XILINX®

Collaborative and Accelerated Design with Vivado IP Integrator

Amir Zeineddini

Agenda

- Introduction
- Designing with the IP Integrator
- Building a Custom IP
- Collaborative Features
- Building a Versal Design
- Summary

What is Vivado IP Integrator?

- IP integration tool for creating complex platform designs

Interactive design environment	• GUI & script flow support • Cross-probing and error correction in design views • Collaboration support for modular team design
Suitable to create complex systems	• Simplifies AXI-based designs • Integration of subsystems (RTL, HLS, …) • Domain support: embedded, DSP, connectivity, analog, and logic IPs
Ease-of-use at every level	• Revision control and packaging support for re-use • System-level optimization • Designer assistance, connection automation, parameter propagation

What are the benefits of Vivado IP Integrator?

Increase Productivity & Decrease Complexity

- **Automation** at various levels by being device- and platform-aware
- Higher level of **abstraction** by enabling interface level connections
- **Re-usability** of IP and subsystems by leveraging AXI standardization

Accelerated Design with IP Integrator

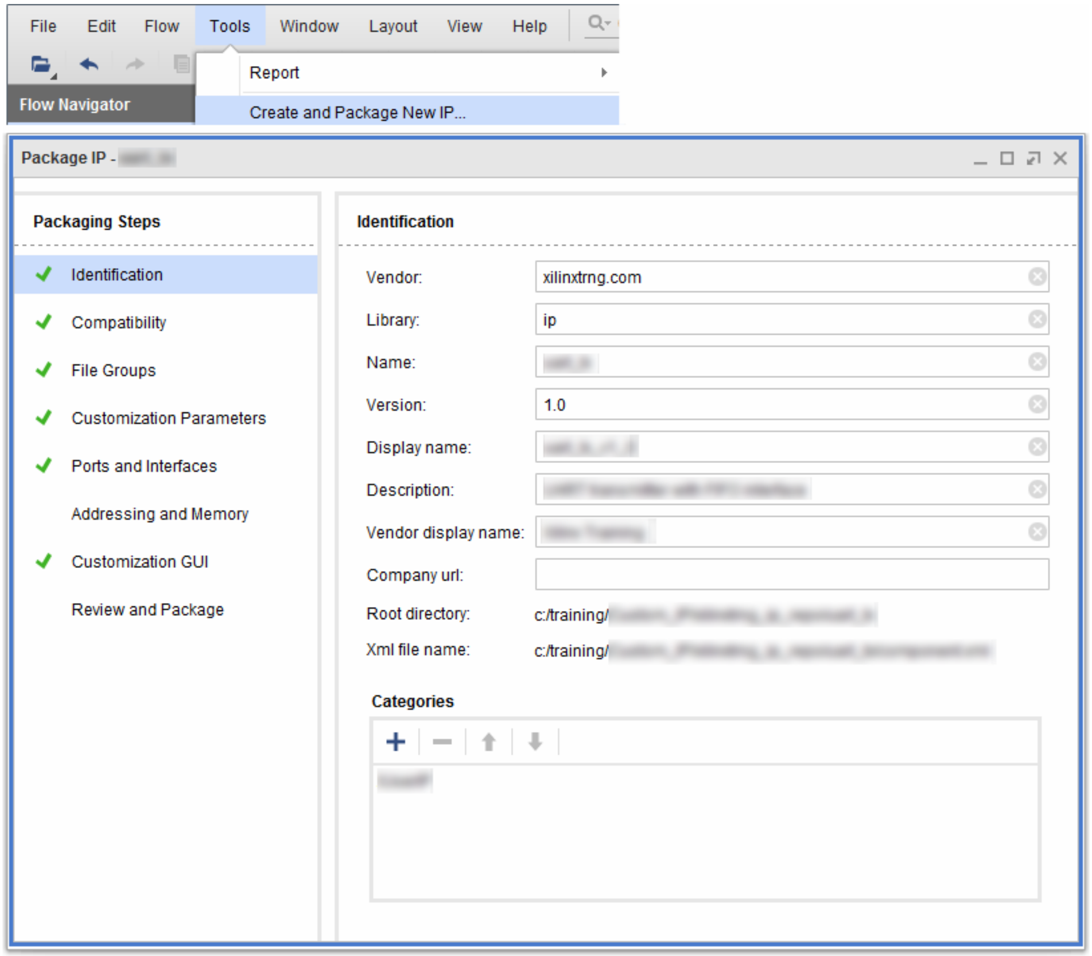
Designing with IP Integrator

- Creating a Block Design
- Use Designer Assistance
- Interface-level or net connections
- Using Connection Automation
- Canvas Toolbar
- Address Editing
- Design Validation
- Parameter Propagation
- Debug
- Tcl Support

Creating and Packaging Custom IP

IP Packager

- The IP Packager is based on the IP-XACT (IEEE-1685) standard
- Converts your design into reusable IP
- Packaging Flow
  - Create a Vivado user design to package into an IP
  - Add the IP to Vivado IP catalog to share among the design team
    1. Unzip the IP to a local directory
    2. Add the directory to the IP repository of the IP catalog



Package IP - [redacted]

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- Addressing and Memory
- ✓ Customization GUI
- Review and Package

Identification

Vendor:	xilinxtrng.com
Library:	ip
Name:	[redacted]
Version:	1.0
Display name:	[redacted]
Description:	[redacted]
Vendor display name:	[redacted]
Company url:	
Root directory:	c:/training/[redacted]
Xml file name:	c:/training/[redacted]

Categories

[redacted]

Custom IP Packaging Files

IP Packaging Inputs

Basic Inputs

Design Sources (IP)
- Verilog/VHDL/System Verilog RTL
- Netlists
- IP instances
- IP Block Designs

Inferred or Specified
- Ports
- Interfaces
- Parameters

Customization GUI Options

+

Optional Inputs

Example Design (Instantiates the IP Design)
- Verilog/VHDL/System Verilog
- Netlists
- IP instances

Implementation Files
- XDC contraints

Simulation Sources
- Verilog/VHDL/System Verilog Models
- Test Benches

Software Related
- Driver Source Code

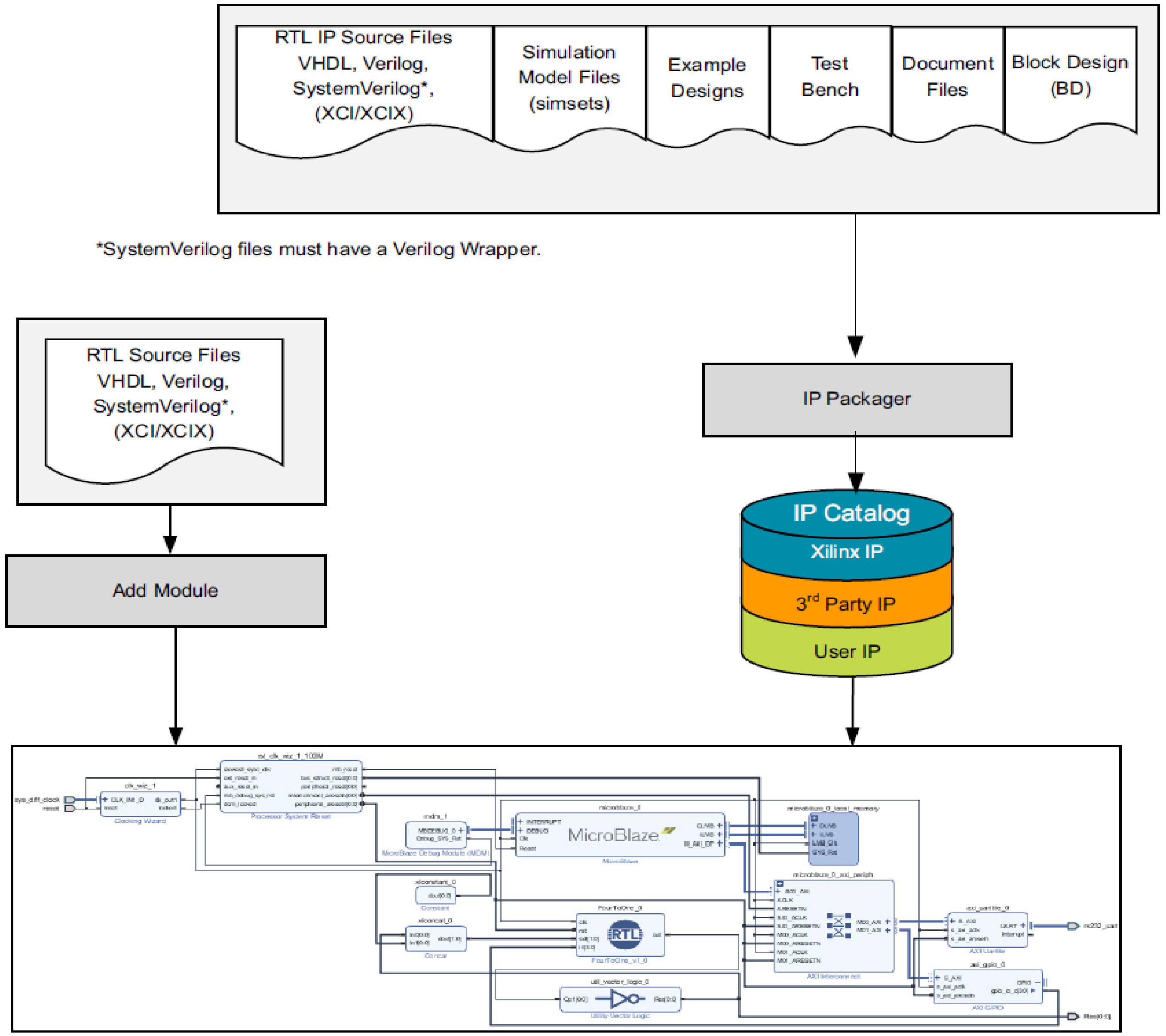
Documentation
- User Guide
- Getting Started

