Embedded Vision Workshop Series: From Zero to Hero with PYNQ

Session 2

Getting Up and Running with PYNQ

Lab Workbook

Presented by

Sponsored by

PYNQ Workshop Series

About This Workbook

This workbook is intended to be used with **Session 2: Getting Up and Running with PYNQ** of the **Embedded Workshop Series: From Zero to Hero with PYNQ**, presented by element14.

To access the workbook from Session 1: Getting Started with PYNQ, please go here.

The contents of this workbook are created and owned by Adiuvo Engineering and Training, Ltd.

Your Instructor

Adam Taylor
Founder and
Principal Consultant

ADIUVO
ENGINEERING AND TRAINING, LTD

Please email any questions you may have to your instructor at adam@adiuvoengineering.com.

Required Hardware and Software

To complete this lab series, you will need the following hardware:

1. PYNQ-Z2 board
2. Micro SD card greater than equal to 16 GB
3. Micro SD card adapter
4. Micro USB cable
5. Ethernet cable
6. Ethernet access to your WIFI network or Ethernet connector on a PC
7. Sports camera e.g. https://www.amazon.co.uk/Crosstour-Waterproof-Underwater-Wide-angle-Rechargeable/dp/B073WWSYJK?ref_=fsclp_pl_dp_1
8. HDMI to micro HDMI cable https://www.amazon.co.uk/AmazonBasics-High-speed-latest-standard-meters/dp/B014I8TZXW

To be able to complete these three labs you will need the following software on your development machine:

1. Vivado Design Suite 2019.1 – used for the development of custom overlays
2. 7-Zip - https://www.7-zip.org/
3. Etcher - https://www.balena.io/etcher/
4. Tera Term - https://ttssh2.osdn.jp/index.html.en
5. Pynq-Z2 board definition files - https://d2m32eurp10079.cloudfront.net/Download/pynq-z2.zip
6. WinSCP - https://winscp.net/eng/index.php

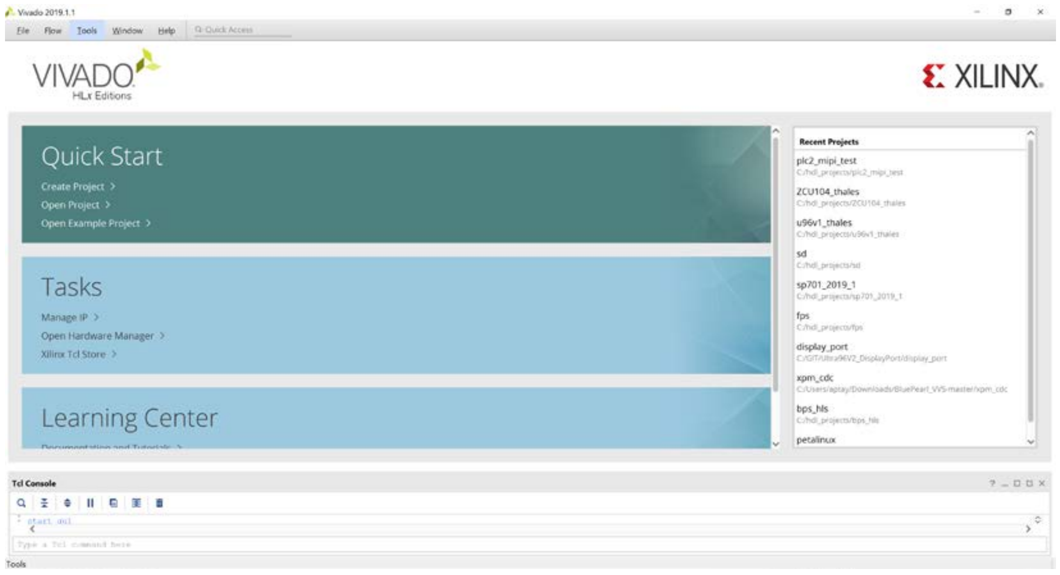
All project files and support files can be found at https://github.com/ATaylorCEngFIET/Element14_PYNQ

Lab Two

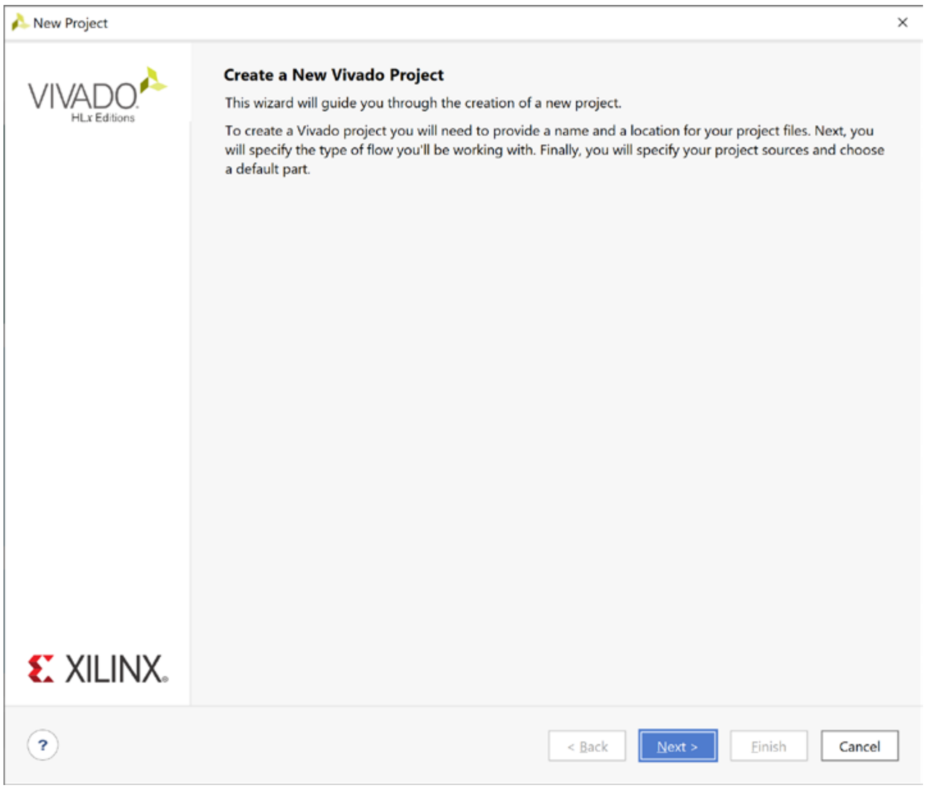
In this lab, we will pick up from where we left off in Lab 1. To review Lab 1, go here.

Creating a project in Vivado

1. Open **Vivado Design Suite 2019.1**.



2. Click **Create Project** and click **Next** on the open dialog.

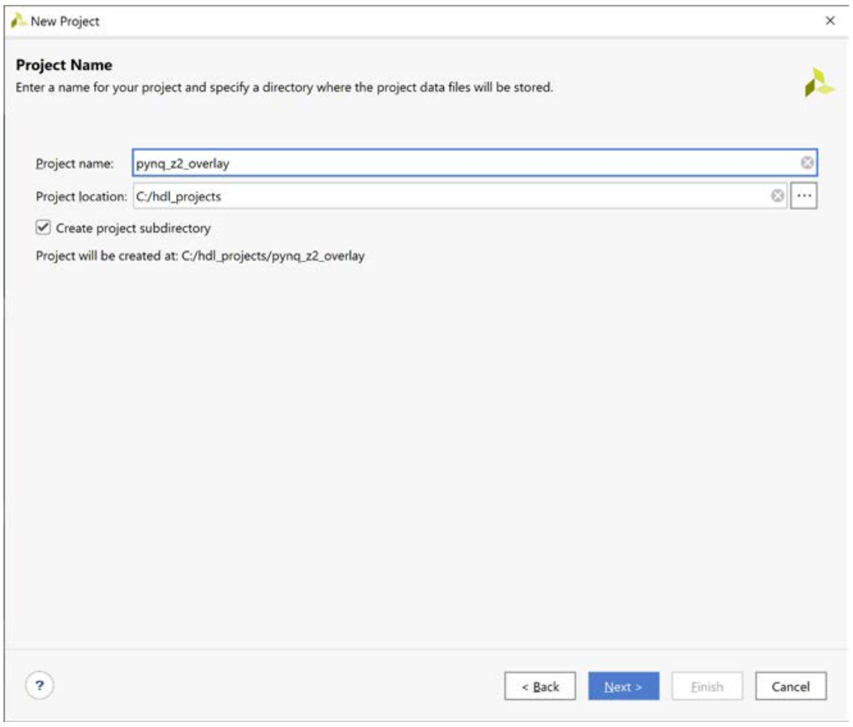


Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

3. Enter the project name as **pynq_z2_overlay** and select a directory to save the project in. Click **Next**.



New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

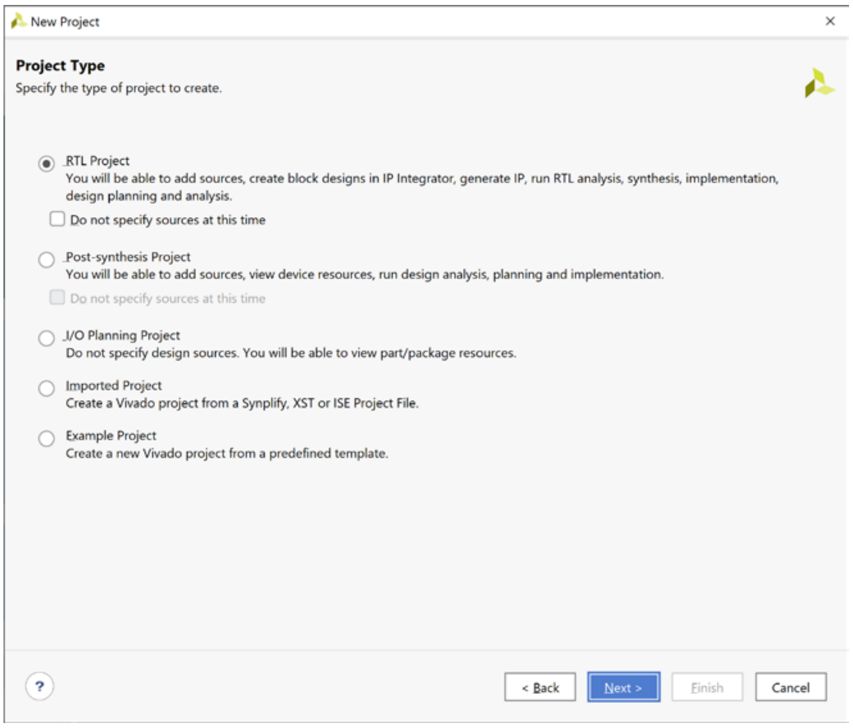
Project name: pynq_z2_overlay

Project location: C:/hdl_projects

[✓] Create project subdirectory

Project will be created at: C:/hdl_projects/pynq_z2_overlay

4. Select **RTL Project**. Click **Next**.



New Project

Project Type

Specify the type of project to create.

● RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☐ Do not specify sources at this time

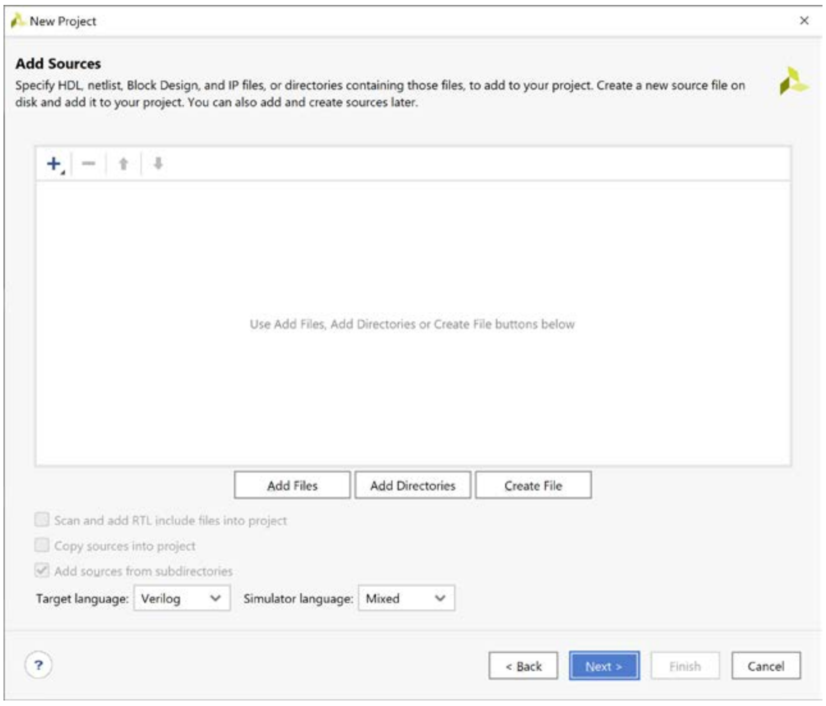
○ Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

○ I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

○ Imported Project
Create a Vivado project from a Synplify, XST or ISE Project File.

○ Example Project
Create a new Vivado project from a predefined template.

5. Click **Next** on the next frame as we have no sources to add.



New Project

Add Sources

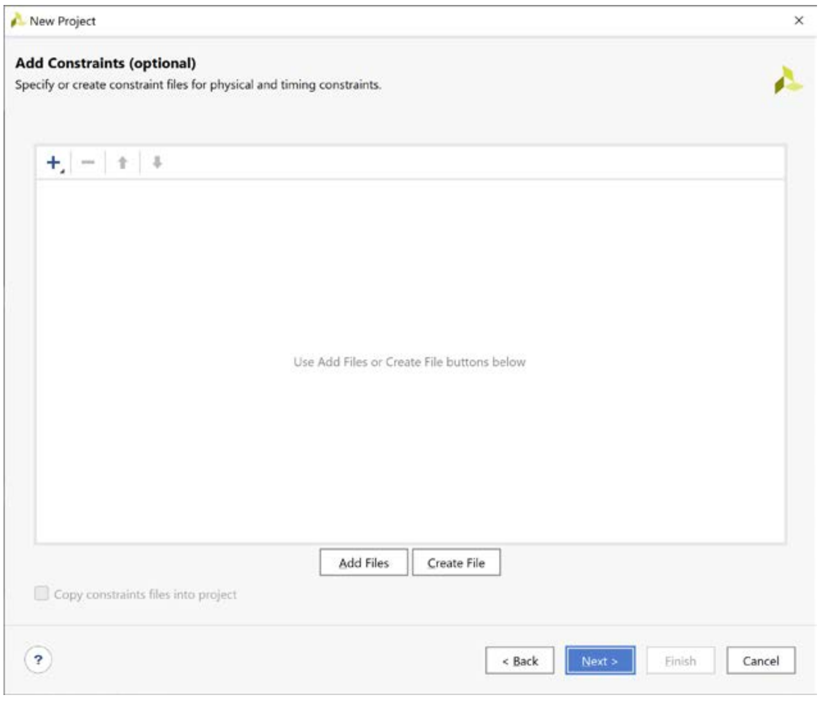
Specify HDL, netlist, Block Design, and IP files, or directories containing those files, to add to your project. Create a new source file on disk and add it to your project. You can also add and create sources later.

