



Signal Processing with FPGA, Python & no RTL Design!

Adam Taylor

Adam@AdiuvoEngineering.com

Let's Use an FPGA



Project Managers Despair

- Timescales Increasing
- Costs Exploding
- Difficult conversations with engineering
- Even more difficult conversations with senior management.

Of course, it does not have to be this way!

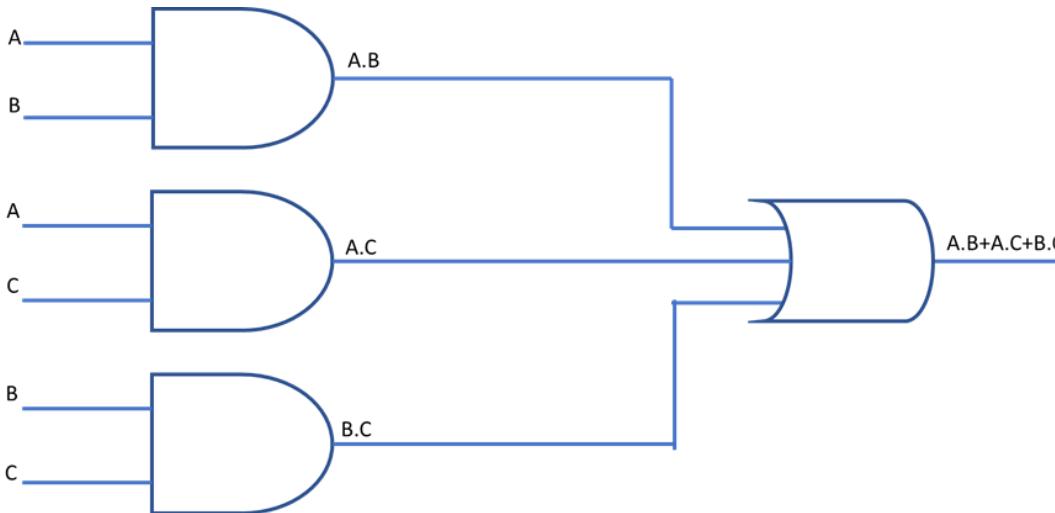


FPGA

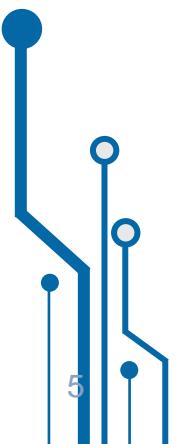
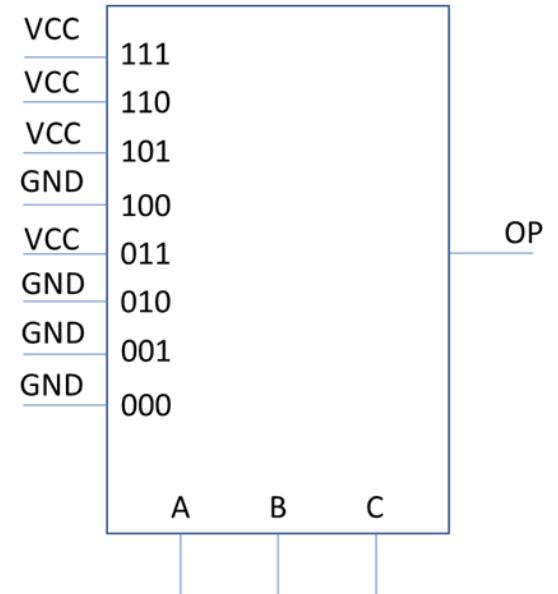
FPGA Architecture



What is programmable logic



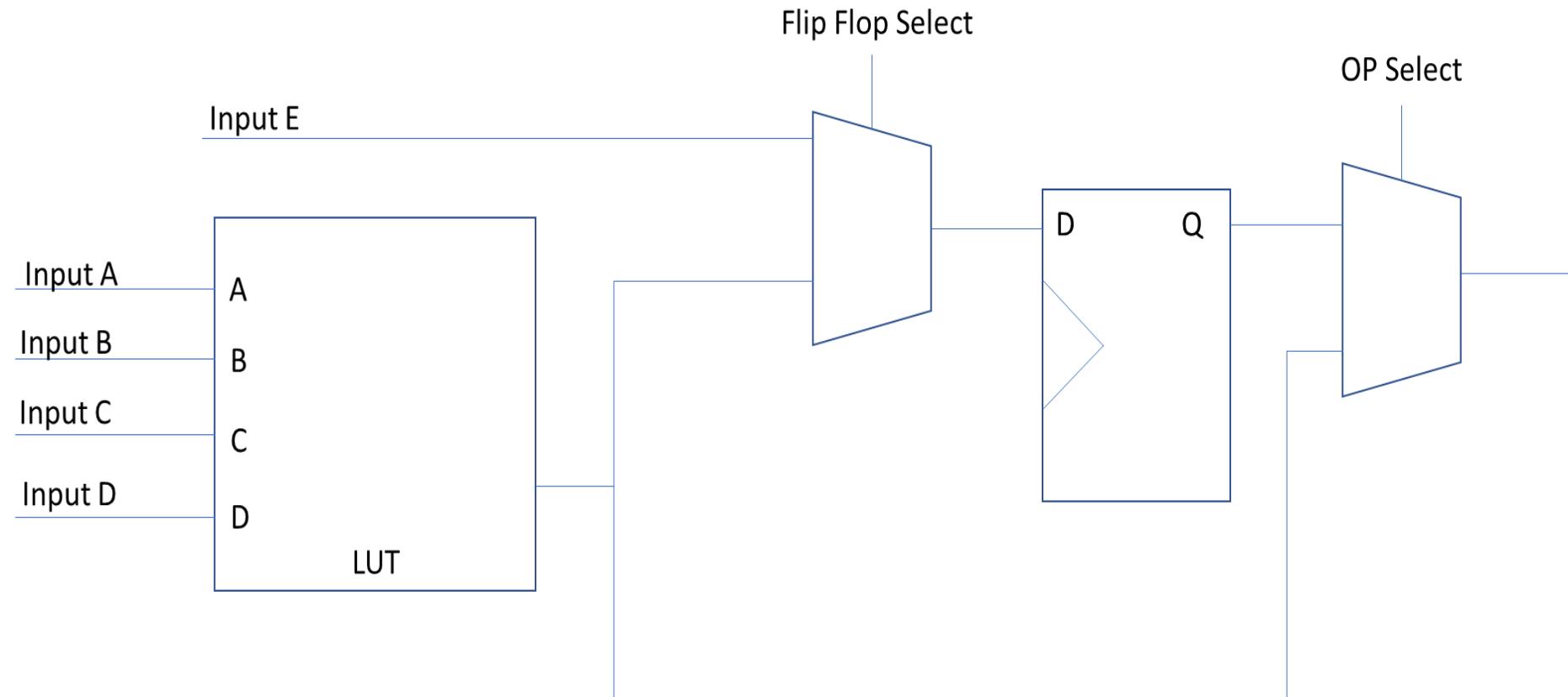
| Input A | Input B | Input C | OP |
|---------|---------|---------|----|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



Configurable Logic Block

- LUT allows implementation of Combinatorial Circuit
- BUT we design synchronous circuits what about the FF
- flip flop to act as storage for the combinatorial output such that we can implement sequential structures
- This combination of a LUT and Flip Flop is often called a Configurable Logic Block (CLB) and a programmable logic device will consist of many thousands of these CLBs. To provide the most flexibility additional multiplexers will be used to support a wide range of CLB configurations.

Configurable Logic Block

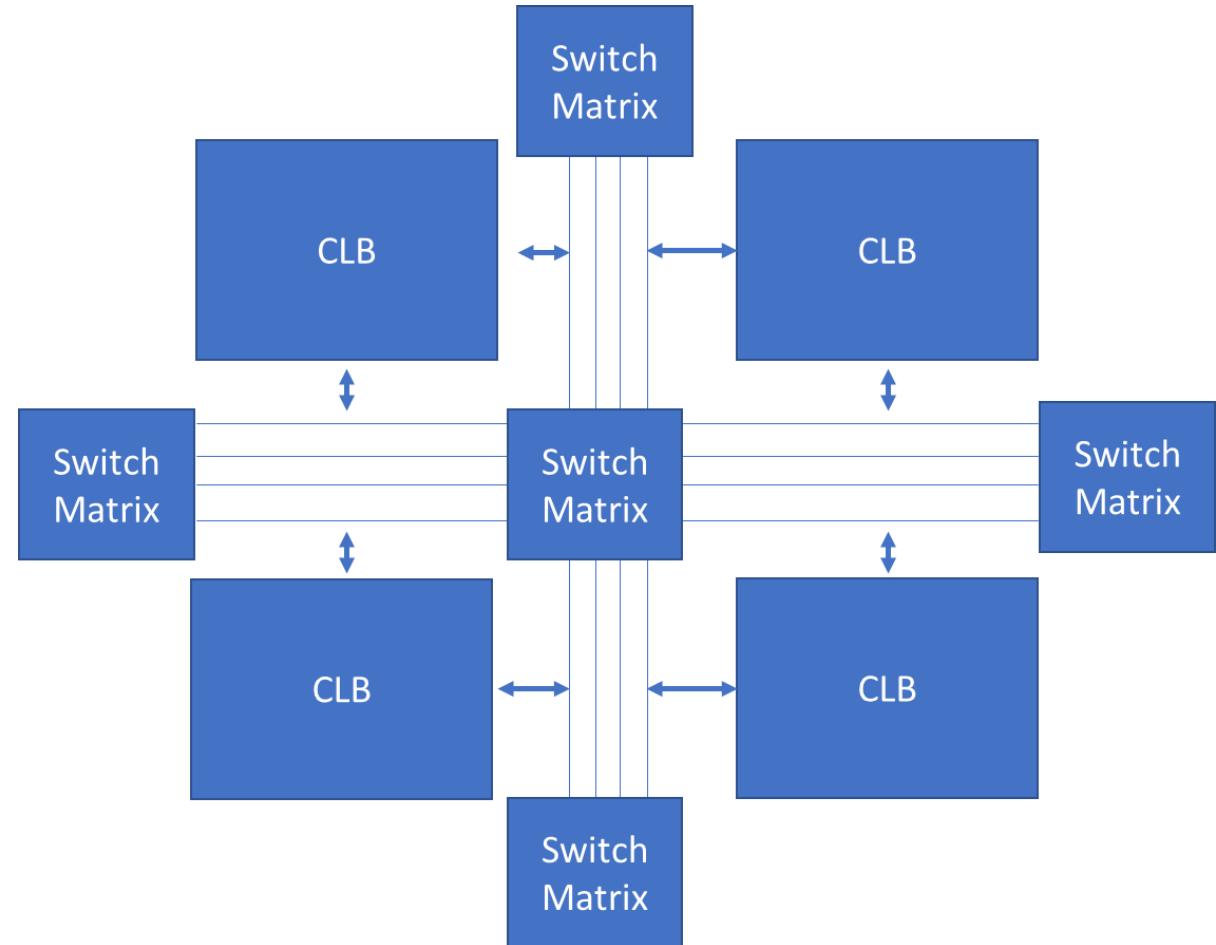


Interconnect Routing

A collection of wires and programmable switches.

These are responsible for connecting CLBs and other building blocks within the FPGA.

These are also called routing channels.





Heterogeneous SoC

Heterogeneous SOC

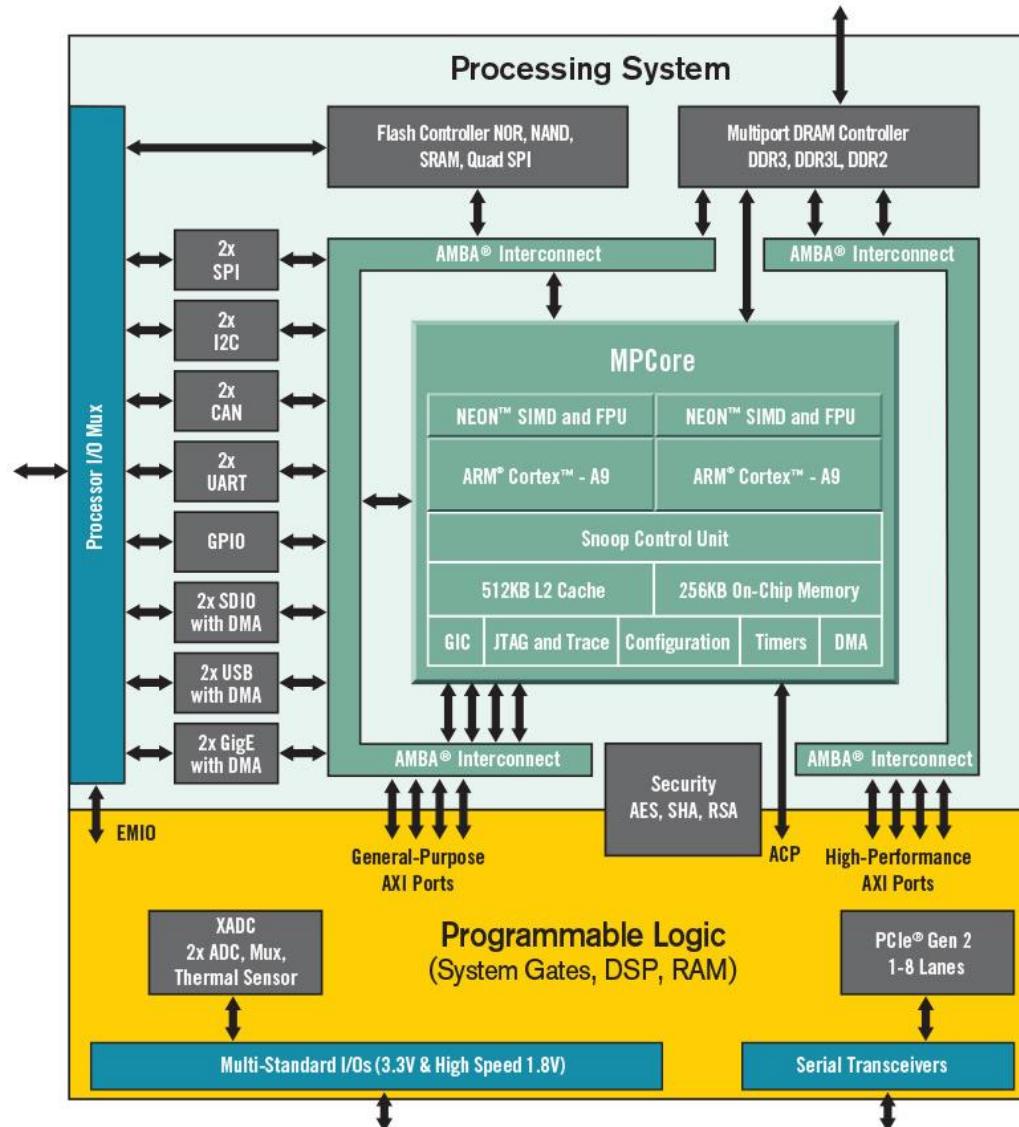
Combine diverse processing elements with programmable logic

Processing elements are silicon implementations

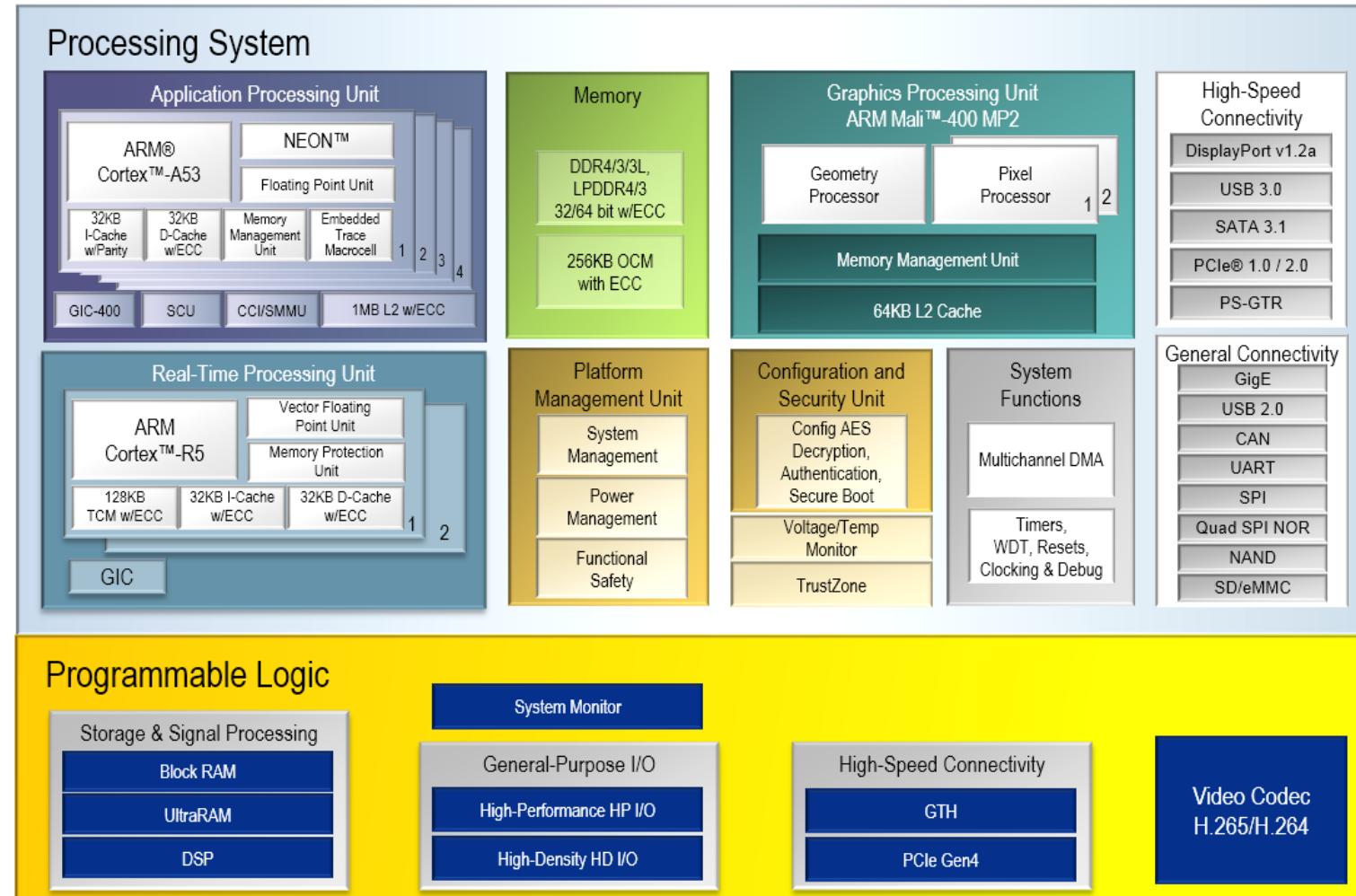
Processing element is the master – it boots just like any other processor

Enables highly optimized solutions be implemented using processor and logic to exploit natural strengths

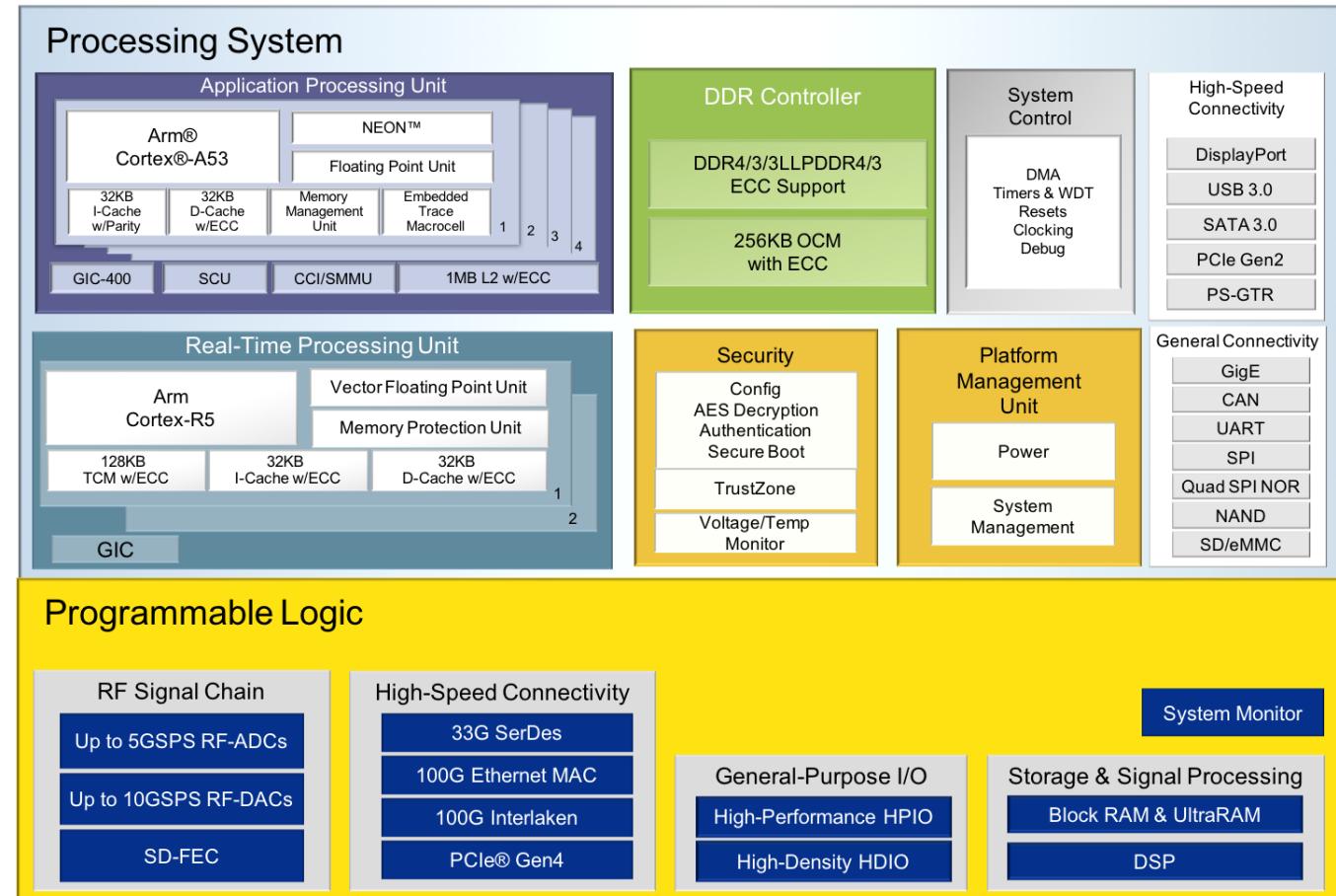
Zynq



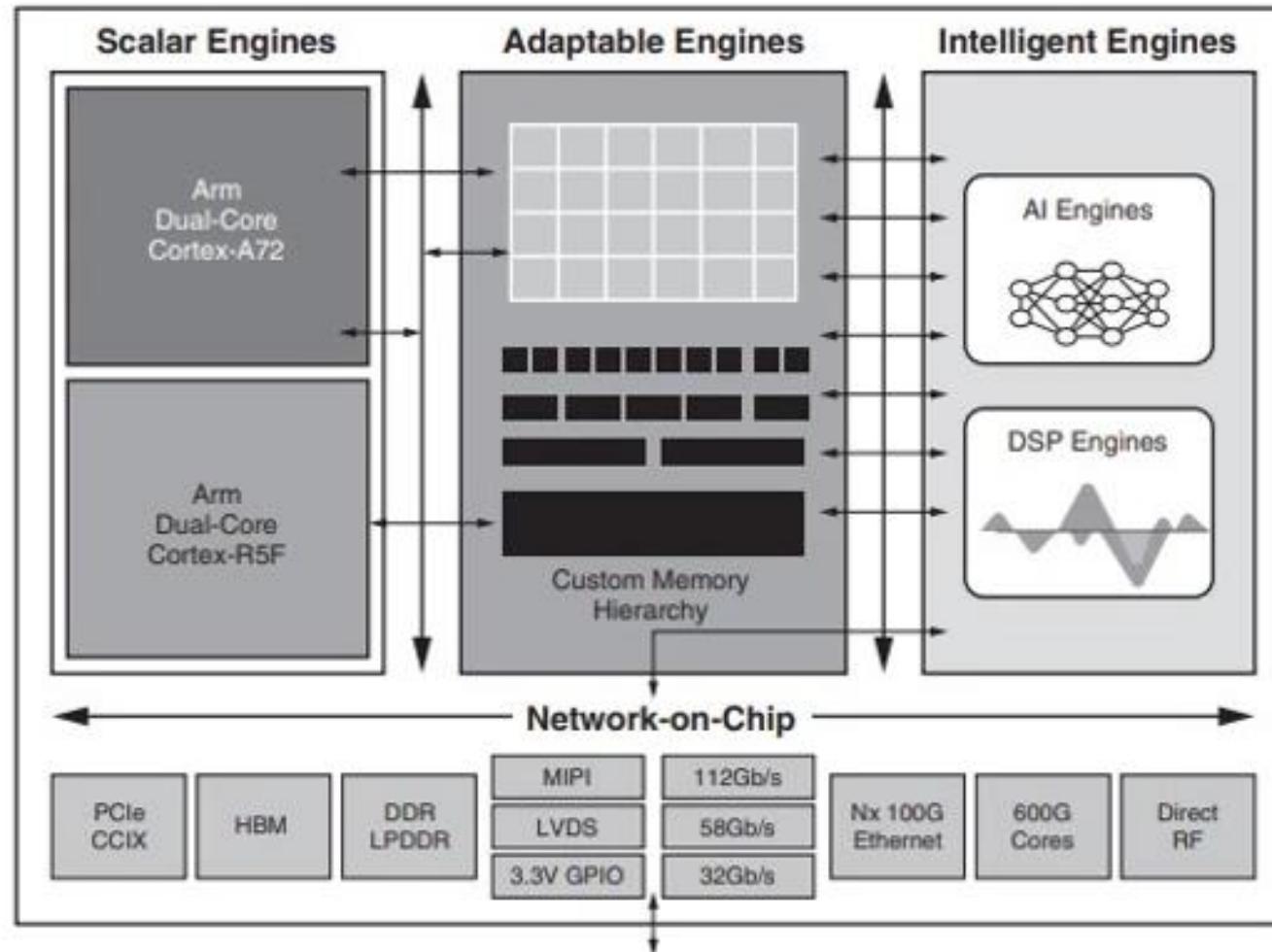
Zynq MPSoC



Zynq RFSoC

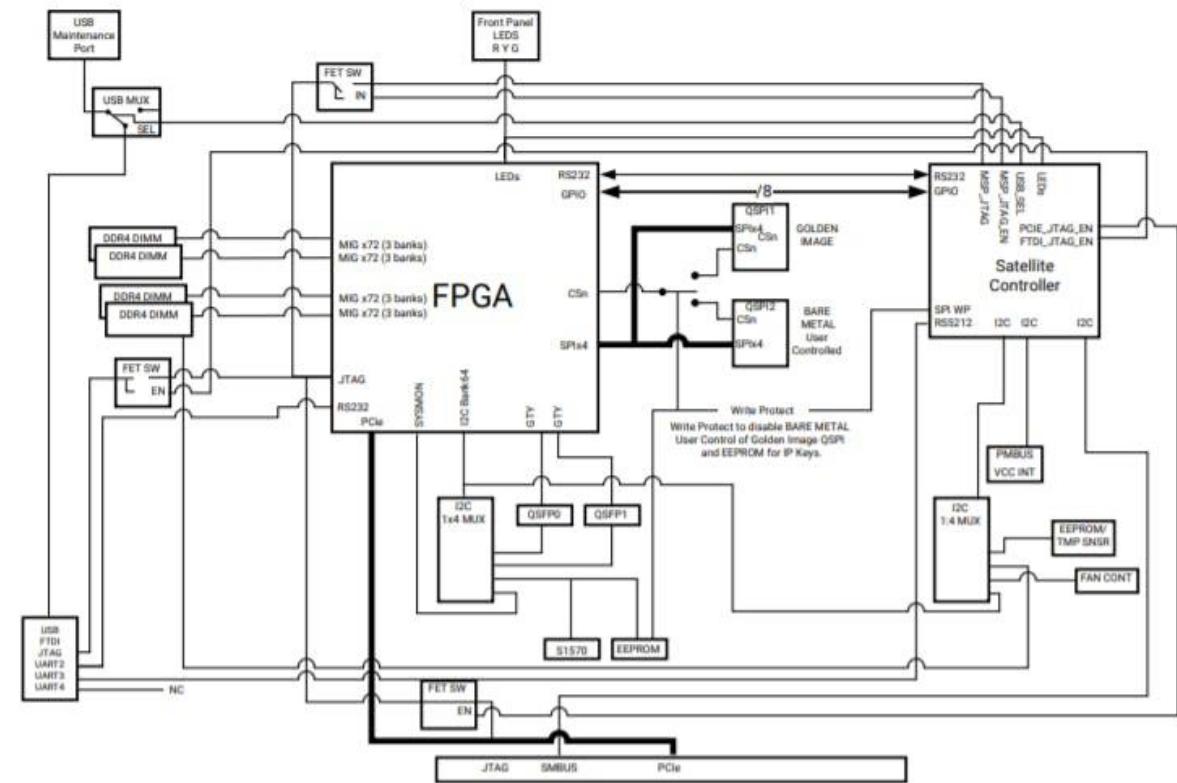
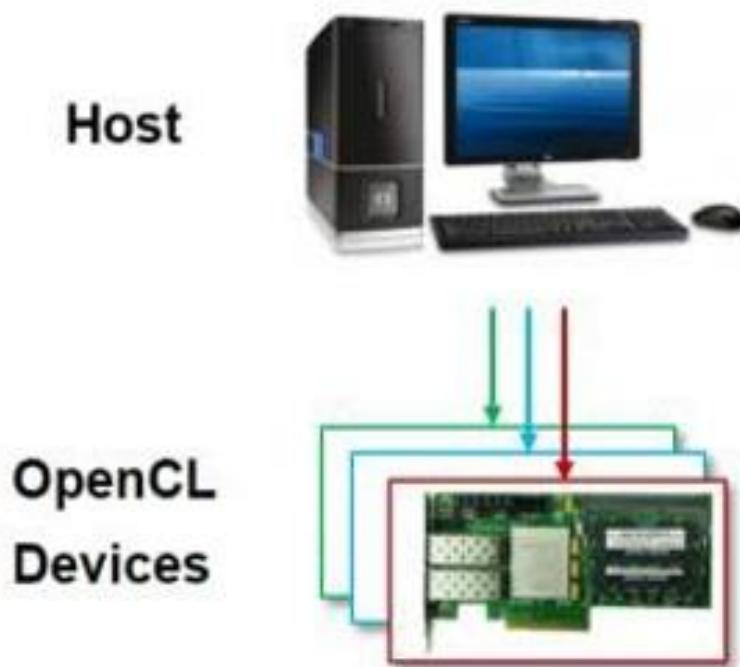


Versal



Alveo

Accelerator card based around OpenCL





Tool Chain

Xilinx Tools and Frameworks

VitisAI

Enables the implementations of machine learning inference, using Tensor Flow, Caffe and PyTorch.

*Requires a SoC/RFSoC/Alveo

Vitis

Embedded and accelerated SW development. Used to develop software solutions for MicroBlaze, Arm R5, A9, A53 and A72.

Petalinux

Embedded Linux solutions

PYNQ

Python framework for rapid prototyping on SoC/RFSoC/Alveo

Vitis / Vivado HLS

High Level Synthesis tool supporting C/C++/OpenCL

Vivado

Design Capture and implementation for the base platform

Vivado Overview

- Foundation of all design and higher-level tools is Vivado
- Vivado enables us to capture designs using VHDL or Verilog
- Large IP library to accelerate our designs
- Integrates Vivado / Vitis HLS IP cores



- ✓ Simulate designs using Vivado Simulator
- ✓ Synthesize, place and route the design
- ✓ Generate power estimations
- ✓ Create Xilinx Support Architecture

Vivado Overview

FPGA Implementation Flow

Synthesis – Translates the HDL design into a series of logic equations which are then mapped onto the resources available in the target FPGA.



Synthesis – Place – The logic resources determined by the synthesis tool are placed at available locations within the target device.



Routing – The placed logic resources in the design are interconnected using routing and switch matrixes to implement the final application.



Bit File – The generation of the final programming file for the target FPGA.



