

Vivado Design Suite User Guide:

Embedded Processor Hardware Design

UG898 (v2013.1) March 20, 2013





Notice of Disclaimer

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

©Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, Vivado and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Changes
03/20/13	2013.1	New release for Vivado Design Suite version 2013.1.

Table of Contents

Introduction.....	4
Overview	4
Hardware and Software Tool Flow Overview.....	5
Using a Zynq-7000 Processor in an Embedded Design.....	7
Introduction.....	7
Designing for Zynq-7000 AP SoC Devices in the Vivado IDE.....	7
Overview of the Zynq Block Diagram and Configuration Window	11
Using the Programmable Logic (PL).....	25
Vivado Pin Planner View of PS I/O.....	37
Vivado IDE Generated Embedded Files.....	37
Using the Software Development Kit (SDK)	38
Using a MicroBlaze Processor in an Embedded Design.....	40
Introduction to MicroBlaze Processor Design.....	40
Creating an IP Integrator Design with the MicroBlaze Processor.....	41
MicroBlaze Configuration Window.....	44
Custom Logic	61
Embedded IP Catalog	62
Completing Connections.....	63

Introduction

Overview

This chapter provides an introduction to using the Xilinx® Vivado™ Design Suite flow for programming an embedded design using the Zynq™-7000 All Programmable SoC device or the Microblaze™ processor.

Note: *This document contains information about the new Vivado IP integrator environment, which is a licensed early access feature in the 2013.1 release. Please contact your field applications engineer to obtain a license.*

Embedded systems are complex. Hardware and software portions of an embedded design are projects in themselves. Merging the two design components so that they function as one system creates additional challenges. Add an FPGA design project to the mix, and the situation has the potential to become very complicated.

To simplify the design process, Xilinx offers several sets of tools. It is a good idea to get to know the basic tool names, project file names, and acronyms for these tools, which are located:

http://www.xilinx.com/support/documentation/sw_manuals/glossary

The Vivado Integrated Design Environment (IDE) includes the IP integrator tool, which lets you *stitch* together a processor-based design relatively quickly and easily. This tool, in combination with the Xilinx Software Development Kit (SDK), provides an integrated environment to design and debug microprocessor-based systems and embedded software applications.

Hardware and Software Tool Flow Overview

Based on the processor, the Vivado tools provide specific flows for programming. The Vivado Integrated Design Environment (IDE) uses the IP integrator with graphic connectivity screens to rapidly specify the device, select peripherals, and configure hardware settings.

Zynq-7000 uses the Vivado IP integrator to capture hardware platform information in XML format applications for Zynq-7000 AP SoC, along with other data files, which are then used by software design tools to create and configure Board Support Package (BSP) libraries, infer compiler options, program the PL, define JTAG settings, and automate other operations that require information about the hardware. The Zynq-7000 AP SoC solution reduces the complexity of an embedded design by offering an ARM Cortex A9 dual core as Hard IP and programmable logic along with it on a single SoC. It is the first of its kind in the market and has tremendous potential as a complete system.

You can program Microblaze devices from either the ISE Design Suite Embedded Development Kit (EDK) Xilinx Platform Studio (XPS) or the Vivado IP integrator.

Xilinx provides design tools for developing and debugging software applications for Zynq-7000 AP SoC and Microblaze processor devices, which include:

- Software IDE
- GNU-based compiler toolchain
- JTAG debugger

These tools let you develop both bare-metal applications that do not require an operating system, and applications for the open source Linux operating system. The Vivado IP integrator captures information about the Processing System (PS) and peripherals, including configuration settings, the register memory map, and the bitstream for Processing Logic (PL) initialization.

Software solutions are also available from third-party sources that support Cortex-A9 processors, including but not limited to:

- Software IDEs
- Compiler toolchains
- Debug and trace tools
- Embedded OS and software libraries
- Simulators
- Models and virtual prototyping tools

Third party tool solutions vary in the level of integration and direct support for Zynq-7000 AP SoC devices.

See the *Zynq-7000 Software Developers Guide* ([UG821](#)) for more information about the SDK and programming for Zynq devices. The SDK software is a standalone product, and is available for download from www.xilinx.com.

The following figure illustrates the tools flow for embedded hardware.

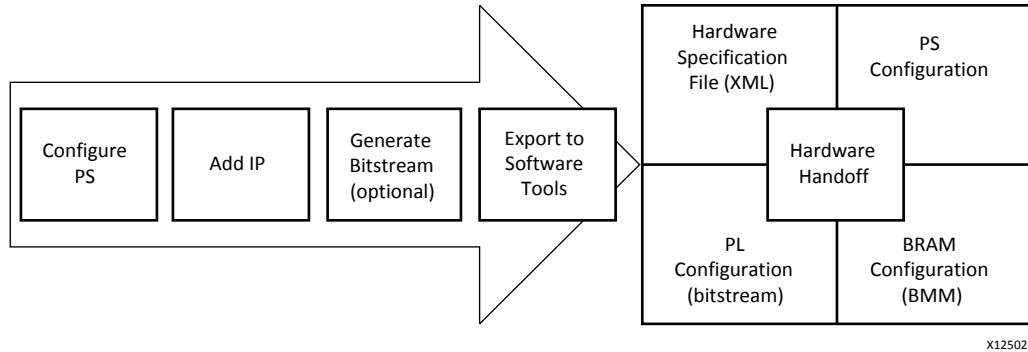


Figure 1: Hardware Design Tool Handoff to Software Tools

To start a Zynq-7000 based design, do the following:

- Create a new Vivado IDE project.
- Create a block diagram in the IP Integrator tool and start instantiating the Zynq Processing System 7 IP along with any other Xilinx IP or your own custom IP.
- Run the design through synthesis and implementation and export the hardware to SDK.
- In SDK, you create your software application, which you can then program into the target board.

Using a Zynq-7000 Processor in an Embedded Design

Introduction

This chapter describes how to use the Xilinx® Vivado™ Design Suite flow for using the Zynq™-7000 All Programmable (AP) SoC device.

The examples target the Xilinx ZC702 Rev 1.0 evaluation board and the tool versions in the 2013.1 Vivado Design Suite release.



IMPORTANT: The Vivado IP integrator is the replacement for Xilinx Platform Studio (XPS) for embedded processor designs, including designs targeting Zynq™ devices and MicroBlaze™ processors. XPS only supports designs targeting MicroBlaze processors, not Zynq devices. Both IP integrator and XPS are available from the Vivado IDE.

Designing for Zynq-7000 AP SoC Devices in the Vivado IDE

Designing for Zynq-7000 AP SoC devices is different using the Vivado IDE than it was using the ISE® Design Suite and Embedded Development Kit (EDK).

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

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

Creating an IP Integrator Design with the Zynq-7000 Processor

When you click the IP integrator **Create Block Design** button,  **Create Block Design** a dialog box opens for you to enter the **Design Name**, as shown in the following figure.

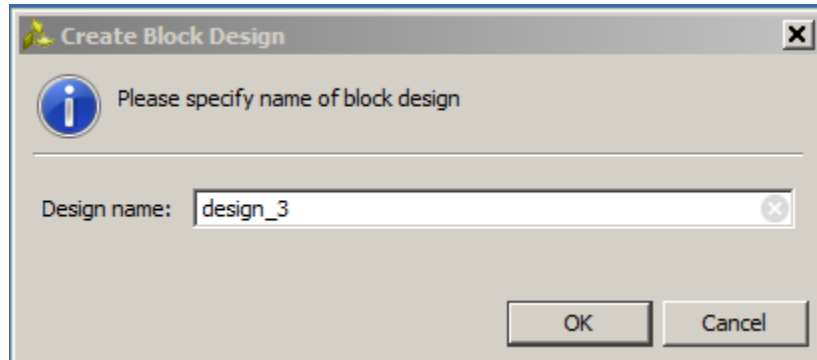


Figure 2: Design Name Dialog Box

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

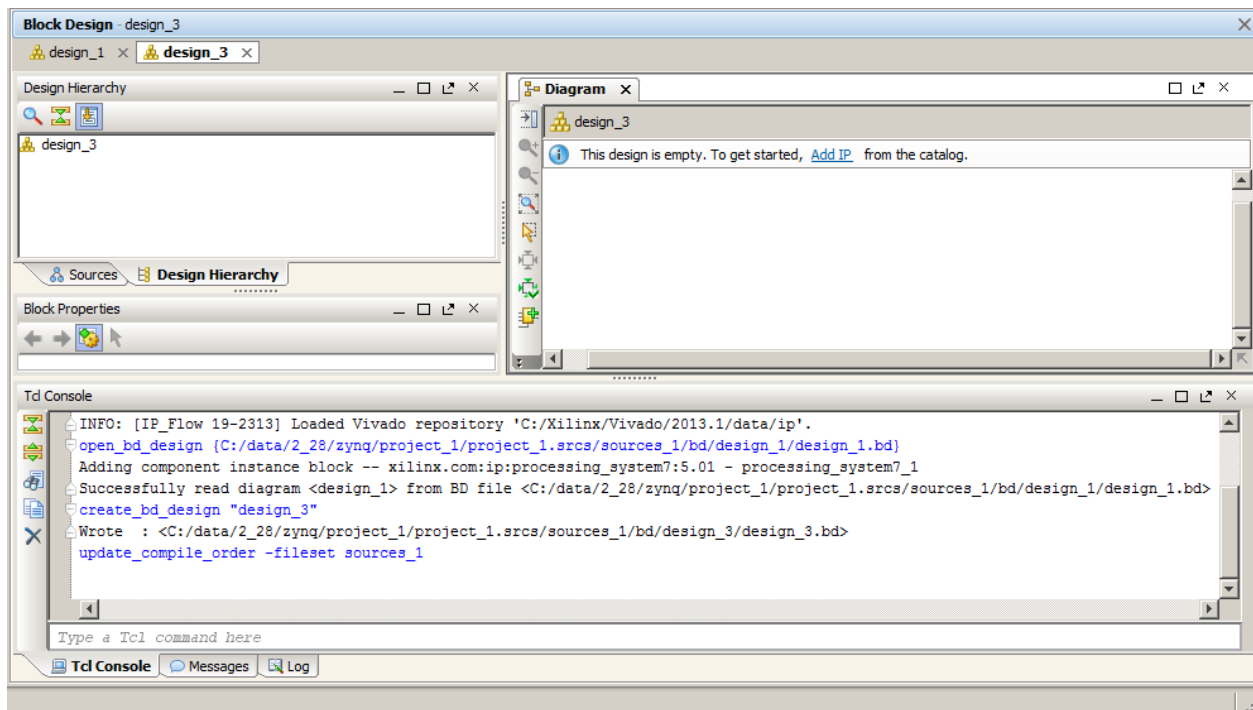


Figure 3: Block Design Window

Within the empty design, there is an option to **Add IP** from the IP Catalog. You can also right-click in the canvas to bring up the selection option to add IP.

Select the **Add IP** option, and a Search box opens where you can search for, and select the **ZYNQ7 Processing System**, shown in the following figure.

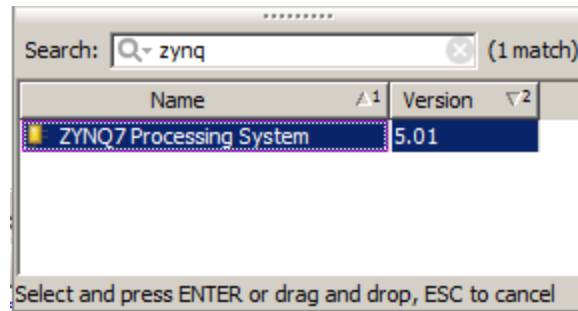


Figure 4: Search IP with ZYNQ7 Processing System

When you select the Zynq 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.

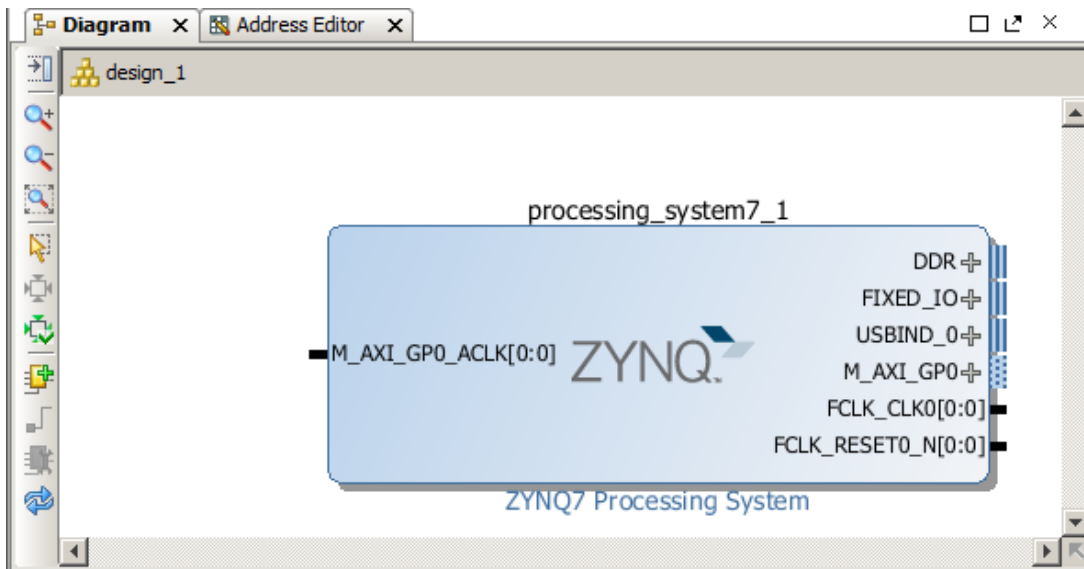


Figure 5: Graphical Display of Default Zynq7 Processing System

The Tcl command to perform this operation is as follows:

```
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.01
processing_system7_1
```

Double-click within the processing system graphic to invoke the **Recustomize IP** process, which displays the Re-customize IP for the ZYNQ7 Processing System dialog box as shown in the following figure. Review the contents of the block diagram. The green colored blocks in the ZYNQ7 Processing System are configurable items. You can click a green block to open the coordinating configuration options.

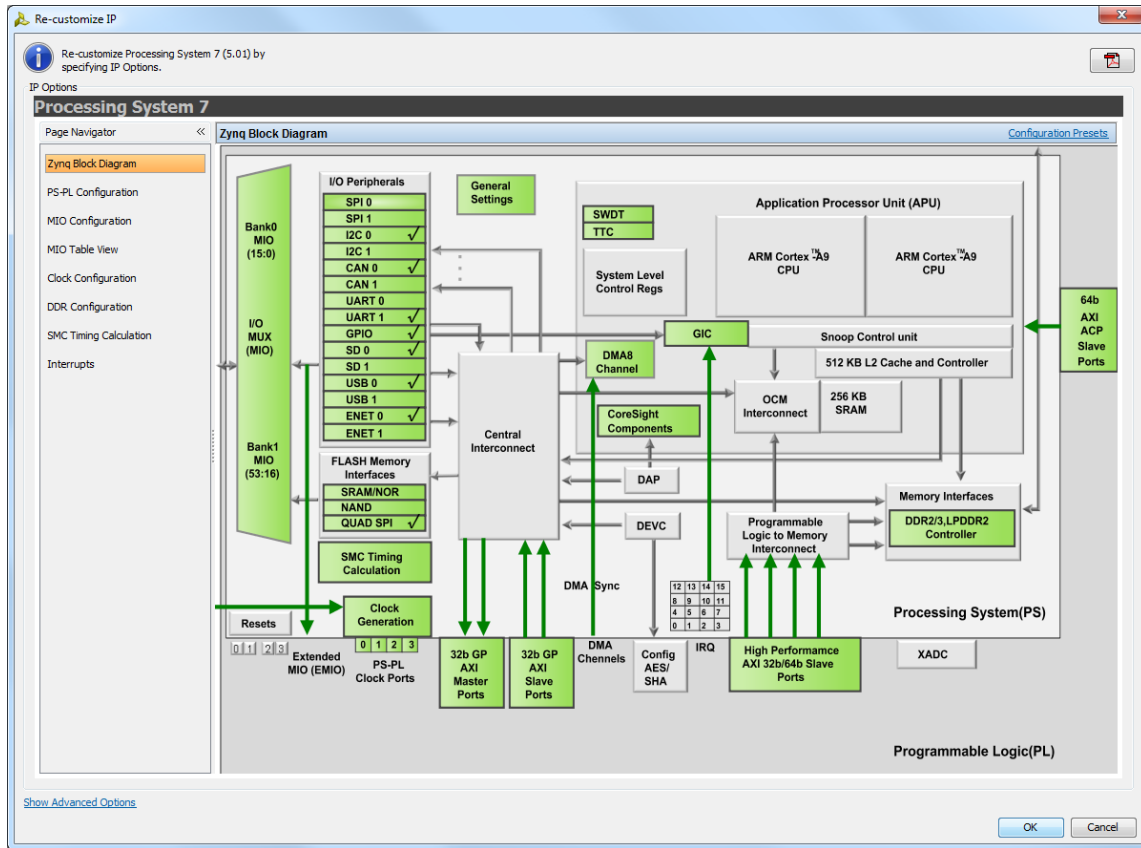


Figure 6: Zynq Block Diagram/Configuration Dialog Box

Alternatively, you can select the options from the Page Navigator on the left, as shown in the following figure.

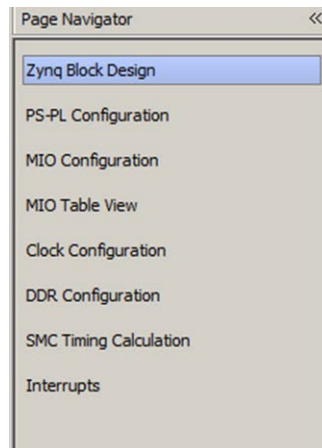


Figure 7: Page Navigator

Overview of the Zynq Block Diagram and Configuration Window

The *Zynq-7000 AP SoC Technical Reference Manual (UG585)* provides the details on the default options available in the Page Navigator. In brief, the following subsections describe the Page Navigator selection options.

Processing System (PS)-Processing Logic (PL) Configuration Options

The **PS-PL Configuration** option tree, displays with the collapsed options as shown in the following figure.

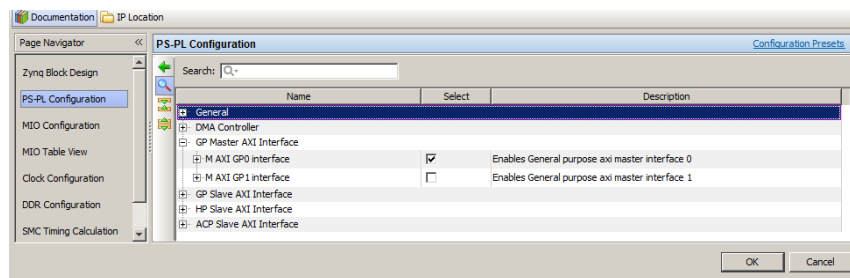


Figure 8: PL-PS Configuration Pane

General Options

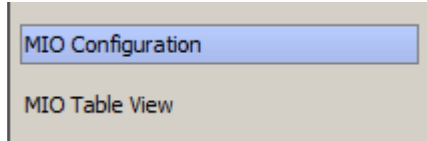
When you expand the **General Options**, the following selections are available.

