Embedded Vision Workshop Series:

From Zero to Hero with PYNQ

Session 3

Unlocking Your Inner PYNQ Hero

Lab Workbook

Presented by

element14
AN AVNET COMMUNITY

Sponsored by

XILINX

PYNQ Workshop Series

About This Workbook

This workbook is intended to be used with **Session 3**: **Unlocking Your Inner PYNQ Hero** 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.

To access the workbook from Session 2: Getting Up and Running 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

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 Three

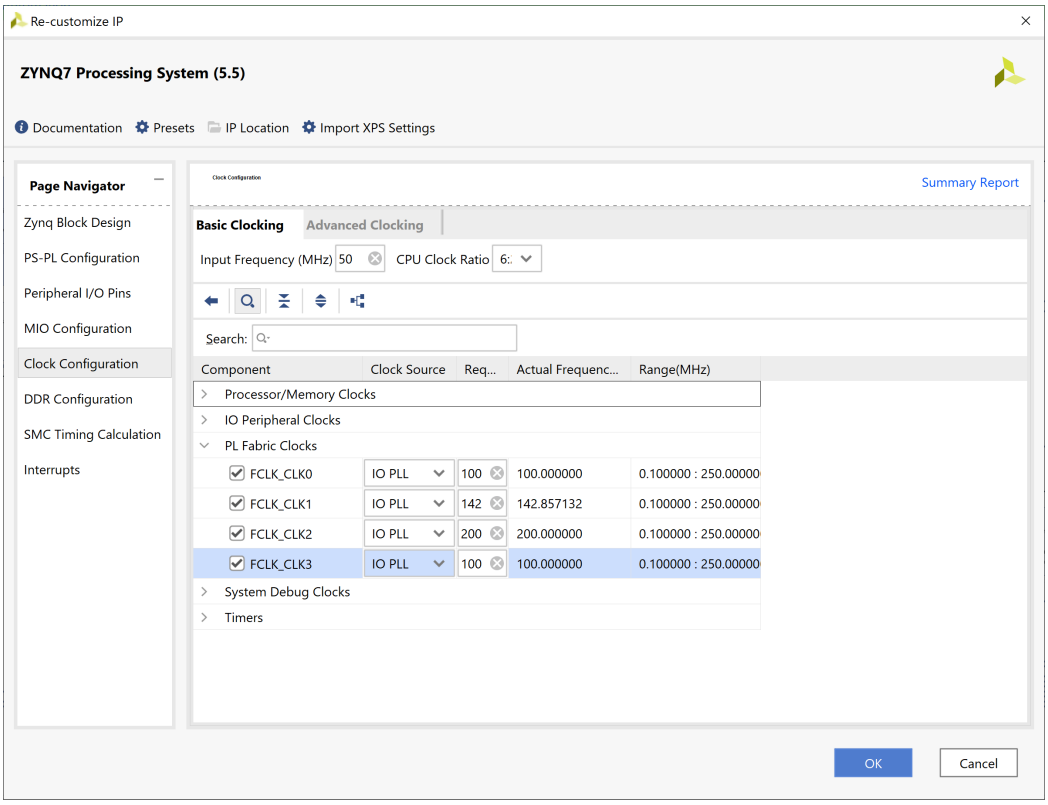
In this lab we are going to finish building our embedded vision project by updating the overlay created in previous lab to create an image processing application. At the end of the lab, you should have a camera system that is able to scan and read a barcode.

It is recommended you view this workbook in a PDF viewer where you can zoom in to ~300%, especially if you are attempting this lab prior to the live workshop session. This lab will require you to reference diagrams that appear small at 100%.

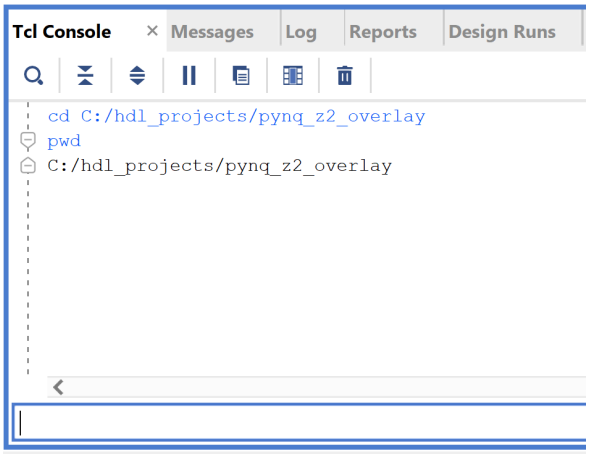
1. Starting with the Vivado Design Suite project we created in the last lab, select the clock connected to **FCLK0** and delete it.