We can control the flow implementation settings by using Constraints (XDC file) and implementation Strategies

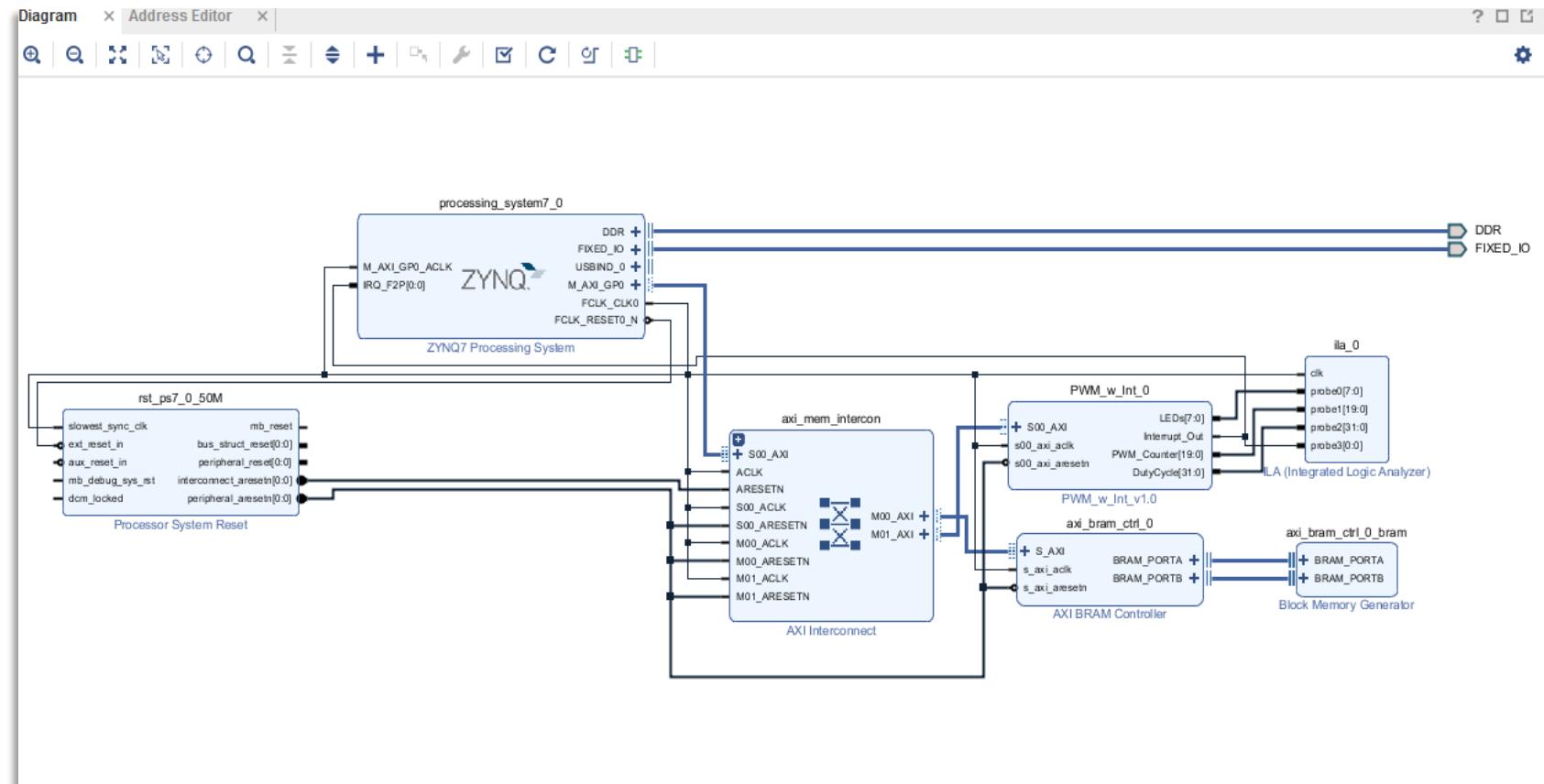


Vivado IP Integrator

Vivado IP Integrator (IPI)

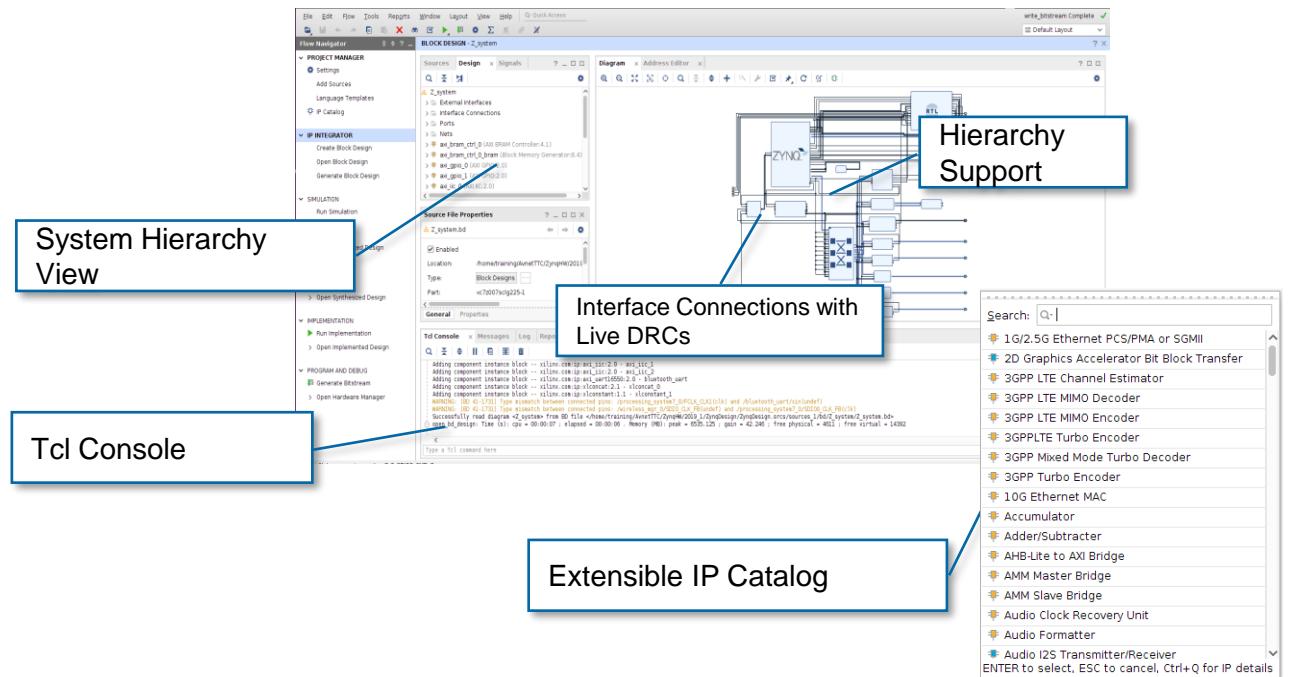
Create system-level designs

- Instantiate and interconnect IP cores
- IP-centric design flow
 - Plug-and-play IP
 - Vast IP catalog
- Accelerates
 - Integration
 - Productivity
- Example applications
 - Embedded
 - DSP
 - Video
 - Analog
 - Networking



Vivado IP Integrator: Intelligent IP Integration

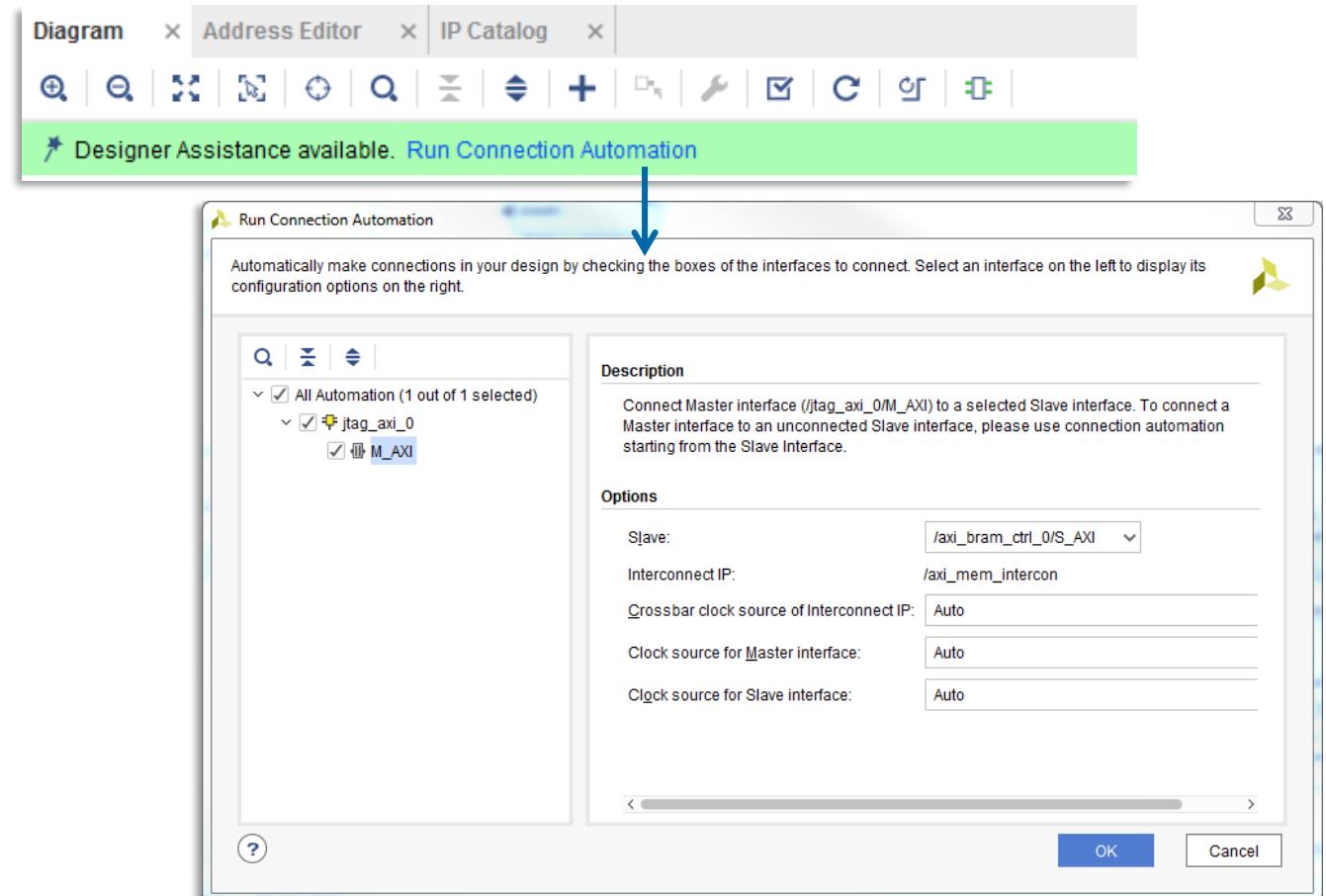
- Automated IP subsystems
- Block automation for rapid design creation
- One click IP customization
- Board aware
- Support all 7 Series FPGAs and Zynq SoCs
- Built-in presets, accelerating design creation



Vivado IP Integrator: Intelligent IP Integration

Correct-by-construction

- Interface level connections
- Extensible IP repository
- Real-time DRCs and parameter propagation / resolution
- Designer assistance



IP Integrator Library

| Category | IP Name | Interface | Production Status | Purchase Status | Link |
|------------------|------------------------------------|-------------------|-------------------|-----------------|-----------|
| Error Correction | 3GPPLTE Turbo Encoder | | Production | Purchase | xilinx.cc |
| | 3GPP Mixed Mode Turbo Decoder | AXI4-Stream | Production | Purchase | xilinx.cc |
| | 3GPP Turbo Encoder | | Production | Purchase | xilinx.cc |
| | Convolution Encoder | AXI4-Stream | Production | Included | xilinx.cc |
| | Fibre Channel 32GFC RS-FEC | | | Purchase | xilinx.cc |
| | FlexO 100G RS-FEC | AXI4 | | Purchase | xilinx.cc |
| | G.709 FEC Encoder/Decoder | AXI4, AXI4-Stream | Not-Supported | Purchase | xilinx.cc |
| | G.709 FEC Encoder/Decoder | AXI4, AXI4-Stream | Not-Supported | Purchase | xilinx.cc |
| | G.975.1 EFEC I.4 Encoder/Decoder | AXI4-Stream | Not-Supported | Purchase | xilinx.cc |
| | G.975.1 EFEC I.7 Encoder/Decoder | AXI4-Stream | Not-Supported | Purchase | xilinx.cc |
| | IEEE 802.3 25G RS-FEC | AXI4 | | Purchase | xilinx.cc |
| | IEEE 802.3 50G RS-FEC | AXI4 | | Purchase | xilinx.cc |
| | IEEE 802.3 50G RS-FEC | AXI4 | | Purchase | xilinx.cc |
| | IEEE 802.3 400G RS-FEC | | | Purchase | xilinx.cc |
| | IEEE 802.3bj 100G RS-FEC | AXI4 | | Purchase | xilinx.cc |
| | IEEE 802.3bj 100G RS-FEC | AXI4 | | Purchase | xilinx.cc |
| | IEEE 802.3 Clause 74 FEC | | | Purchase | xilinx.cc |
| | IEEE 802.3 Multi-channel 25G RSFEC | | | Purchase | xilinx.cc |
| | Interleaver/De-interleaver | AXI4-Stream | Production | Purchase | xilinx.cc |
| | LDPC Encoder/Decoder | AXI4, AXI4-Stream | Production | Purchase | xilinx.cc |
| | Polar Encoder/Decoder | AXI4, AXI4-Stream | Production | Purchase | xilinx.cc |
| | Reed-Solomon Decoder | AXI4-Stream | Production | Purchase | xilinx.cc |
| | Reed-Solomon Encoder | AXI4-Stream | Production | Purchase | xilinx.cc |
| | Soft-Decision FEC | AXI4, AXI4-Stream | | Purchase | xilinx.cc |
| | Viterbi Decoder | AXI4-Stream | Production | Purchase | xilinx.cc |

IP Integrator Library

| Category | IP Name | Interface | Status | License | Link |
|----------|--|-------------------|------------|----------|----------------------------|
| Wireless | 3GPP LTE Channel Estimator | AXI4-Stream | Production | Purchase | xilinx.com |
| | 3GPP LTE MIMO Decoder | | Production | Purchase | xilinx.com |
| | 3GPP LTE MIMO Encoder | AXI4-Stream | Production | Purchase | xilinx.com |
| | CPRI | AXI4, AXI4-Stream | | Purchase | xilinx.com |
| | DUC/DDC Compiler | AXI4-Stream | Production | Included | xilinx.com |
| | JESD204 | AXI4, AXI4-Stream | | Purchase | xilinx.com |
| | JESD204C | AXI4, AXI4-Stream | | Purchase | xilinx.com |
| | JESD204 PHY | AXI4 | | Included | xilinx.com |
| | LDPC Encoder/Decoder | AXI4, AXI4-Stream | Production | Purchase | xilinx.com |
| | LTE DL Channel Encoder | AXI4, AXI4-Stream | Production | Purchase | xilinx.com |
| | LTE Fast Fourier Transform | | Production | Purchase | xilinx.com |
| | LTE PUCCH Receiver | AXI4-Stream | Production | Purchase | xilinx.com |
| | LTE UL Channel Decoder | AXI4, AXI4-Stream | Production | Purchase | xilinx.com |
| | Peak Cancellation Crest Factor Reduction | AXI4, AXI4-Stream | Production | Purchase | xilinx.com |
| | Polar Encoder/Decoder | AXI4, AXI4-Stream | Production | Purchase | xilinx.com |
| | Serial RapidIO Gen2 | AXI4, AXI4-Stream | | Purchase | xilinx.com |
| | Soft-Decision FEC | AXI4, AXI4-Stream | | Purchase | xilinx.com |

IP Integrator Library

| Digital Signal Processing | | | | |
|----------------------------|-------------|------------|----------|------------|
| Building Blocks | | | | |
| Complex Multiplier | AXI4-Stream | Production | Included | xilinx.com |
| CORDIC | AXI4-Stream | Production | Included | xilinx.com |
| Filters | | | | |
| CIC Compiler | AXI4-Stream | Production | Included | xilinx.com |
| DUC/DDC Compiler | AXI4-Stream | Production | Included | xilinx.com |
| FIR Compiler | AXI4-Stream | Production | Included | xilinx.com |
| Modulation | | | | |
| DDS Compiler | AXI4-Stream | Production | Included | xilinx.com |
| Transforms | | | | |
| DFTs | | | | |
| Discrete Fourier Transform | | Production | Included | xilinx.com |
| FFTs | | | | |
| Fast Fourier Transform | AXI4-Stream | Production | Included | xilinx.com |
| LTE Fast Fourier Transform | | Production | Purchase | xilinx.com |
| Trig Functions | | | | |
| CORDIC | AXI4-Stream | Production | Included | xilinx.com |
| DDS Compiler | AXI4-Stream | Production | Included | xilinx.com |
| Waveform Synthesis | | | | |
| DDS Compiler | AXI4-Stream | Production | Included | xilinx.com |



What is PYNQ – Introduction to PYNQ framework

What is PYNQ?

PYNQ is an open source project started by Xilinx to increase productivity of Python with the acceleration provided by the Zynq / Zynq MPSoC

Hosted at [PYNQ.io](https://pynq.io)



Why should I learn PYNQ ?

Tight coupling of processing system (PS) and programmable logic (PL) in the Zynq / Zynq MPSoC creates a system which is

- » Responsive – Leverage the parallel processing capability provided by the PL
- » Deterministic – Creates processing pipeline competing for fewer shared resources
- » Power Efficient – Less off chip transactions to or from DDR memory, dedicated hardware implementation is more efficient than

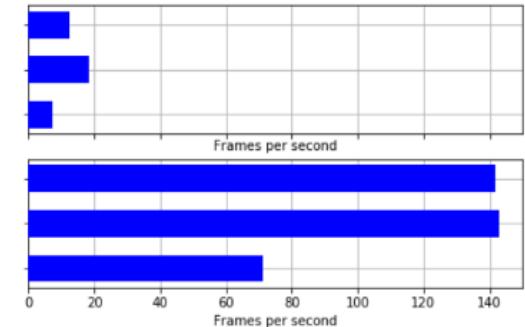
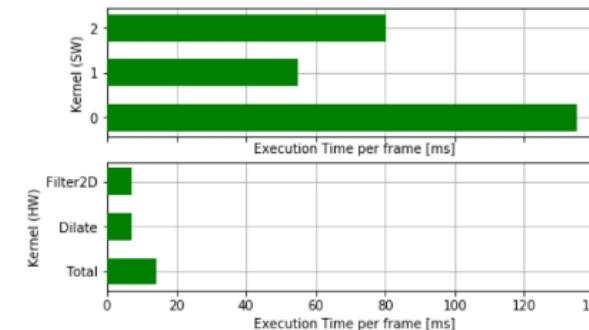
PYNQ frees Python programmers from the sequential software world and opens up the acceleration of programmable logic without the need to be a digital designer.

Why should I learn PYNQ ?

Simple example of PYNQ in a image processing application

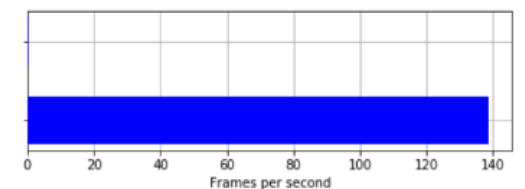
» Image Filtering

- SW < 20 Frames per Second
- HW > 60 Frames per Second

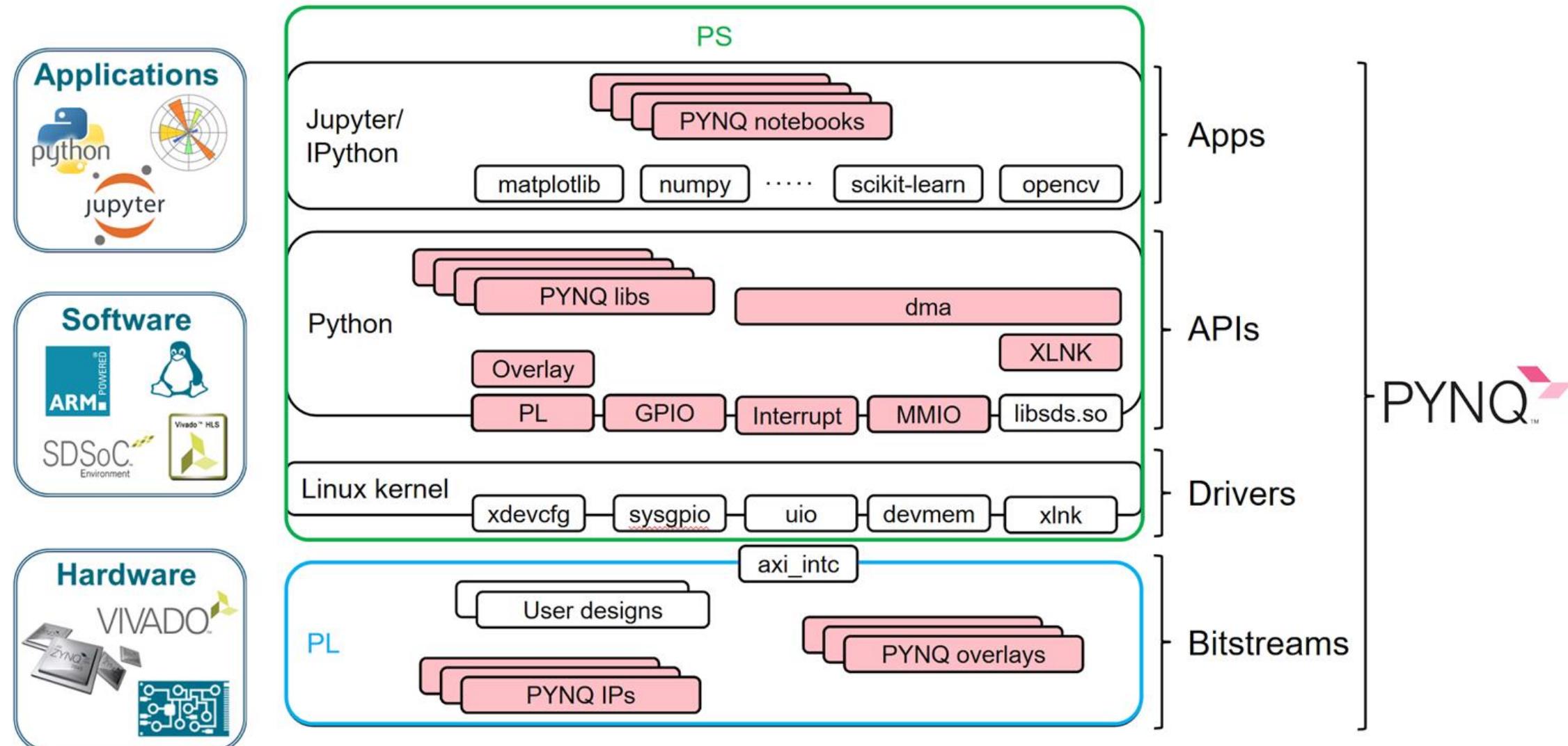


» Optical Flow

- SW < 1 Frame per Second
- HW > 120 Frames per Second



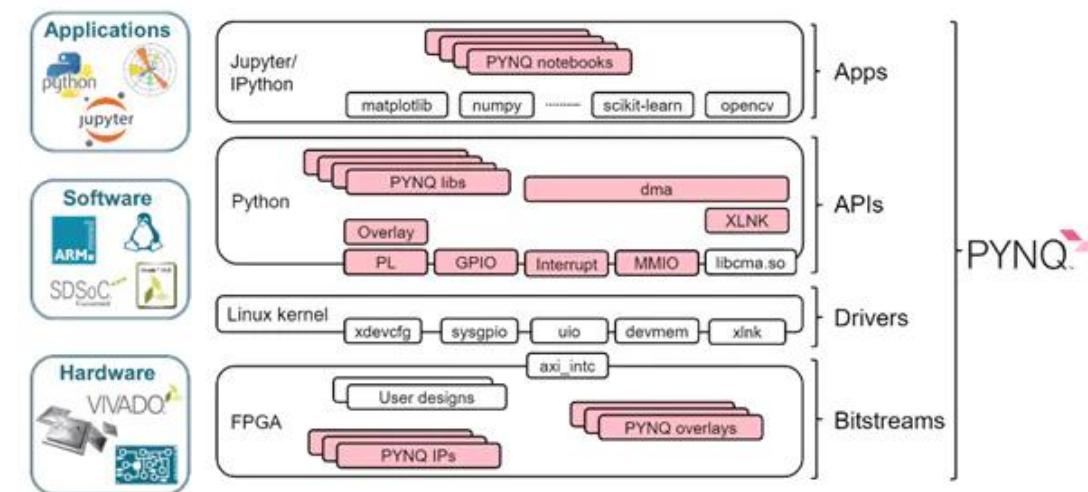
PYNQ Framework: Interfacing Python with Xilinx SoC



PYNQ Components

To achieve performance PYNQ is comprised of

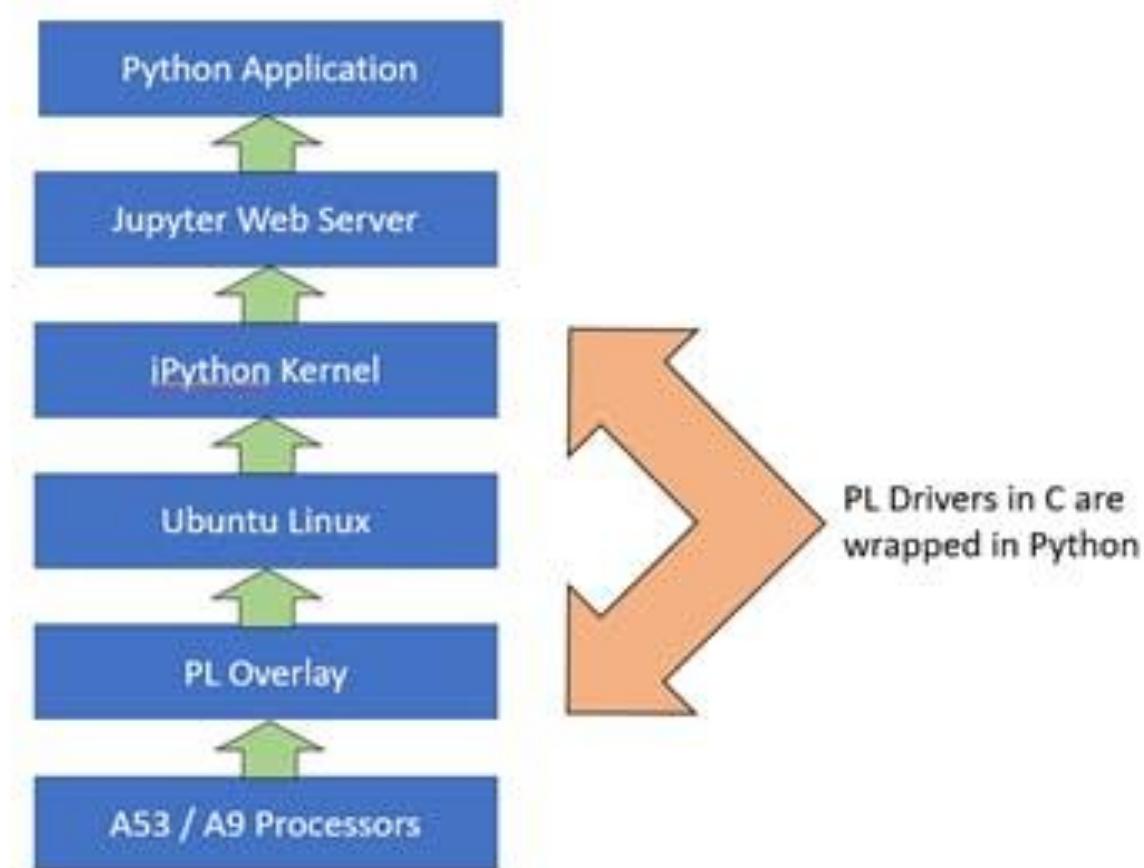
- » Jupyter Notebooks
- » PYNQ Package
- » PYNQ Libs
- » PYNQ Classes
- » PYNQ IP
- » PYNQ Overlays



PYNQ Architecture

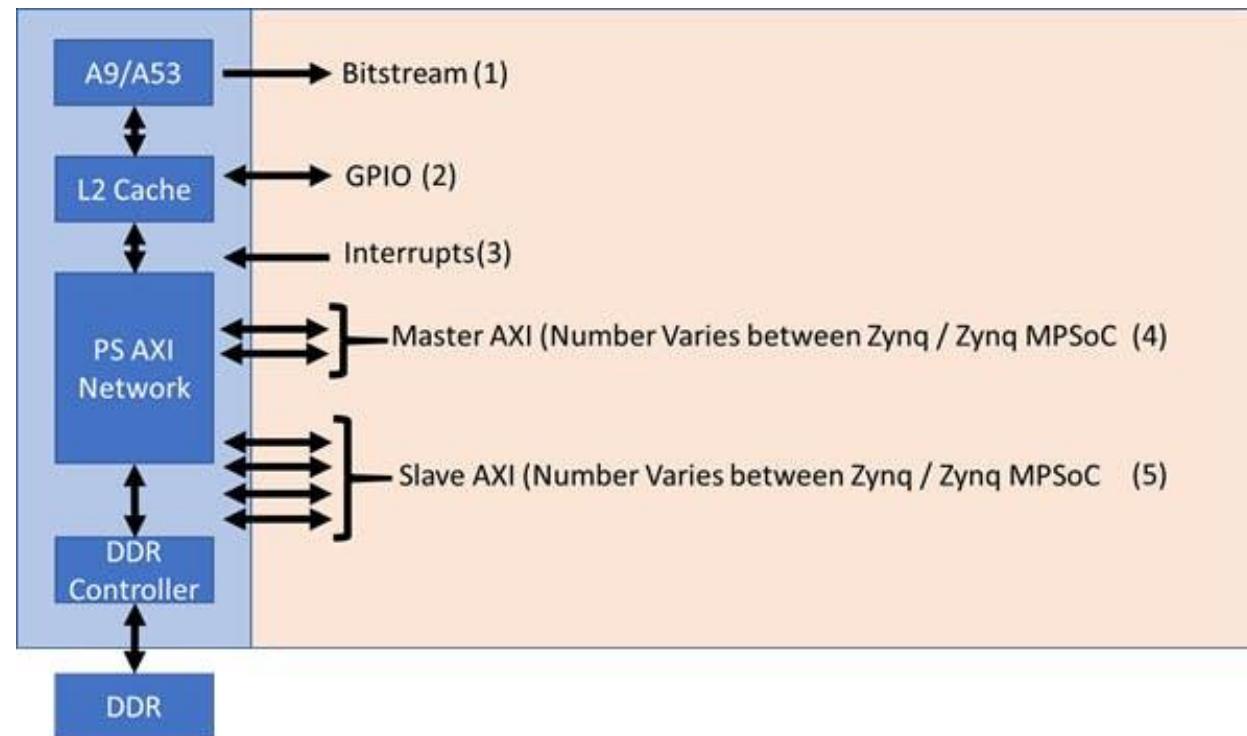
Pynq is built upon Xilinx Petalinux flow

Standard way to create PYNQ for a custom board we will see



Working with the PL

1. Bitstream — This configures the programmable logic for the desired application. In the PYNQ framework, the **xdevcfg** driver is used.
2. GPIO — This provides simple IO in both directions. In the PYNQ framework, this is supported by the **sysgpio** driver.
3. Interrupts — Support interrupt generation from the programmable logic to the processing system. In the PYNQ framework, this is supported by the **Userspace IO** driver.
4. Master AXI Interfaces — These are used to transfer data between the PS to the PL when the PS is the initiator of the transaction. The PYNQ framework uses **devmem** when employing master AXI interface.
5. Slave AXI Interfaces — These are used to transfer data between the PS and PL when the PL is the initiator of the transaction. The PYNQ framework uses **xlnk** to enable these transfers.



Available PYNQ boards

Several PYNQ builds for existing boards:

- » Pynq Z1 — Zynq SoC 7020
- » Pynq Z2 — Zynq SoC 7020
- » ZCU104 — Zynq MPSoC XCZU7EV
- » ZCU111 — Zynq RFSoC XCZU28DR
- » Ultra96 — Zynq MPSoC ZU3EG



Arty Z7-20 can use the Pynq Z1 image a basically same board.

Overlays

Overlays are the design loaded into the programmable logic

Can be custom created or accessed via the [PYNQ.IO community](#)

Range of Overlays in the community including

- » Machine Learning
- » Image Processing
- » RISC-V
- » Kalman filter

Base Overlay is the initial overlay which is created with the PYNQ Image

Download and work with new overlays as required

Of course you can also create your own – As we will see

PYNQ Libraries

PYNQ provides several libraries which provide support for management of the processor and allow access to the low level hardware including

IP Cores – Audio, AXI GPIO, AXI IIC, DMA, Logic Tools, Video

IOP – Arduino, Grove, RPI, PMOD

PYNQ MicroBlaze – MicroBlaze Subsystem RPC and Library

PS / PL Interface – Interrupt, MMIO, PS GPIO, Xlnk

PS Control – PMBus

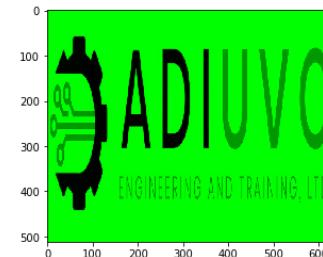
PL Control – Overlay, PL and Bitstream Classes

PYNQ In Industry

Working with THALES – Major Defense Client – High Speed Image processing (3000 > 10,000 FPS)

ZCU102 Development board

"The use of Pynq to rapidly test and evaluate design patterns for image processing has been invaluable. Not only has it sped up design, but it also reduces the necessary team size to a manageable level."



```
In [21]: from pynq.ps import Clocks
Clocks.fc1k0_mhz = 350
Clocks.fc1k0_mhz
/usr/local/lib/python3.6/dist-packages/pynq/ps.py:312: UserWarning: Setting frequency to the closest possible value 374.99625MHz.
    round(freq_high / q0, 5)))
Out[21]: 374.99625

In [22]: gpio = overlay.axi_gpio_0
gpio.write(0x00,374996250)

In [33]: FPS = gpio.read(0x08)
print("frames per second =",FPS)
frames per second = 1121
```

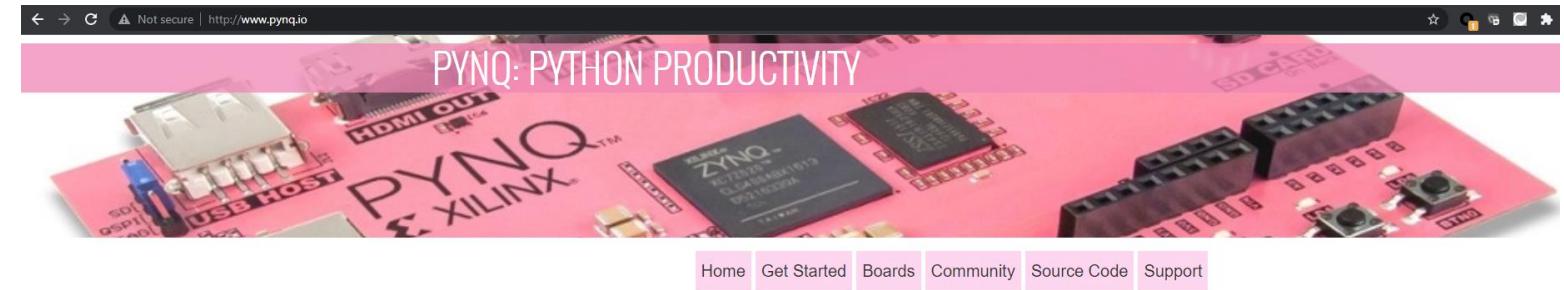


Accelerating FFT with PYNQ

Lab : FFT Acceleration

Open a browser and go to

www.pynq.io



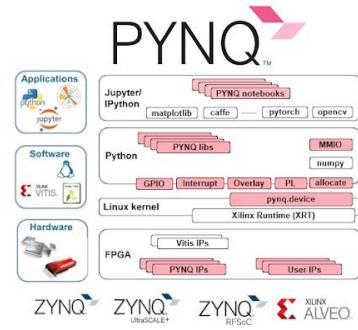
What is PYNQ?

PYNQ is an open-source project from Xilinx® that makes it easier to use Xilinx platforms.

Using the Python language and libraries, designers can exploit the benefits of programmable logic and microprocessors to build more capable and exciting electronic systems.

PYNQ can be used with Zynq, Zynq UltraScale+, Zynq RFSoC, Alveo accelerator boards and AWS-F1 to create high performance applications with:

- parallel hardware execution
- high frame-rate video processing
- hardware accelerated algorithms
- real-time signal processing
- high bandwidth IO
- low latency control



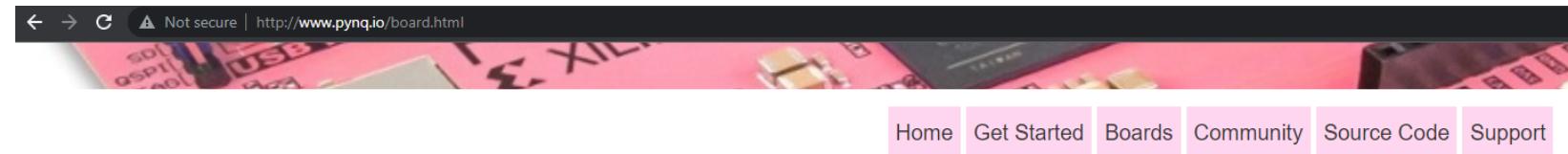
Who is PYNQ for?