Name	Select	Description
General		
UART Baud Rate	115200	Configure baud rate to determine UART operating frequency
PL AXI Idle Port	<input type="checkbox"/>	Enables idle AXI signal to the PS used to indicate that there are no outstanding ...
DDR ARB bypass Port	<input type="checkbox"/>	Enables DDR urgent/arb signal used to signal a critical memory starvation situati...
PS-PL Debug interface	<input type="checkbox"/>	Enables PL debug signals to PS and vice-versa
FTM Trace data interface	<input type="checkbox"/>	Enables FTM Trace axi stream interface used to capture data from PL to PS deb...
FTM Trace buffer	<input type="checkbox"/>	Generates a FIFO to hold trace data
FTM data edge detector	<input type="checkbox"/>	Stores trace data in the FIFO when the data changes as marked by edge detector
PS-PL Cross Trigger interface	<input type="checkbox"/>	Enables PL cross trigger signals to PS and vice-versa
power-on reset(POR) 4k timer	<input type="checkbox"/>	Enables power-on reset(POR) 4k timer. By default, 64k timer is used.
Processor event interface	<input type="checkbox"/>	Enables event bus which provides a low-latency and direct mechanism to transf...
Allow Access to High OCM	<input type="checkbox"/>	Allow address mapping to PS internal OCM at High Address
Detailed IOP address space	<input type="checkbox"/>	Provide Individual address spaces for PS internal Peripherals
FTM Trace buffer FIFO size	128	FTM Trace buffer FIFO size
FTM Trace buffer clock delay	12	Number of clock cycles interval for a trace data output from FIFO being present...
Enable Clock Triggers		
FLCK_CLKTRIG0	<input type="checkbox"/>	Enables PL clock trigger signal 0 used to halt the PL clock when counting a progr...
FLCK_CLKTRIG1	<input type="checkbox"/>	Enables PL clock trigger signal 1 used to halt the PL clock when counting a progr...
FLCK_CLKTRIG2	<input type="checkbox"/>	Enables PL clock trigger signal 2 used to halt the PL clock when counting a progr...
FLCK_CLKTRIG3	<input type="checkbox"/>	Enables PL clock trigger signal 3 used to halt the PL clock when counting a progr...
Enable Clock Resets		
FCLK_RESET0_N	<input checked="" type="checkbox"/>	Enables general purpose reset signal 0 for PL logic
FCLK_RESET1_N	<input type="checkbox"/>	Enables general purpose reset signal 1 for PL logic
FCLK_RESET2_N	<input type="checkbox"/>	Enables general purpose reset signal 2 for PL logic
FCLK_RESET3_N	<input type="checkbox"/>	Enables general purpose reset signal 3 for PL logic
DMA Controller		

Figure 9: General Options (First Tier)

MIO and EMIO Configuration

From the Page Navigator, there are two page options in which to view and configure MIO, I/O Peripherals, APUs, and Programmable Logic Test and Debug.



The Zynq™ PS has over 20 peripherals available for selection. You can either directly route these peripherals to the dedicated Multiplexed I/Os (MIO) on the device or through the Extended Multiplexed I/Os (EMIOs) routing to the fabric.

The configuration interface also lets you select I/O standards and slew settings for the MIO. The I/O peripheral block presents with a checkmark when you enable a peripheral. The block diagram depicts the status of enabled and disabled peripherals.

Chapter 2, “Signals, Interfaces, and Pins” of the *Zynq-7000 AP SoC Technical Reference Manual* ([UG585](#)) describes the MIOs and EMIOs for the 7z010 CLG225 device. The following subsection on Pin Limitations is a brief description.

Pin Limitations

The 32 MIO pins available in the 7z010 CLG225 device restrict the functionality of the PS as follows:

Either one USB or one Ethernet controller using MIO

No boot from SDIO

No NOR/SRAM interfacing

Width of NAND Flash limited to 8 bits

Bank Settings

After you select peripherals, the individual I/O signals for the peripheral show in the respective MIO locations. Use this section primarily for selecting I/O standards for the various peripherals. The PS MIO I/O buffers split into two voltage domains: within each domain, each MIO is independently programmable.

Two I/O voltage banks:

- Bank 0 voltage bank consists of pins 0:15
- Bank 1 voltage bank consists of pins 16:53

Each MIO pin is individually programmed for voltage signaling:

- 1.8 and 2.5/3.3 volts
- CMOS single-ended or HSTL differential receiver mode

The voltage should be the same for the entire bank, but they can have different IO standards.

When you configure MIOs in the MIO Configuration dialog box on the Zynq tab, you can view a read-only image of the peripheral and respective MIO selections. The left side of the window lists the available peripherals. A checkmark on the peripheral indicates when a peripheral is selected or deselected.

Flash Memory Interfaces

Select within the configuration wizard, one of the following:

SRAM/NOR Controller:

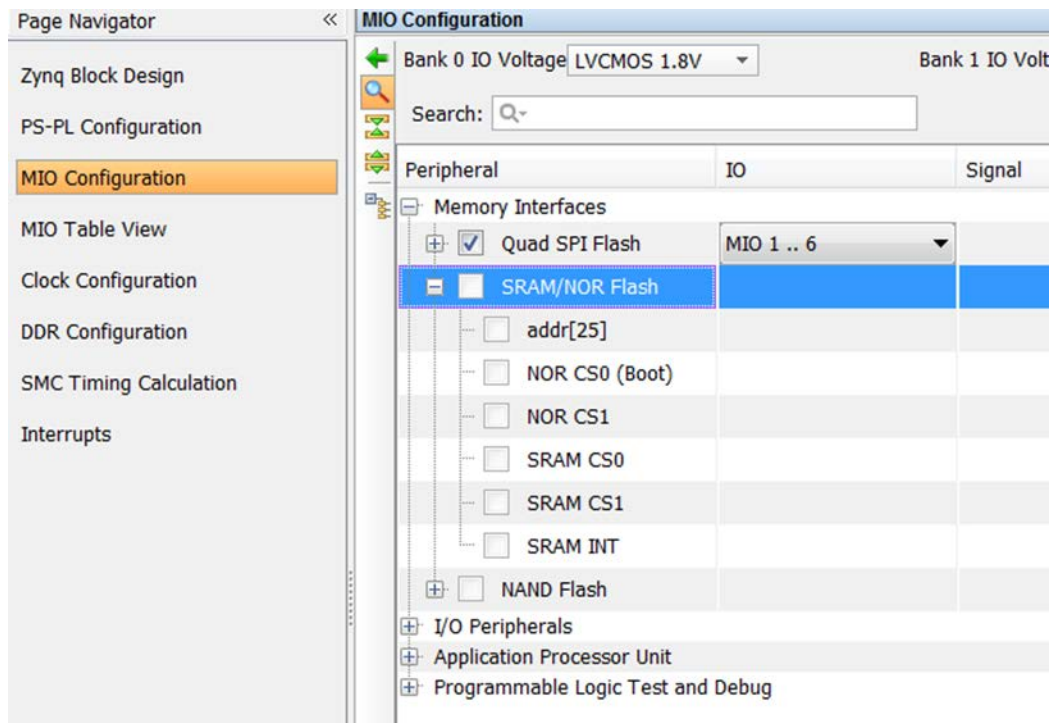


Figure 10: SRAM/NOR Flash Configuration Options

The SRAM/NOR controller has the following features:

- 8-bit data bus width
- One chip select with up to 26 address signals (64 MB)
- Two chip selects with up to 25 address signals (32 MB + 32 MB)
- 16-word read and 16-word write data FIFOs
- 8-word command FIFO
- Programmable I/O cycle timing on a per chip select basis
- Asynchronous memory operating mode

NAND Controller:

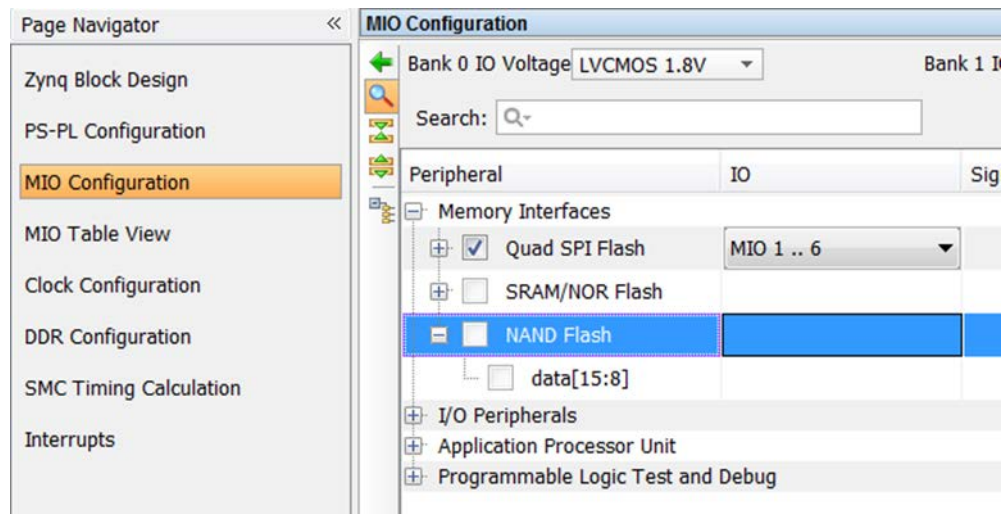


Figure 11: NAND Controller Options

The NAND controller has the following features:

- 8/16-bit I/O width with one chip select signal
- ONFI specification 1.0
- 16-word read and 16-word write data FIFOs
- 8-word command FIFO
- Programmable I/O cycle timing
- ECC assist
- Asynchronous memory operating mode

Quad-SPI Controller:

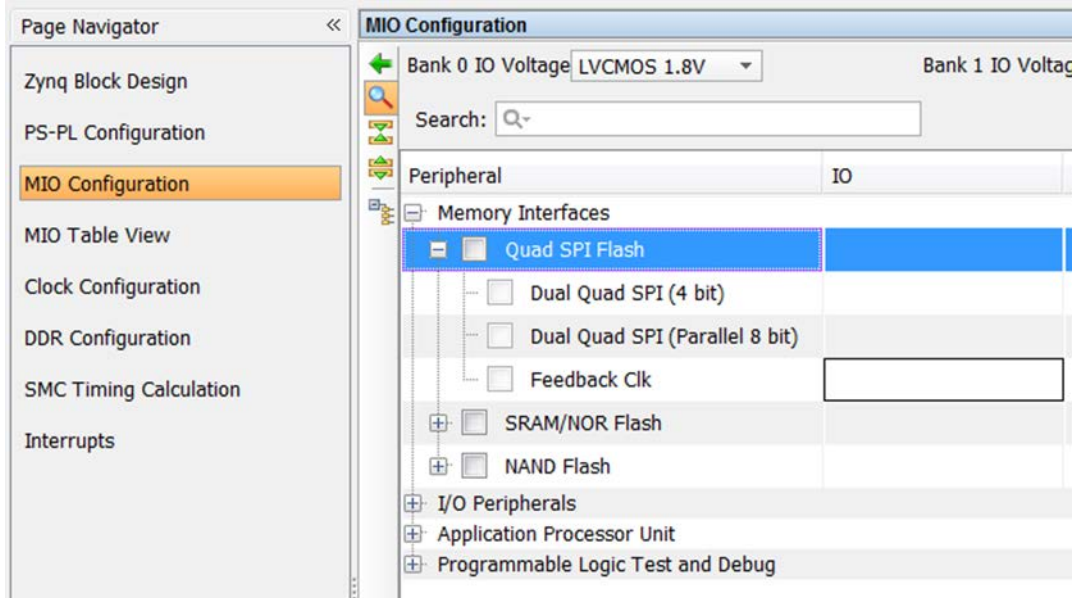


Figure 12: Quad SPI Controller Options

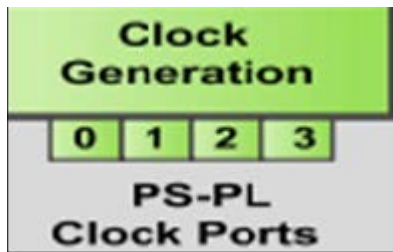
Key features of the linear Quad-SPI controller are:

- Single or dual 1x and 2x read support
- 32-bit APB 3.0 interface for I/O mode that allows full device operations including program, read, and configuration
- 32-bit AXI linear address mapping interface for read operations
- Single chip select line support
- Supports write protection signal
- Four-bit bidirectional I/O signals
- Read speed of x1, x2, and x4
- Write speed of x1 and x4
- Maximum Quad-SPI clock at master mode is 100 MHz
- 252-byte entry FIFO depth to improve Quad-SPI read efficiency
- Supports Quad-SPI device up to 128 Mb density
- Supports dual Quad-SPI with two Quad-SPI devices in parallel

Additionally, the linear address mapping mode features include:

- Supports regular read-only memory access through the AXI interface
- Up to two SPI flash memories
- Up to 16 MB addressing space for one memory and 32 MB for two memories
- AXI read acceptance capability of four
- Both AXI incrementing and wrapping-address burst read
- Automatically converts normal memory read operation to SPI protocol, and vice versa
- Serial, Dual, and Quad-SPI modes

Clock Configuration



You can configure clocks in the Zynq-7000 device using one of the following methods:

- From the Page Navigator, click **Clock Configuration**.
- In the Zynq block diagram, click the **Clock Generation** block.

The following figure shows the Clock Configuration page.

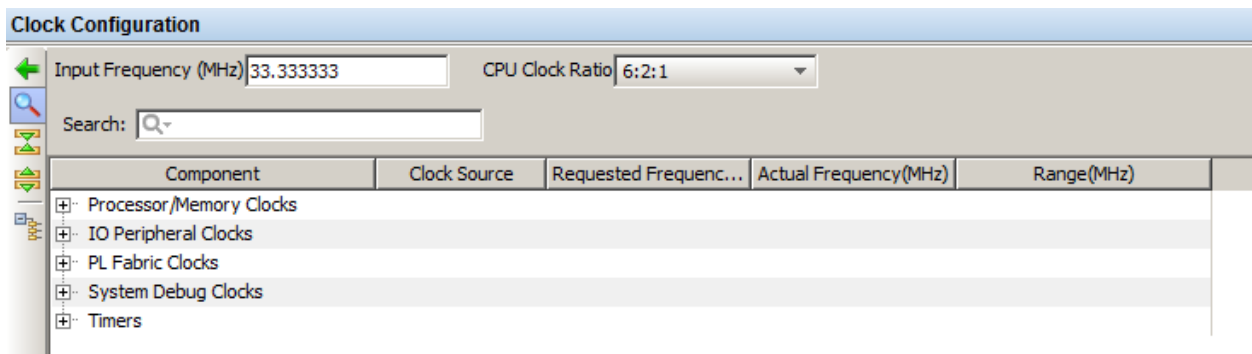


Figure 13: Clock Configuration Page

The following figure shows the Processor/Memory Clock configuration page.

Clock Configuration				
Input Frequency (MHz) <input type="text" value="33.333333"/>		CPU Clock Ratio <input type="text" value="6:2:1"/>		
Search: <input type="text" value="Q-"/>				
Component	Clock Source	Requested Frequenc...	Actual Frequency(MHz)	Range(MHz)
[-] Processor/Memory Clocks				
CPU	ARM PLL	666.666666	666.666687	50.0 : 667.0
DDR	DDR PLL	533.333333	533.333374	200.0 : 534.0
[-] IO Peripheral Clocks				
SMC	IO PLL	100	10.000000	10.0 : 100.0
QSPI	ARM PLL	200	190.476196	10.0 : 200.0
ENET0	IO PLL	100 Mbps	25.000000	
ENET1	IO PLL	1000 Mbps	10.000000	
SDIO	IO PLL	50	50.000000	10.0 : 125.0
SPI	IO PLL	166.666666	10.000000	10.0 : 200.0
[-] CAN				
CAN CLK	IO PLL	23.8095	23.809523	0.1 : 100.0
CAN0 MIOCLK	External	100		0.1 : 100.0
CAN1 MIOCLK	External	100		0.1 : 100.0
[-] PL Fabric Clocks				
<input checked="" type="checkbox"/> FCLK_CLK0	IO PLL	50	50.000000	0.1 : 250.0
<input type="checkbox"/> FCLK_CLK1	IO PLL	50	50.000000	0.1 : 250.0
<input type="checkbox"/> FCLK_CLK2	IO PLL	50	50.000000	0.1 : 250.0
<input type="checkbox"/> FCLK_CLK3	IO PLL	50	50.000000	0.1 : 250.0
[-] System Debug Clocks				
TPIU	IO PLL	200	10.000000	10.0 : 300.0
[-] Timers				
WDT	CPU_1X	133.333333	111.111115	0.1 : 200.0
[-] TTC0				
TTC0 CLKIN0	CPU_1X	133.333333	111.111115	0.1 : 200.0
TTC0 CLKIN1	CPU_1X	133.333333	111.111115	0.1 : 200.0
TTC0 CLKIN2	CPU_1X	133.333333	111.111115	0.1 : 200.0
[-] TTC1				
TTC1 CLKIN0	CPU_1X	133.333333	111.111115	0.1 : 200.0
TTC1 CLKIN1	CPU_1X	133.333333	111.111115	0.1 : 200.0
TTC1 CLKIN2	CPU_1X	133.333333	111.111115	0.1 : 200.0

Figure 14: Processor and Memory Clock Configurations Page

The *Zynq-7000 AP SoC Technical Reference Manual* ([UG585](#)) describes the clocking of the processing system (PS) in detail. The Zynq clocking dialog box lets you set the peripheral clocks. The peripherals on the PS typically allow clock source selection to be either from internal PLLs or from an external clock source. Most of the clocks can select the PLL to generate the clock.

Because the same PLL generates multiple frequencies, it might not be possible to get the exact frequency as entered in the Requested Frequency (MHz) column. The achievable frequency is in the Actual Frequency (MHz) column.

Note: The frequency for a specific peripheral depends upon many factors, such as input frequency, frequency for other peripherals being driven from the same PLL, and restrictions from the architecture. Details of the M & D values chosen by the tool are available in the log file.

DDR Configuration

DDR2/3,LPDDR2 Controller

You can configure DDR with one of two methods:

- From the Page Navigator, select the DDR2/3/LPDDR2
- In the Zynq block diagram, click the DDR2/3, LPDDR2 Controller block.

The DDR memory controller supports DDR2, DDR3, DDR3L, and LPDDR2 devices and consists of three major blocks: an AXI memory port interface - DDR interface (DDRI), a core controller with transaction scheduler (DDRC), and a controller with digital PHY (DDRP).

The DDRI block interfaces with four 64-bit synchronous AXI interfaces to serve multiple AXI masters simultaneously. Each AXI interface has a dedicated transaction FIFO. The DDRC contains two 32-entry content addressable memories (CAMs) to perform DDR data service scheduling to maximize DDR memory efficiency. It also contains a "fly-by" channel for a low latency channel to allow access to DDR memory without going through the CAM.

The PHY processes read/write requests from the controller and translates them into specific signals within the timing constraints of the target DDR memory. The PHY uses signals from the controller to produce internal signals that connect to the pins using the digital PHYs. The DDR pins connect directly to the DDR device(s) using the PCB signal traces.

The system accesses the DDR using the DDRI through its four 64-bit AXI memory ports:

- One AXI port is dedicated to the L2-cache for the CPUs and ACP
- Two ports are dedicated to the AXI_HP interfaces
- The other masters on the AXI interconnect share the fourth port

The DDRI arbitrates the requests from the eight ports (four reads and four writes). The arbiter selects a request and passes it to the DDR controller and transaction scheduler (DDRC).

The arbitration is based on a combination of how long the request has been waiting, the urgency of the request, and if the request is within the same page as the previous request.

The DDRC receives requests from the DDRI through a single interface for both read and write flows. Read requests include a tag field that the DDR returns with the data. The DDR controller PHY (DDRP) drives the DDR transactions.

The following figure shows the DDR Controller configuration page.

Note: 8 bit interfaces are not supported; however, 8-bit parts can be used to create 16/32-bit interfaces.

