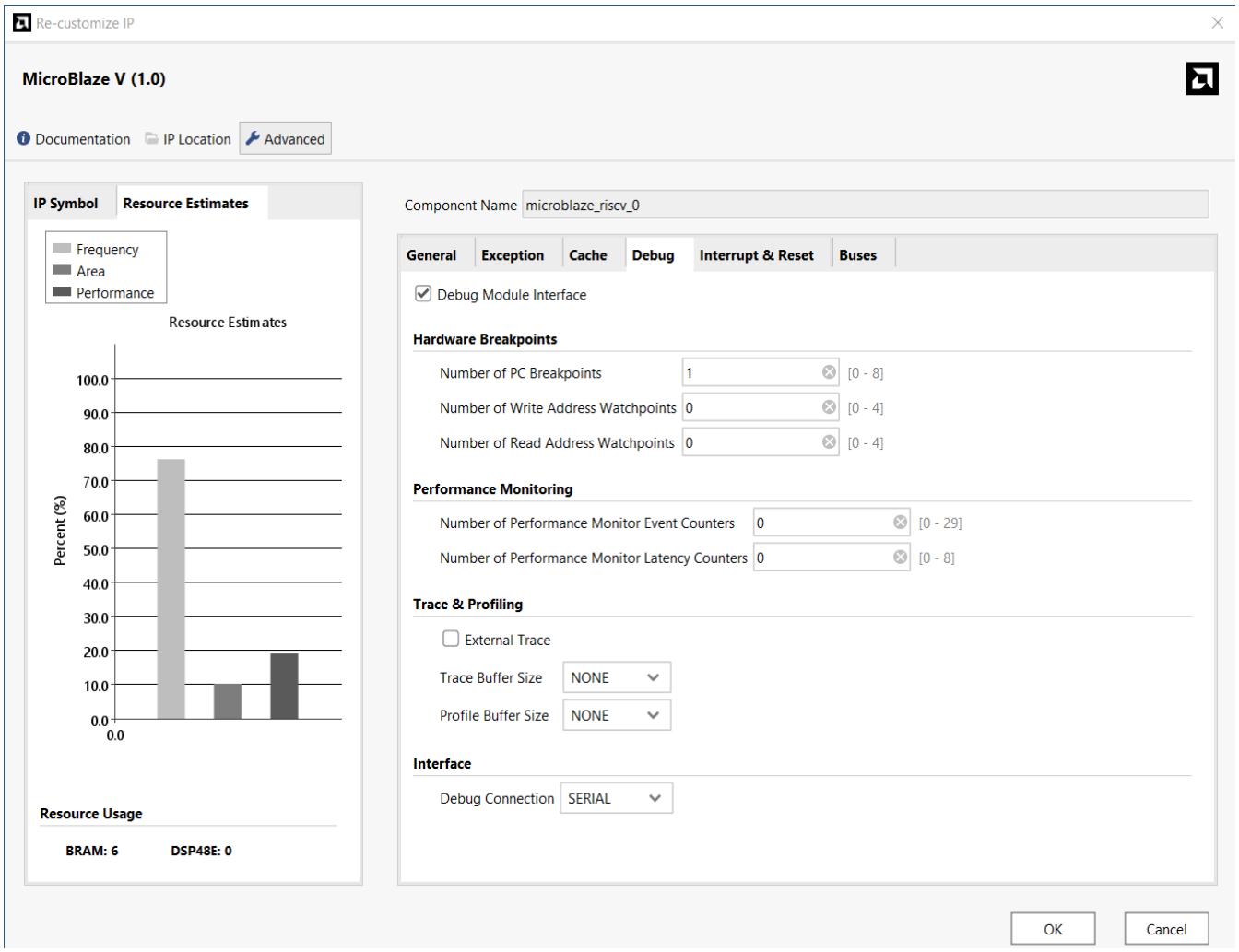
- **Enable Write-back Storage Policy**: This parameter enables use of a write-back data storage policy. When this policy is in effect, the data cache only writes data to memory when necessary, which improves performance in most cases. With write-back enabled, data is stored by writing an entire cache 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.
- **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 AXI Coherency Extension (ACE).

MicroBlaze V Configuration Wizard: Debug Page

Figure 11: **MicroBlaze V Configuration Wizard: Debug**



Debug Options

MicroBlaze Debug Module Interface

- Enables the MicroBlaze Debug Module V (MDM V) 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.

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.*

Performance Monitoring

MicroBlaze V provides the following hardware 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.

With the default configuration, the counter width is set to 64-bit.

Trace and Profiling

MicroBlaze V provides program trace, based on the RISC-V N-Trace standard, storing information in an embedded RAM Sink or sending it to an external trace PIB Sink.

Use the parameter `C_DEBUG_TRACE_SIZE` to configure the size of the embedded trace buffer from 4KB to 128KB. By setting `C_DEBUG_TRACE_SIZE` to `0` (None), embedded program trace is disabled.

Use the parameter `C_DEBUG_EXTERNAL_TRACE` to enable external trace. In this case the MDM V must also be configured to use external trace output.

MicroBlaze V also provides non-intrusive profiling, identical to the equivalent feature in the classic MicroBlaze processor, 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 4KB to 128KB. By setting the parameter to 0 (None), profiling is disabled.

MicroBlaze V 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 V 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.

AXI and ACE Interfaces

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

  ***Note*:** To use ACE, do not set area optimization, write-back data cache, instruction cache streams, or victims cache data widths other than 32-bit. 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.

- **Enable AXI Slave Interface**: If this parameter is set, the AXI slave interface is available. This interface can be used to access Debug Trace and Debug Profiling registers, instead of using the default MDM System Bus.

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 traps. This can be useful to find error conditions or interrupt related issues.

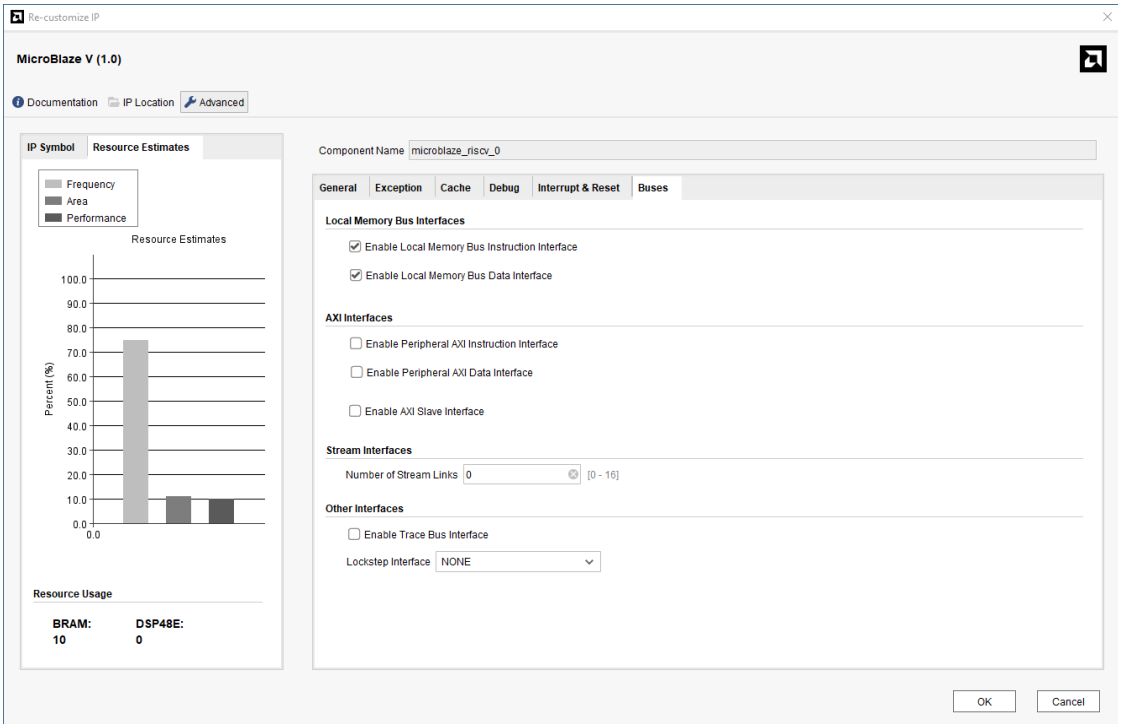
  For a complete description of all the Trace bus interface signals, see section Trace Interface Description in *MicroBlaze V Processor Reference Guide* (UG1629).

- **Lockstep Interface**: When you enable lockstep support, two MicroBlaze V 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 12: **MicroBlaze V Configuration Wizard: Buses**



- **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 instruction 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.

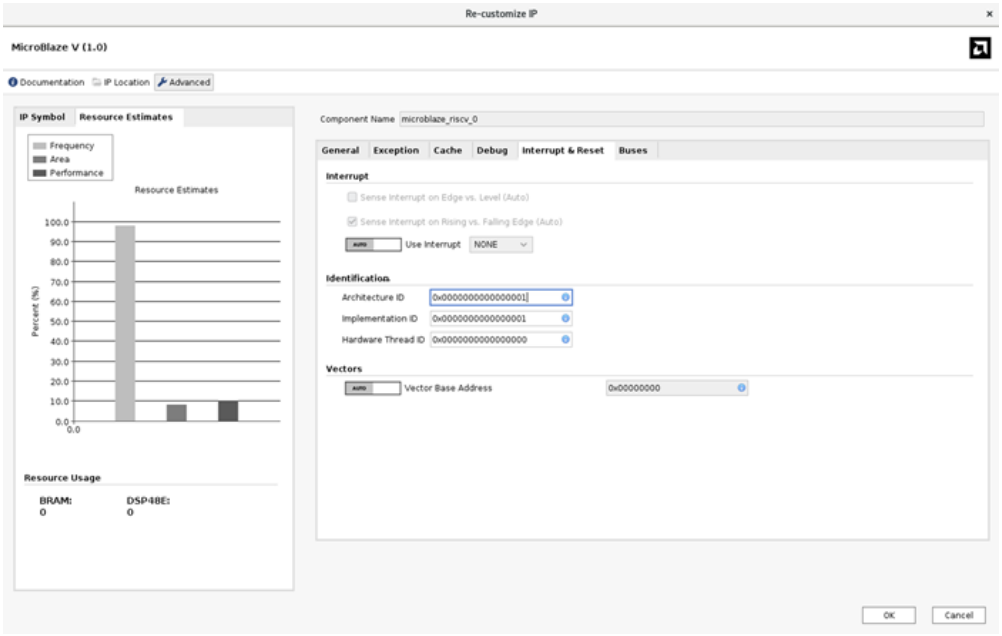
There is also a monitor option for interrupt on the **Interrupt & Reset** tab:

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 V Configuration Wizard: Advanced Mode

Accessible through the Advanced button on the Welcome page of the MicroBlaze V 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 13: **Advanced Mode: Interrupt and Reset Tab**



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

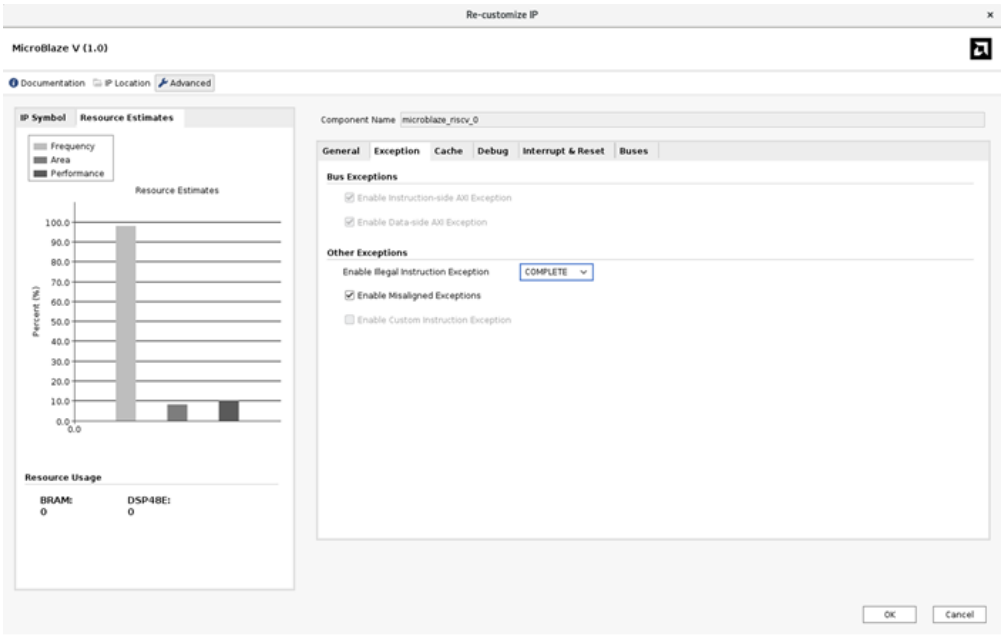
- MicroBlaze V Configuration Wizard: General Page
- MicroBlaze V Configuration Wizard: Cache Page
- MicroBlaze V Configuration Wizard: Debug Page
- MicroBlaze V Configuration Wizard: Buses Page

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

MicroBlaze V Advanced Mode Exception Tab

The following figure shows the MicroBlaze V Exception options page.