PYNQ is intended to be used by a wide range of designers and developers including:

- Software developers who want to take advantage of the capabilities of Xilinx platforms without having to use ASIC-style design tools to design hardware.
- System architects who want an easy software interface and framework for rapid prototyping and development of their Zynq, Alveo and AWS-F1 designs.

Lab : FFT Acceleration

Select the boards page and
download the SD card
image for Pynq Z1 v2.6



Development Boards

PYNQ supports Zynq based boards (Zynq, Zynq Ultrascale+, Zynq RFSoC), and **Xilinx Alveo** accelerator boards and **AWS-F1** instances.
See the [PYNQ Alveo Getting Started guide](#) for details on installing PYNQ for use with Alveo and AWS-F1.

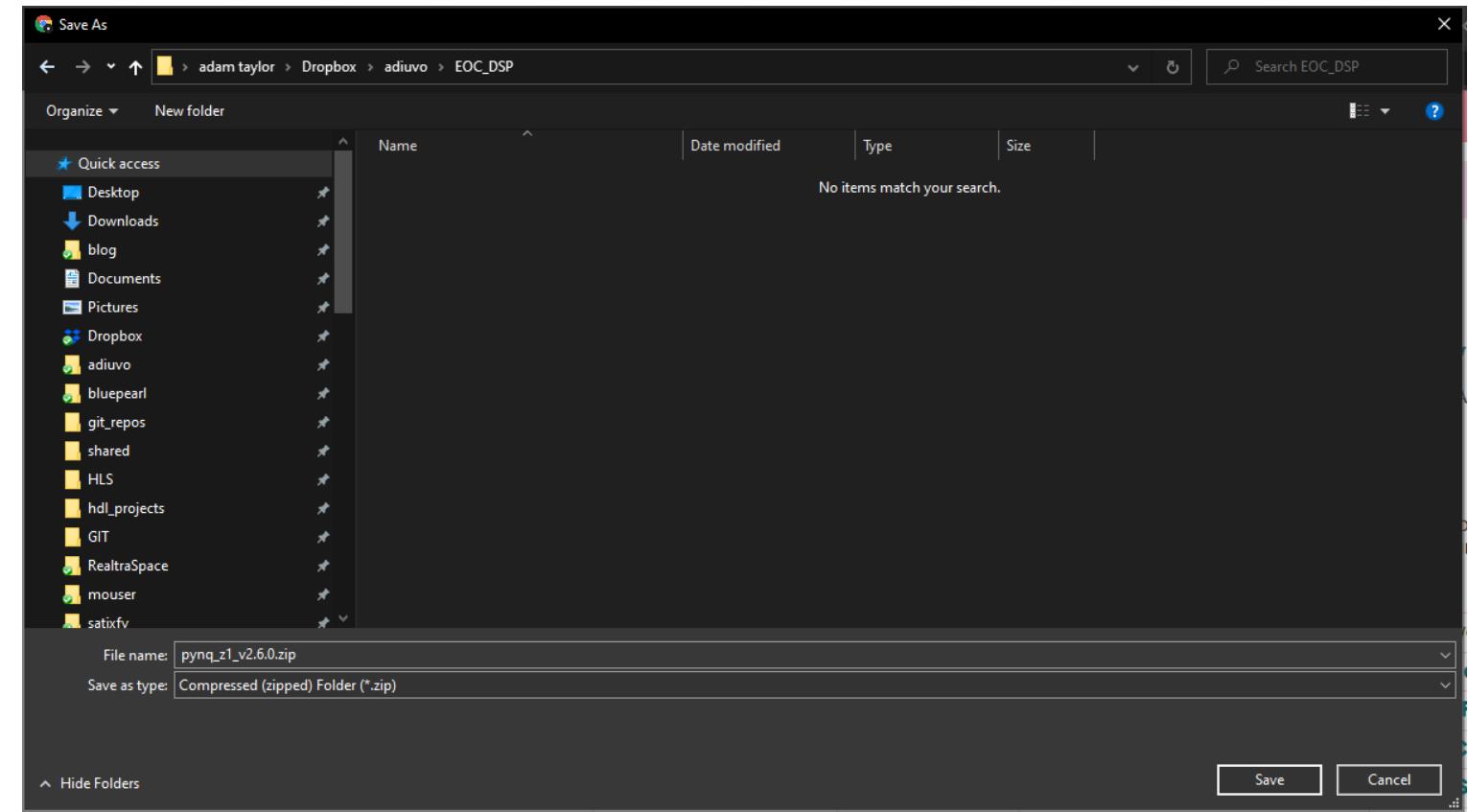
Downloadable PYNQ images

If you have a Zynq board, you need a PYNQ SD card image to get started. You can download a pre-compiled PYNQ image from the table below. If an image is not available for your board, you can build your own SD card image (see details below).

| Board | SD card image | Documentation | Vendor webpage |
|------------------|----------------------|--|---------------------------------------|
| PYNQ-Z2 | v2.6 | PYNQ setup guide | TUL Pynq-Z2 |
| PYNQ-Z1 | v2.6 | PYNQ setup guide | Digilent Pynq-Z1 |
| ZCU104 | v2.6 | PYNQ setup guide | Xilinx ZCU104 |
| RFSoC 2x2 | v2.6 | RFSoC 2x2 GitHub Pages | XUP RFSoC 2x2 |
| ZCU111 | v2.6 | PYNQ RFSoC workshop | Xilinx ZCU111 |
| Ultra96V2 | v2.6 | Avnet PYNQ documentation | Avnet Ultra96V2 |
| Ultra96 (legacy) | v2.6 | See Ultra96V2 | See Ultra96V2 |
| TySOM-3-ZU7EV | v2.5 | GitHub project page | Aldec TySOM-3-ZU7EV |
| TySOM-3A-ZU19EG | v2.5 | GitHub project page | Aldec TySOM-3A-ZU19EG |

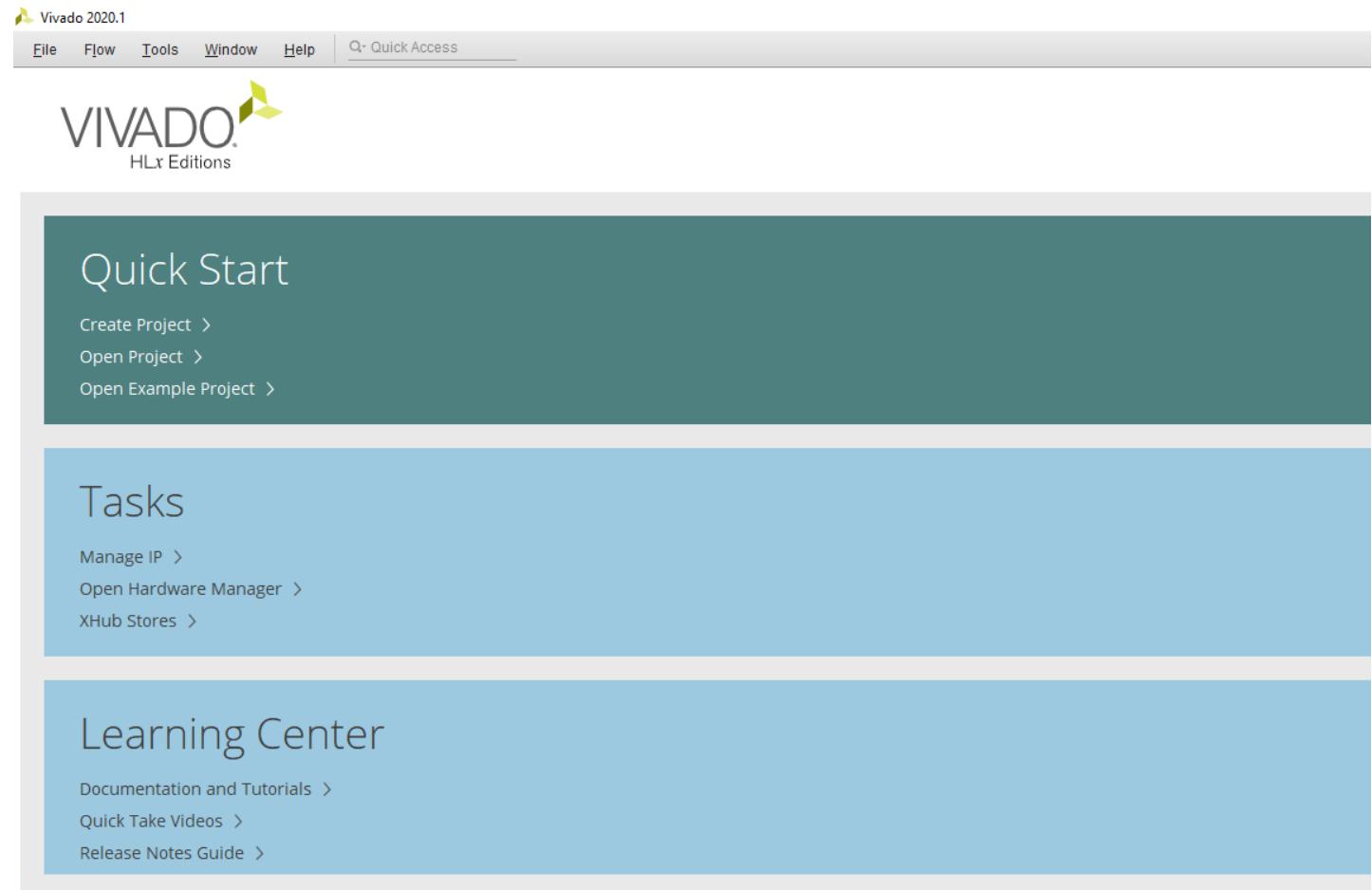
Lab : FFT Acceleration

Save the SD Card image to a
preferred location on your
local computer



Lab : FFT Acceleration

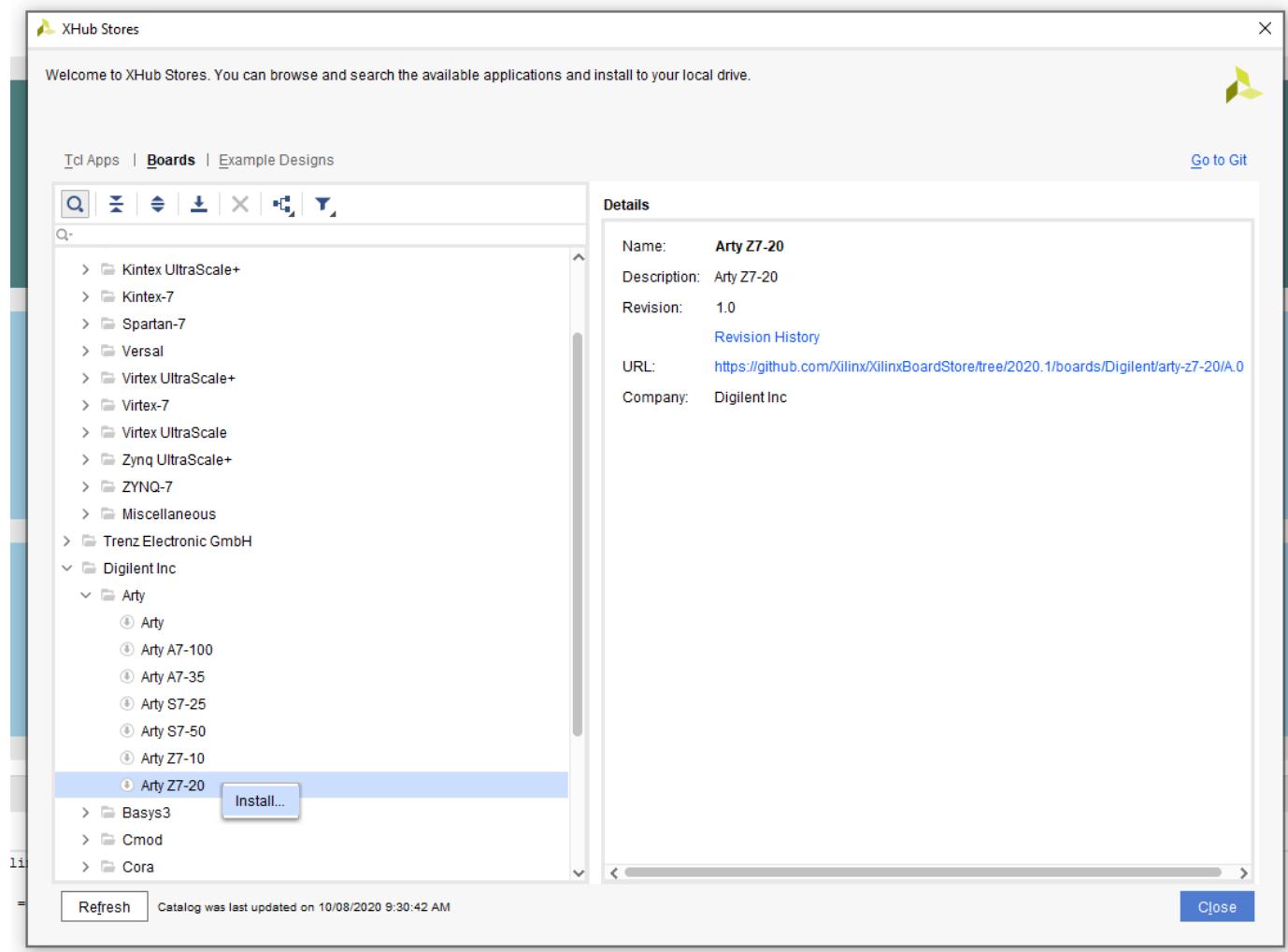
Open Vivado and select Xhub
Stores



Lab : FFT Acceleration

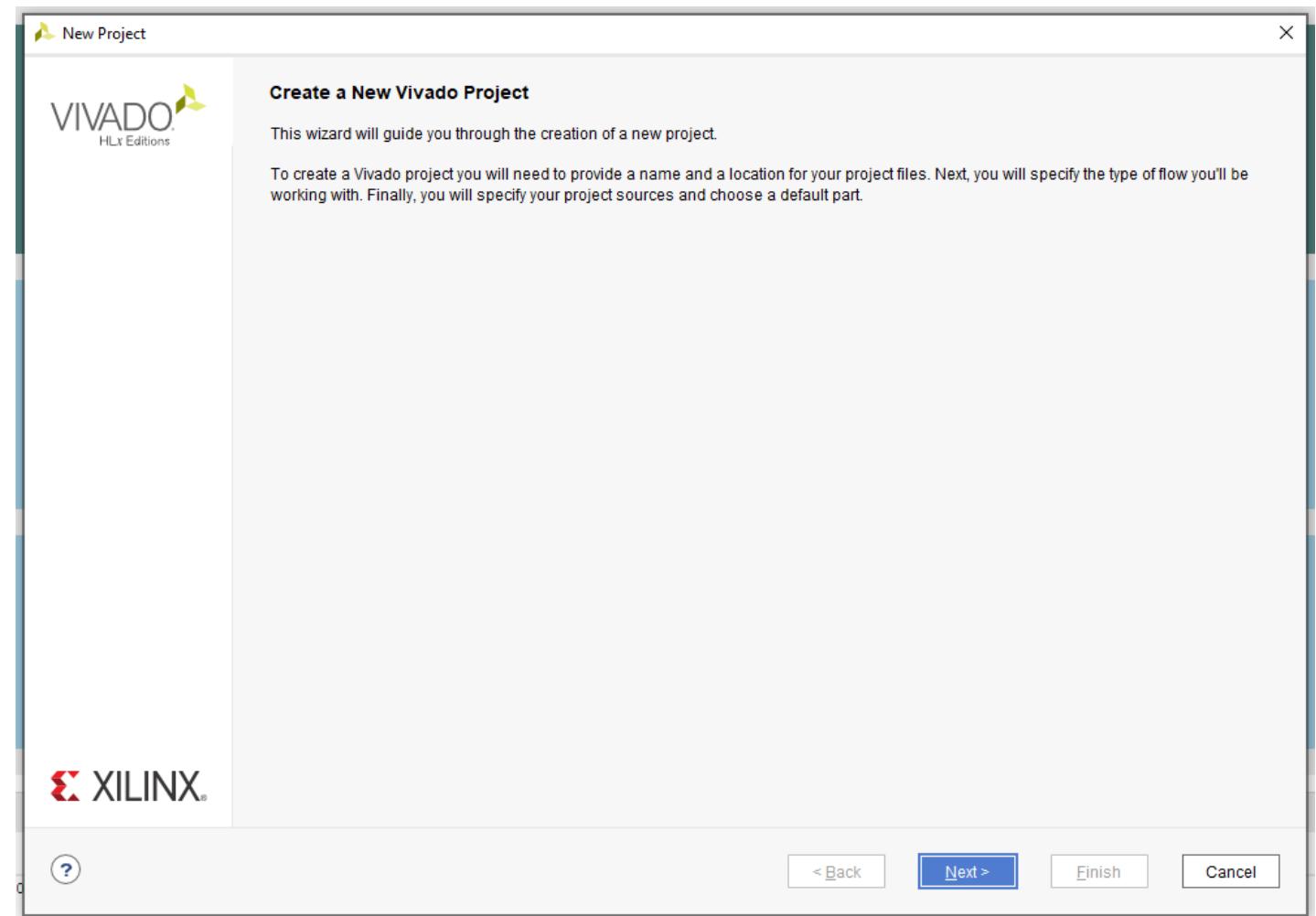
From the boards tab, select
Digilent Inc folder. Expand the
Arty directory and select Arty.