DDR Configuration

Enable DDR

Search:

Name	Select	Description
DDR Controller Configuration		
Memory Type	DDR 3	Select the type of memory interface/types. Please refer to the TR...
Memory Part	MT41J256M8 HX-15E	Select the needed memory part. Select the the "Custom" in memo...
Effective DRAM Bus Width	32 Bit	Select the data Width. Please refer to the TRM for a detailed list o...
ECC	Disabled	Zynq IP supports ECC for 16 Bit only. To be able to Enable/Disable...
Burst Length	8	Select the burst Length. It refers to the amount of data read/writt...
DDR	533.333333	Chose the clock period for the desired frequency. The allowed fre...
Internal Vref	<input checked="" type="checkbox"/>	Enable the Internal Vref. It can be used to allow the use of the Vr...
Operating Temperature (C)	Normal (0-85)	Select the operating temperature
Memory Part Configuration		
DRAM IC Bus Width	8 Bits	Provide the width of the DRAM chip
DRAM Device Capacity	2048 MBits	Provide the capacity for DDR Memory devices
Speed Bin	DDR3_1066F	Provide the Speed Bin
Bank Address Count (Bits)	3	Defines the bank to which an active an ACTIVE, READ, WRITE, or ...
Row Address Count (Bits)	15	Provide the Row address for ACTIVE commands
Col Address Count (Bits)	10	Provide the Row address for READ/WRITE commands
CAS Latency (cycles)	7	Select the Column Access Strobe (CAS) Latency. It refers to the a...
CAS Write Latency (cycles)	6	Select the CAS Write Latency
RAS to CAS Delay (cycles)	7	Provide the row address to column address delay time. tRCD is th...
Precharge Time	7	Precharge Time (tRP) is the number of clock cycles needed o term...
tRC (ns)	49.5	Provide the Row cycle time tRC (ns)
tRASmin (ns)	36.0	tRASmin (ns) is the minimum number of clock cycles required bet...
tFAW (ns)	30.0	It restricts the number of activates that can be done within a certa...
Training/Board Details		
User Input		
+ DRAM Training		
+ DQS to Clock Delay (ns)		
+ Board Delay (ns)		
Additive Latency (ns)	0	Provide the Additive Latency (ns). Increases the efficiency of the c...

Figure 15: DDR Controller Configurations Page

GIC - Interrupt Controller

You can configure the Generic Interrupt Controller (GIC) in one of two methods:

In the Page Navigator, click **Interrupts**.

In the Zynq block diagram, click the GIC block.



The following figure shows the Interrupt Port Configuration page.

Interrupt Port	ID	Description
<input type="checkbox"/> Fabric Interrupts		Enable PL Interrupts to PS and vice versa
<input type="checkbox"/> PL-PS Interrupt Ports		
<input type="checkbox"/> IRQ_F2P[15:0]	[91:84], [68:...	Enables 16-bit shared interrupt port from the PL. MSB is assigned the highest In...
<input type="checkbox"/> Core0_nFIQ	28	Enables Fast private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core1_nFIQ	28	Enables Fast private interrupt signal for CPU1 from the PL
<input type="checkbox"/> Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL
<input type="checkbox"/> PS-PL Interrupt Ports		
<input type="checkbox"/> IRQ_P2F_DMAC_ABORT		Enables shared interrupt abort signal from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC0		Enables shared interrupt signal 0 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC1		Enables shared interrupt signal 1 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC2		Enables shared interrupt signal 2 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC3		Enables shared interrupt signal 3 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC4		Enables shared interrupt signal 4 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC5		Enables shared interrupt signal 5 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC6		Enables shared interrupt signal 6 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_DMAC7		Enables shared interrupt signal 7 from DMAC to the PL
<input type="checkbox"/> IRQ_P2F_SMC		Enables shared interrupt signal from SMC to the PL
<input type="checkbox"/> IRQ_P2F_QSPI		Enables shared interrupt signal from QSPI to the PL
<input type="checkbox"/> IRQ_P2F_CTI		Enables shared interrupt signal from CTI to the PL
<input type="checkbox"/> IRQ_P2F_GPIO		Enables shared interrupt signal from GPIO to the PL
<input type="checkbox"/> IRQ_P2F_USB0		Enables shared interrupt signal from USB 0 to the PL
<input type="checkbox"/> IRQ_P2F_ENET0, IRQ_P2F_ENE...		Enables shared interrupt and wake signals from ETHERNET 0 to the PL
<input type="checkbox"/> IRQ_P2F_SDIO0		Enables shared interrupt signal from SDIO 0 to the PL
<input type="checkbox"/> IRQ_P2F_I2C0		Enables shared interrupt signal from I2C 0 to the PL
<input type="checkbox"/> IRQ_P2F_SPI0		Enables shared interrupt signal from SPI0 to the PL
<input type="checkbox"/> IRQ_P2F_UART0		Enables shared interrupt signal from UART 0 to the PL
<input type="checkbox"/> IRQ_P2F_CAN0		Enables shared interrupt signal from CAN 0 to the PL
<input type="checkbox"/> IRQ_P2F_USB1		Enables shared interrupt signal from USB 1 to the PL
<input type="checkbox"/> IRQ_P2F_ENET1, IRQ_P2F_ENE...		Enables shared interrupt and wake signals from ETHERNET 1 to the PL
<input type="checkbox"/> IRQ_P2F_SDIO1		Enables shared interrupt signal from SDIO 1 to the PL
<input type="checkbox"/> IRQ_P2F_I2C1		Enables shared interrupt signal from I2C 1 to the PL
<input type="checkbox"/> IRQ_P2F_SPI1		Enables shared interrupt signal from SPI 1 to the PL
<input type="checkbox"/> IRQ_P2F_UART1		Enables shared interrupt signal from UART 1 to the PL
<input type="checkbox"/> IRQ_P2F_CAN1		Enables shared interrupt signal from CAN 1 to the PL

Figure 16: GIC Interrupts

GIC is a centralized resource for managing interrupts sent to the CPUs from the PS and PL. The controller enables, disables, masks, and prioritizes the interrupt sources and sends them to the selected CPU (or CPUs) in a programmed manner as the CPU interface accepts the next interrupt. In addition, the controller supports security extension for implementing a security-aware system.

The controller is based on the ARM Generic Interrupt Controller Architecture version 1.0 (GIC v1), non-vectored.

The private bus on the CPU accesses the registers for fast read/write response by avoiding temporary blockage or other bottlenecks in the Interconnect.

The interrupt distributor centralizes all interrupt sources before dispatching the one with the highest priority to the individual CPUs.

The GIC ensures that, when you target an interrupt to several CPUs, only one CPU takes the interrupt at a time. All interrupt sources contain a unique interrupt ID number. All interrupt sources have their own configurable priority and list of targeted CPUs.

Both the *Zynq-7000 AP SoC Technical Reference Manual* ([UG585](#)) and the *Zynq-7000 Software Developers Guide* ([UG821](#)) contain information regarding the logic blocks in the Zynq-7000 device.

Interconnect between PS and PL

AXI_HP Interfaces



The four AXI_HP interfaces provide PL bus masters with high bandwidth data paths to the DDR and OCM memories. Each interface includes two FIFO buffers for read and write traffic. The PL to the memory interconnect routes the high-speed AXI_HP ports either to two DDR memory ports or to the OCM. The AXI_HP interfaces are also referenced as AXI FIFO interface (AFI), to emphasize their buffering capabilities.

You must enable the PL level shifters using `LVL_SHIFTTR_EN` before PL logic communication can occur.

Enable these interfaces by selecting **PS-PL Configuration** from the Page Navigator and expanding the **HP Slave AXI Interface** option.

Alternatively, click the following block in the ZYNQ7 block diagram.

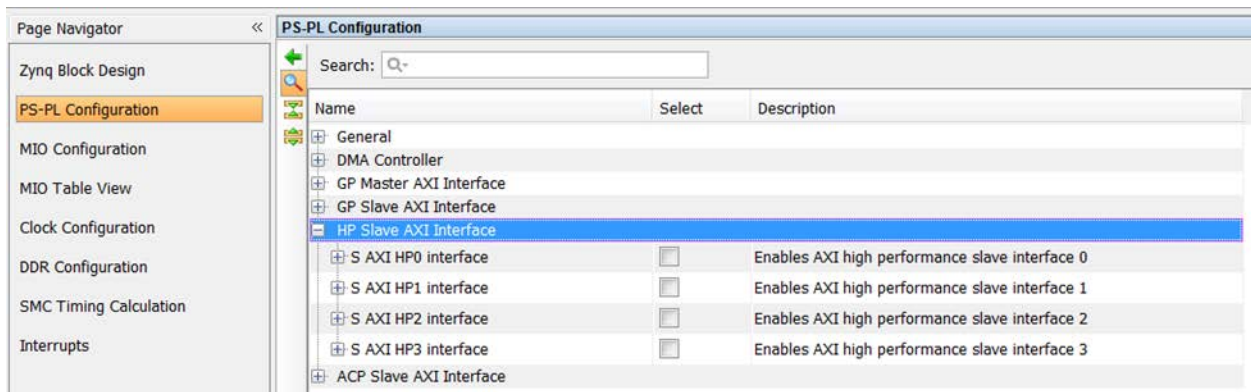


Figure 17: Enabling AXI HP Interfaces

The interfaces provide a high throughput data path between PL masters and PS memories including the DDR and on-chip RAM. The main features include:

32- or 64-bit data wide master interfaces (independently programmed per port)

- Efficient dynamic upsizing to 64 bits for aligned transfers in 32-bit interface mode, controllable using `AxCACHE`
- Automatic expansion to 64-bits for unaligned 32-bit transfers in 32-bit interface mode
- Programmable release threshold of write commands
- Asynchronous clock frequency domain crossing for all AXI interfaces between the PL and PS
- Smoothing out of “long-latency” transfers using 1 KB (128 by 64 bit) data FIFOs for both reads and writes
- QoS signaling available from PL ports
- Command and Data FIFO fill-level counts available to the PL
- Standard AXI 3.0 interfaces supported
- Programmable command issuance to the interconnect, separately for read and write commands
- Large slave interface read acceptance capability in the range of 14 to 70 commands (burst length dependent)
- Large slave interface write acceptance capability in the range of 8 to 32 commands (burst length dependent)

AXI_ACP Interface

The Accelerator Coherency Port (ACP) provides low-latency access to programmable logic masters, with optional coherency and L1 and L2 cache.

From a system perspective, the ACP interface has similar connectivity as the APU CPUs. Due to this close connectivity, the ACP directly competes for resource access outside of the APU block. You must enable the PL level shifters using the `LVL_SHFTR_EN` before PL logic communication can occur.

In the ZYNQ7 block diagram, click the following block to configure the AXI_ACP.



Alternatively, in the Page Navigator, select the **PS-PL Configuration** and expand the **ACP Slave AXI Interface** option.

The following figure shows the ACP AXI Slave Configuration page.

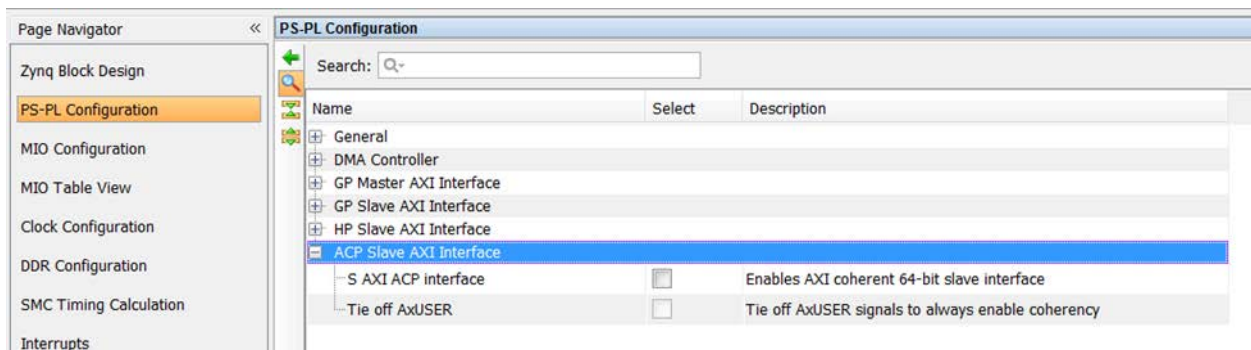


Figure 18: ACP Slave AXI Interface Page

AXI_GP Interfaces

AXI_GP features include:

- Standard AXI protocol
- Data bus width: 32
- Master port ID width: 12
- Master port issuing capability: 8 reads, 8 writes
- Slave port ID width: 6
- Slave port acceptance capability: 8 reads, 8 writes

These interfaces are connected directly to the ports of the master interconnect and the slave interconnect, without any additional FIFO buffering, unlike the AXI_HP interfaces, which have elaborate FIFO buffering to increase performance and throughput. Therefore, the performance is constrained by the ports of the master interconnect and the slave interconnect. These interfaces are for general-purpose use only; they are not intended to achieve high performance.

You must enable the PL level shifters using `LVL_SHFTR_EN` before PL logic communication can occur.

In the ZYNQ7 block diagram, click the following block to configure the AXI_GP interfaces.



Alternatively, in the Page Navigator, select the **PS-PL Configuration** and expand **GP Master AXI Interface** and **GP Slave AXI Interface** options.

The following figure shows the GP AXI Master and Slave Configuration page.

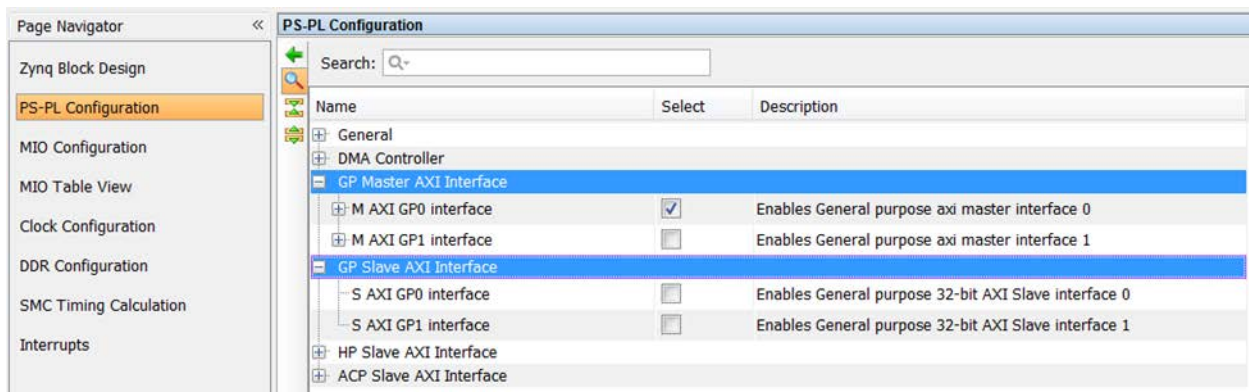


Figure 19: GP Master and Slave AXI Interfaces

Using the Programmable Logic (PL)

The PL provides a rich architecture of user-configurable capabilities.

Configurable logic blocks (CLB)

- 6-input look-up tables (LUTs) with memory capability within the LUT
- Register and shift register functionality
- Adders that can be cascaded

36 Kb block RAM

- Dual ports, up to 72-bits wide
- Configurable as dual 18 Kb
- Programmable FIFO logic
- Built-in error correction circuitry

Digital signal processing — DSP48E1 Slice

- 25 × 18 two's complement multiplier/accumulator high-resolution (48 bit) signal processor
- Power saving 25-bit pre-adder to optimize symmetrical filter applications
- Advanced features: optional pipelining, optional ALU, and dedicated buses for cascading

Clock management

- UHigh-speed buffers and routing for low-skew clock distribution
- Frequency synthesis and phase shifting
- Low-jitter clock generation and jitter filtering

Configurable I/Os

- High-performance SelectIO™ technology
- High-frequency decoupling capacitors within the package for enhanced signal integrity
- Digitally controlled impedance that can be tri-state for lowest power, high-speed I/O operation
- High range (HR) I/Os support 1.2 V to 3.3 V
- High performance (HP) I/Os support 1.2 V to 1.8 V (7z030, 7z045, and 7z100 devices)

Low-power gigabit transceivers

- (7z030, 7z045, and 7z100 devices)
- High-performance transceivers capable of up to 12.5 Gb/s (GTX)
- Low-power mode optimized for chip-to-chip interfaces

- Advanced transmit pre- and post-emphasis, and receiver linear (CTLE) and decision feedback equalization (DFE), including adaptive equalization for additional margin

Analog-to-digital converter (XADC)

- Dual 12-bit 1 MSPS analog-to-digital converters (ADCs)
- Up to 17 flexible and user-configurable analog inputs
- On-chip or external reference option
- On-chip temperature ($\pm 4^{\circ}\text{C}$ max error) and power supply ($\pm 1\%$ max error) sensors
- Continuous JTAG access to ADC measurements

Integrated interface blocks for PCI Express designs (7z030, 7z045, and 7z100 devices)

- Compatible to the PCI Express base specification 2.1 with Endpoint and Root Port capability
- Supports Gen1 (2.5 Gb/s) and Gen2 (5.0 Gb/s) speeds

Advanced configuration options, advanced error reporting (AER), end-to-end CRC (ECRC)

Custom Logic

The Vivado™ IP packager lets you and third party IP developers use the Vivado IDE to easily prepare an Intellectual Property (IP) design for use in the Vivado IP catalog.

The IP user can then instantiate this third party IP into their design in the Vivado Design Suite.

When IP developers use the Vivado Design Suite IP packaging flow, the IP user has a consistent experience whether using Xilinx IP, third party IP, or customer-developed IP within the Vivado Design Suite.

IP developers can use the IP packager feature to package IP files and associated data into a ZIP file. The IP user receives this generated ZIP file and installs the IP into the Vivado Design Suite IP Catalog. The IP user then customizes the IP through parameter selections and generates an instance of the IP.



RECOMMENDED: *To verify the proper packaging of the IP before handing it off to the IP user, Xilinx® recommends that the IP developer run each IP module completely through the IP user flow to validate that the IP is ready for use.*

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.

Name	Version
AHB-Lite to AXI Bridge	2.0
AXI-Stream FIFO	4.0
AXI4-Stream to Video Out	3.0
AXI AHB-Lite Bridge	2.0
AXI APB Bridge	2.0
AXI BFM Cores	4.0
AXI BRAM Controller	3.0
AXI CAN	5.0
AXI Central Direct Memory Access	4.0
AXI Chip2Chip Bridge	4.0
AXI Clock Converter	2.0
AXI Crossbar	2.0
AXI Data FIFO	2.0
AXI DataMover	5.0
AXI Data Width Converter	2.0
AXI Direct Memory Access	7.0
AXI EMC	2.0
AXI EPC	2.0
AXI Ethernet	4.0
AXI Ethernet Buffer	1.0
AXI Ethernet Clocking	1.0
AXI EthernetLite	2.0
AXI GPIO	2.0
AXI HWICAP	3.0
AXI IIC	2.0
AXI INTC	3.0
AXI Interconnect	2.0
AXI Memory Mapped To PCI Express	2.0
AXI Performance Monitor	4.0
AXI Protocol Checker	1.0
AXI Protocol Converter	2.0
AXI Quad SPI	3.0
AXI Register Slice	2.0
AXI TFT Controller	2.0
AXI Timebase Watchdog Timer	2.0
AXI TIMER	2.0
AXI UART16550	2.0

Figure 20: IP Integrator IP Catalog

Completing Connections

After you have configured the ZYNQ7 Processing System, you can start to instantiate other IPs that go in the Programmable Logic portion of the device.

In the IP integrator canvas, right-click and select **Add IP**.

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

Block Automation

The Block Automation feature is available when a microprocessor such as the Zynq Processing System 7 or the MicroBlaze processor is instantiated in the block diagram of the IP integrator tool.

Click **Run Block Automation** to get assistance with putting together a simple ZYNQ7 Processing System, as shown in the following figure.

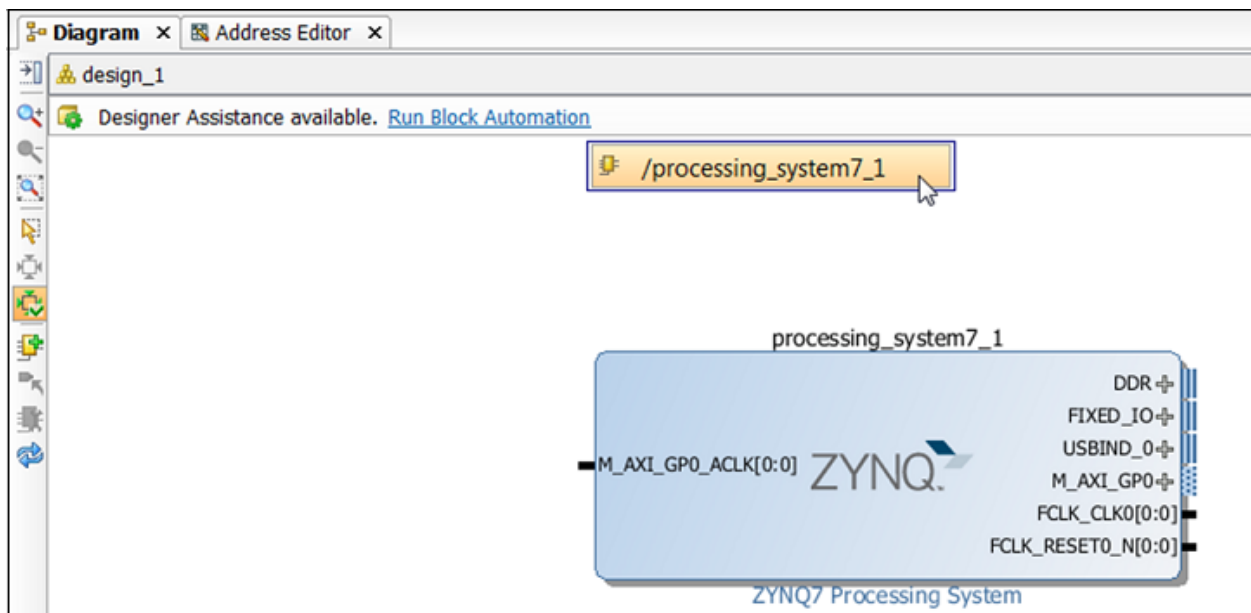


Figure 21: Run Block Automation Feature

The Run Block Automation dialog box informs you about the options that are available for automation as shown in the following figure.