2. Delete the **video test pattern generator**.

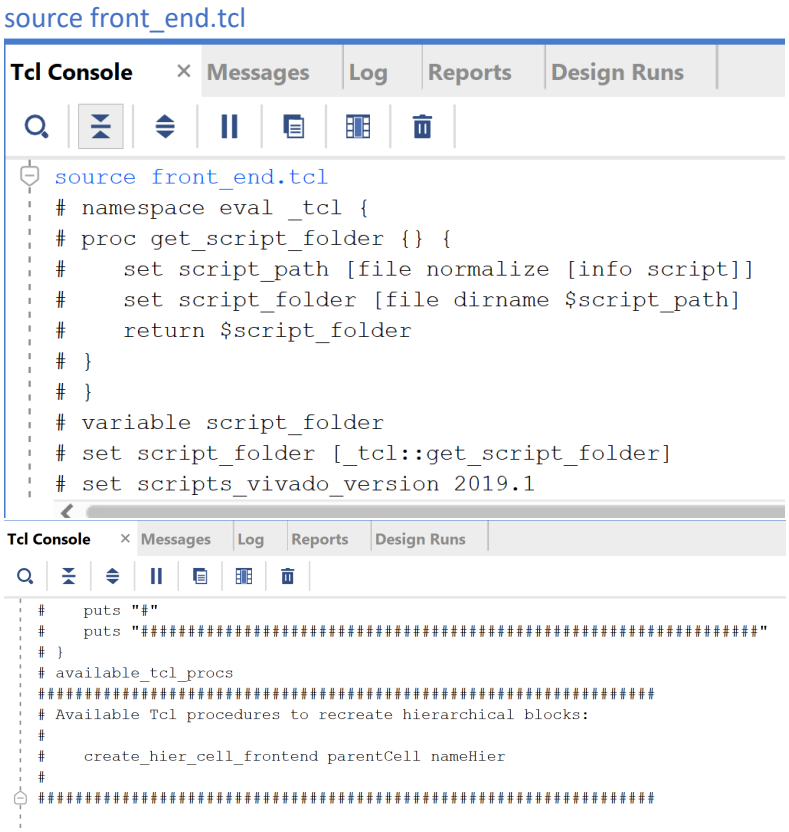
3. Open the **Zynq processing system** and select the **clock configuration** tab. Enable <u>all</u> the PS-PL clocks and set the following frequencies:
    a. FCLK0 = 100 MHz
    b. FCLK1 = 142 MHz
    c. FCLK2 = 200 MHz
    d. FCLK3 = 100 MHz



4. Change the directory in the Vivado TCL window to the project directory.



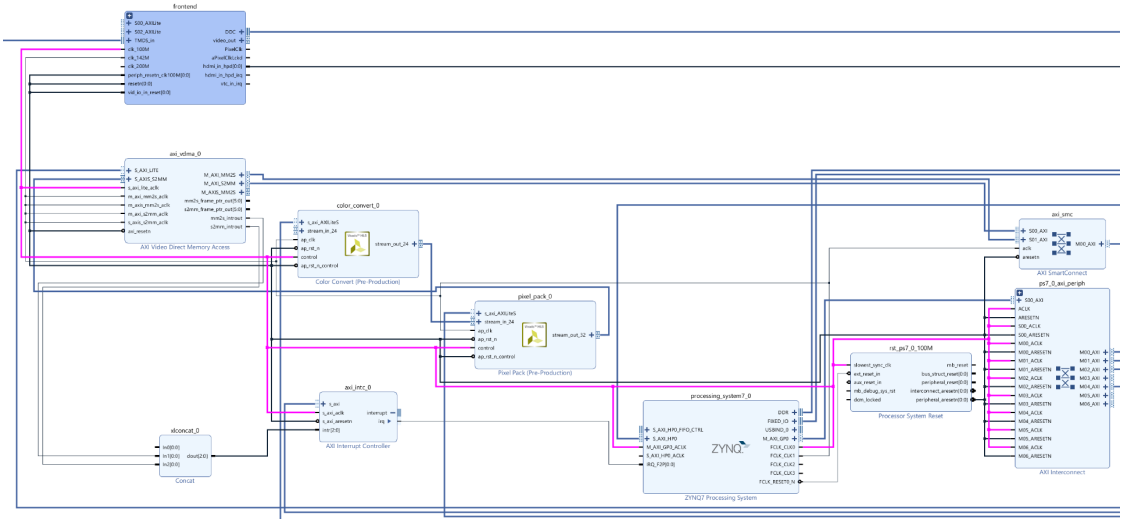
5. Copy the provided TCL file **front_end.tcl** into your project directory and source the file with the command. This will give us a new command which will create a TCL front end. It came from an extraction from the PYNQ-Z2 base design.