Figure 14: **MicroBlaze V Configuration Wizard: Exception Options**



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

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 instruction is invalid.
  - When set to NONE, no illegal instruction checks are done.
  - When set to BASIC, only illegal opcodes are detected.
  - When set to COMPLETE, all illegal instructions and CSR addresses are detected.

- **Enable Misaligned Data Exception**: When enabled, the tools automatically insert software to handle misaligned accesses.
- **Enable Stream Exception**: Enables stream exception handling for Advanced eXtensible Interface (AXI) read accesses.

> **IMPORTANT!** *You must enable additional stream instructions to use custom exception handling.*

MicroBlaze Advanced Mode Interrupt & Reset Tab

MicroBlaze V 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, 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 is triggered.

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

Vectors

- **Vector Base Address**: Change the base address used for MicroBlaze V reset vectors. See the *MicroBlaze V Processor Reference Guide* (UG1629) 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.

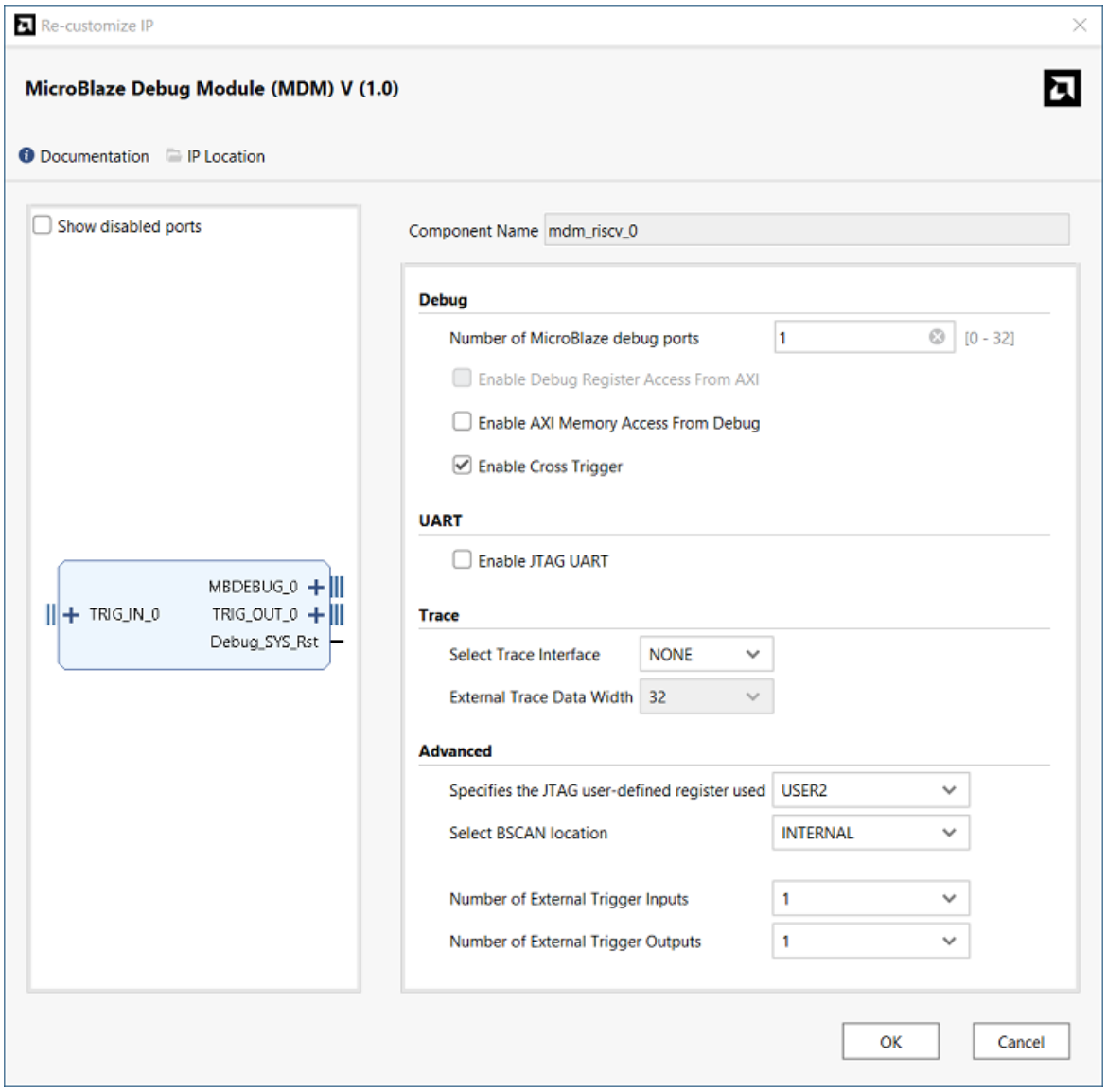
Cross-Trigger Feature of MicroBlaze Processors

Cross trigger support is available with the MDM V. The MDM V provides programmable cross triggering between all connected processors, external trigger inputs, and outputs using the RISC-V Halt Groups, Resume Groups, and External Triggers functionality. For details, see the *MicroBlaze Debug Module V (MDM V) LogiCORE IP Product Guide* (PG428).

MicroBlaze V can handle halt and resume cross trigger actions. Cross trigger actions are generated by the corresponding MDM V cross trigger outputs, connected using the Debug bus.

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

Figure 15: **Enable Cross Trigger Check Box in MDM V**

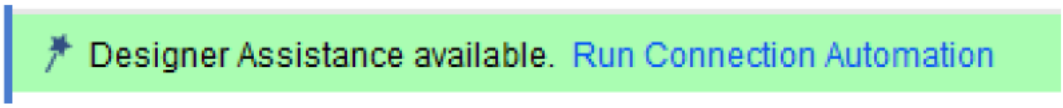


You can also select up to four external trigger inputs and external trigger outputs. When enabled, the block design is updated to show the MDM details, as shown in the following figure.

Figure 16: **MDM V in Block Design after enabling Cross Trigger**

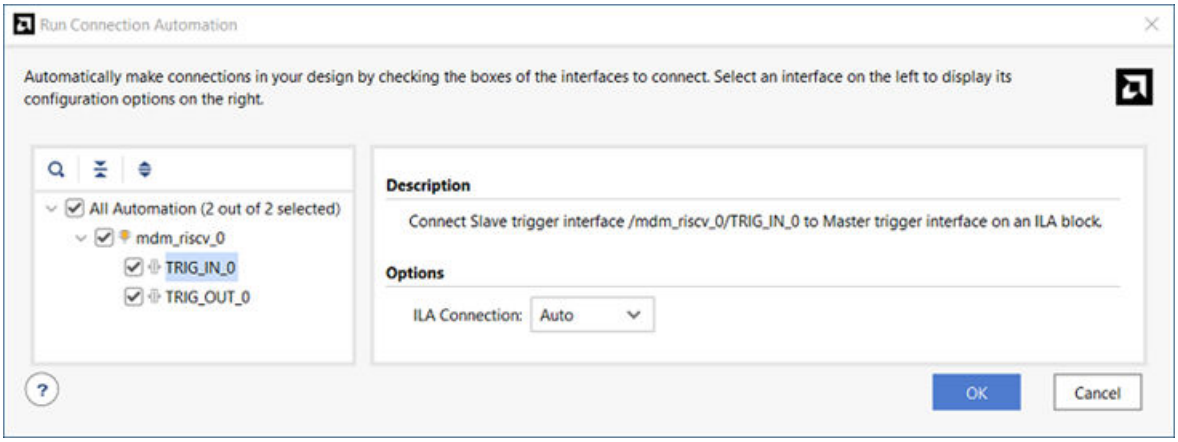
Next, run Connection Automation, shown in the following figure, to connect the cross trigger signals to an ILA.

Figure 17: **Connecting the TRIG_IN_0 Interface Pin to an ILA**



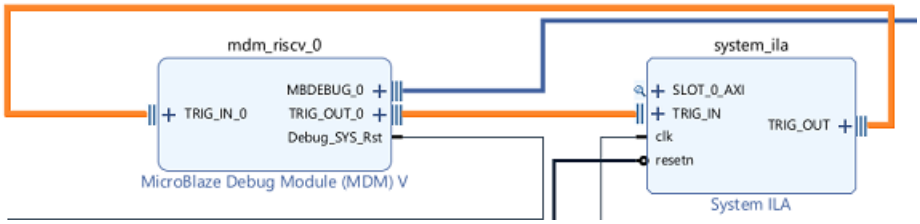
Leaving the settings of Auto, as shown in the following figure, on both `TRIG_IN_0` and`TRIG_OUT_0` in the Run Connection Automation dialog box, instantiates a new ILA and connects the `TRIG_IN_0` and `TRIG_OUT_0` signal of the MDM V to the corresponding pin of the System ILA.

Figure 18: **Run Connection Automation Confirmation Dialog Box**



The following figure shows the resulting block design.

Figure 19: **Block Design after connecting Cross Trigger Pins to the ILA**



Completing Connections

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

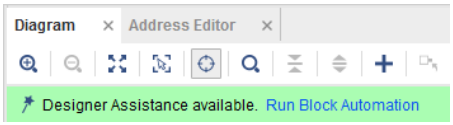
In the IP integrator canvas, right-click and select **Add IP**. You can use two built-in features of the IP integrator to complete the rest of the IP subsystem design: the Block Automation and Connection Automation features assist you with putting together a basic microprocessor system in the IP integrator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

Block Automation

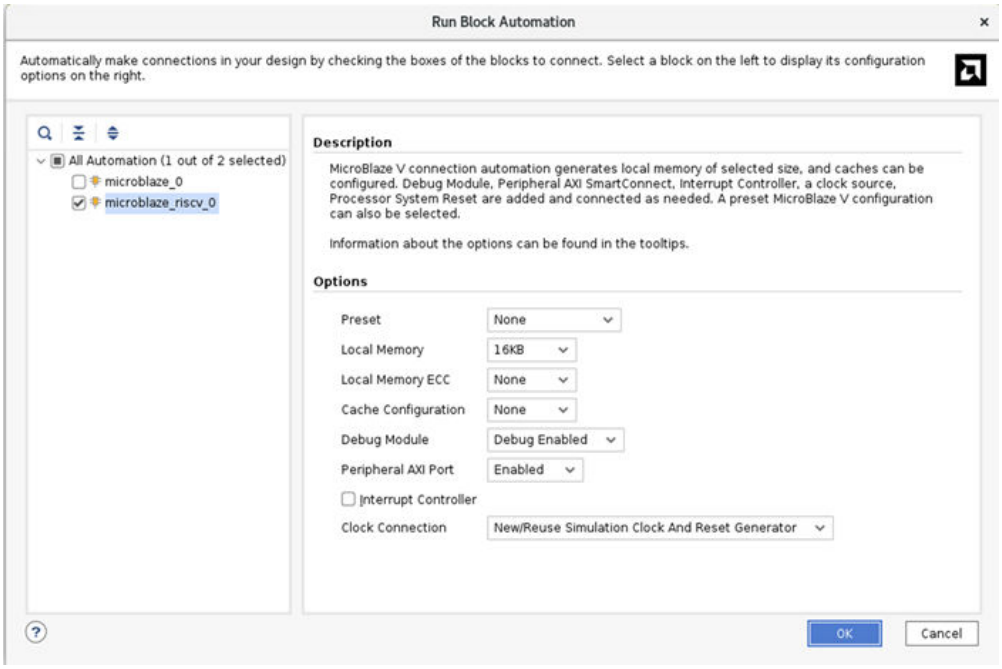
The Block Automation feature is available when you instantiate a microprocessor in the block design of the IP integrator.