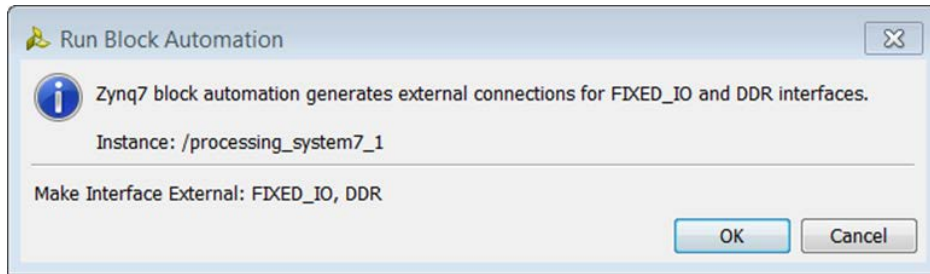


Figure 22: Run Block Automation for Zynq Processor Dialog Box

After you click **OK**, the Block Automation feature creates the basic system, as shown in the following figure.

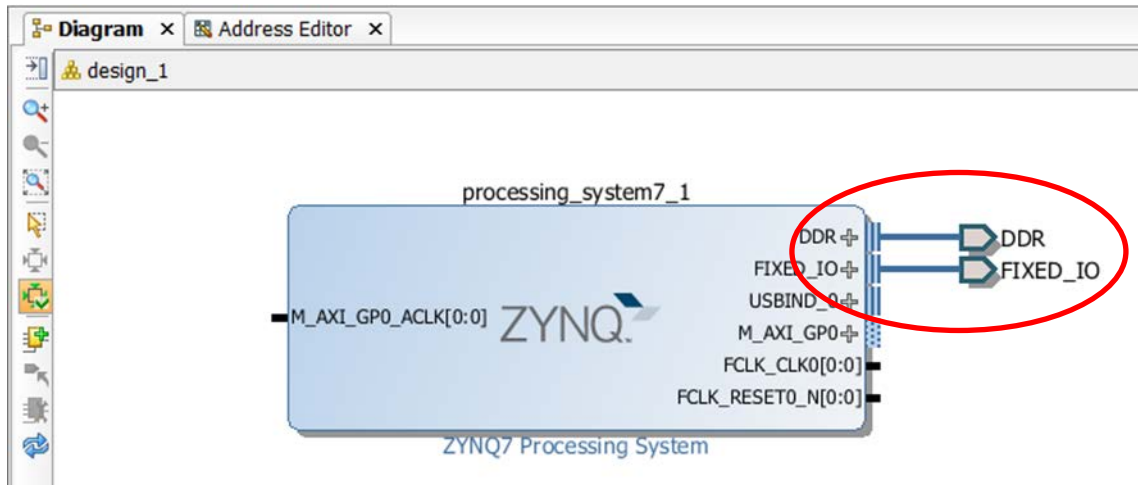


Figure 23: IP Integrator Window after Running Block Automation

In the previous figure, observe that the external DDR and `FIXED_IO` interfaces connect to external ports.

Using Connection Automation

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

In the following figure, two IP: the AXI BRAM Controller and the Block Memory Generator, are instantiated along with the ZYNQ7 Processing System IP.

The IP integrator tool determines that there is a potential connection between the AXI BRAM Controller and the ZYNQ7 IP; consequently, the Connection Automation feature becomes available.

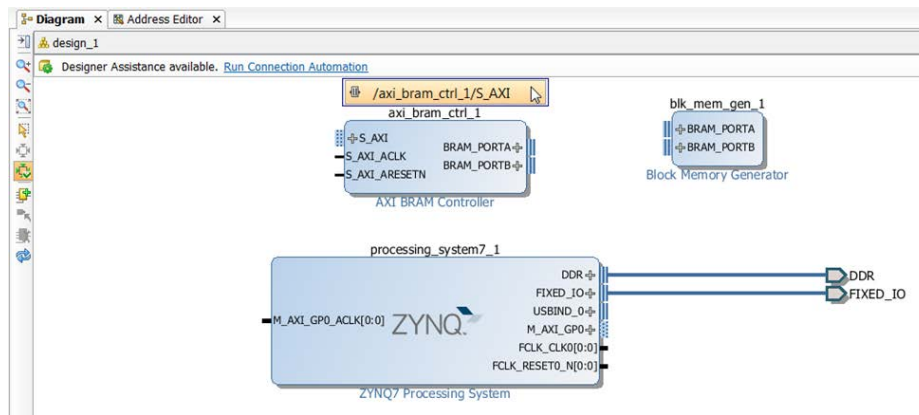


Figure 24: Connection Automation Feature

When you click **Run Connection Automation**, the Connection Automation feature does the following:

- Instantiates an AXI Interconnect, and a `proc_sys_reset` IP
- Connects the `axi_bram_ctrl` to the ZYNQ7 using the AXI Interconnect
- Appropriately connects the `proc_sys_reset` IP as shown in the following figure

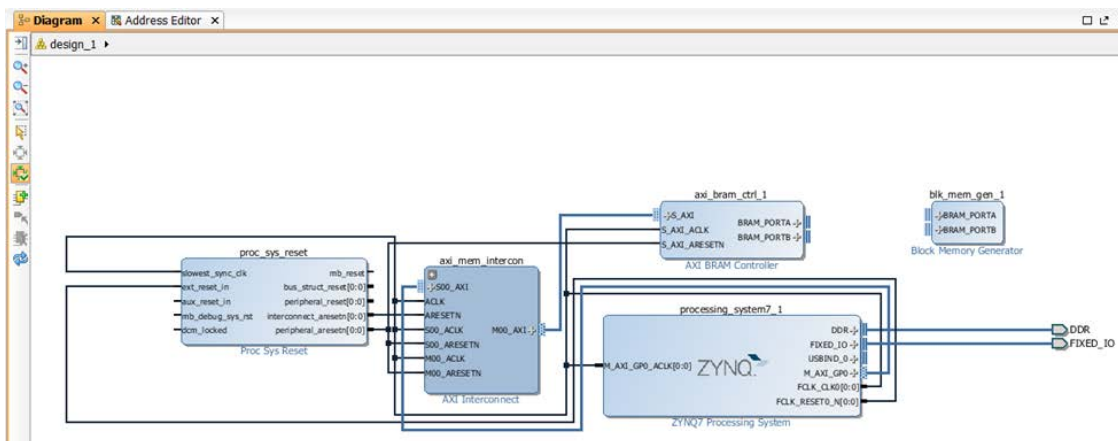


Figure 25: Block Diagram after Connection Automation

Using the Board Automation

The Vivado IP integrator tool also provides a Board Automation feature when using a Xilinx Target Reference Platform, such as the ZC702.

This feature provides connectivity of the ports of an IP to the FPGA pins on the target board. The IP configures accordingly, and based upon your selections, connects the I/O ports. The Board Automation feature automatically generates the physical constraints for those IP that require physical constraints.

In the following figure, a GPIO IP is instantiated in the subsystem design. As soon as the IP is instantiated, the Run Connection Automation feature is available again.

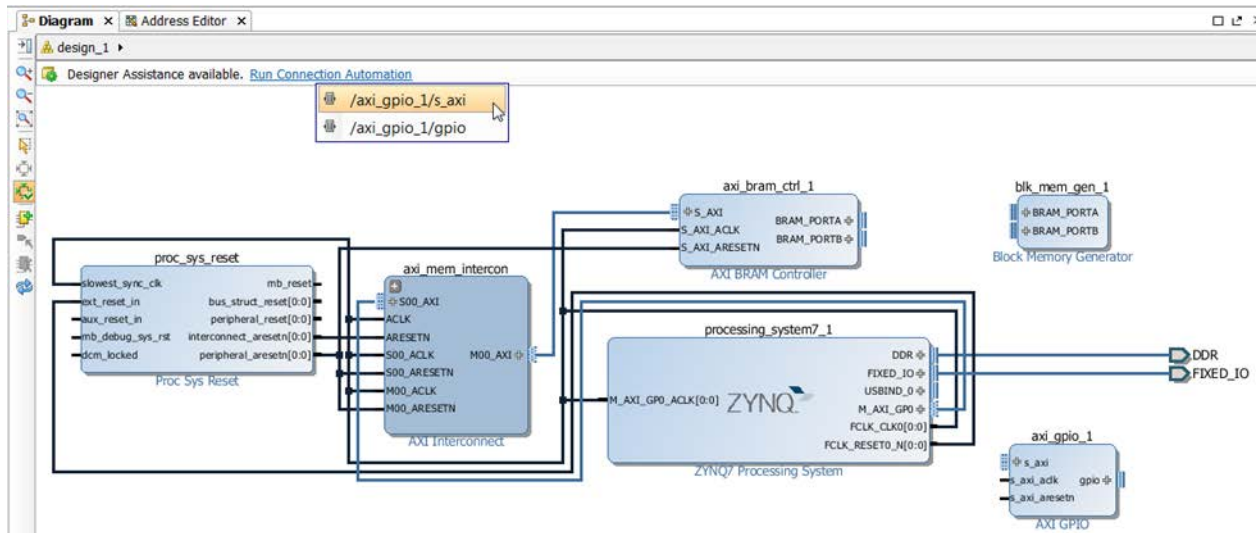


Figure 26: Connection Automation for Xilinx Target Reference Platform

When you click **Run Connection Automation**, the feature informs you that two connections are possible:

- The first possible connection is the S_AXI port and the associated AXI clock and reset ports to the ZYNQ7 using the AXI interconnect.
- The second is the GPIO I/O port of the GPIO IP, which can connect to an external I/O port.

When you select both of these options, the canvas looks like the following figure:

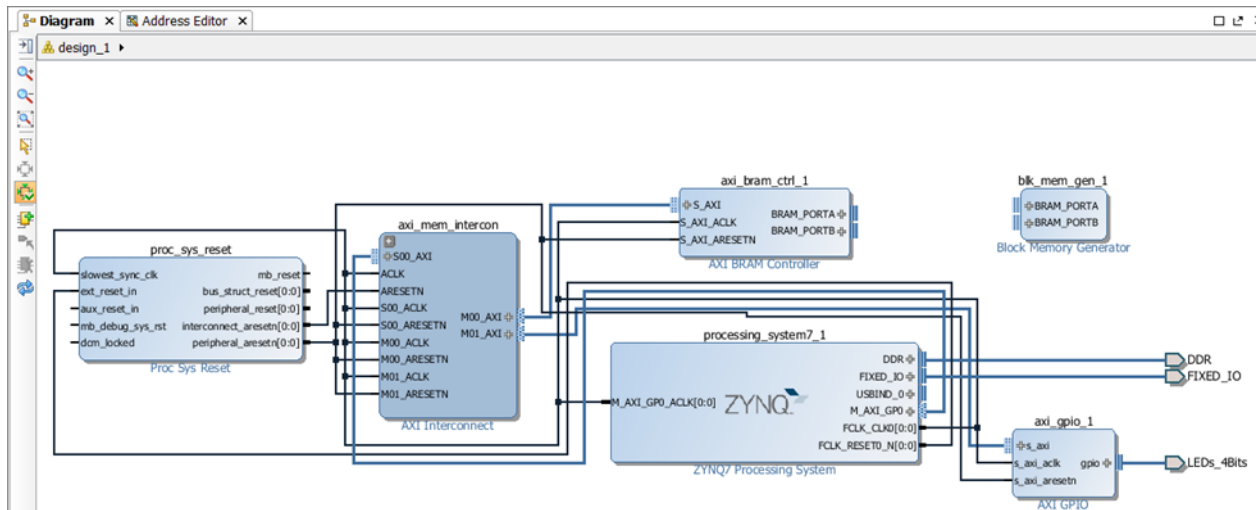


Figure 27: Board Automation Feature in IP Integrator

The GPIO ports connect to the appropriate LEDs on the target board.

When you generate the Output Products for the subsystem, the appropriate physical constraints generate for the GPIO I/O port.

Manual Connections in a Design

The following figure shows that you need to connect the AXI BRAM Controller to the Block Memory Generator. You can do this manually.

As you move the cursor near an interface or pin connector on an IP block, the cursor changes into a pencil.

Click an interface or pin connector on an IP block, and drag and drop the connection to the destination block, as shown in the following figure.

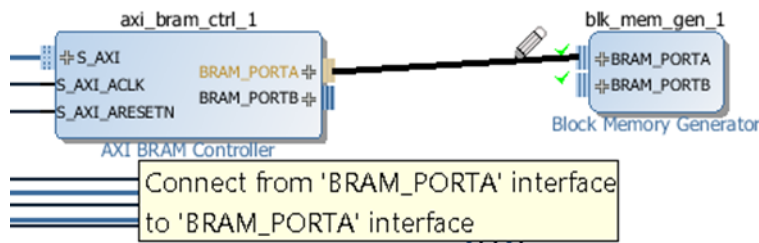


Figure 28: Manually Connecting Ports

Manually Creating and Connecting to I/O Ports

You can manually create external I/O ports in the Vivado IP integrator. There are three ways to connect signals or interfaces to external I/O ports: selecting a pin, a bus, or an interface connection.

- Right-click and select **Make External**. It is possible to 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 diagram.

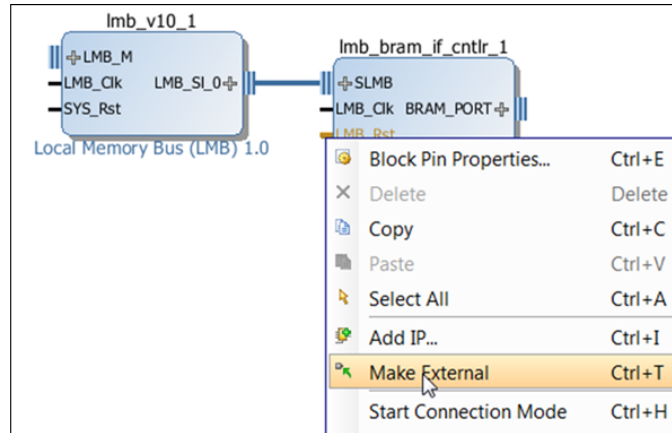


Figure 29: Make External Connection

- Right-click and select **Create Port**, as shown in the following figure.

Use this command for non-interface signals, such as a `clock`, `reset`, or `uart_txd`.

The Create Port option gives more control in terms of specifying the input/output, the bit-width and the type (clk, reset, or data). In case of a clock, you can even specify the input frequency.

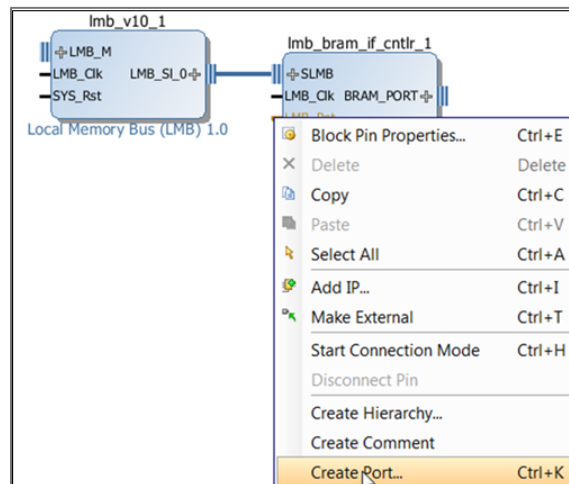


Figure 30: Create Port Option

- Right-click and select **Create Interface Port**.

This command creates ports on the interface for groupings of signals that share a common function.

For example, the S_AXI is an interface port on several Xilinx IP. The command gives more control in terms of specifying the interface type and the mode (master or slave).

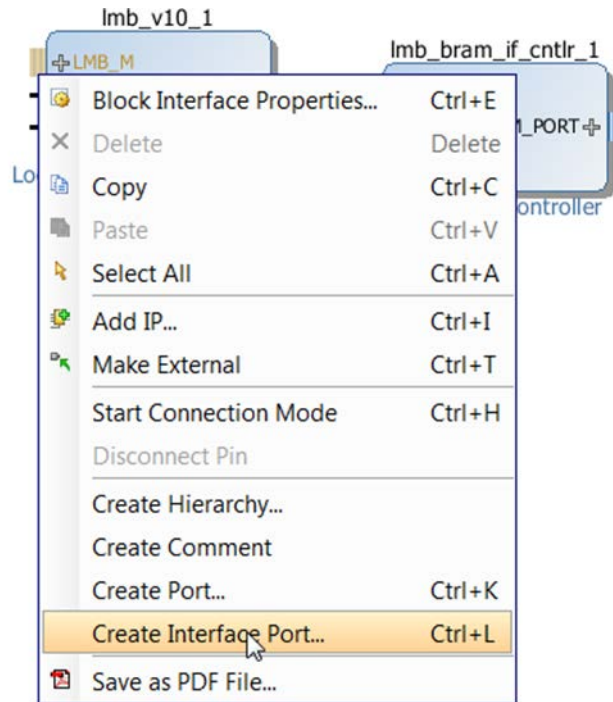


Figure 31: Create Interface Port Option

Memory Mapping in Address Editor

To generate the address map for this design:

1. Click the Address Editor tab above the diagram.
2. Click the **Auto Assign Address** button (bottom on the left side).

If you generate the RTL from an IP integrator diagram without first generating addresses, a prompt opens that lets you select to have the tool automatically assign addresses.

You can manually set addresses by entering values in the **Offset Address** and **Range** columns.

Note: The Address Editor tab only opens if the diagram contains an IP block that functions as a bus master (such as the ZYNQ7 processor) in the design.

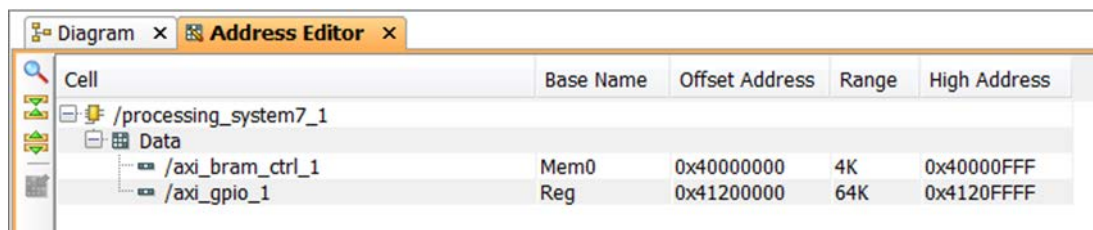


Figure 32: Memory Mapping Peripherals

Running Design Rule Checks

The Vivado IP integrator runs basic DRCs in real time as you put the design together.

However, there is a potential for something to go wrong during design creation. For example, the frequency on a clock pin might not be set correctly.

To run a comprehensive DRC, click the **Validate Design** button, as shown in the following figure.