Use Add Files, Add Directories or Create File buttons below

- [ ] Scan and add RTL include files into project
- [ ] Copy sources into project
- [x] Add sources from subdirectories

Target language: Verilog Simulator language: Mixed

6. Click **Next** on the next frame as we have no constraints to add.



New Project

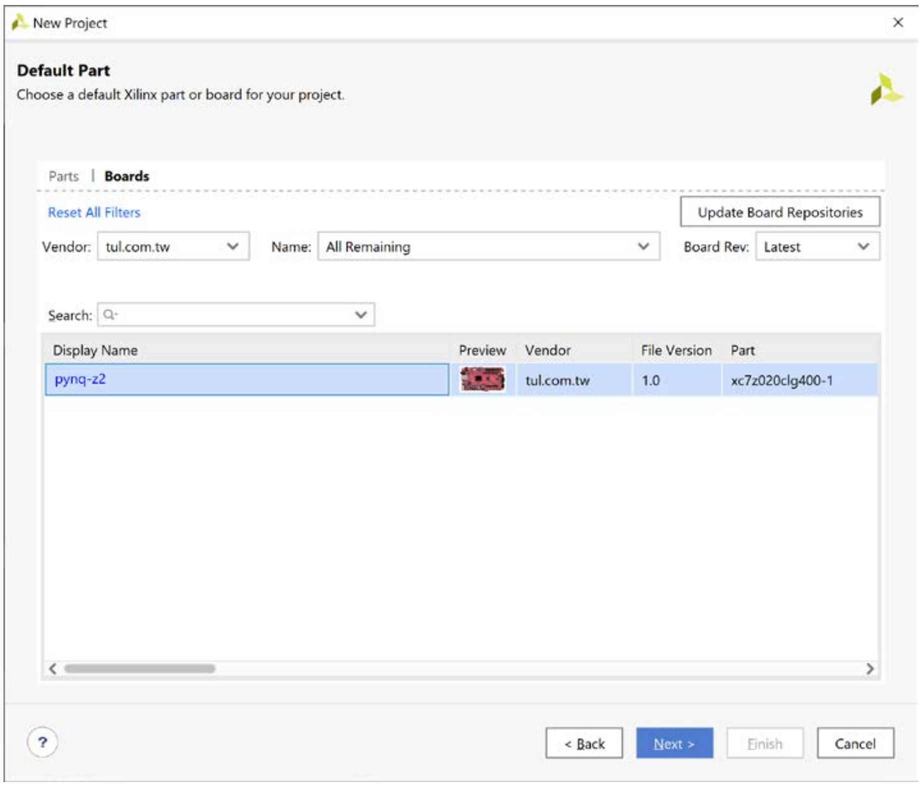
Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

Use Add Files or Create File buttons below

- [ ] Copy constraints files into project

7. On the next dialog, select the **Boards** tab, **TUL** as the vendor, and the **PYNQ-Z2** as the target board.



New Project

Default Part

Choose a default Xilinx part or board for your project.

Parts | **Boards**

Reset All Filters

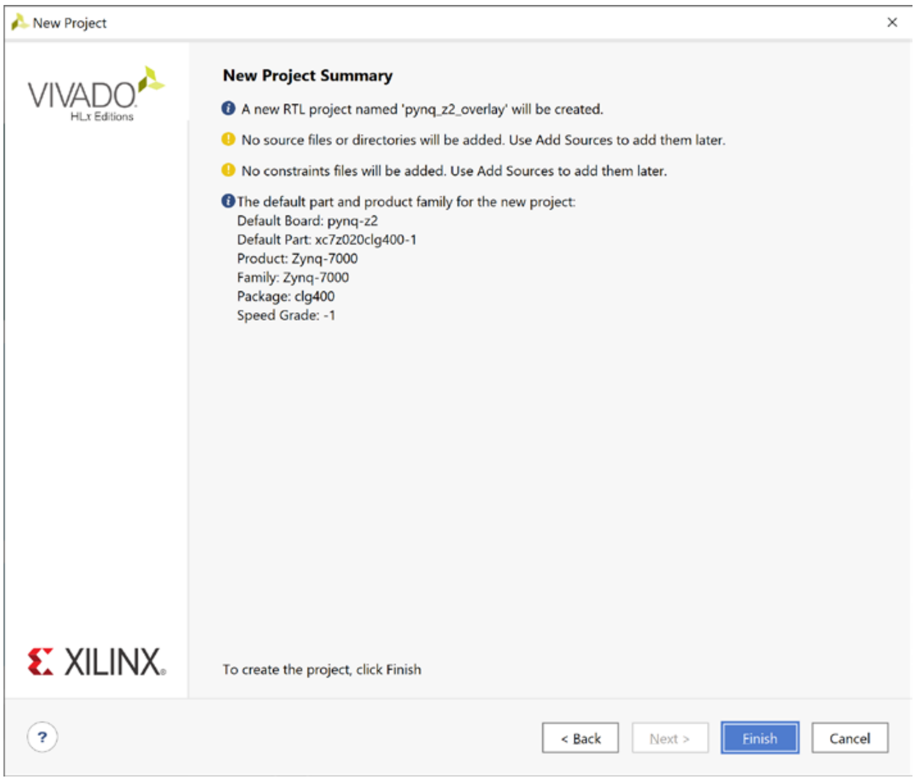
Update Board Repositories

Vendor: tul.com.tw   Name: All Remaining   Board Rev: Latest

Search:

Display Name	Preview	Vendor	File Version	Part
pynq-z2		tul.com.tw	1.0	xc7z020clg400-1

8. Select **Finish** on the summary page.



New Project

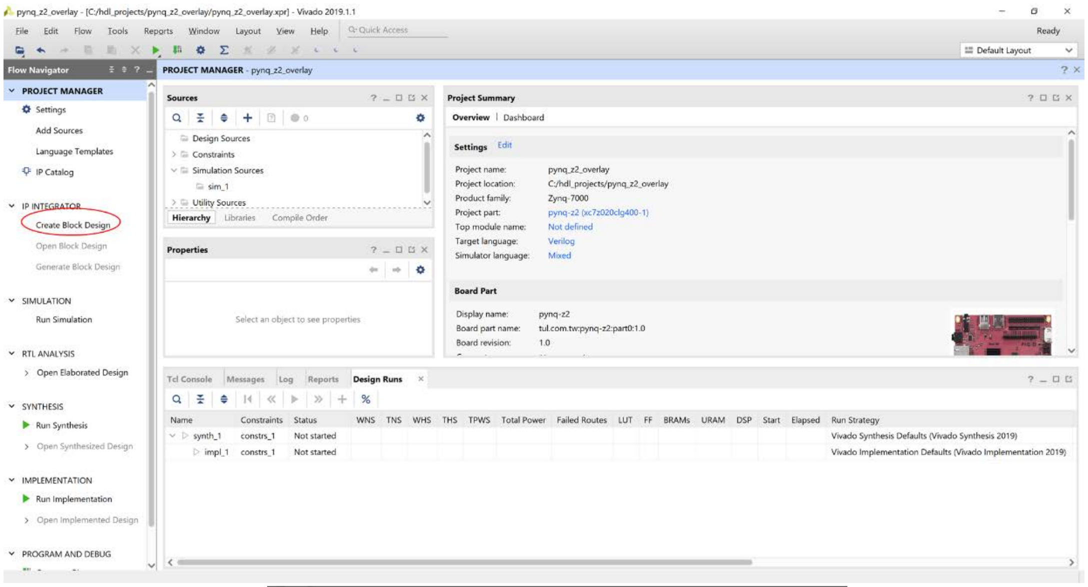
New Project Summary

- A new RTL project named 'pynq_z2_overlay' will be created.
- No source files or directories will be added. Use Add Sources to add them later.
- No constraints files will be added. Use Add Sources to add them later.
- The default part and product family for the new project:
  Default Board: pynq-z2
  Default Part: xc7z020clg400-1
  Product: Zynq-7000
  Family: Zynq-7000
  Package: clg400
  Speed Grade: -1

To create the project, click Finish

Creating the Block Diagram

9. Once the project is open, click on **Create Block Diagram** and leave the default name. Click **Ok**.



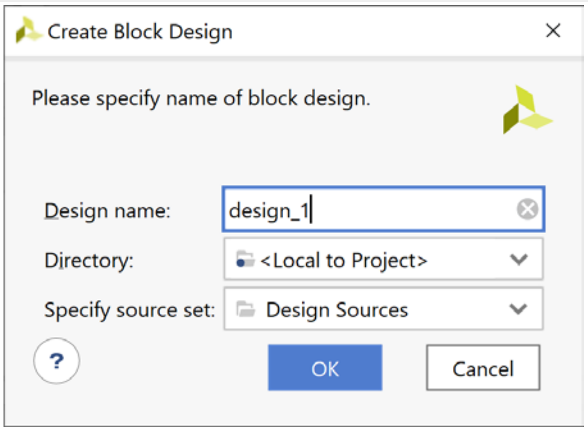
Settings Edit

Project name:	pynq_z2_overlay
Project location:	C:/hdl_projects/pynq_z2_overlay
Product family:	Zynq-7000
Project part:	pynq-z2 (xc7z020clg400-1)
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Board Part

Display name:	pynq-z2
Board part name:	tul.com.tw:pynq-z2:part0:1.0
Board revision:	1.0

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)



Please specify name of block design.

Design name:	design_1
Directory:	<Local to Project>
Specify source set:	Design Sources

10. Once the block diagram is open, click on the **+ symbol** and add in a **ZYNQ7 processing system**.

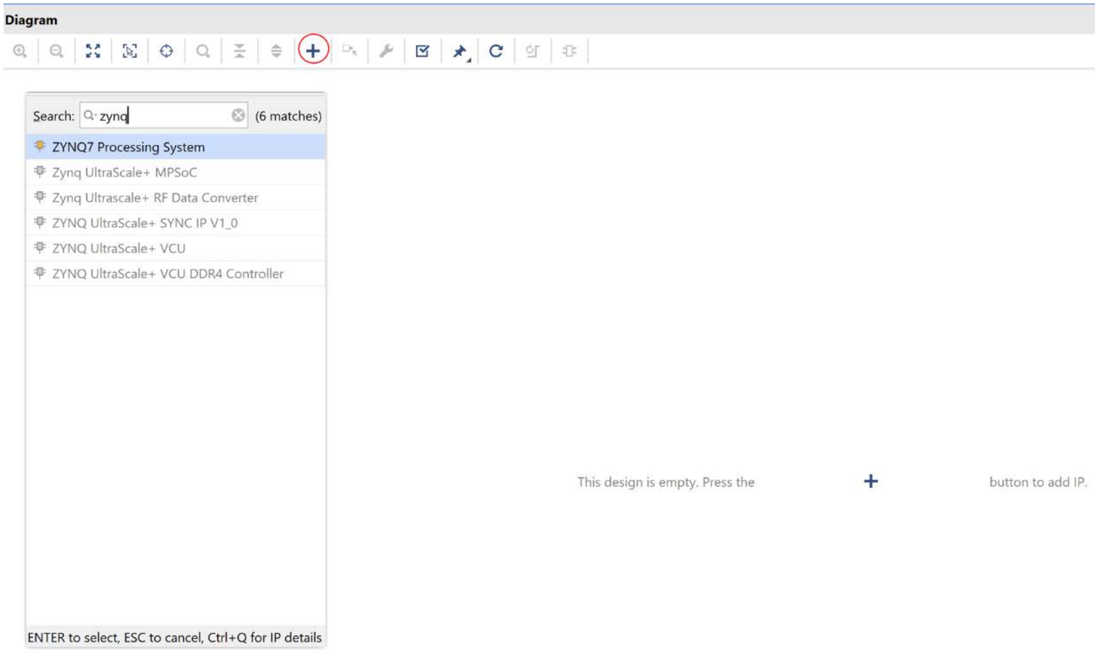


Diagram
Search: zynq
(6 matches)
ZYNQ7 Processing System
Zynq UltraScale+ MPSoC
Zynq Ultrascale+ RF Data Converter
ZYNQ UltraScale+ SYNC IP V1_0
ZYNQ UltraScale+ VCU
ZYNQ UltraScale+ VCU DDR4 Controller
ENTER to select, ESC to cancel, Ctrl+Q for IP details
This design is empty. Press the
button to add IP.

11. Once the Zynq PS is added to the block diagram, run the block automation of the Zynq PS for the settings of the PYNQ-Z2 by click on **Run Block Automation**.

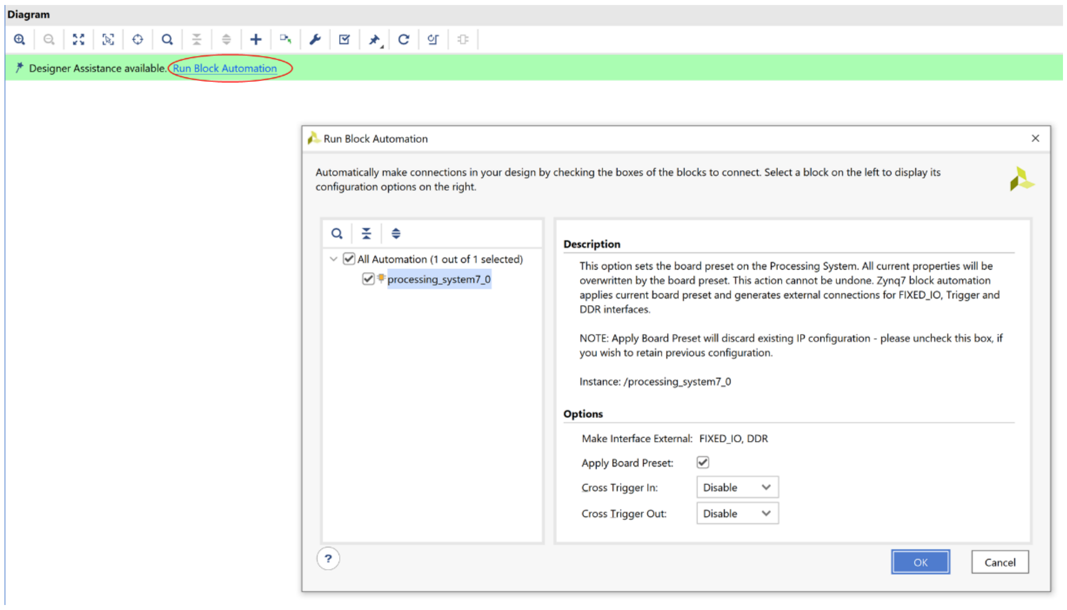


Diagram
Designer Assistance available. Run Block Automation
Run Block Automation
All Automation (1 out of 1 selected)
processing_system7_0
OK
Cancel

Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

Description

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Options

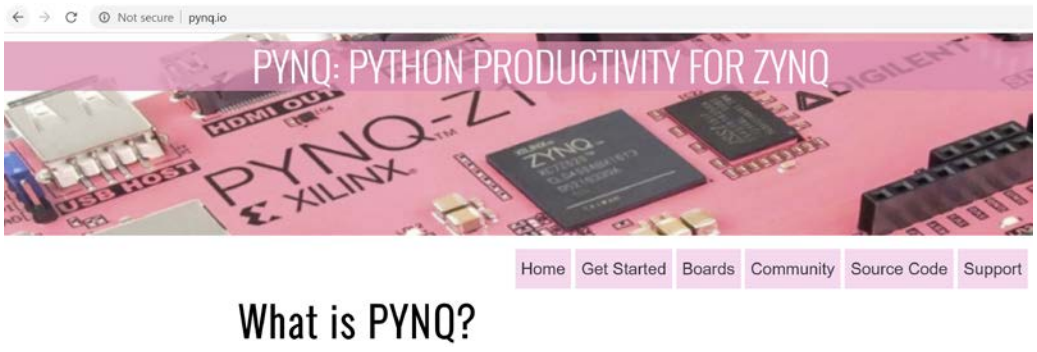
Make Interface External:	FIXED_IO, DDR
Apply Board Preset:	✓
Cross Trigger In:	Disable
Cross Trigger Out:	Disable

12. Click on **OK** and wait for the automation to complete. Once completed minimize Vivado.

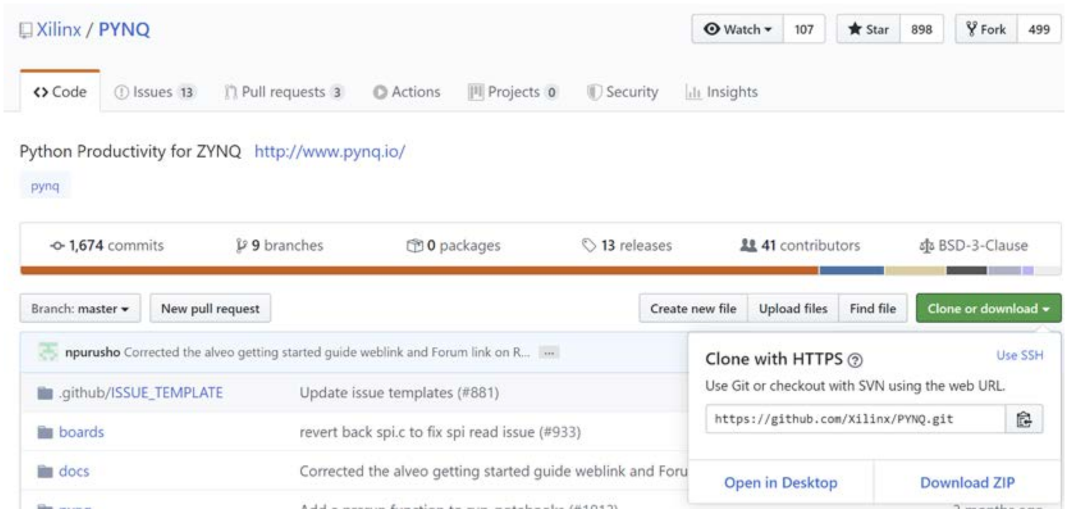
Getting PYNQ IP

13. Open a web browser and navigate to [www.Pynq.io.](http://www.Pynq.io)

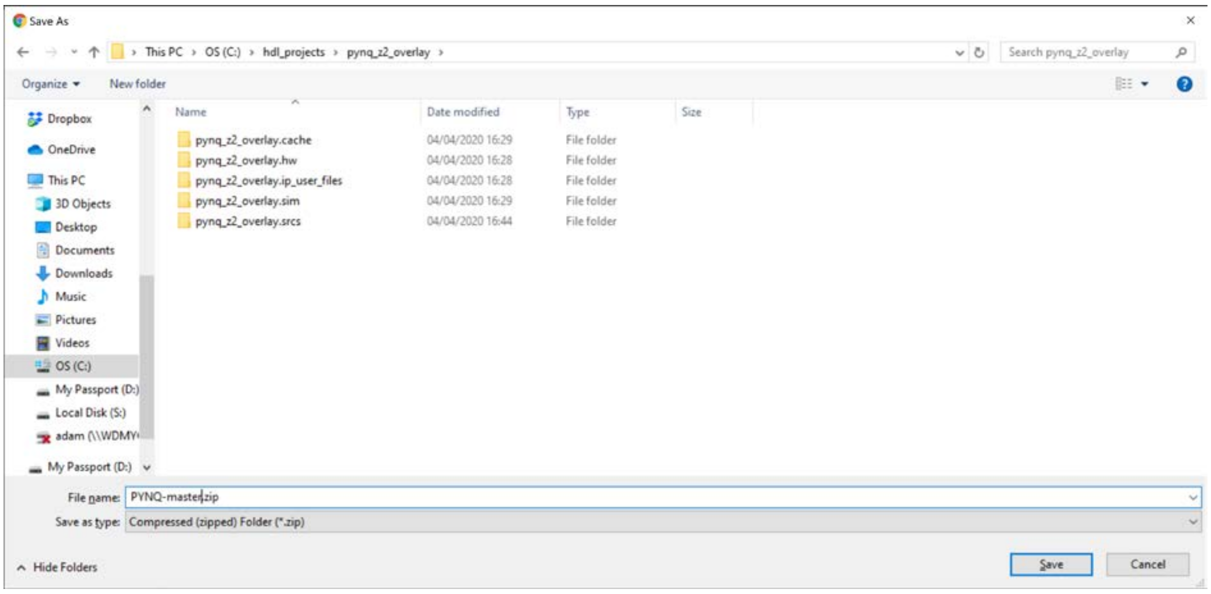
14. Click on **Source Code**.



15. This will open a Xilinx GitHub page. Select the option to **download as a ZIP file**.


Xilinx / PYNQ
Watch 107
Star 898
Fork 499
Code
Issues 13
Pull requests 3
Actions
Projects 0
Security
Insights
Python Productivity for ZYNQ http://www.pynq.io/
pynq
1,674 commits
9 branches
0 packages
13 releases
41 contributors
BSD-3-Clause
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
npurusho Corrected the alveo getting started guide weblink and Forum link on R...
.github/ISSUE_TEMPLATE
Update issue templates (#881)
boards
revert back spi.c to fix spi read issue (#933)
docs
Corrected the alveo getting started guide weblink and Foru
Clone with HTTPS
Use SSH
Use Git or checkout with SVN using the web URL.
https://github.com/Xilinx/PYNQ.git
Open in Desktop
Download ZIP

16. Download the ZIP to your computer. A suggested, but not mandatory location, is within your Vivado project.

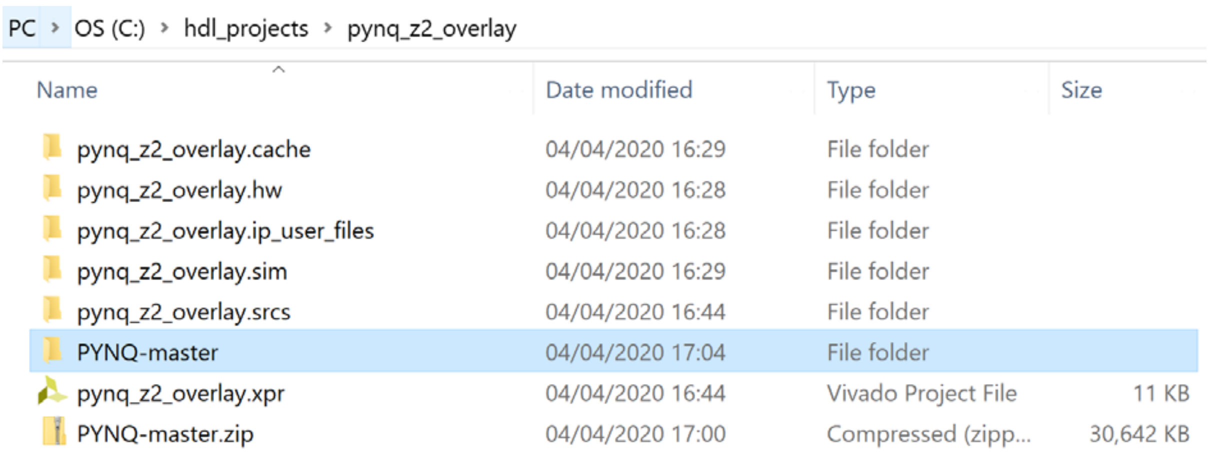


Name	Date modified	Type	Size
pynq_z2_overlay.cache	04/04/2020 16:29	File folder	
pynq_z2_overlay.hw	04/04/2020 16:28	File folder	
pynq_z2_overlay.ip_user_files	04/04/2020 16:28	File folder	
pynq_z2_overlay.sim	04/04/2020 16:29	File folder	
pynq_z2_overlay.srcs	04/04/2020 16:44	File folder	

File name: PYNQ-master.zip

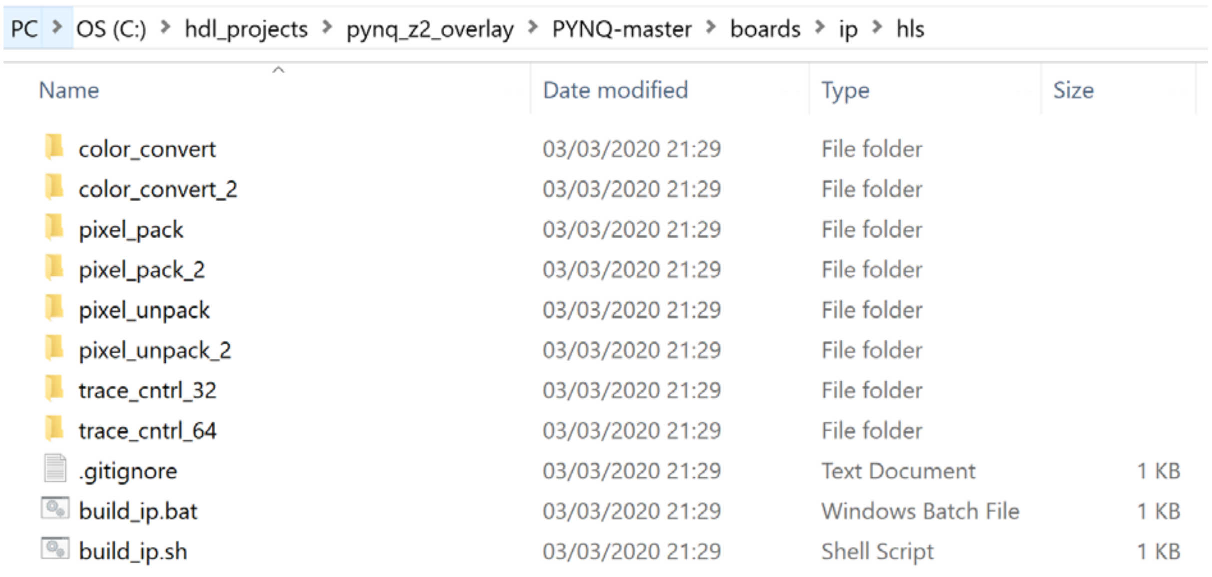
Save as type: Compressed (zipped) Folder (*.zip)

17. Extract the just downloaded ZIP file.



Name	Date modified	Type	Size
pynq_z2_overlay.cache	04/04/2020 16:29	File folder	
pynq_z2_overlay.hw	04/04/2020 16:28	File folder	
pynq_z2_overlay.ip_user_files	04/04/2020 16:28	File folder	
pynq_z2_overlay.sim	04/04/2020 16:29	File folder	
pynq_z2_overlay.srcs	04/04/2020 16:44	File folder	
PYNQ-master	04/04/2020 17:04	File folder	
pynq_z2_overlay.xpr	04/04/2020 16:44	Vivado Project File	11 KB
PYNQ-master.zip	04/04/2020 17:00	Compressed (zipp...	30,642 KB

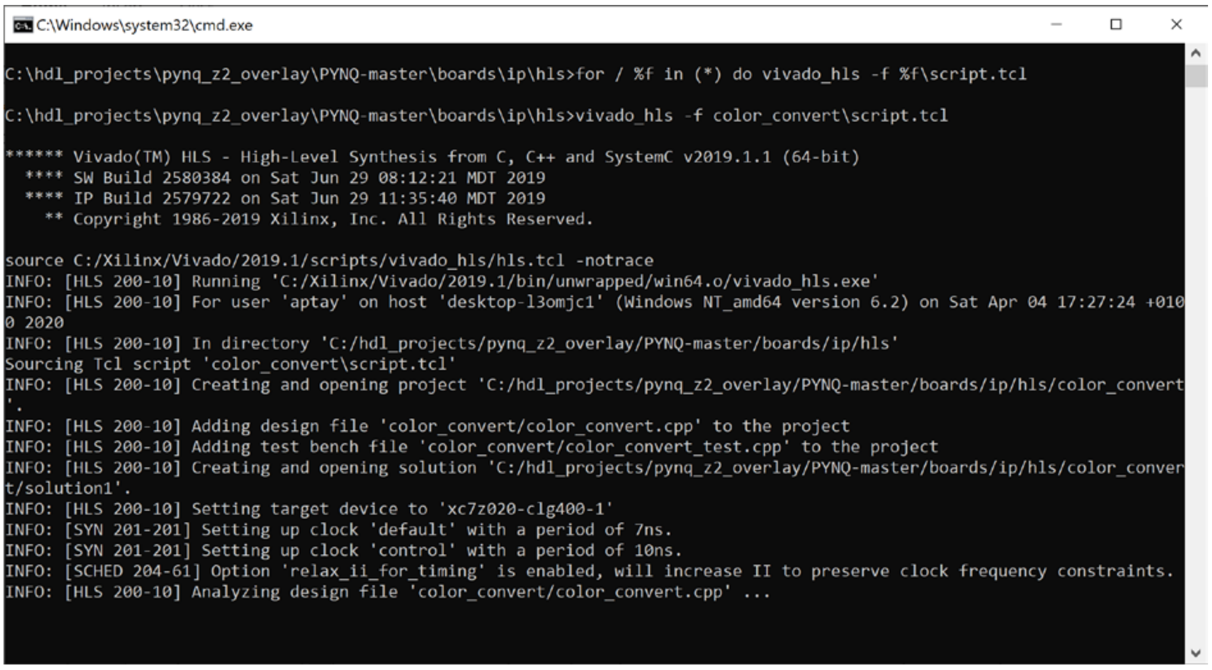
18. Open the PYNQ-master folder and navigate to **\PYNQ-master\boards\ip\hls**.