=

IP Packaging Outputs

IP-XACT Definition File
- component.xml
- XGUI customization script
- All the input files are copied to the package

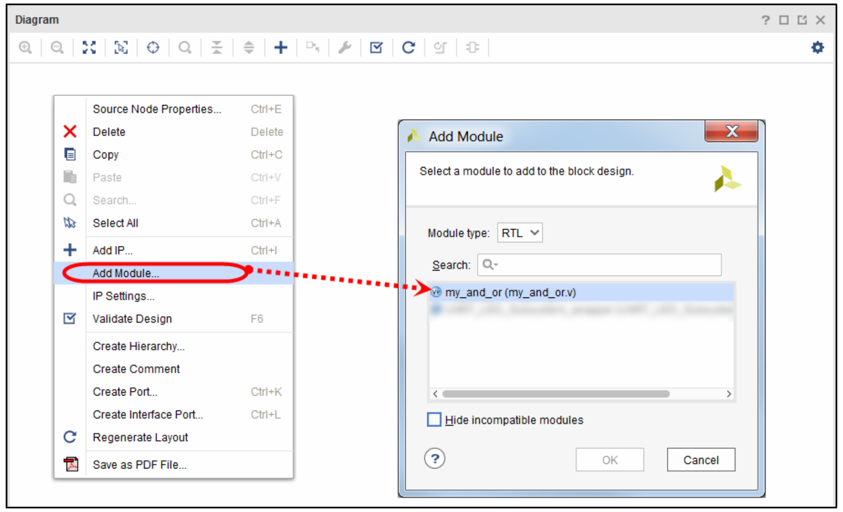
IP Packaging and Usage Flow


RTL IP Source Files VHDL, Verilog, SystemVerilog*, (XCI/XCIX)
Simulation Model Files (simsets)
Example Designs
Test Bench
Document Files
Block Design (BD)
IP Packager
IP Catalog
Xilinx IP
3rd Party IP
User IP
RTL Source Files VHDL, Verilog, SystemVerilog*, (XCI/XCIX)
Add Module
MicroBlaze
RTL

*SystemVerilog files must have a Verilog Wrapper.

Module Referencing

- RTL file needs to be added to the project
- The Add Module dialog box lets you add a module in the current block design
- It displays the modules that are available to add to the block design
- If the entity/module name changes in the source RTL file, the referenced module instance must be deleted from the block design and a new module should be added



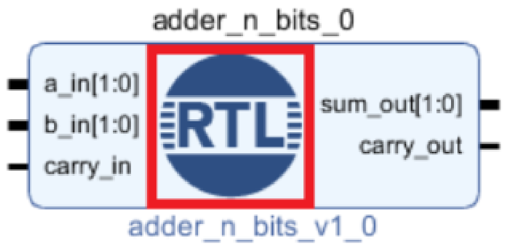
Select a module to add to the block design.

Module type: RTL

Search:

my_and_or (my_and_or.v)

Hide incompatible modules

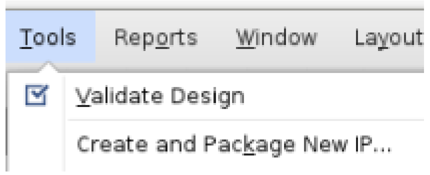


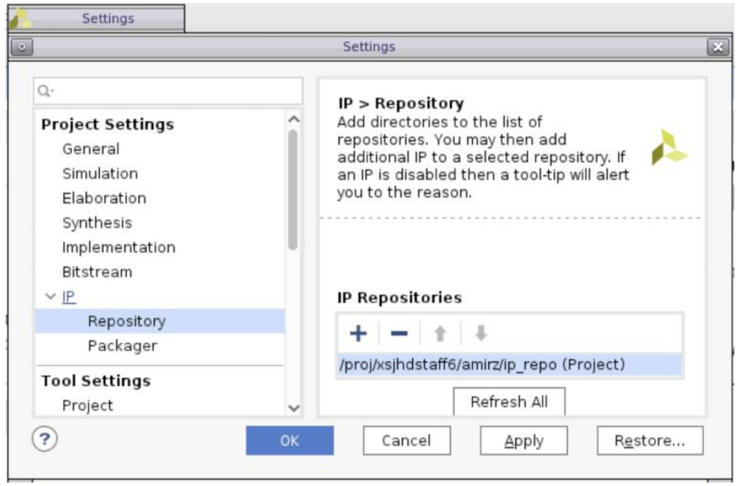
Managing Remote IP Repository and Cache

- User IP repositories allow users to add their own IP to the Vivado IP catalog
- Significant reduction of compile time when used alongside a Remote IP Cache
- Steps:
  1. Create the following directory structure:

```
/iprepo
  /<IP 1 Name>
  /<IP 2 Name>
  /ipcache
```

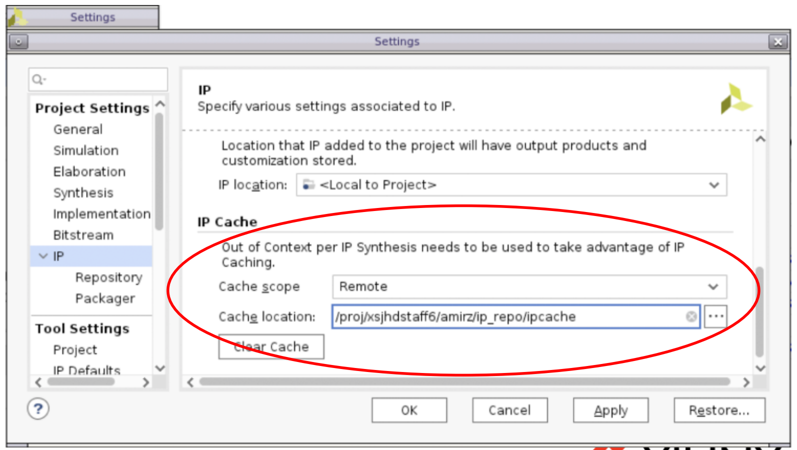
  2. Package IP & point to remote IP repository and cache locations





IP > Repository
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories
/proj/xsjhdstaff6/amirz/ip_repo (Project)



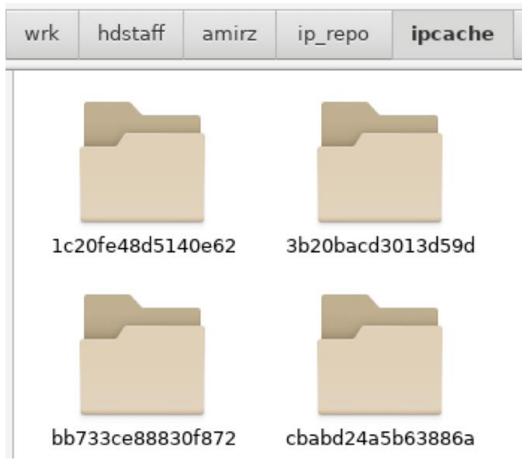
IP
Specify various settings associated to IP.