***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 V System.



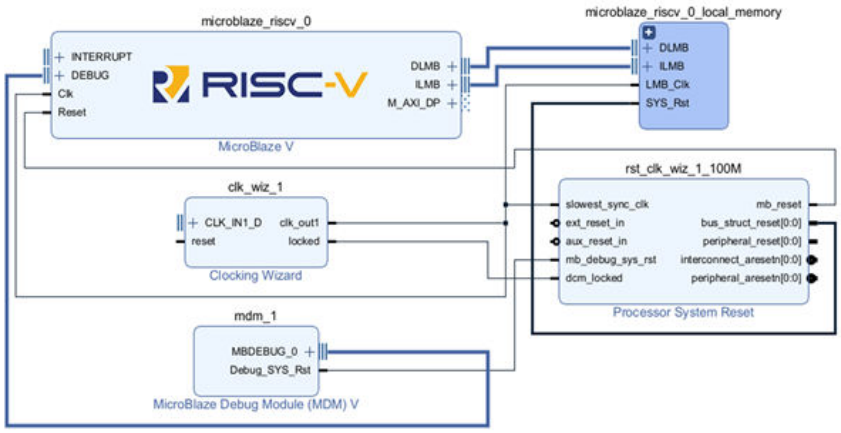
   The Run Block Automation dialog box lets you provide input about basic features that the microprocessor system requires. The following figure shows the Run Block Automation dialog box.



The MicroBlaze V Preset option provides a convenient way of configuring the processor settings according to the particular use case: microcontroller, real-time, or application. If necessary, the MicroBlaze V Configuration wizard provides configurations.

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

   The Run Block Automation creates the following MicroBlaze V system.

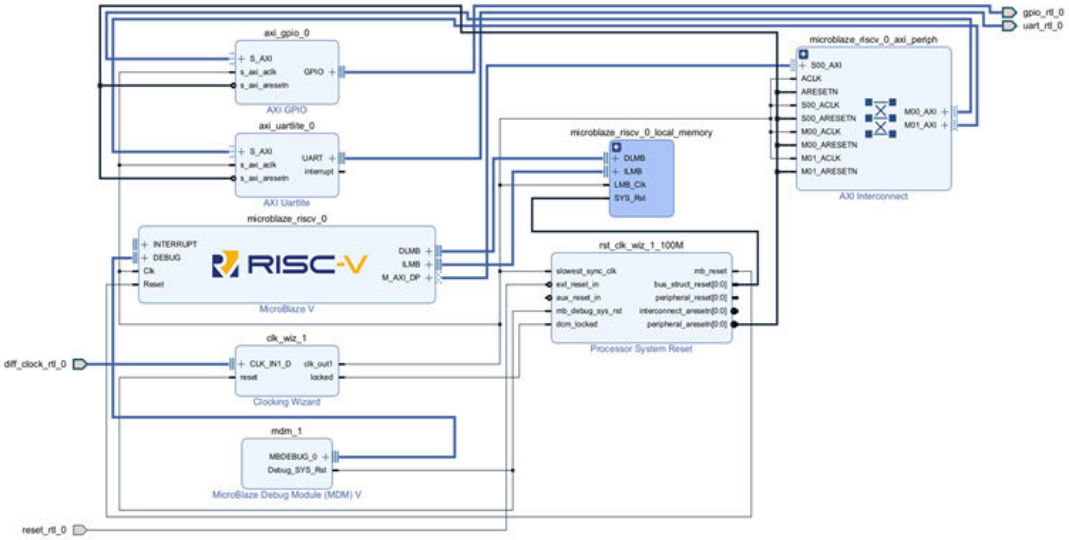


Using Connection Automation

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

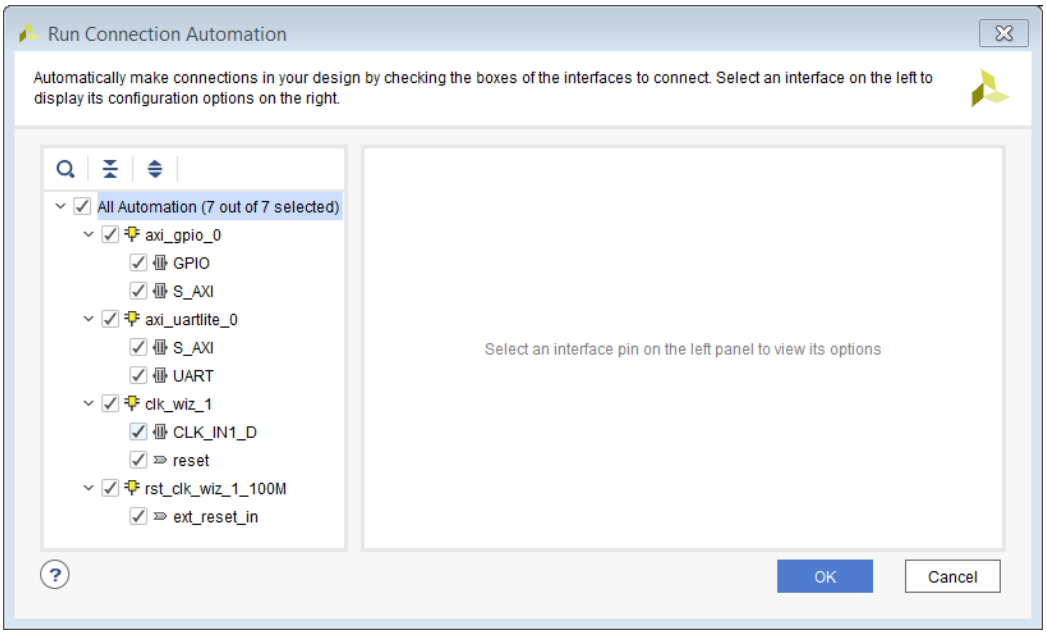
In the following figure, two IP, the GPIO, and the UARTLite are instantiated along with the MicroBlaze V processor subsystem.

Figure 20: **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 21: **The Run Connection Automation Dialog Box**

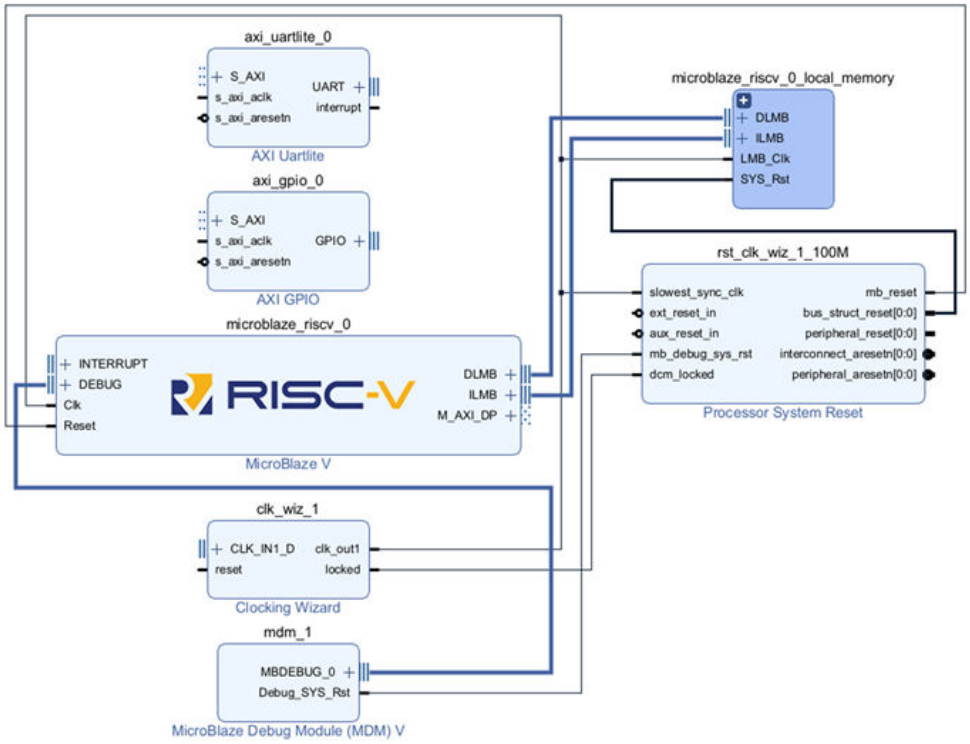


The IP integrator determines that there is a potential connection for the following objects:

- 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 22: **Running Connection Automation for a Sample MicroBlaze V 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 V 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 does not assess or improve the performance of the design.

  IMPORTANT! *The Vivado Design Suite does not support UCF format.*

The following are the options on how to use constraint sets:

- 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 AMD 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, run synthesis, implementation, and 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 download the device and run the software on the processor.

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

For detailed procedure, see Exporting a Hardware Description.

Multiple MicroBlaze Processor Designs

Multiple MicroBlaze processors are allowed in a block design. The configurations of these multiple MicroBlaze designs varies based on the design requirements. A simple dual MicroBlaze design is discussed in the following sections.

Instantiate MicroBlaze V IP Cores

Create a block design and instantiate two instances of MicroBlaze V IP as shown in the following figure.

Figure 23: **Multiple MicroBlaze V Instances in a Block Design**

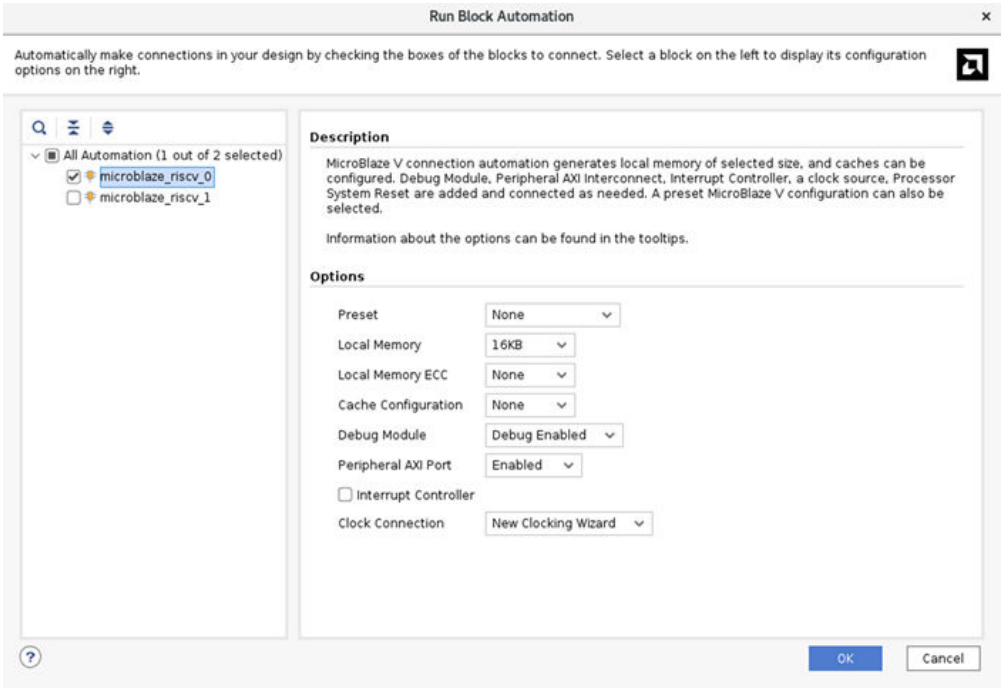
The Run Block Automation link is active in the banner. Click the **Run Block Automation** link to run block automation on both the MicroBlaze V instances. Again, the options here varies on the design requirements.