PC › OS (C:) › hdl_projects › pynq_z2_overlay › PYNQ-master › boards › ip › hls

Name	Date modified	Type	Size
color_convert	03/03/2020 21:29	File folder	
color_convert_2	03/03/2020 21:29	File folder	
pixel_pack	03/03/2020 21:29	File folder	
pixel_pack_2	03/03/2020 21:29	File folder	
pixel_unpack	03/03/2020 21:29	File folder	
pixel_unpack_2	03/03/2020 21:29	File folder	
trace_cntrl_32	03/03/2020 21:29	File folder	
trace_cntrl_64	03/03/2020 21:29	File folder	
.gitignore	03/03/2020 21:29	Text Document	1 KB
build_ip.bat	03/03/2020 21:29	Windows Batch File	1 KB
build_ip.sh	03/03/2020 21:29	Shell Script	1 KB

19. Double click on the build_ip.bat. This will build the HLS IP we need to use in our project.

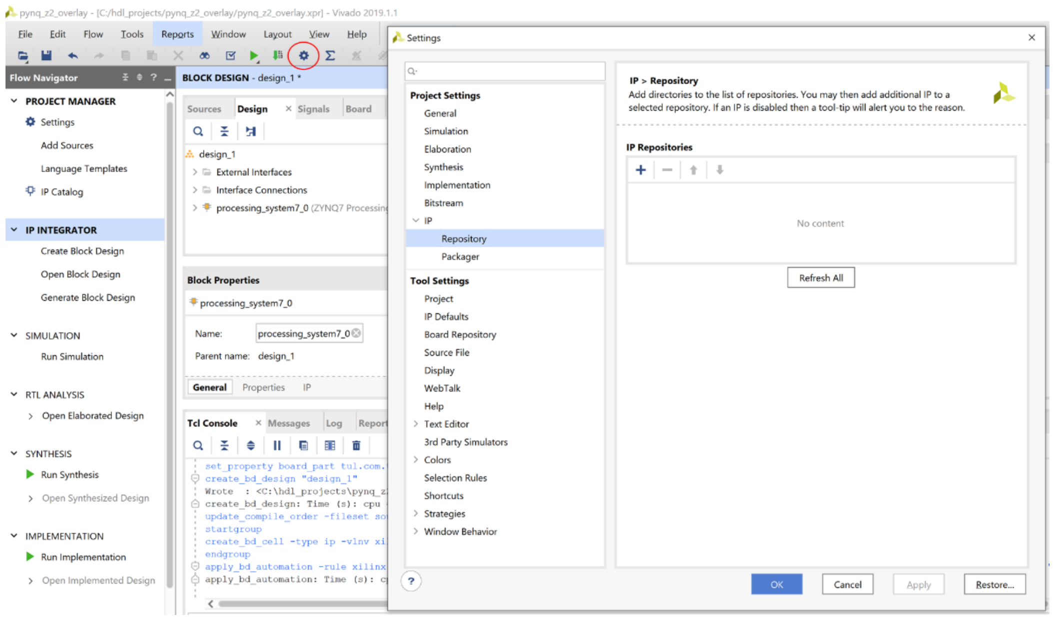


C:\Windows\system32\cmd.exe

```
C:\hdl_projects\pynq_z2_overlay\PYNQ-master\boards\ip\hls>for / %f in (*) do vivado_hls -f %f\script.tcl

C:\hdl_projects\pynq_z2_overlay\PYNQ-master\boards\ip\hls>vivado_hls -f color_convert\script.tcl

****** Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC v2019.1.1 (64-bit)
  **** SW Build 2580384 on Sat Jun 29 08:12:21 MDT 2019
  **** IP Build 2579722 on Sat Jun 29 11:35:40 MDT 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

source C:/Xilinx/Vivado/2019.1/scripts/vivado_hls/hls.tcl -notrace
INFO: [HLS 200-10] Running 'C:/Xilinx/Vivado/2019.1/bin/unwrapped/win64.o/vivado_hls.exe'
INFO: [HLS 200-10] For user 'aptay' on host 'desktop-l3omjc1' (Windows NT_amd64 version 6.2) on Sat Apr 04 17:27:24 +010
0 2020
INFO: [HLS 200-10] In directory 'C:/hdl_projects/pynq_z2_overlay/PYNQ-master/boards/ip/hls'
Sourcing Tcl script 'color_convert\script.tcl'
INFO: [HLS 200-10] Creating and opening project 'C:/hdl_projects/pynq_z2_overlay/PYNQ-master/boards/ip/hls/color_convert
'.
INFO: [HLS 200-10] Adding design file 'color_convert/color_convert.cpp' to the project
INFO: [HLS 200-10] Adding test bench file 'color_convert/color_convert_test.cpp' to the project
INFO: [HLS 200-10] Creating and opening solution 'C:/hdl_projects/pynq_z2_overlay/PYNQ-master/boards/ip/hls/color_conver
t/solution1'.
INFO: [HLS 200-10] Setting target device to 'xc7z020-clg400-1'
INFO: [SYN 201-201] Setting up clock 'default' with a period of 7ns.
INFO: [SYN 201-201] Setting up clock 'control' with a period of 10ns.
INFO: [SCHED 204-61] Option 'relax_ii_for_timing' is enabled, will increase II to preserve clock frequency constraints.
INFO: [HLS 200-10] Analyzing design file 'color_convert/color_convert.cpp' ...
```

Adding the IP into Vivado

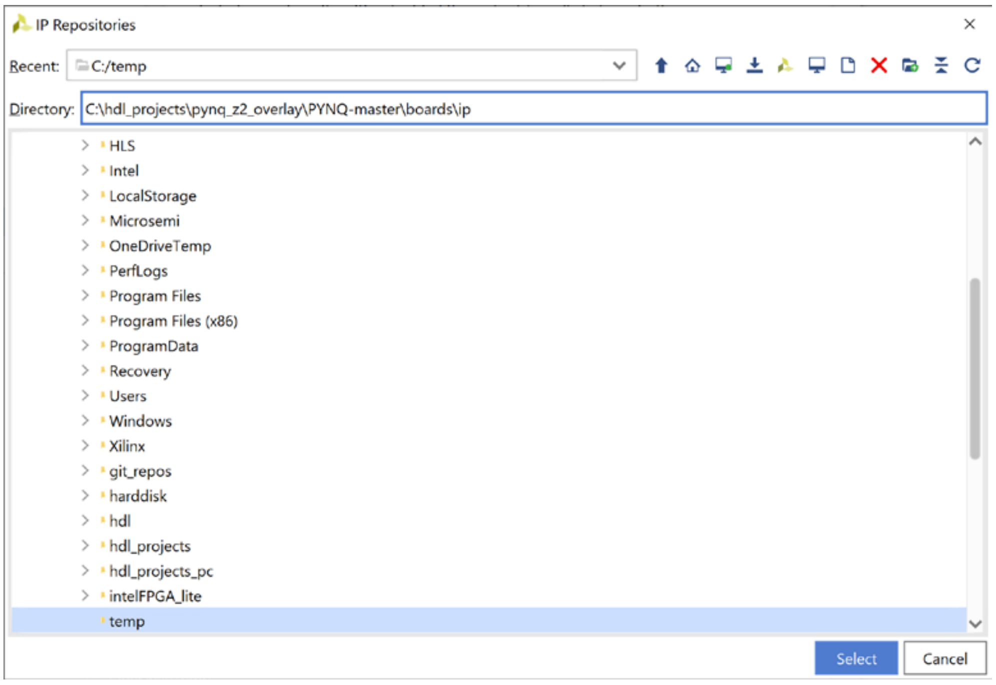
20. Once these IP have been built, we need to add in the IP created to our Vivado project so we can use it. Click on the **settings tab** and select **IP repository**.



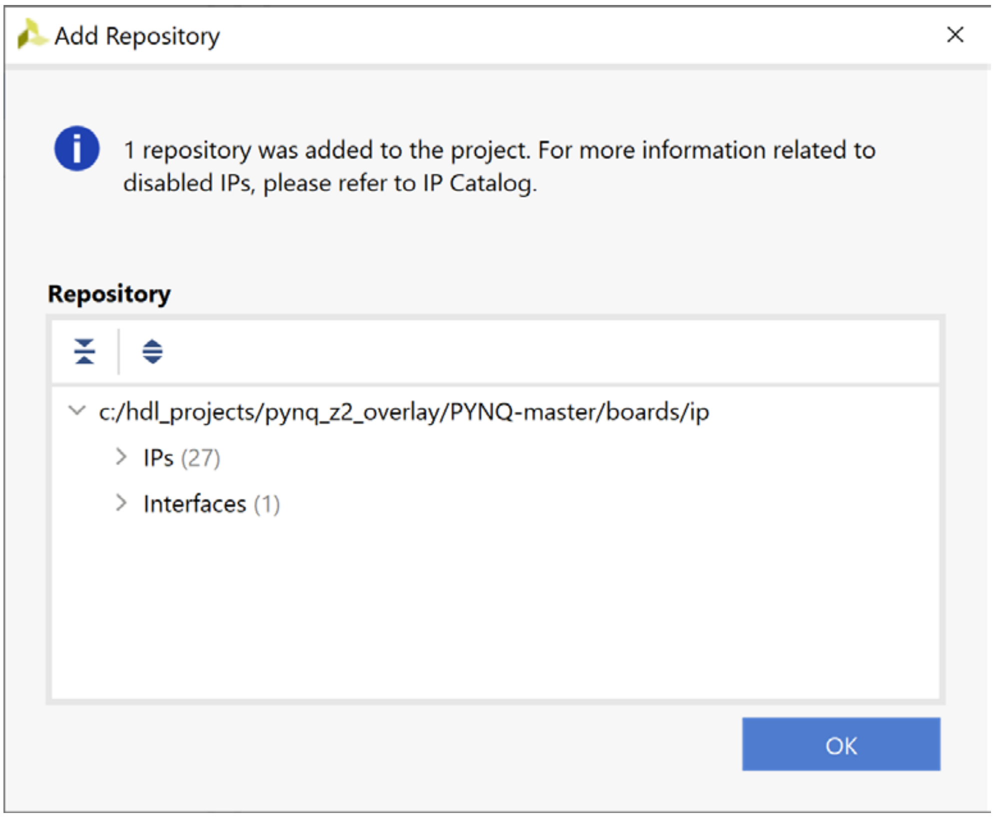
IP > Repository
Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

```
set_property board_part tul.com.
create_bd_design "design_1"
Wrote  : <C:\hdl_projects\pynq_z
create_bd_design: Time (s): cpu
update_compile_order -fileset so
startgroup
create_bd_cell -type ip -vlnv xi
endgroup
apply_bd_automation -rule xilinx
apply_bd_automation: Time (s): c
```

21. To add a new repository, select the **+ symbol** and add in the PYNQ IP repository **<save directory>pynq_z2_overlay\PYNQ-master\boards\ip**. Click **Ok**.



Directory: C:\hdl_projects\pynq_z2_overlay\PYNQ-master\boards\ip



Add Repository

1 repository was added to the project. For more information related to disabled IPs, please refer to IP Catalog.

Repository

c:/hdl_projects/pynq_z2_overlay/PYNQ-master/boards/ip
- IPs (27)
- Interfaces (1)

OK

22. We are now able to add in PYNQ IP into our block diagram. Add in the following IP:
    a. Color Convert
    b. Pixel Pack
    c. VDMA
    d. Test Pattern Generator

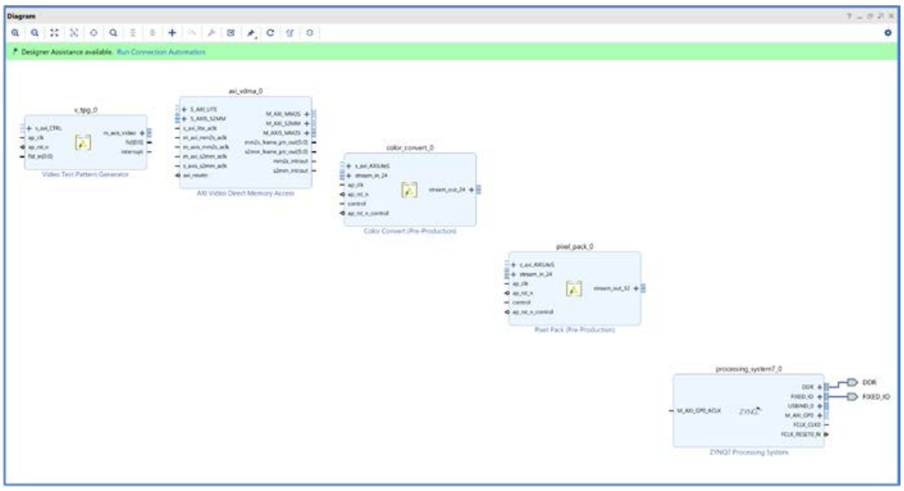
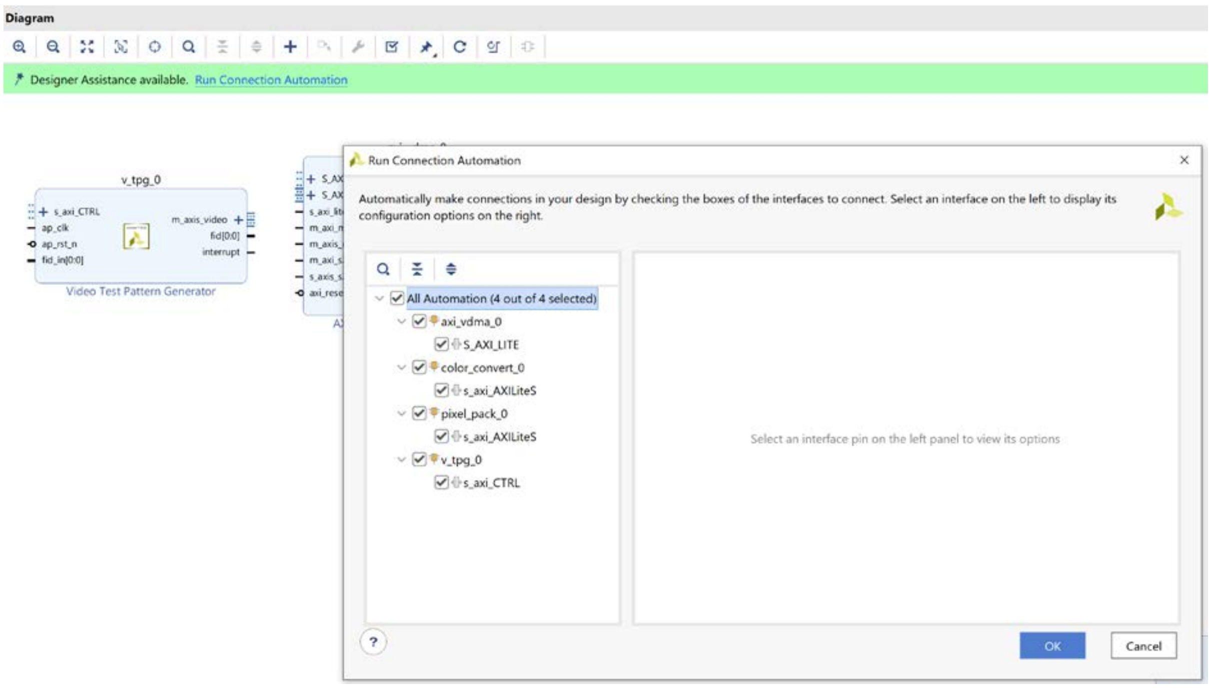