Right click on the Arty Z7-20
and select install



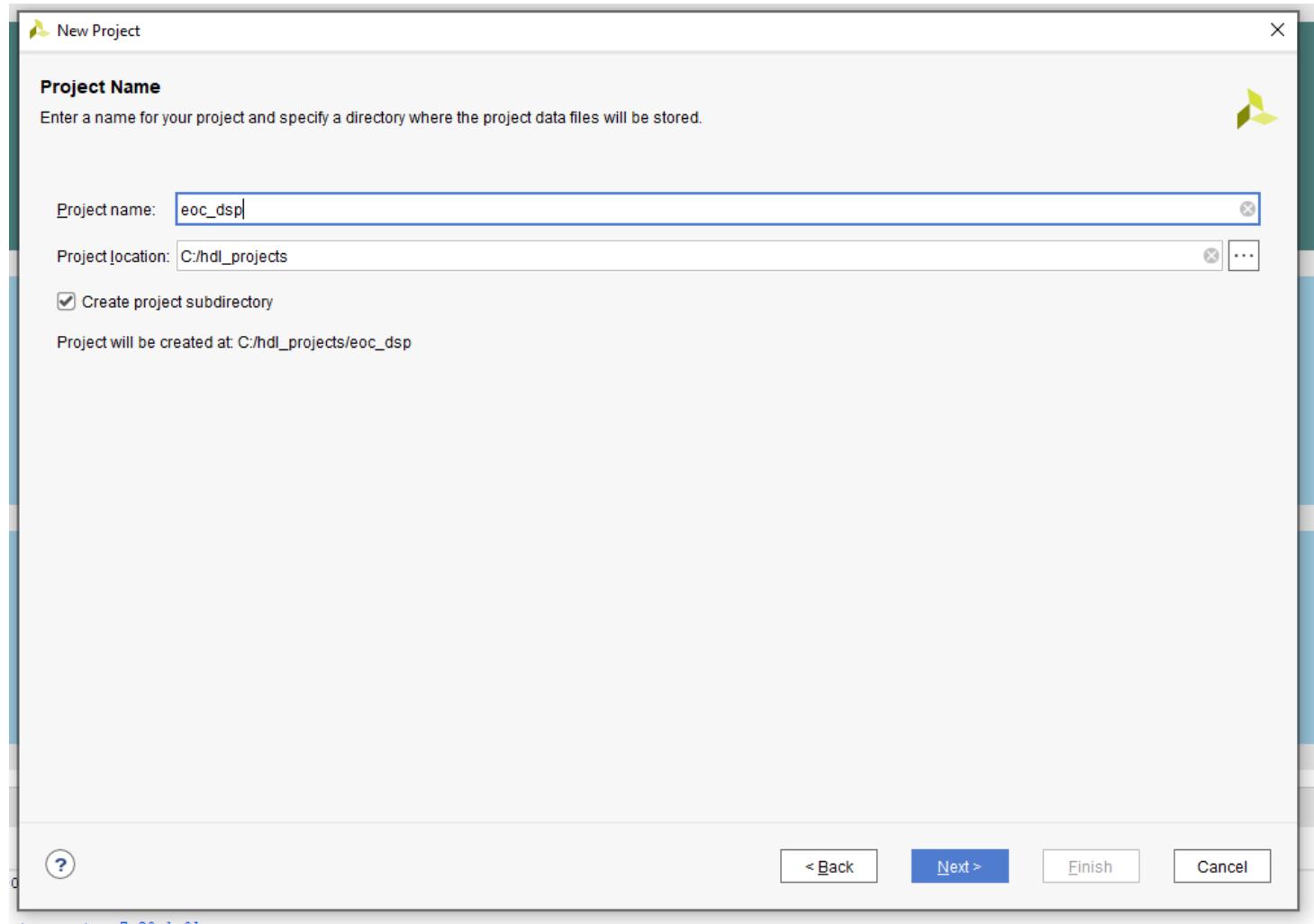
Lab : FFT Acceleration

Create a new project



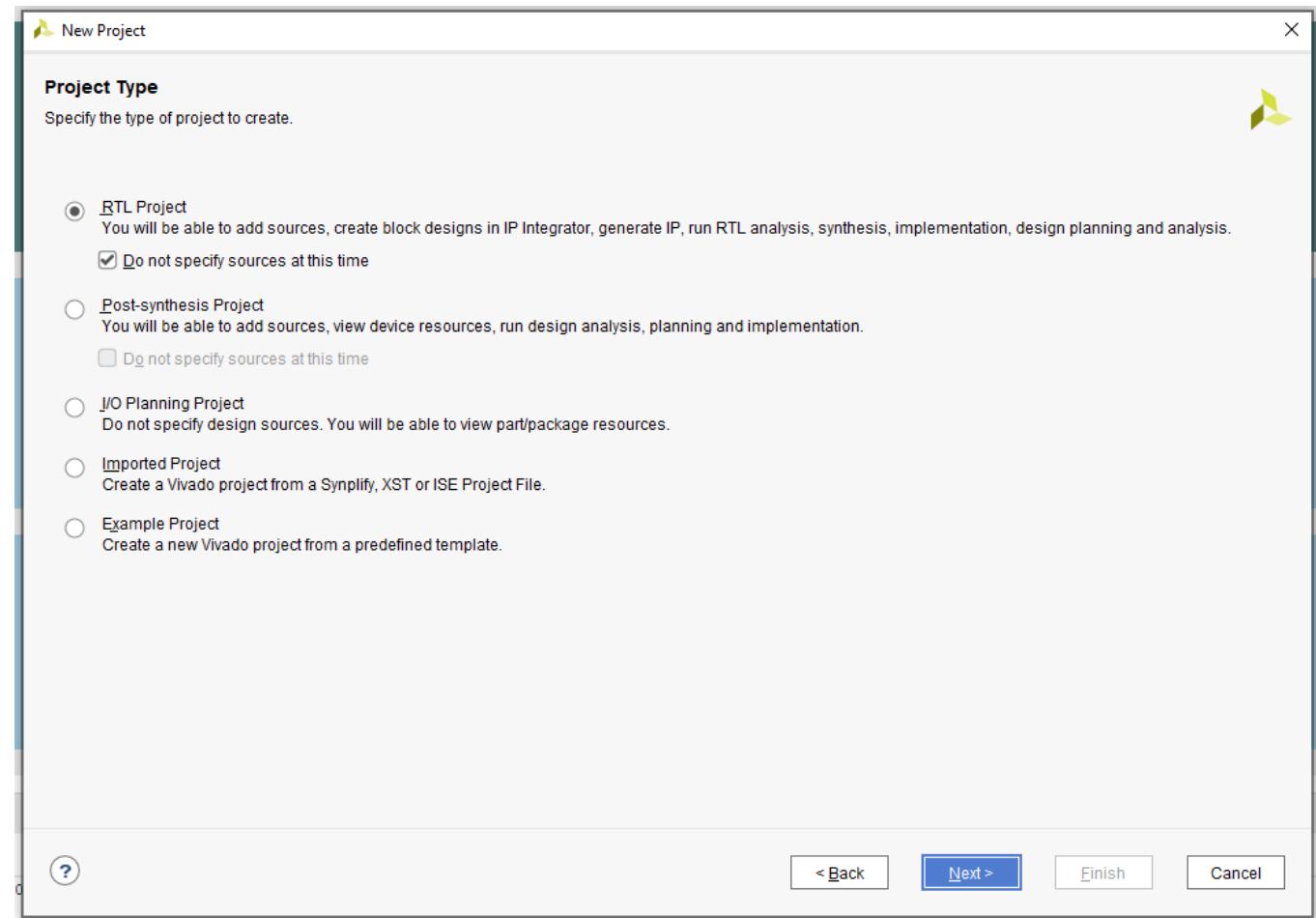
Lab : FFT Acceleration

Enter a name and location



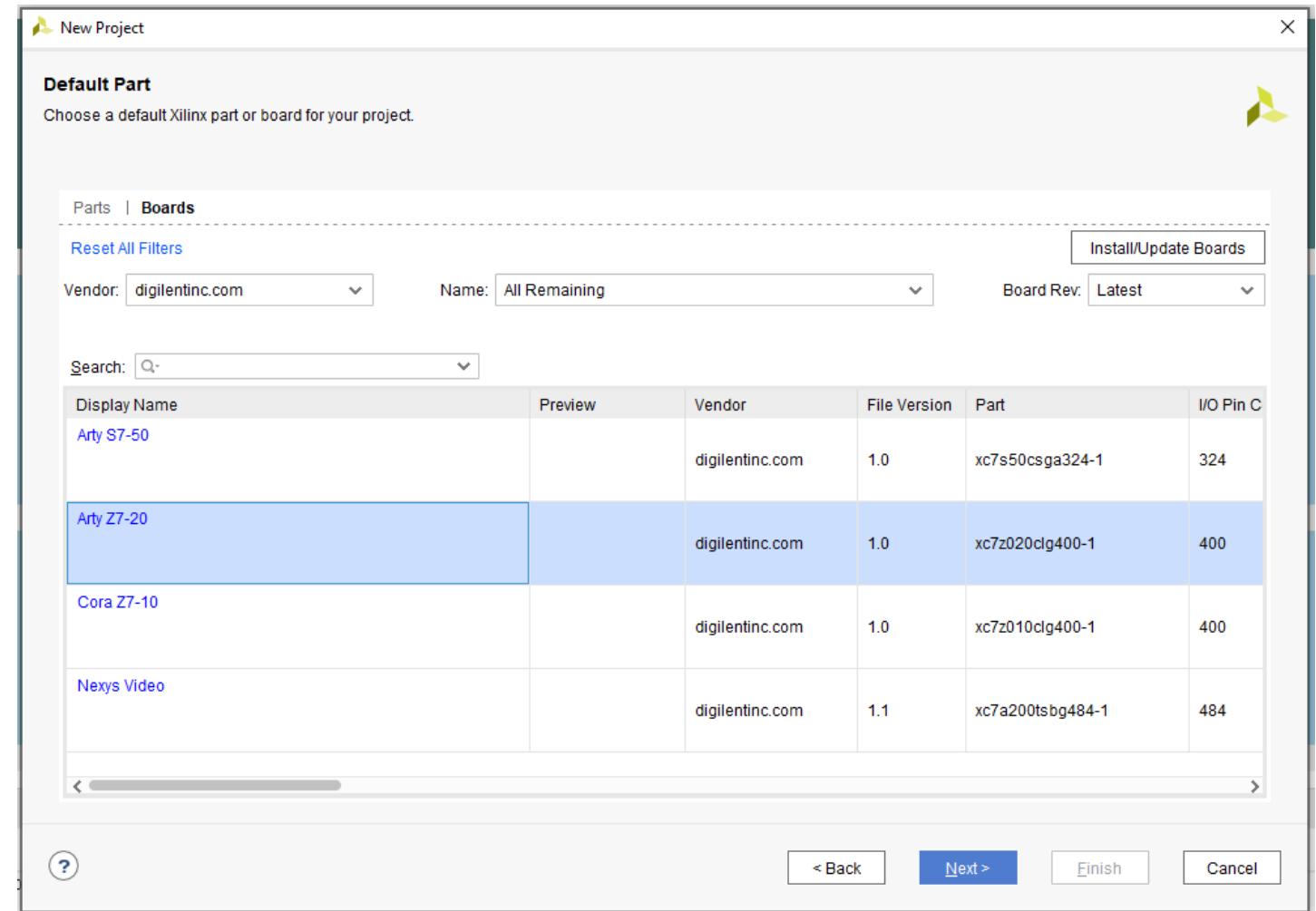
Lab : FFT Acceleration

Select RTL Project



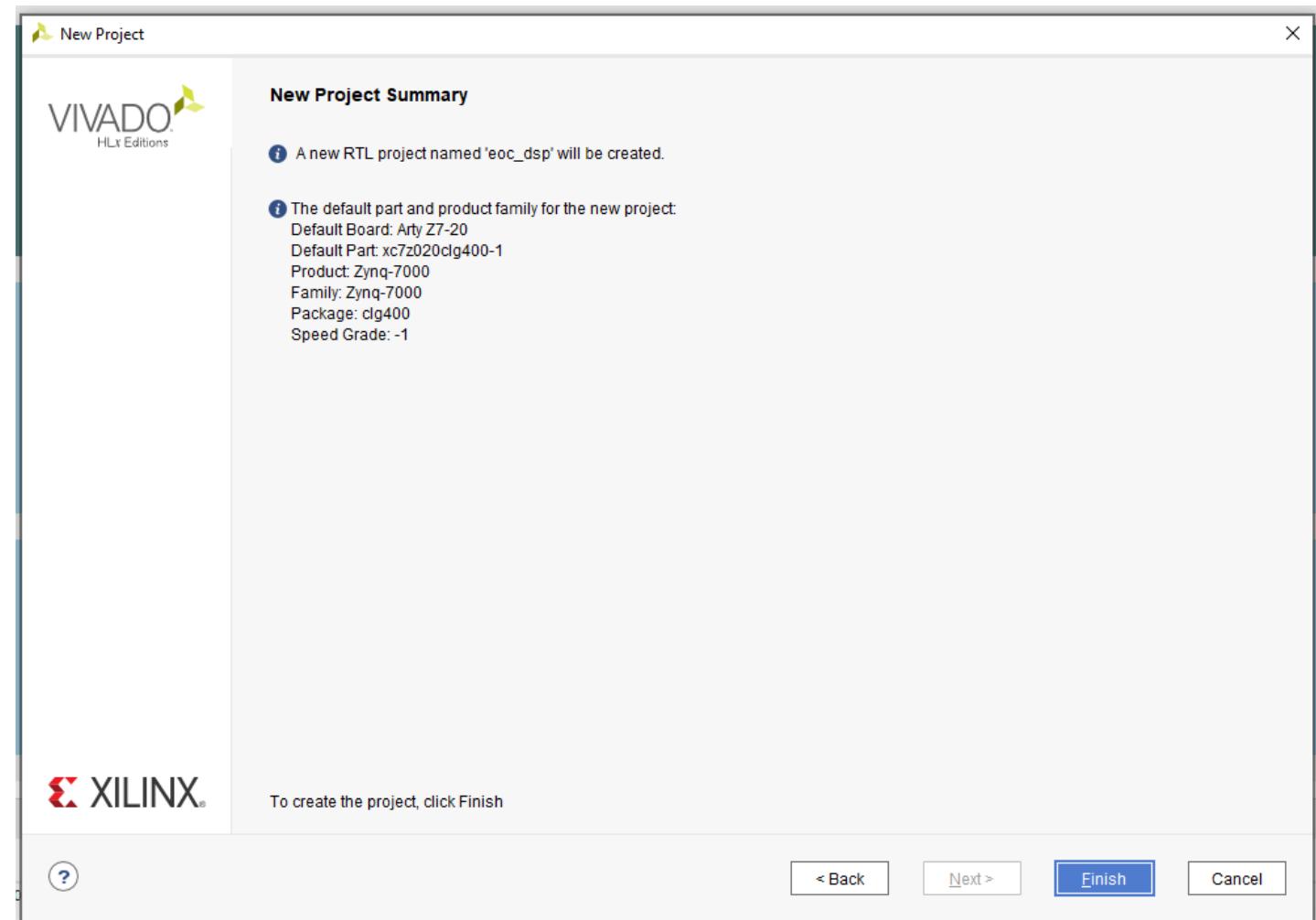
Lab : FFT Acceleration

Select the Arty Z7-20 board



Lab : FFT Acceleration

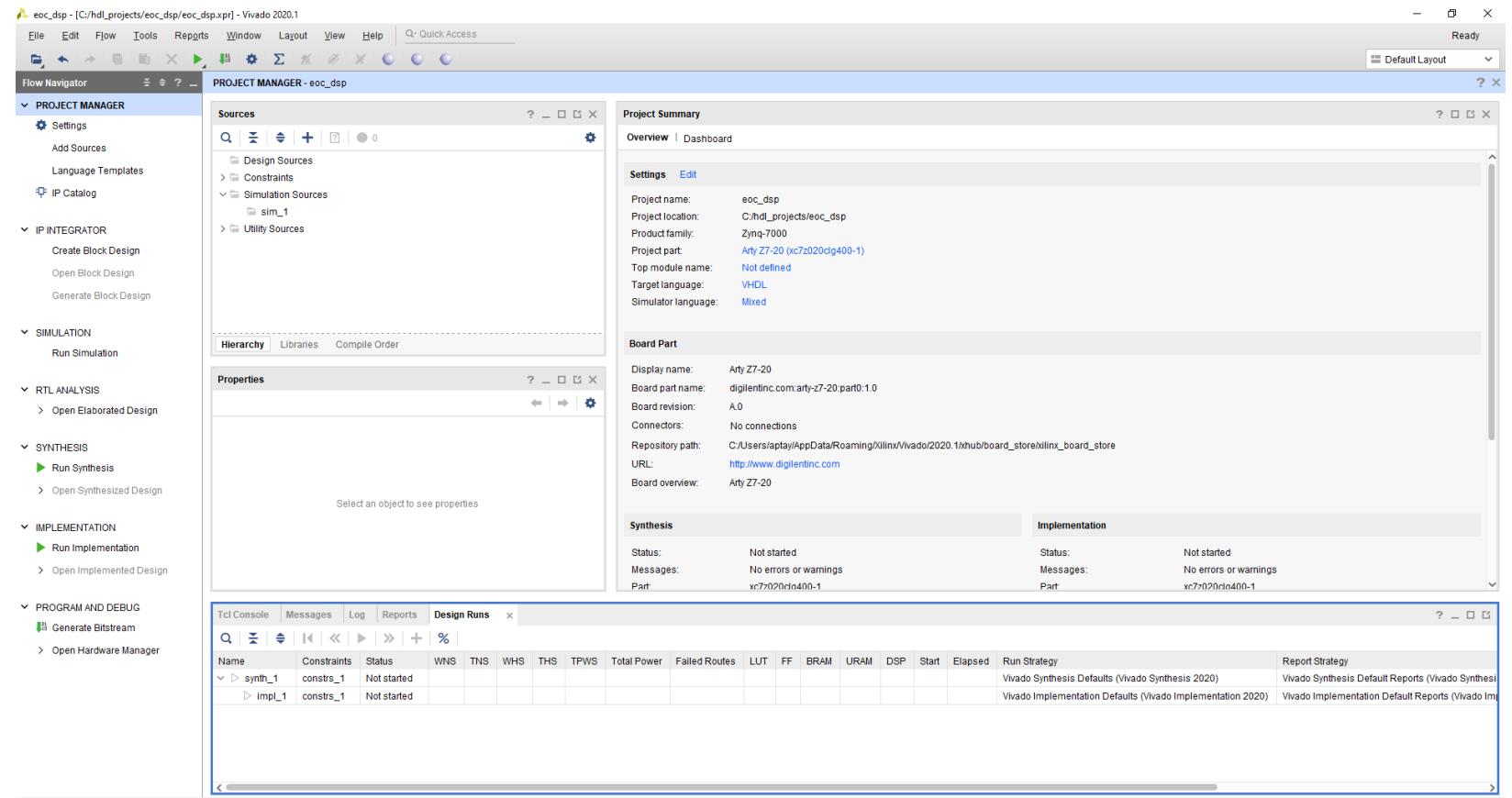
Click Finish to create the project



Lab : FFT Acceleration

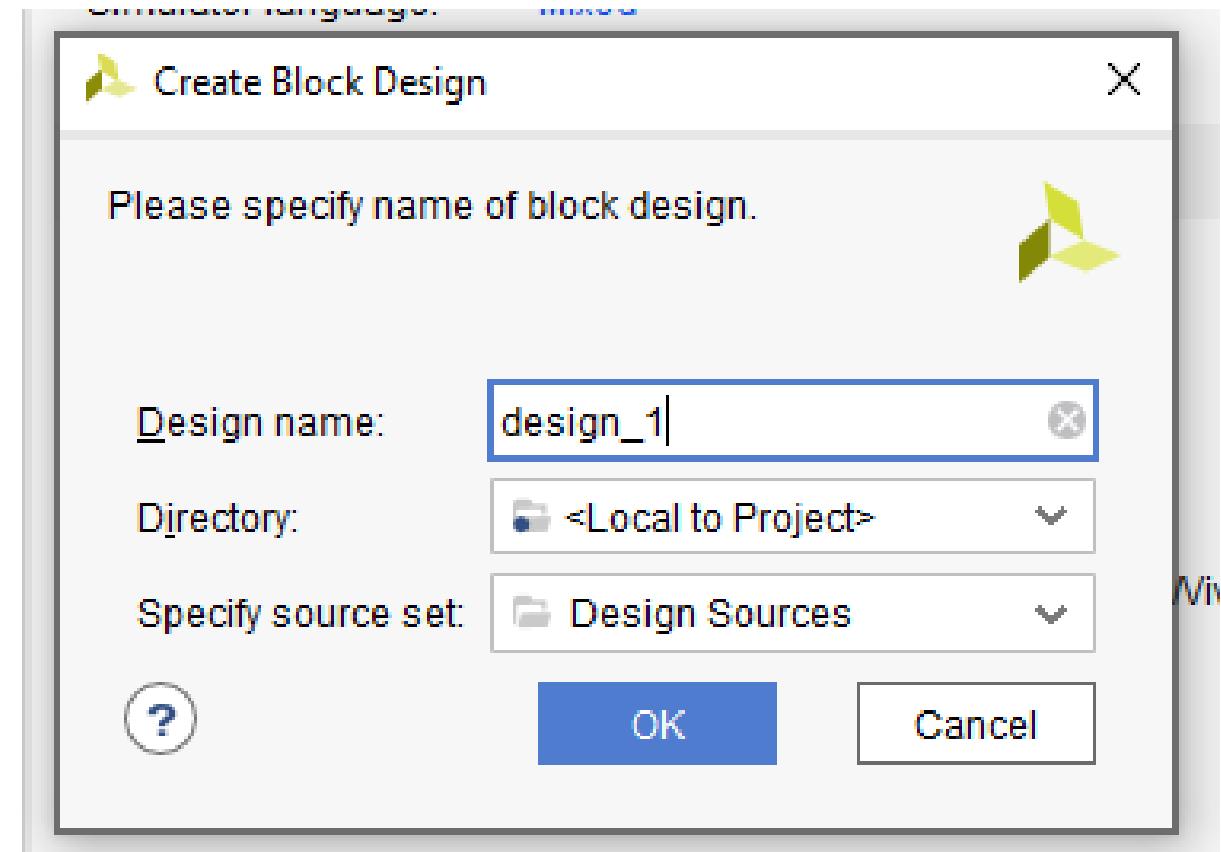
From the Project

Manager, select create
block diagram



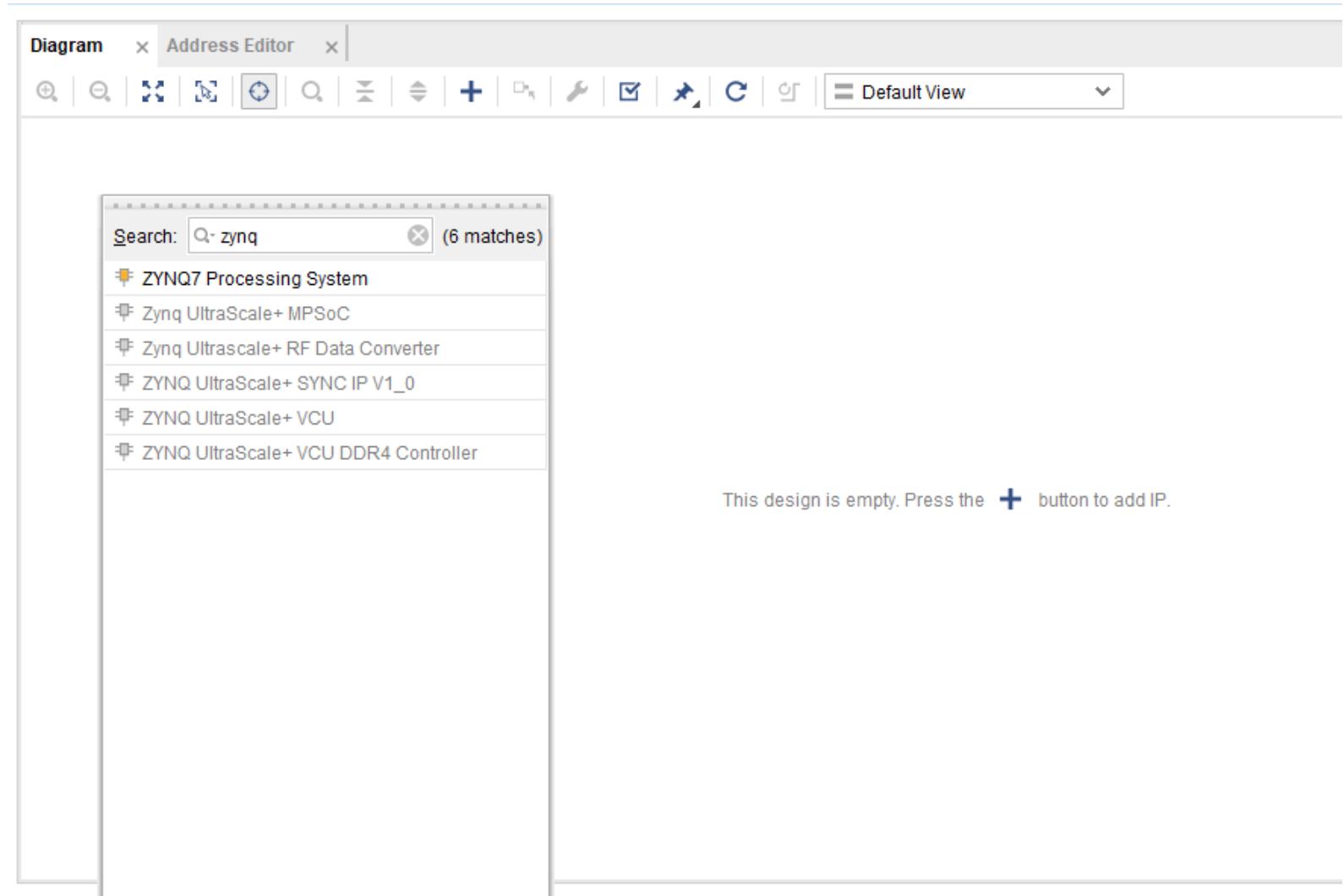
Lab : FFT Acceleration

Leave, defaults unchanged and click OK



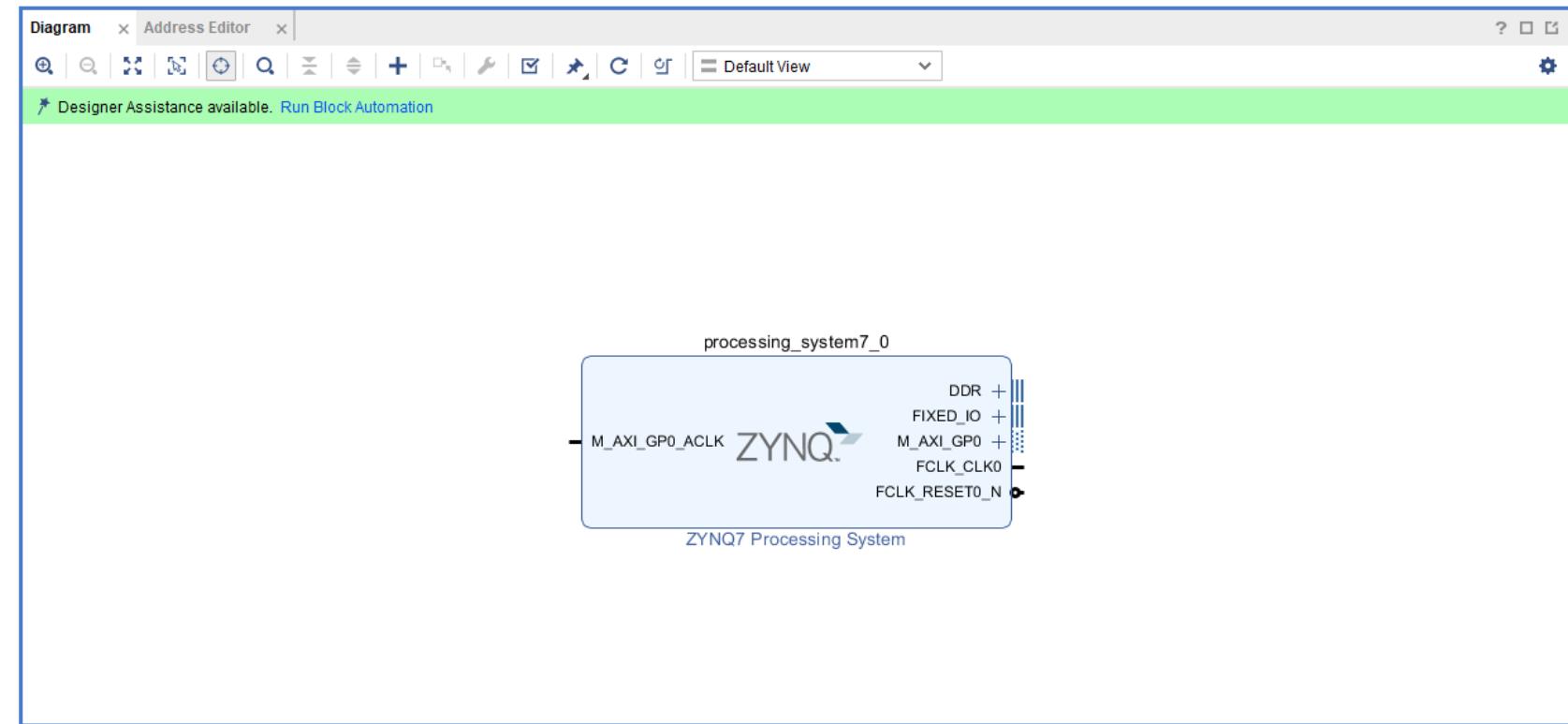
Lab : FFT Acceleration

Click on + and in the search bar type in Zynq and press enter



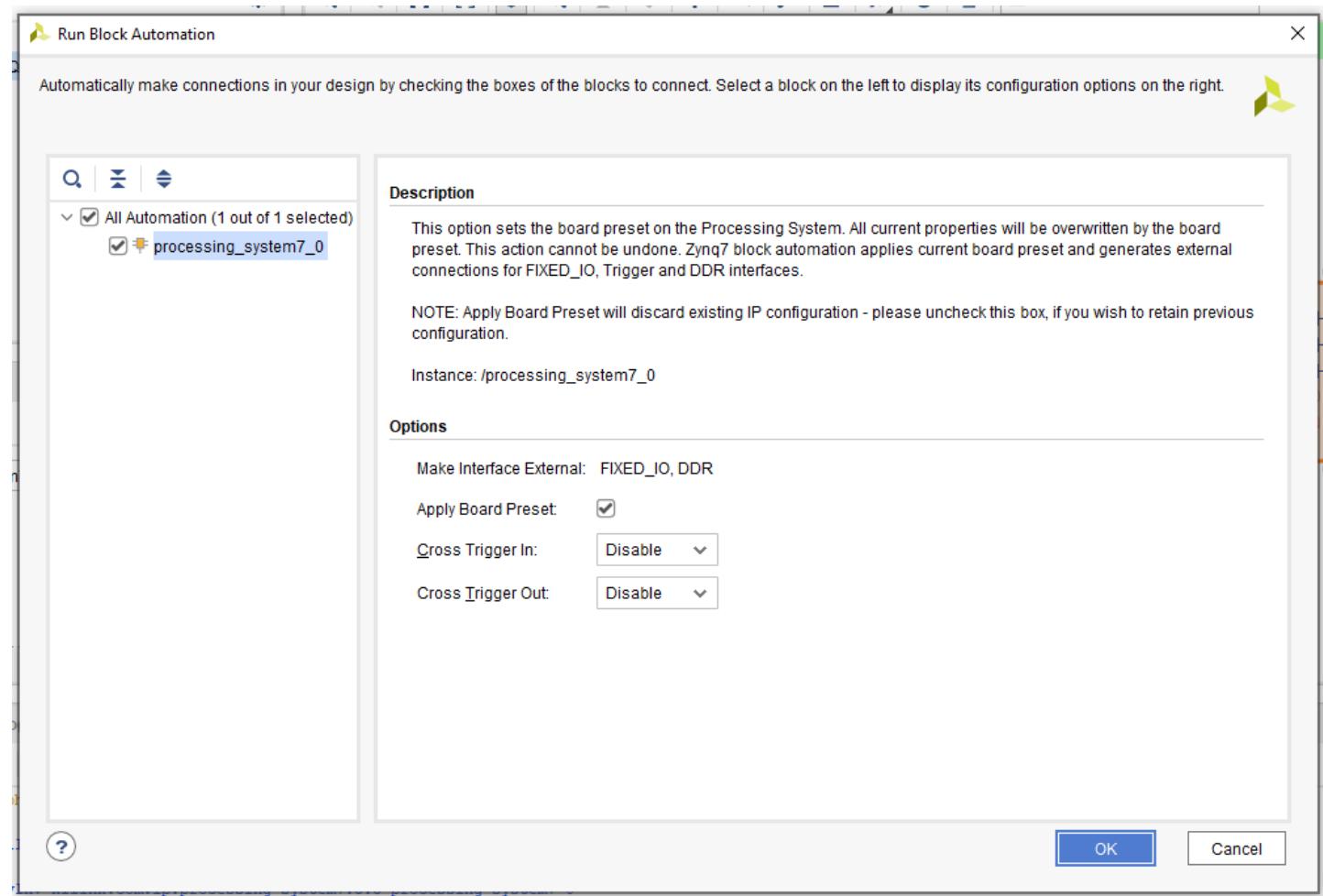
Lab : FFT Acceleration

Run the block automation



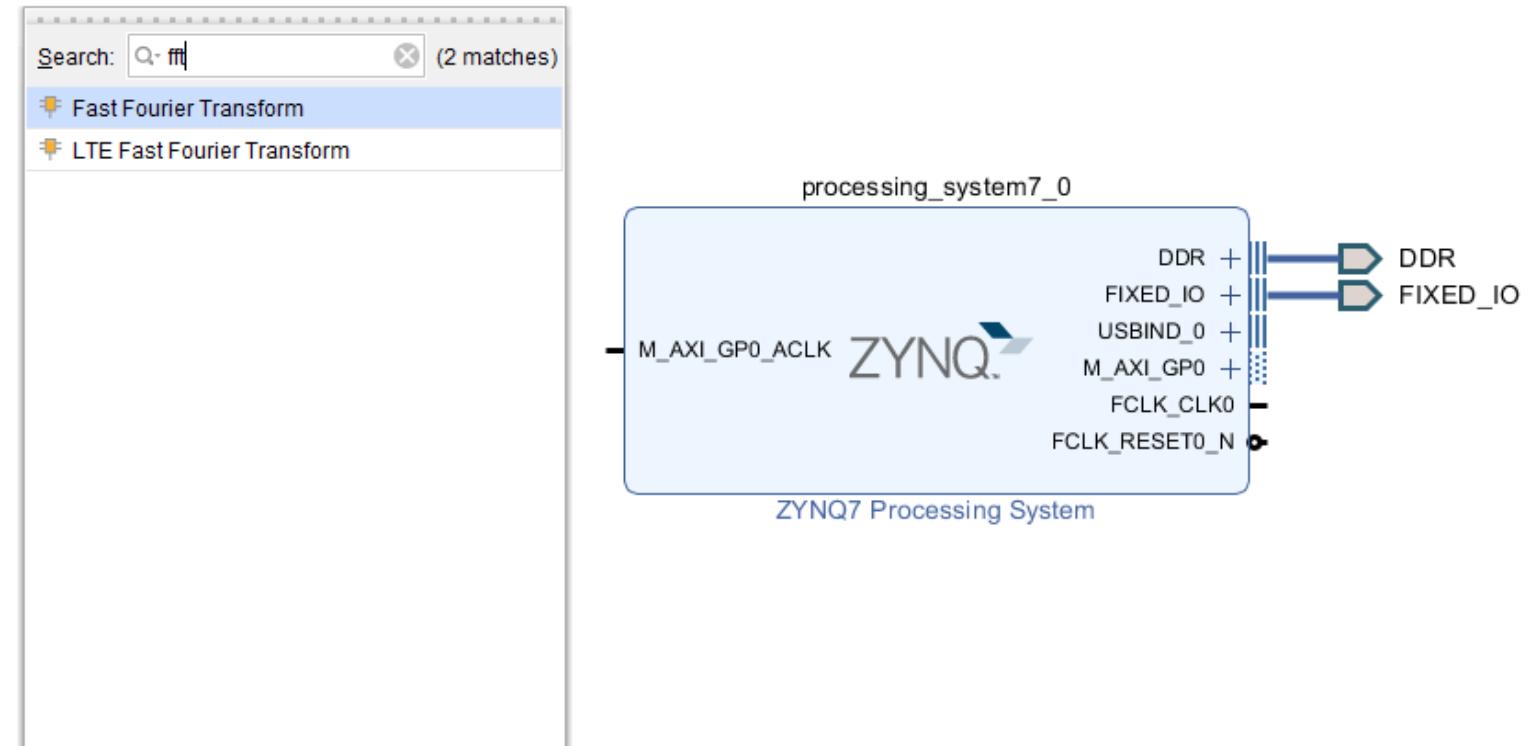
Lab : FFT Acceleration

Leave the settings as default and
click OK



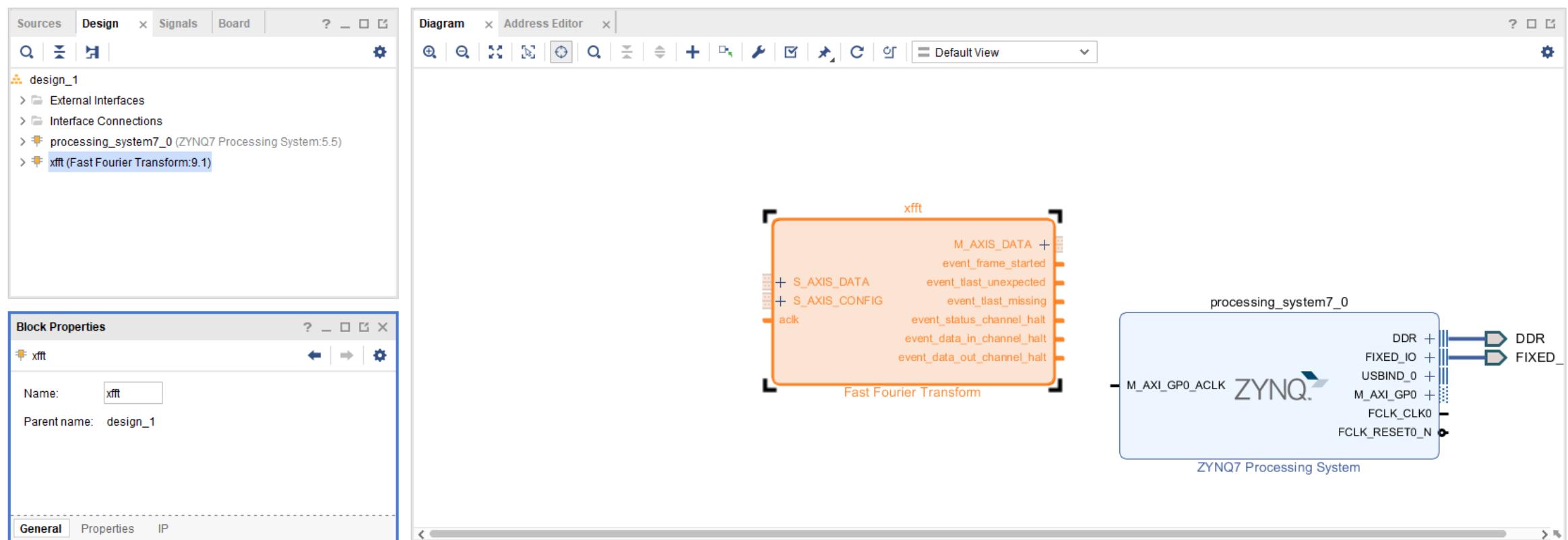
Lab : FFT Acceleration

Click on + and add in the
FFT



Lab : FFT Acceleration

Click on the Fast Fourier Transform and change its name to xfft. Double click on the block to customize it.

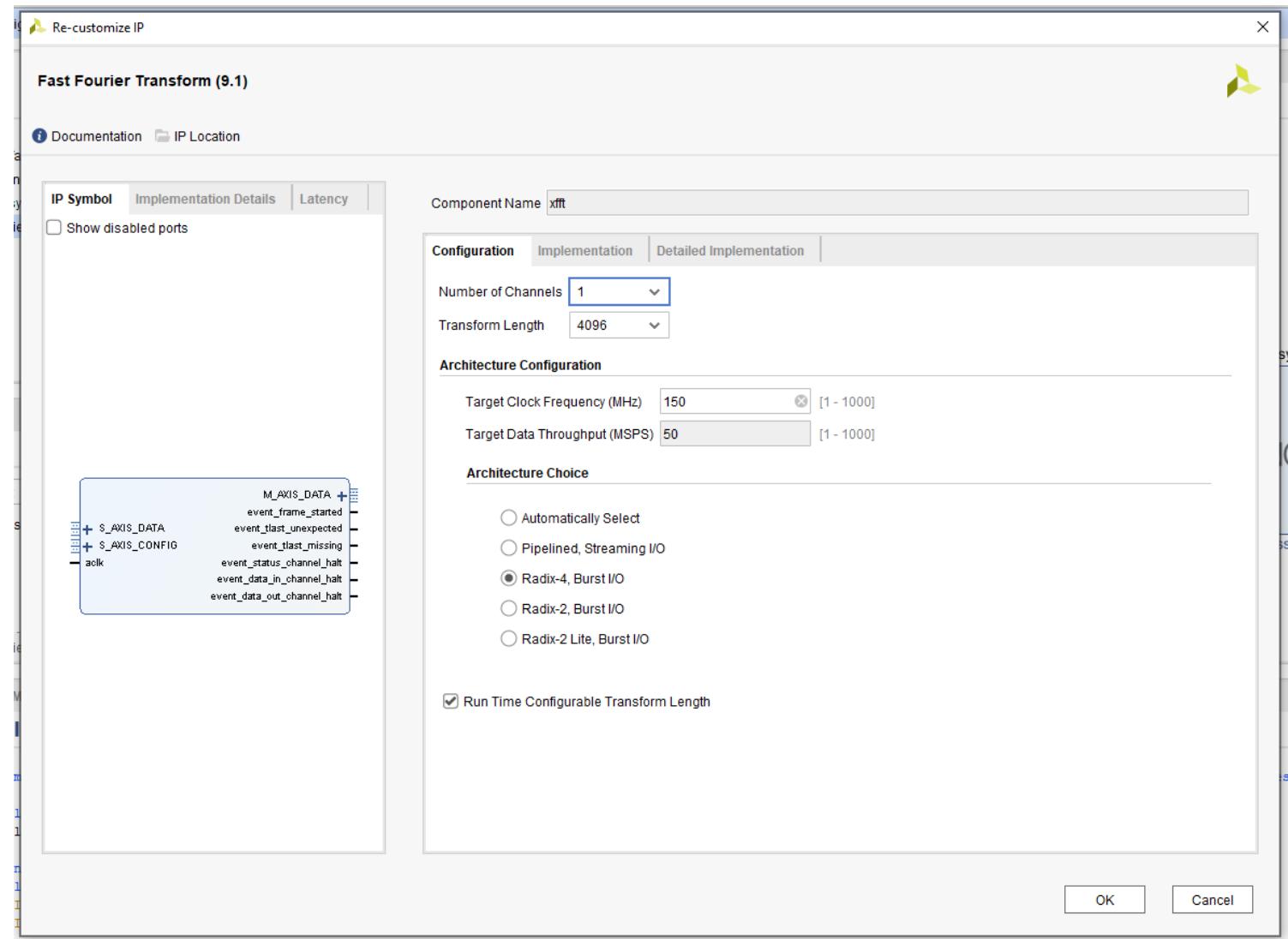


Lab : FFT Acceleration

On the configuration tab, select

- Transform length 4096
- Radix-4 Burst I/O
- Target Frequency 150Mhz
- Enable Run Time

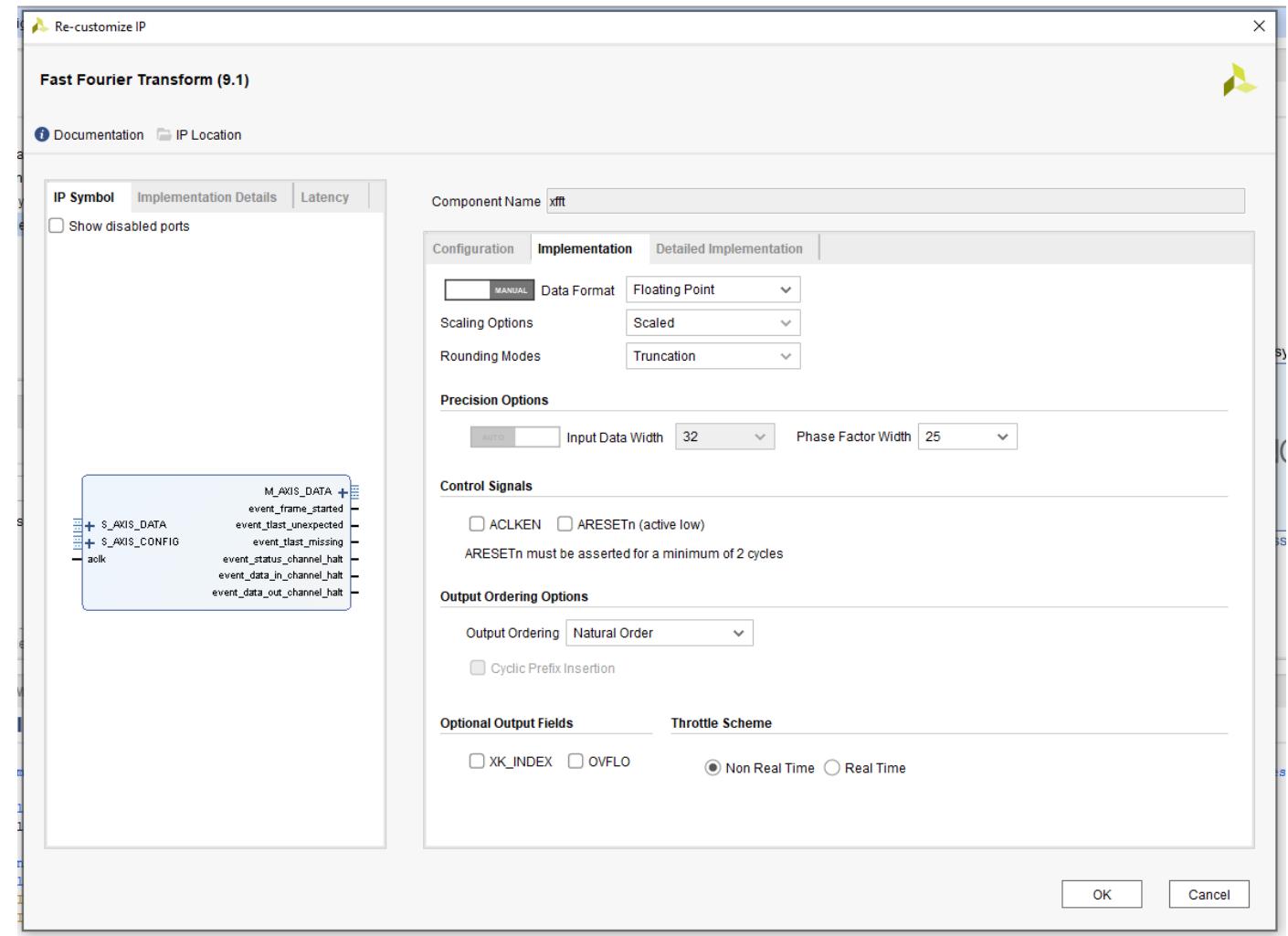
Configurable transform length



Lab : FFT Acceleration

On the implementation tab select

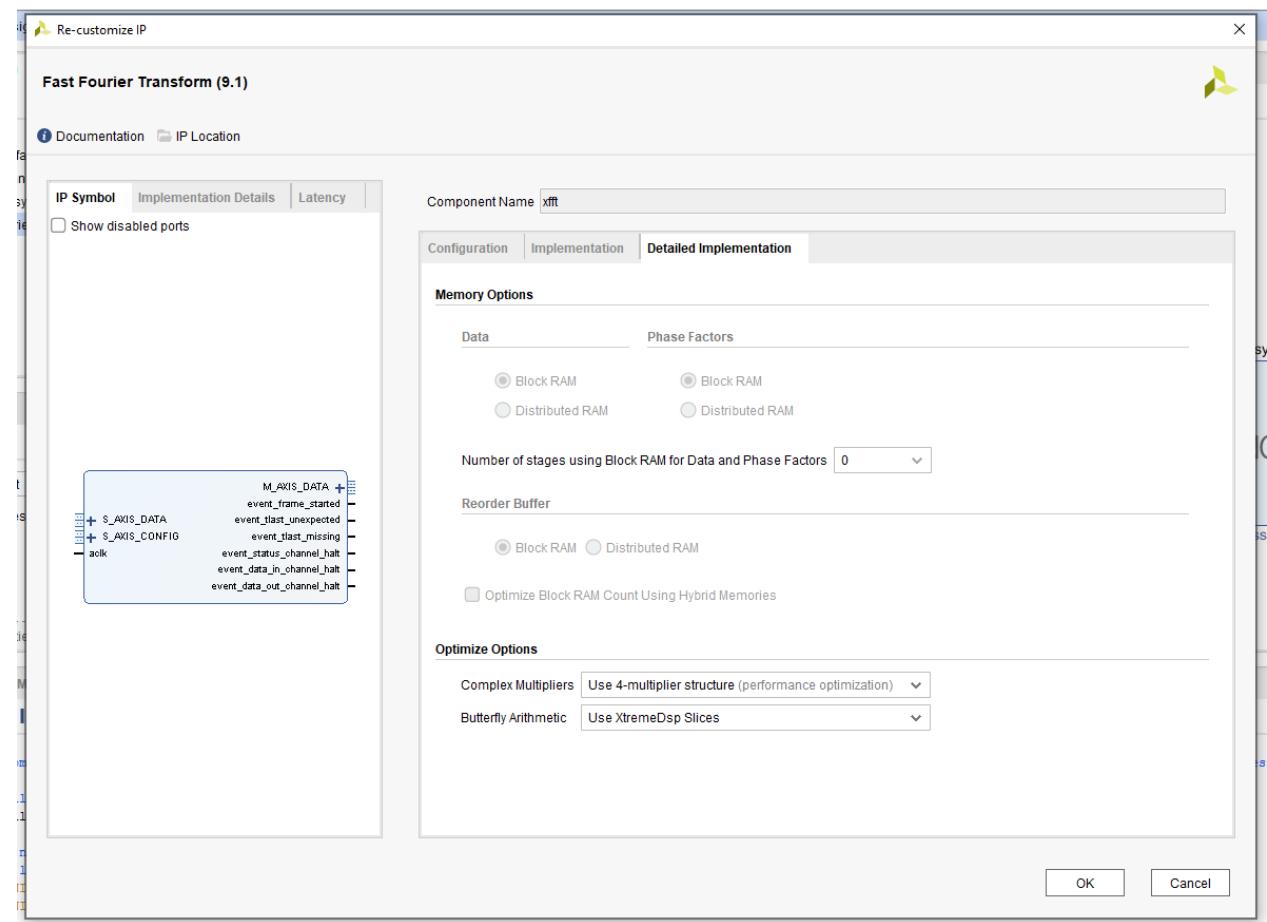
- Floating Point
- Phase Factor Width 25
- Output Ordering Natural
- Non-Real Time Throttle scheme