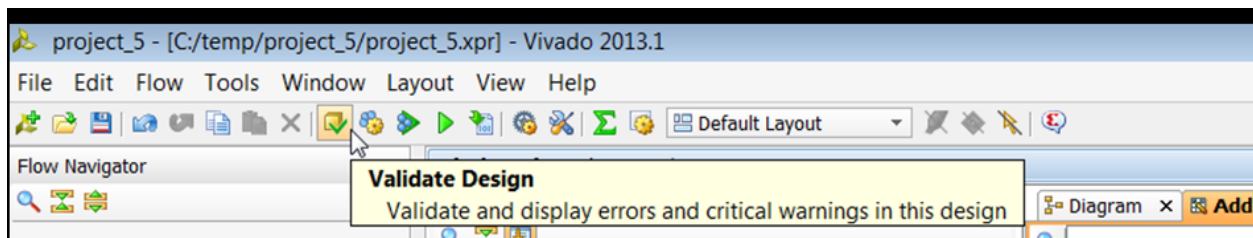


Figure 33: Validate Design

If there are no warnings or errors about the design, a dialog box, such as in the following figure, displays:



Figure 34: Validate Design Dialog Box

Integrating a Block Diagram in the Top-Level Design

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

- Generate the output products.
- Create a HDL wrapper

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

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

In the Vivado IDE Sources window, right-click the block diagram and select **Generate Output Products**, as shown in the following figure.

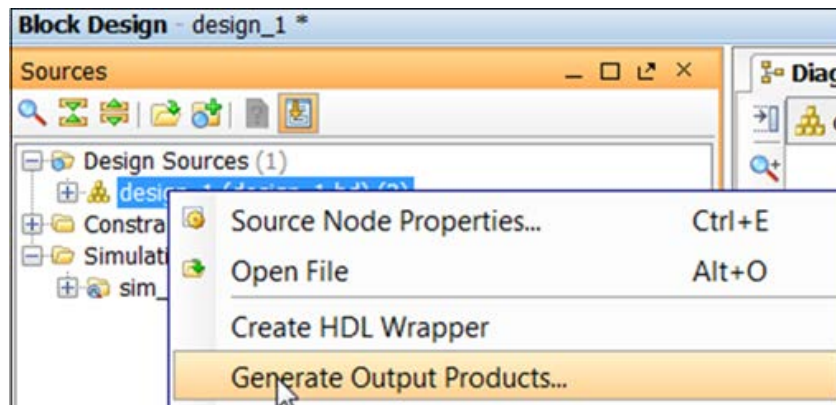


Figure 35: Generate Output Products Option

You can integrate an IP integrator block diagram into a higher-level design. To do so, instantiate the design in a higher-level HDL file.

To instantiate at a higher-level, select the Vivado IDE Sources block diagram and select **Create HDL Wrapper**, as shown in the following figure.

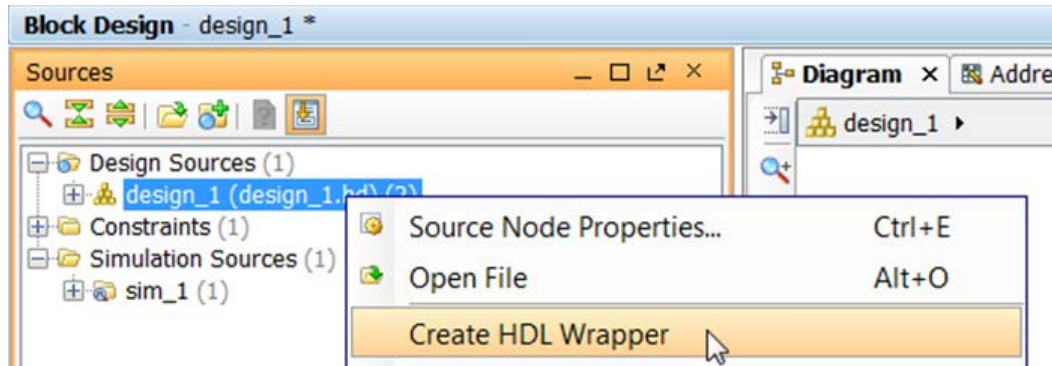


Figure 36: Creating an HDL Wrapper

This generates a top-level HDL file for the IP integrator subsystem. At this point, you are ready to take your design through the other design flows: elaboration, synthesis, and implementation.

Vivado Pin Planner View of PS I/O

The *Zynq-7000 All Programmable SoC PCB Design and Pin Planning Guide*, ([UG933](#)) provides a detailed description of guidelines for PCB Design and Pin Planning for Zynq-7000 devices.

Vivado IDE Generated Embedded Files

When you export a Zynq-7000 processor hardware design from the IP integrator tool to SDK, the IP integrator generates the following files:

system.xml	This file opens by default when you launch SDK and displays the address map of your system.
ps7_init.c ps7_init.h	The ps7_init.c and ps7_init.h files contain the initialization code for the Zynq Processing System and initialization settings for DDR, clocks, PLLs, and MIOs. SDK uses these settings when initializing the processing system so applications can run on top of the processing system. Some settings in the processing system are in a fixed state for the ZC702 evaluation board.
ps7_init.tcl	This is the Tcl version of the INIT file.
ps7_init.html	The INIT file describes the initialization data.

See the *Zynq-7000 AP SoC Software Developer User Guide* ([UG821](#)) for more information about generated files.

Using the Software Development Kit (SDK)

The Xilinx Software Development Kit (SDK) provides a complete environment for creating software applications targeted for Xilinx embedded processors. It includes a GNU-based compiler toolchain (GCC compiler, GDB debugger, utilities, and libraries), JTAG debugger, flash programmer, drivers for Xilinx IP and bare-metal board support packages, middleware libraries for application-specific functions, and an IDE for C/C++ bare-metal and Linux application development and debugging. Based upon the open source Eclipse platform, SDK incorporates the C/C++ Development Toolkit (CDT).

Features include:

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

SDK Availability

SDK is available from the Xilinx Vivado Design Suite installation package or as a standalone installation. SDK also includes an application template for creating a First Stage Bootloader (FSBL), as well as a graphical interface for building a boot image. SDK contains a help system that describes concepts, tasks, and reference information.

Exporting a Hardware Description

1. In the Flow Navigator, click **Open Block** to invoke the IP integrator design as shown in the following figure.



Figure 37: IP Integrator: Open Block Design

Now you are ready to export your design to SDK.

- From the main Vivado IDE, select **File > Export Hardware for SDK** as shown in the following figure.



Figure 38: Export Hardware for SDK

The Export Hardware for SDK dialog box opens.

- Ensure that you have checked the **Export Hardware**, **Include Bitstream**, and **Launch SDK** as shown in the following figure.

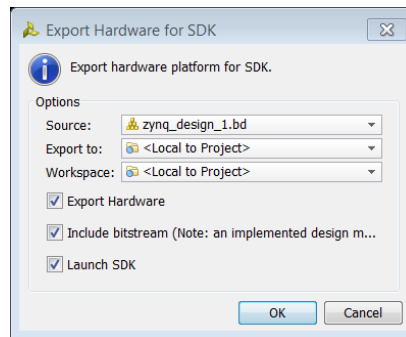


Figure 39: Export Hardware for SDK

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

You can do further debug and downloading of the software from SDK.

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

Using a MicroBlaze Processor in an Embedded Design

Introduction to MicroBlaze Processor Design

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

The MicroBlaze™ embedded processor is a Reduced Instruction Set Computer (RISC) core, optimized for implementation in Xilinx® Field Programmable Gate Arrays (FPGAs).

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

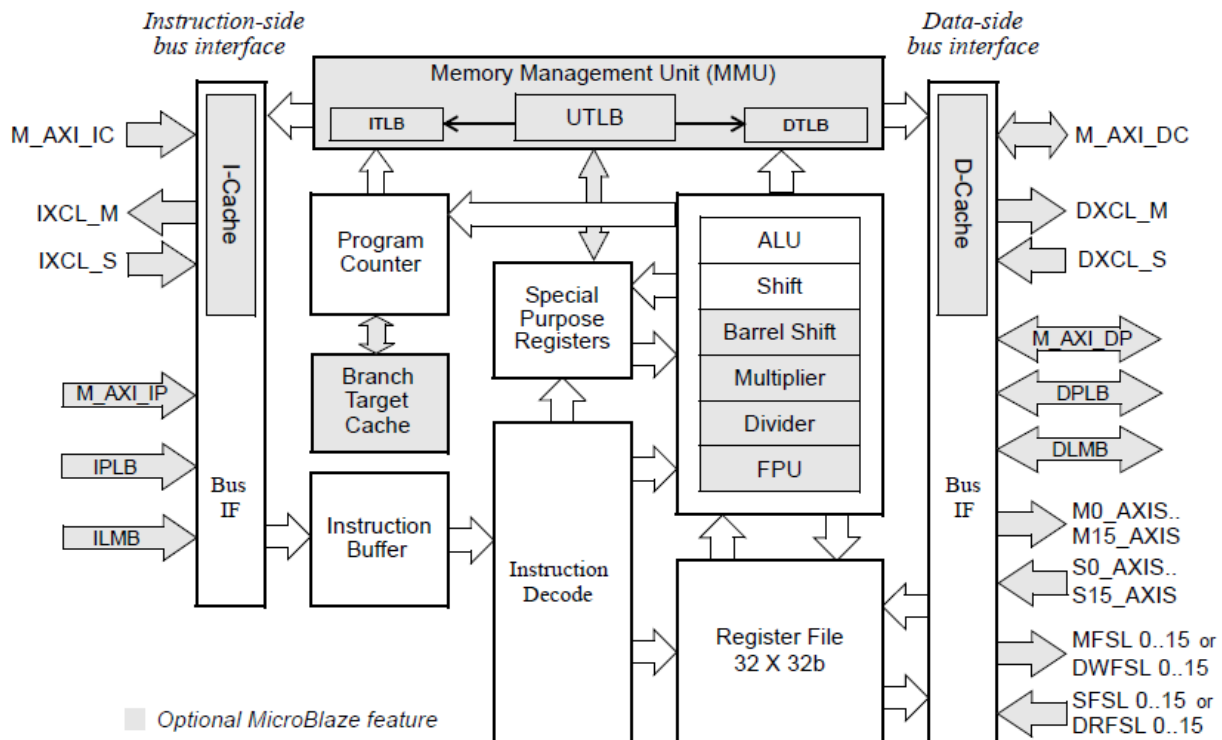


Figure 40 Block Diagram of MicroBlaze Core

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

The fixed feature set of the processor includes:

- Thirty-two 32-bit general purpose registers
- 32-bit instruction word with three operands and two addressing modes
- 32-bit address bus
- Single issue pipeline

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



RECOMMENDED: Older (deprecated) versions of MicroBlaze support a subset of the optional features described in this manual. Only the latest (preferred) version of MicroBlaze (v9.0) supports all options. Xilinx recommends that new designs use the latest preferred version of the MicroBlaze processor.

See the *MicroBlaze Processor Reference Guide* ([UG081](#)) for more information about the MicroBlaze processor design.

Creating an IP Integrator Design with the MicroBlaze Processor

Designing with a MicroBlaze processor is different using the Vivado IDE than it was using the ISE® Design Suite and Embedded Development Kit (EDK).

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

A variety of IP are available in the Vivado IDE IP Catalog to meet the needs of complex designs.

You can also add custom IP to the IP Catalog.

Creating an IP Integrator Design with the MicroBlaze processor

When you select the IP integrator **Create Block Design** button,  **Create Block Design** a dialog box opens for you to enter the **Design Name**, as shown in the following figure.

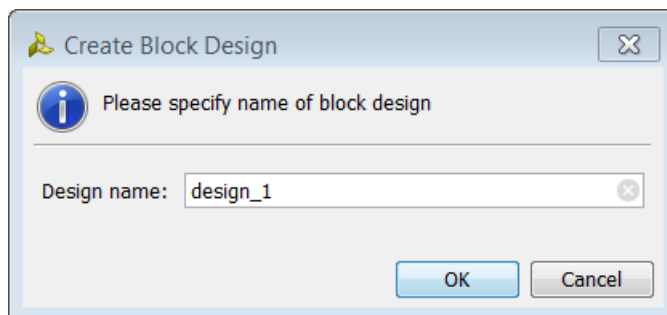


Figure 41: Design Name Dialog Box

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

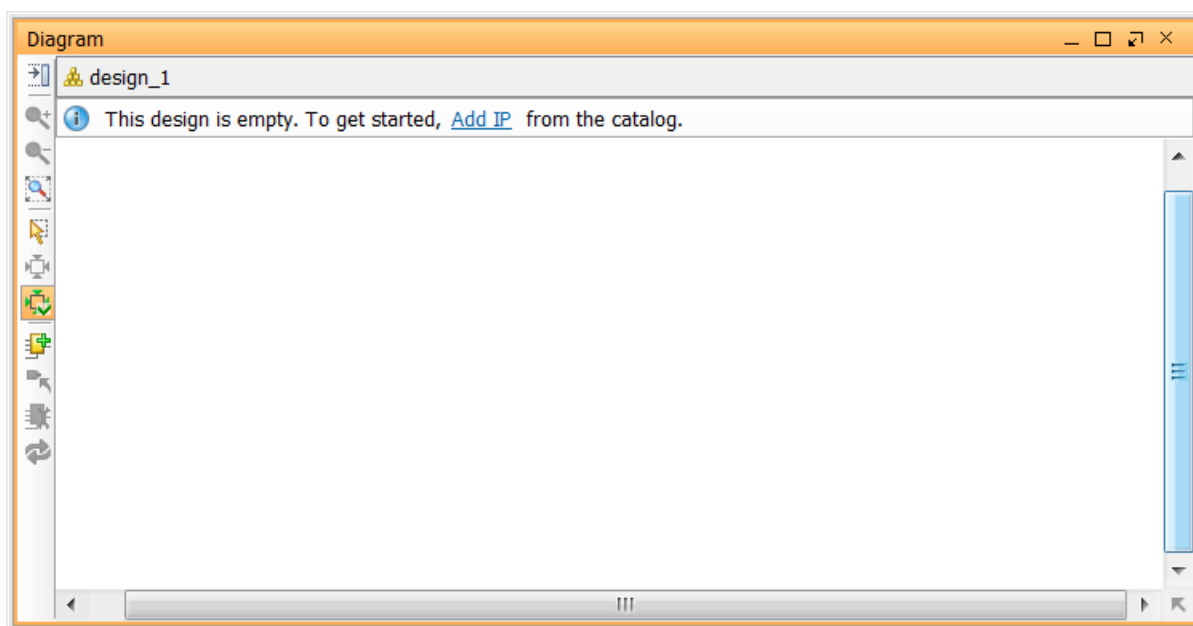


Figure 42: Block Design Window

Within the empty design, there is an option to **Add IP** from the IP Catalog. You can also right-click in the canvas to open an option to add IP.

Select the **Add IP** option, and a Search box opens where you can search for, and select the **MicroBlaze** processor, as shown in the following figure.

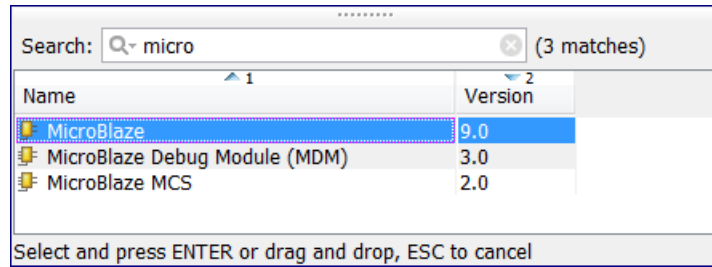


Figure 43: Search IP with MicroBlaze Processing System

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

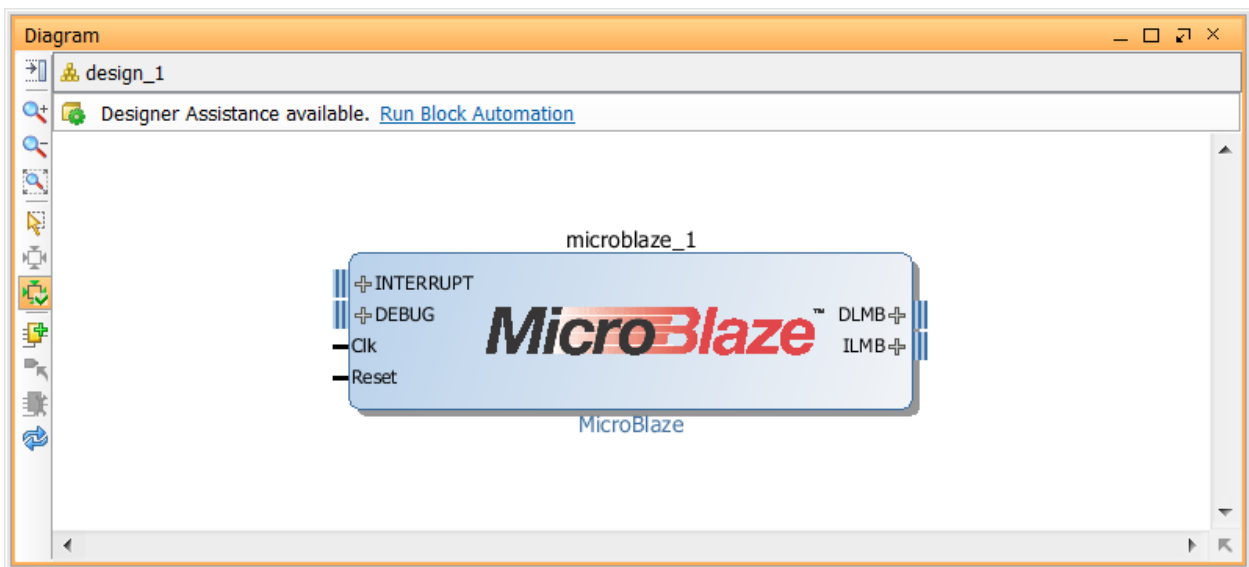


Figure 44: Graphical Display of Default MicroBlaze Processing System

The Tcl command is specified as follows:

```
create_bd_cell -type ip -vlnv xilinx.com:ip:microblaze:9.0
microblaze_1
```

Double-click the MicroBlaze IP in the canvas to invoke the **Recustomize IP** process, which displays the Re-customize IP for the MicroBlaze processor, dialog box.

MicroBlaze Configuration Window

The MicroBlaze Configuration wizard provides:

- A template-based configuration dialog box for one-click configuration
- Estimates of MicroBlaze relative area, frequency, and performance, based on options set in the dialog boxes, giving immediate feedback
- Guidance through the configuration process
- Tool tips for all configuration options to understand the effect of each option
- Direct access to all options in the tabbed interface using the **Advanced** button

The MicroBlaze Configuration wizard has the following wizard pages, which enable based on the selected General Settings options:

- **Configuration Wizard:** First page that showing template selection and general settings.
- **General:** Selection of execution units, optimization that is always shown
- **Exceptions:** Exceptions to enable, which is shown if exceptions are selected on the first page
- **Debug:** Number of breakpoints and watchpoints, which is shown if debug is enabled
 - **Cache:** Cache settings, which is shown if caches are selected
 - **MMU:** MMU settings, which is shown if memory management is selected
 - **Buses:** Bus settings. Last page, always shown

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

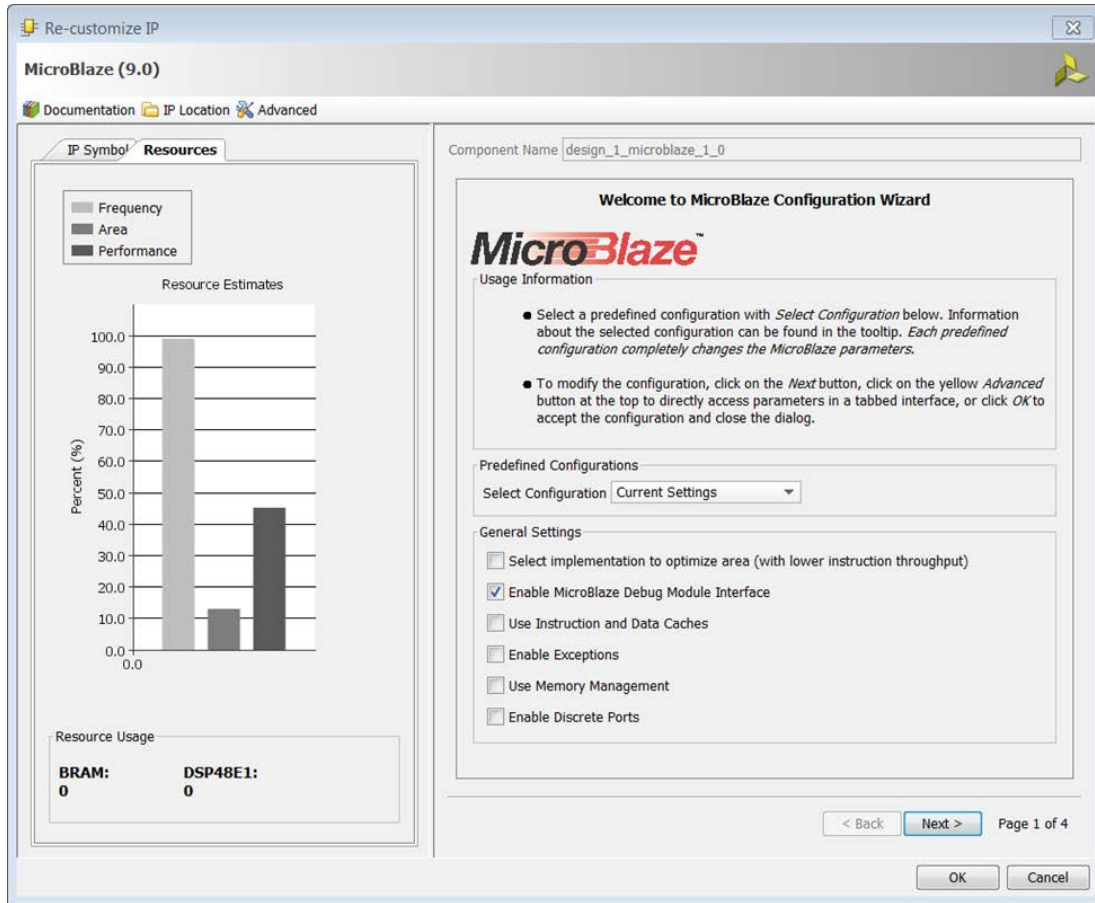


Figure 45: MicroBlaze Configuration Wizard

The left of the dialog box shows the relative values of the frequency, area and performance for the current settings.