Diagram
Designer Assistance available. Run Connection Automation
v_tpg_0
Video Test Pattern Generator
axi_vdma_0
AXI Video Direct Memory Access
color_convert_0
Color Convert (Pre-Production)
pixel_pack_0
Pixel Pack (Pre-Production)
processing_system7_0
ZYNQ7 Processing System
DDR
FIXED_IO

Connecting the IP Blocks

23. Click on the **Run Connection Automation** in the dialog box which opens. Select **All Automation**. Click **Ok**.



Diagram

Designer Assistance available. Run Connection Automation

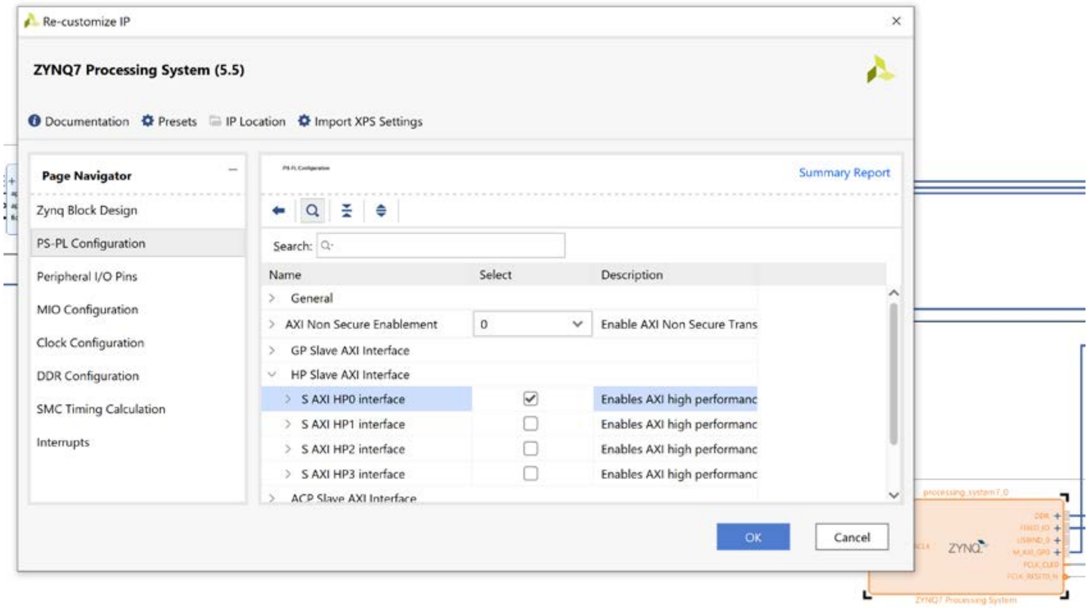
Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- ☑ All Automation (4 out of 4 selected)
  - ☑ axi_vdma_0
    - ☑ S_AXI_LITE
  - ☑ color_convert_0
    - ☑ s_axi_AXILiteS
  - ☑ pixel_pack_0
    - ☑ s_axi_AXILiteS
  - ☑ v_tpg_0
    - ☑ s_axi_CTRL

Select an interface pin on the left panel to view its options

24. Double click on the **ZYNQ7 processing system** for reconfiguration and select **PS-PL Configuration**. Enable one of the slave interfaces.



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation Presets IP Location Import XPS Settings

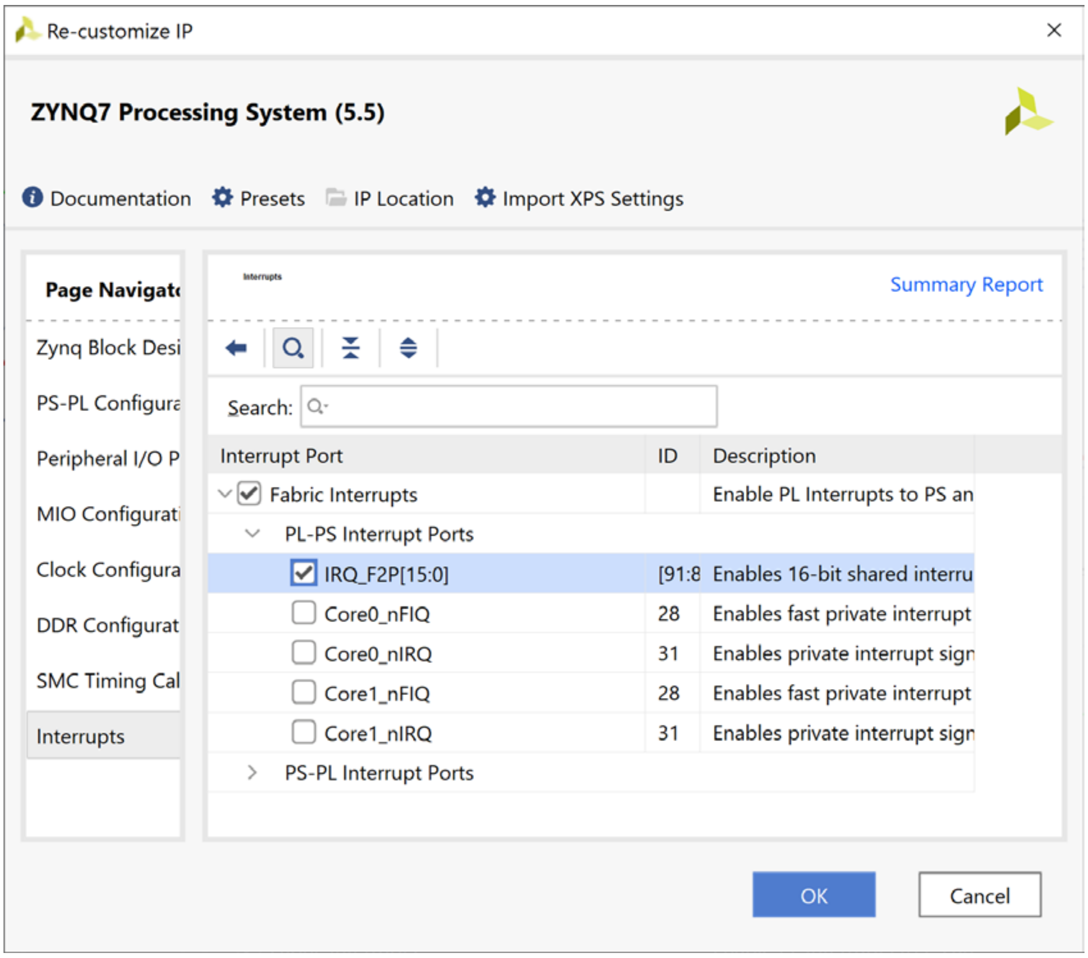
Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

Summary Report

Search:

Name	Select	Description
General		
AXI Non Secure Enablement	0	Enable AXI Non Secure Trans
GP Slave AXI Interface		
HP Slave AXI Interface		
S AXI HP0 interface	✓	Enables AXI high performanc
S AXI HP1 interface	☐	Enables AXI high performanc
S AXI HP2 interface	☐	Enables AXI high performanc
S AXI HP3 interface	☐	Enables AXI high performanc
ACP Slave AXI Interface		

25. While you are here, also select **Interrupts** from the left hand side and enable **Fabric Interrupts**. Enable **PL-PS IRQ_F2P[15:0]**. Click **Ok**.



Interrupt Port	ID	Description
✓ Fabric Interrupts		Enable PL Interrupts to PS an
PL-PS Interrupt Ports		
✓ IRQ_F2P[15:0]	[91:8	Enables 16-bit shared interru
Core0_nFIQ	28	Enables fast private interrupt
Core0_nIRQ	31	Enables private interrupt sign
Core1_nFIQ	28	Enables fast private interrupt
Core1_nIRQ	31	Enables private interrupt sign
PS-PL Interrupt Ports		

26. **Run Connection Automation** again. Select **S_AXI_HP0** on the left-hand side and select the master as **axi_vdma_0/M_AXI_S2MM**. Click **Ok**.

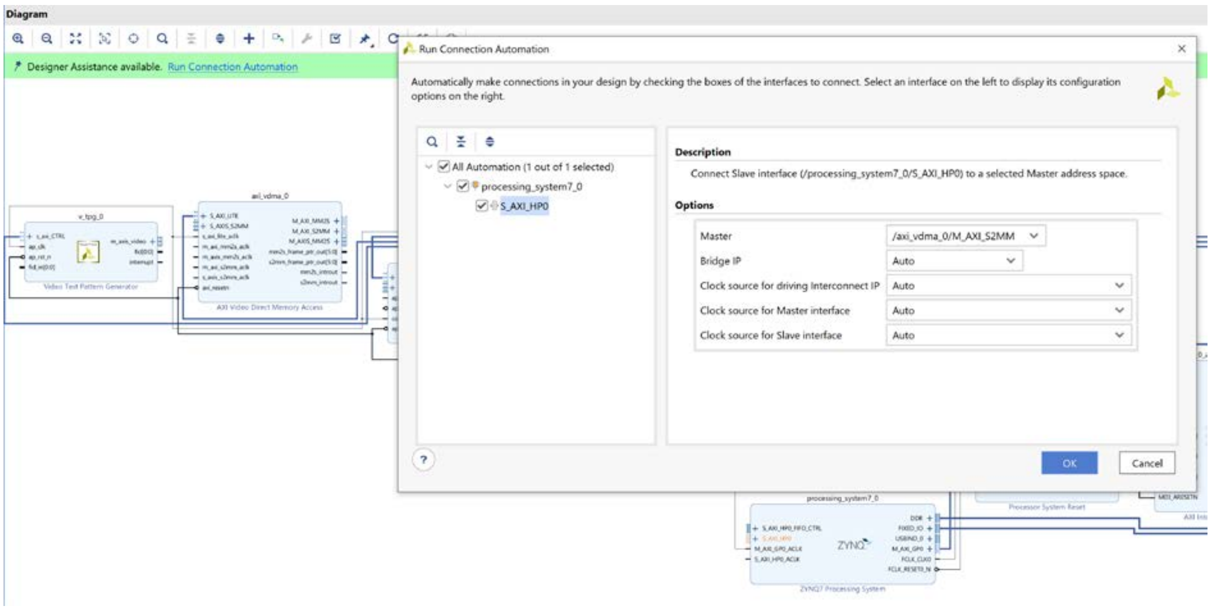


Diagram
Designer Assistance available. Run Connection Automation
Run Connection Automation
OK
Cancel

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

✓ All Automation (1 out of 1 selected)
✓ processing_system7_0
✓ S_AXI_HP0

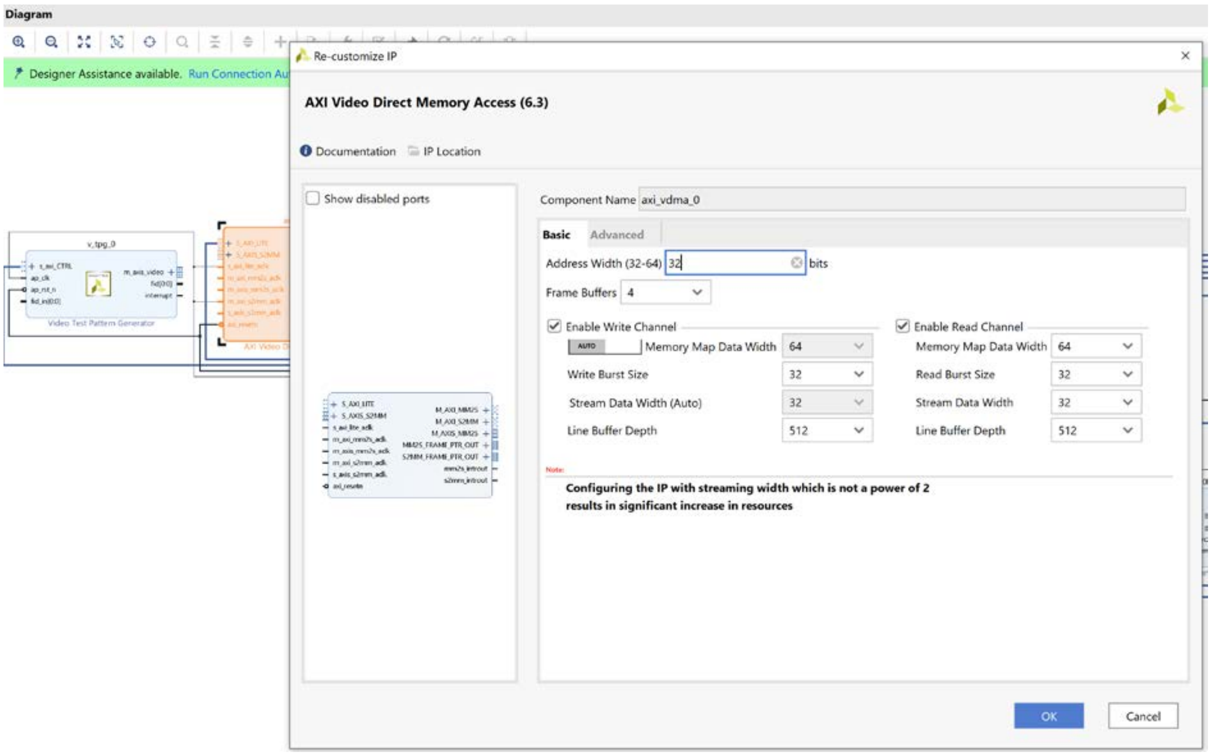
Description

Connect Slave interface (/processing_system7_0/S_AXI_HP0) to a selected Master address space.

Options

Master	/axi_vdma_0/M_AXI_S2MM
Bridge IP	Auto
Clock source for driving Interconnect IP	Auto
Clock source for Master interface	Auto
Clock source for Slave interface	Auto

27. Double click on the **VDMA** to reconfigure the VDMA. Select **4 Frame Buffers** and **write and read burst sizes** of **32**. Click **Ok**.