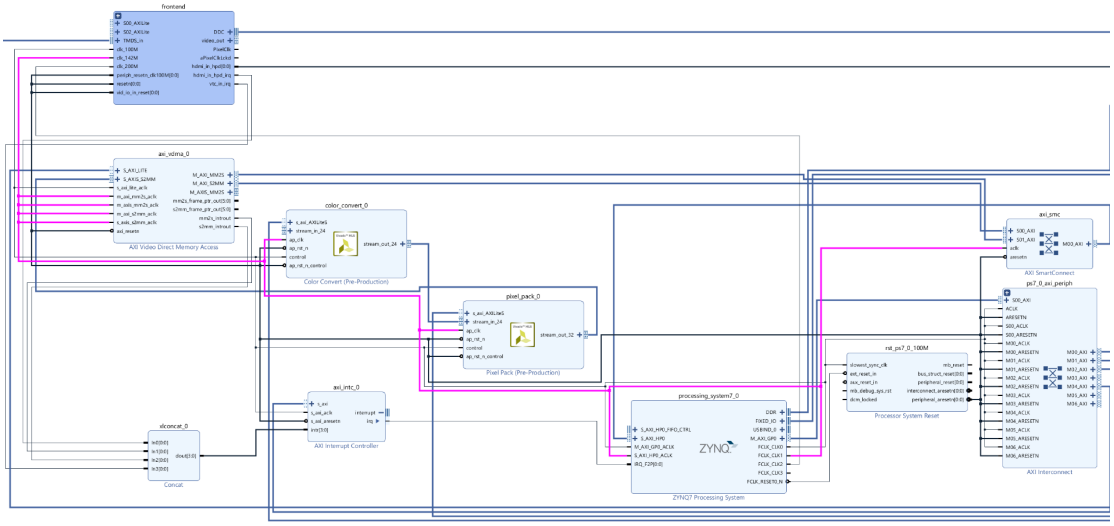
6. Call the new command to create the front end video module. This will put a new module on our block diagram.

   create_hier_cell_frontend . frontend

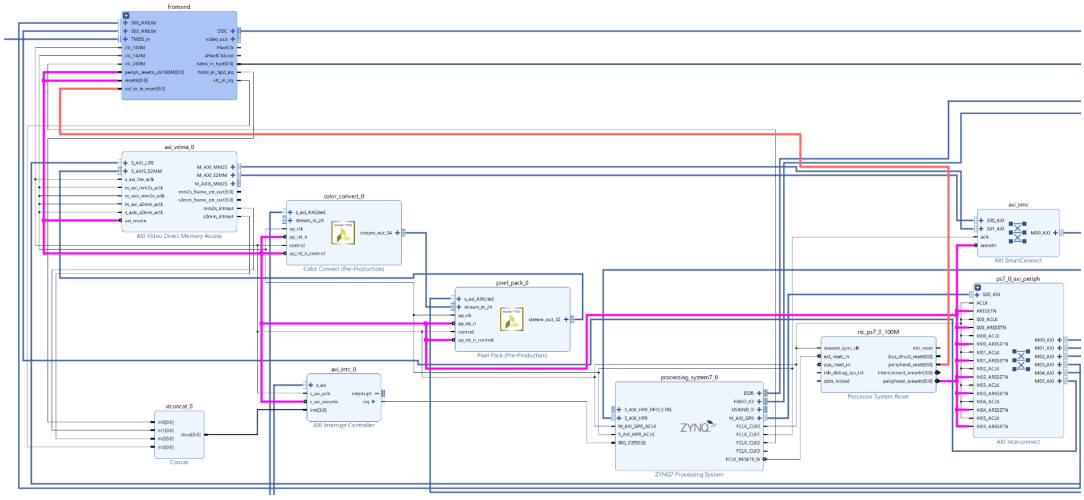
7. Connect up the **FCLK0** to our **AXI network**. This is the slow speed network, so it will be used for all the AXI-Lite communications.



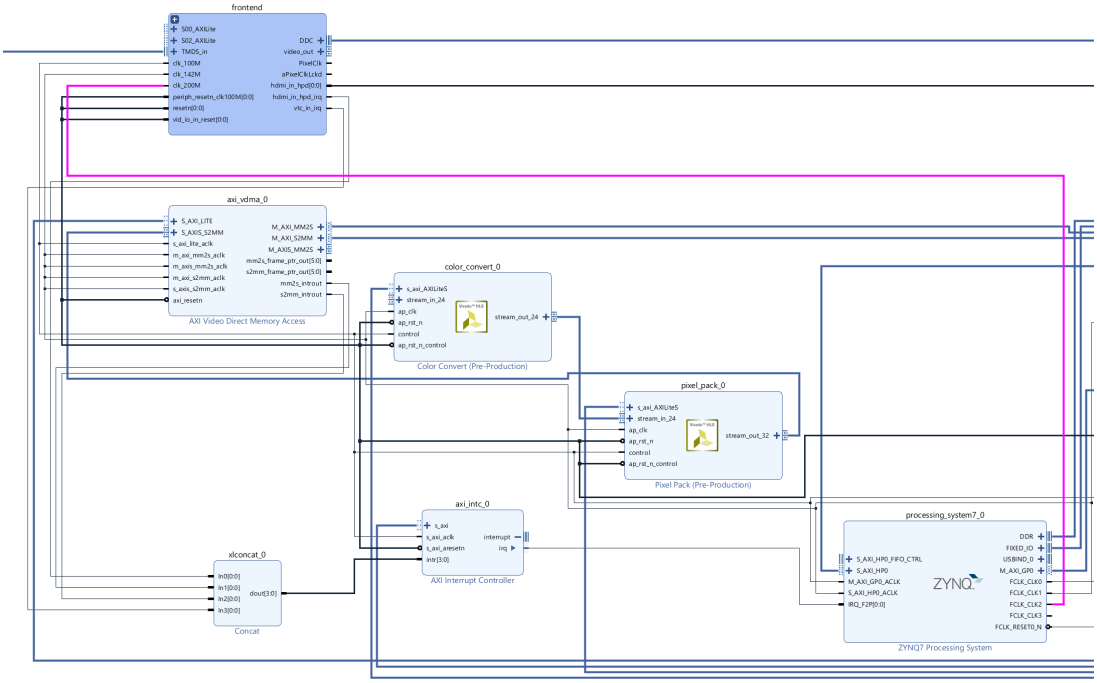
8. Connect **FCLK1** as shown in the diagram below.