- **Frequency:** This value is the 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 benchmark systems, 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:** This value is the estimated area percentage in LUTs relative to the maximum area using this architecture, which gives an indication of the relative MicroBlaze area achievable with the current settings.

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

- **Performance:** This value indicates the relative MicroBlaze 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:** This value is the total number of block RAMs used by MicroBlaze. The instruction and data caches, and the branch target cache use block RAMs, and well as the Memory Management Unit (MMU), which uses one block RAM in virtual or protected mode.
- **DSP48 or MULT18:** This value is the total number of DSP48 or MULT18 used by MicroBlaze. The integer multiplier, and the Floating Point Unit (FPU) use this total value to implement float multiplication.

MicroBlaze Configuration Wizard Welcome Page

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

Whenever you modify an option, you received direct feedback that shows the estimated relative change in performance, frequency, and area in the information display. The options are:

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

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

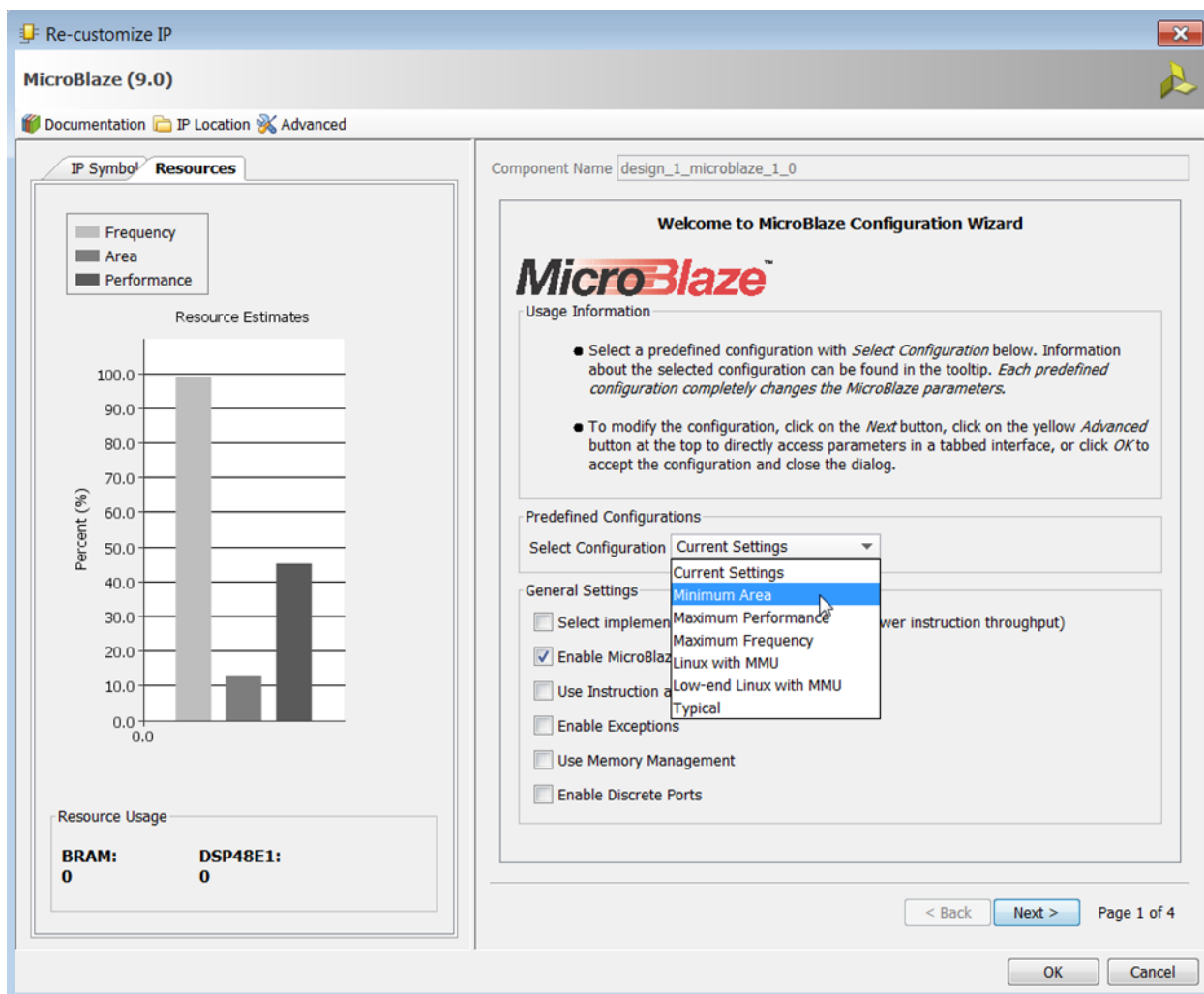


Figure 46: MicroBlaze Predefined Configuration Settings

General Settings

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

- **Select implementation to optimize area** (with lower instruction throughput): Enables area optimized MicroBlaze. When this parameter is set, the implementation optimizes area, particularly by reducing the pipeline from five stages to three.



RECOMMENDED: *It is recommended to enable area optimization on architectures with limited resources such as Artix®-7. However, if performance is critical, this parameter should not be set, because then some instructions require additional clock cycles to execute.*

Note: You cannot use the *Memory Management Unit (MMU), Branch Target Cache, Instruction Cache Streams, Instruction Cache Victims, Data Cache Victims, and AXI Coherency Extension (ACE) with area optimization.*

- **Enable Microblaze Debug Module Interface:** Enable debug to be able to download and debug programs using Xilinx Microprocessor Debugger. Unless area resources are very critical, it is recommended that debugging be always enabled.
- **Use Instruction and Data Caches:** You can use MicroBlaze with an optional instruction cache for improved performance when executing code that resides outside the LMB address range. The instruction cache has the following features:
 - Direct mapped (1-way associative)
 - User selectable cacheable memory address range
 - Configurable cache and tag size
 - Caching over AXI4 interface (M_AXI_IC) or CacheLink (XCL) interface
 - Option to use 4 or 8 word cacheline
 - Cache on and off controlled using a bit in the MSR
 - Optional WIC instruction to invalidate instruction cache lines
 - Optional stream buffers to improve performance by speculatively prefetching instructions
 - Optional victim cache to improve performance by saving evicted cache lines
 - Optional parity protection that invalidates cache lines if a Block RAM bit error is detected
 - Optional data width selection to either use 32 bits, an entire cache line, or 512 bits

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

Enable Exceptions: Enables exceptions when using an operating system with exception support, or when explicitly adding exception handlers in a standalone program.

Use Memory Management: Enables Memory Management if planning to use an operating system – such as Linux –with support for virtual memory or memory protection.

Note: *When you enable area optimized MicroBlaze or stack protection, the Memory Management Unit is not available.*

- **Enable Discrete Ports:** Enables discrete ports on the MicroBlaze instance, which is useful for:
 - Generating software breaks (Ext_BRK, Ext_NM_BRK),
 - Managing processor sleep and wakeup (Sleep, Wakeup, Dbg_Wakeup),
 - Handling debug events (Debug_Stop, MB_Halted)
 - Signaling error when using fault tolerance (MB_Error).

MicroBlaze Configuration Wizard General Page

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

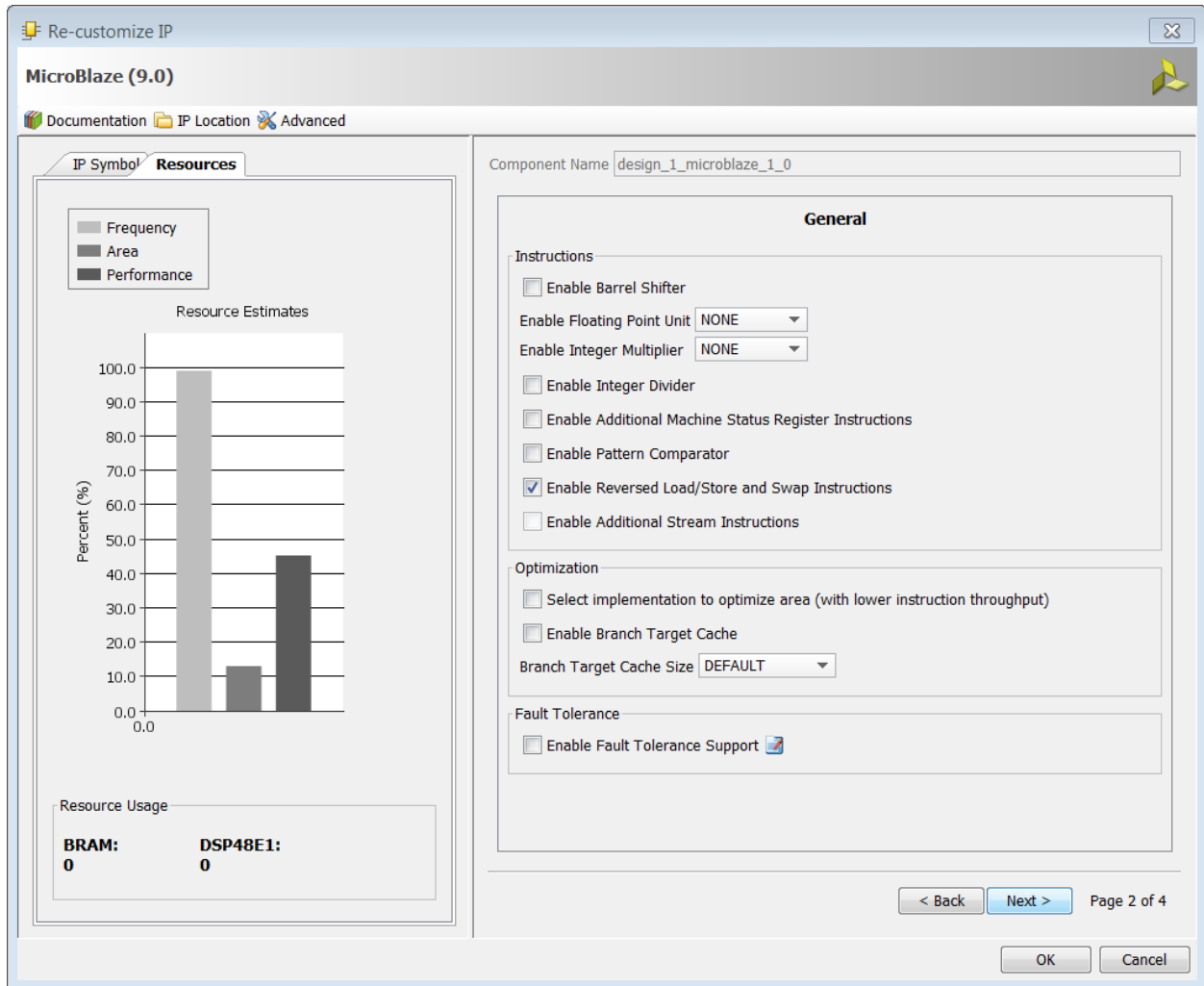


Figure 47: General Page of the MicroBlaze Configuration Wizard

Instructions

- **Enable Barrel Shifter:** Enables a hardware barrel shifter in MicroBlaze. This parameter enables the instructions `bsrl`, `bsra`, `bsll`, `bsrli`, `bsrai`, and `bslli`. Enabling the barrel shifter can dramatically improve the performance of an application, but increases the size of the processor. The compiler uses the barrel shift instructions automatically if this parameter is enabled.
- **Enable Floating Point Unit:** Enables a single-precision Floating Point Unit (FPU) based on the IEEE-754 standard. Using the FPU significantly improves the single-precision, floating point performance of the application and significantly increases the size of MicroBlaze.

Setting this parameter to `BASIC` enables the instructions `fadd`, `frsub`, `fmul`, `fdiv`, and `fcmp`. Setting it to `EXTENDED` also enables the instruction `flt`, `finit`, and `fsqrt`. The compiler will automatically use the FPU instructions corresponding to setting of this parameter.

- **Enable Integer Multiplier:** Enables a hardware integer multiplier in MicroBlaze. This parameter enables the instructions `mul` and `muli` when set to `MUL32`.

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

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

The `pcmpeq` and `pcmpne` instructions return 1 or 0 based on the equality of the two words. These instructions improve the performance of setting flags and the compiler uses them automatically.

Selecting this option also enables count leading zeroes instruction, `clz`. The `clz` instruction can improve performance of priority decoding, and normalization.

- **Enable Reversed Load/Store and Swap Instructions:** Enables reversed load/store and swap instructions `lbur`, `lhur`, `lwr`, `sbr`, `shr`, `swr`, `swapb`, and `swaph`. The reversed load/store instructions read or write data with opposite endianness, and the swap instructions allow swapping bytes or half-words in registers. These instructions are mainly useful to improve performance when dealing with big-endian network access with a little-endian MicroBlaze.
- **Enable Additional Stream Instructions:** Provides additional functionality when using AXI4-Stream links, including dynamic access instruction `GETD` and `PUTD` that use registers to select the interface. The instructions are also extended with variants that provide:
 - Atomic `GET`, `GETD`, `PUT`, and `PUTD` instructions
 - Test-only `GET` and `GETD` instructions
 - `GET` and `GETD` instructions that generate a stream exception if the control bit is not set.



IMPORTANT: The stream exception must be enabled to use these instructions, and at least one stream link must be selected.

Optimization

Select implementation to optimize area (with lower instruction throughput): This option is the same as in the [General Settings](#) options. **Enable Branch Target Cache:** When set, implements the branch target, which improves branch performance by predicting conditional branches and caching branch targets.

Note: To be able to use the Branch Target Cache, do not enable area optimization.

Fault Tolerance

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

MicroBlaze Configuration Wizard Exception Page

The following figure shows the MicroBlaze exception options page.

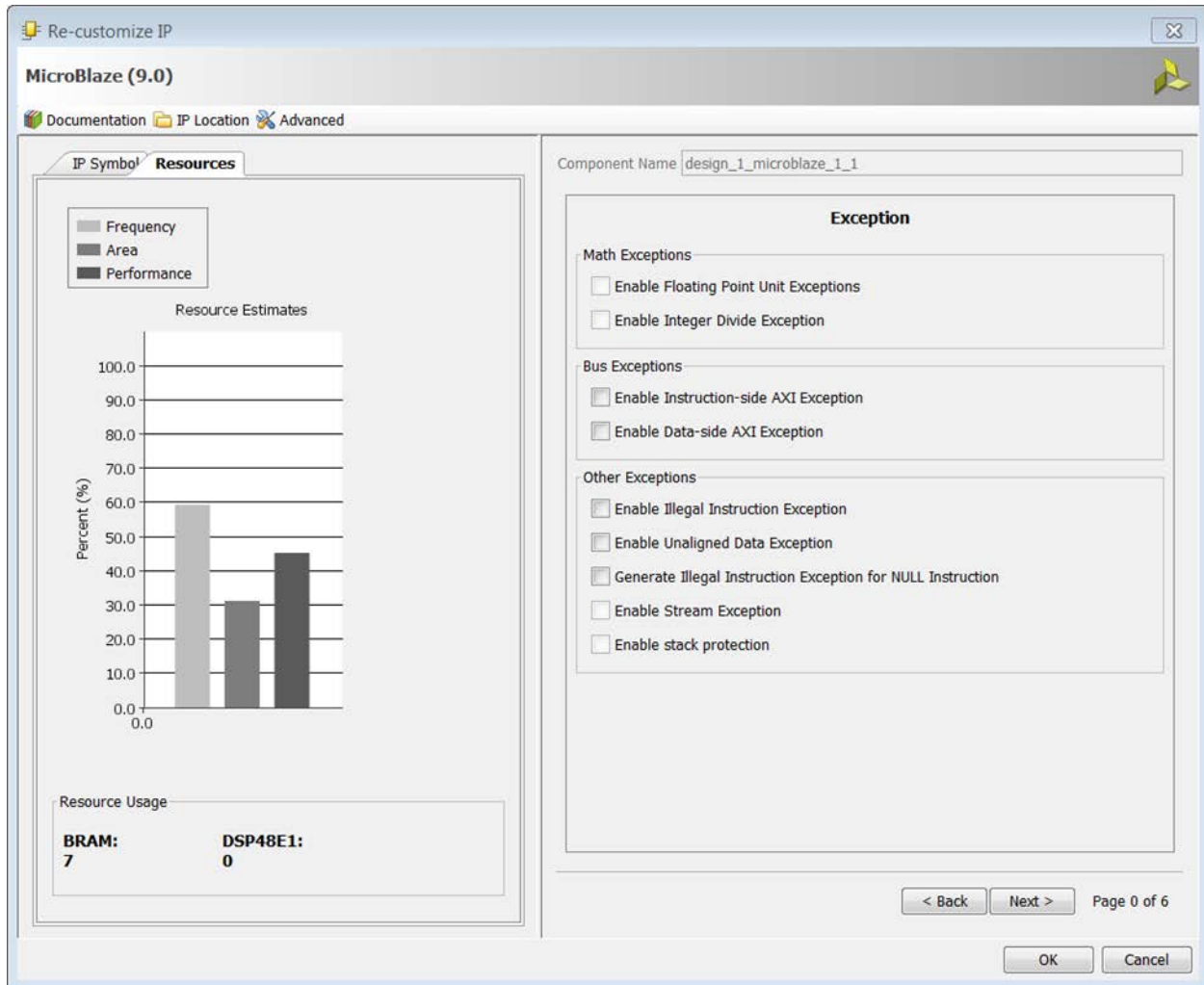


Figure 48: Exception Options in the MicroBlaze Configuration Wizard



IMPORTANT: You must provide your own exception handler.

Math Exceptions

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

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

Bus Exceptions

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

Other Exceptions

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

If `C_OPCODE_0x_ILLEGAL` is set, MicroBlaze traps this condition; otherwise, it treats the command as a `NOE`. This setting is only available if you have enabled Illegal Instruction Exception.

- **Enable Stream Exception:** Enables stream exception handling for Advanced eXtensible Interface (AXI) read accesses. You must enable additional stream instructions to use stream exception handling.
- **Enable Stack Protection:** Ensures that memory accesses using the stack pointer ($R1$) to ensure they are within the limits set by the Stack Low Register (SLR) and Stack High Register (SHR). If the check fails with exceptions enabled, a Stack Protection Violation exception occurs. The Xilinx Microprocessor Debugger (XMD) also reports if the check fails.