Re-customize IP

AXI Video Direct Memory Access (6.3)

Component Name axi_vdma_0

Address Width (32-64) 32 bits

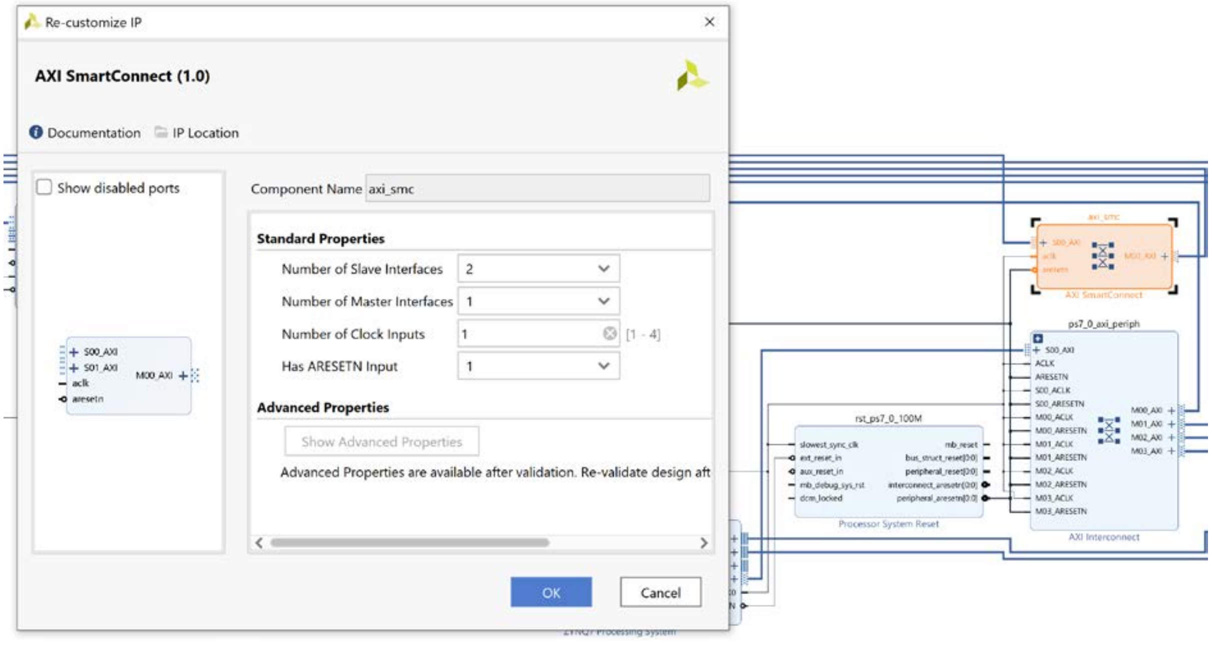
Frame Buffers 4

Enable Write Channel: Memory Map Data Width 64; Write Burst Size 32; Stream Data Width (Auto) 32; Line Buffer Depth 512

Enable Read Channel: Memory Map Data Width 64; Read Burst Size 32; Stream Data Width 32; Line Buffer Depth 512

Note:
Configuring the IP with streaming width which is not a power of 2 results in significant increase in resources

28. Double click on the **AXI SmartConnect** and enable **2 slave ports**.


Re-customize IP

AXI SmartConnect (1.0)

Component Name axi_smc

Standard Properties

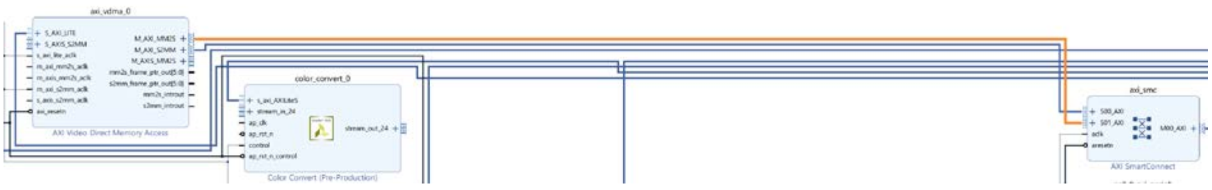
Number of Slave Interfaces 2
Number of Master Interfaces 1
Number of Clock Inputs 1 [1 - 4]
Has ARESETN Input 1

Advanced Properties

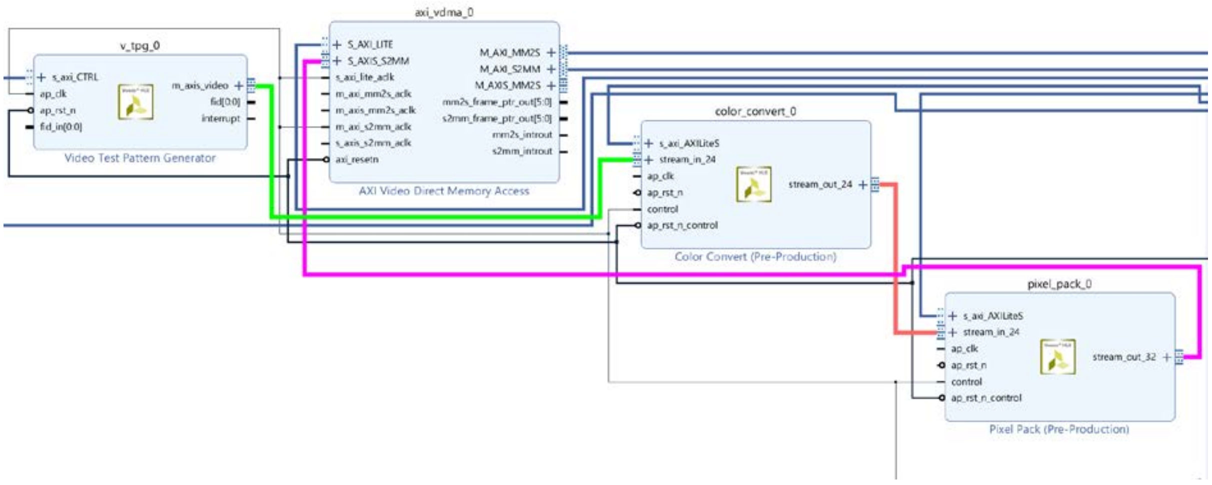
Show Advanced Properties

Advanced Properties are available after validation. Re-validate design aft

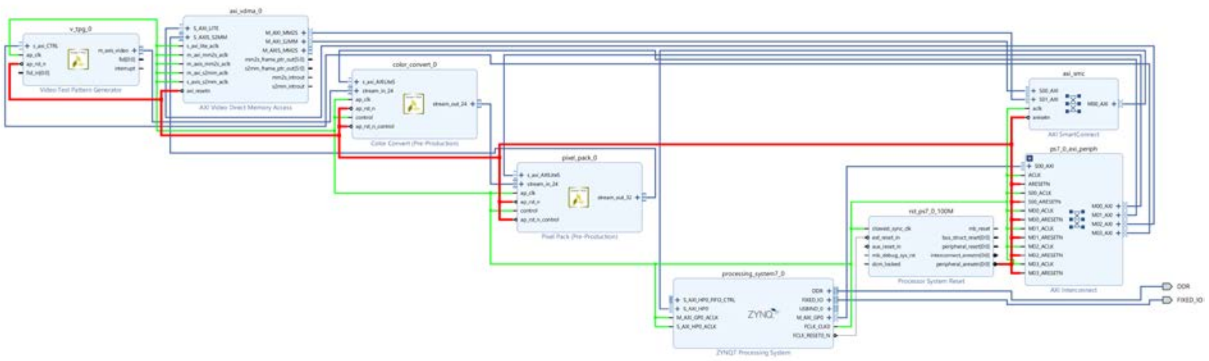
29. Connect the **VDMA M_AXI_MM2S** port to the **S01_AXI** on the **AXI SmartConnect**.



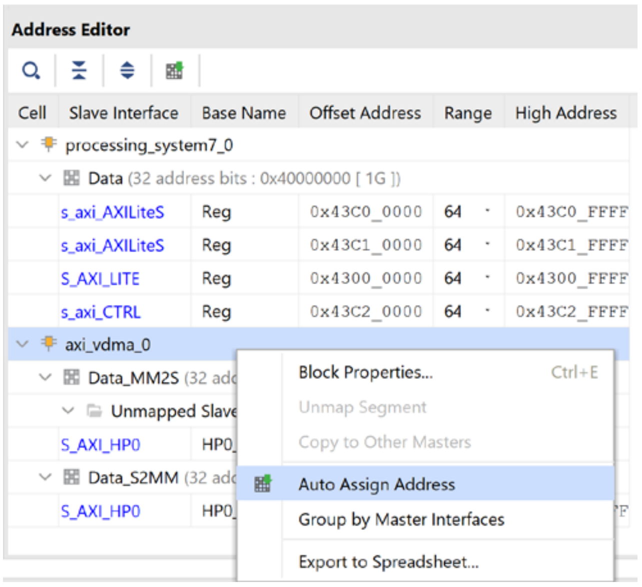
30. The next step is to connect the video stream together. Connect the following:
    a. (TPG) m_axis_video -> stream_in_24 (color convert)
    b. (color convert) stream_out_24 -> stream_in_24 (pixel pack)
    c. (pixel pack) stream_out32 -> S_AXIS_S2MM (VDMA)



31. Connect the **unconnected clock and reset** on the **color convert, pixel pack** and **VDMA** to the **clock and reset** already in the design.



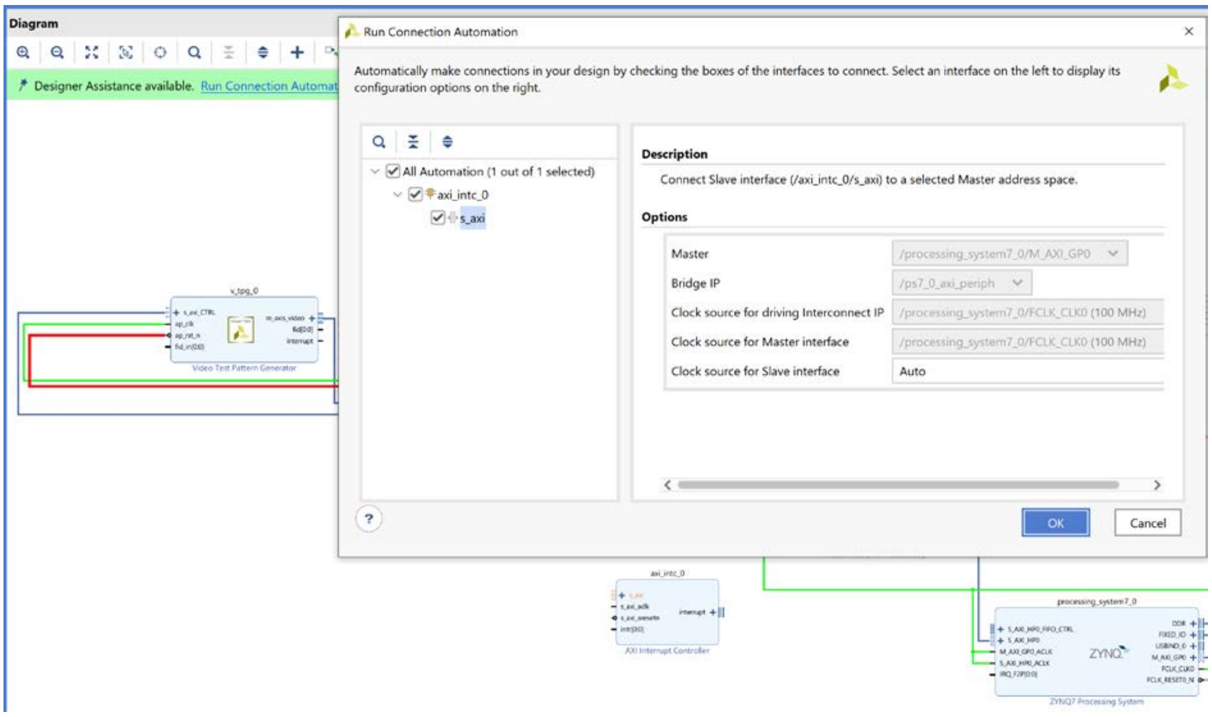
32. Click on the **Address Editor tab** and select **Auto Assign Addresses**.



Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
	s_axi_AXILiteS	Reg	0x43C0_0000	64	0x43C0_FFFF
	s_axi_AXILiteS	Reg	0x43C1_0000	64	0x43C1_FFFF
	S_AXI_LITE	Reg	0x4300_0000	64	0x4300_FFFF
	s_axi_CTRL	Reg	0x43C2_0000	64	0x43C2_FFFF
axi_vdma_0					
Data_MM2S (32 add					
Unmapped Slave					
	S_AXI_HP0	HP0_			
Data_S2MM (32 add					
	S_AXI_HP0	HP0_			

33. Using the **add IP + button**, add in an **AXI interrupt controller** and run the **Connection Automation** to map it into the AXI-Lite network. Click **Ok**.



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

- ☑ All Automation (1 out of 1 selected)
  - ☑ axi_intc_0
    - ☑ s_axi

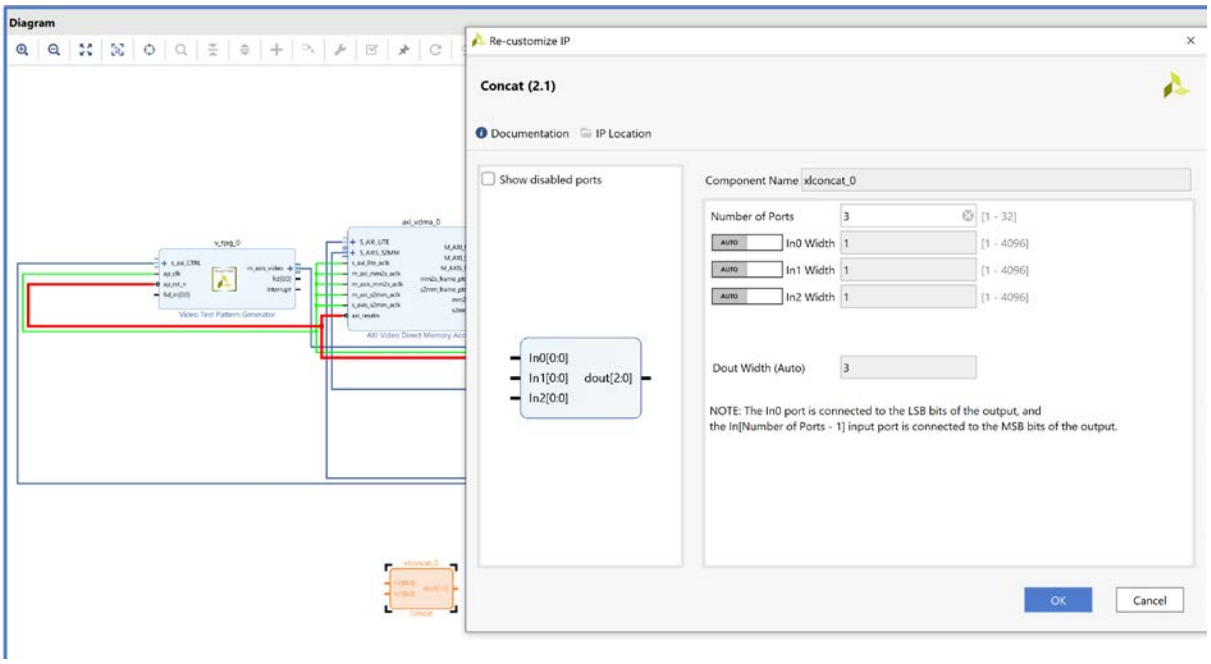
Description

Connect Slave interface (/axi_intc_0/s_axi) to a selected Master address space.