Location that IP added to the project will have output products and customization stored.
IP location: <Local to Project>

IP Cache
Out of Context per IP Synthesis needs to be used to take advantage of IP Caching.
Cache scope: Remote
Cache location: /proj/xsjhdstaff6/amirz/ip_repo/ipcache

Managing Remote IP Repository and Cache

3. Add the IP to the BD in IPI
   - If IP is configurable, add multiple configurations to further populate the IP Cache with common configurations
4. Generate output products in Out-Of-Context (OOC) per IP
5. After generation, cache folder is populated for each IP
   - Each IP gets a hash code as the directory name
6. When generating the design, re-synthesis will not occur if part/board & IP configuration options are not changed
7. In other projects, just point to the top-level IP repository to use both the User IP and the Remote IP Cache.

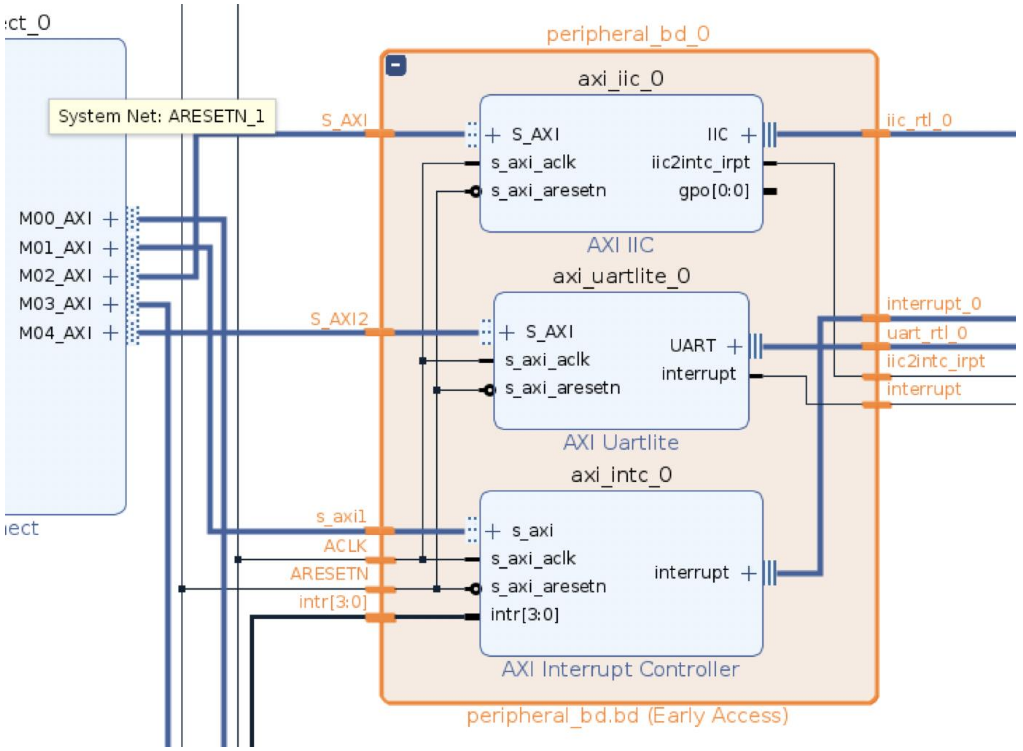


Collaborative Design in IPI

Block Design Container

- Ability to instantiate/create one Block Design inside another
  - Enables Modular Designing for Reusability
  - Allows Team Based Designs
  - Enables DFX Flow

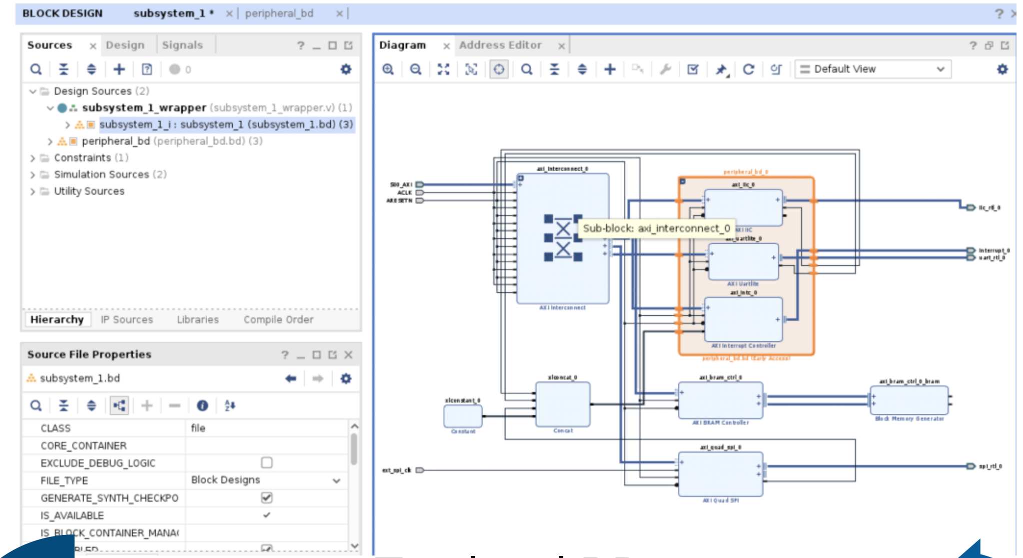
- Overcomes limitations in the current solutions
  - Parameter Propagation
  - Addressing Limitations
  - Module Reference



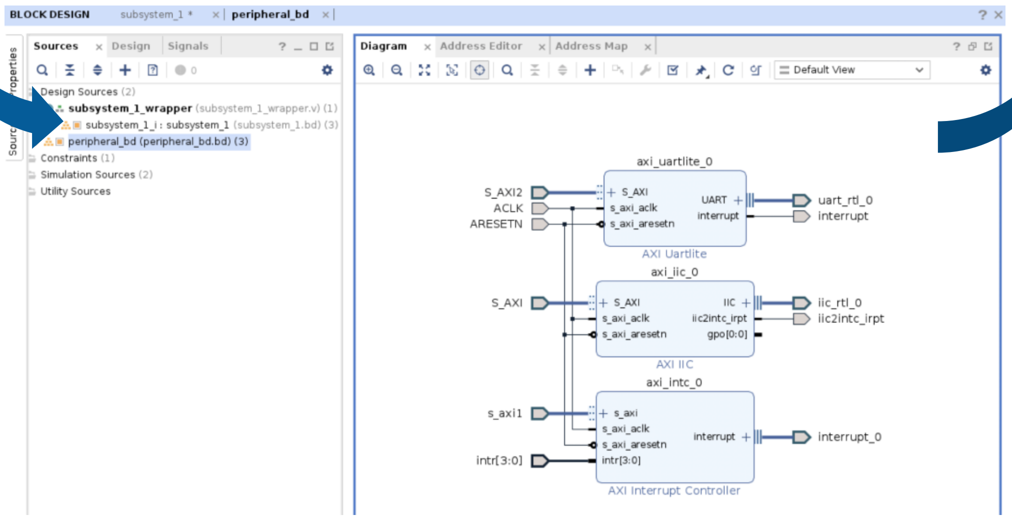
BDC Capabilities

- Supported features from the top-level BD:
  - Connection automation
  - Parameter propagation
    - Can lock boundaries to prevent propagation
  - Addressing view/edit of sub-level BDCs
  - Variants:
    - Ability to specify a variant for synthesis or simulation
  - In-place expansion on canvas

- Change sub-level BD independently
  - Refresh the top-level BD to apply changes



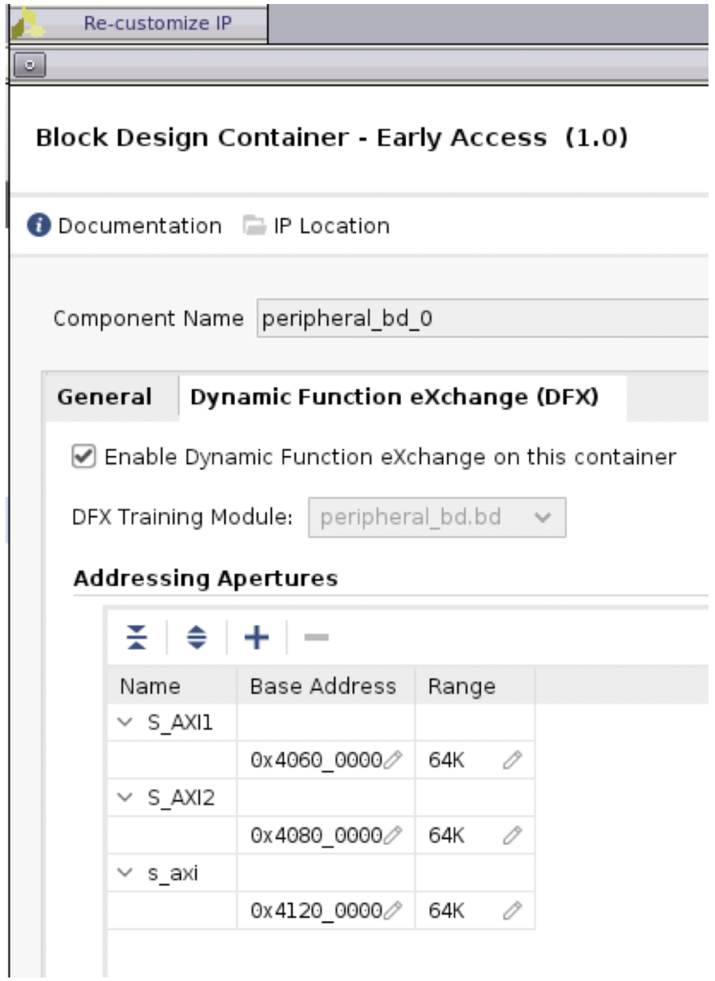
Top-level BD



Sub-level BD

BDC & DFX Flow

- BDC variants can be used as RMs (Reconfigurable Module) in the DFX Flow
- Users can toggle between DFX and non-DFX mode seamlessly
- Variants must keep the same boundary when in DFX mode.