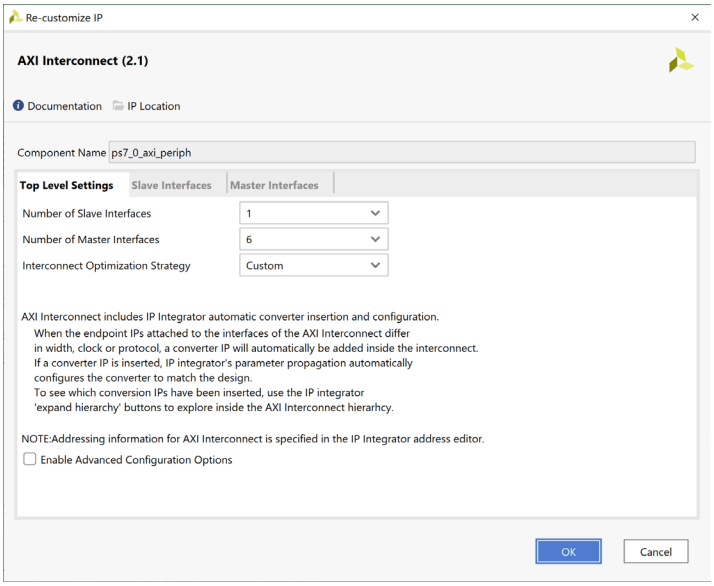
9. Connect the resets as shown.



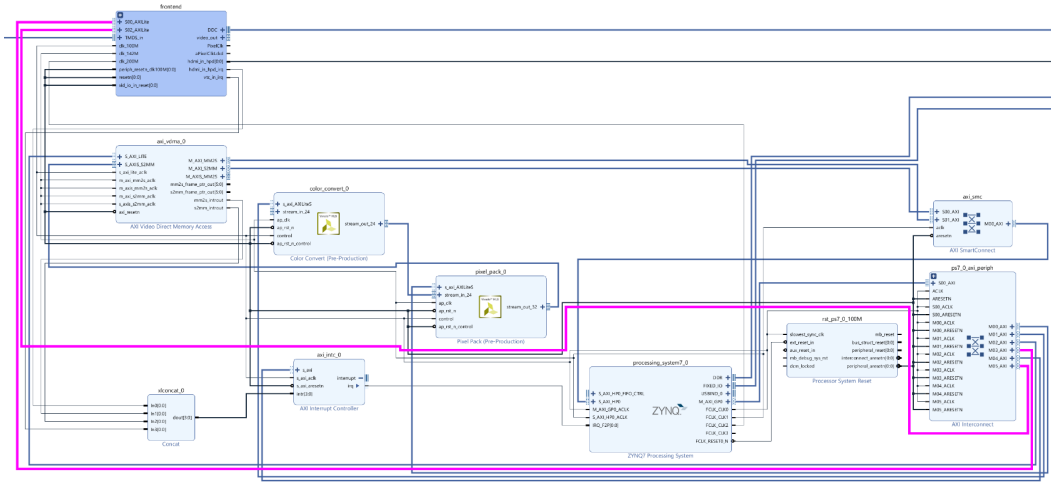
10. Connect the **FCLK2** to the front end **200 MHz clock input** and the **S_AXI_HP0_ACLK**.