Options

Master	/processing_system7_0/M_AXI_GP0
Bridge IP	/ps7_0_axi_periph
Clock source for driving Interconnect IP	/processing_system7_0/FCLK_CLK0 (100 MHz)
Clock source for Master interface	/processing_system7_0/FCLK_CLK0 (100 MHz)
Clock source for Slave interface	Auto

34. Add in a **Concat** block from the add IP option. Double click and select **3 input ports**. Click **Ok**.



Re-customize IP

Concat (2.1)

Documentation IP Location

Component Name xlconcat_0

Number of Ports 3 [1 - 32]

In0 Width 1 [1 - 4096]

In1 Width 1 [1 - 4096]

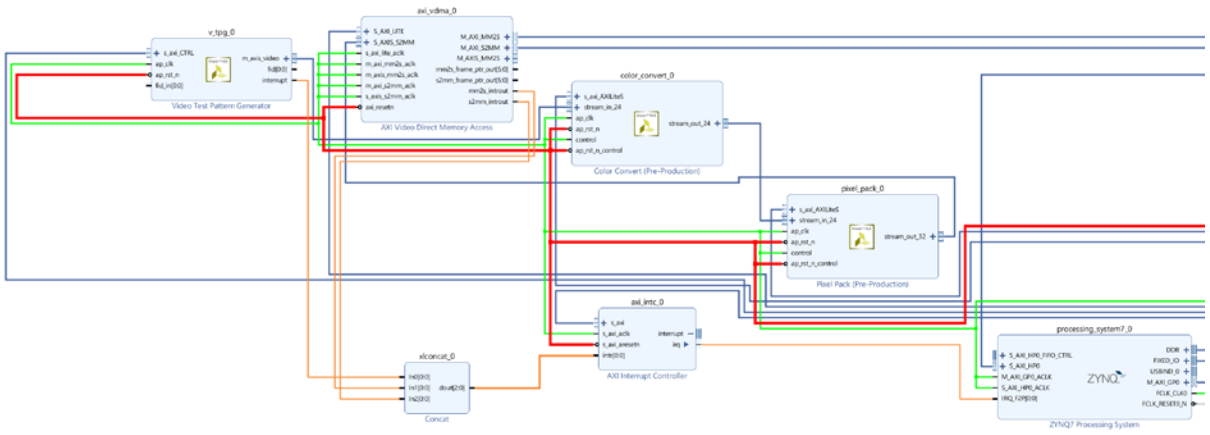
In2 Width 1 [1 - 4096]

Dout Width (Auto) 3

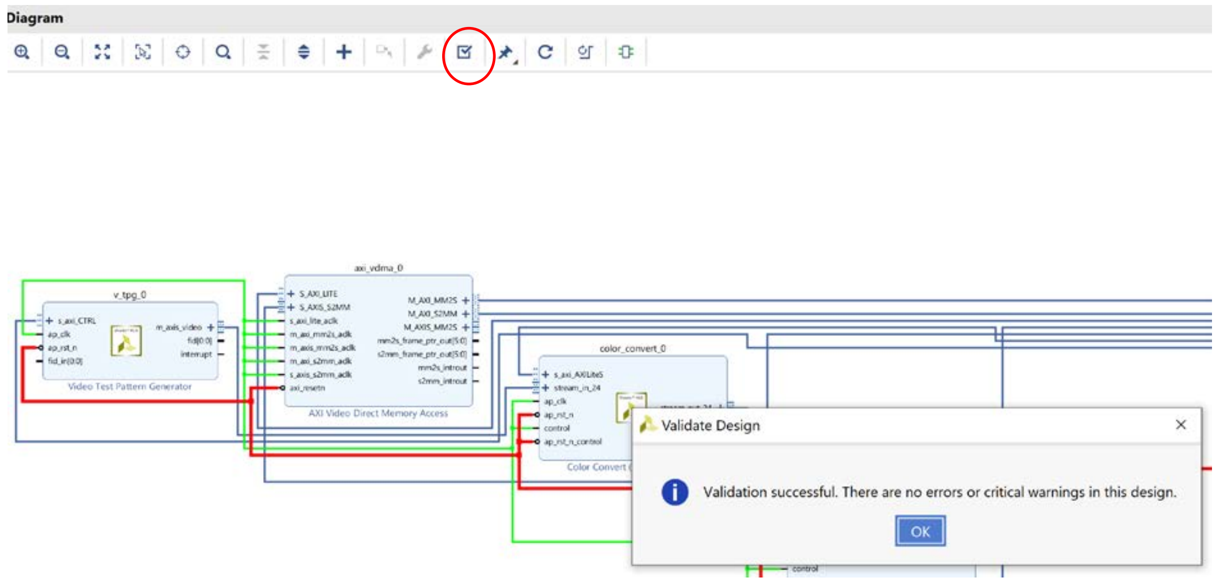
NOTE: The In0 port is connected to the LSB bits of the output, and the In[Number of Ports - 1] input port is connected to the MSB bits of the output.

35. Connect the following signals to the Concat input blocks:
    a. V_tpg Interupt
    b. Mm2s_introut
    c. S2mm_introut

36. Connect the **output** of the **Concat block** to the **interrupt input** on the **AXI interrupt controller** and connect the **output of the AXI interrupt controller** to the **Zynq interrupt input.**

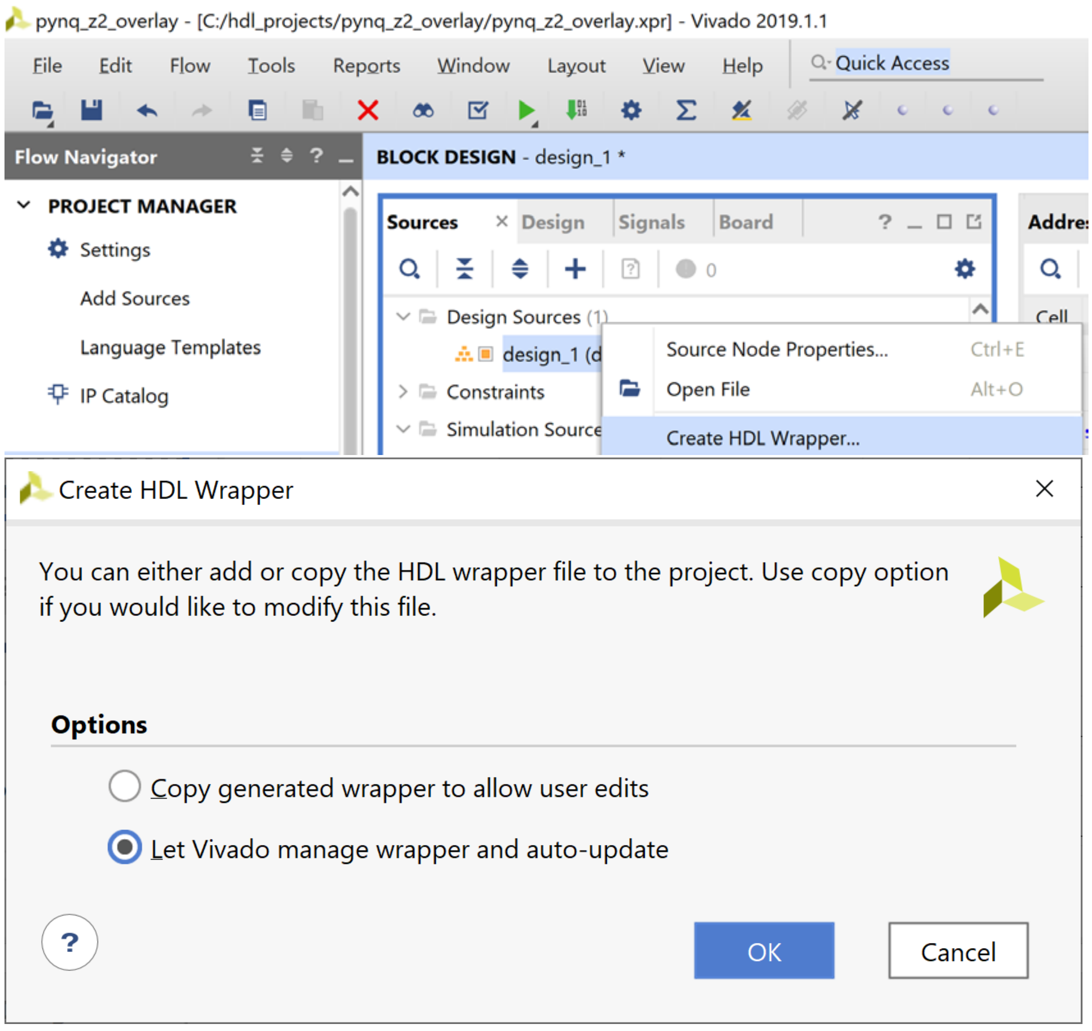


37. Validate the block diagram.



Validation successful. There are no errors or critical warnings in this design.

38. Save and close the block diagram. In the sources window, right click on the block diagram and select **Create HDL Wrapper**. When prompted, let Vivado manage it.



You can either add or copy the HDL wrapper file to the project. Use copy option if you would like to modify this file.

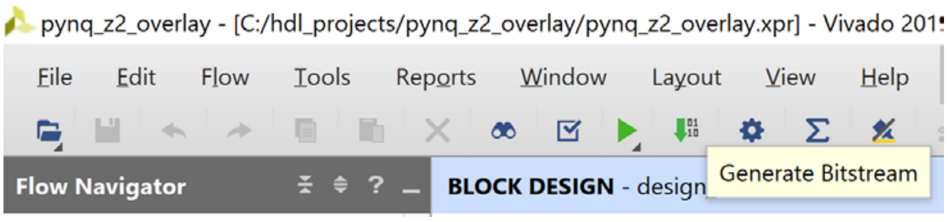
Options

( ) Copy generated wrapper to allow user edits

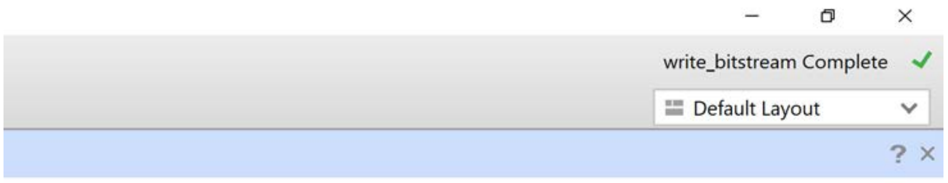
[✓ Let Vivado manage wrapper and auto-update]

Building the Design

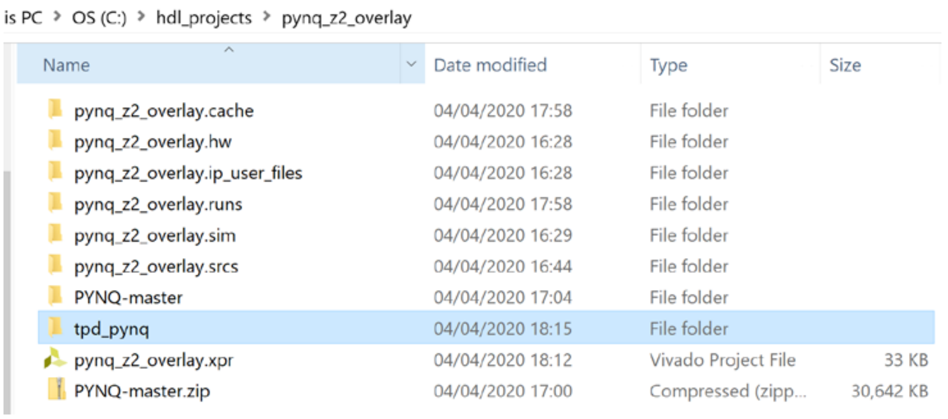
39. We can now build the design. Click on **Generate Bitstream**. This may take a while.



40. Wait until you see **write_bitstream** complete in Vivado.



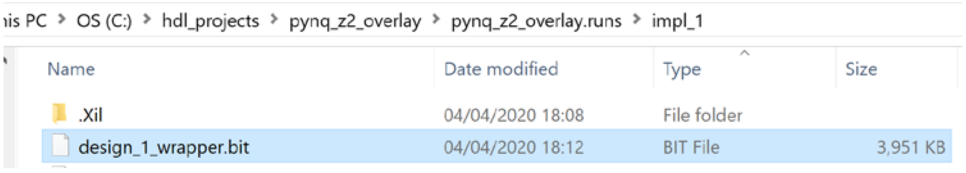
41. In the same directory as your Vivado project, create a new directory called **tpd_pynq**.



is PC › OS (C:) › hdl_projects › pynq_z2_overlay

Name	Date modified	Type	Size
pynq_z2_overlay.cache	04/04/2020 17:58	File folder	
pynq_z2_overlay.hw	04/04/2020 16:28	File folder	
pynq_z2_overlay.ip_user_files	04/04/2020 16:28	File folder	
pynq_z2_overlay.runs	04/04/2020 17:58	File folder	
pynq_z2_overlay.sim	04/04/2020 16:29	File folder	
pynq_z2_overlay.srcs	04/04/2020 16:44	File folder	
PYNQ-master	04/04/2020 17:04	File folder	
tpd_pynq	04/04/2020 18:15	File folder	
pynq_z2_overlay.xpr	04/04/2020 18:12	Vivado Project File	33 KB
PYNQ-master.zip	04/04/2020 17:00	Compressed (zipp...	30,642 KB

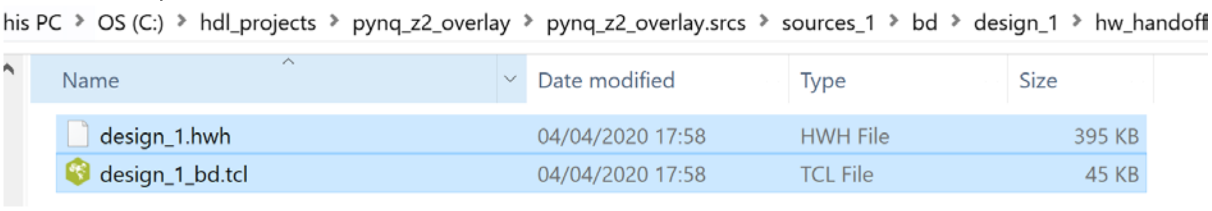
42. From the **runs/implementation directory**, copy across the **BIT** file into the directory you just created.



is PC › OS (C:) › hdl_projects › pynq_z2_overlay › pynq_z2_overlay.runs › impl_1

Name	Date modified	Type	Size
.Xil	04/04/2020 18:08	File folder	
design_1_wrapper.bit	04/04/2020 18:12	BIT File	3,951 KB

