Department of Design Engineering and Mathematics

Middlesex University

Image signal processing on FPGA

Name: Shubham Haria

Supervisor: Zhijun Yang

PDE3400 Major project

B.Eng. Electronics engineering

May 2022

Table of Contents

1. Acknowledgement

First of all, I would like to thank Middlesex university, London to give me such a good opportunity to work Major project like this one and also providing all the support to complete the project.

I would also like to express my special thanks to both the supervises, Dr. Zhijun yang for guidance thought out the project. Also, for helping me out at every step of the project by having regular meetings regarding the project updates and making sure I can give my best for this project.

Along with that I would like to show gratitude to my second supervisor Inas El-Aroussi, for supporting me with constant feedbacks on my progress of the report.

I would also like to extent my gratitude to the module leader Dr. Eris Chinellato for assuring that I am focused on my project and keeping me motivated.

I would like to recognize the help from Adam and Mario for providing ample of knowledge for on the required topic that helped me to work my way out to finish the project.

Furthermore, I would like to thank Nick weldin, for making all the required components available for this project. along with that I would like to appreciate all the lectures, GAAs and all my classmate for their support they provided throughout the journey.

Finally, I would like to appreciate the moral support provided by my family members and friends.

2. Abstract

Considering the growth and upcoming demands of computer vision applications, it is evident that extraction of data from images and videos is going to become a very crucial aspect of the technology. This would need the work to be done as fast as possible and in the lowest possible cost so that the implementation can become viable. This being the moto of the project, main focus is put on the relevant software and hardware like PYNQ pipeline and OpenCV.

So, the relevant understanding of PYNQ base pipeline and Open CV is going to be the the first priority. And moving forward, doing enhanced modifications to them by merging and adding useful features will be the intent. Technically, direct memory access would be provided to the software which would reduce the time consumption and parallel operation of the data would add to this. The single software would carry out the complete process in no time and exclude the need of human intervention through hardware. Decreasing the complexity of the whole system, much of energy would eventually be saved and thus, the

whole thing would become cost efficient. In a nutshell, making the whole process easier as well as cheaper to build and carry out.

3. Introduction

Image processing plays vital role in the advancement of artificial intelligence. The ability to detect the boundaries of any images help in knowing a lot of features present in the image which has help artificial intelligence to collect the raw data. So initially, the intent of the project is to study and understand the working process of the PYNQ design flow for hardware and software alongside analysing and implementation of basic image processing libraries like OpenCV. Also, look for compatible pipelines possessing different types of features which could be merged with the PYNQ base pipeline to enhance and improvise it's working. This might require modifying or rebuilding of the same to get most desirable features.

Over the previous decade, computer vision applications have become quite significant with steady increase in numbers and diverse areas of application being developed. So, Image processing needs to be conducted in numerous sectors like robotics, autonomous cars, medical systems, etc. for getting useful results. For instance, if we want to ensure proper working of the autonomous car, it needs to recognise the lanes in front of it at a good pace if it must run smoothly without colliding. The image processing units being used currently though are cost efficient but lag in delivering sufficient computing power, especially for high resolution images. Now, when the SOCs are implemented, the FPGA part gets a significant boost which is because of the hardware accelerators getting in parallel and calculation getting faster. This provides good acceleration to the computation process and decreases the load on main CPU to a good extent. On top, the use of SOCs help in energy conservation which is achieved through specialized hardware. It was not only these but python as well which gained measurable popularity in the last few years. Being simple and capable of high abstraction, fast paced application development becomes the important use of it. It can be conveniently said that python is the tool for programming in the application without showing any change to the user in hardware. All these things look perfectly viable as well as implementable for the objective of the project.

Project overview:

This project begins with the setting up the FPGA board with web cam and display, it also needs to be flash the image on FPGA and connect it with system. After that the base hardware for that FPGA was rebuilt in the vivado design IDE, which was followed by the add new IPs from different open-sourced projects. The open-sourced project supported image processing acceleration. And this new IPs were built in a new subsystem which was later connected to the base pipeline. And then the new bitstream was generated and uploaded to FPGA board. That was later tested with other examples.

4. Literature review

The following literature review explores the current pipeline which has been used in image processing on Pynq soft core microprocessor. As computer version plays a major role in artificial intelligence, it can be said that this allows systems to extract information from images or videos. The increasing use of IOT in modern world is creating demand for faster and economical image processing at larger scale. [1]

3.1 base pipeline

Diverse types of image processing can be performed on the general pynq pipeline(base.bit) which is already included in the image. The pipeline supports most of peripheral parts that Pynq supports like Arduino, raspberry Pi alongside all other pmods and HDMI ports. The pipeline consist of 5 major components as follows,

- Frontend (HDMI): - in this stage is used to establish a connection between HDMI Ports and the Processor.
- Frontend (Others): - in this stage is used to setup all other elements (timing logic, clock and etc.) for the process.
- ColorSpace converter: - in this stage, the pixels are transformed with the help of matrix.
- Pixel format converter: - in this stage the pixel format is converted from 24 bit to required bits for processor.
- VDMA: - in this stage the video frames are transferred to and from memory.

Pynq and get familiar with the all the peripheral components. Along with that the Pynq interface works on python as program system which enables the use of openCV library for the image processing.

3.2 CV2pynq

On the PYNQ platform, this project accelerates OpenCV functionalities. A number of common image filters and feature detection methods are implemented in the library. The ZYNQ chip's Programmable Logic (PL) manages the calculation of time-consuming activities. Direct Memory Access is used to transport all data, which is then broadcasted back to the DRAM, and therefore to the Python program. The Video-Subsystem of the PYNQ based project is also included in cv2PYNQ. As a result, you can use the HDMI In or USB webcam and HDMI Out interfaces in your program. If the input and output buffers are in the chip's contiguous memory, the library calculates every filter for gray-channel images with 1080p in 16 milliseconds. [2]

The description provided valuable info about the hardware and software implemented for image processing in CV2 in detail.

Conclusion: - The main intent of pacifying and economizing image processing has finally been achieved by mounting the base pipeline with CV2PYNQ. Hence, complete system of hardware and software can be supported and operated with least number of complications. This would also make it possible to do all the work using python alone without needing any external agency. Therefore, it can be said that the work Is done in milliseconds despite the use of filters alongside sustaining the quality of the same.

5. Material required

Hardware

- PYNQ-Z2 FPGA board
- Micro SD card greater than equal to 16 GB
- Micro SD card adapter
- Micro USB cable × 2
- Ethernet cable
- Web camera
- HDMI cable
- HDMI Display

Software

- Vivado design suite
- 7-Zip
- Etcher*
- Pynq-Z2 board definition files
- WinSCP

GitHub

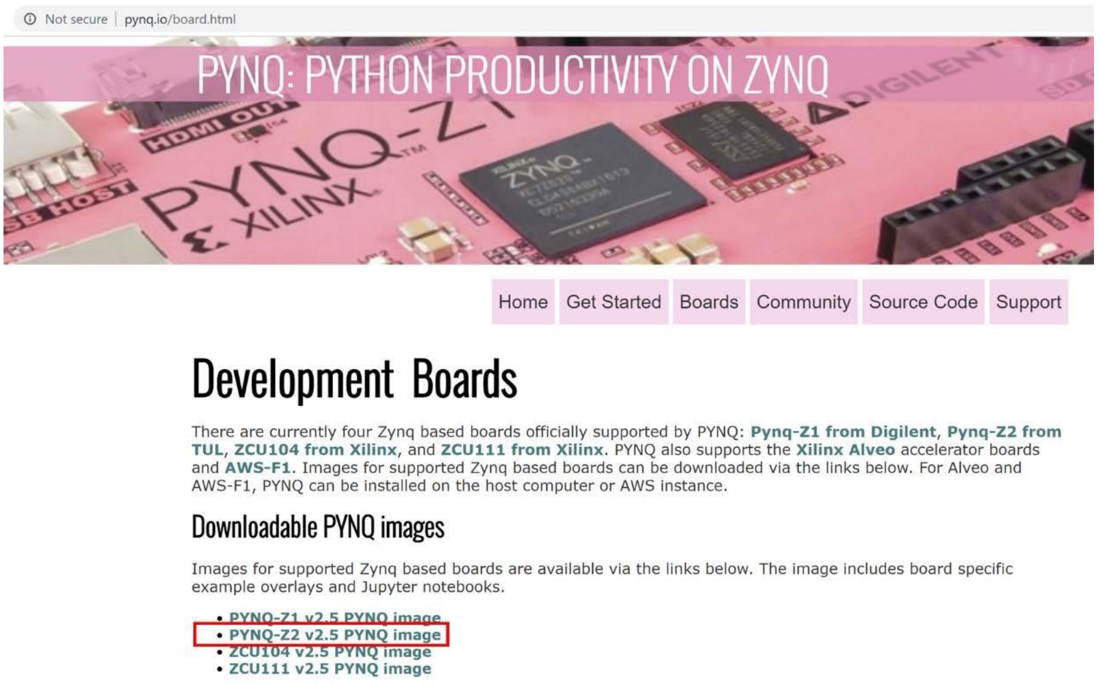
- Pynq: - https://github.com/Xilinx/PYNQ/tree/image_v2.6.0
- Cv2PYNQ: - https://github.com/wbrueckner/cv2pynq

6. Project development

6.1. Testing pynq example

This phase was carried out to get familiar with the PYNQ design process and how setup the hardware and required software to work with PYNQ. This was done by following the lab given by element 14. It was Instructed by Adam Taylor. [3]

- Install the webpack version of Vivado Design Suite 2020.1.
- Add PYNQ board files to vivado library.