For example:

- Both the MicroBlaze V processors either run from a single system clock or totally independent.
- Both the MicroBlaze V processors either share Clocking Wizard IP or contain independent Clocking Wizard IP.

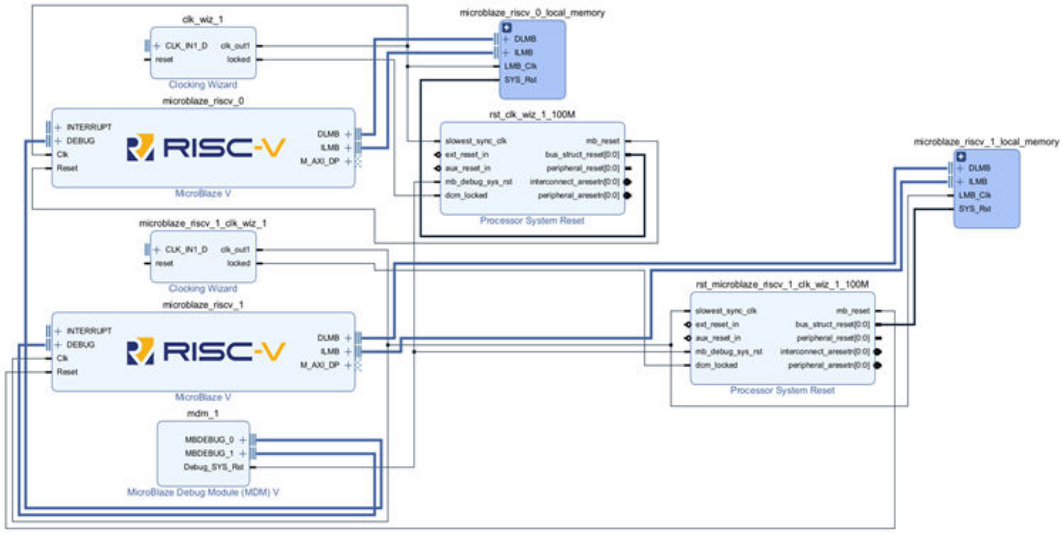
This topology shows two independent Clocking Wizard IP for each MicroBlaze V processor as in the following figure.

Figure 24: **Run Block Automation Dialog Box for Dual MicroBlaze V Design**



The block design looks as shown in the following figure:

Figure 25: **Block Design After Running Block Automation**



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

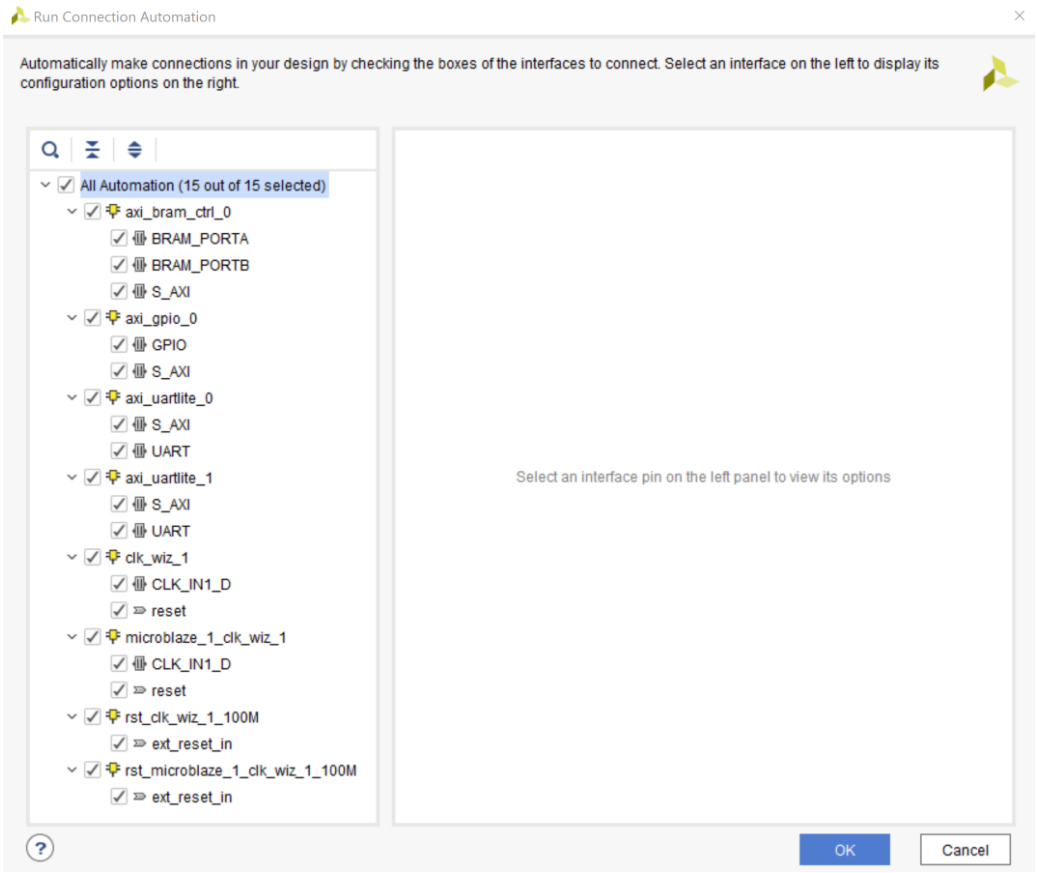
At this point you can add peripherals to your design as needed. In this case, two instances of AXI UART Lite and one GPIO were added.

- The AXI UART Lite IP is connected to each of the MicroBlaze V processor instances.
- The GPIO is connected to one instance of the MicroBlaze V IP.
- 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 26: **Run Connection Automation Dialog Box**



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

Table 1: **Run Connection Automation Options**

Connection	More Information	Setting
axi_gpio_0 • GPIO	The GPIO interface can be tied to several on-board interfaces.	Set the Select Board Part Interface to led_8bits (LED).
axi_gpio_0 • S_AXI	The Master field is set to its default value of/ microblaze_riscv_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_riscv_1 (Periph). All other fields is set to its default value of Auto.	Set the Master field to `/ microblaze_riscv_1 (Periph)`.
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.	Set the Select Board Part Interface field to Custom.
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 V (`microblaze_0`) is connected to two slaves AXI Uartlite (`axi_uartlite_0`) and AXI GPIO (`axi_gpio_0`). The other instance of MicroBlaze V (`microblaze_1`) is connected to the AXI Uartlite (`axi_uartlite_1`).

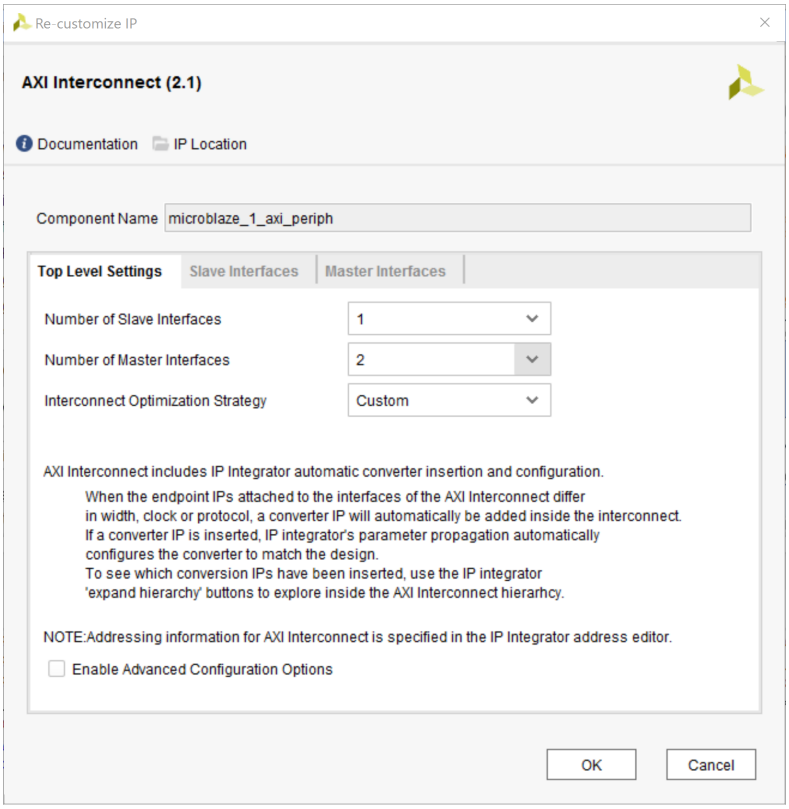
Re-Customizing AXI Interconnects

If you want the `microblaze_1` instance of MicroBlaze V to access the peripherals, the two interconnects instances must be reconfigured.

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

   The Re-customize IP dialog box opens.

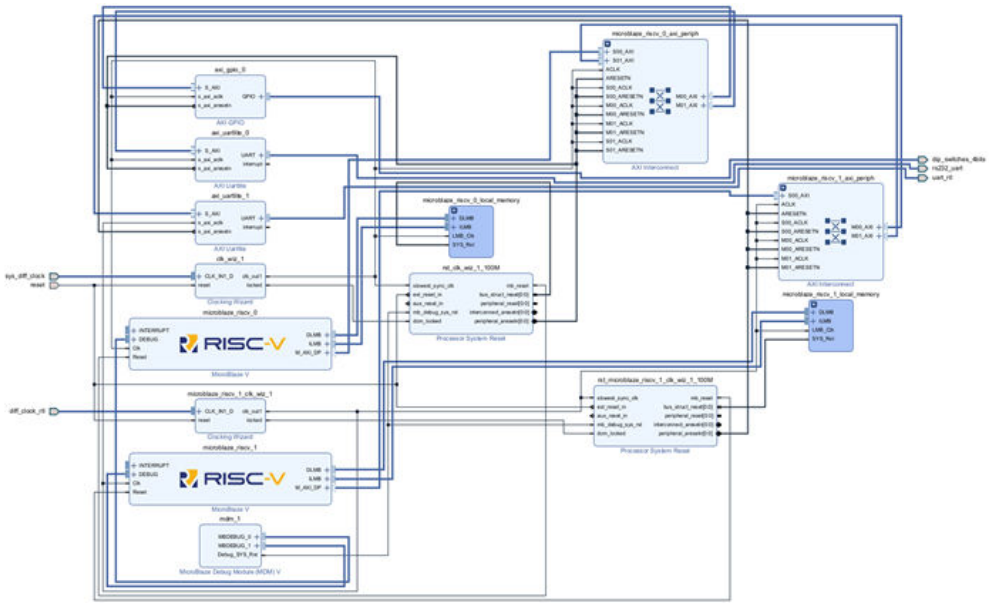
2. From the drop-down menu, set the **Number of Master Interfaces** field to **2**, as shown in the following figure.