Lab : FFT Acceleration

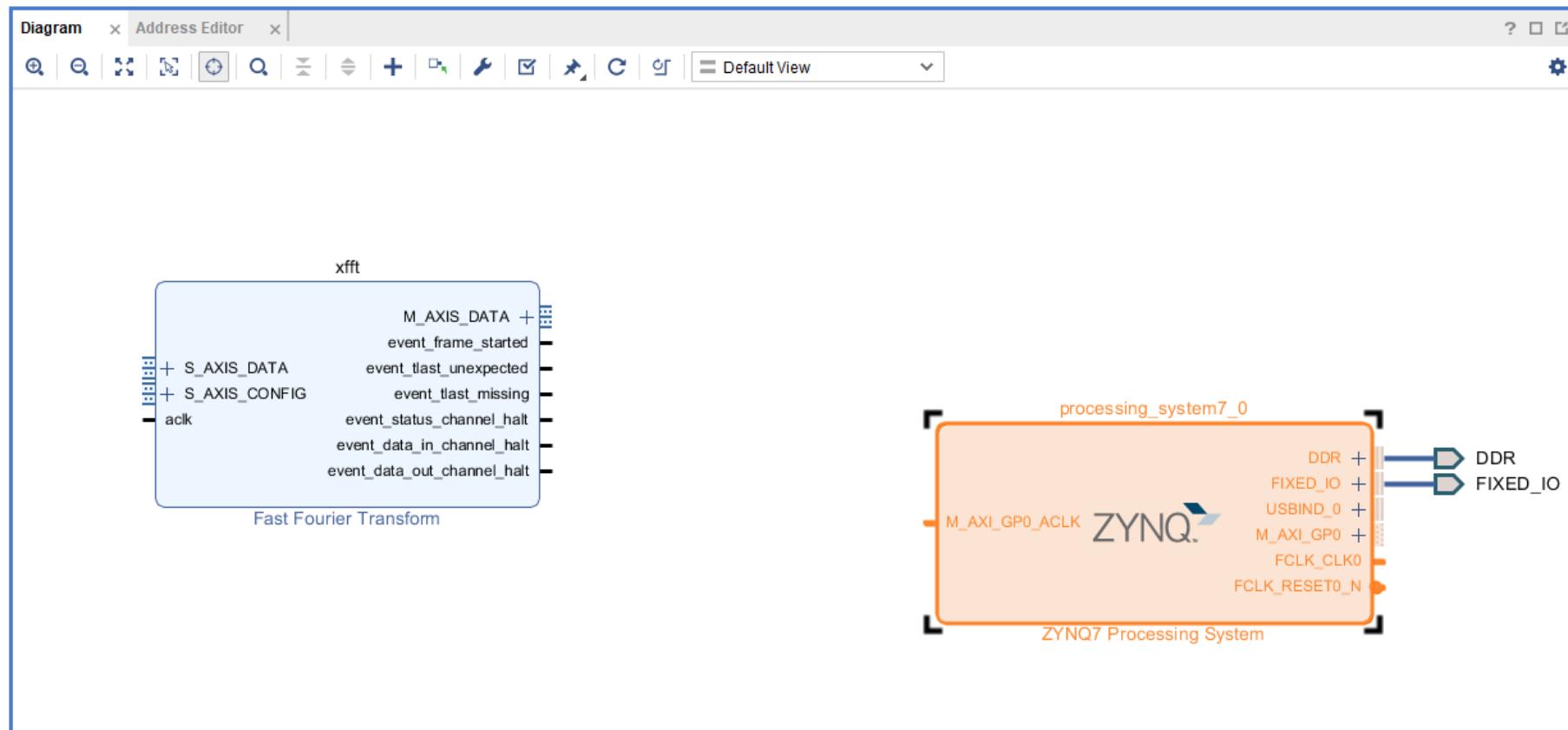
On the Detailed Implementation tab
select

- Use 4-Multiplier Structure
- Use XtremeDSP Slices



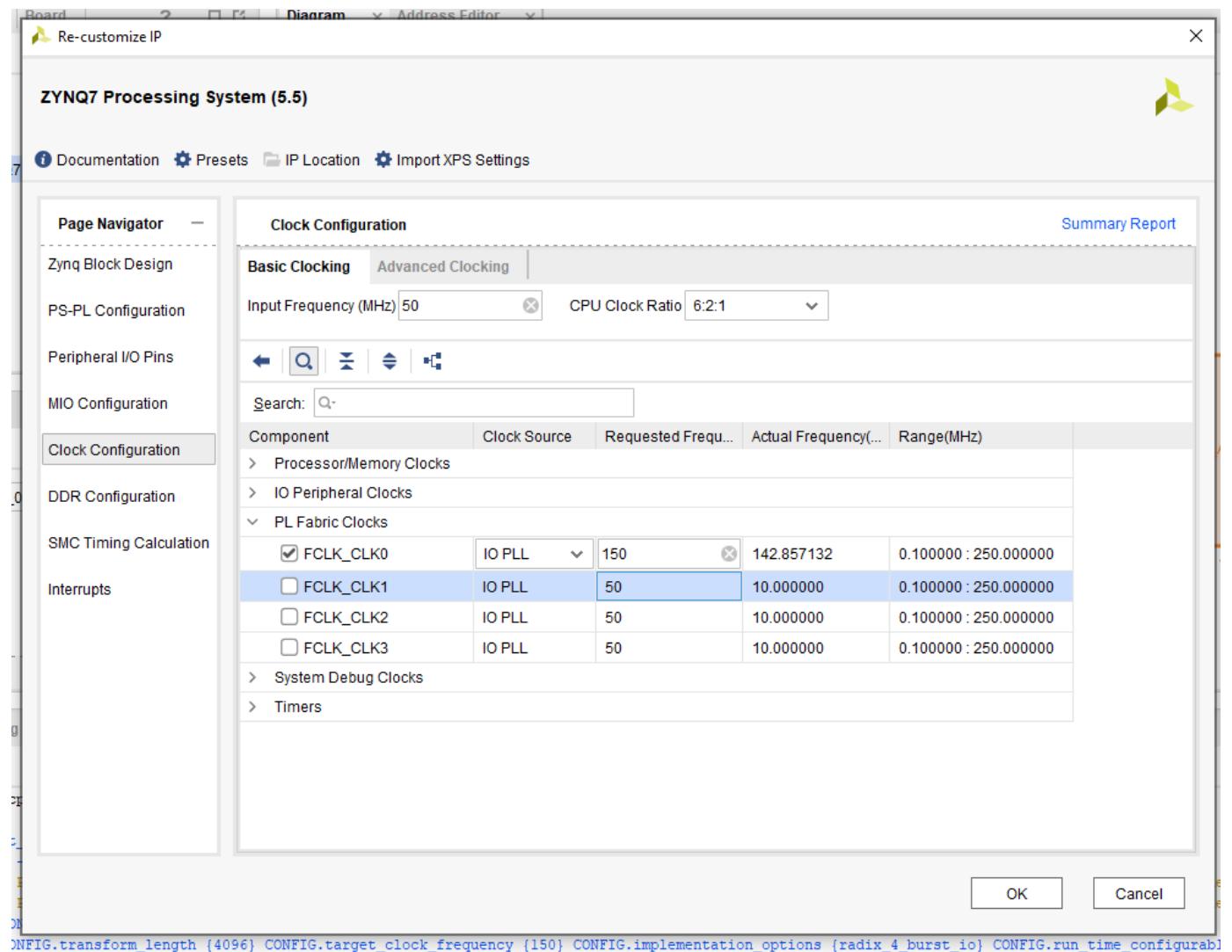
Lab : FFT Acceleration

Double click on the Processing System to reconfigure it



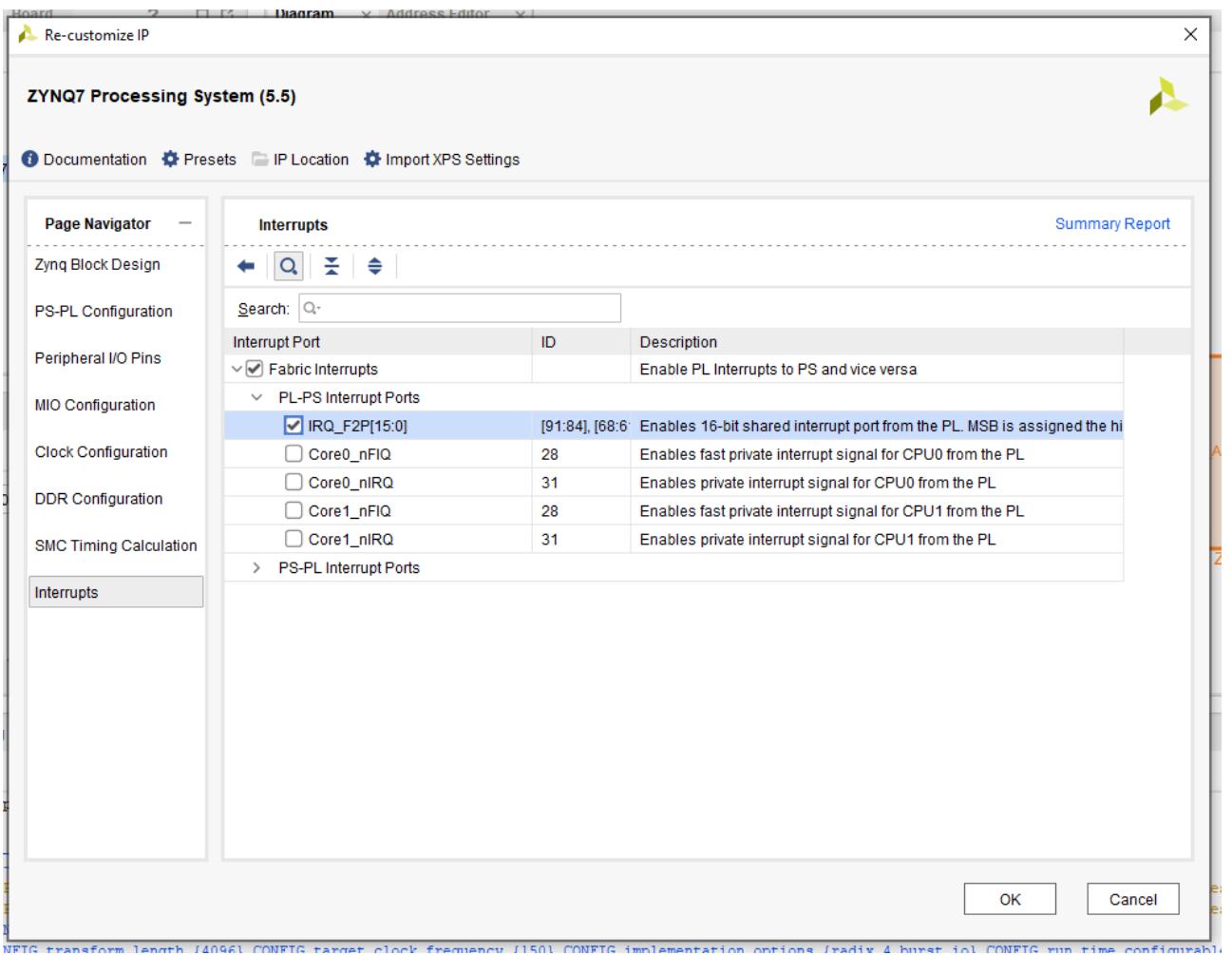
Lab : FFT Acceleration

On the clocking tab change the frequency of clock one to 150MHz



Lab : FFT Acceleration

On the Interrupts Tab enable the
IRQ_F2P[15:0]

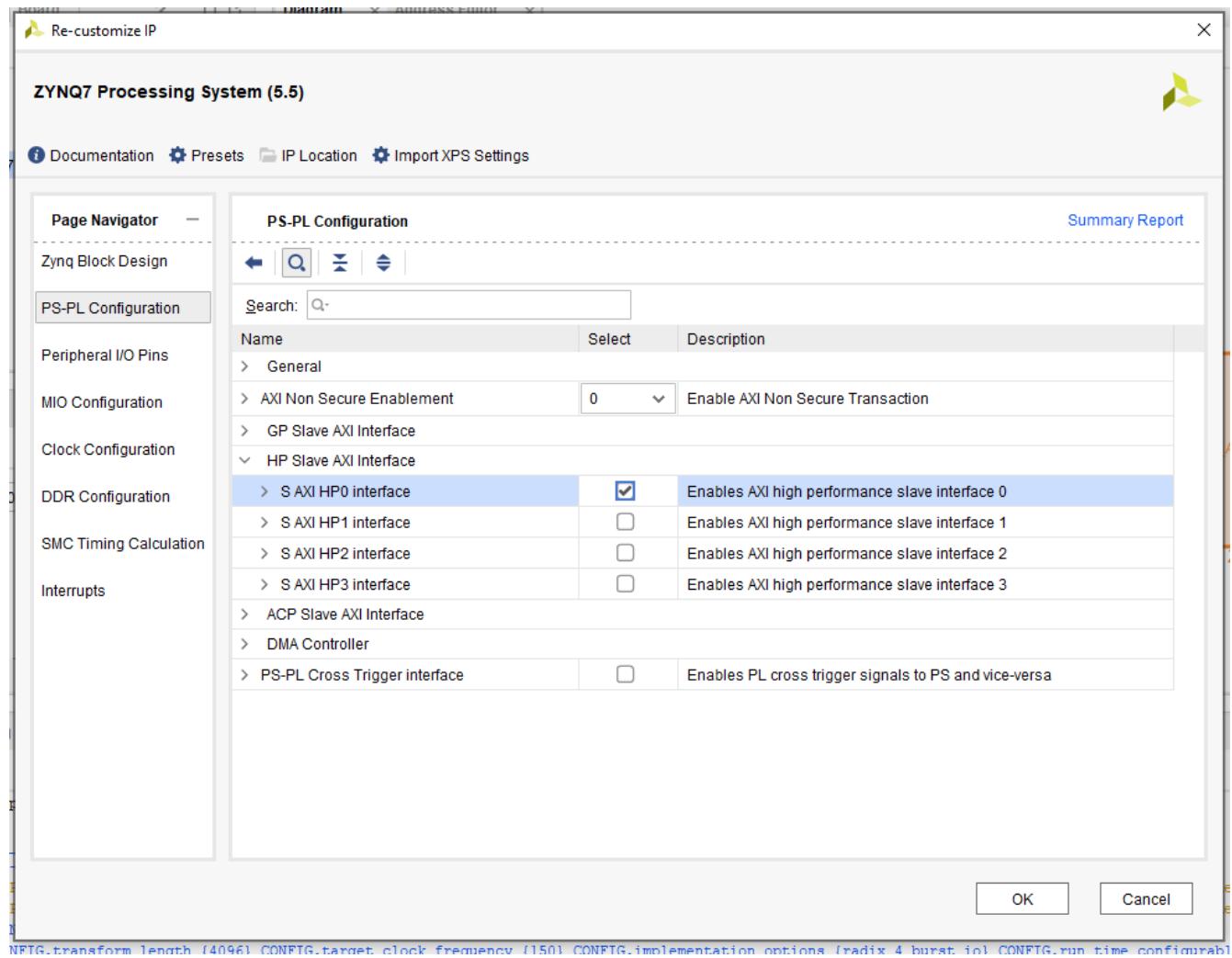


Lab : FFT Acceleration

On the PS/PL interface select the HP

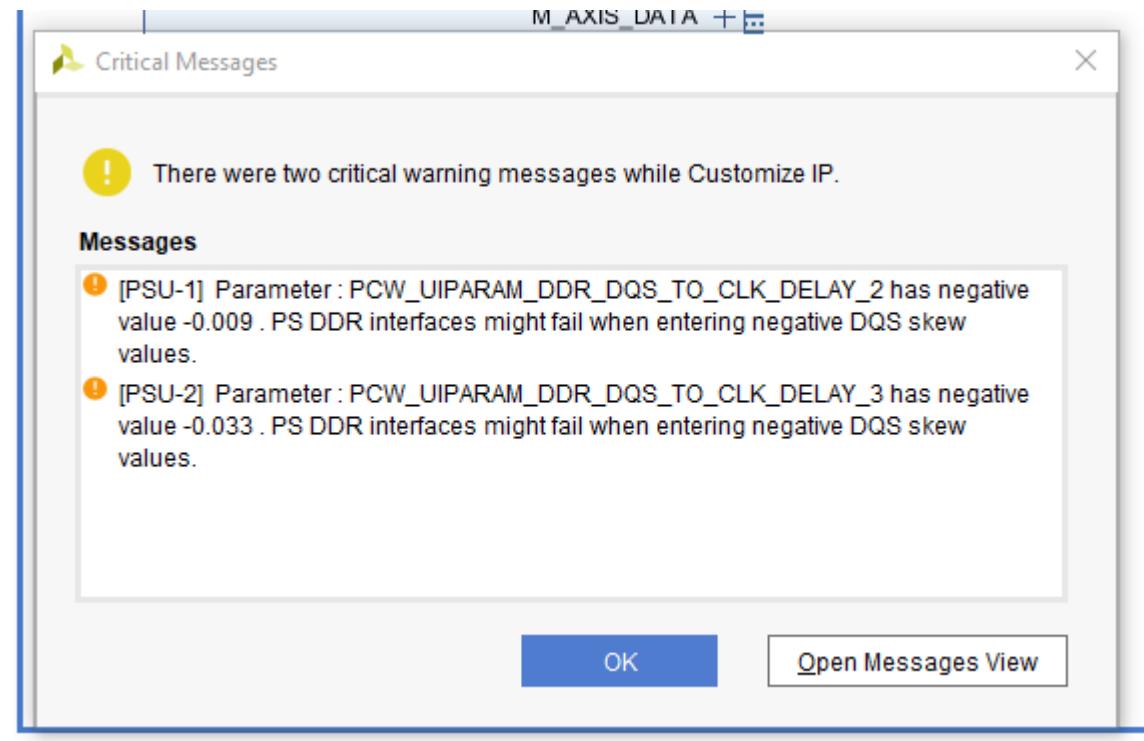
Slave AXI Interface

Enable S AXI HP0 Interface



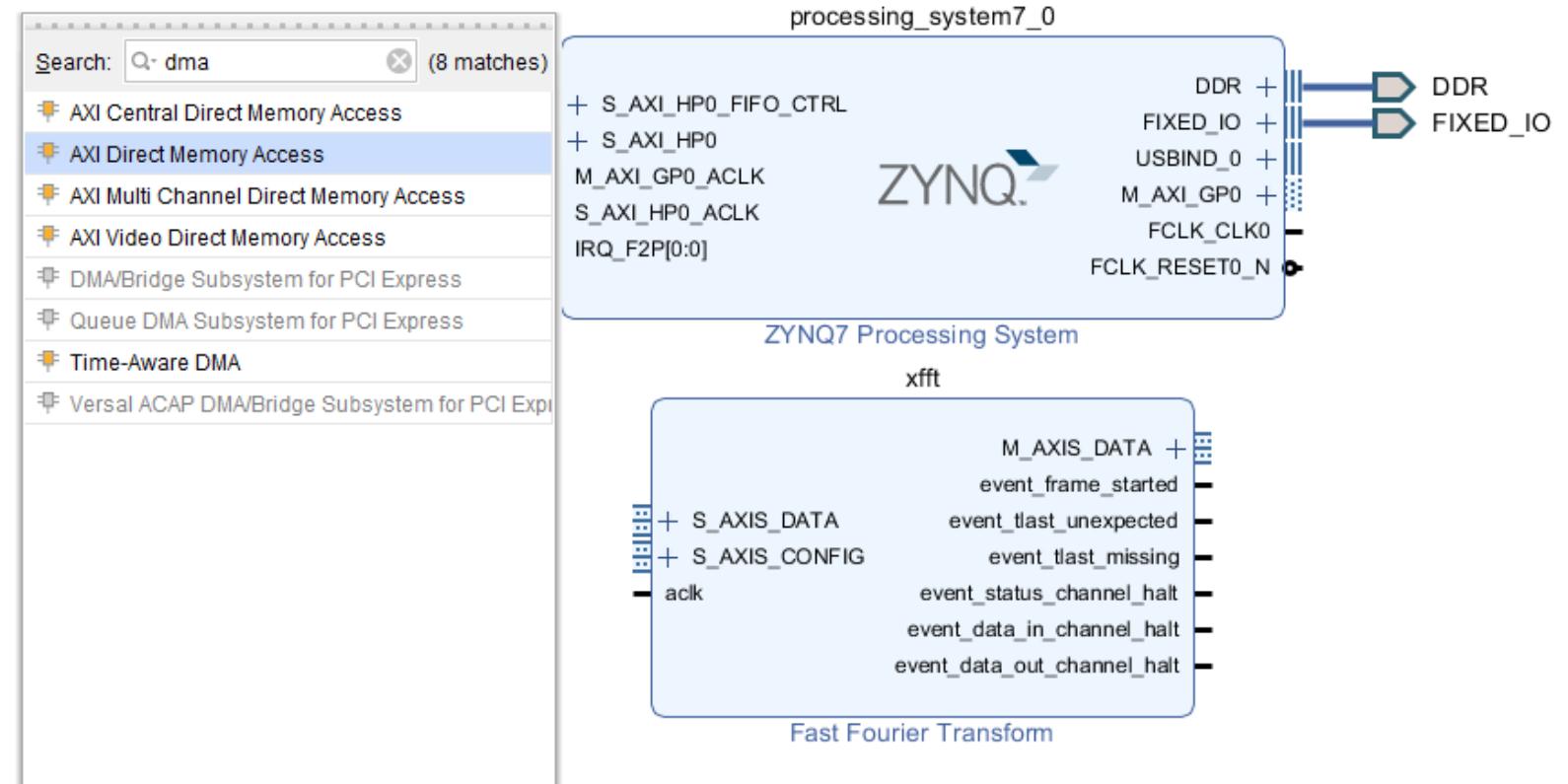
Lab : FFT Acceleration

Click OK if the warning appears



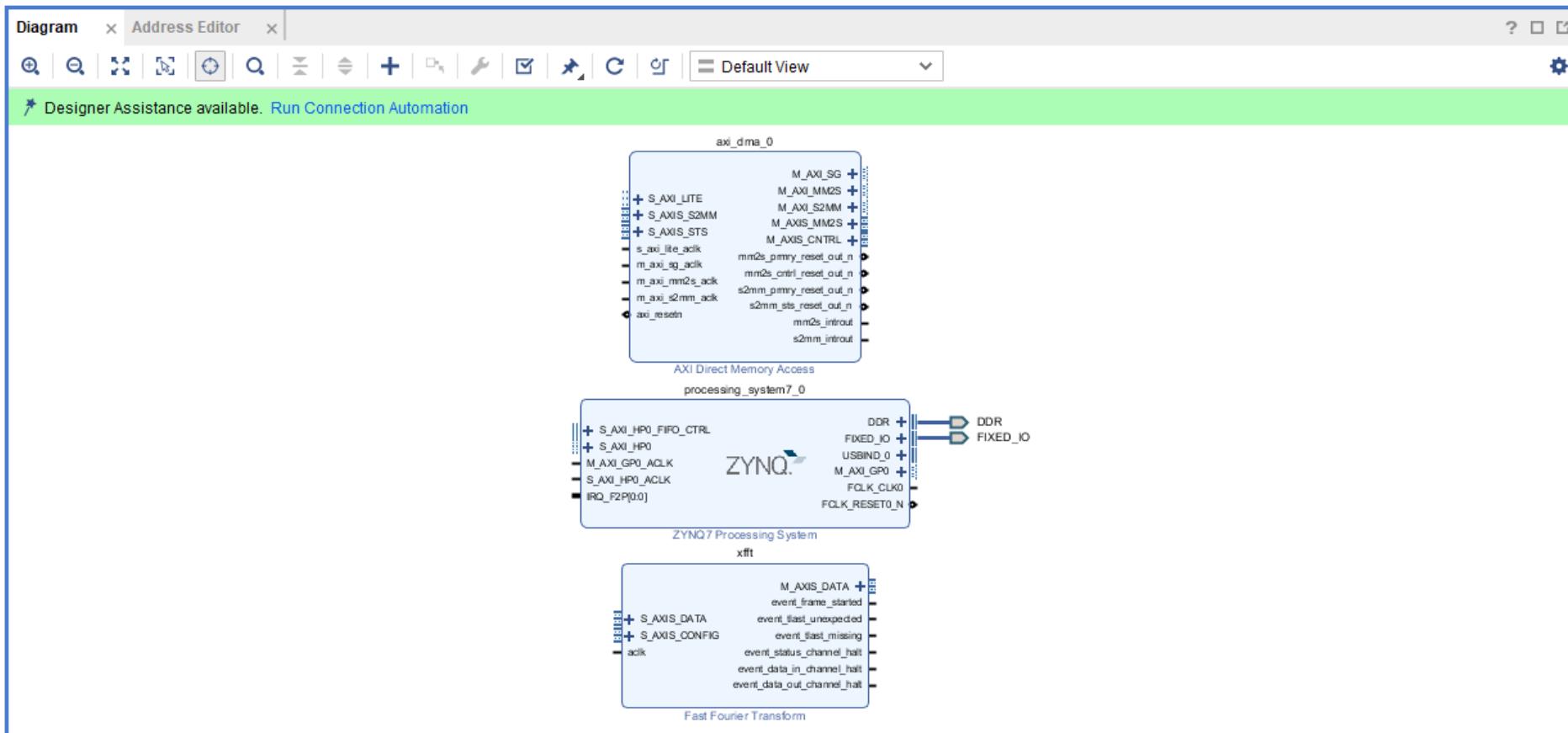
Lab : FFT Acceleration

Click + and select AXI
Direct Memory Access



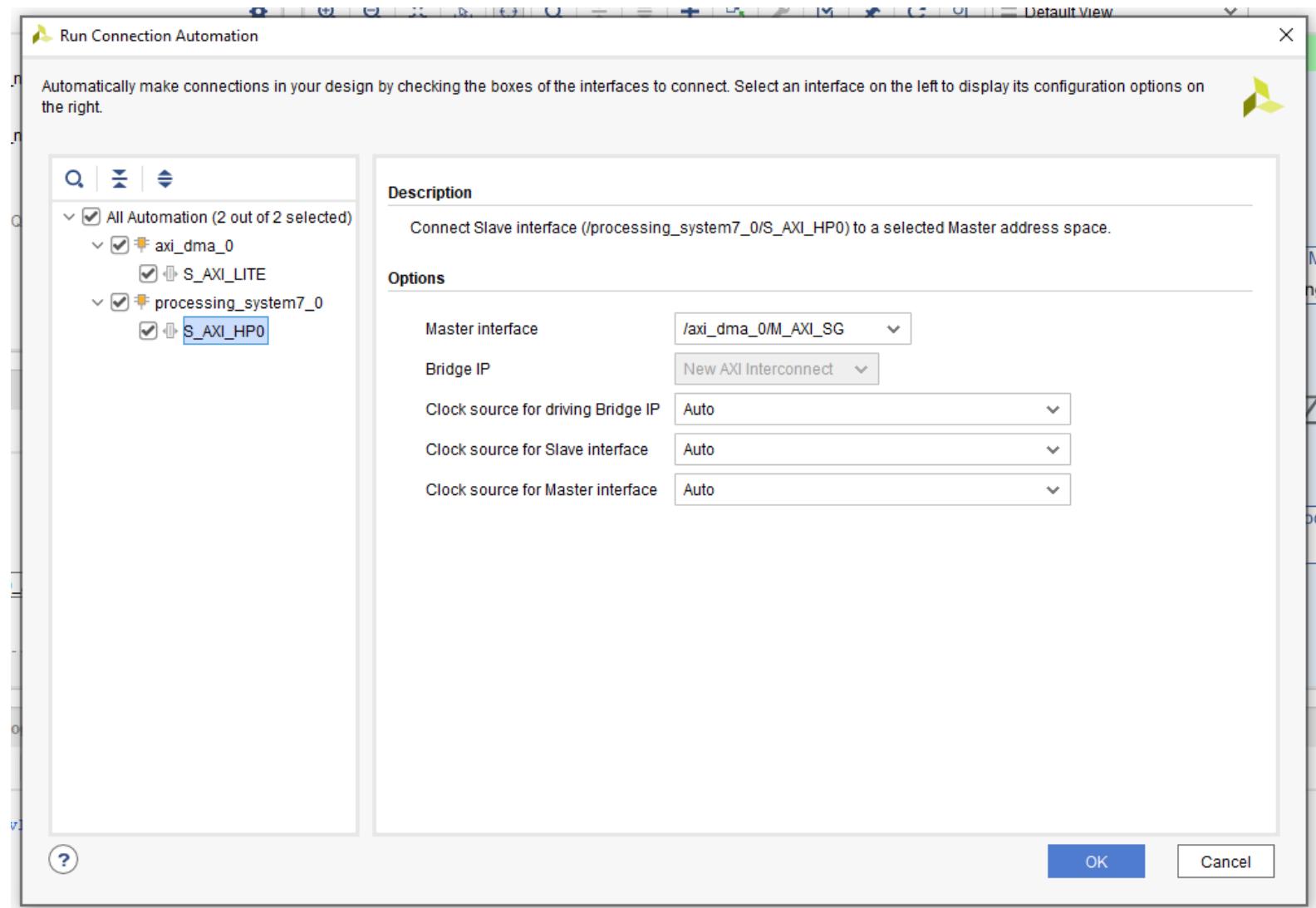
Lab : FFT Acceleration

Run the Connection Automation



Lab : FFT Acceleration

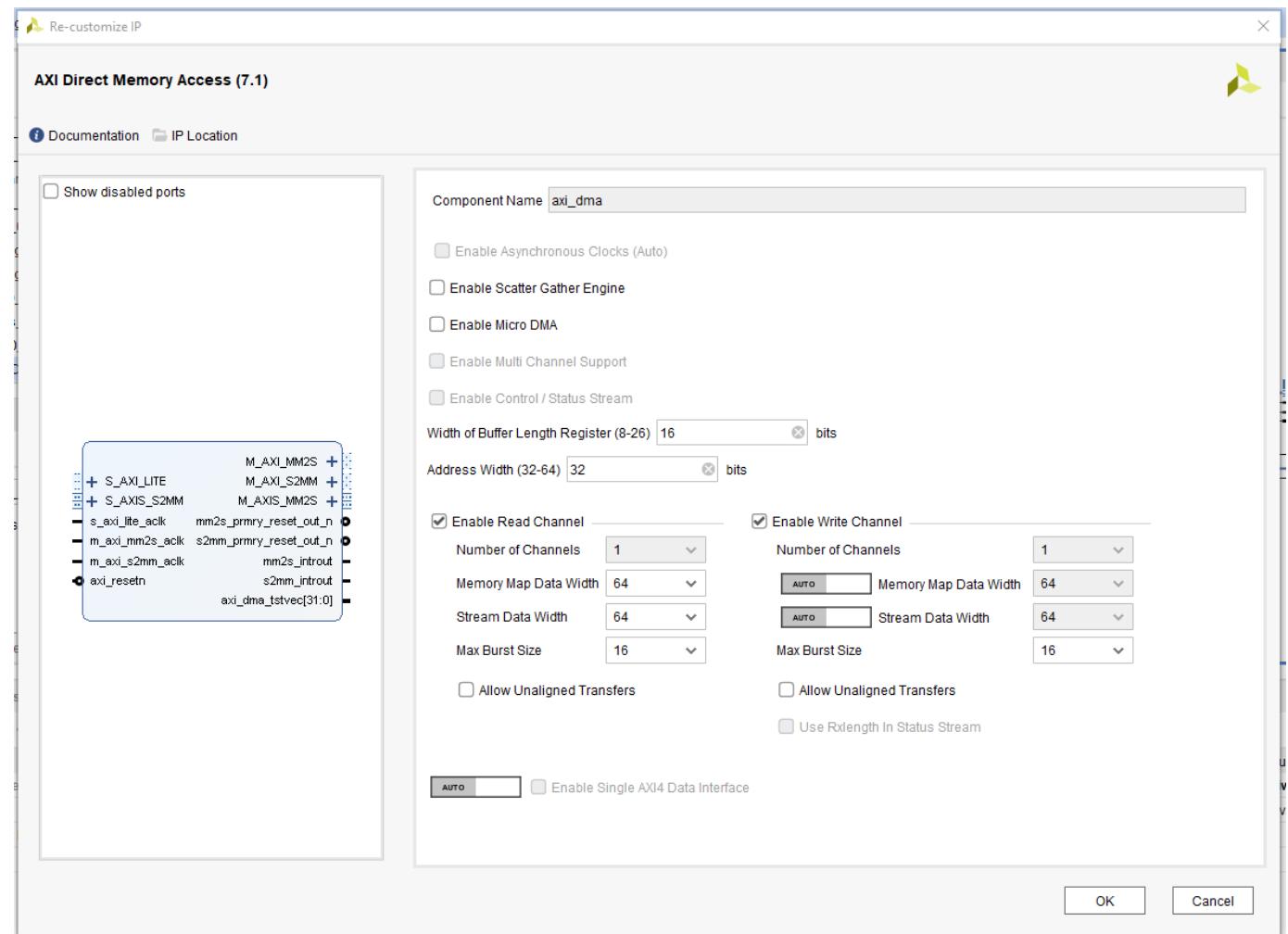
Leave the defaults as
standard and click OK



Lab : FFT Acceleration

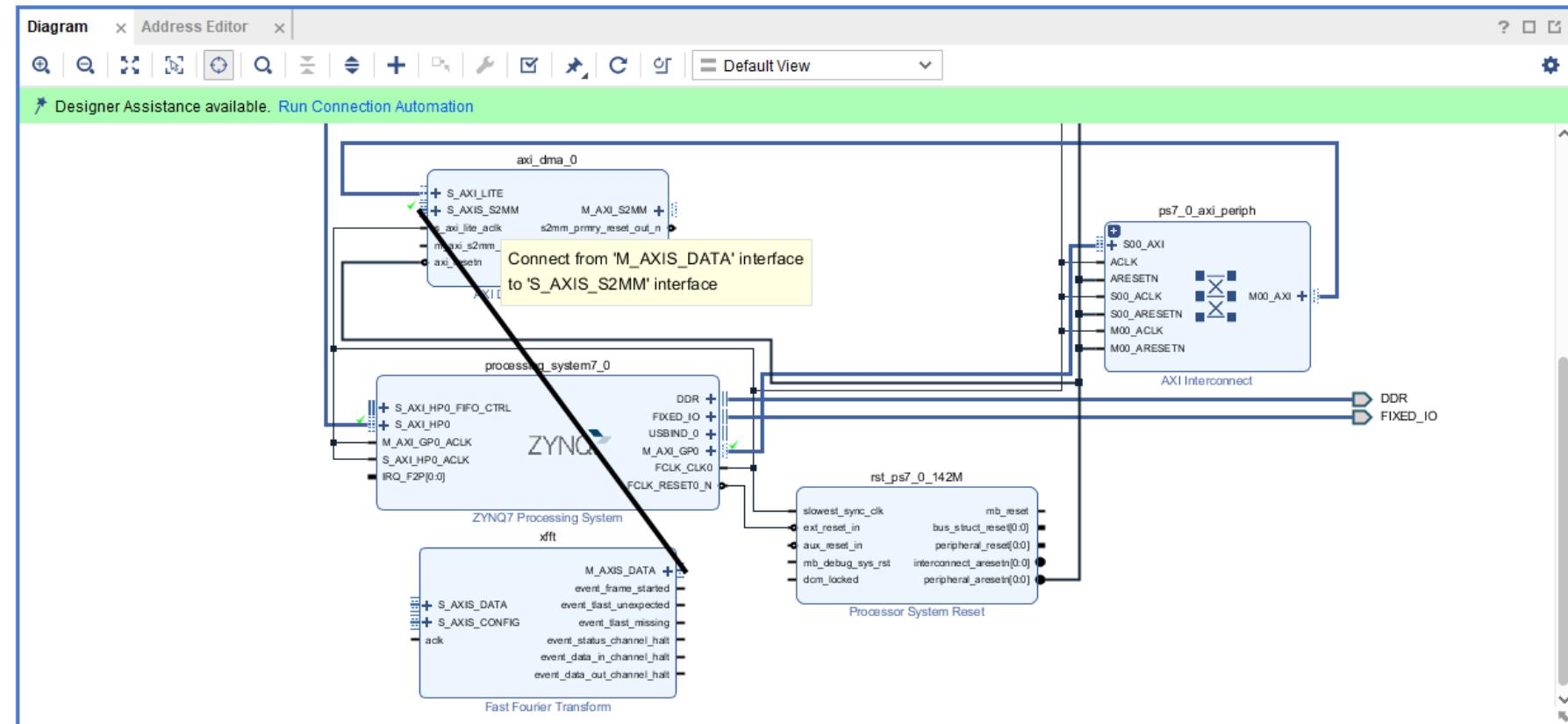
Select the DMA, double click on it
and configure it