Not secure | pynq.io/board.html

Development Boards

There are currently four Zynq based boards officially supported by PYNQ: Pynq-Z1 from Digilent, Pynq-Z2 from TUL, ZCU104 from Xilinx, and ZCU111 from Xilinx. PYNQ also supports the Xilinx Alveo accelerator boards and AWS-F1. Images for supported Zynq based boards can be downloaded via the links below. For Alveo and AWS-F1, PYNQ can be installed on the host computer or AWS instance.

Downloadable PYNQ images

Images for supported Zynq based boards are available via the links below. The image includes board specific example overlays and Jupyter notebooks.

- PYNQ-Z1 v2.5 PYNQ image
- PYNQ-Z2 v2.5 PYNQ image
- ZCU104 v2.5 PYNQ image
- ZCU111 v2.5 PYNQ image

Figure 1Pynq-Z2 Image

- Download and unzip the PYNQ image for ZYNQ Z2 board and write the image on SD card.
- Make all the connections as follows
  - Connect the web camera to Fpga board.
  - Connect the HDMI cable to FPGA board and the display.
  - Connect the usb cable from laptop to the display to power up the display.
  - Connect Ethernet and usb cable from laptop to FPGA board.

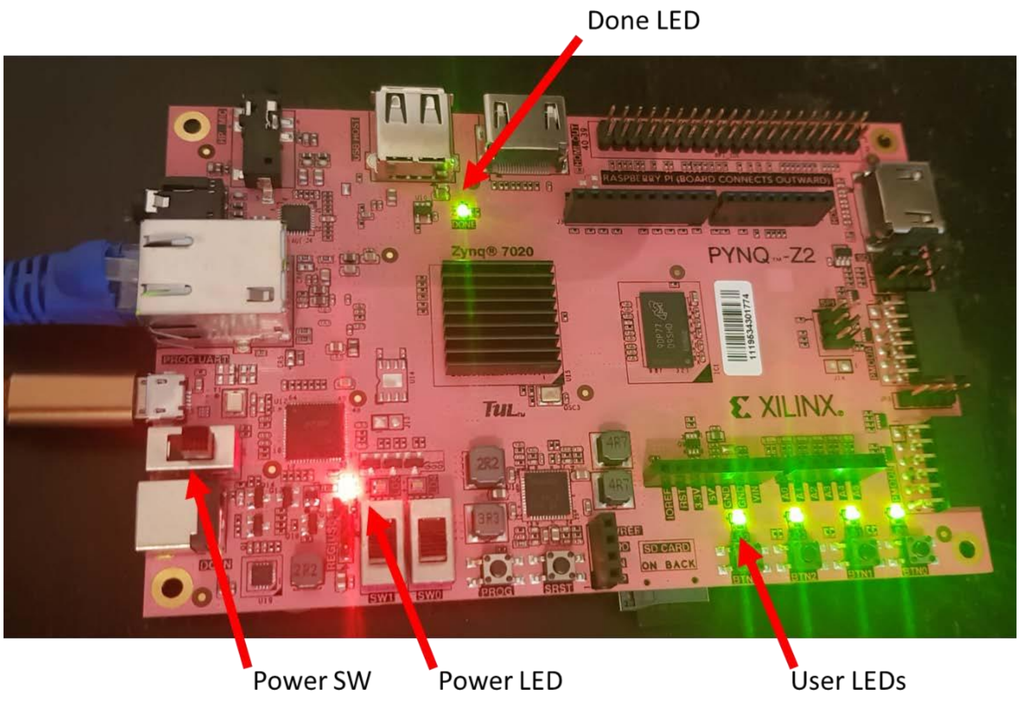
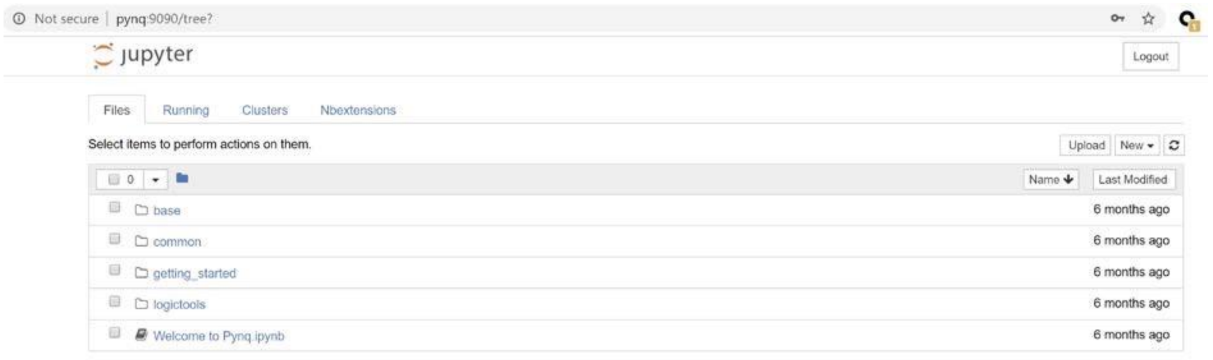


Figure 2 FPGA board after booting up

- Launch pynq 9090 on web browser



Select items to perform actions on them.

Name	Last Modified
base	6 months ago
common	6 months ago
getting_started	6 months ago
logictools	6 months ago
Welcome to Pynq.ipynb	6 months ago

Figure 3Jupiter notebook

6 and test any of the examples

6.2. Base Pipeline Introduction

The base overlay design allows PYNQ to use peripherals on a board right out of the box. Hardware IP is used to control peripherals on the target board, and these IP blocks are connected to the Zynq PS. Peripherals can be used from the Python environment immediately after the system boots if a base overlay is available for the board. [4]

The board includes following hardware:

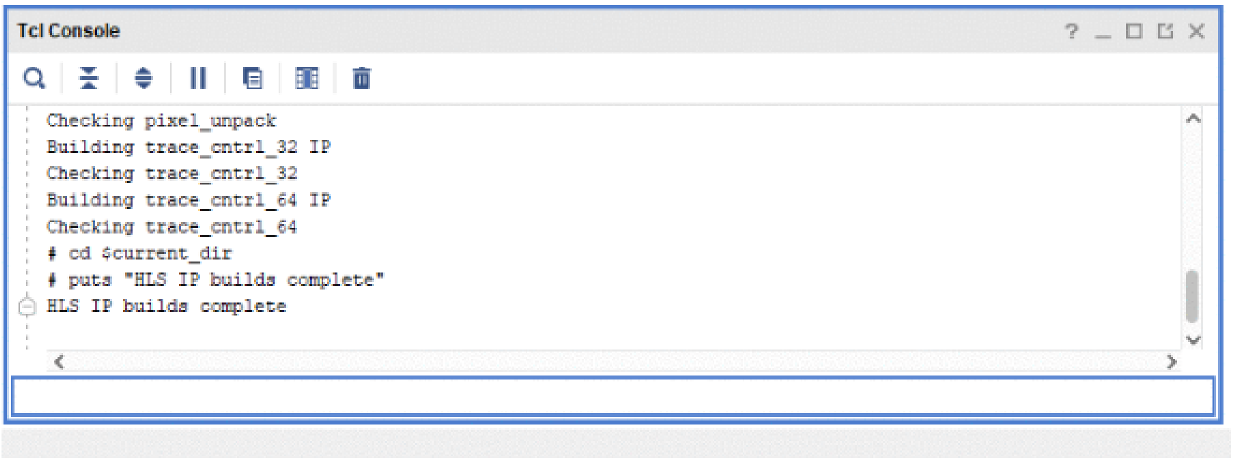
- HDMI (Input and Output)
- Microphone in
- Audio out
- User LEDs, Switches, Pushbuttons
- 2x Pmod PYNQ MicroBlaze
- Arduino PYNQ MicroBlaze
- 3x Trace Analyzer (PMODA, PMODB, ARDUINO)

6.3. Rebuilding Base Pipeline

In this phase the base.bit was reproduced so that in the later stage it can be updated. For this the tutorial by Cathal was referred [5]. For this the Pynq repository from was cloned from GitHub.