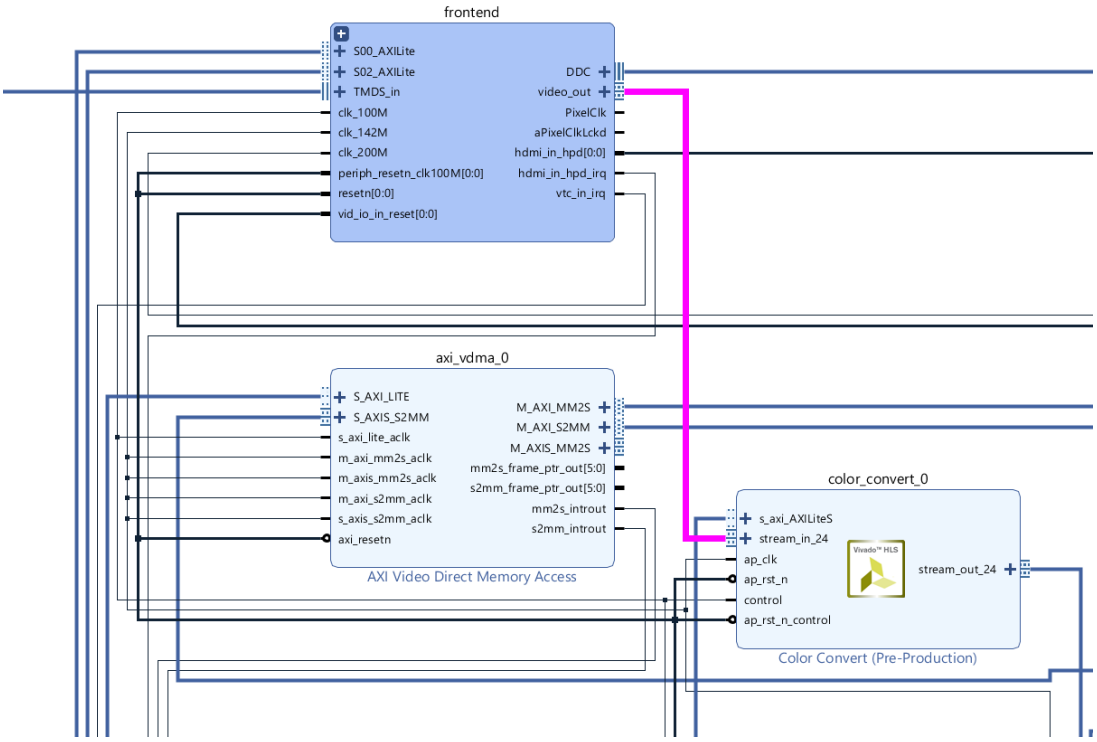
11. Re-customize the AXI Interconnect to have 6 master outputs.



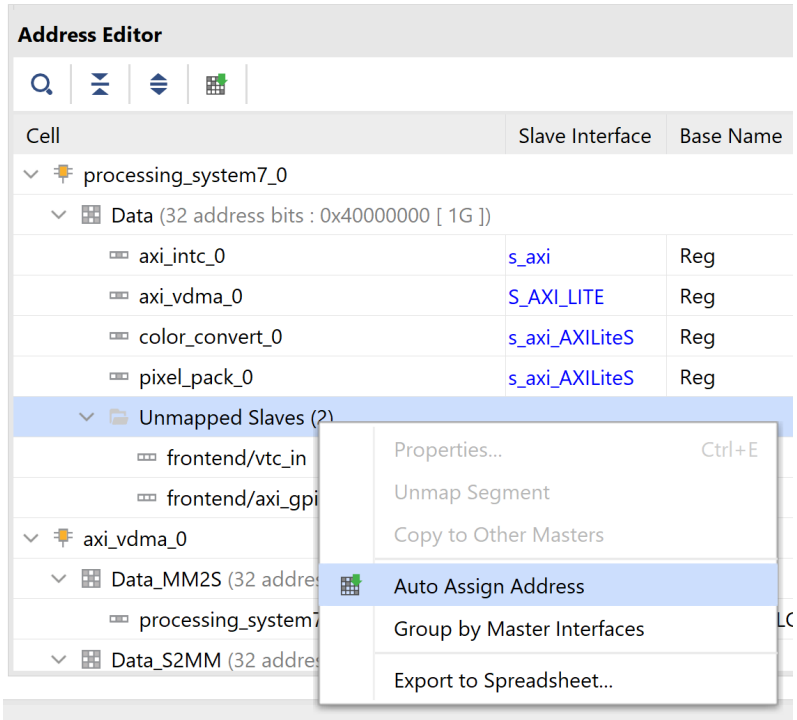
12. Connect the new master AXI ports on the **AXI Interconnect** to the **Slave AXI Interconnects** on the front end block.



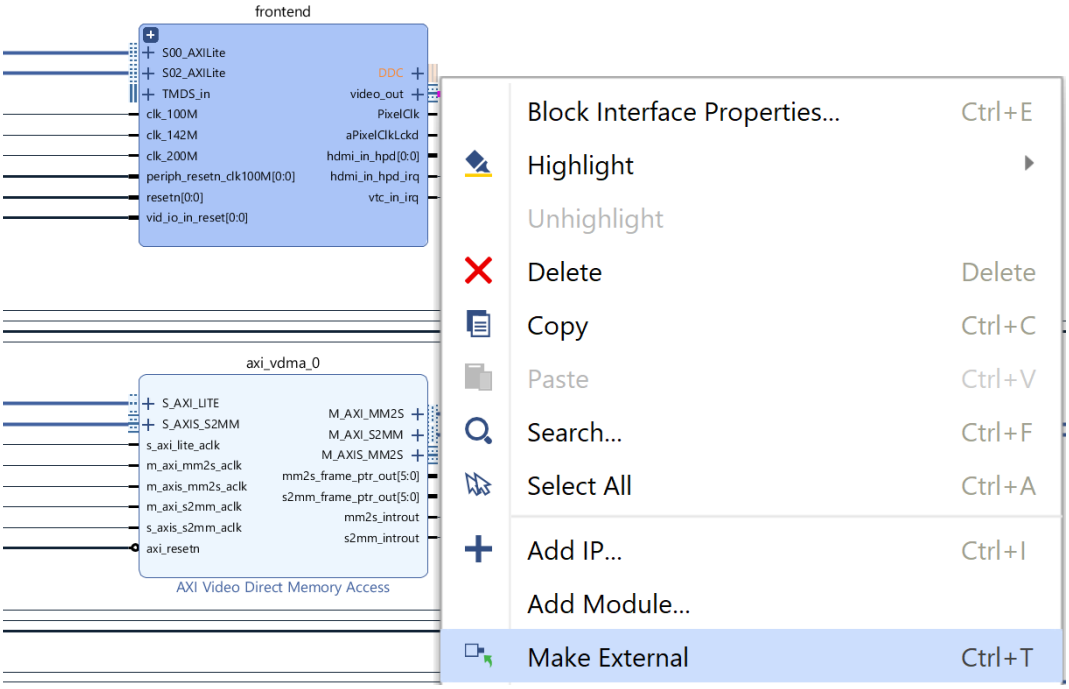
13. Connect the **output** (video_out) from the front end video to the **color convert module**.