- Width of Buffer length 16
- Stream data width 64
- Max burst size 16



Lab : FFT Acceleration

Connect the xFFT M
AXIS data to the
DMA, S AXIS S2MM
port

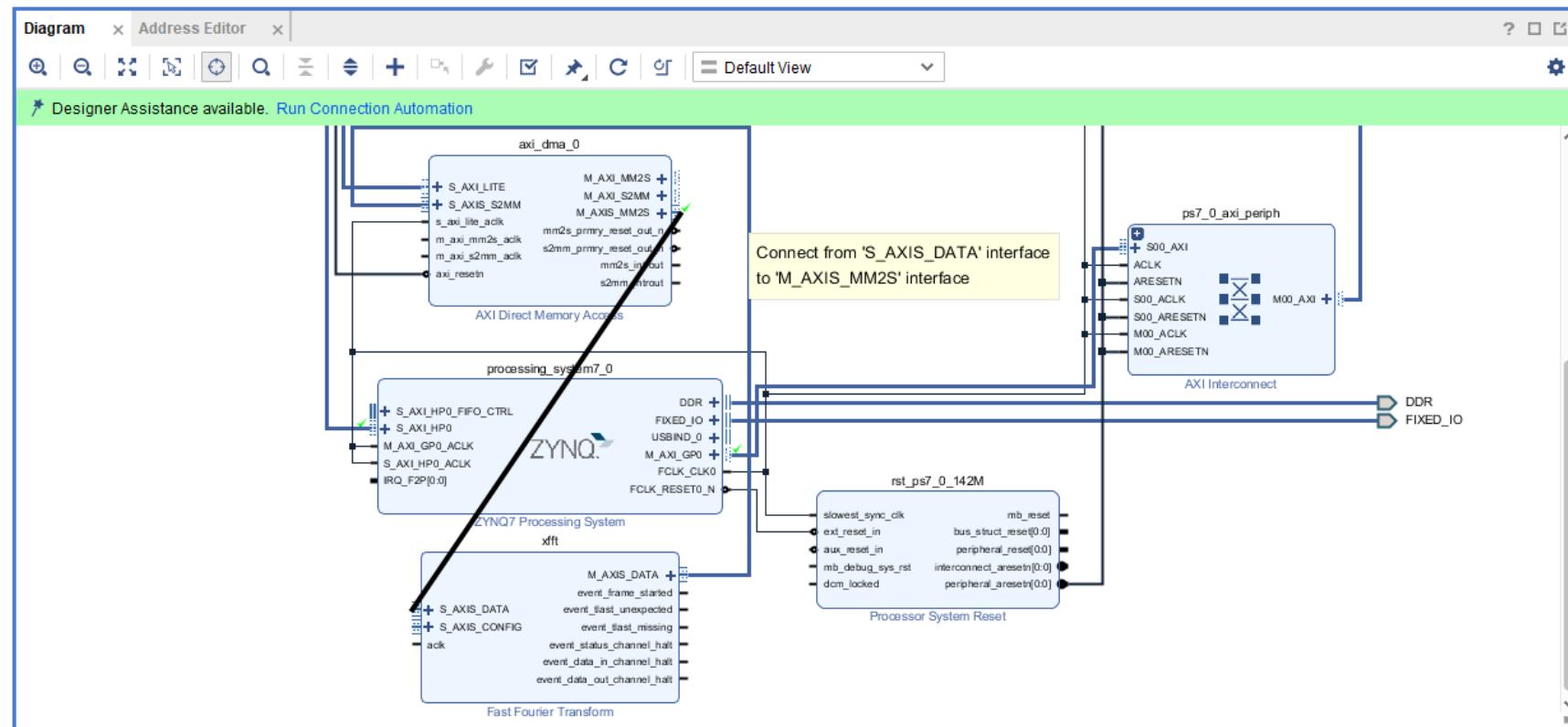


Lab : FFT Acceleration

Connect the DMA M

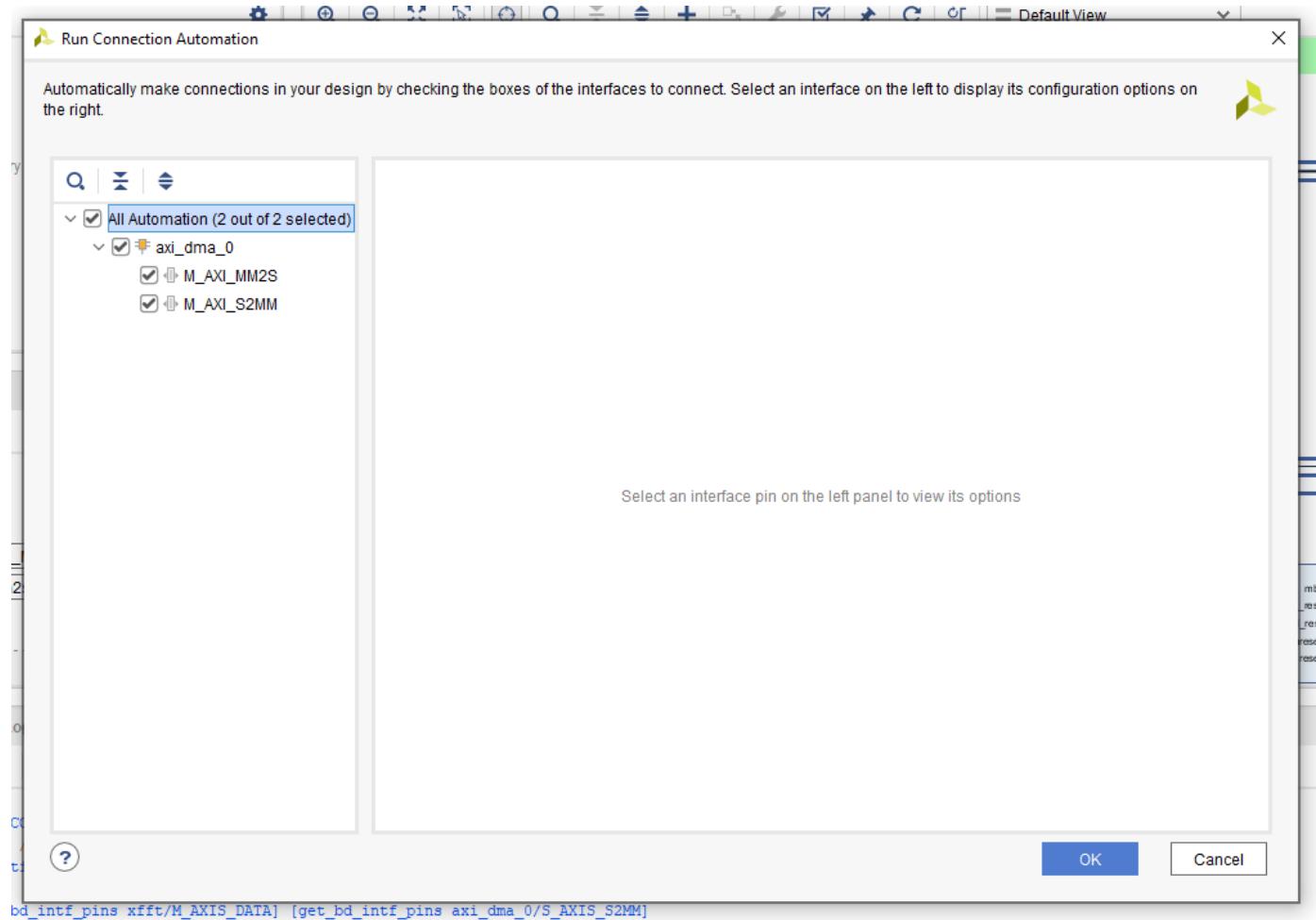
AXIS MM2S to the xFFT

S AXIS Data



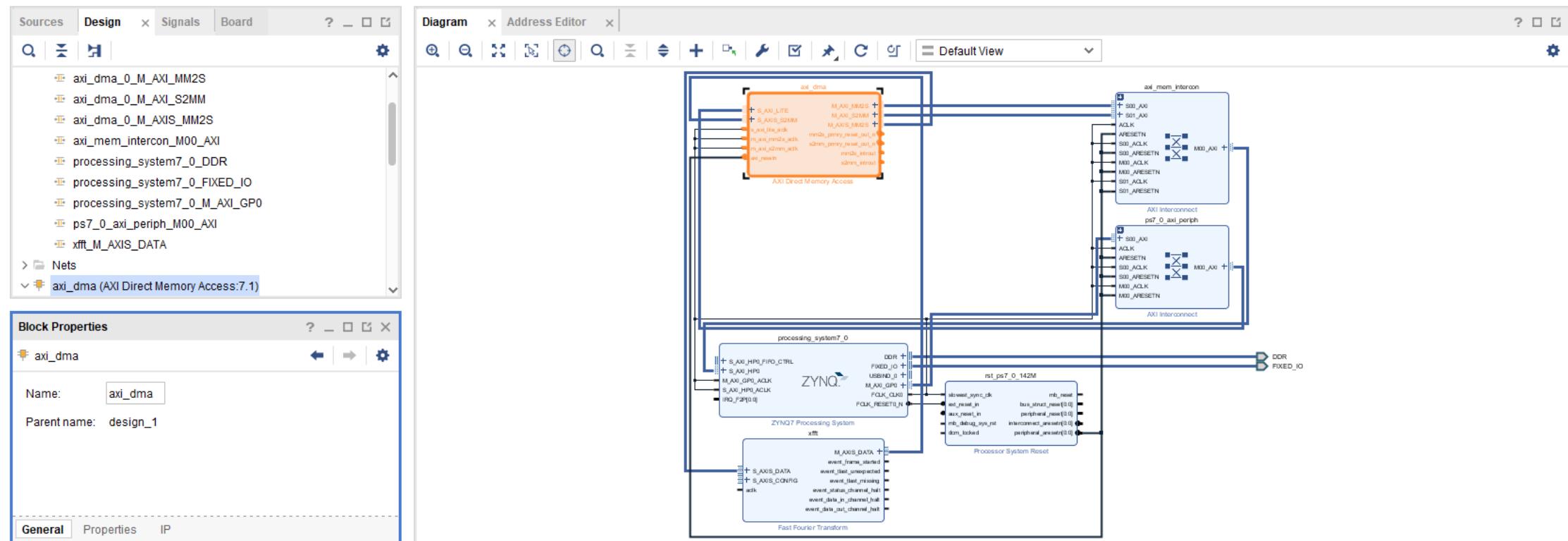
Lab : FFT Acceleration

Run the connection automation



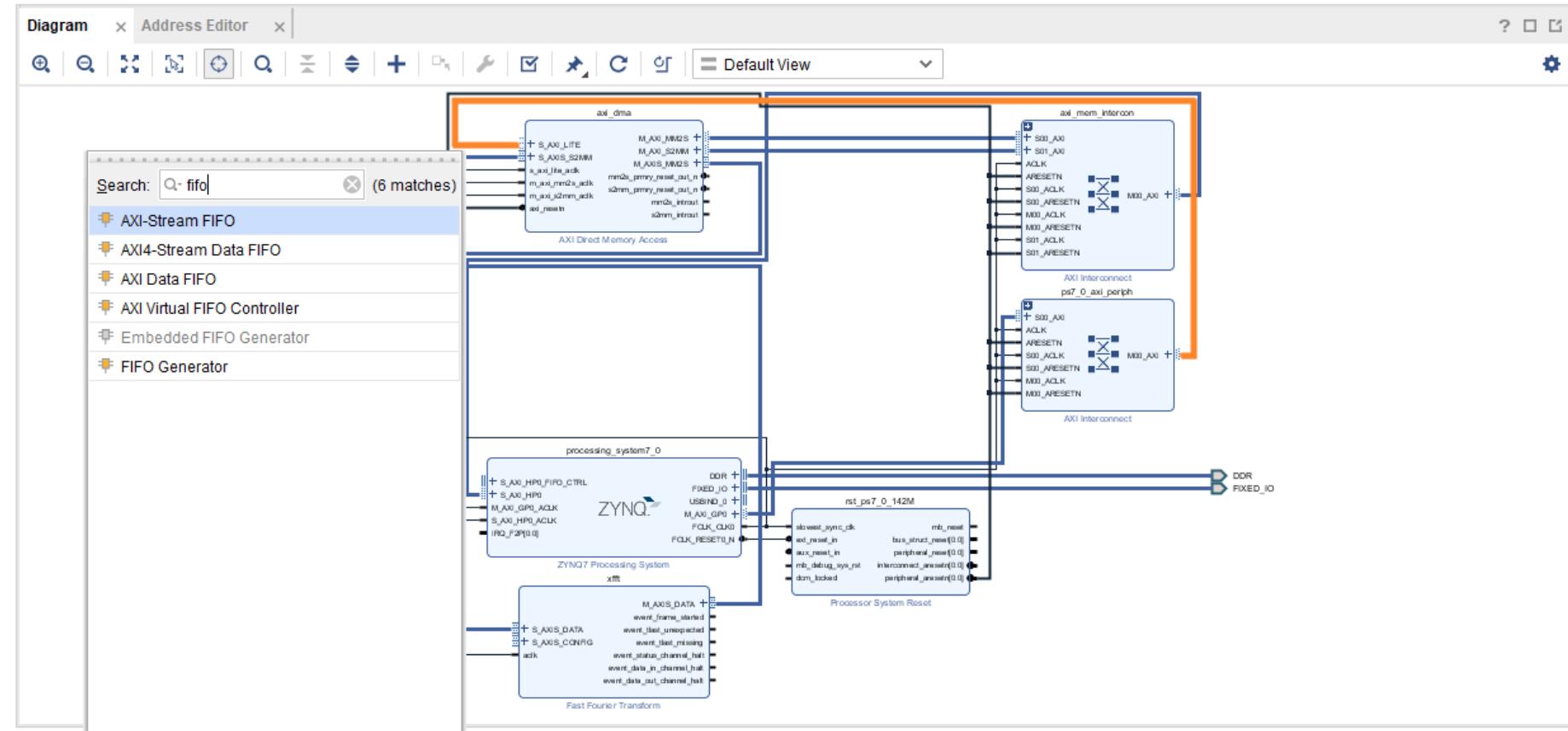
Lab : FFT Acceleration

The diagram should look like below



Lab : FFT Acceleration

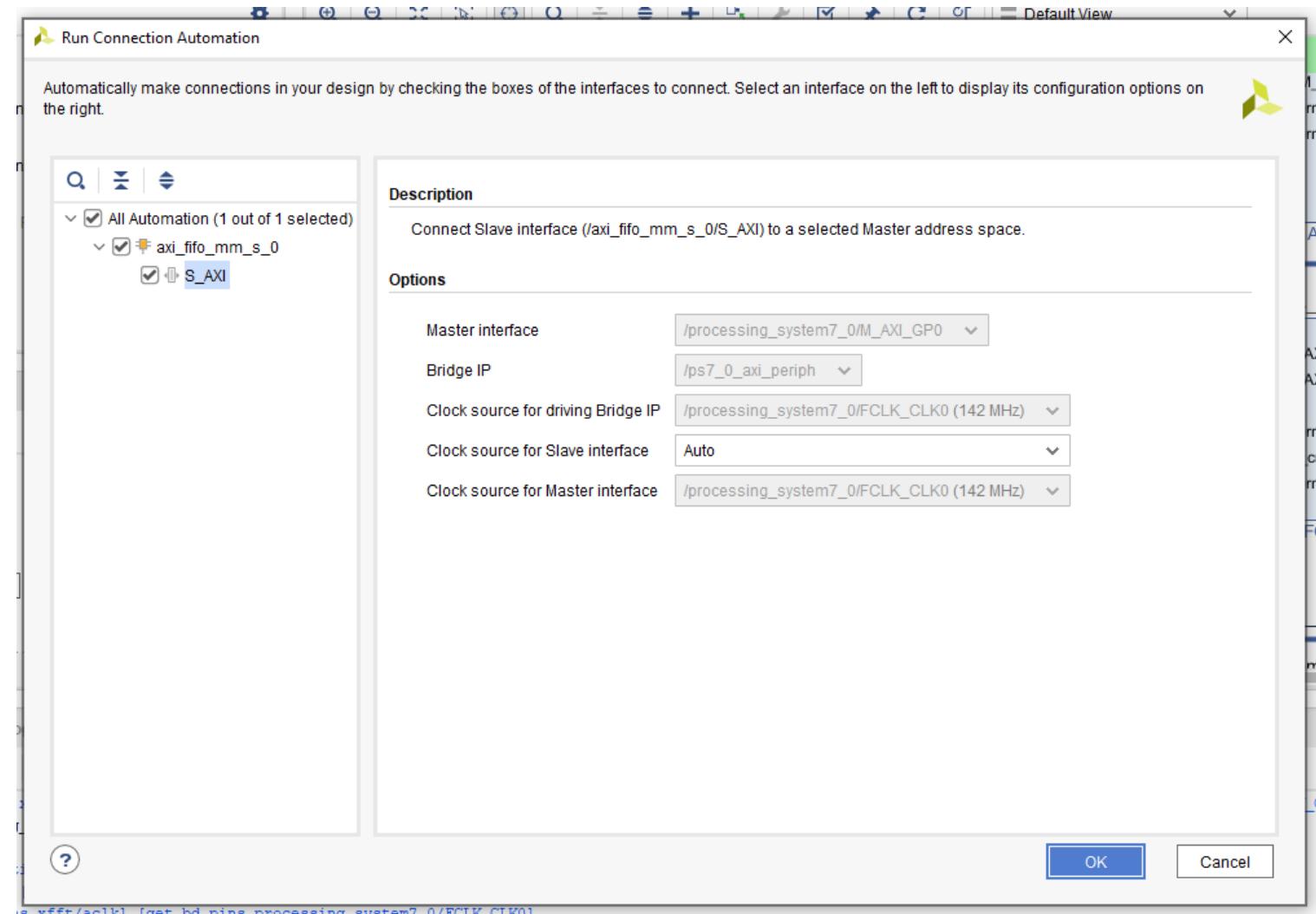
Click on + and add
in an AXI-Stream
FIFO



Lab : FFT Acceleration

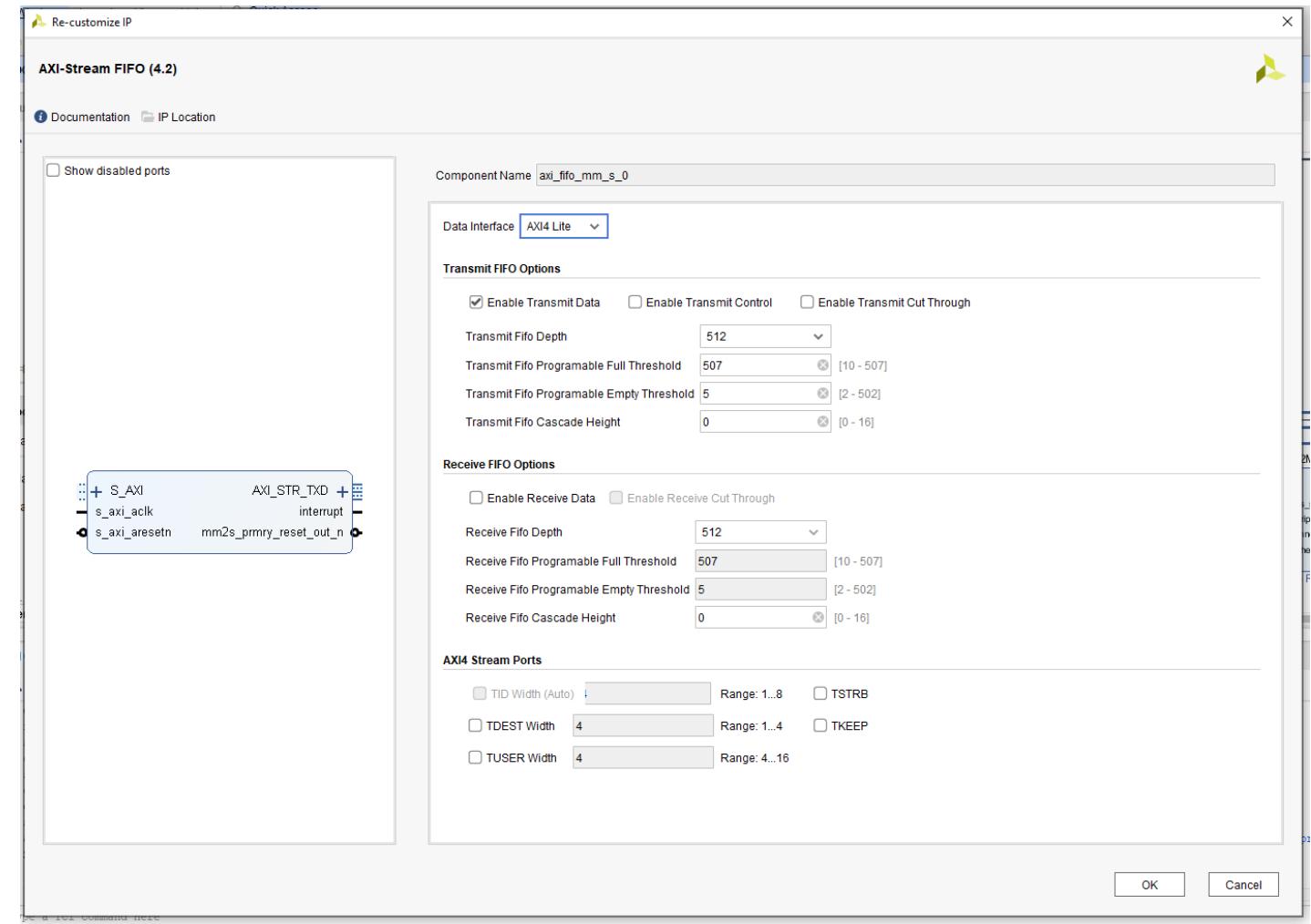
Run the connection

automation and click on OK



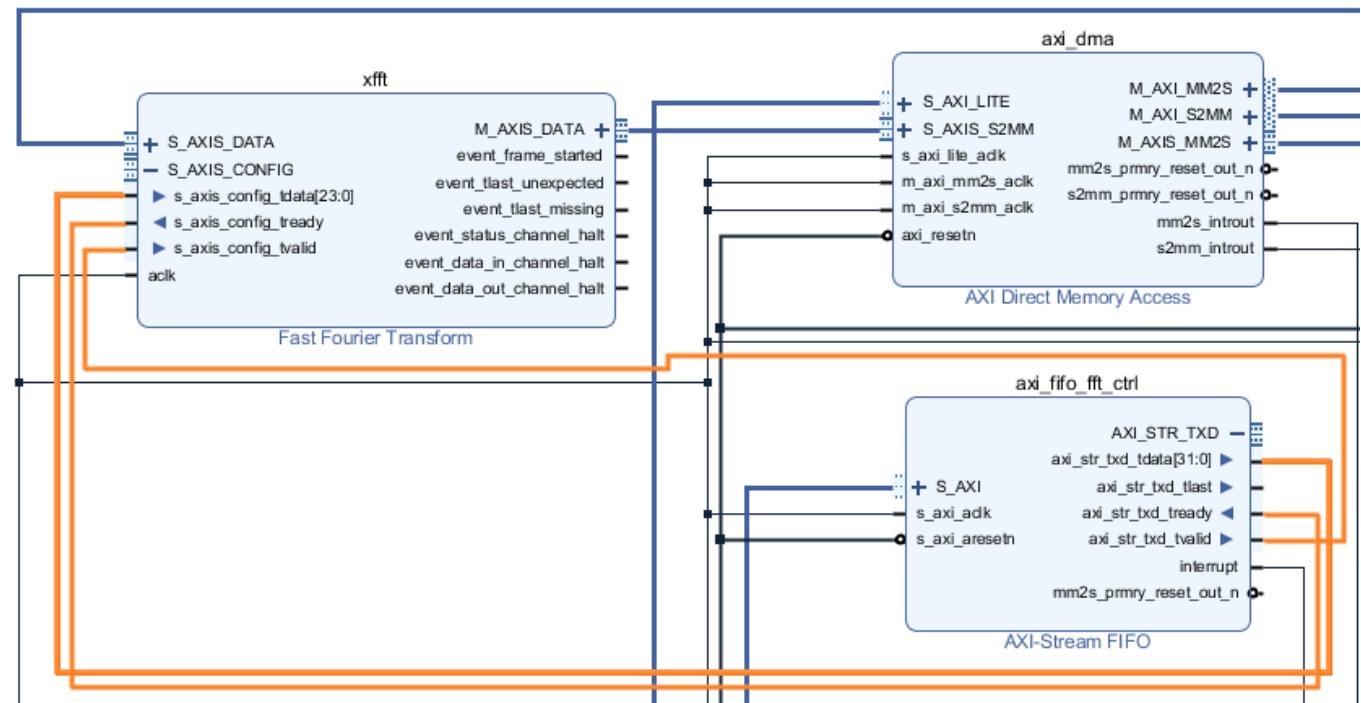
Lab : FFT Acceleration

Double click on the AXI Stream FIFO and configure it to have an AXI Lite Interface and only enable the Transmit Data Interface leave all else unchanged.



Lab : FFT Acceleration

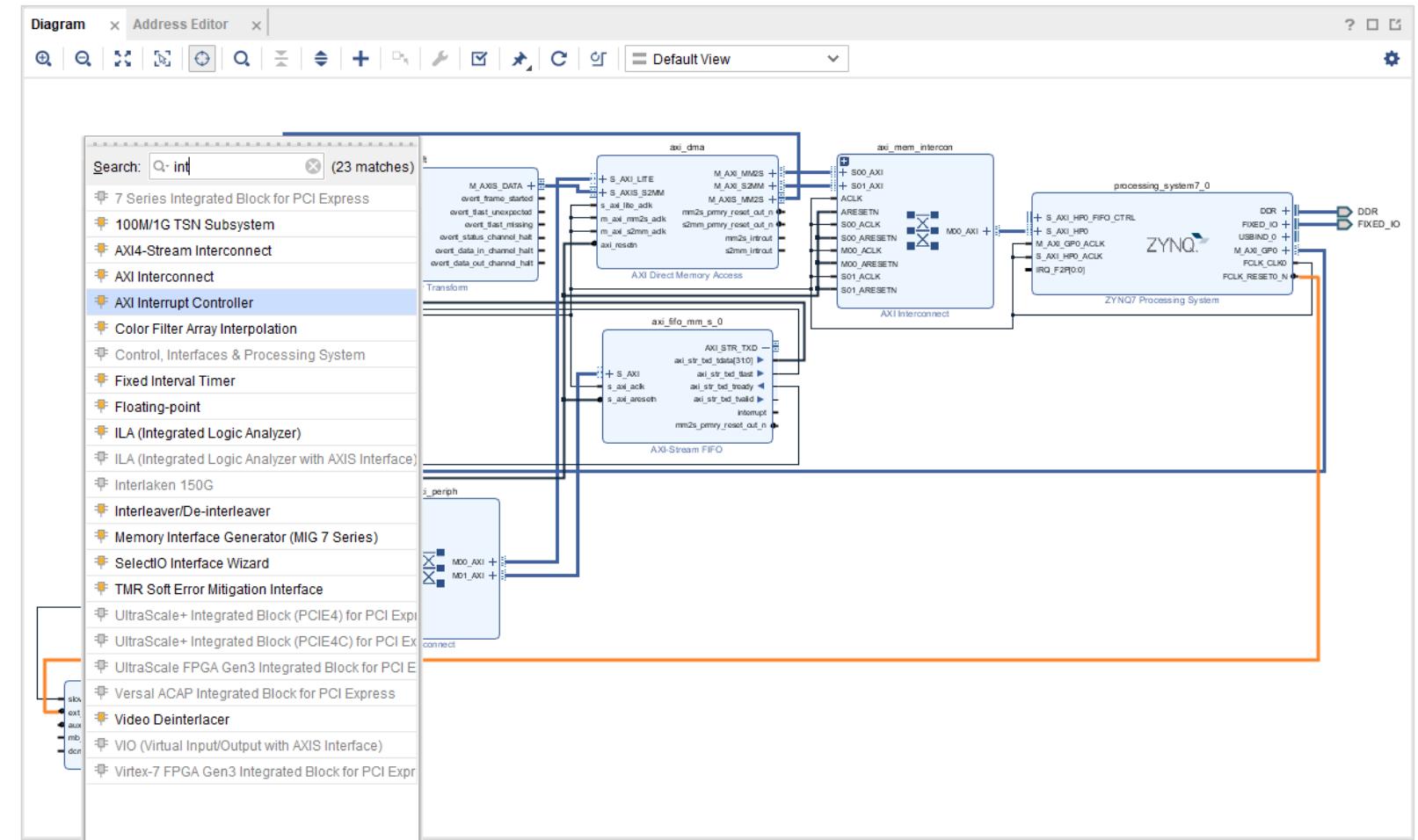
Connect the AXI STR TxD tdata,
tlast and tvalid signals to the
xFFT S AXIS config