- To begin with the Pynq repository with 2.6 version was cloned.
- Open the vivado 2020.1.
- In tcl script change directory with cd to
  - C:/Users/Home/Documents/GitHub/PYNQ/boards/Pynq-Z2/base

- After that run the command to build the IPs required. This is done by the file named build_ip.tcl. the command used to do this is: - source ./build_ip.tcl



```
Checking pixel_unpack
Building trace_cntrl_32 IP
Checking trace_cntrl_32
Building trace_cntrl_64 IP
Checking trace_cntrl_64
# cd $current_dir
# puts "HLS IP builds complete"
HLS IP builds complete
```

Figure 4building IPs

- After building required IPs the base can be rebuilt with the help of base.tcl file. Just run source ./base.tcl. This will regenerate the base overlay in new project.

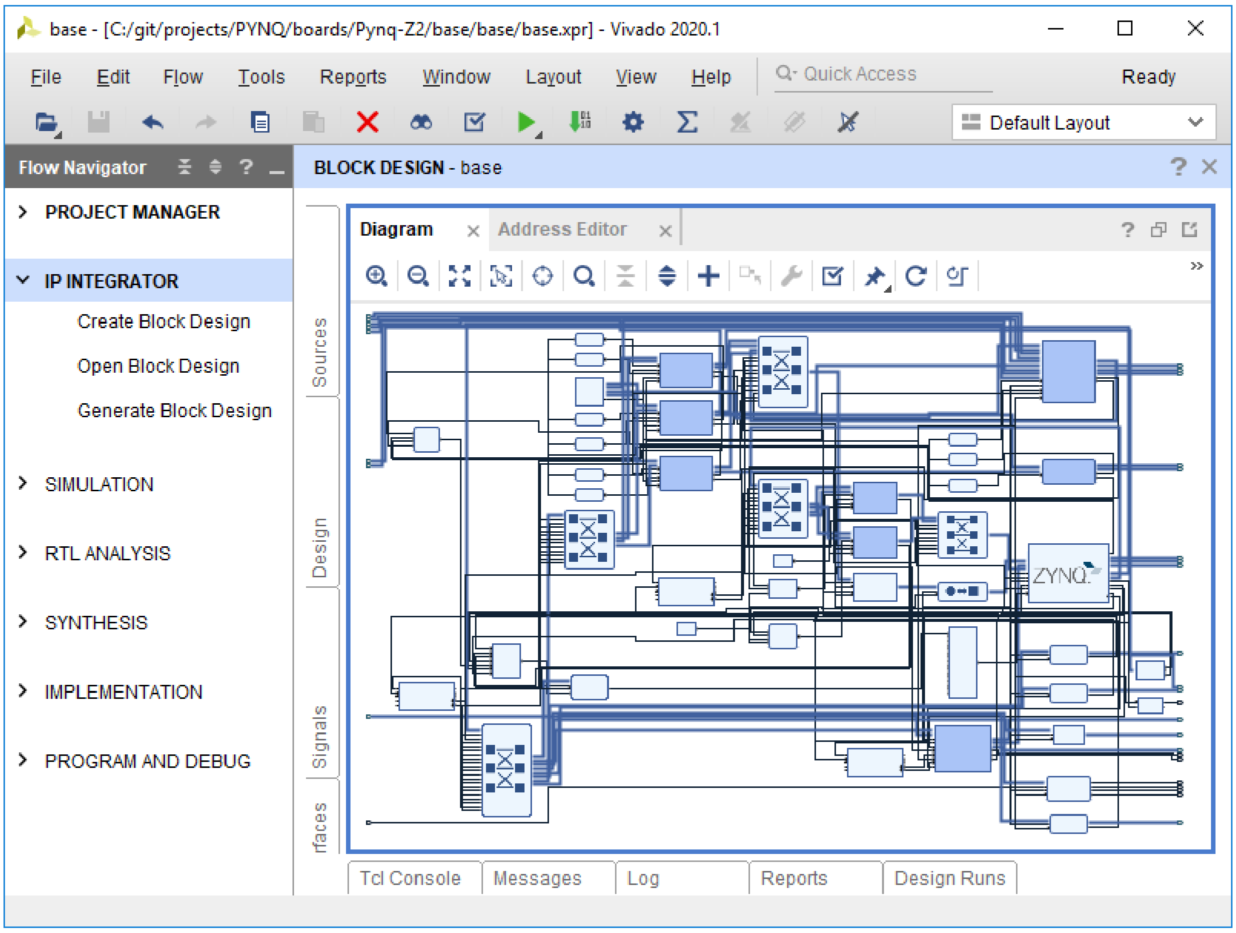


Figure 5Base Pipeline

The image above shows that the rebuilding of the Pipeline was successful.

6.4. Developing image acceleration sub system

In this step a hierarchy name image filter was created inside the base pipeline. This hierarchy contained the acceleration image filters from cv2Pynq library. For this example, only 3 IPs were used canny, filter2d and erode for the simplicity of the project. But can be altered in a simple way. Initial the library is cloned, and then the hierarchy was developed from the reference doc image given in the library.

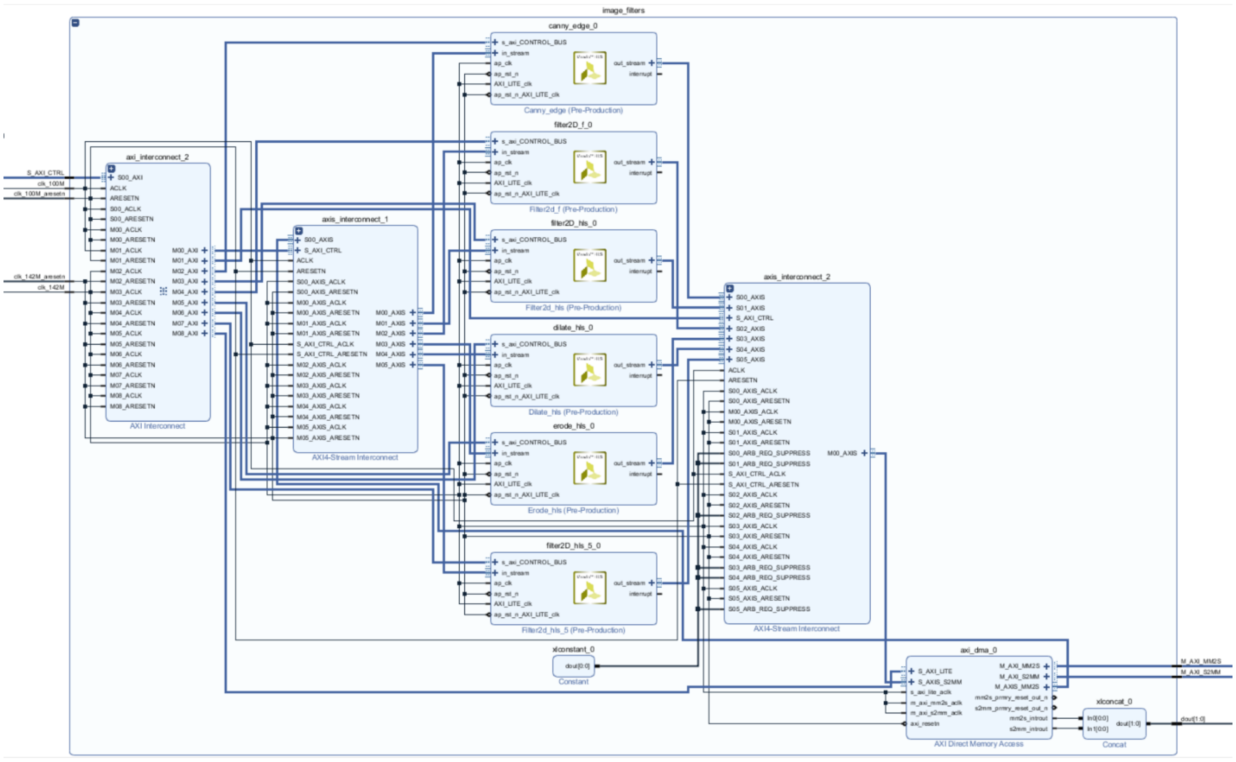
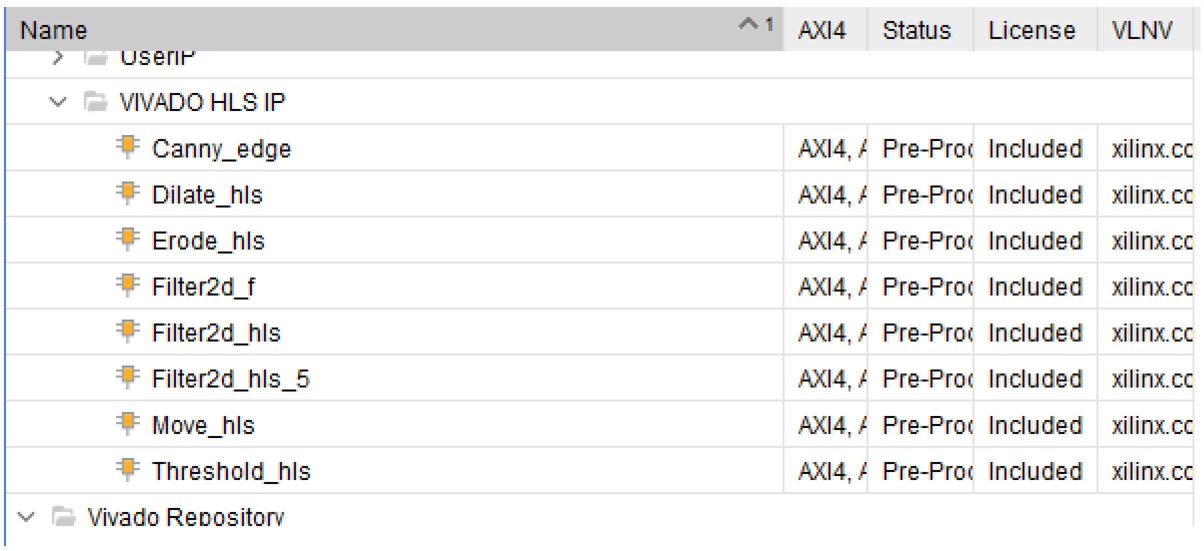