Block Design Container - Early Access (1.0)

Documentation IP Location

Component Name peripheral_bd_0

General | Dynamic Function eXchange (DFX)

[✓] Enable Dynamic Function eXchange on this container

DFX Training Module: peripheral_bd.bd

Addressing Apertures

Name	Base Address	Range
S_AXI1		
	0x4060_0000	64K
S_AXI2		
	0x4080_0000	64K
s_axi		
	0x4120_0000	64K

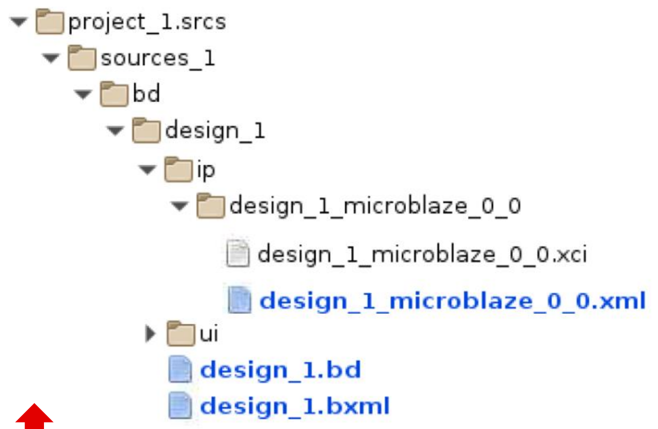
Strategy for Successful Revision Control

- Use Vivado / IPI frameworks to develop your revision control strategy
- Use scripted flows for revision control
- Keep source files in a repository
- Revision control the repository
- Create a Tcl script to recreate the project
- Revision control the script
- Test your scripts

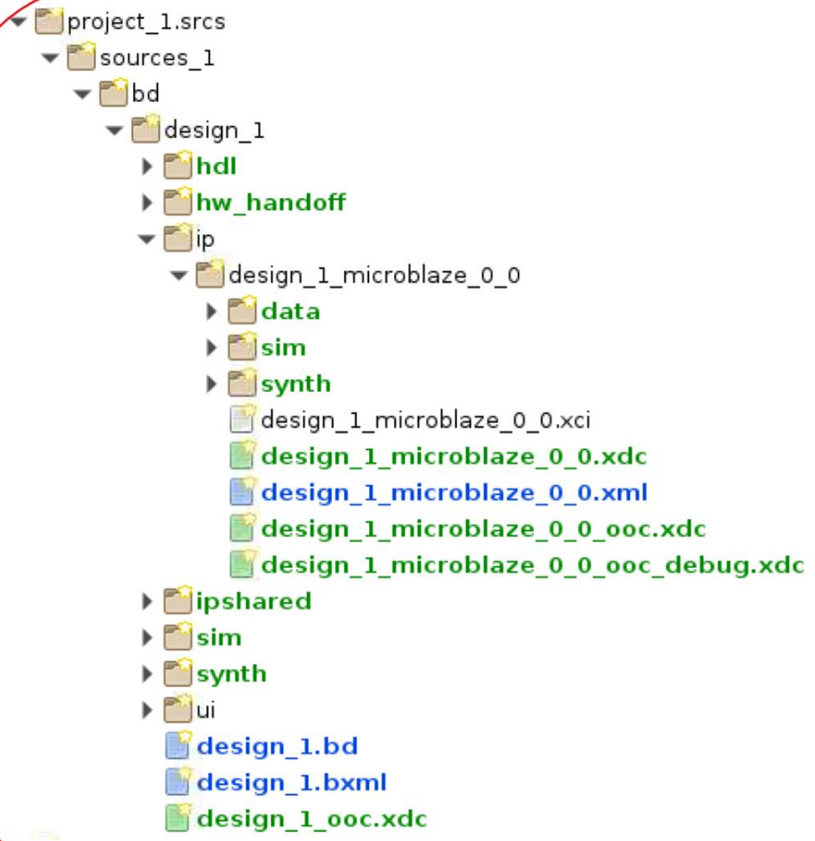
IP & IPI Directory Structure in Previous Releases

2020.1 & older releases generated output products in *<project>.srcs* directory

Before Generate Design	After Generate Design



```
project_1.srcs
  sources_1
    bd
      design_1
        ip
          design_1_microblaze_0_0
            design_1_microblaze_0_0.xci
            design_1_microblaze_0_0.xml
        ui
        design_1.bd
        design_1.bxml
```



```
project_1.srcs
  sources_1
    bd
      design_1
        hdl
        hw_handoff
        ip
          design_1_microblaze_0_0
            data
            sim
            synth
            design_1_microblaze_0_0.xci
            design_1_microblaze_0_0.xdc
            design_1_microblaze_0_0.xml
            design_1_microblaze_0_0_ooc.xdc
            design_1_microblaze_0_0_ooc_debug.xdc
        ipshared
        sim
        synth
        ui
        design_1.bd
        design_1.bxml
        design_1_ooc.xdc
```

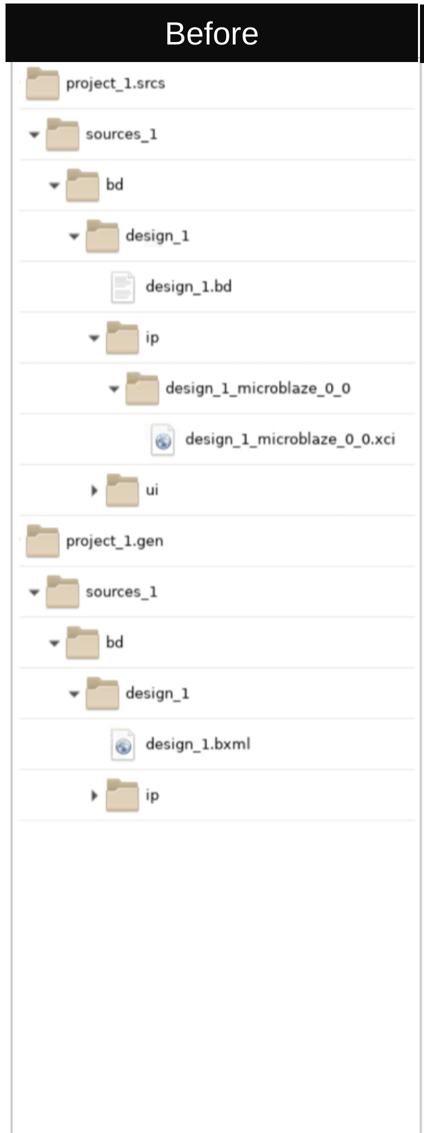
Only **.srcs** directory exists

Source and output files reside side by side

Legend
Green - new files
Blue - modified files

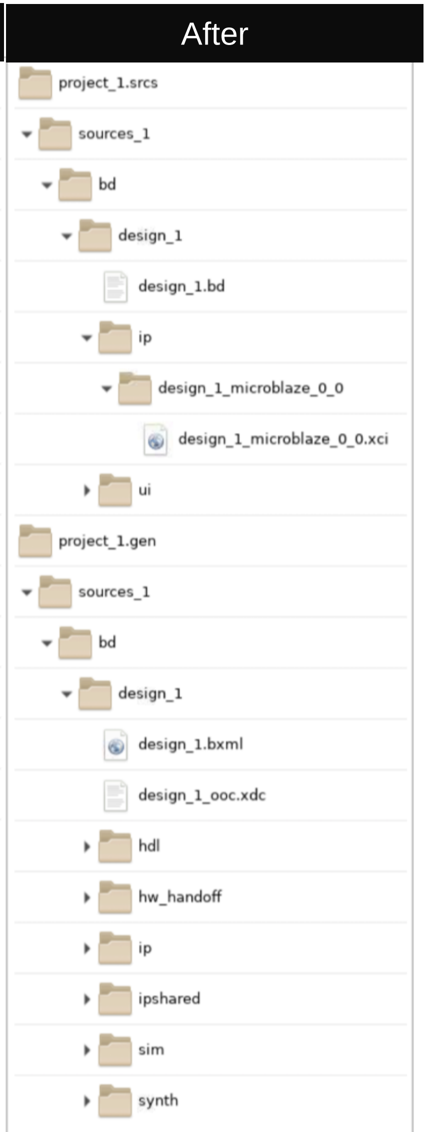
IP & IPI Directory Structure in 2020.2 & Future Releases