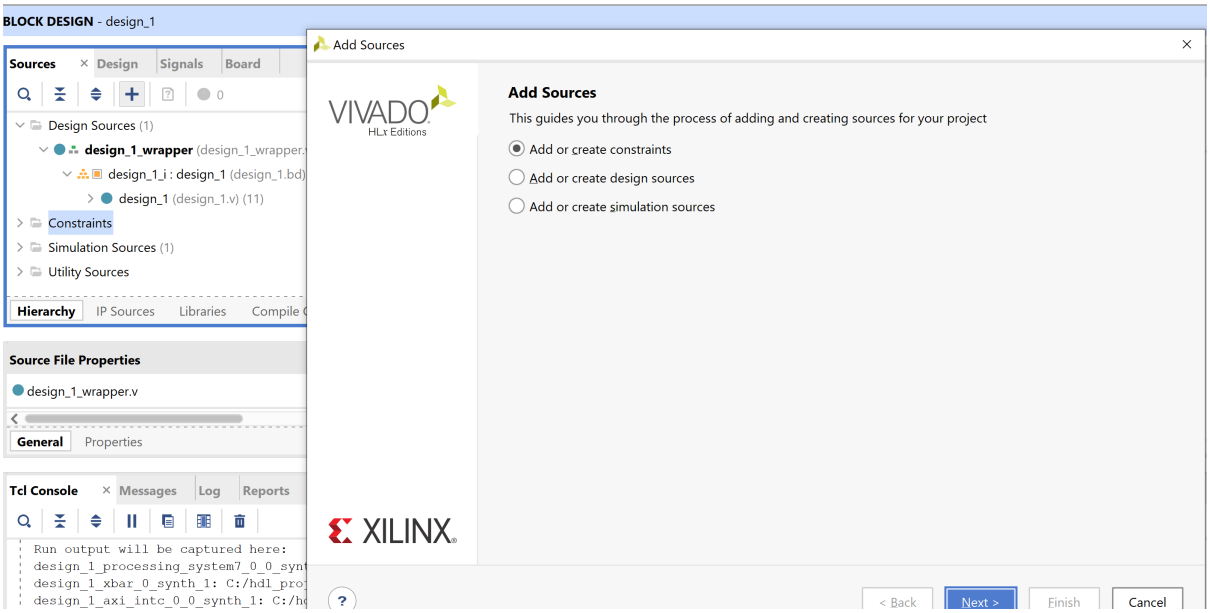
14. On the **Address Editor** tab, ensure all address are mapped by using the **Auto Assign Address**.