Figure 6Acceleration pipeline of CV2PYNQ library

- To begin with clone cv2pynq-the project behind the library from GitHub
    - https://github.com/wbrueckner/cv2PYNQ-The-project-behind-the-library.
- Add IP repository as learned in Adams labs.

- 

Name	AXI4	Status	License	VLNV
› UserIP				
⌄ VIVADO HLS IP				
Canny_edge	AXI4, /	Pre-Proc	Included	xilinx.cc
Dilate_hls	AXI4, /	Pre-Proc	Included	xilinx.cc
Erode_hls	AXI4, /	Pre-Proc	Included	xilinx.cc
Filter2d_f	AXI4, /	Pre-Proc	Included	xilinx.cc
Filter2d_hls	AXI4, /	Pre-Proc	Included	xilinx.cc
Filter2d_hls_5	AXI4, /	Pre-Proc	Included	xilinx.cc
Move_hls	AXI4, /	Pre-Proc	Included	xilinx.cc
Threshold_hls	AXI4, /	Pre-Proc	Included	xilinx.cc
⌄ Vivado Repository				

Figure 7List of filter offered by CV2pynq library

- Now create hierarchy inside the base design (the design that was generated in) by right click, and name it as image filter.
- In this hierarchy add the following IPs
    - AXI interconnect
    - AXI4 stream interconnect ×2
    - Image filter as required from the figure
    - AXI direct memory access
    - Concat
    - Constant
- Change the number of slaves and master in the AXI interconnect as follows

IP name	Number of slaves	Number of Master	Using control register routing
AXI interconnect	1	3 + 3(number of Image filter)	no
AXI stream interconnect (input)	1	3(number of Image filter)	yes
AXI stream interconnect (output)	3(number of Image filter)	1	yes

Figure AXI interconnect organiser

- After the add following Pins by right click and select create pin
    - S0_AXI
    - Clk_100M
    - Clk_100M_areset
    - M_AXI_MM2S_0
    - M_AXI_S2MM_0
    - Dout_0
- Make all the required connections as shown in the figure bellow.

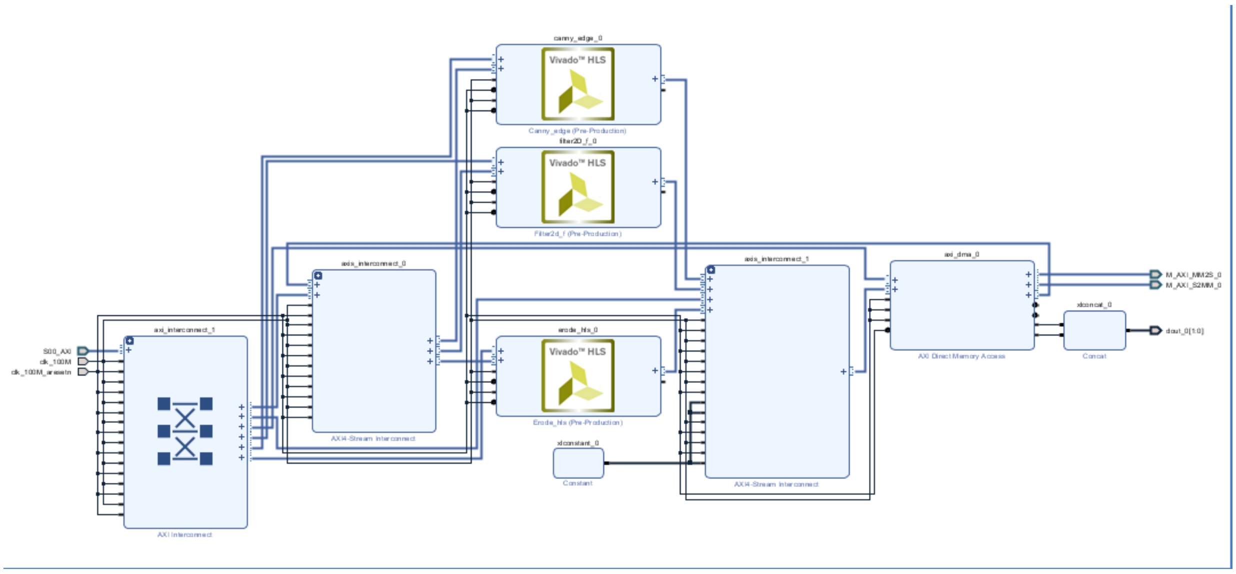


Figure 8Pynq Sub-system

After that validate design by pressing F6. To make sure that the design is correct.

6.5. Connecting the Subsystem to base

In this step the hierarchy will be connected to the main pipeline this is done with the help of the reference document

- To connect S00_AXI a master is required to be added to ps7_0_axi_periph_1.
- Along with that connect the required aclk and areset from fclk 3.