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

3. 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 shown in the following figure.

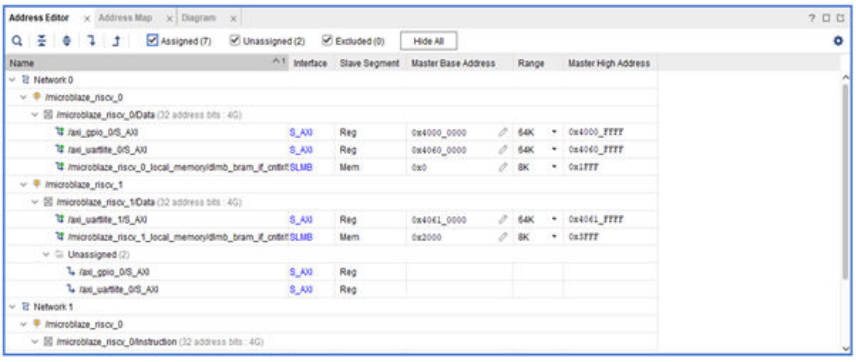


After successful completion of the above procedure, `microblaze_1` access all the slaves accessible by `microblaze_0`. This is an optional decision.

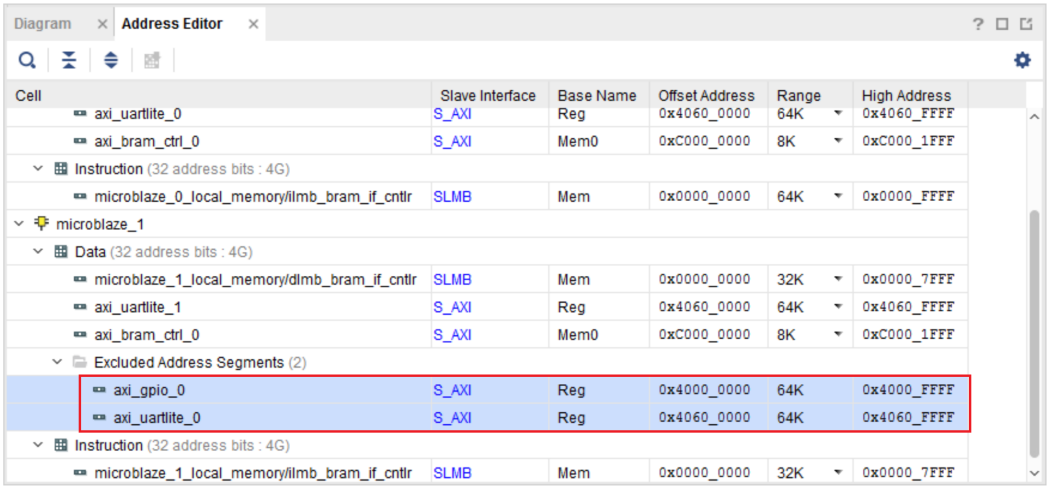
Mapping and Excluding Unwanted Slaves

If you want to access the AXI block RAM Controller from `microblaze_riscv_0`, you can exclude the other slaves from the `microblaze_riscv_0` memory space. This can be done in the Address Editor window. As can be seen in the following figure, the `microblaze_riscv_0` memory space has three unmapped slaves.

Figure 27: **microblaze_riscv_0 Memory Map**



1. Assign addresses to all the unmapped slaves by selecting them, right-clicking, and selecting **Assign Address** from the context menu.
2. Exclude the unwanted slaves from the memory map of `microblaze_1` by selecting them in the Address Editor window. Right-click and select **Exclude**.



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

Designing with the Memory IP Core

The AMD 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.

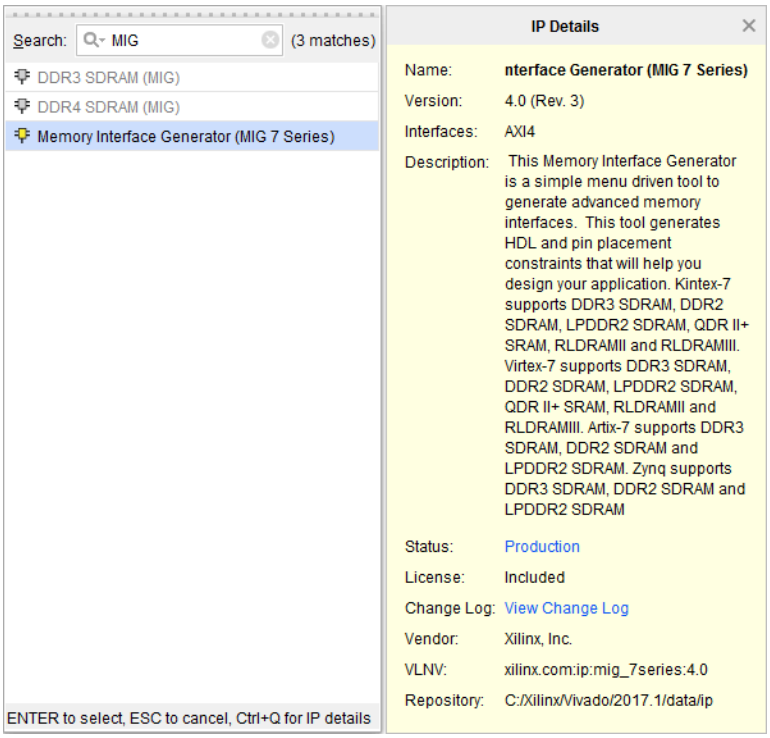
TIP: *Although the information in this chapter is tailored for the KC705, Kintex 7 board, the differences for UltraScale devices, and the KCU105 board, are highlighted throughout this text. These guidelines can also be applied to AMD devices on custom boards.*

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  .

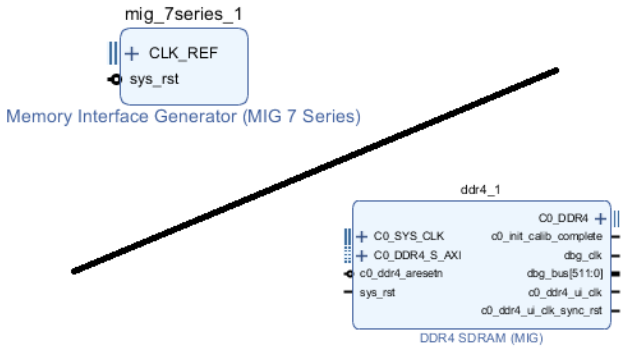
Double-click to select the Memory Interface Generator IP and add it to your block design.



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.

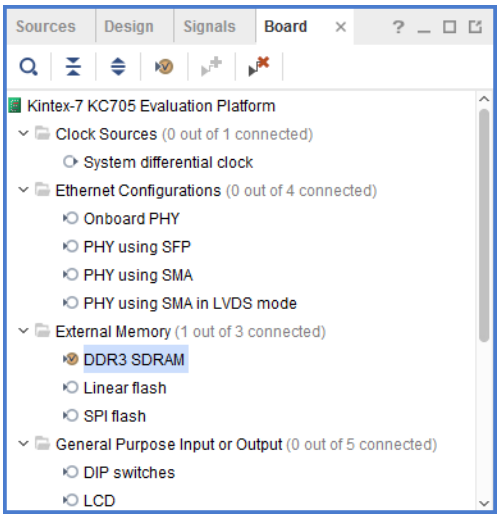
   The following figure shows both the Memory IP and the 7 series IP core in the upper-left, and the DDR4 Memory IP core for UltraScale devices in the lower-right. The Memory IP that is available in the IP catalog depends on the target part or platform board selected for your project. There are separate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



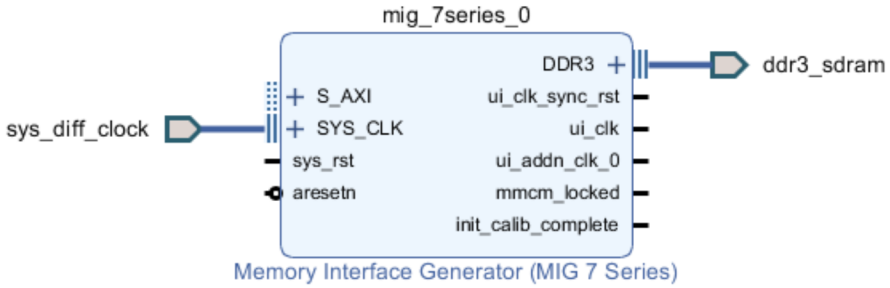
   This example targets the KC705 board for the project. As shown in the following figure, the Board window of the platform board flow is available to let you select components to interface to your design.

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

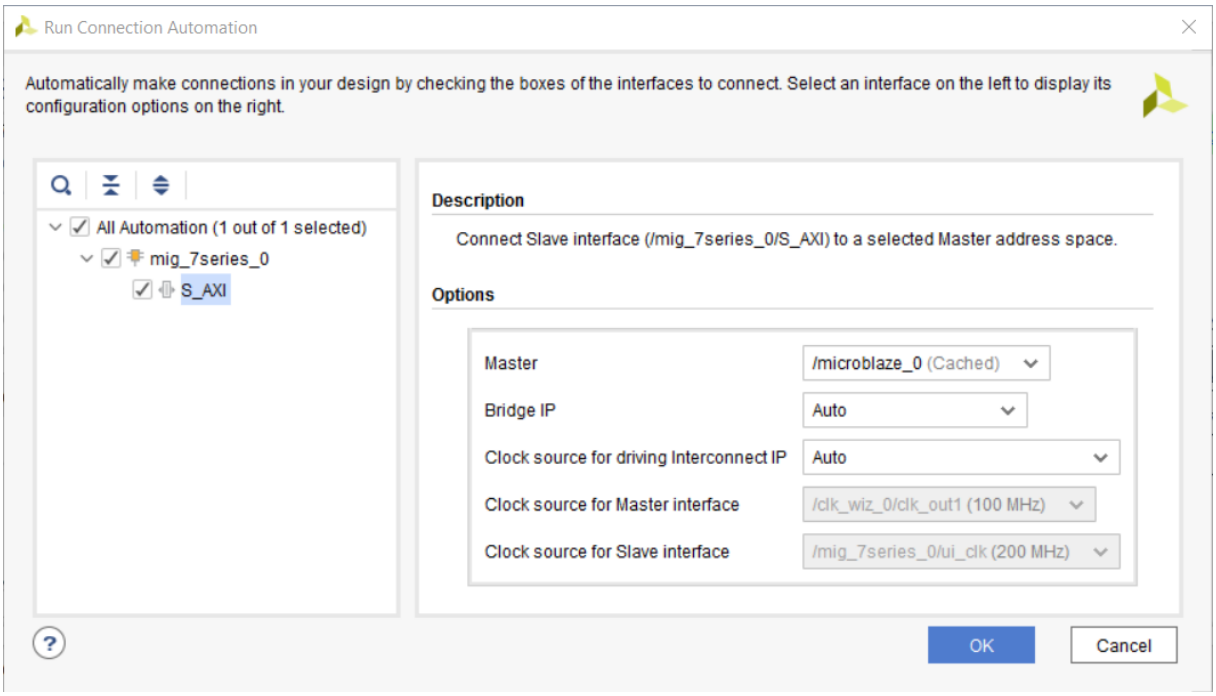
   ***Note*:** In the case of the UltraScale KCU105 board, you can also use the DDR4 SDRAM component.



   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. This connects the Memory IP to the system FPGA reset on the platform board.