2020.2 generates output products in a separate <project>.gen directory than source files



Before

```
project_1.srcs
▾ sources_1
  ▾ bd
    ▾ design_1
        design_1.bd
      ▾ ip
        ▾ design_1_microblaze_0_0
            design_1_microblaze_0_0.xci
      ▸ ui
project_1.gen
▾ sources_1
  ▾ bd
    ▾ design_1
        design_1.bxml
      ▸ ip
```



After

```
project_1.srcs
▾ sources_1
  ▾ bd
    ▾ design_1
        design_1.bd
      ▾ ip
        ▾ design_1_microblaze_0_0
            design_1_microblaze_0_0.xci
      ▸ ui
project_1.gen
▾ sources_1
  ▾ bd
    ▾ design_1
        design_1.bxml
        design_1_ooc.xdc
      ▸ hdl
      ▸ hw_handoff
      ▸ ip
      ▸ ipshared
      ▸ sim
      ▸ synth
```

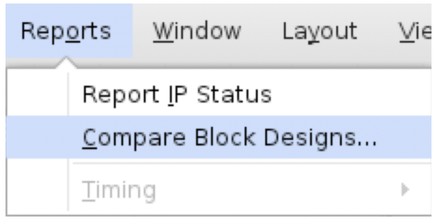
New project.**gen** directory contains all subcore IP and scoped BD files

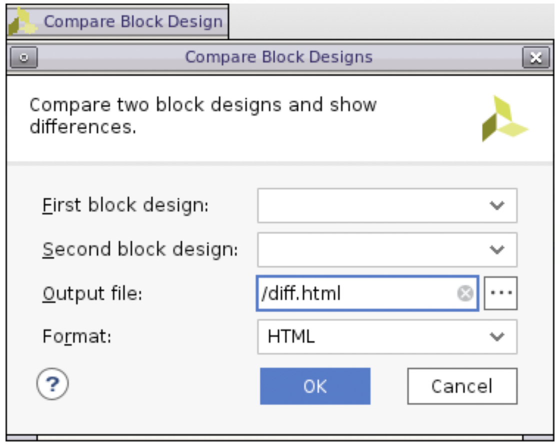
IP & BD source files in **.srcs** directory

Generated outputs in **.gen** directory

Useful Tool for Revision Control

- Diffbd Utility
  - Standalone command to compare two block designs
  - Same as Compare Block Designs… command in GUI
  - Reads two .bd files and generates a diff report
  - Diff report in text (default) or HTML





Compare two block designs and show differences.

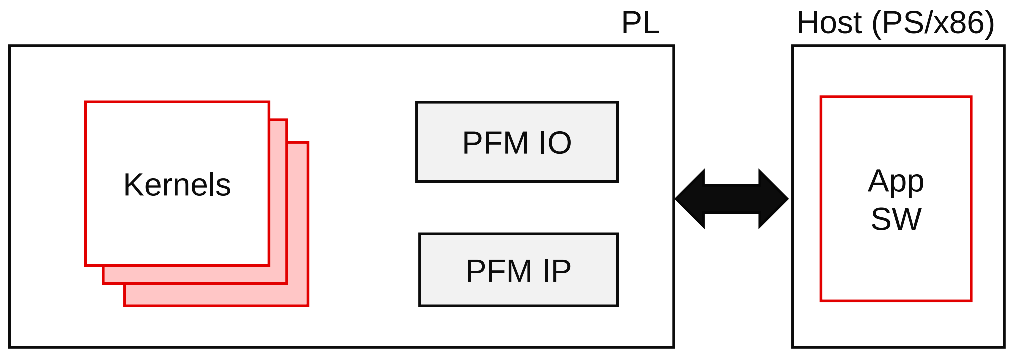
First block design:
Second block design:
Output file: /diff.html
Format: HTML

```
Reading JSON: ext_platform.bd
Reading JSON: mpsoc_preset.bd
< ext_platform.bd
> mpsoc_preset.bd

design_info
 < device=xcvc1902-vsva2197-2MP-e-S
 > device=xczu9eg-ffvb1156-2-e

components
 > zynq_ultra_ps_e_0
 < CIPS_0
 < axi_intc_0
 < axi_noc_ddr4
< clk_wizard_0
 < proc_sys_reset_0
< smartconnect_1
 < ai_engine_0
 < axi_noc_lpddr4

nets
 > zynq_ultra_ps_e_0_pl_clk0
 < CIPS_0_pl_clk0
 < CIPS_0_pl_resetn1
 < CIPS_0_ps_pmc_noc_axi0_clk
 < CIPS_0_ps_ps_noc_cci_axi0_clk
 < axi_intc_0_irq
 < clk_wizard_0_clk_out1
 < clk_wizard_0_locked
 < proc_sys_reset_0_peripheral_aresetn
 < ai_engine_0_s00_axi_aclk

design
 < interface_ports
 < interface_nets
 < comments
 < addressing
```

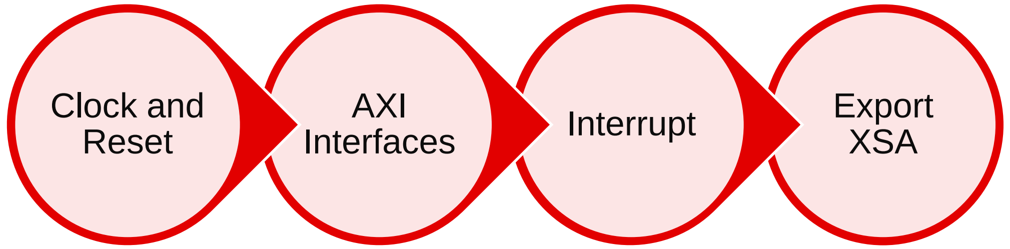
XILINX

Platforms for Acceleration Development

- Platform provides hardware capabilities that matches the needs of the software application