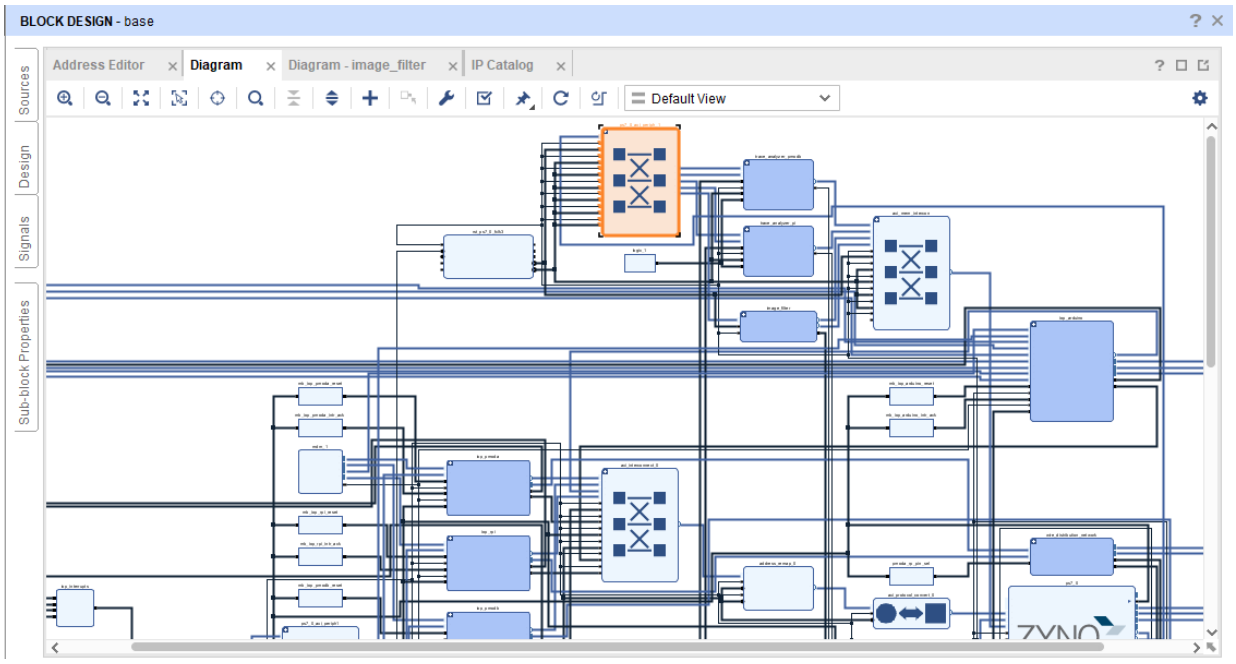


Figure 9AXI interconnect for image filter

- To connect the outputs of the image filter 2 slaves are added to axi_mem_intercon.

  Along with that connect the aclk 3 and 4 to fclk3(100M) and areset 3 and 4 to fclk areset.

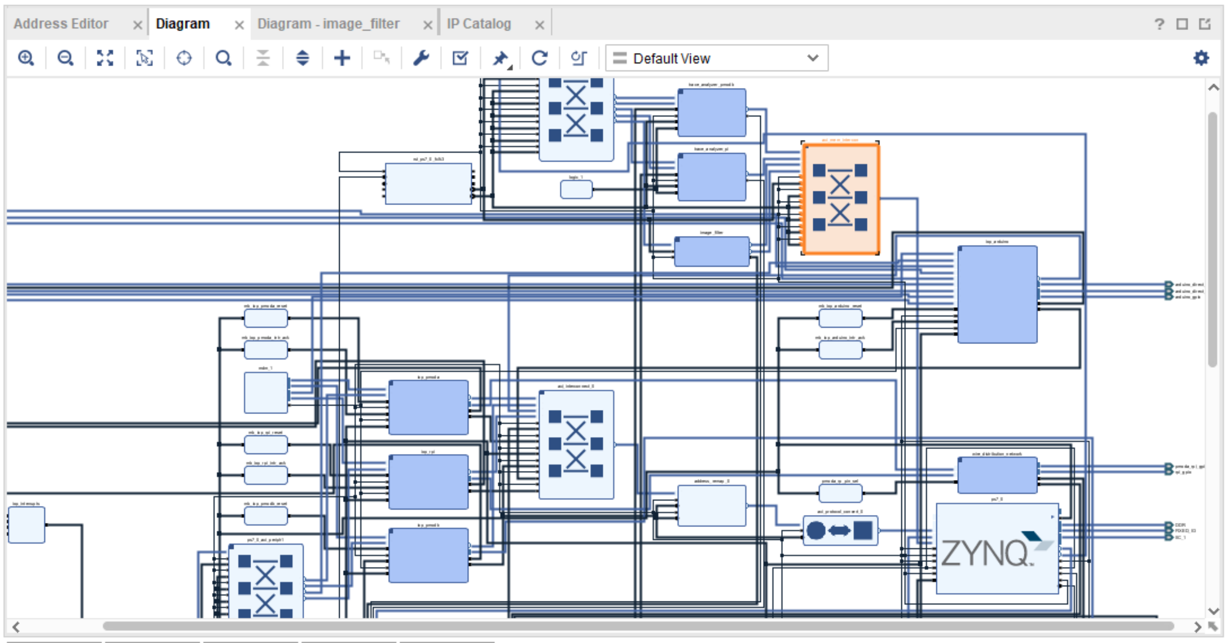


Figure 10AXI interconnect from output of Image filter

- Add a port to concat_interrupts (total of 8 ports).

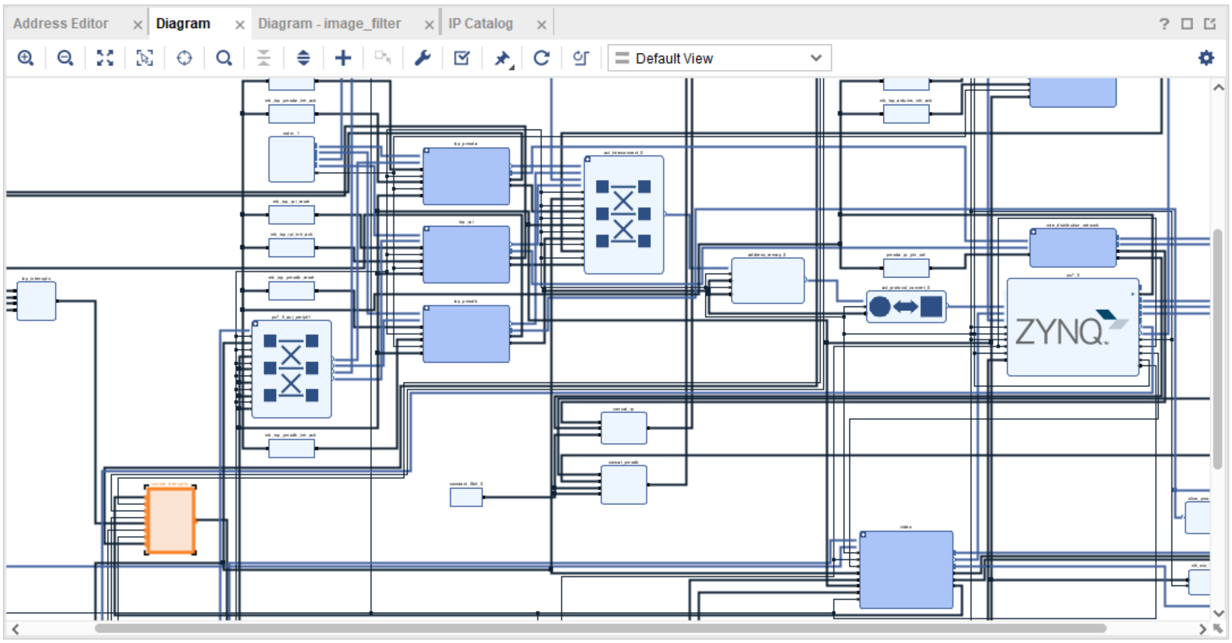


Figure 11Concat interrupt

- Make all the connections as given in the table for image filter sub block.

Port name on image filter sub system	Port name on pipeline (IP name)
S00_AXI	M04_AXI (ps7_0_axi_periph_1)
Clk_100M	Clk3
Clk_100M_aresetn	Clk3_interconnect_areset

M_AXI_MM2S_0	S02_AXI (axi_mem_intercon)
M_AXI_S2MM_0	S03_AXI (axi_mem_intercon)
Dout_0	In7(concat_interrupts)

Figure 12 Image filter port map

- After making all the connection the final pipeline would look like the figure 13.