Lab : FFT Acceleration

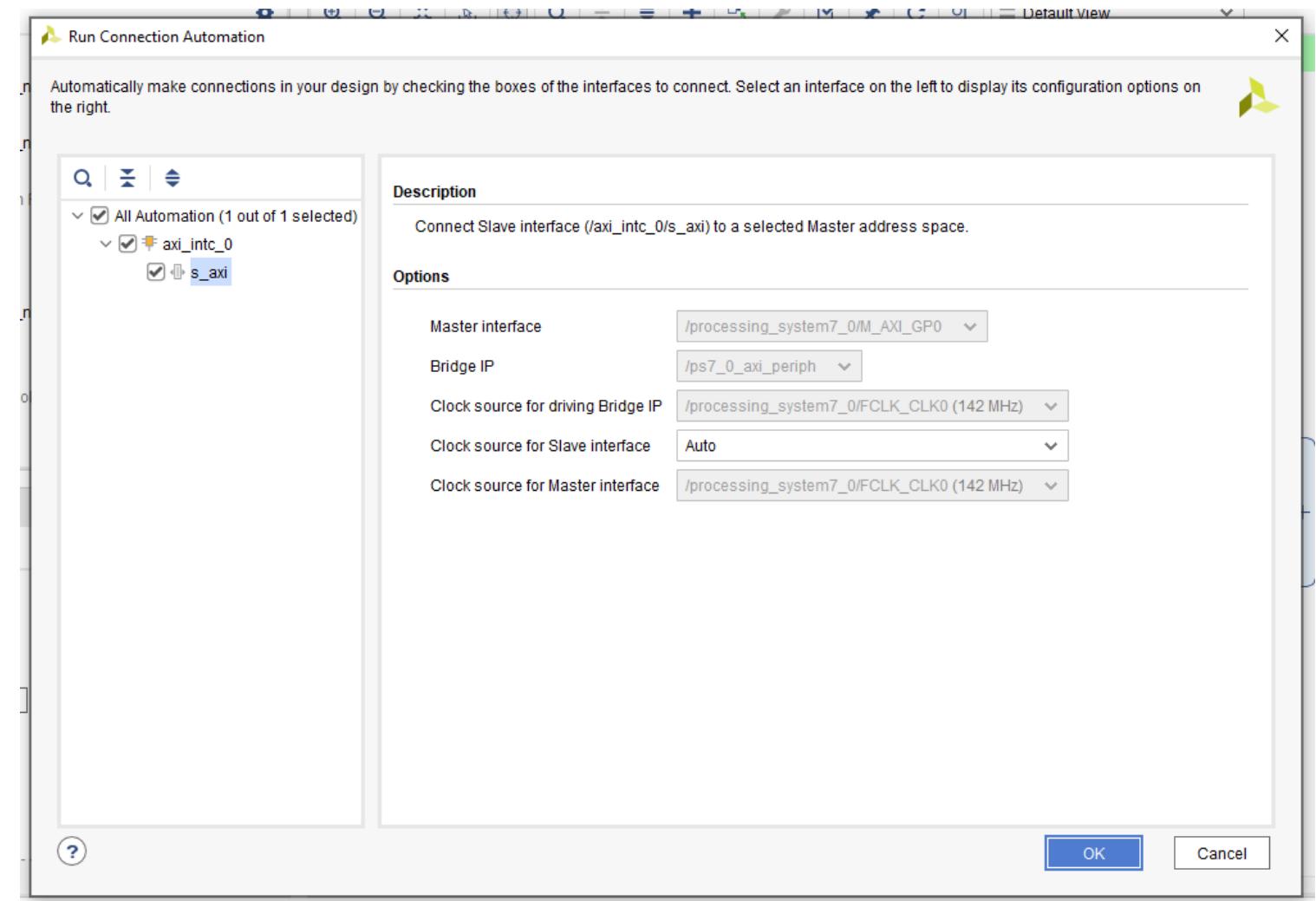
Click on + and add in a AXI

Interrupt Controller



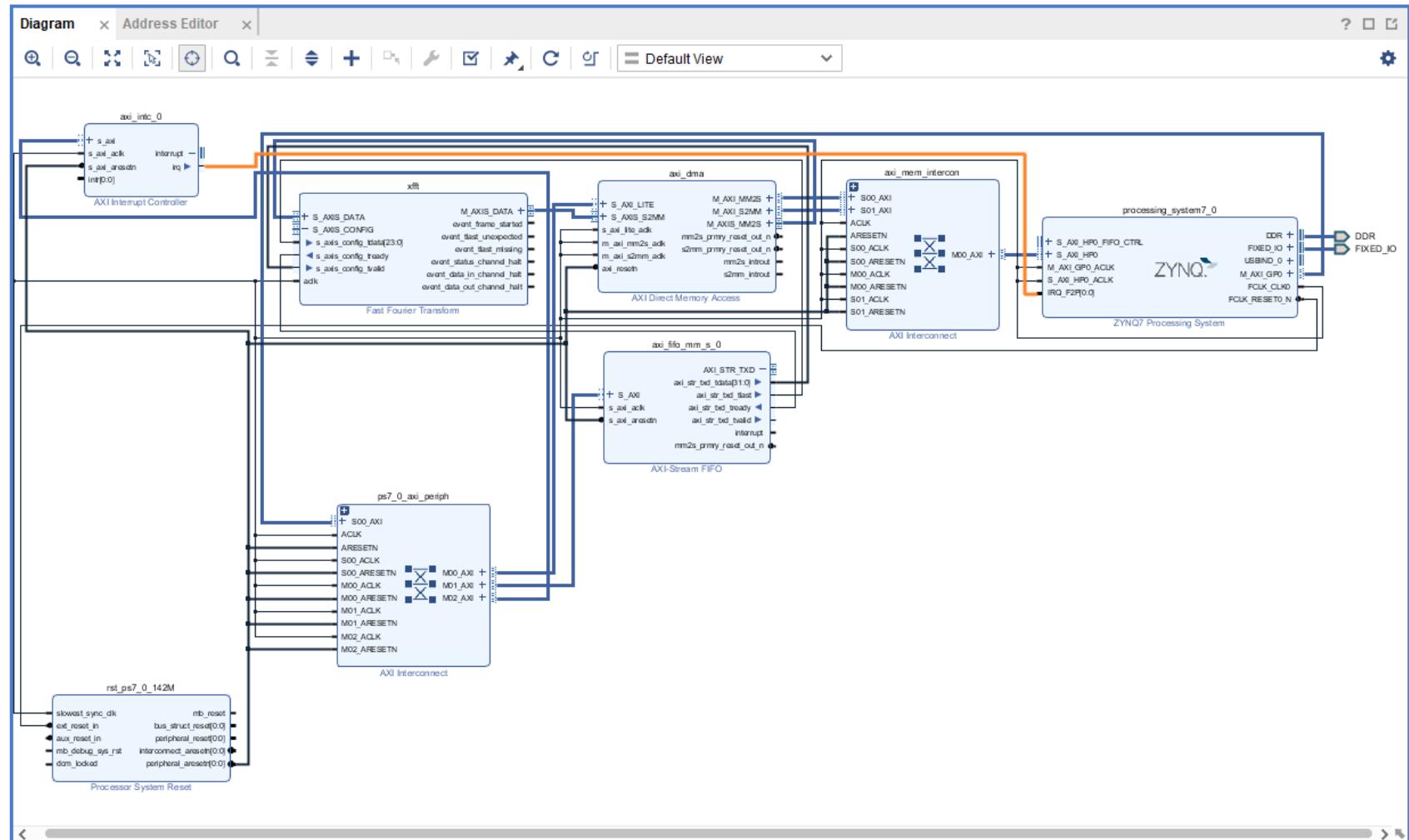
Lab : FFT Acceleration

Run the connection
automation



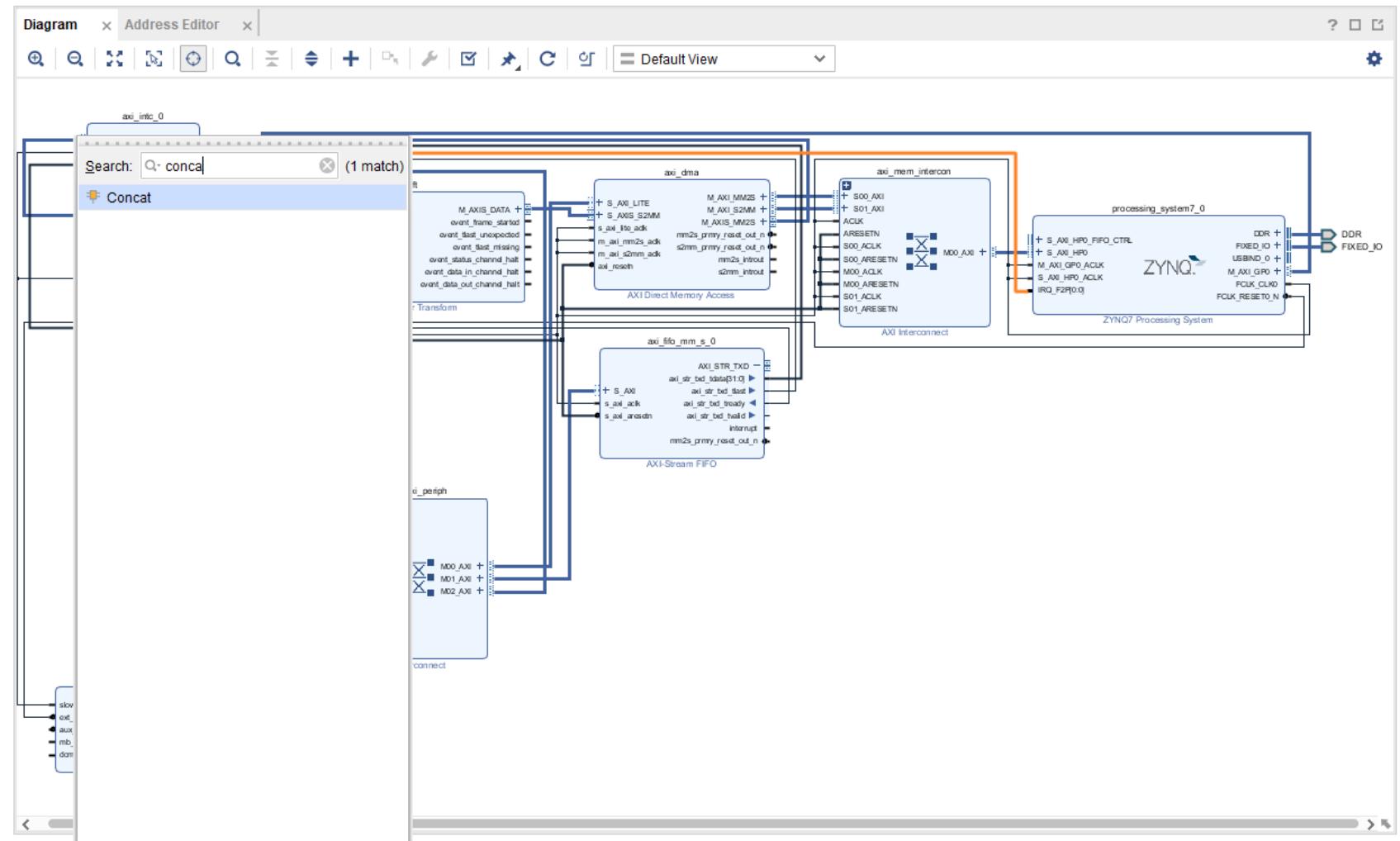
Lab : FFT Acceleration

Connect the IRQ output
from the AXI Interrupt
Controller to the
IRQF2P port on the
Zynq



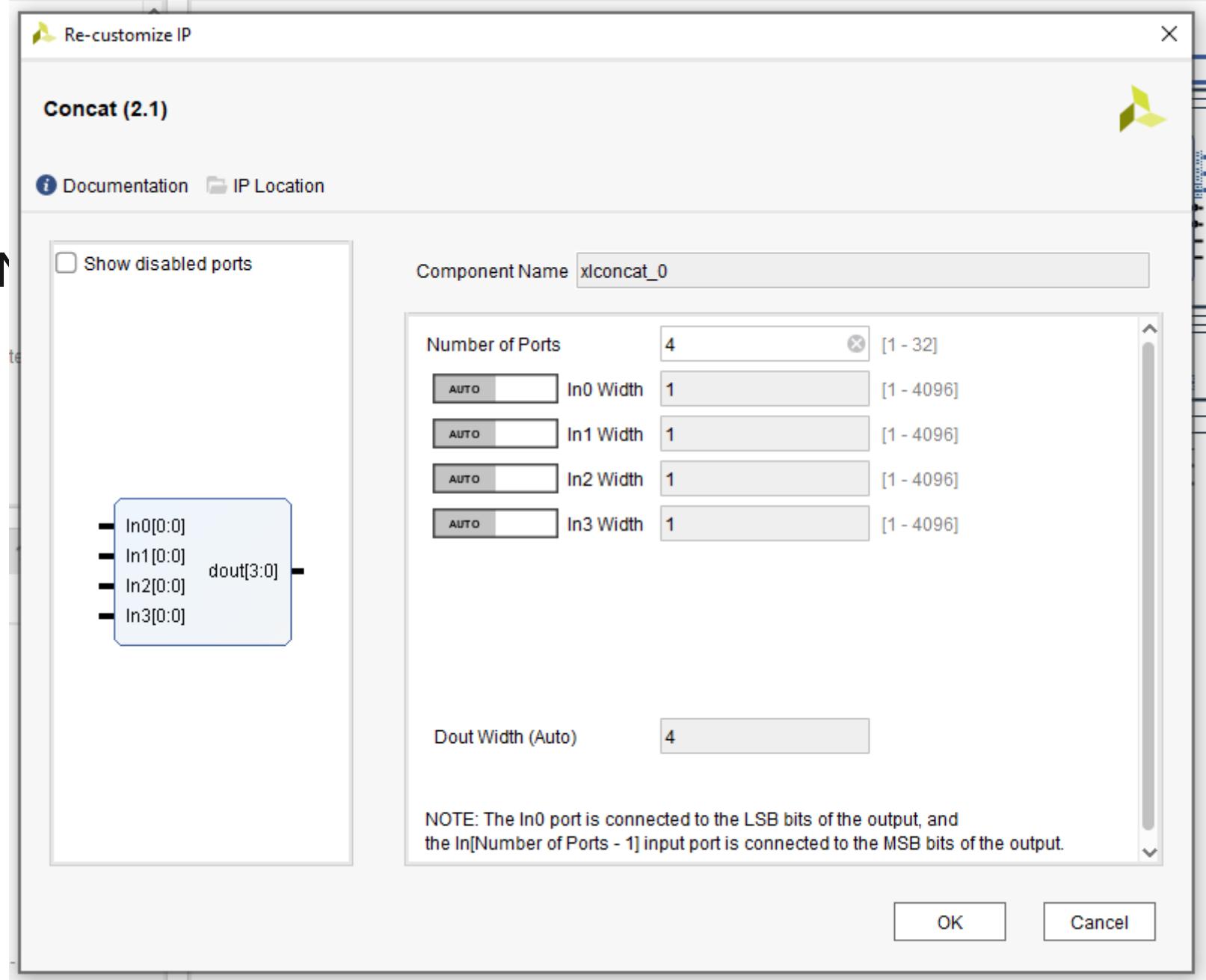
Lab : FFT Acceleration

Click on + and add in a concat block



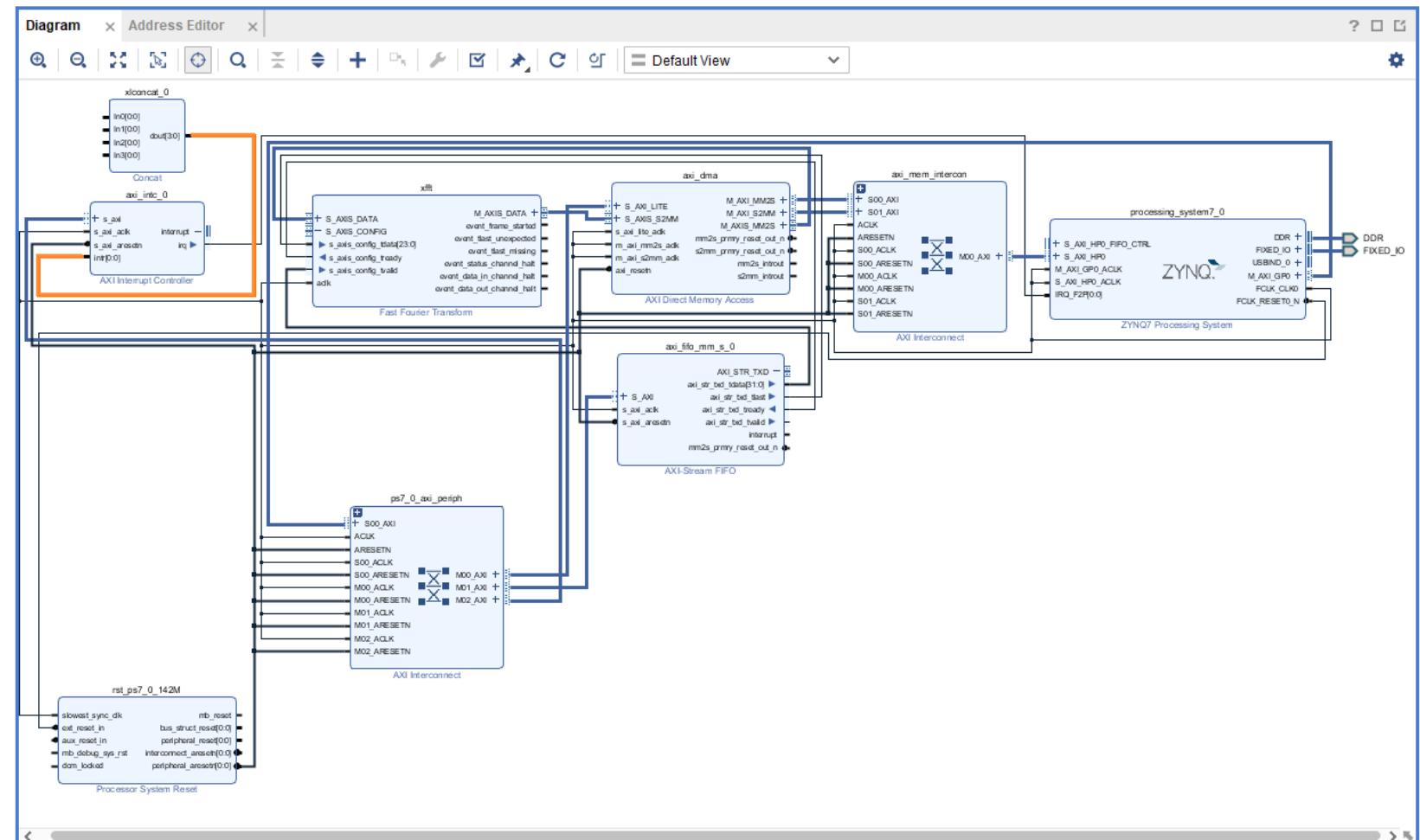
Lab :