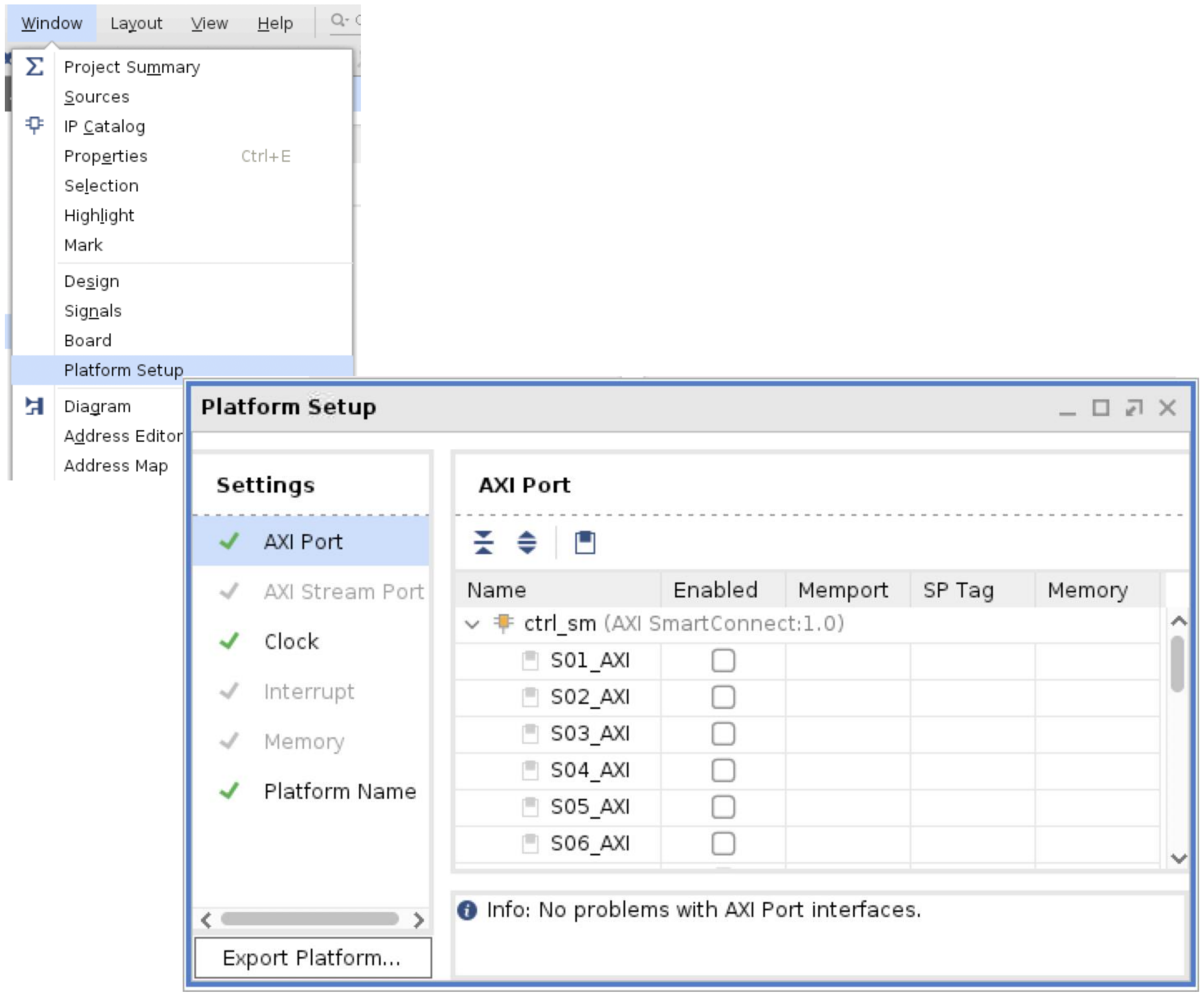
- IPI supports preparing the hardware design for export the platform to software environment



XILINX.

Platform Setup

- *Control Interfaces*
  - *Minimum of one for kernel control*
- *Memory Interfaces*
  - *Minimum of one for data exchange*
- *Streaming Interfaces*
- *Clocks and Reset*
- *Interrupt*

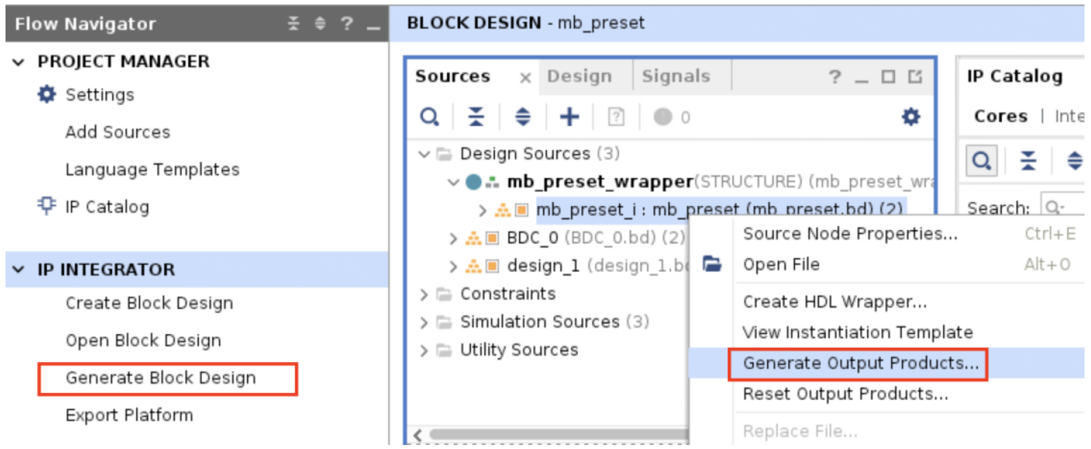


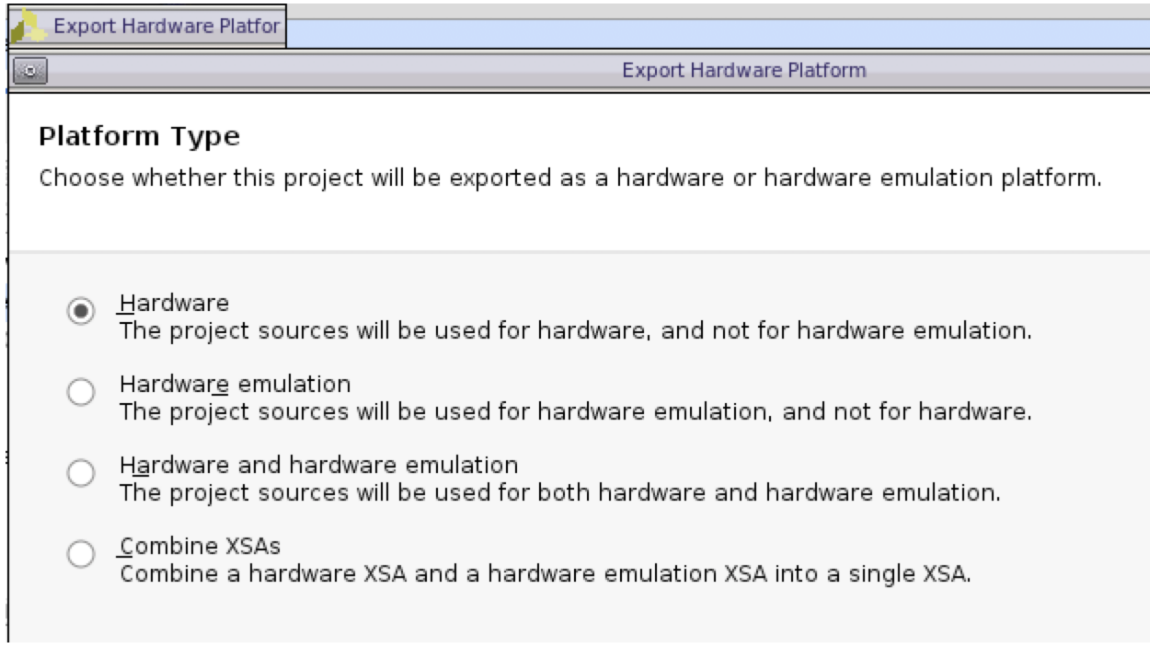
Name	Enabled	Memport	SP Tag	Memory
ctrl_sm (AXI SmartConnect:1.0)				
S01_AXI	☐			
S02_AXI	☐			
S03_AXI	☐			
S04_AXI	☐			
S05_AXI	☐			
S06_AXI	☐			

Info: No problems with AXI Port interfaces.

Export to Vitis

1. Generate Block Design
2. Generate Output Products
3. Run Export Platform Wizard

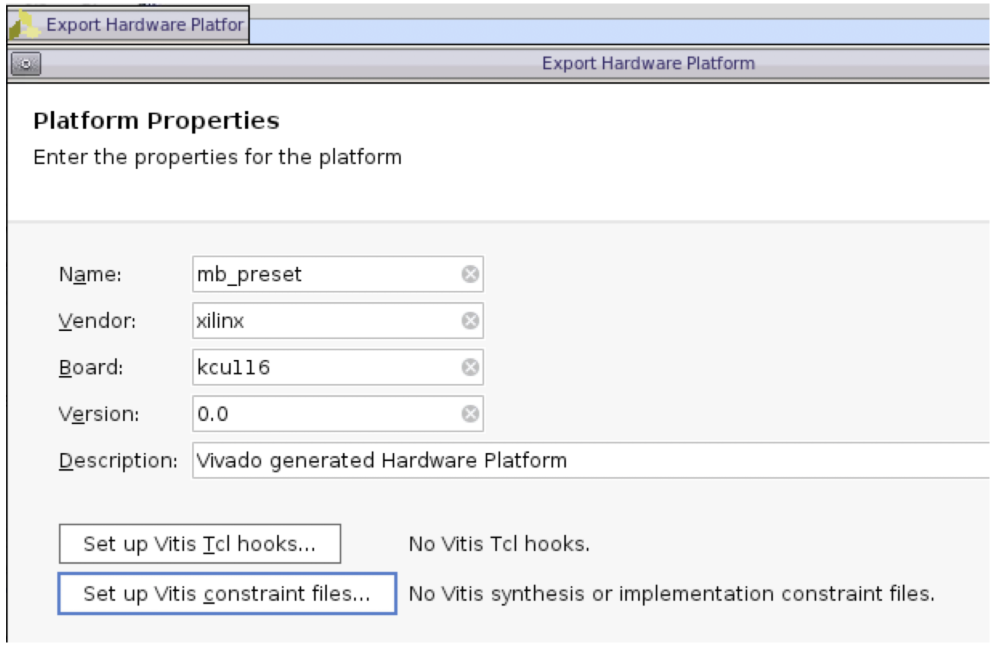




Platform Type

Choose whether this project will be exported as a hardware or hardware emulation platform.