MicroBlaze Configuration Wizard Cache Page

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

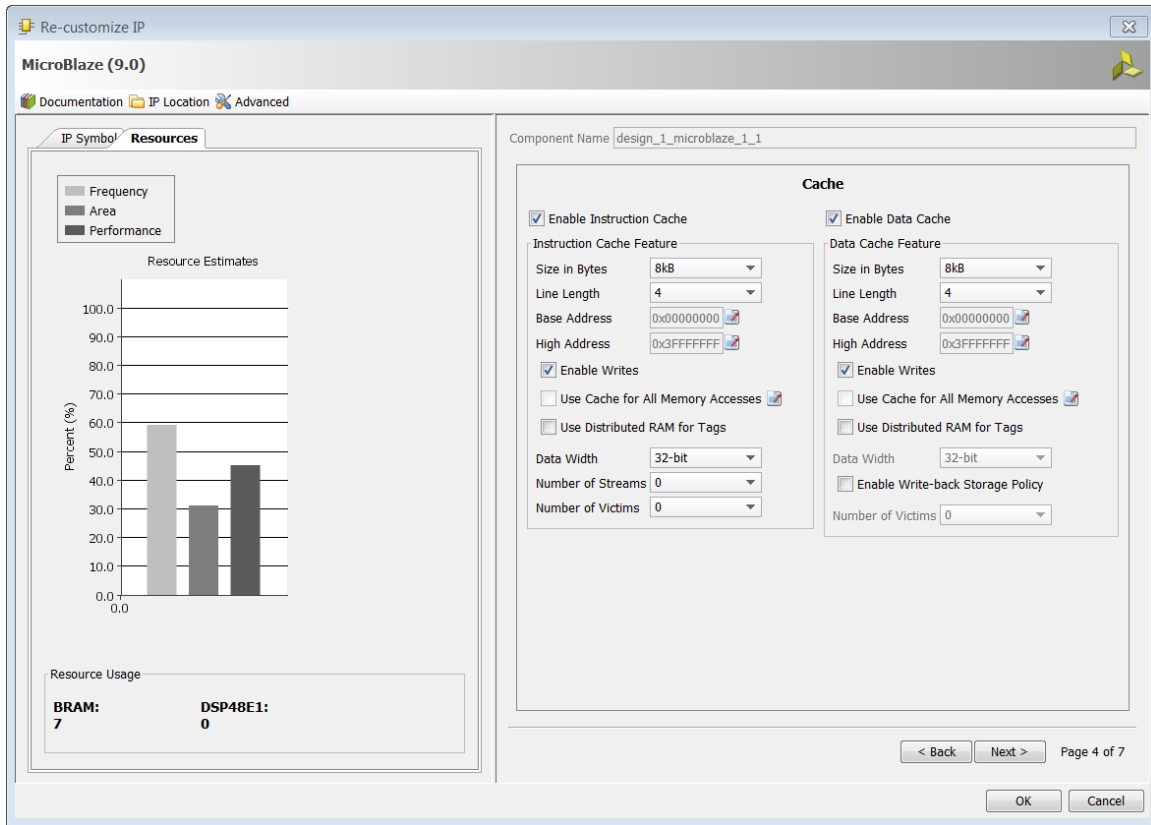


Figure 49: Cache Options Page of the MicroBlaze Configuration Wizard

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

Instruction Cache Features:

- **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 or 8 word cache line length for cache miss-transfers from external instruction memory.
- **Base Address:** Specifies the base address of the instruction cache. This parameter is used only if `C_USE_ICACHE` is enabled.
- **High Address:** Specifies the high address of the instruction cache. This parameter is used only if `C_USE_ICACHE` is enabled.
- **Enable Writes:** When enabled, one can invalidate instruction cache lines with the `wic` instruction. This parameter is used only if `C_USE_ICACHE` is enabled.

- **Use Cache for All Memory Accesses:** When enabled, uses the dedicated cache interface on MicroBlaze is for all accesses within the cacheable range to external instruction memory, even when the instruction cache is disabled.

Otherwise, the instruction cache uses the peripheral AXI for these accesses when the instruction cache is disabled. When enabled, an external memory controller must provide only a cache interface MicroBlaze instruction memory. Enable this parameter when using AXI Coherency Extension (ACE).

- **Use Distributed RAM for Tags:** Uses the instruction cache tags to hold the address and a valid bit for each cacheline. 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 Cacheline:** A single transfer is performed for each cache line, with data width 128 or 256 bits depending on cache line length
 - **512-bit:** Performs a single transfer, but only 128 or 256 bits are used depending on cacheline length.

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

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

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

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

- **Number of Victims:** Specifies the number of instruction cache victims to save. A victim is a cacheline 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.

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

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

MicroBlaze Configuration Wizard MMU Page

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

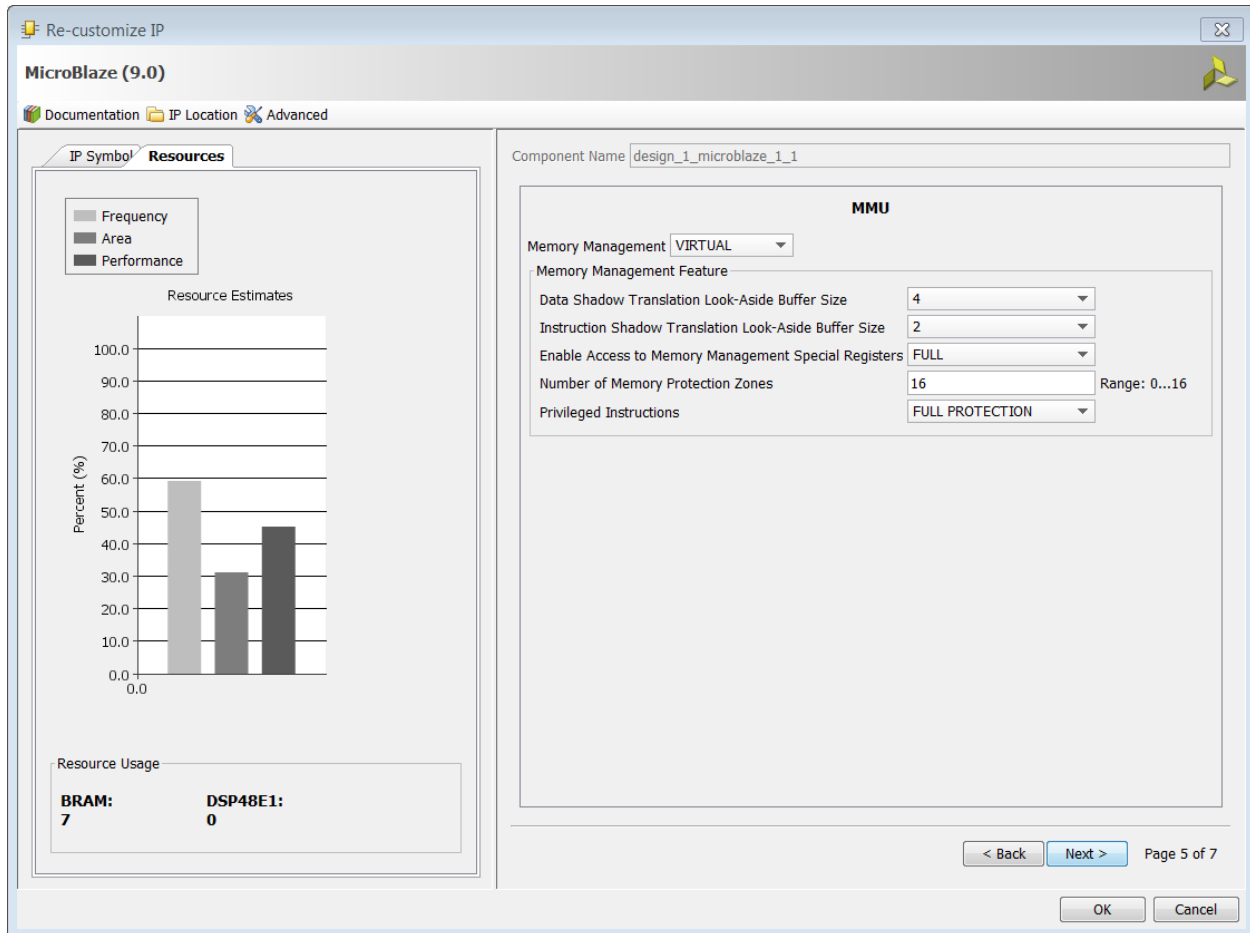


Figure 50: MicroBlaze Configuration Wizard MMU Page

Memory Management: Specifies the Memory Management Unit (MMU) implementation.

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

When `Usermode` is set, it enables the Privileged Instruction exception. When `Protection` or `Virtual` is set, it enables the Privileged Instruction exception and the four MMU exceptions (Data Storage, Instruction Storage, Data TLB Miss, and Instruction TLB Miss).

Memory Management Features:

- **Data Shadow Translation Look-Aside Buffer Size:** Defines the size of the instruction shadow Translation Look-Aside Buffer (TLB). This TLB caches data address translation information, to improve performance of the translation. The selection is a trade-off between smaller size and better performance: the default value is 4.
- **Instruction Shadow Translation Look-Aside Buffer Size:** Defines the size of the instruction shadow Translation Look-Aside Buffer (TLB). This TLB caches instruction address translation information to improve performance of the translation. The selection is a trade-off between smaller size and better performance: the default value is 2.
- **Enable Access to Memory Management Special Registers:** Enables access to the Memory Management Special Register using the MFS and MTS instructions:
 - `Minimal (0)` only allows writing `TLBLO`, `TLBHI`, and `TLBX`.
 - `Read (1)` adds reading to `TLBLO`, `TLBHI`, `TLBX`, `PID`, and `ZPR`.
 - `Write (2)` allows writing all registers, and reading `TLBX`.
 - `Full (3)` adds reading of `TLBLO`, `TLBHI`, `TLBX`, `PID`, and `ZPR`.

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

- **Number of Memory Protection Zones:** Specifies the number of memory protection zones to implement. In many cases memory management software does not use all available zones. For example, the Linux Memory Management code only uses two zones. In this case, it is safe to reduce the number of implemented zones, to save area.
- **Privileged Instructions:** Specifies which instructions to allow in User Mode.
 - The `Full Protection (0)` setting ensures full protection between processes.
 - The `Allow Stream Instructions (1)` setting makes it possible to use AXI4-Stream instructions in User Mode.



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

MicroBlaze Configuration Wizard Debug Page

The following figure shows the MicroBlaze Configuration Wizard Debug page.

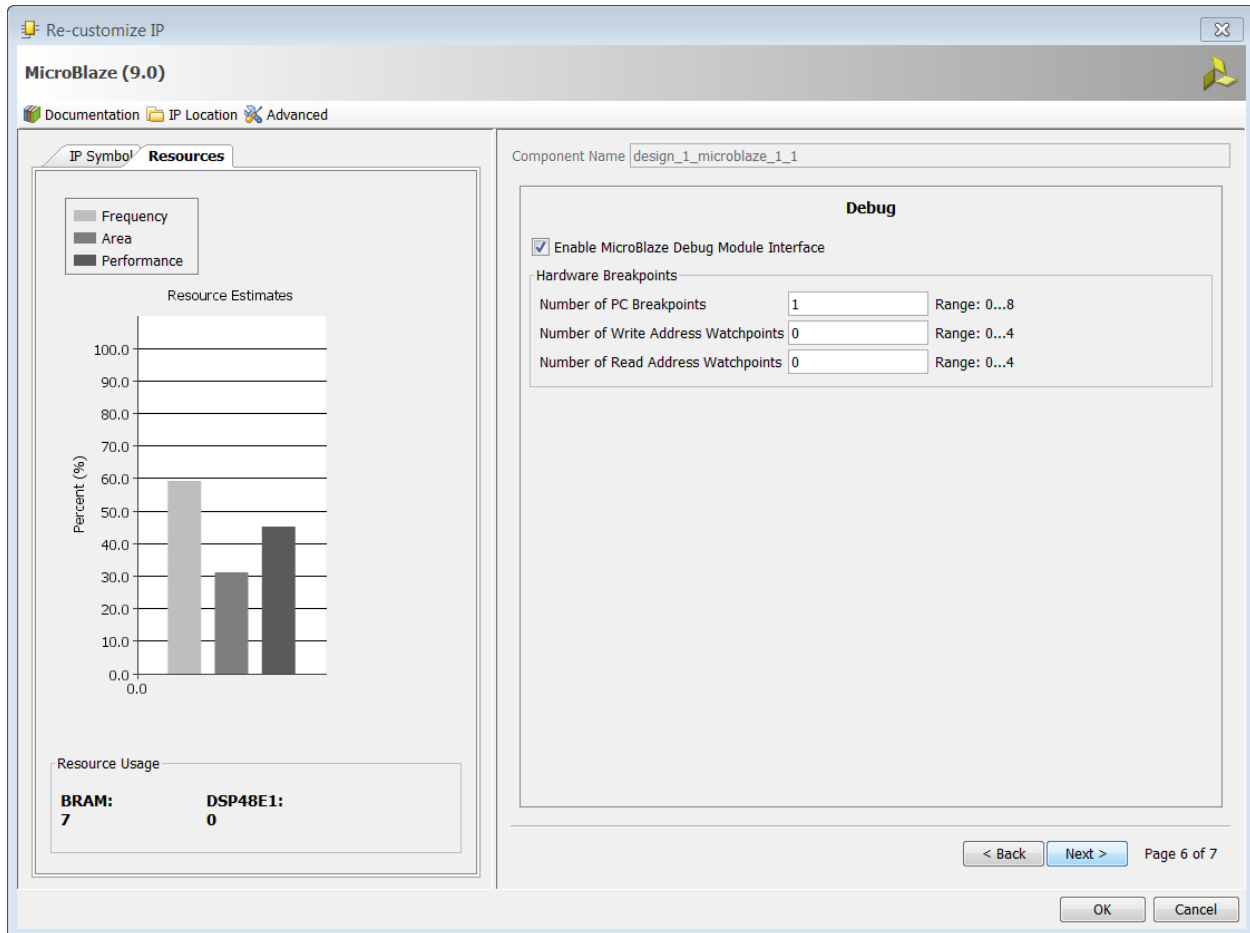


Figure 51: MicroBlaze Configuration Wizard Debug Page

Debug Options

Enable Microblaze Debug Module Interface: Enables the MicroBlaze Debug Module (MDM) interface to MicroBlaze for debugging. With this option, you can use Xilinx Microprocessor Debugger (XMD) to debug the processor over the Joint Test Action Group (JTAG) boundary-scan interface. You can disable this option after you finish debugging to reduce the size of MicroBlaze.

Hardware Breakpoints:

- Number of PC Breakpoints:** Specifies the number of program counter (PC) hardware breakpoints for debugging. This parameter controls the number of hardware breakpoints Xilinx Microprocessor Debugger (XMD) can set. This option only has meaning if `C_DEBUG_ENABLED` is on. The MicroBlaze processor takes a noticeable frequency hit the larger this parameter is set.

- Number of Write Address Watchpoints:** Specifies the number of write address breakpoints for debugging. This parameter controls the number of write watchpoints Xilinx Microprocessor Debugger (XMD) can set. This option only has meaning if C_DEBUG_ENABLED is on. MicroBlaze take a noticeable frequency hit, the larger this parameter is set.
- Number of Read Address Watchpoints:** Specifies the number of read address breakpoints for debugging. This parameter controls the number of read watch points Xilinx Microprocessor Debugger (XMD) can set. This option only has meaning if C_DEBUG_ENABLED is on. The MicroBlaze processor takes a noticeable frequency hit the larger this parameter is set.



RECOMMENDED: *It is recommended that these two options be set to 0 if you are not using watch points for debugging*

MicroBlaze Configuration Wizard Buses Page

The following figure shows the Buses options page of the MicroBlaze configuration wizard.

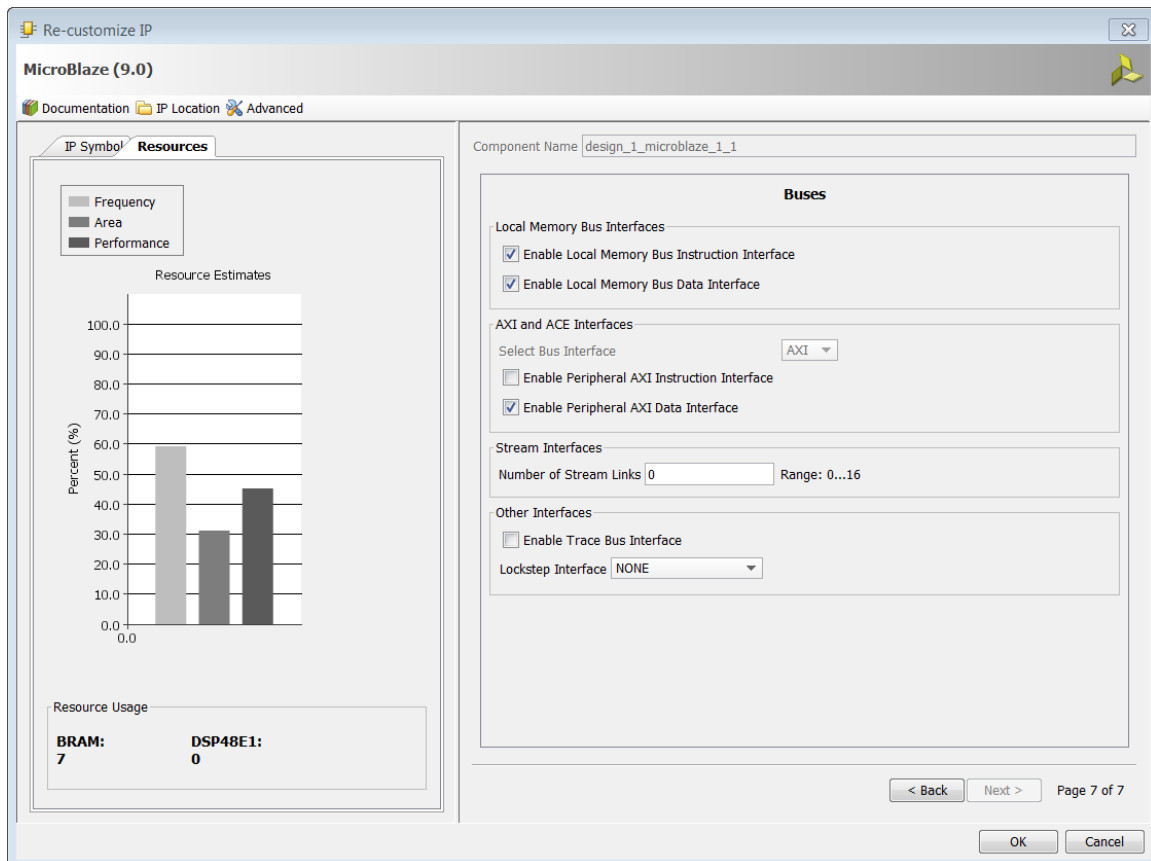


Figure 52: MicroBlaze Configuration Wizard Buses Page

Local Memory Bus Interfaces:

- **Enable Local Memory Bus Instruction Interface:** Enables LMB instruction interface. When this instruction is set, the Local Memory Bus (LMB) instruction interface is available. A typical MicroBlaze system uses this interface to provide fast local memory for instructions. Normally, it connects to an LMB bus using an LMB Bus Interface Controller to access a common Block RAM.
- **Enable Local Memory Bus Data Interface:** Enables LMB data interface. When this parameter is set, the Local Memory Bus (LMB) data interface is available. A typical MicroBlaze system uses this interface to provide fast local memory for data and vectors. Normally, it connects to an LMB bus using an LMB Bus Interface Controller to access a common Block RAM.

AXI and ACE Interfaces:

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

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

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

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 coprocessor. 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.

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

Custom Logic

The Vivado™ IP packager lets you and third party IP developers use the Vivado IDE to easily prepare an Intellectual Property (IP) design for use in the Vivado IP catalog.