Go to PY



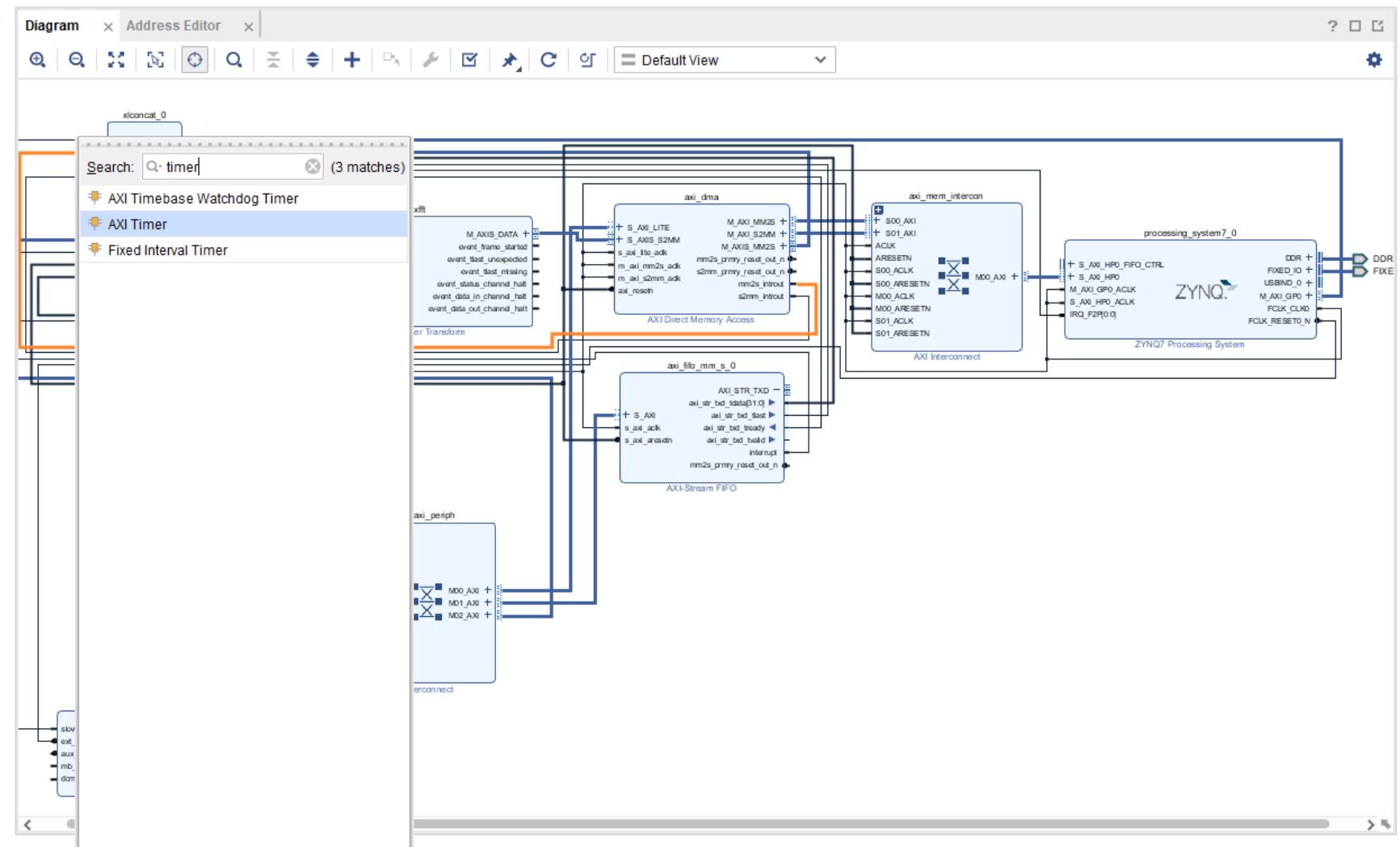
Lab : FFT Acceleration

Connect the output of the concat block to the AXI Interrupt controller INT input



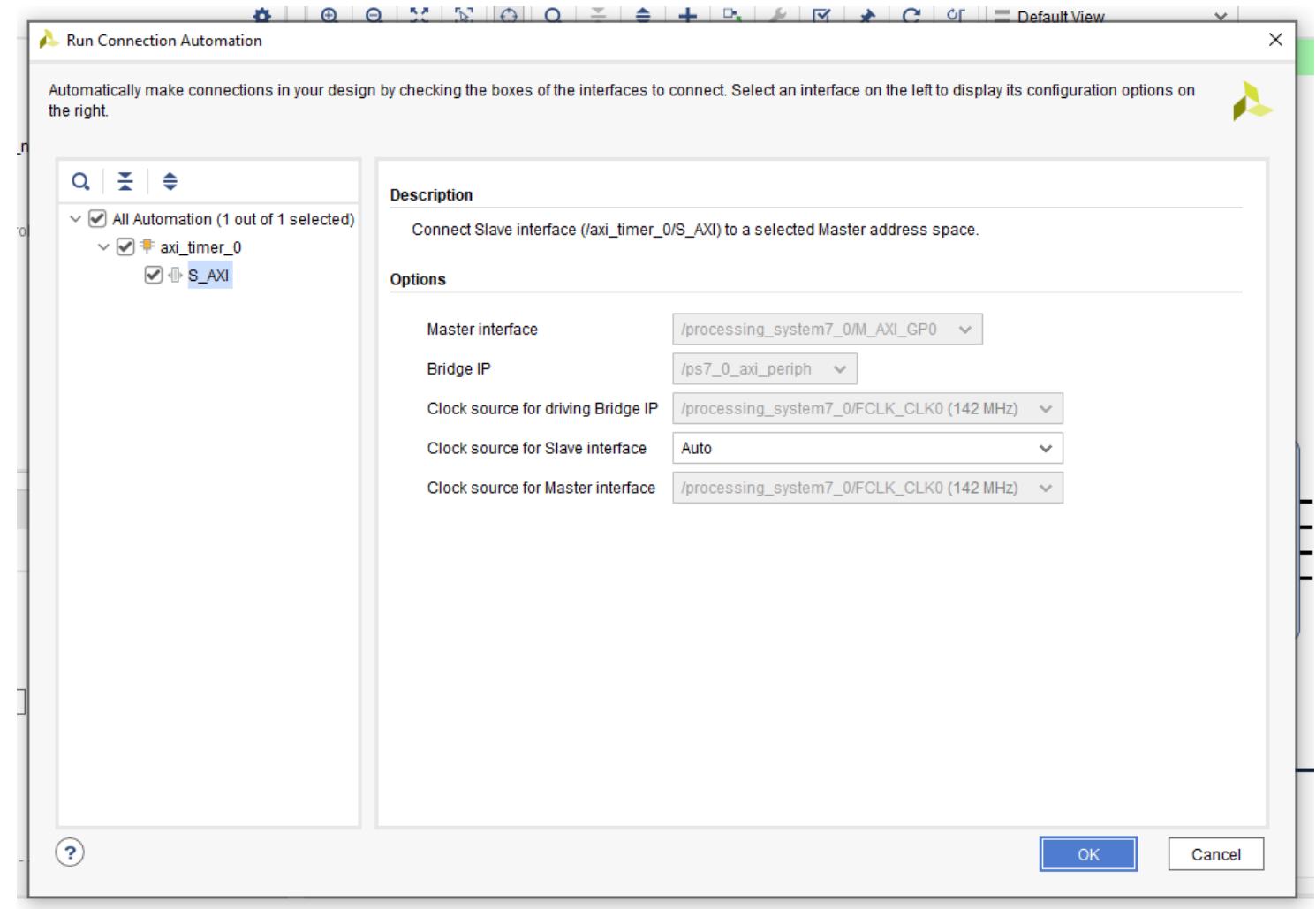
Lab : FFT Acceleration

Click on the + symbol
and add in a AXI Timer



Lab : FFT Acceleration

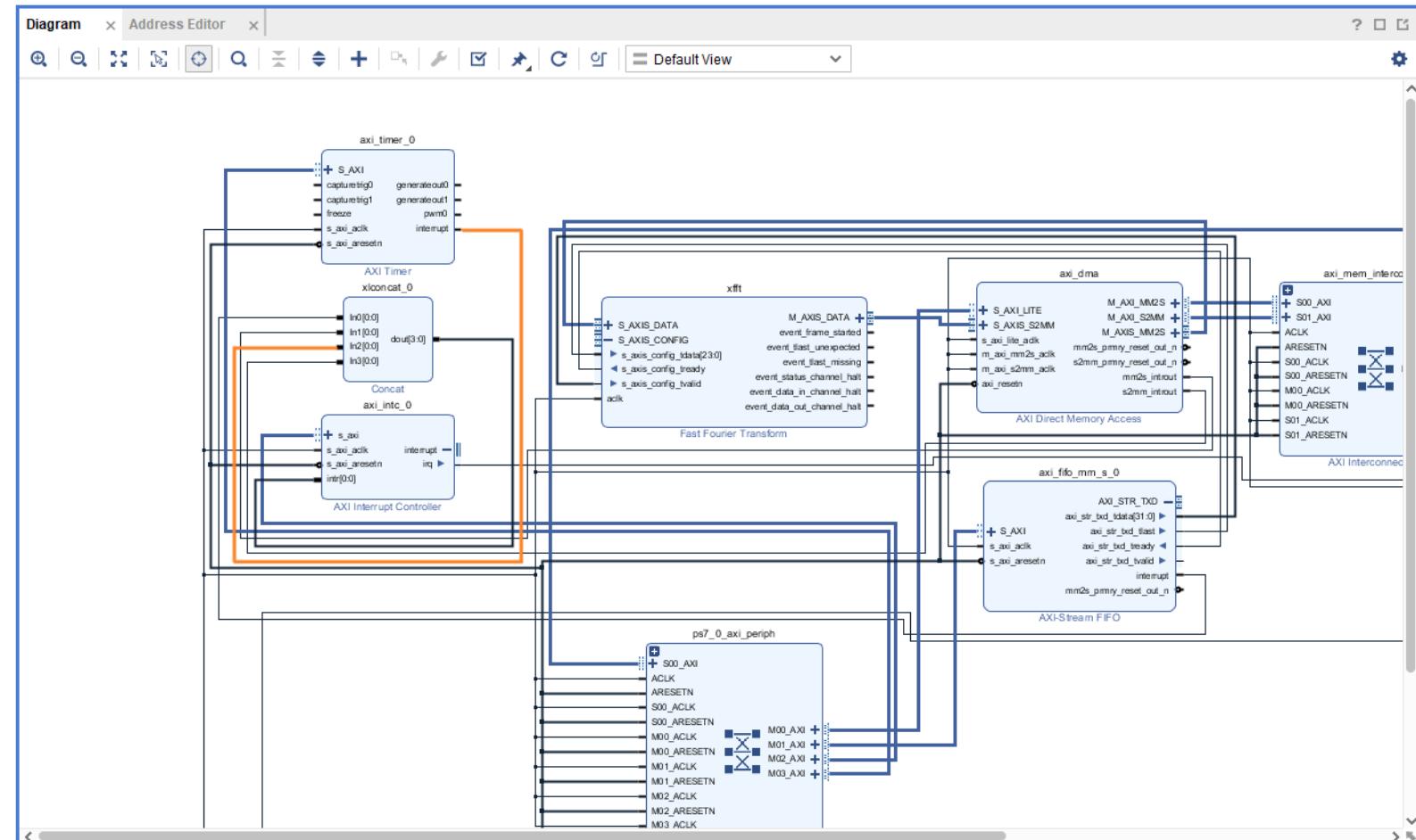
Run the connection automation



Lab : FFT Acceleration

Connect the interrupts to the concat block

- AXI Timer
- AXI DMA MM2S_INTOUT
- AXI DMA S2MM_INTOUT
- AXI Stream FIFO



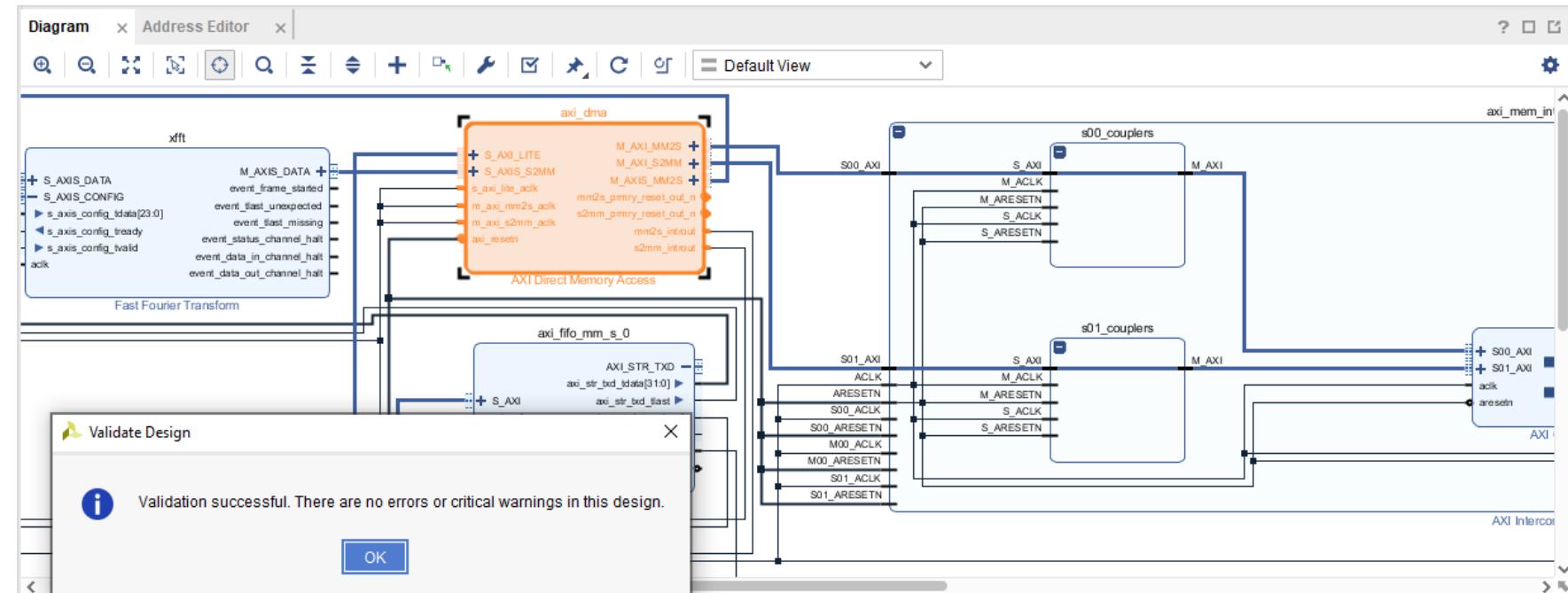
Lab : FFT Acceleration

Validate the design

there should be no

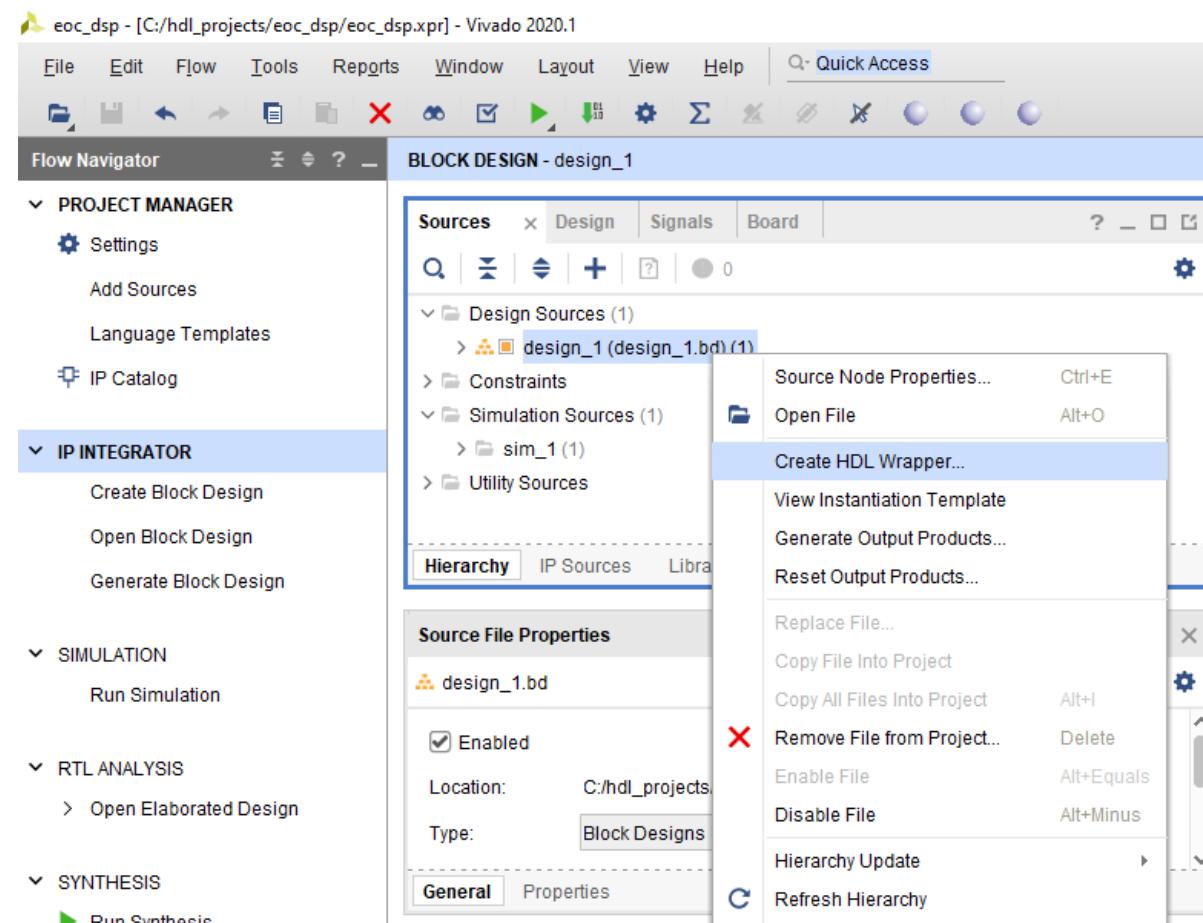
error or critical

warnings



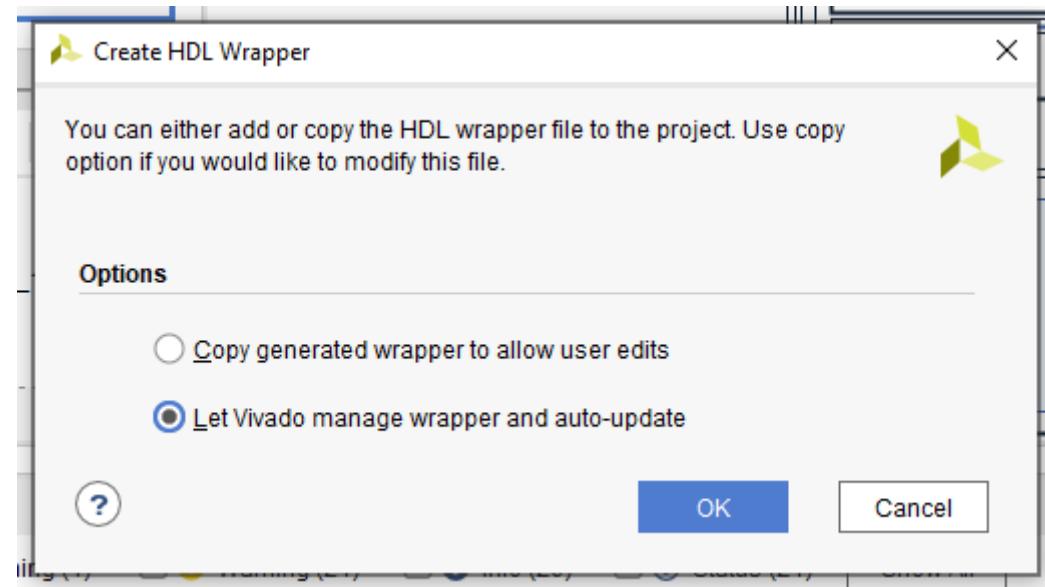
Lab : FFT Acceleration

Right click on the design and
select Create HDL Wrapper



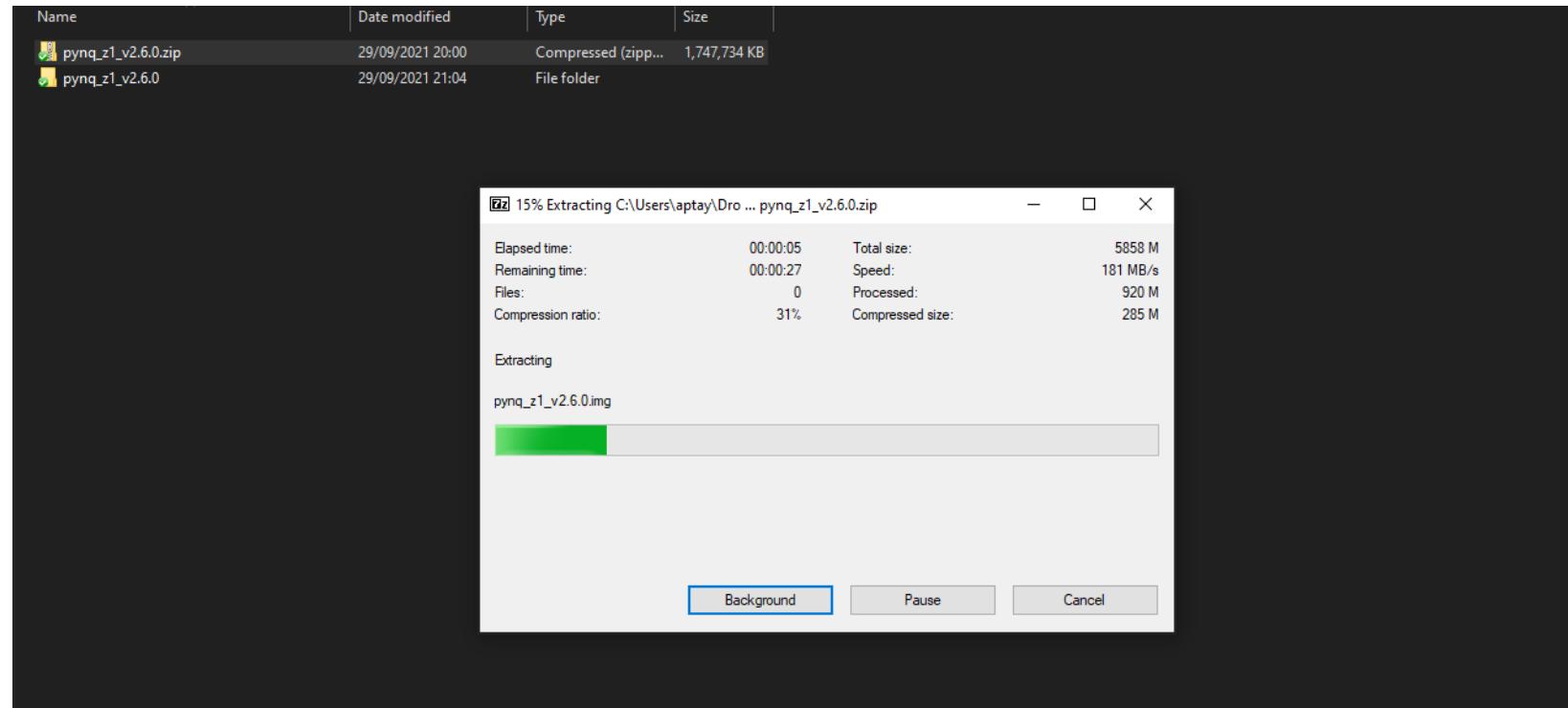
Lab : FFT Acceleration

Let Vivado manage the wrapper



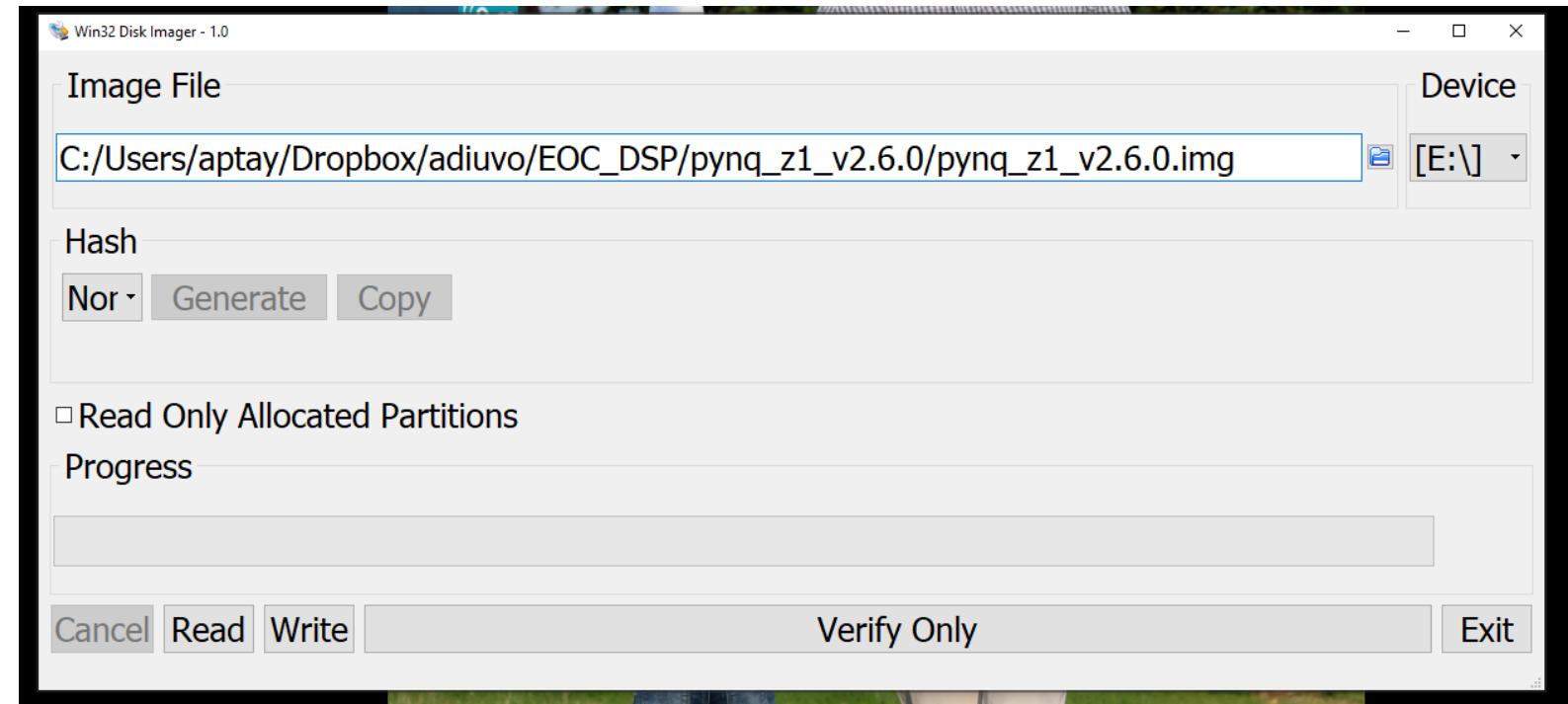
Lab : FFT Acceleration

Extract the downloaded PYNQ image



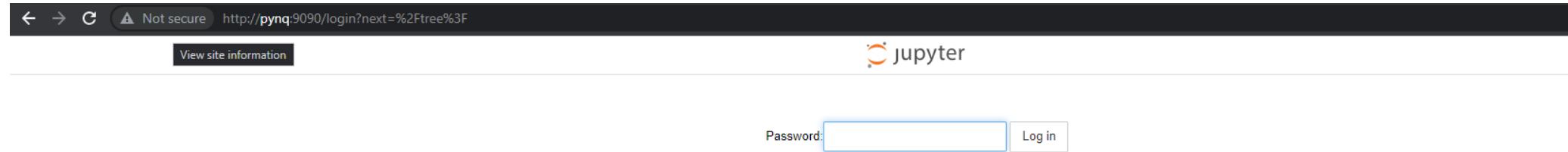
Lab : FFT Acceleration

Write the image to a SD card. Once completed insert the SD card in the Arty Z7-20. Connect a Ethernet cable and power via a USB cable



Lab : FFT Acceleration

Once the board boots, wait for the LEDs to flash. In a browser enter the address pynq:9090
when prompted enter the password xilinx



Lab : FFT Acceleration

Once logged in you should see the folder structure below



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter, Logout
- Menu Bar:** Files (selected), Running, Clusters, Nbextensions
- Toolbar:** Select items to perform actions on them, Upload, New, Refresh
- File List:**

| checkbox | Name | Last Modified |
|--------------------------|-----------------------|---------------|
| <input type="checkbox"/> | base | a year ago |
| <input type="checkbox"/> | common | a year ago |
| <input type="checkbox"/> | getting_started | a year ago |
| <input type="checkbox"/> | logictools | a year ago |
| <input type="checkbox"/> | Welcome to Pynq.ipynb | a year ago |

Lab : FFT Acceleration

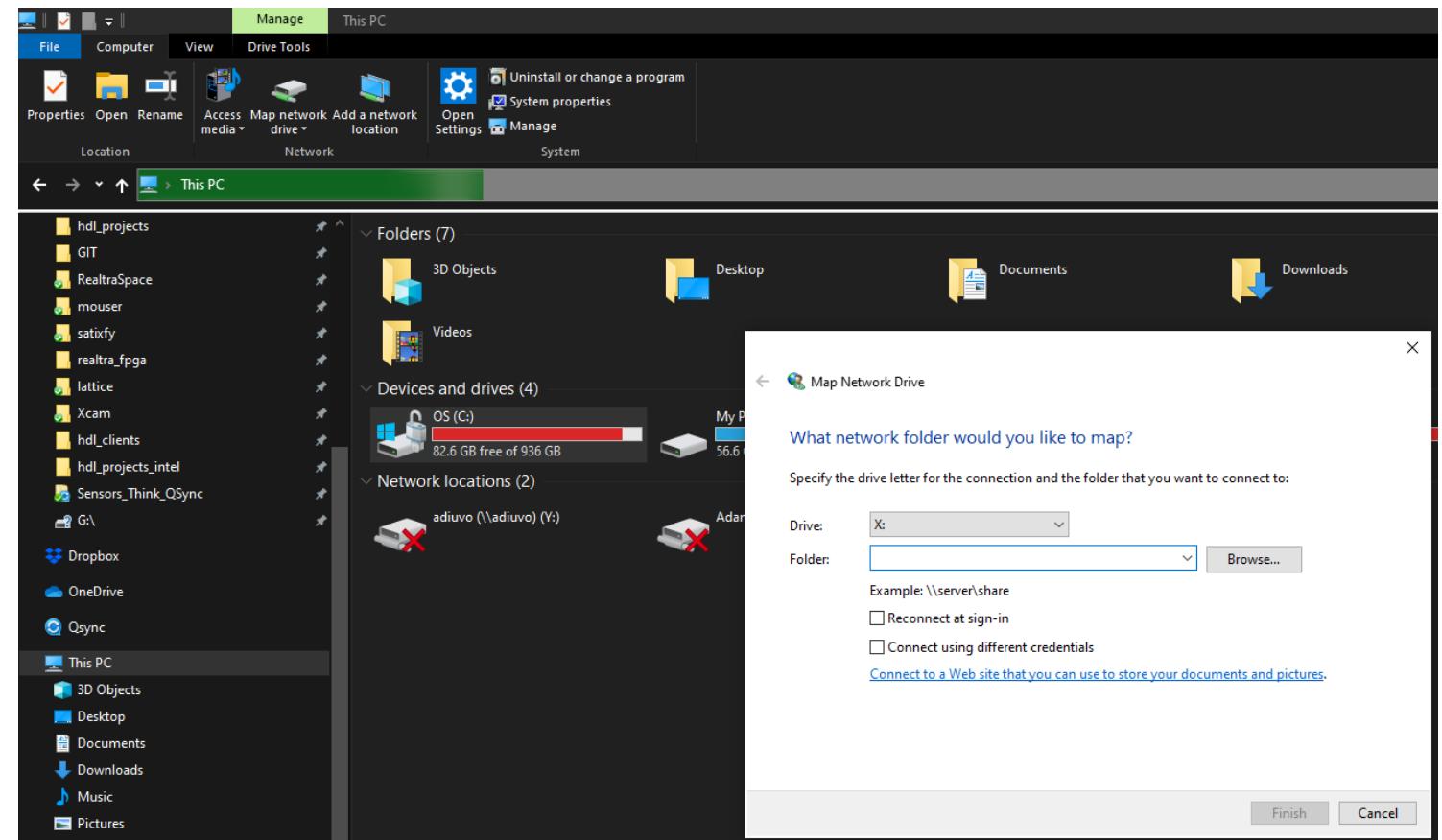
In a file explorer map a

network drive to

\pynq\xilinx

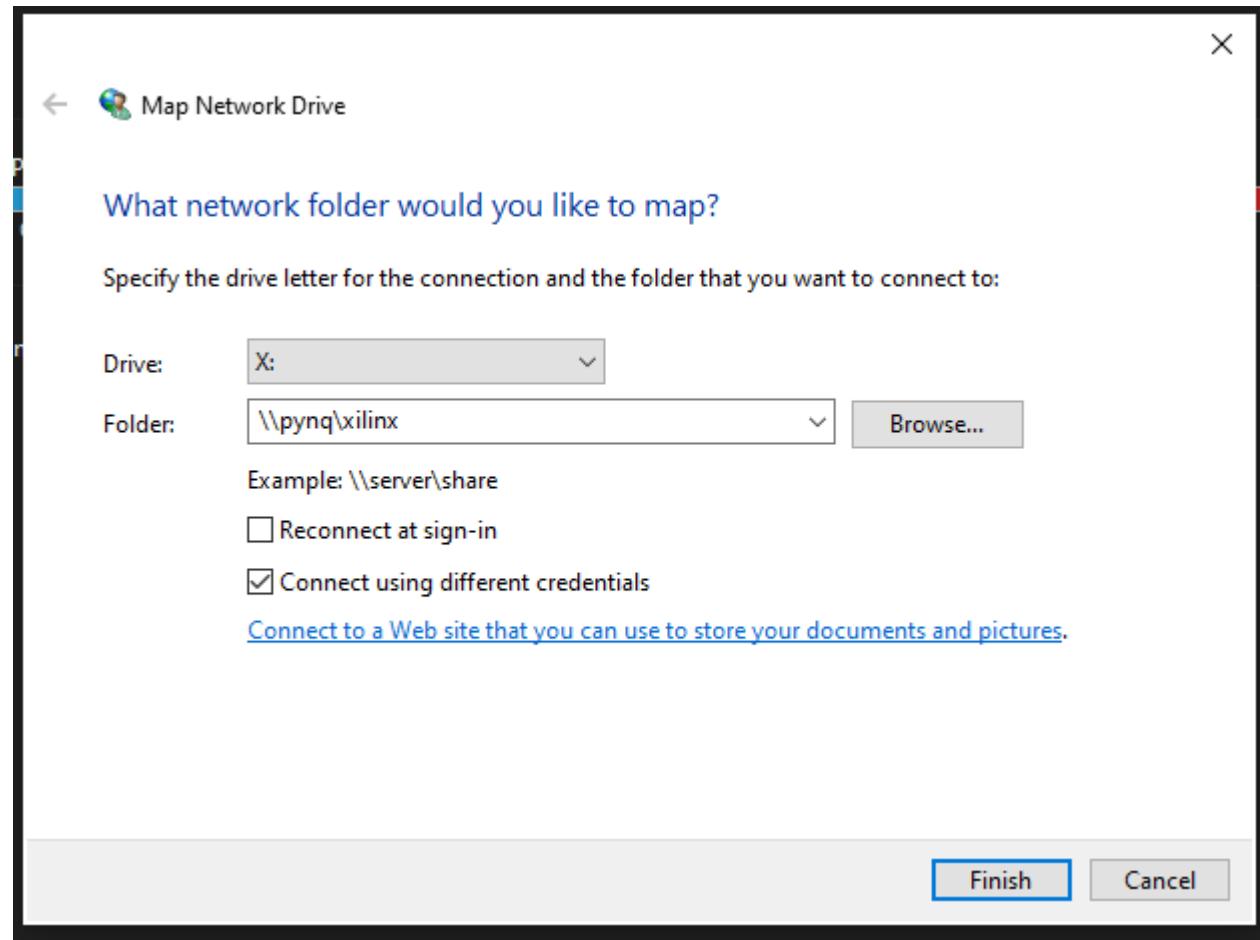
Select connect using different

credentials



Lab : FFT Acceleration

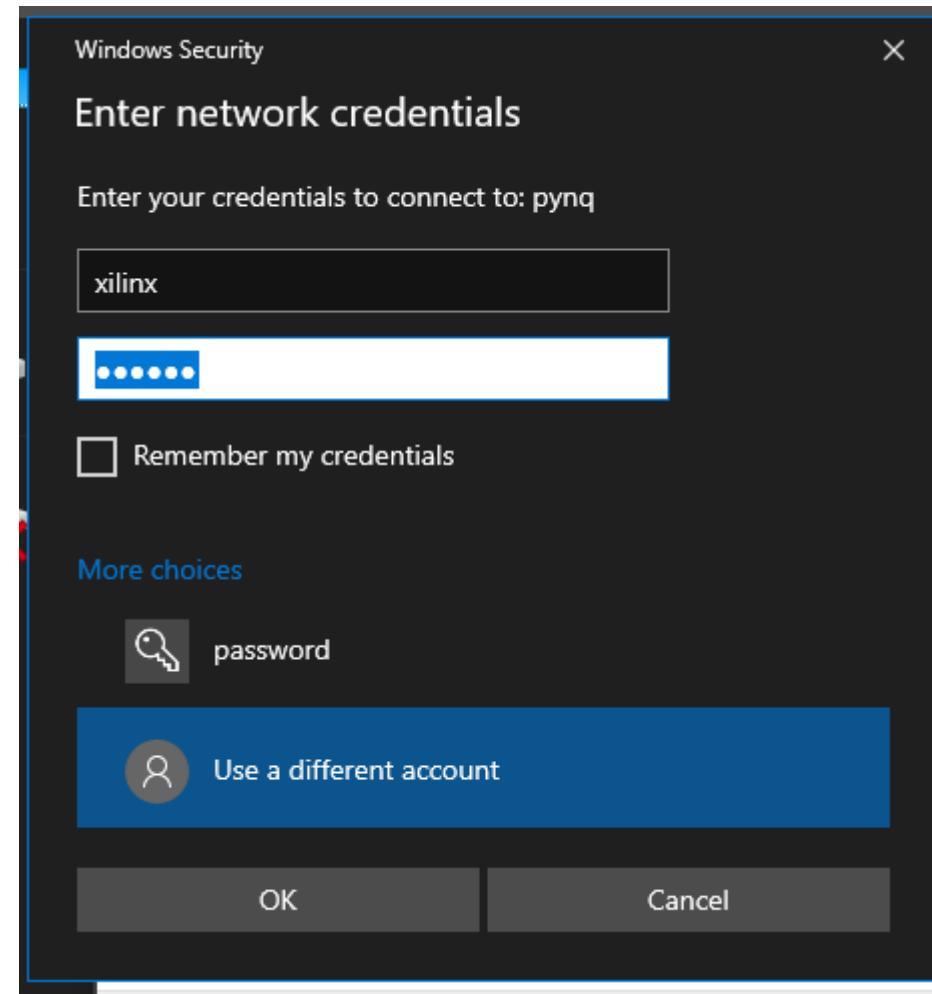
Completed Map drive, click OK



Lab : FFT Acceleration

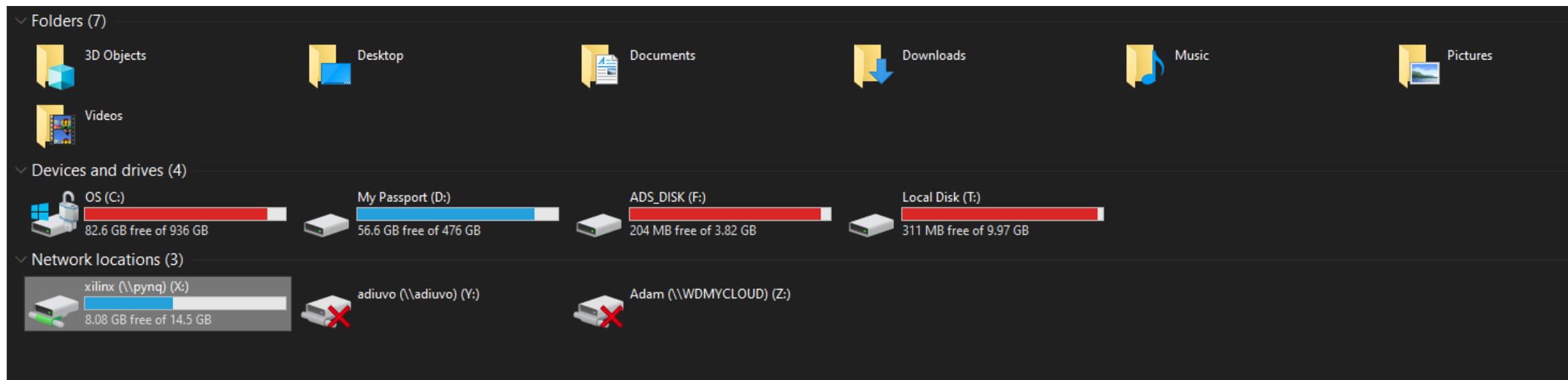
Enter the username and password as

Xilinx click OK



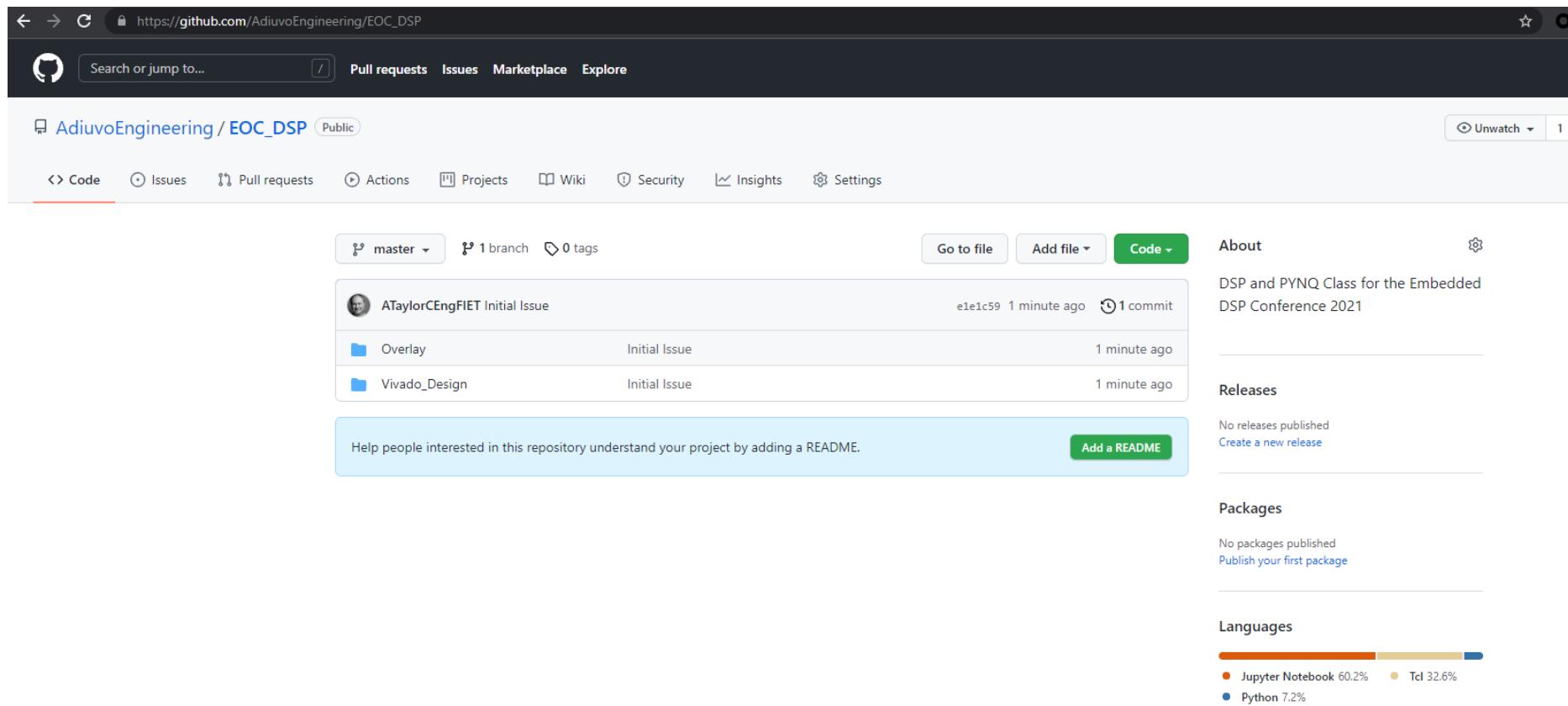
Lab : FFT Acceleration

The pynq drive should not appear as a samba server



Lab : FFT Acceleration

Clone the repository - https://github.com/AdiuvoEngineering/EOC_DSP



The screenshot shows the GitHub repository page for `AdiuvoEngineering/EOC_DSP`. The page has a dark theme. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below the header, the repository name is displayed along with its status as Public. There are tabs for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings, with Code being the active tab. In the main content area, there's a summary showing 1 branch and 0 tags. A list of issues is shown, with the first one being an initial issue from ATaylorCEngFIET. Below the issues, there's a call-to-action to add a README. To the right, sections for About, Releases, Packages, and Languages are present. The About section describes the repository as "DSP and PYNQ Class for the Embedded DSP Conference 2021". The Languages section shows a breakdown: Jupyter Notebook (60.2%), Tcl (32.6%), and Python (7.2%).

Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

master

1 branch

0 tags

ATaylorCEngFIET Initial Issue

e1e1c59 1 minute ago 1 commit

Overlay Initial Issue 1 minute ago

Vivado_Design Initial Issue 1 minute ago

Add a README

About

DSP and PYNQ Class for the Embedded DSP Conference 2021

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

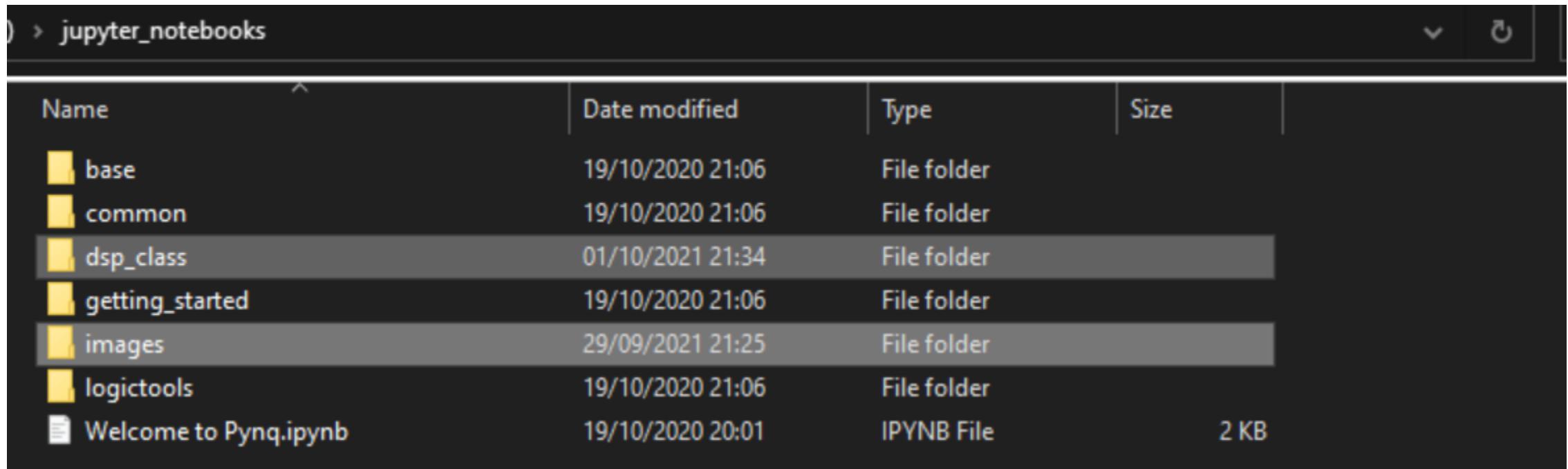
Languages

Jupyter Notebook 60.2% Tcl 32.6%

Python 7.2%

Lab : FFT Acceleration

From the Cloned Repo copy the directory Images and dsp_class to the Pynq boards Jupyter notebooks directory



| Name | Date modified | Type | Size |
|-----------------------|------------------|-------------|------|
| base | 19/10/2020 21:06 | File folder | |
| common | 19/10/2020 21:06 | File folder | |
| dsp_class | 01/10/2021 21:34 | File folder | |
| getting_started | 19/10/2020 21:06 | File folder | |
| images | 29/09/2021 21:25 | File folder | |
| logictools | 19/10/2020 21:06 | File folder | |
| Welcome to Pynq.ipynb | 19/10/2020 20:01 | IPYNB File | 2 KB |

Lab : FFT Acceleration

You should see a new directory in the PYNQ environment, select DSP_CLASS

jupyter Logout

Files Running Clusters Nbextensions

Select items to perform actions on them. Upload New ▼ 

| <input type="checkbox"/> |  0 |  |  | Name ▼ | Last Modified |
|--------------------------|---|---|---|---|---------------|
| <input type="checkbox"/> |  |  |  | base | a year ago |
| <input type="checkbox"/> |  |  |  | common | a year ago |
| <input type="checkbox"/> |  |  |  | dsp_class | 16 hours ago |
| <input type="checkbox"/> |  |  |  | getting_started | a year ago |
| <input type="checkbox"/> |  |  |  | images | 3 days ago |
| <input type="checkbox"/> |  |  |  | logictools | a year ago |
| <input type="checkbox"/> |  |  |  | Welcome to Pynq.ipynb | a year ago |

Lab : FFT Acceleration

Select fft.ipynb it will open and start running

jupyter Logout

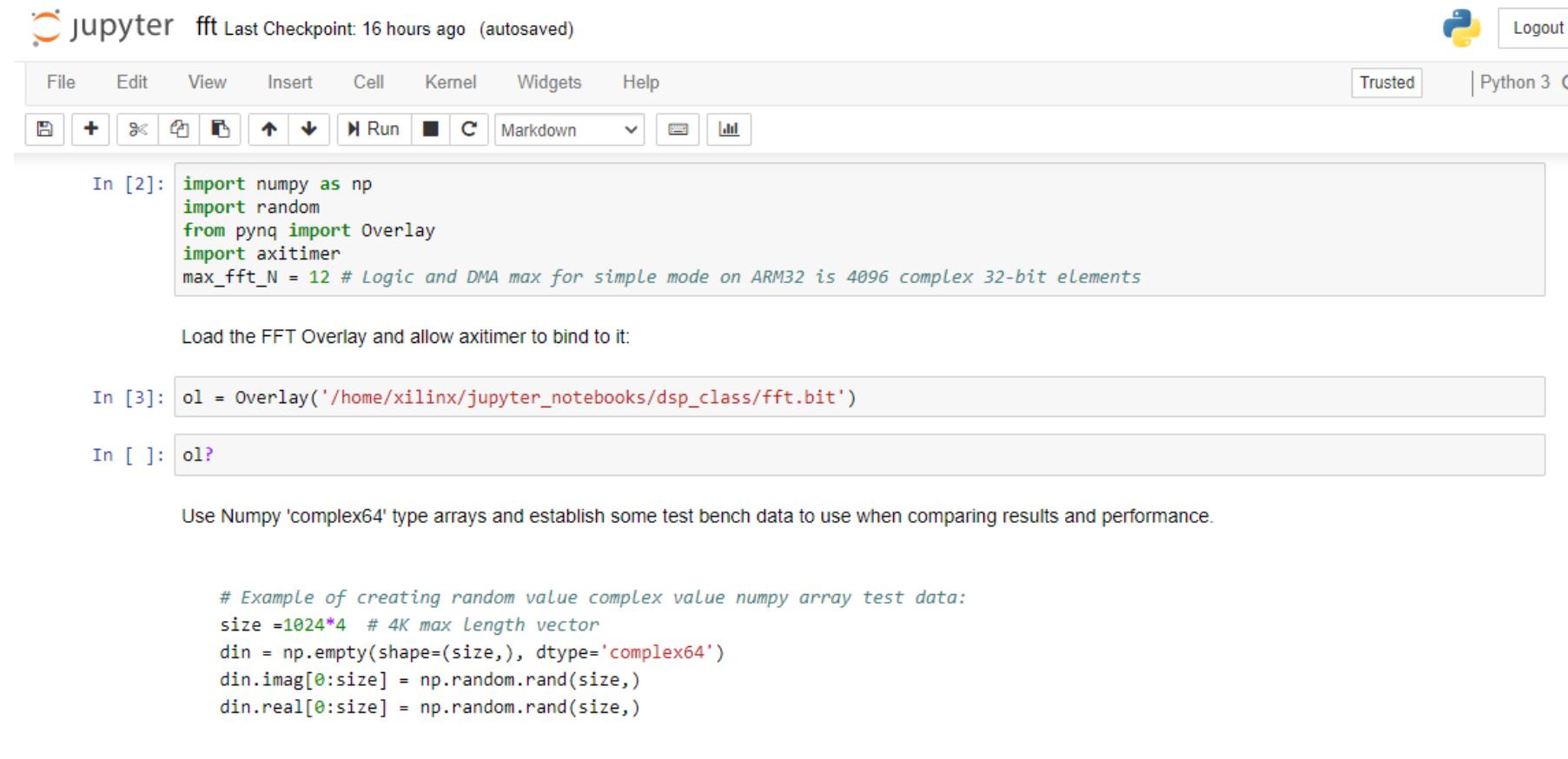
Files Running Clusters Nbextensions

Select items to perform actions on them. Upload New 

| <input type="checkbox"/> | Name | Last Modified |
|--------------------------|--------------------|----------------------|
| <input type="checkbox"/> | 0 / dsp_class | seconds ago |
| <input type="checkbox"/> | .. | |
| <input type="checkbox"/> | fft.ipynb | Running 14 hours ago |
| <input type="checkbox"/> | axidma.py | 7 months ago |
| <input type="checkbox"/> | axififo.py | 7 months ago |
| <input type="checkbox"/> | axitimer.py | 7 months ago |
| <input type="checkbox"/> | fft.bit | 15 hours ago |
| <input type="checkbox"/> | fft.hwh | 15 hours ago |
| <input type="checkbox"/> | sds_trace_data.dat | 3 days ago |

Lab : FFT Acceleration

Run each cell in turn in the notebook and notice the difference in performance between SW and HW Implementations

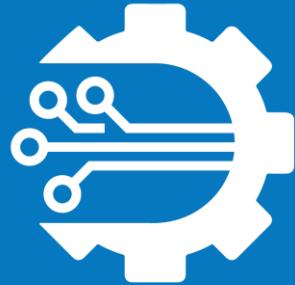


The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** jupyter fft Last Checkpoint: 16 hours ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, Python 3
- In [2]:** Python code to import numpy, random, and pynq Overlay, and define max_fft_N = 12. A comment indicates the max value for simple mode on ARM32 is 4096 complex 32-bit elements.
- In [3]:** Python code to load an Overlay from a bitfile.
- In []:** Python code to print the Overlay object.
- Text:** A note to use Numpy 'complex64' type arrays and establish some test bench data to use when comparing results and performance.
- Code:** Example code for creating random value complex value numpy array test data.



Questions?



ADIUVO
ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com



adam@adiuvoengineering.com