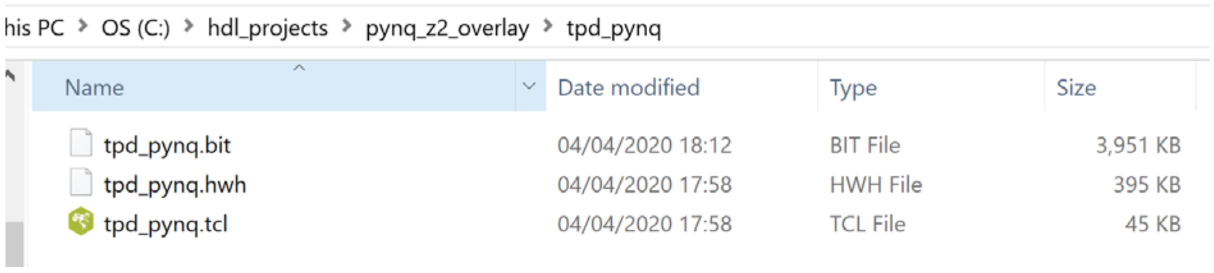
43. From under the source hardware hand off directory, copy across the **tcl** and **hwh** file into the new directory.



his PC › OS (C:) › hdl_projects › pynq_z2_overlay › pynq_z2_overlay.srcs › sources_1 › bd › design_1 › hw_handoff

Name	Date modified	Type	Size
design_1.hwh	04/04/2020 17:58	HWH File	395 KB
design_1_bd.tcl	04/04/2020 17:58	TCL File	45 KB

44. Rename the files in the tpd_pynq directory as **tpd_pynq.*** ** (e.g. tpd_pynq.bit)



his PC › OS (C:) › hdl_projects › pynq_z2_overlay › tpd_pynq

Name	Date modified	Type	Size
tpd_pynq.bit	04/04/2020 18:12	BIT File	3,951 KB
tpd_pynq.hwh	04/04/2020 17:58	HWH File	395 KB
tpd_pynq.tcl	04/04/2020 17:58	TCL File	45 KB

45. Create a new file in the tpd_pynq directory called **tpg_pynq.py** and copy in the code below.

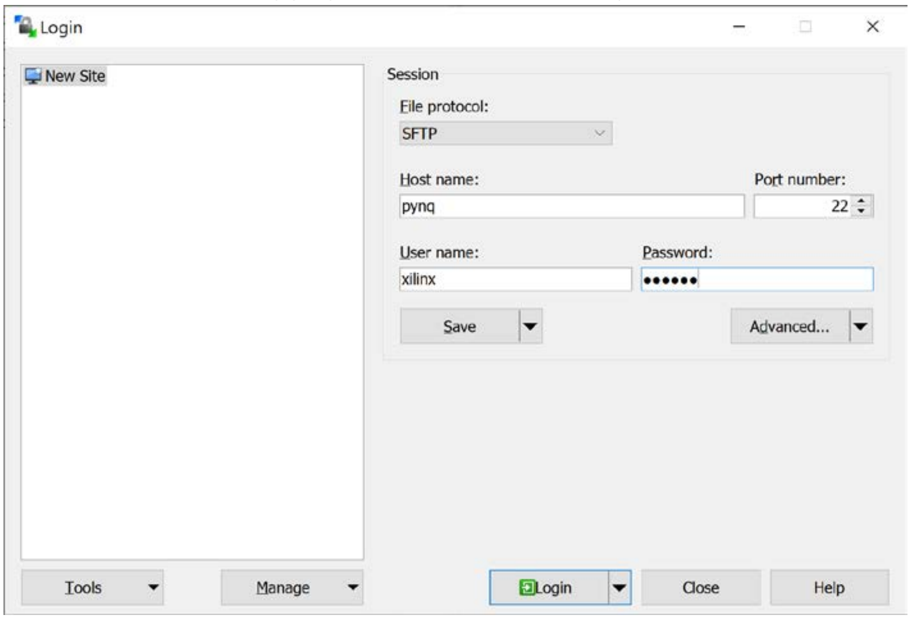
```
import pynq
from pynq import GPIO

__author__ = "Adam Taylor"
__copyright__ = "Copyright 2020, Adiuvo"
__email__ = "Adam@adiuvoengineering.com"


class tpd_pynqOverlay(pynq.Overlay):
    """.
    """
    def __init__(self, bitfile, **kwargs):
        super().__init__(bitfile, **kwargs)
        if self.is_loaded():
            pass
```

46. Create a new file in the tpd_pynq directory called **__init__.py**. Copy in the code below.

```
from .tpd_pynq import tpd_pynqOverlay
```

Uploading to the PYNQ-Z2

47. We are now ready to upload the overlay to our PYNQ-Z2. Power up the board and open **WinSCP**. Use the host name of **pynq** and the username and password of **xilinx**.


Login

Session

File protocol: SFTP

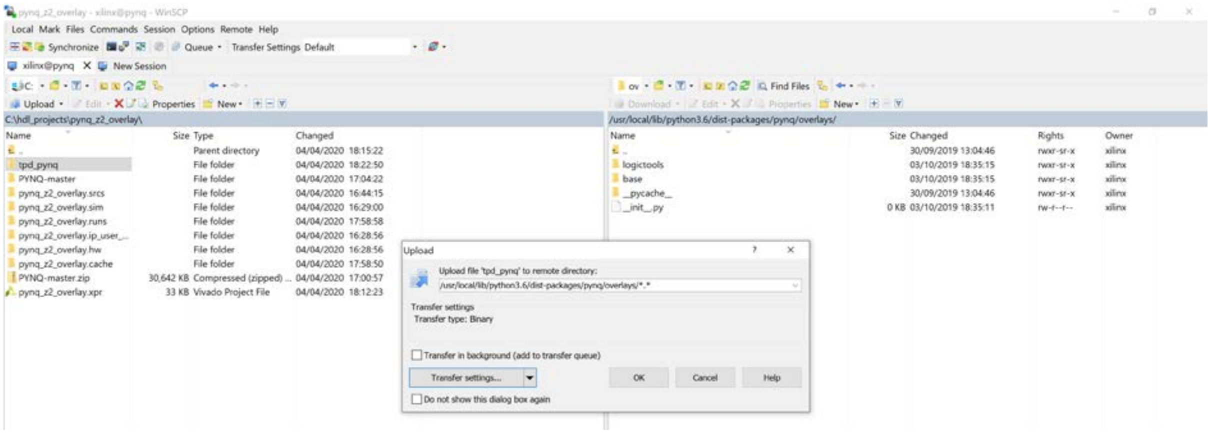
Host name: pynq

Port number: 22

User name: xilinx

Password: ●●●●●●

48. In WinSCP on your computer, see the left-hand side and select the **tpd_pynq** directory on the target. Navigate to the **pynq/overlays**. Upload the **tpd_pynq** directory.


pynq_z2_overlay - xilinx@pynq - WinSCP

C:\hdl_projects\pynq_z2_overlay\

Name	Size	Type	Changed
..		Parent directory	04/04/2020 18:15:22
tpd_pynq		File folder	04/04/2020 18:22:50
PYNQ-master		File folder	04/04/2020 17:04:22
pynq_z2_overlay.srcs		File folder	04/04/2020 16:44:15
pynq_z2_overlay.sim		File folder	04/04/2020 16:29:00
pynq_z2_overlay.runs		File folder	04/04/2020 17:58:58
pynq_z2_overlay.ip_user_...		File folder	04/04/2020 16:28:56
pynq_z2_overlay.hw		File folder	04/04/2020 16:28:56
pynq_z2_overlay.cache		File folder	04/04/2020 17:58:50
PYNQ-master.zip	30,642 KB	Compressed (zipped) ...	04/04/2020 17:00:57
pynq_z2_overlay.xpr	33 KB	Vivado Project File	04/04/2020 18:12:23

/usr/local/lib/python3.6/dist-packages/pynq/overlays/

Name	Size	Changed	Rights	Owner
..		30/09/2019 13:04:46	rwxr-sr-x	xilinx
logictools		03/10/2019 18:35:15	rwxr-sr-x	xilinx
base		03/10/2019 18:35:15	rwxr-sr-x	xilinx
__pycache__		30/09/2019 13:04:46	rwxr-sr-x	xilinx
__init__.py	0 KB	03/10/2019 18:35:11	rw-r--r--	xilinx

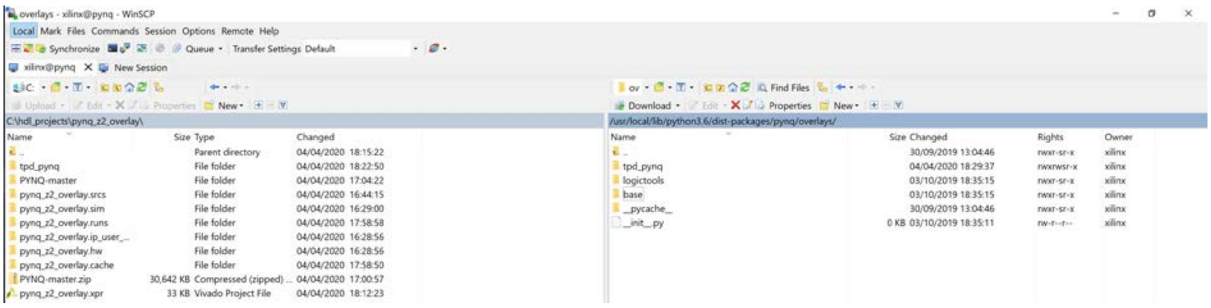
Upload

Upload file 'tpd_pynq' to remote directory:
/usr/local/lib/python3.6/dist-packages/pynq/overlays/*.*

Transfer settings
Transfer type: Binary

☐ Transfer in background (add to transfer queue)

☐ Do not show this dialog box again


overlays - xilinx@pynq - WinSCP

C:\hdl_projects\pynq_z2_overlay\

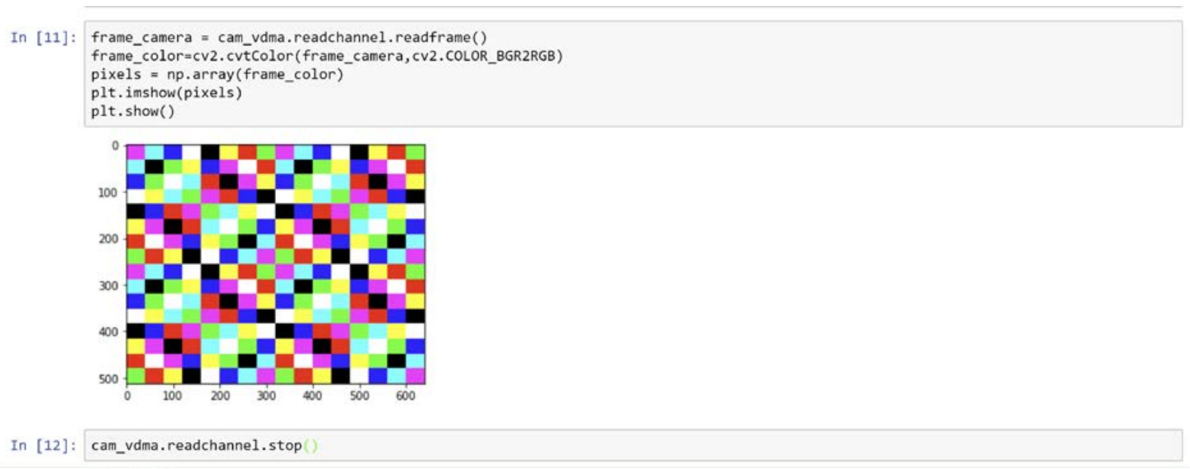
Name	Size	Type	Changed
..		Parent directory	04/04/2020 18:15:22
tpd_pynq		File folder	04/04/2020 18:22:50
PYNQ-master		File folder	04/04/2020 17:04:22
pynq_z2_overlay.srcs		File folder	04/04/2020 16:44:15
pynq_z2_overlay.sim		File folder	04/04/2020 16:29:00
pynq_z2_overlay.runs		File folder	04/04/2020 17:58:58
pynq_z2_overlay.ip_user_...		File folder	04/04/2020 16:28:56
pynq_z2_overlay.hw		File folder	04/04/2020 16:28:56
pynq_z2_overlay.cache		File folder	04/04/2020 17:58:50
PYNQ-master.zip	30,642 KB	Compressed (zipped) ...	04/04/2020 17:00:57
pynq_z2_overlay.xpr	33 KB	Vivado Project File	04/04/2020 18:12:23

/usr/local/lib/python3.6/dist-packages/pynq/overlays/

Name	Size	Changed	Rights	Owner
..		30/09/2019 13:04:46	rwxr-sr-x	xilinx
tpd_pynq		04/04/2020 18:29:37	rwxrwsr-x	xilinx
logictools		03/10/2019 18:35:15	rwxr-sr-x	xilinx
base		03/10/2019 18:35:15	rwxr-sr-x	xilinx
__pycache__		30/09/2019 13:04:46	rwxr-sr-x	xilinx
__init__.py	0 KB	03/10/2019 18:35:11	rw-r--r--	xilinx

49. Using a browser, open the **Jupyter environment** and create a new Python 3 notebook called **tpd.ipynb** on the homepage. Add in the code below.

```
import time
from pynq.overlays.tpd_pynq import tpd_pynqOverlay
import numpy as np
from pynq import pl
from pynq import overlay
from pynq.lib.video import *
from pynq import Xlnk
import cv2
import matplotlib.pyplot as plt
overlay = tpd_pynqOverlay('tpd_pynq.bit')

pixel_in = overlay.pixel_pack_0
pixel_in.bits_per_pixel = 24

colourspace_in = overlay.color_convert_0
rgb2bgr = [0.0, 1.0, 0.0,
        1.0, 0.0, 0.0,
        0.0, 0.0, 1.0,
        0.0, 0.0, 0.0]

colourspace_in.colorspace = rgb2bgr

cam_vdma = overlay.axi_vdma_0
lines = 512
framemode = VideoMode(640, lines, 24)
cam_vdma.readchannel.mode = framemode
cam_vdma.readchannel.start()

tpg = overlay.v_tpg_0
tpg.write(0x10,512)
tpg.write(0x18,640)
tpg.write(0x40,0)
tpg.write(0x30,0)
tpg.write(0x20,0xB)
tpg.write(0x00,0x81)

frame_camera = cam_vdma.readchannel.readframe()
frame_color=cv2.cvtColor(frame_camera,cv2.COLOR_BGR2RGB)
pixels = np.array(frame_color)
plt.imshow(pixels)
plt.show()

cam_vdma.readchannel.stop()
```



```
In [11]: frame_camera = cam_vdma.readchannel.readframe()
         frame_color=cv2.cvtColor(frame_camera,cv2.COLOR_BGR2RGB)
         pixels = np.array(frame_color)
         plt.imshow(pixels)
         plt.show()
```

```
In [12]: cam_vdma.readchannel.stop()
```

Now we know how to build our own overlay and work within PYNQ. In the next lab we will be updating this design to create a simple image processing application.