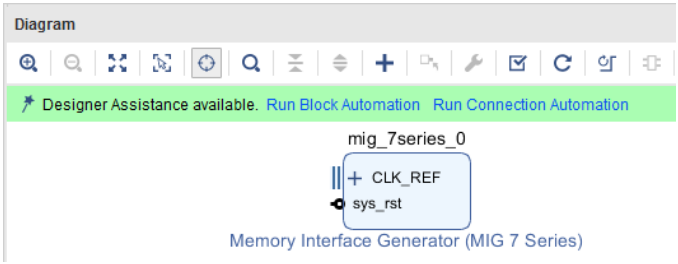


***Note*:** For the KCU105 board, the Run Connection Automation dialog box includes both the `CO_SYS_CLK` and the `sys_rst` interfaces for the Memory IP.

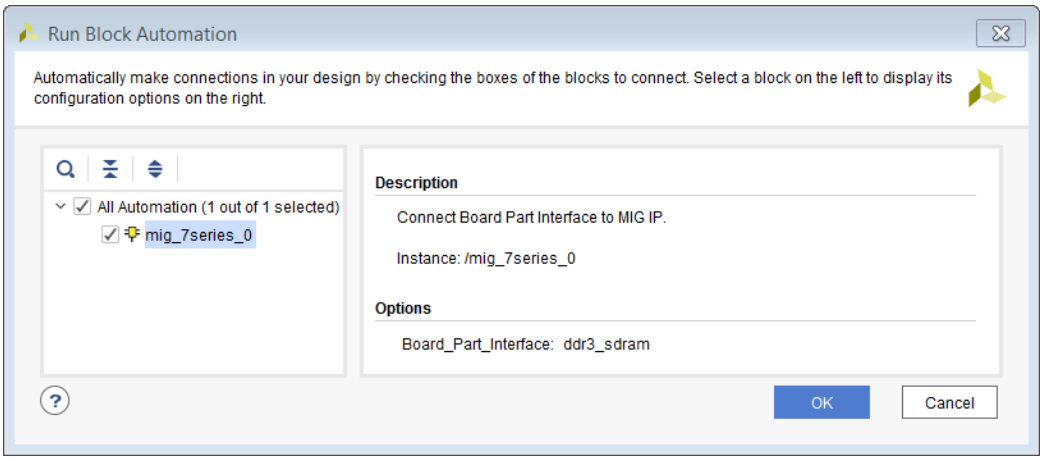
Making Connections with Block Automation

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.



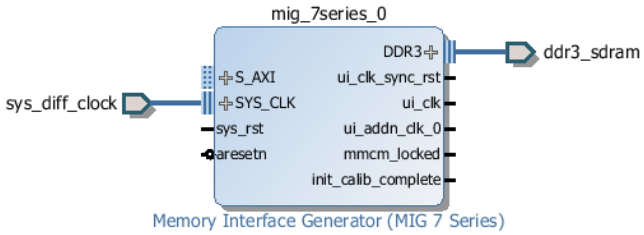
This opens the Run Block Automation dialog box as shown in the following figure.



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**.

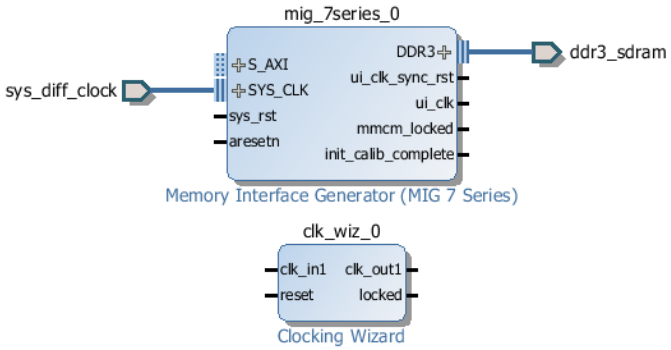
   The `SYS_CLK` and DDR interfaces of the Memory IP are connected to the DDR memory components on the platform board. The Memory IP core is configured for 400 MHz operation with the correct pins selected to interface to the KC705 board. The following figure shows the Memory IP core after running Block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



Adding a Clocking Wizard

A Clocking wizard IP is required in the block design to fulfill the clocking requirement in addition to the clock generated by the Memory IP core.

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.

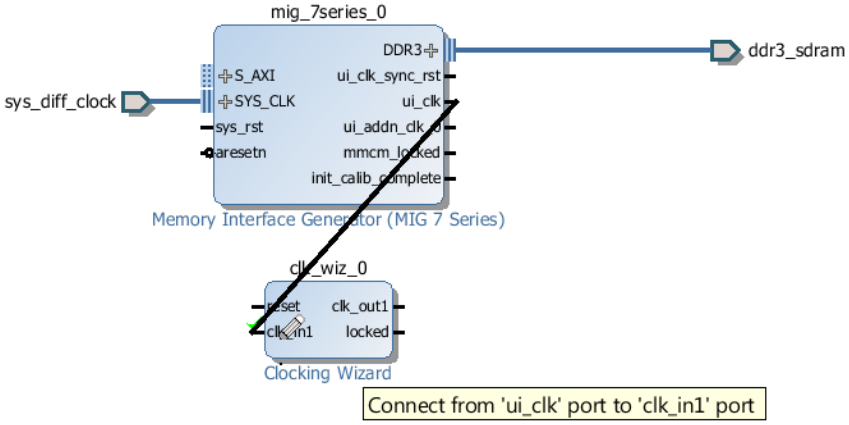


Follow these steps to connect the Clocking Wizard to the Memory IP core:

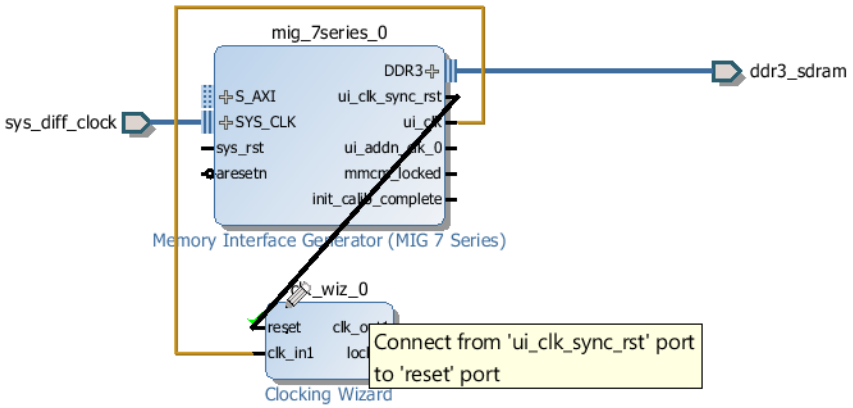
2. Connect the `ui_clk` or `ui_addn_clk_0` output of the Memory IP and any other clocks generated to the `clk_in1` input of the Clocking wizard, as shown in the following figure.

> **TIP:** *Ensure 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.



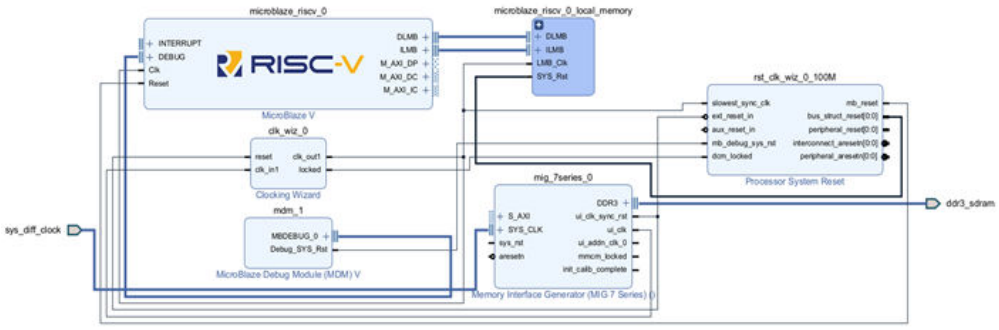
6. Configure the Clocking wizard to generate any required clocks for the design, by double-clicking the IP.

Adding an AXI Master

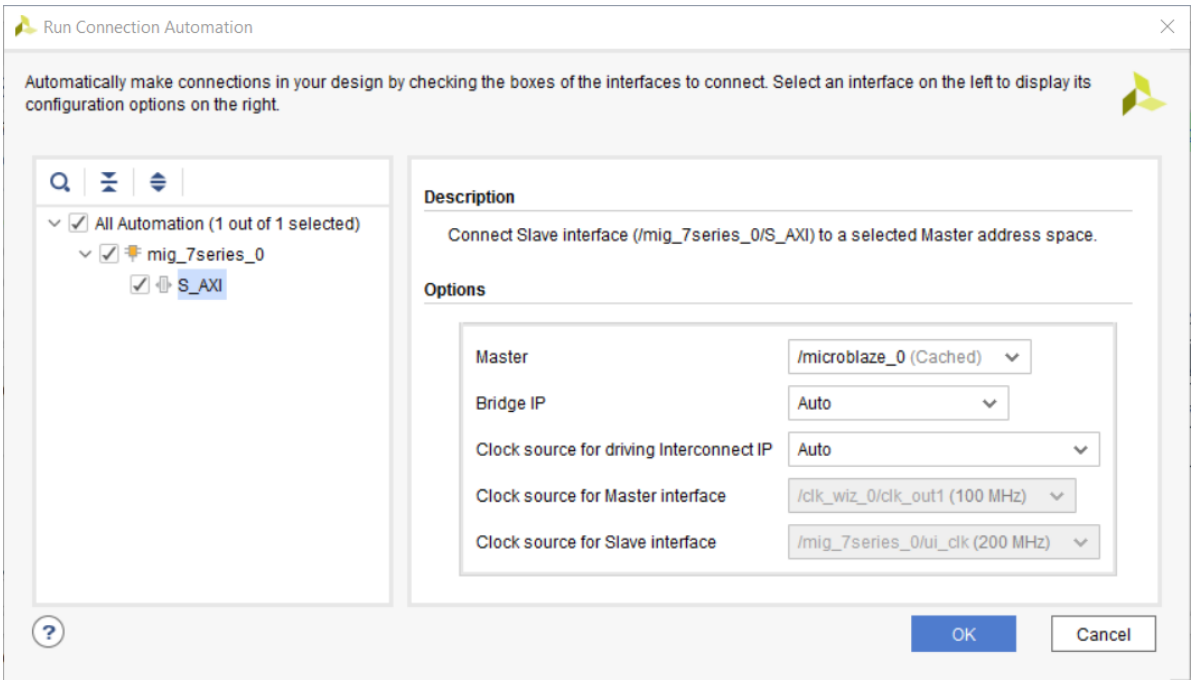
To complete the Memory IP design, an AXI master such as a MicroBlaze V embedded processor, or an external processor is required. The following procedure lists the steps to instantiate a MicroBlaze V processor into the block design.

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

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



4. At the top of the design canvas, click **Run Connection Automation** to connect the Memory IP core to the MicroBlaze V 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. 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 generates the required connection between the Memory IP core and the MicroBlaze processor.

Ensure to 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 V 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 Creating a memory map in the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for more information.

TIP: *The Address Editor window 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 AMD Vitis™ software platform.

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

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, use the AMD 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 using 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 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.