- [✓ Hardware] The project sources will be used for hardware, and not for hardware emulation.
- Hardware emulation The project sources will be used for hardware emulation, and not for hardware.
- Hardware and hardware emulation The project sources will be used for both hardware and hardware emulation.
- Combine XSAs Combine a hardware XSA and a hardware emulation XSA into a single XSA.



Platform Properties

Enter the properties for the platform

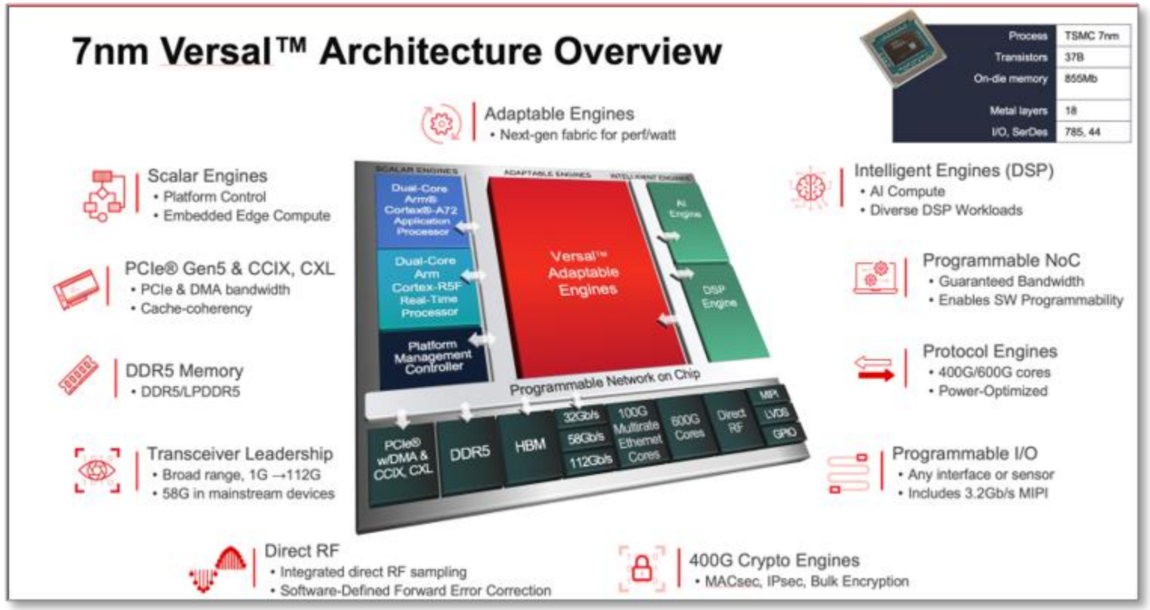
Name:	mb_preset
Vendor:	xilinx
Board:	kcu116
Version:	0.0
Description:	Vivado generated Hardware Platform

Set up Vitis Tcl hooks...	No Vitis Tcl hooks.
Set up Vitis constraint files...	No Vitis synthesis or implementation constraint files.

XILINX.

Creating a Simple Versal Design

IPI Advantage for Versal

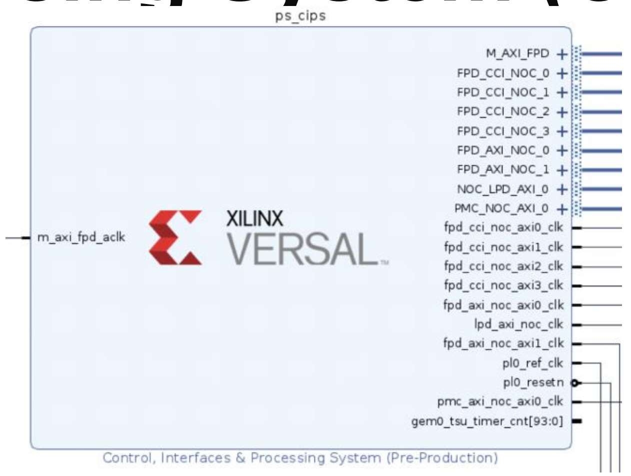


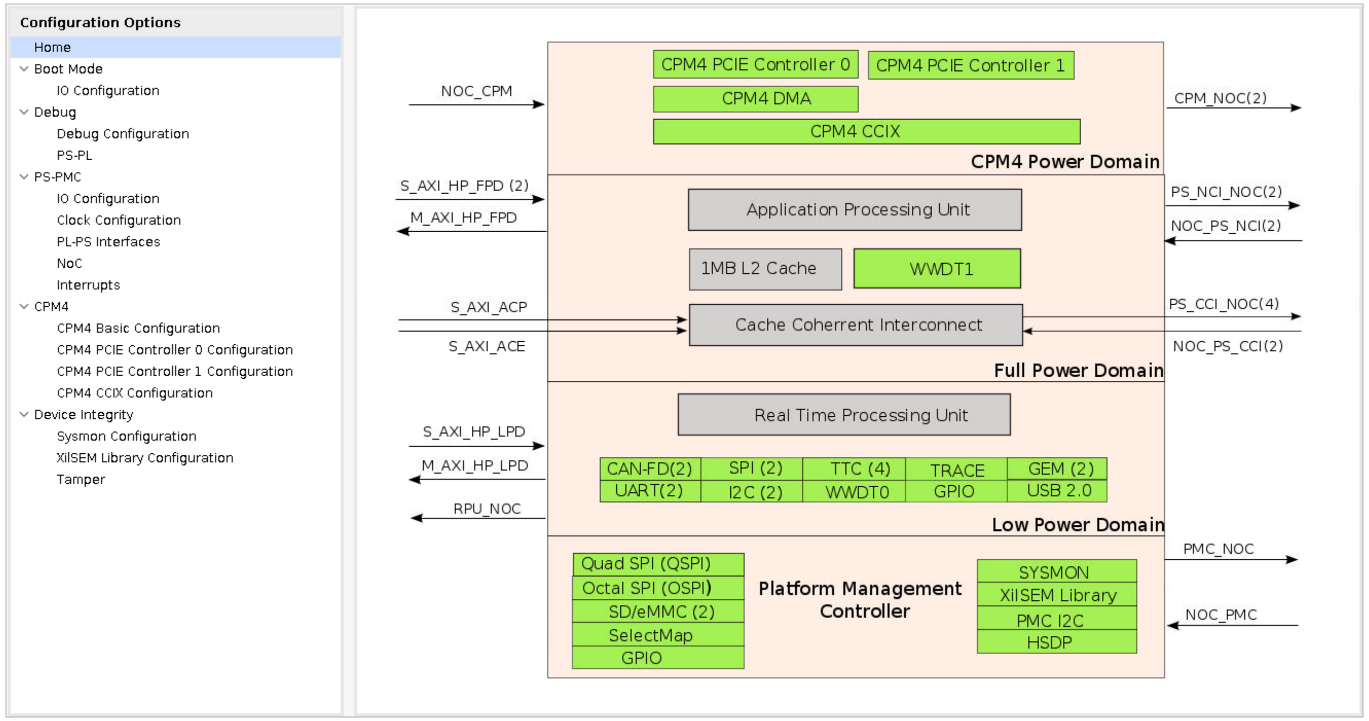
- Major changes in Versal architecture:
  - AIE, NoC, new PS
  - Shared DDR through NoC (no PS DDR)
  - HW & SW programmability
    - PL configuration through PMC
    - Debug through PMC
  - PCIe/CPM/GT-based IP sharing methodology (quad)
  - AXI interface for all hard, soft IP, AIE, NoC

- IPI offering for Versal:
  - Automatic configuration updates between Versal device-specific blocks (incl. CIPS & NoC)
  - Automatic connectivity between various blocks, which prevents errors
  - Seamless interaction with the Vitis tools, allowing export of custom hardware platforms

Versal Control, Interface, & Processing System (CIPS)