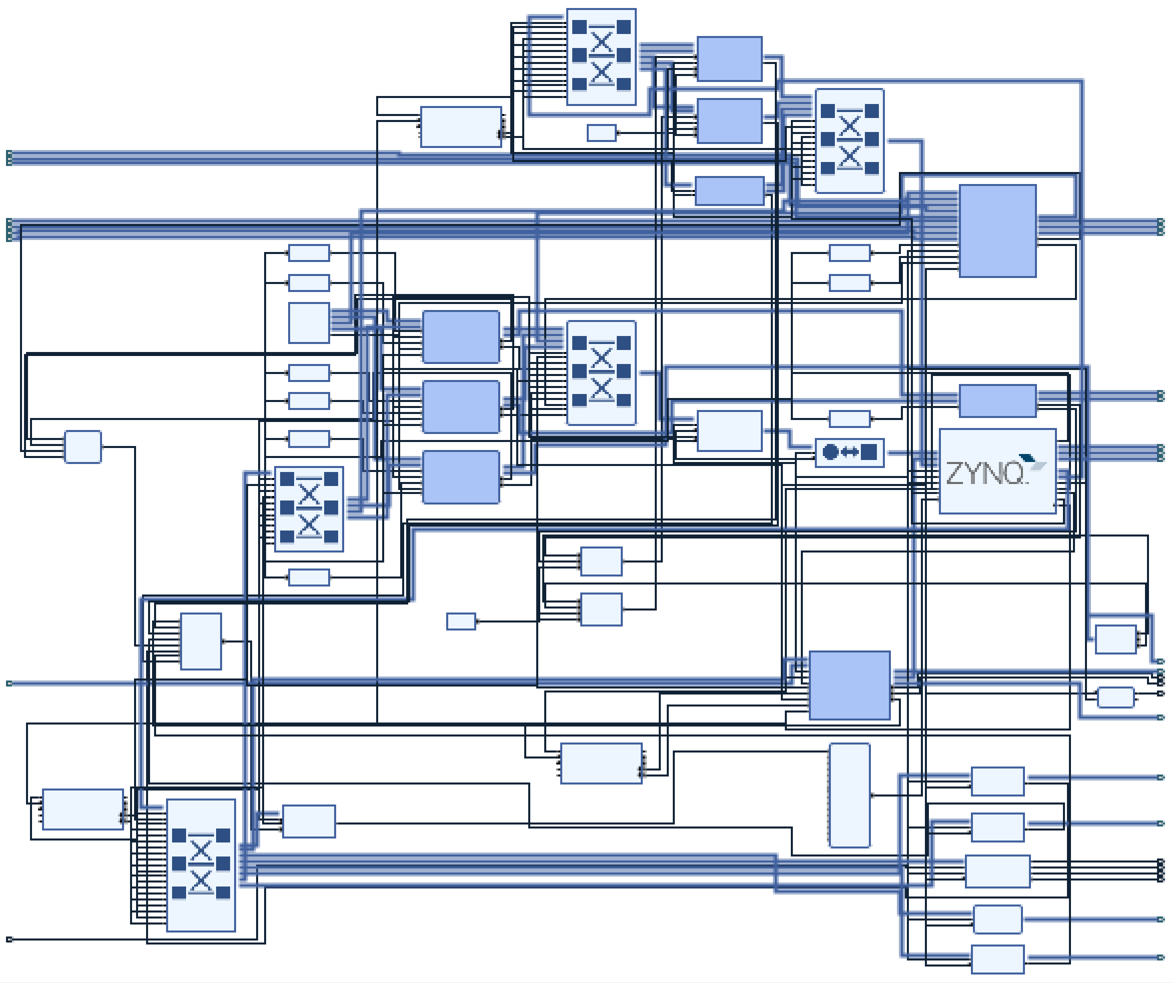
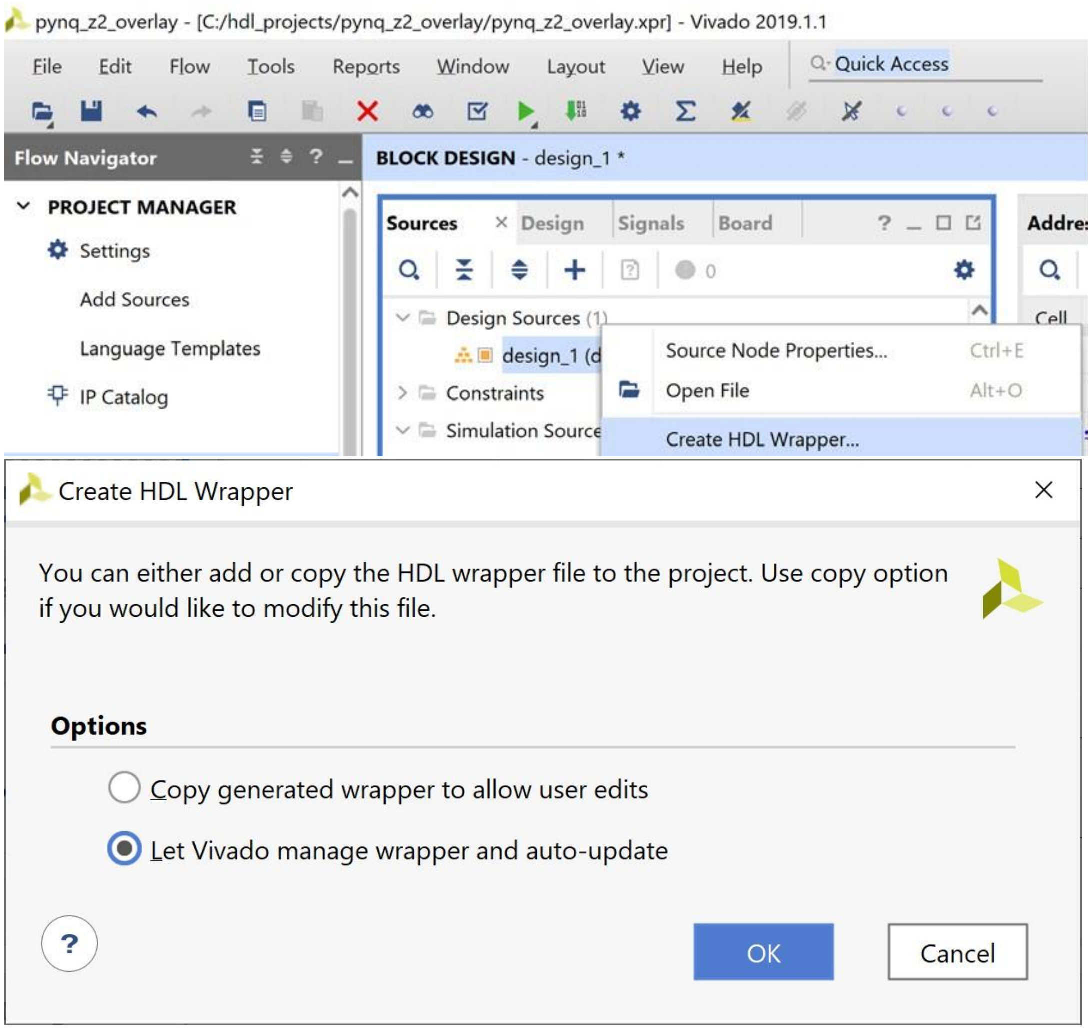


Figure 13New pipeline

After that validate design by pressing F6. To make sure that the design is correct.

6.6. Generating the new bitstream

- After completing the Vivado Block Design, the final step is to instantiate and produce the HDL wrapper, as well as generate Bitstream. from the source tab ‘Create HDL Wrapper' to generate the HDL wrapper.



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]

Figure 14creating HDL wrapper

- Similarly, the bitstream was generated by selecting Run synthesis from the flow navigator

- The user could run implementation from the synthesis log tab. This, on the other hand, clarifies that the block design is working as intended and without problems.
- When the implementation is finished, the Bitstream can be created.
- And then generate the Bitstream.