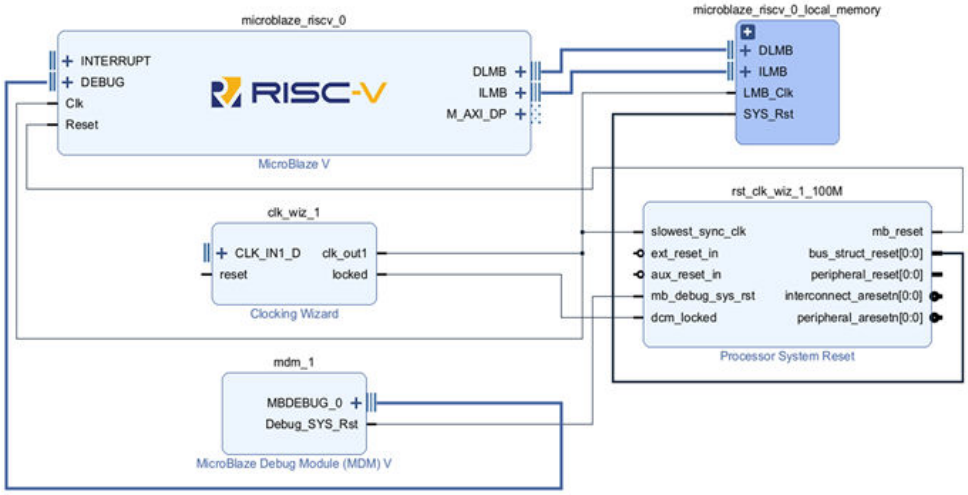
MicroBlaze V Design without a Memory IP Core

For any design that uses a MicroBlaze V 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 V processor in the design, click **Run Block Automation** link. This creates the MicroBlaze V subsystem.
2. In the Run Block Automation dialog box, select the **New Clocking Wizard** option to instantiate the Clocking Wizard IP, and click **OK**.

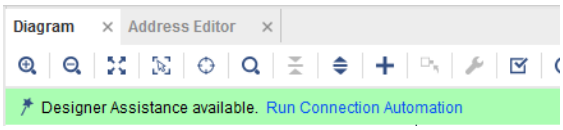
   Running Block Automation also instantiates and connects the Proc Sys Reset IP to the various blocks in the design.

   The IP integrator canvas looks like the following figure.

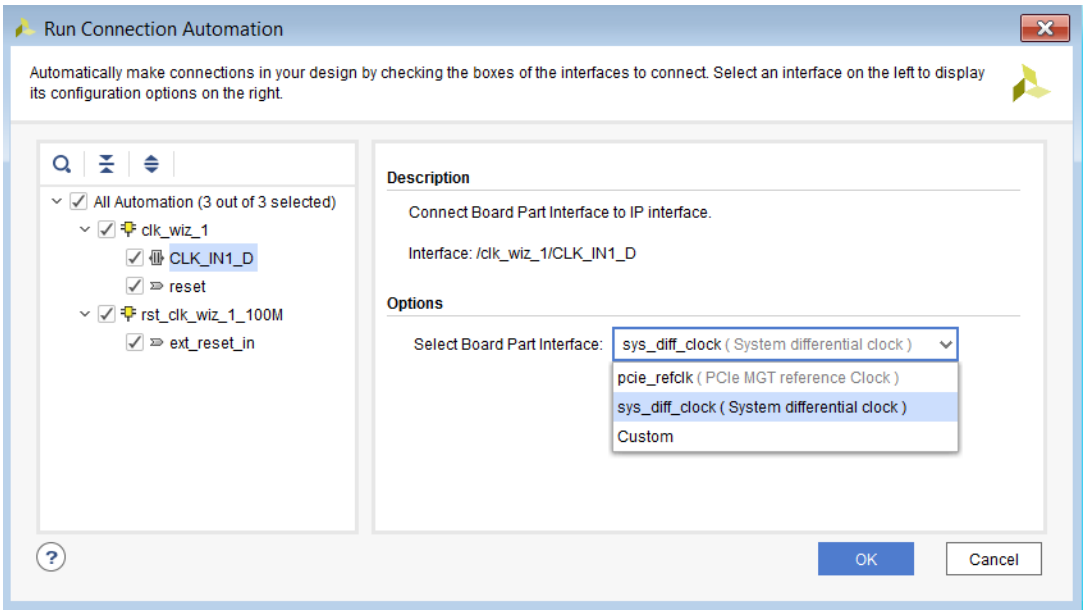


3. Click **Run Connection Automation** and select **/clk_wiz_1/CLK_IN1_D** to connect the on-board clock to the input of the Clocking Wizard IP, according to the board definition.

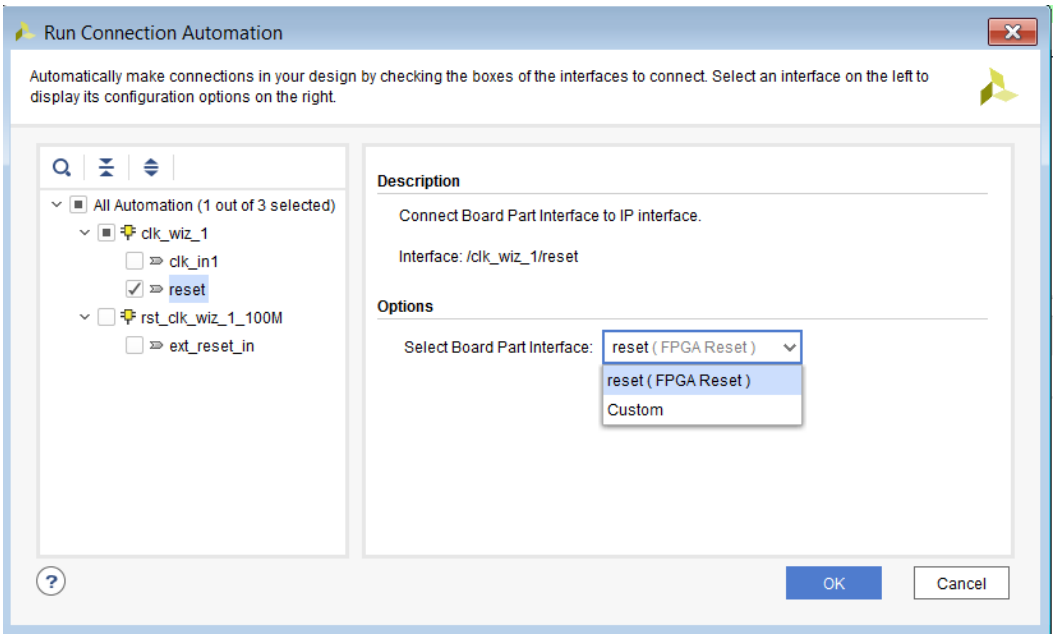
   ***Note*:** You can customize the Clocking Wizard to generate the various clocks required by the design.



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, click **OK**.



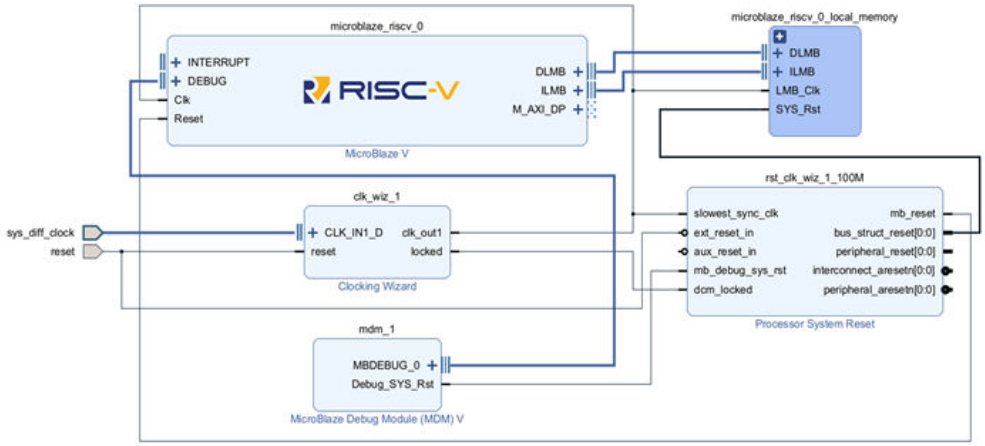
5. For the reset pin of the Clocking Wizard, select the dedicated reset interface on the target board or a Custom reset input source.



***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.*

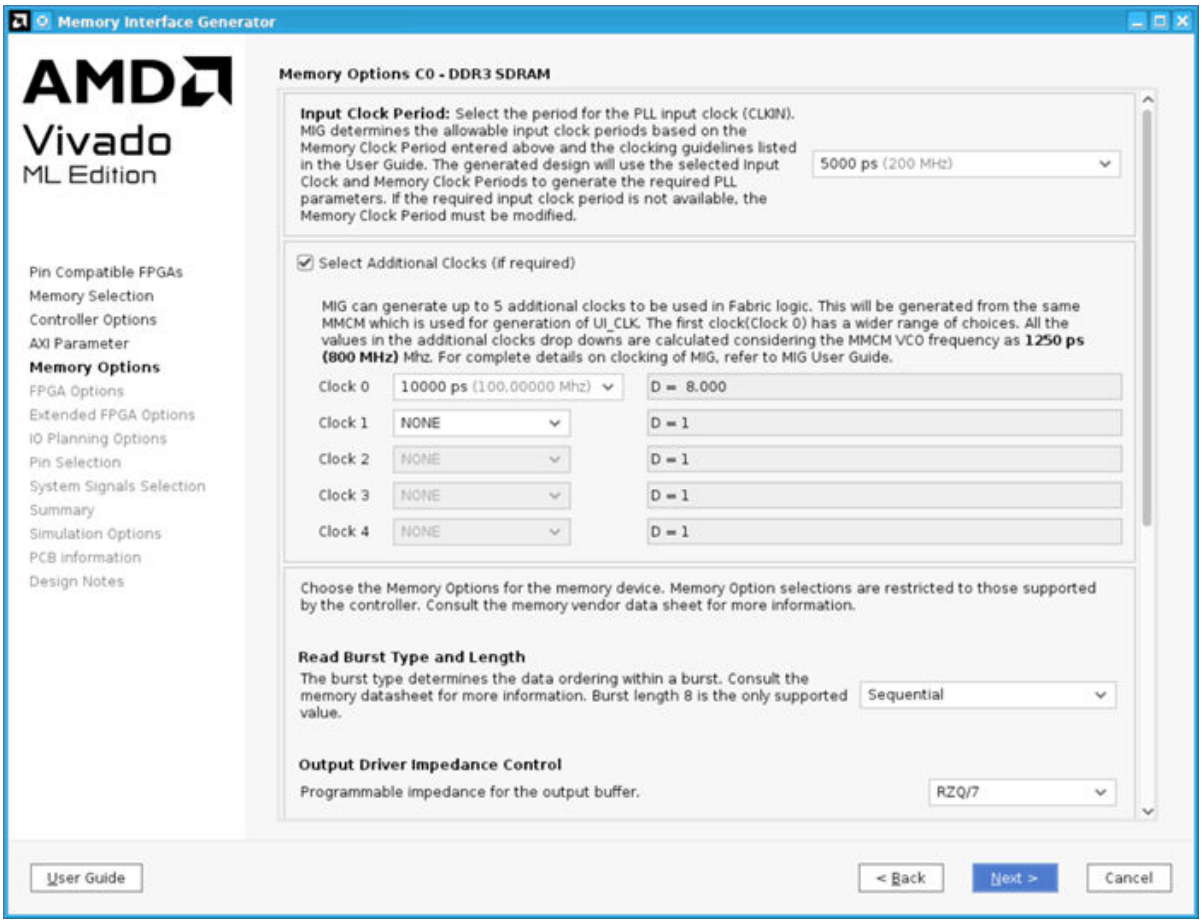
MicroBlaze V Design with a Memory IP Core

RECOMMENDED: *As mentioned in the introduction, the Memory IP is a clock source, and AMD 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 run Block Automation. This connects the on-board clock to the Memory IP core.