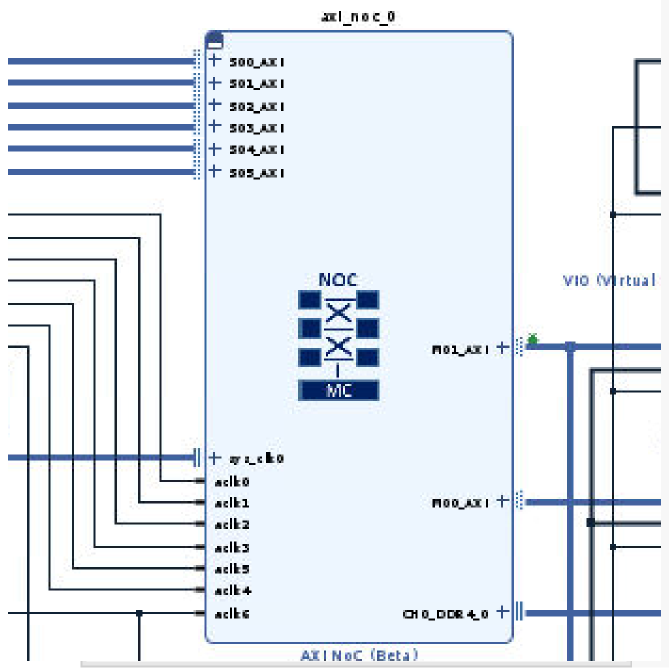
- Centralized device control
  - Covers many functions for PS, PMC, debug, NoC, CPM, system monitors, SEM, tamper
  - Required for basic non-JTAG boot
  - CIPS design flow via IPI
- Must have for all Versal designs

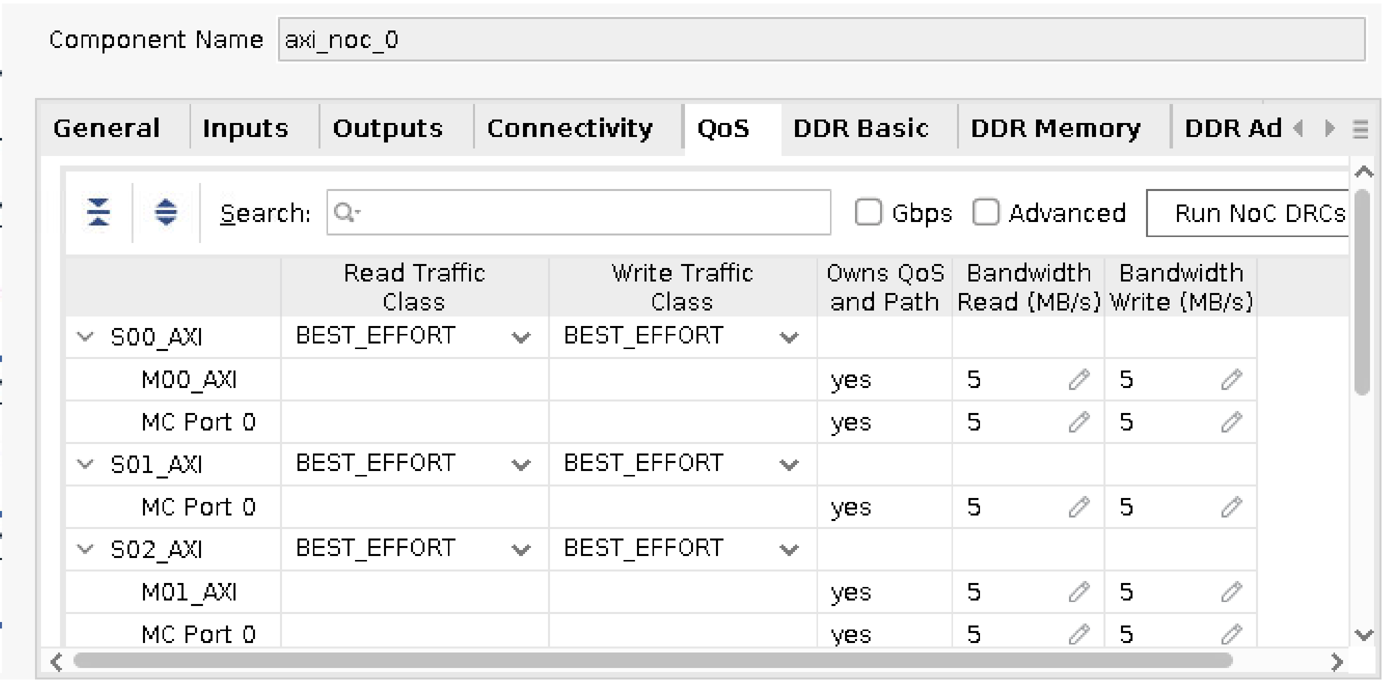




Versal NoC

- Facilitates communication among PS, DDR, AI Engines, Programmable Logic, and any other hardened components
  - Shared connectivity to move packetized data around the SoC

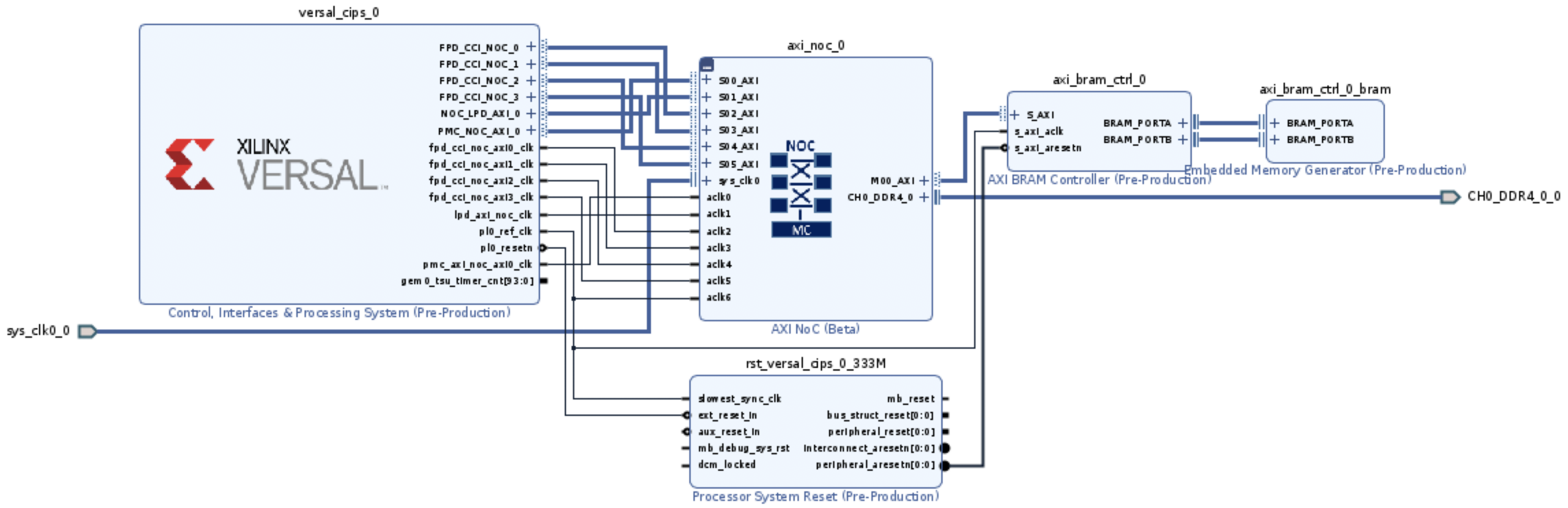



Component Name axi_noc_0

	Read Traffic Class	Write Traffic Class	Owns QoS and Path	Bandwidth Read (MB/s)	Bandwidth Write (MB/s)
S00_AXI	BEST_EFFORT	BEST_EFFORT			
M00_AXI			yes	5	5
MC Port 0			yes	5	5
S01_AXI	BEST_EFFORT	BEST_EFFORT			
MC Port 0			yes	5	5
S02_AXI	BEST_EFFORT	BEST_EFFORT			
M01_AXI			yes	5	5
MC Port 0			yes	5	5

Building A Simple Versal Design

- Xilinx recommends using IPI to instantiate and configure the CIPS & NOC IPs



Summary

Summary

- Leverage IPI automation and ease-of-use features to tackle complex designs

- Use IPI block design containers for design re-use and team collaboration

- Use Vivado IP integrator for Versal designs
  - Especially for instantiation and configuration of the CIPS and NOC IPs

- Additional references:
  - For IPI usage information and general hardware platform generation information, see UG994
  - For a tutorial on creating and packaging custom IP, see UG1119
  - For a high-level overview of the Versal ACAP design flow, see UG1273

XILINX®

Thank You