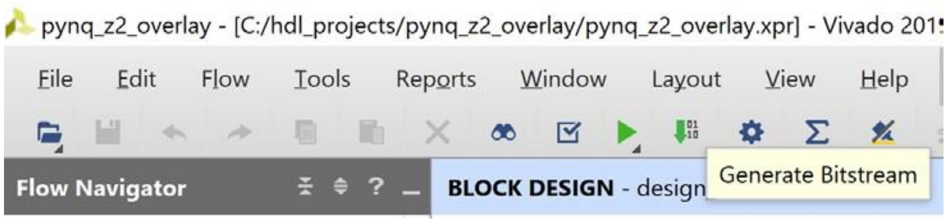


Figure 15generate bitstream

- Wait till you see write_bitstream complete in vivado

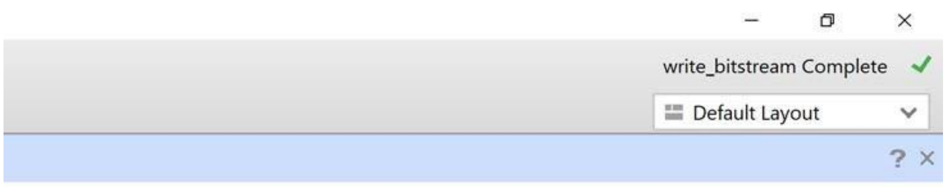
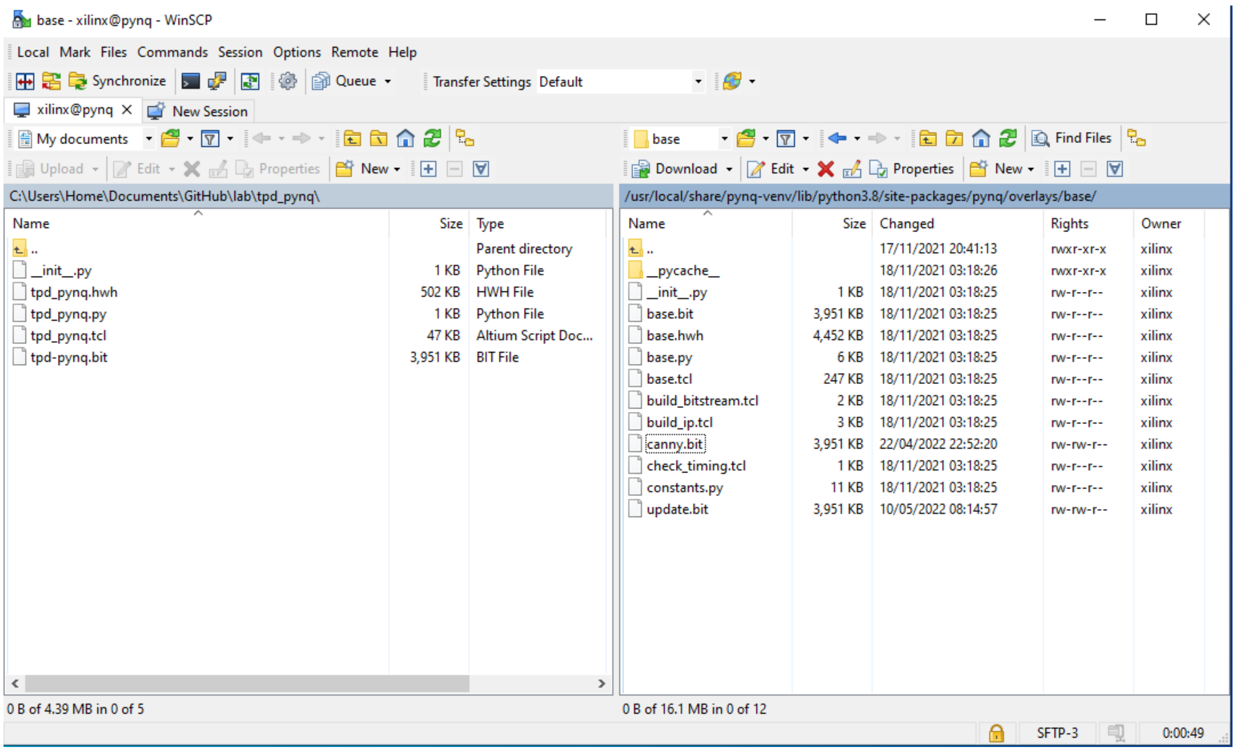


Figure 16bitstream completed
Uploading the bitstream to FPGA memory

In this step the generated bitstream will be

- Now the bitstream is ready to upload to the Pynq-Z2. For this power up the board and open WinSCP. Use the host name of pynq and the username and password of Xilinx.
- In WinSCP on your computer, see the left-hand side and select the base_wapper.bit file from C:\Users\Home\Documents\GitHub\PYNQ\boards\Pynq-Z2\base\base\base.runs\impl_1 directory on the target. Navigate to the pynq/overlays/base. Upload the base_wapper.bit file.
- Remane the bitstream to canny.bit.



C:\Users\Home\Documents\GitHub\lab\tpd_pynq\

Name	Size	Type
..		Parent directory
__init__.py	1 KB	Python File
tpd_pynq.hwh	502 KB	HWH File
tpd_pynq.py	1 KB	Python File
tpd_pynq.tcl	47 KB	Altium Script Doc...
tpd-pynq.bit	3,951 KB	BIT File

/usr/local/share/pynq-venv/lib/python3.8/site-packages/pynq/overlays/base/

Name	Size	Changed	Rights	Owner
..		17/11/2021 20:41:13	rwxr-xr-x	xilinx
__pycache__		18/11/2021 03:18:26	rwxr-xr-x	xilinx
__init__.py	1 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
base.bit	3,951 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
base.hwh	4,452 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
base.py	6 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
base.tcl	247 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
build_bitstream.tcl	2 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
build_ip.tcl	3 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
canny.bit	3,951 KB	22/04/2022 22:52:20	rw-rw-r--	xilinx
check_timing.tcl	1 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
constants.py	11 KB	18/11/2021 03:18:25	rw-r--r--	xilinx
update.bit	3,951 KB	10/05/2022 08:14:57	rw-rw-r--	xilinx

0 B of 4.39 MB in 0 of 5

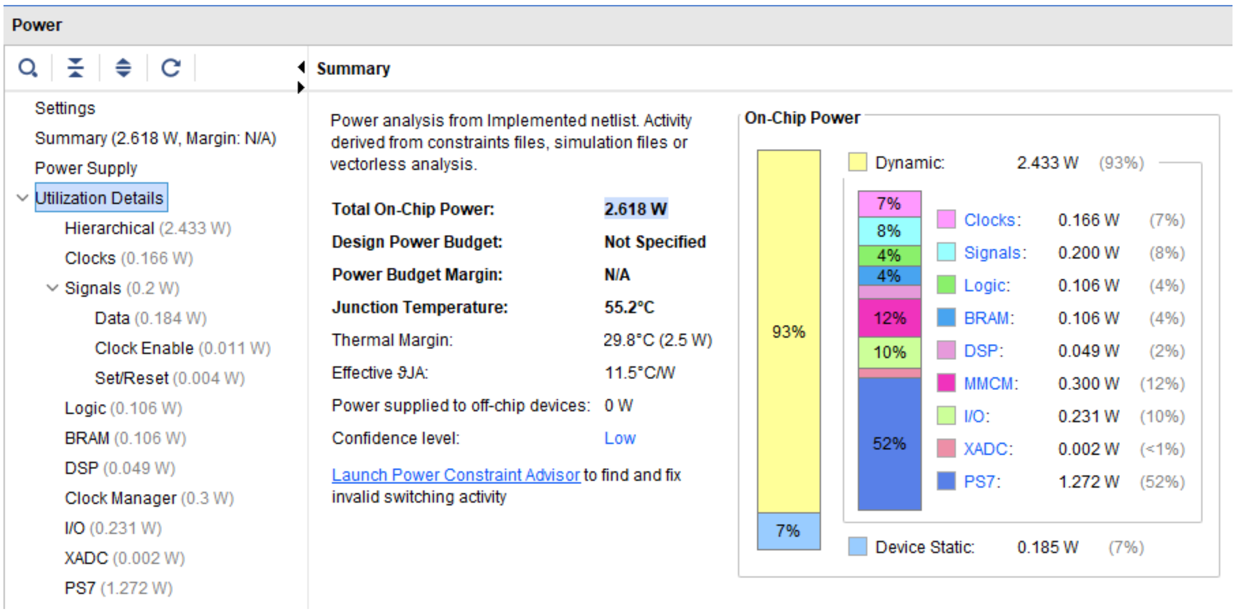
0 B of 16.1 MB in 0 of 12

SFTP-3 0:00:49

Figure 17uploading canny.bit
And this confirms that the new bitstream is uploaded to the board.

Testing

As the implementation and generate bitstream was successful. Now, the block design can be analysed, by selecting the appropriate report in the Flow navigator's implementation section, all of the design parameters may be assessed. The following investigation is carried out.


Power

Summary

- Settings
- Summary (2.618 W, Margin: N/A)
- Power Supply
- Utilization Details
  - Hierarchical (2.433 W)
  - Clocks (0.166 W)
  - Signals (0.2 W)
    - Data (0.184 W)
    - Clock Enable (0.011 W)
    - Set/Reset (0.004 W)
  - Logic (0.106 W)
  - BRAM (0.106 W)
  - DSP (0.049 W)
  - Clock Manager (0.3 W)
  - I/O (0.231 W)
  - XADC (0.002 W)
  - PS7 (1.272 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**2.618 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**55.2°C**
Thermal Margin:	29.8°C (2.5 W)
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		2.433 W	(93%)
	Clocks:	0.166 W	(7%)
	Signals:	0.200 W	(8%)
	Logic:	0.106 W	(4%)
	BRAM:	0.106 W	(4%)
	DSP:	0.049 W	(2%)
	MMCM:	0.300 W	(12%)
	I/O:	0.231 W	(10%)
	XADC:	0.002 W	(<1%)
	PS7:	1.272 W	(52%)
Device Static:		0.185 W	(7%)

Figure 18RTL analysis

Timing

To check the time constraints, utilise the report timing summary report. It also clarifies whether the design was finished on schedule. The Vivado IDE timing engine determines net delays based on connectivity and fanout. The accuracy of delamination is higher in nets between cells established by restrictions.