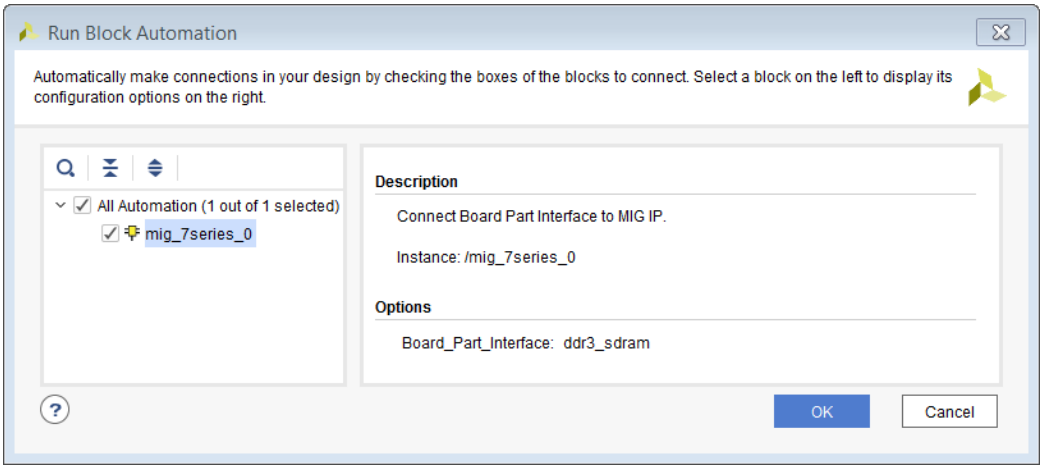
   You can customize Memory IP to generate additional clocks, as shown in the following figure.



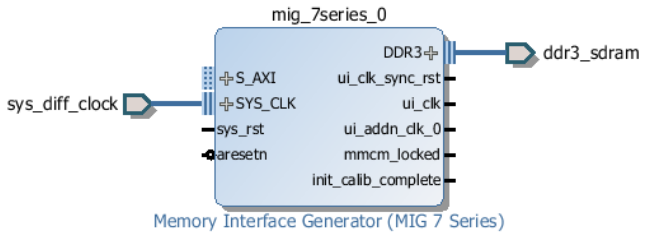
2. After configuring the MIG to generate additional clocks, click the **Run Connection Automation** link at the top of the banner.

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

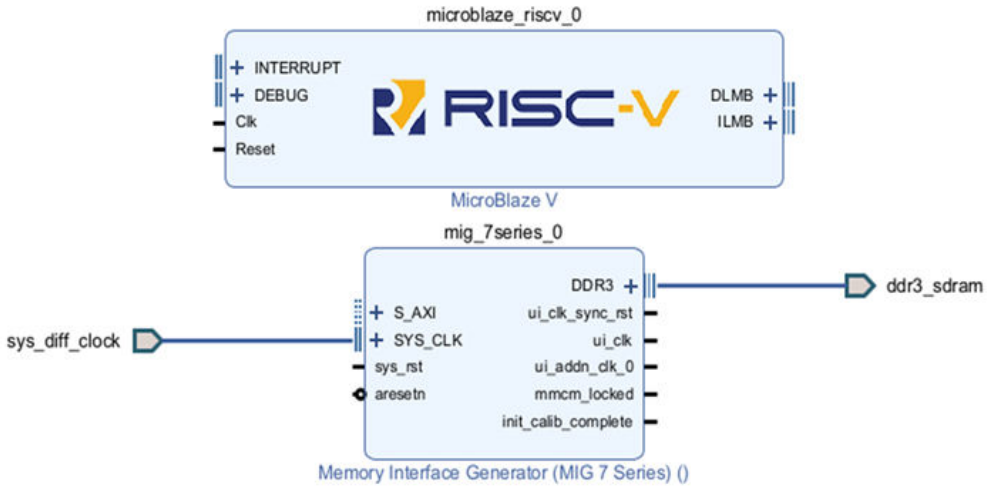
3. Click **OK**.



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

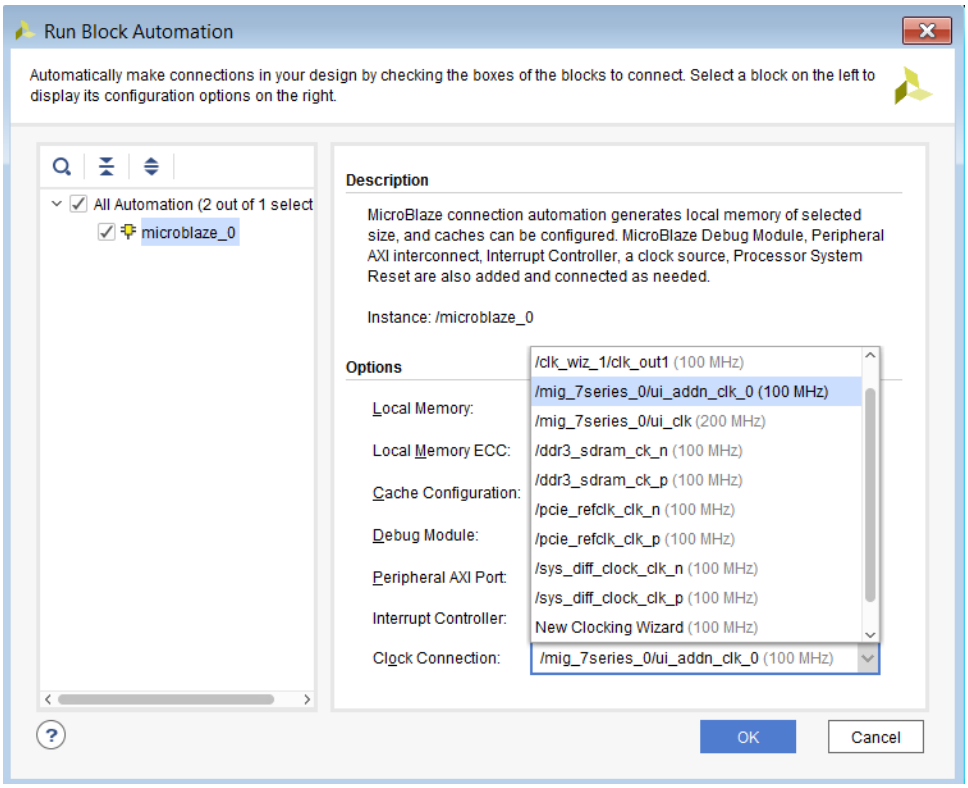


4. Add the MicroBlaze V processor to the design and run Block Automation, as shown in the following figure.

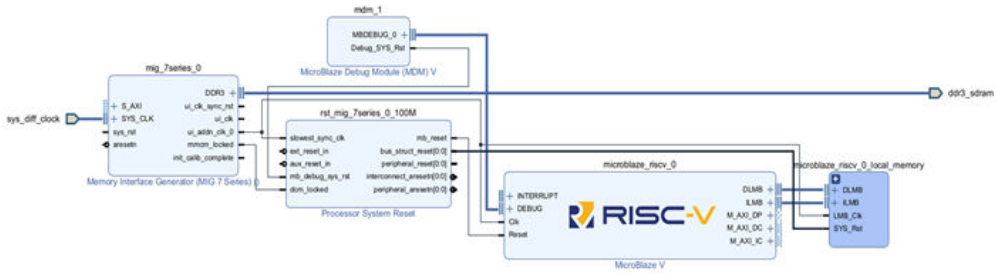


5. In the **Clock Connection** field of the **Run Block Automation** dialog box, select the Memory IP `ui_clk` (**/mig_7series_0/ui_clk** or **mig_7series/u_addn_clk_0**) as the clock source for the MicroBlaze V processor, as shown in the following figure, and click **OK**.

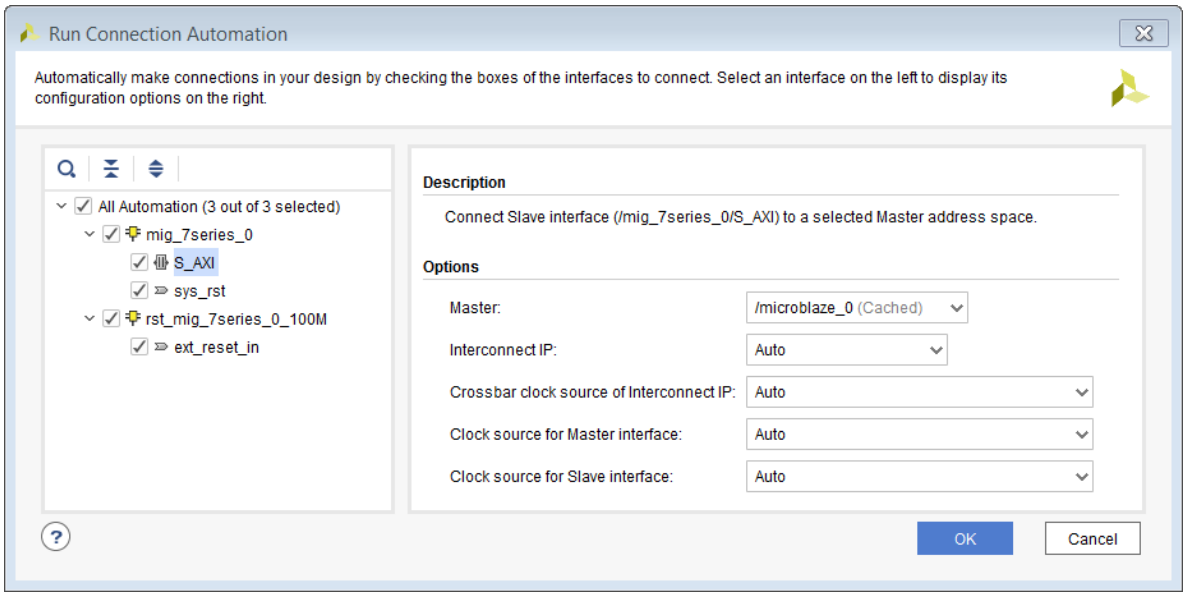
TIP: *The **mig_7series_0/ui_addn_clk_0** is selected by default.*



This creates a MicroBlaze V 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 V.
   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**.

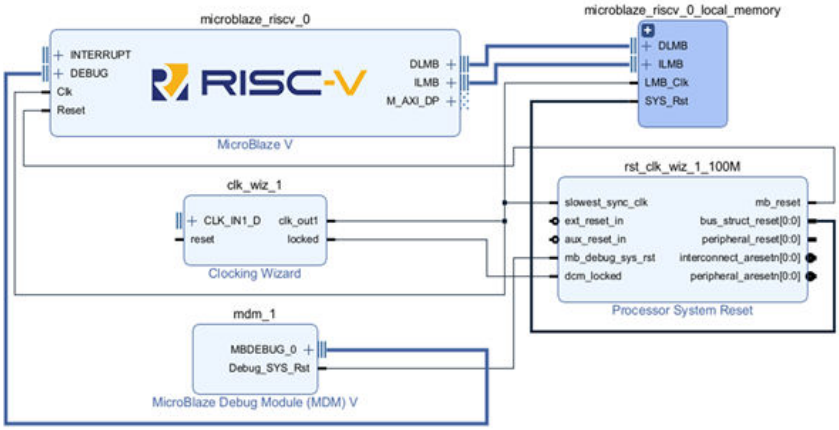


d. Connect the on-board reset to the `sys_rst` input of the Memory IP.

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

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav, do the following:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **AMDDesignTools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD 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, do the following:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

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

References

1. *Triple Modular Redundancy (TMR) LogiCORE IP Product Guide* (PG268)
2. *MicroBlaze Debug Module V (MDM V) LogiCORE IP Product Guide* (PG428)
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 V Processor Reference Guide* (UG1629)
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)
19. *Versal Adaptive SoC PCB Design User Guide* (UG863)
20. *Vitis Embedded Software Development Flow Documentation* (UG1400)

Training Resources

1. Designing FPGAs Using the Vivado Design Suite 1 Training Course

2. Embedded Systems Design Training Course
3. Vivado Design Suite QuickTake Video Tutorials

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
12/03/2025 Version 2025.2	
Entire document	Updated with description of the following features: • Supervisor mode • Cross-trigger
05/29/2025 Version 2025.1	
Entire document	Updated with new features: • External program trace

Please Read: Important Legal Notices

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