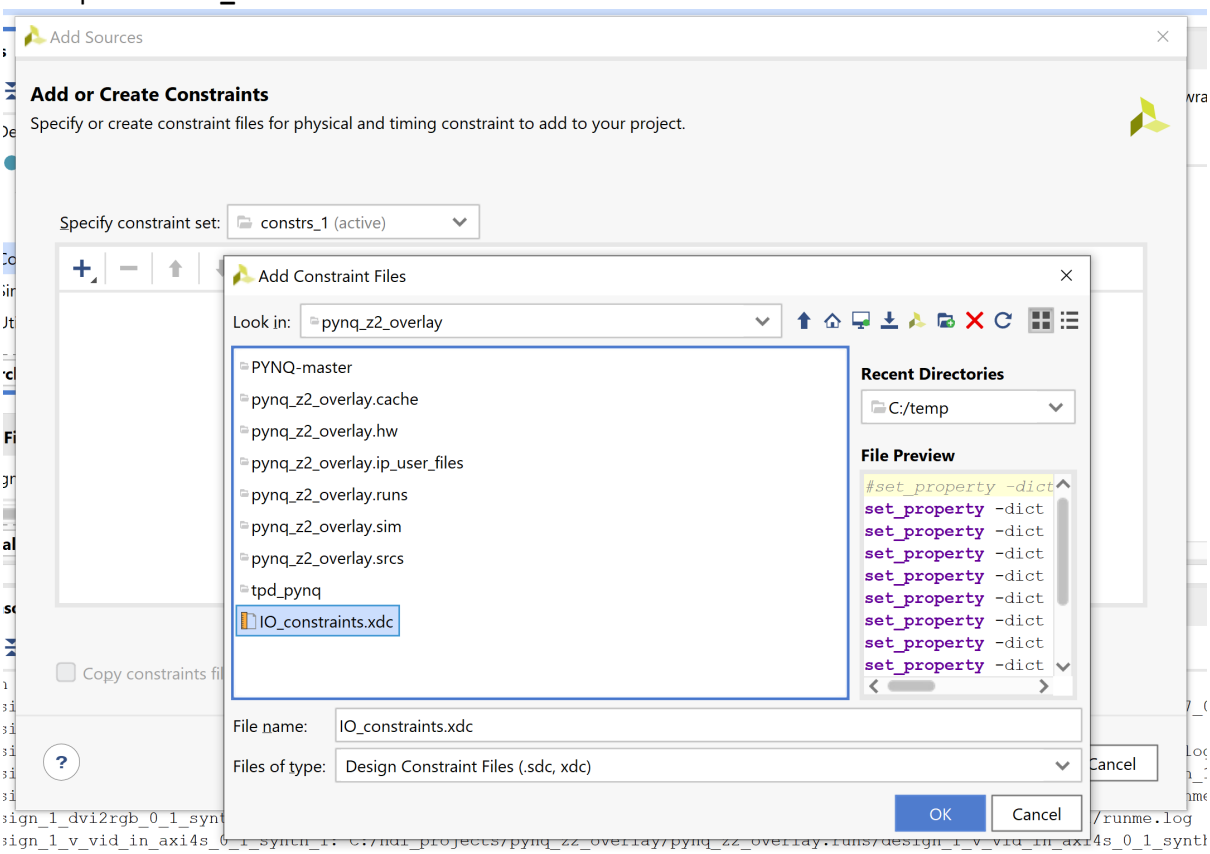
15. Right click on the **DDC port** and select **Make External**. Do the same for **TMDS_in** and **HDMI_in_hpd[0:0].**



16. The next step is to add the constraints. Select **Constraints → Add or create constraints**.

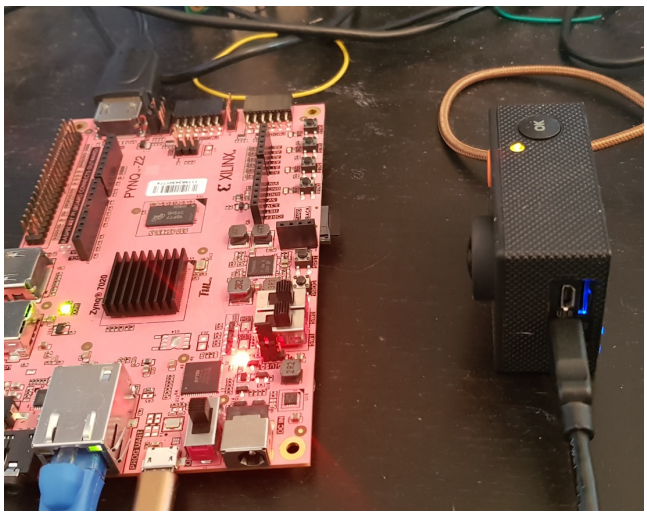


16. Select the provided **IO_constraints.xdc** file.

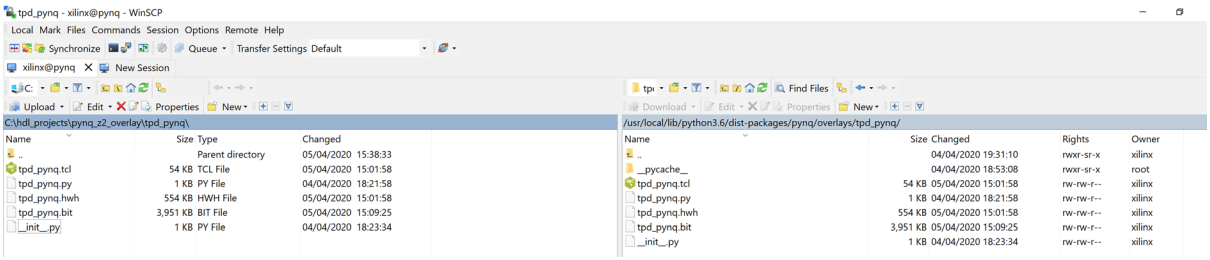


17. Validate and build the FPGA implementation.

18. Connect the PYNQ-Z2 board to the camera and power on the PYNQ-Z2.



19. Copy the updated **BIT file, tcl** and **HWH file** from the build into the **tpd_pynq directory**. Once in the directory, rename the files and use WinSCP to upload the updated application.