Net delays are based on routing information in the implemented design. Because there are three timing constraints stated, Worst Negative Slack (WNS) is used for all time paths, Worst Hold Slack (WHS) is used for minimum delay analysis, and Worst Pulse Width Slack (WPWS) is used to assess the pin switching limit. Similarly, pin switching limit is checked by pulse width, which works in the same way as the others. A positive worst-case setup time slack indicates that the limitation has been satisfied, whereas a negative slack indicates that the longest path has a path delay longer than the circuit's clock period.

Utilisation

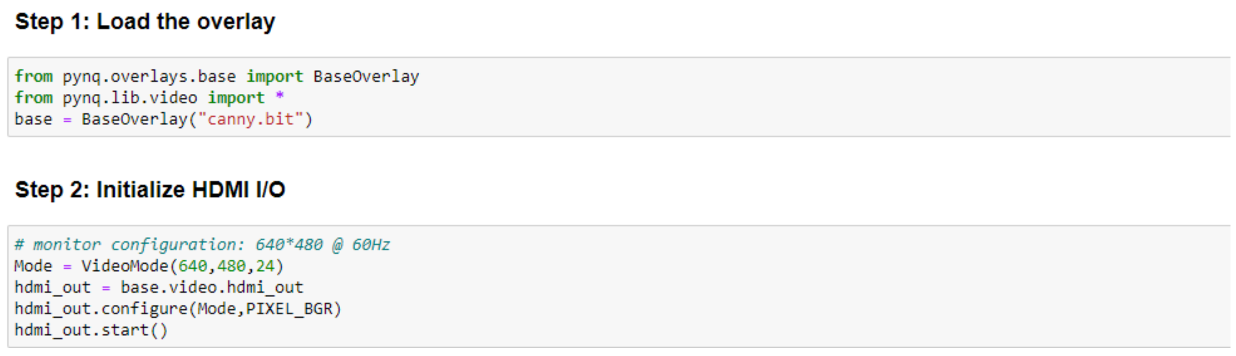
The report aids in the analysis of the designs, and the various resources are referred to as Utilization in the project report. This graph depicts the percentage of performance and resource utilisation for each parameter value for the measured configuration for a 'Linux with MMU' setup.

Power analysis

The power representation of On-Chip Power and dissipated power was used in the following analysis. Furthermore, the On-Chip total power is 2.618W, with pynq processor accounting for 52 percent of power usage. The clock generator consumes 7 percent of the device's power and runs at a junction temperature of 55.2 degrees.

Testing new pipeline.

Now after that the pipeline was tested for it runs all the old programs that base.bit works with. So, to begin the Opencv-filter_webcam was tested with the new bitstream. So, the new bitfile was add in place of base.bit.



Step 1: Load the overlay

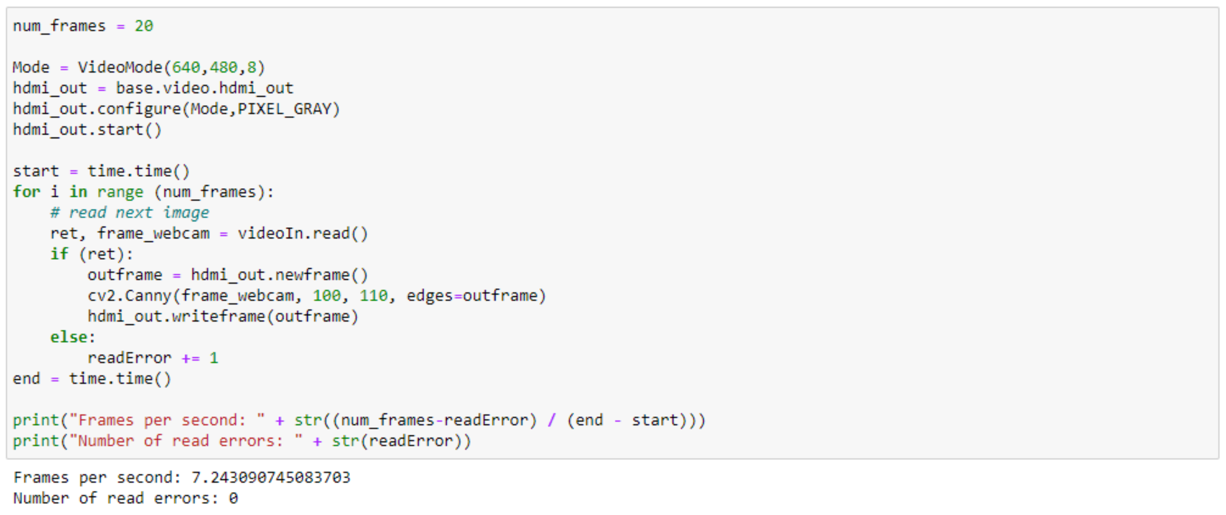
```
from pynq.overlays.base import BaseOverlay
from pynq.lib.video import *
base = BaseOverlay("canny.bit")
```

Step 2: Initialize HDMI I/O

```
# monitor configuration: 640*480 @ 60Hz
Mode = VideoMode(640,480,24)
hdmi_out = base.video.hdmi_out
hdmi_out.configure(Mode,PIXEL_BGR)
hdmi_out.start()
```

Figure 19 Loading canny.bit overlay

And on running this code, the outcome was same as earlier.



```
num_frames = 20

Mode = VideoMode(640,480,8)
hdmi_out = base.video.hdmi_out
hdmi_out.configure(Mode,PIXEL_GRAY)
hdmi_out.start()

start = time.time()
for i in range (num_frames):
    # read next image
    ret, frame_webcam = videoIn.read()
    if (ret):
        outframe = hdmi_out.newframe()
        cv2.Canny(frame_webcam, 100, 110, edges=outframe)
        hdmi_out.writeframe(outframe)
    else:
        readError += 1
end = time.time()

print("Frames per second: " + str((num_frames-readError) / (end - start)))
print("Number of read errors: " + str(readError))
```

```
Frames per second: 7.243090745083703
Number of read errors: 0
```

Figure 20Resulting output

With an expected frames per second: of 7.24.

7. Conclusion and further Improvements

Hence, the project is successful with the intended modifications and memory allocations working properly. It can be said that the pace of processing the image has been significantly increased with lowest requirement of energy. This is because of the merging of PYNQ base pipeline with OpenCV that the whole process is being carried out using python alone without any human intervention in the hardware. The system works smoothly and is manageable with low cost. This has been made possible with the programmable logic unit which does the computation task and exchanges information on parallel pipelines. All the determined features are implemented alongside numerous common image filters. The scope of the resolution which can be processed has extended to 1080 pixels and even further. Not only being economical in terms of creation, but is also quite simple to make, compared to other complex systems. Briefly, this has become an enhanced, simplified, and economised system with efficient working and handy features for the use.

Reference

8. References

[1] Á. Mándi, J. Mátè, D. Rózsa and S. Oniga, "Hardware accelerated image processing," *Carpathian Journal of Electronic and Computer Engineering,* 2021.

[2] "CV2PYNQ," [Online]. Available: https://github.com/wbrueckner/cv2pynq.

[3] A. taylor, "https://community.element14.com/learn/events/w/documents/4814/winners-announcement-pynq-z2-embedded-vision-workshop-zero-to-hero-series-with-adam-taylor," element 14, [Online]. Available: https://community.element14.com/learn/events/w/documents/4814/winners-announcement-pynq-z2-embedded-vision-workshop-zero-to-hero-series-with-adam-taylor.

[4] "PYNQ.read the doc," [Online]. Available: https://pynq.readthedocs.io/en/v2.0/pynq_overlays/base_overlay.html.

[5] Cathalmccabe, "reuilting Pynq," [Online]. Available: https://discuss.pynq.io/t/tutorial-rebuilding-the-pynq-base-overlay-pynq-v2-6/1993.

[6] Xilinx. [Online]. Available: ttps://www.xilinx.com/support/documentation/sw_manual.

[7] "PYNQ read the doc," [Online]. Available: https://pynq.readthedocs.io/en/v2.5/pynq_libraries/video.html.