The IP user can then instantiate this third party IP into their design in the Vivado Design Suite.

When IP developers use the Vivado Design Suite IP packaging flow, the IP user has a consistent experience whether using Xilinx IP, third party IP, or customer-developed IP within the Vivado Design Suite.

IP developers can use the IP packager feature to package IP files and associated data into a ZIP file. The IP user receives this generated ZIP file, installs the IP into the Vivado Design Suite IP Catalog. The IP user then customizes the IP through parameter selections and generates an instance of the IP.



RECOMMENDED: *To verify the proper packaging of the IP before handing it off to the IP user, Xilinx® recommends that the IP developer run each IP module completely through the IP user flow to validate that the IP is ready for use.*

Embedded IP Catalog

The Vivado IDE 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 Catalog.

The screenshot shows a search bar at the top with a magnifying glass icon. Below it is a table with two columns: 'Name' and 'Version'. The table lists various IP cores such as 'AHB-Lite to AXI Bridge', 'AXI-Stream FIFO', 'AXI4-Stream to Video Out', etc., along with their respective version numbers. A vertical scrollbar is visible on the right side of the table.

Name	Version
AHB-Lite to AXI Bridge	2.0
AXI-Stream FIFO	4.0
AXI4-Stream to Video Out	3.0
AXI AHB-Lite Bridge	2.0
AXI APB Bridge	2.0
AXI BFM Cores	4.0
AXI BRAM Controller	3.0
AXI CAN	5.0
AXI Central Direct Memory Access	4.0
AXI Chip2Chip Bridge	4.0
AXI Clock Converter	2.0
AXI Crossbar	2.0
AXI Data FIFO	2.0
AXI DataMover	5.0
AXI Data Width Converter	2.0
AXI Direct Memory Access	7.0
AXI EMC	2.0
AXI EPC	2.0
AXI Ethernet	4.0
AXI Ethernet Buffer	1.0
AXI Ethernet Clocking	1.0
AXI EthernetLite	2.0
AXI GPIO	2.0
AXI HWICAP	3.0
AXI IIC	2.0
AXI INTC	3.0
AXI Interconnect	2.0
AXI Memory Mapped To PCI Express	2.0
AXI Performance Monitor	4.0
AXI Protocol Checker	1.0
AXI Protocol Converter	2.0
AXI Quad SPI	3.0
AXI Register Slice	2.0
AXI TFT Controller	2.0
AXI Timebase Watchdog Timer	2.0
AXI TIMER	2.0
AXI UART 16550	2.0

Figure 53: IP-Integrator IP Catalog

Completing Connections

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

In the IP Integrator canvas, right-click and select **Add IP**.

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

Block Automation

The Block Automation feature is available when a microprocessor such as the ZYNQ7 Processing System or the MicroBlaze Processor is instantiated in the block diagram of the IP integrator tool.

Click **Run Block Automation** to get assistance with putting together a simple MicroBlaze System as shown in the following figure.

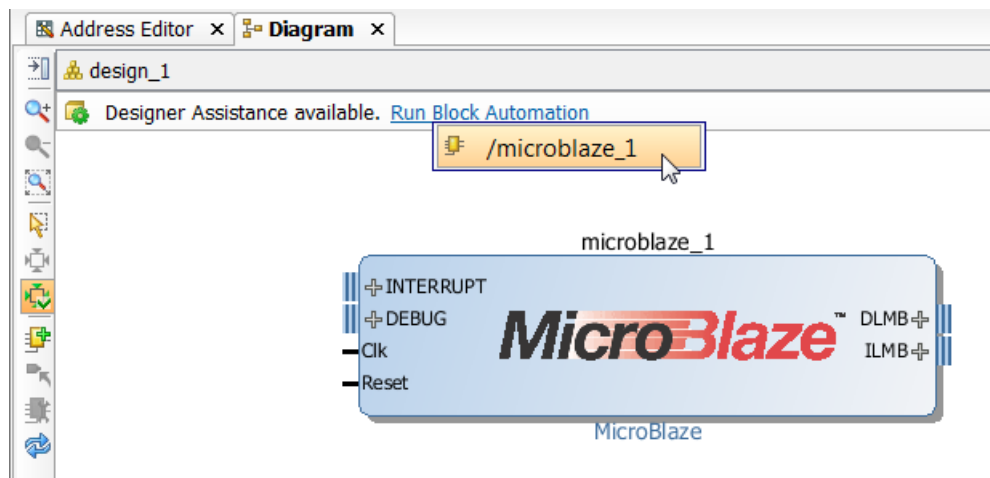


Figure 54: Run Block Automation Feature

The Run Block Automation dialog box lets you provide input about basic features that the microprocessor system needs.

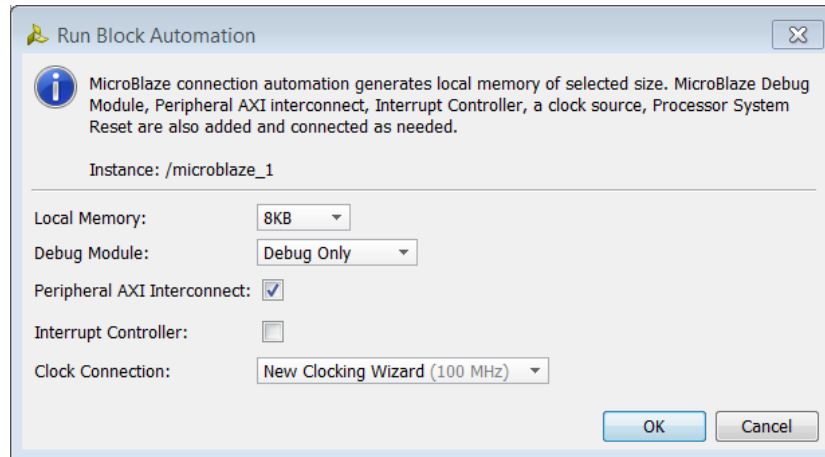


Figure 55: Run Block Automation for MicroBlaze Processor Dialog Box

After selecting the desired options and clicking **OK**, the Run Block Automation feature creates the following MicroBlaze system.

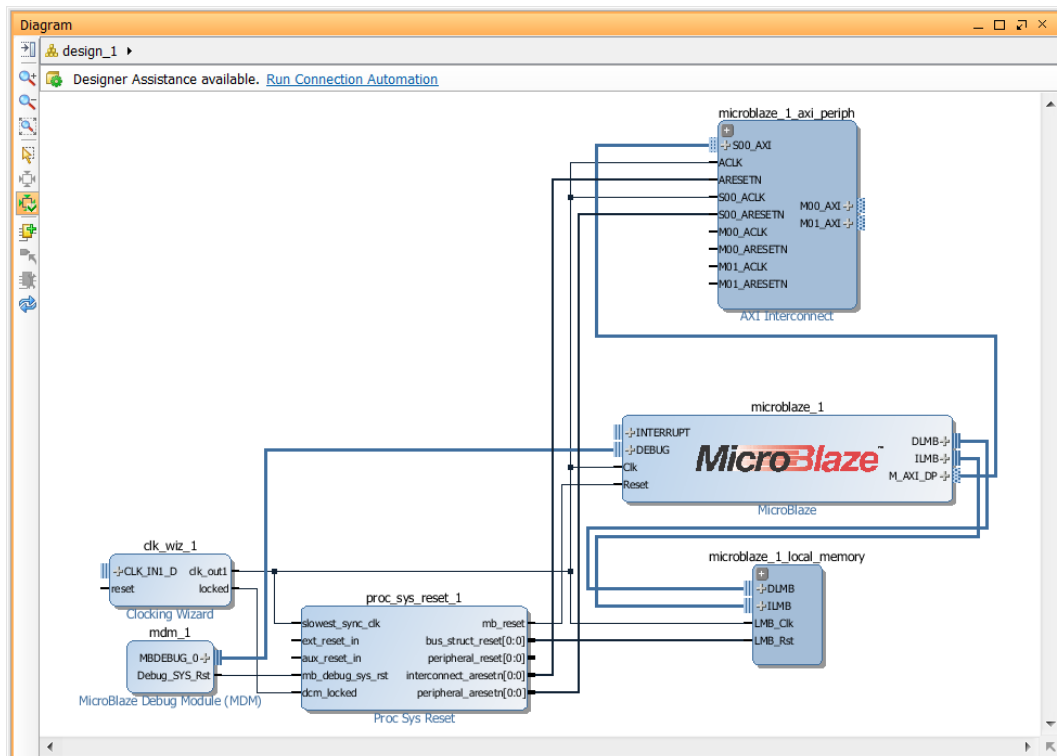


Figure 56: IP Integrator Window after running Block Automation

Using Connection Automation

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

In the following figure, two IP: the GPIO and the Uartlite, are instantiated along with the MicroBlaze sub-system. 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.

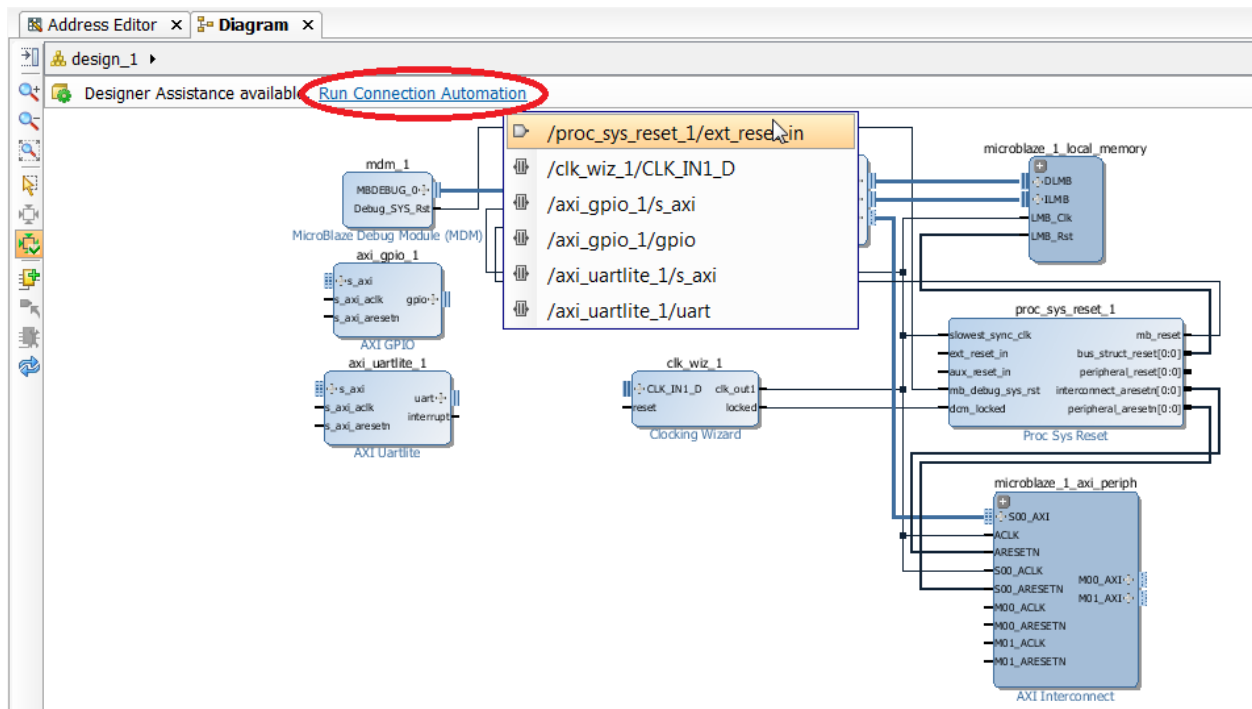


Figure 57: Connection Automation Feature of IP Integrator

When you run connection automation on each of those available options the block diagram looks like the following figure.

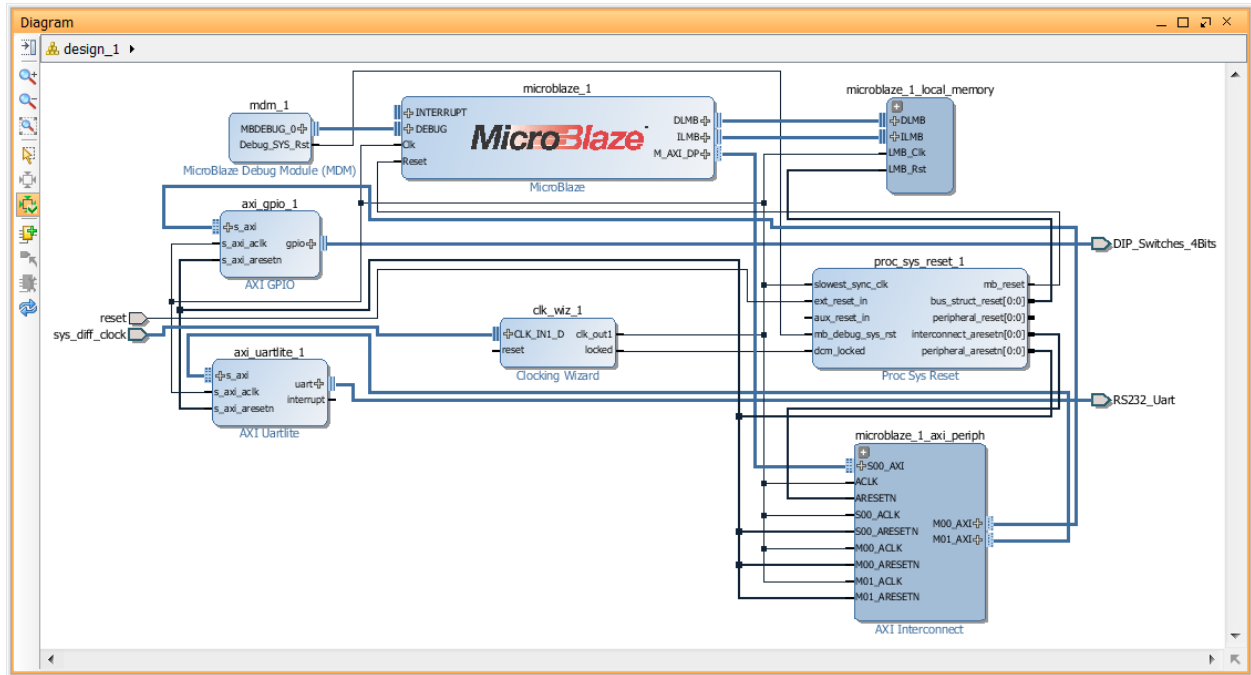


Figure 58: Running Connection Automation on MicroBlaze Design

Using Board Automation

The Vivado IDE IP Integrator tool also provides a Board Automation feature when using a Xilinx Target Reference Platform, such as the KC705.

This feature provides connectivity of the ports of IP to the FPGA pins on the target board. The IP configures accordingly, and, based upon your selections connects the I/O ports. The Board Automation features also generates the physical constraints for those IP that require physical constraints. These features are available to you via the Run Connection Automation feature of the IP integrator tool.

As an example, for the GPIO core, when the target board is KC705, the GPIO interface can connect to one of the following:

- DIP Switches
- LEDs
- Push Buttons

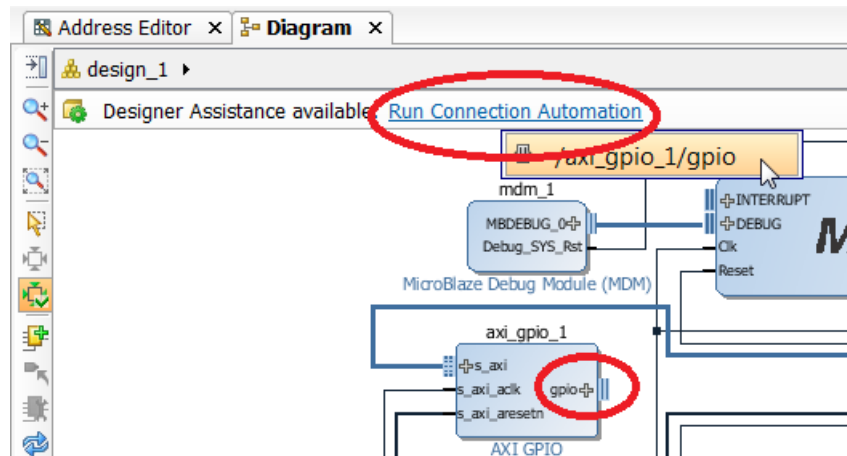


Figure 59: Board Automation on GPIO port

Running connection automation for the `gpio` interface gives you the following options in the dropdown menu.

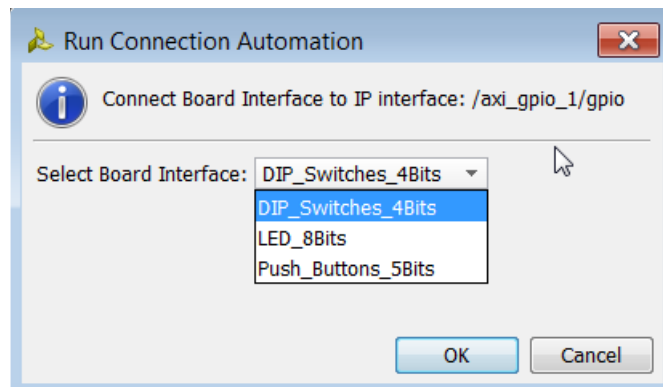


Figure 60: Run Connection Automation Options for gpio

Depending on the selected option, the GPIO IP is not only configured accordingly (as inputs or outputs) but the appropriate sets of physical constraints are also generated.

Manual Connections in an IPI Design

See [Manual Connections in a Design](#).

Manually Creating and Connecting to I/O Ports

See [Manually Creating and Connecting to I/O Ports](#).

Memory Mapping in Address Editor

See [Memory Mapping in Address Editor](#).

Running Design Rule Checks

See [Running Design Rule Checks](#).

Integrating a Block Diagram in the Top-Level Design

See [Integrating a Block Diagram in the Top-Level Design](#).

MicroBlaze Processor Constraints

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

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

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

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



IMPORTANT: *The Vivado Design Suite does not support UCF format. For information on migrating UCF constraints to XDC commands refer to the Vivado Design Suite Migration Methodology Guide [\(UG911\)](#) for more information.*

You have a number of options on how to use constraint sets. You can have:

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

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

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

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



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

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

Taking the Design through Synthesis, Implementation and Bitstream Generation

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

Exporting Hardware to the Software Development Kit (SDK)

See [Using the Software Development Kit \(SDK\)](#) for more information.

In general, after you generate the bitstream for your design, you are ready to export your hardware definition to SDK.

Select **File > Export > Export Hardware for SDK**, as shown in the following figure



Figure 61: Exporting Hardware for SDK

This launches the Export Hardware for SDK dialog box, where you can choose the available export options.

You can export the hardware definition and the bitstream, and launch SDK through the Export Hardware for SDK dialog box.

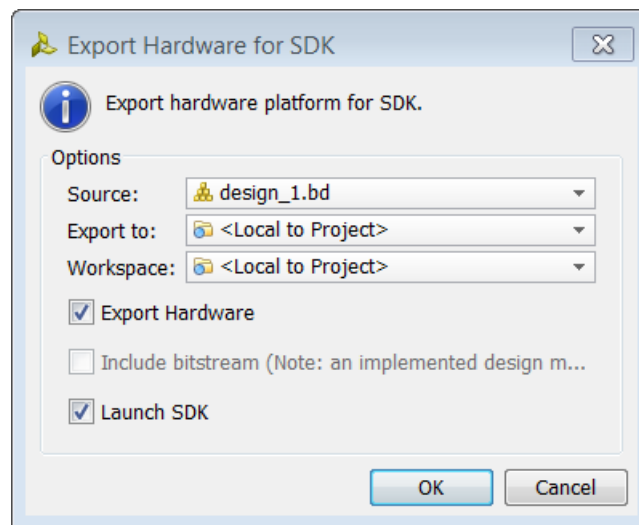


Figure 62: Export Hardware for SDK dialog box

After you export the hardware definition to SDK, and launch SDK, you can start writing your software application. Also, you can perform more debug and software from SDK.

Alternatively, you can import the software ELF file back into a Vivado IDE project, and integrate that file with an FPGA bitstream for further download and testing.