20. In Jupyter, open a new terminal and install the following:

sudo apt-get install libzbar0

```
root@pynq:/home/xilinx# sudo apt-get install libzbar0
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libzbar0
0 upgraded, 1 newly installed, 0 to remove and 409 not upgraded.
Need to get 66.2 kB of archives.
After this operation, 200 kB of additional disk space will be used.
Get:1 http://ports.ubuntu.com/ubuntu-ports bionic/universe armhf libzbar0 armhf 0.10+doc-10.1build2 [66.2 kB]
Fetched 66.2 kB in 0s (359 kB/s)
Selecting previously unselected package libzbar0:armhf.
(Reading database ... 122537 files and directories currently installed.)
Preparing to unpack .../libzbar0_0.10+doc-10.1build2_armhf.deb ...
Unpacking libzbar0:armhf (0.10+doc-10.1build2) ...
Setting up libzbar0:armhf (0.10+doc-10.1build2) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
root@pynq:/home/xilinx#
```

21. In the same terminal enter the following command:

pip3 install pyzbar

```
root@pynq:/home/xilinx# pip3 install pyzbar
Collecting pyzbar
  Downloading https://files.pythonhosted.org/packages/46/7e/d2ad702facc47c0b3106a494f5dfbc3f296aab7a07dac05d1f30bdad0fab/pyzbar-0.1.8-py2.py3-n
ne-any.whl
Installing collected packages: pyzbar
Successfully installed pyzbar-0.1.8
```

22. Open the **tpd.ipynb** and modify the code as shown below. This will allow us to capture images and decode barcodes within the image. It is easier just to copy and paste everything below to replace the existing code.

```
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
from pynq.lib.video import *
import cv2
import matplotlib.pyplot as plt
overlay = tpd_pynqOverlay('tpd_pynq.bit')

hdmiin_frontend = overlay.frontend
hdmiin_frontend.start()
hdmiin_frontend.mode

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 = 480
framemode = VideoMode(640, lines, 24)
cam_vdma.readchannel.mode = framemode
cam_vdma.readchannel.start()

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

```
cv2.imwrite('img.jpg', frame_color)

from pyzbar import pyzbar
from pyzbar.pyzbar import decode

image = cv2.imread("img.jpg")
image =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
cv2.imwrite('gray.jpg', image)
ret,thresh1 = cv2.threshold(image,100,255,cv2.THRESH_BINARY)
blur = cv2.GaussianBlur(thresh1,(5,5),0)
cv2.imwrite('thres.jpg', blur)
barcodes = decode(blur)

# loop over the detected barcodes
for barcode in barcodes:
    # extract the bounding box location of the barcode and draw the
    # bounding box surrounding the barcode on the image
    (x, y, w, h) = barcode.rect
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
    # the barcode data is a bytes object so if we want to draw it on
    # our output image we need to convert it to a string first
    barcodeData = barcode.data.decode("utf-8")
    barcodeType = barcode.type
    # draw the barcode data and barcode type on the image
    text = "{} ({})".format(barcodeData, barcodeType)
    cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
        0.5, (0, 0, 255), 2)
    # print the barcode type and data to the terminal
    print("[INFO] Found {} barcode: {}".format(barcodeType, barcodeData))
# show the output image
#cv2.imshow("Image", image)
cv2.imwrite('final.jpg', image)

barcodes
```

In [17]:
```
frame_camera = cam_vdma.readchannel.readframe()
frame_color=cv2.cvtColor(frame_camera,cv2.COLOR_BGR2RGB)
#frame_color = frame_camera
pixels = np.array(frame_color)
plt.imshow(pixels)
plt.show()
```

In [22]:
```
# loop over the detected barcodes
for barcode in barcodes:
    # extract the bounding box location of the barcode and draw the
    # bounding box surrounding the barcode on the image
    (x, y, w, h) = barcode.rect
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
    # the barcode data is a bytes object so if we want to draw it on
    # our output image we need to convert it to a string first
    barcodeData = barcode.data.decode("utf-8")
    barcodeType = barcode.type
    # draw the barcode data and barcode type on the image
    text = "{} ({})".format(barcodeData, barcodeType)
    cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
        0.5, (0, 0, 255), 2)
    # print the barcode type and data to the terminal
    print("[INFO] Found {} barcode: {}".format(barcodeType, barcodeData))
# show the output image
#cv2.imshow("Image", image)
cv2.imwrite('final.jpg', image)
```

```
[INFO] Found CODE128 barcode: 00123456789101112133
```

Out[22]:
```
True
```

In [23]:
```
barcodes
```

Out[23]:
```
[Decoded(data=b'00123456789101112133', type='CODE128', rect=Rect(left=122, top=158, width=389, height=32), polygon=[Point(x=12
2, y=187), Point(x=511, y=190), Point(x=507, y=158)])]
```

23. On the Jupyter Notebook homepage, you should see a number of image files including **final.jpg** which will show the decoded barcode imposed on the image and the location of the bar code.

Congratulations! You have completed the workshop series and built an embedded vision application using PYNQ. You have unlocked your inner PYNQ hero!