tags: `MSOC`

MSOC Video edge Report

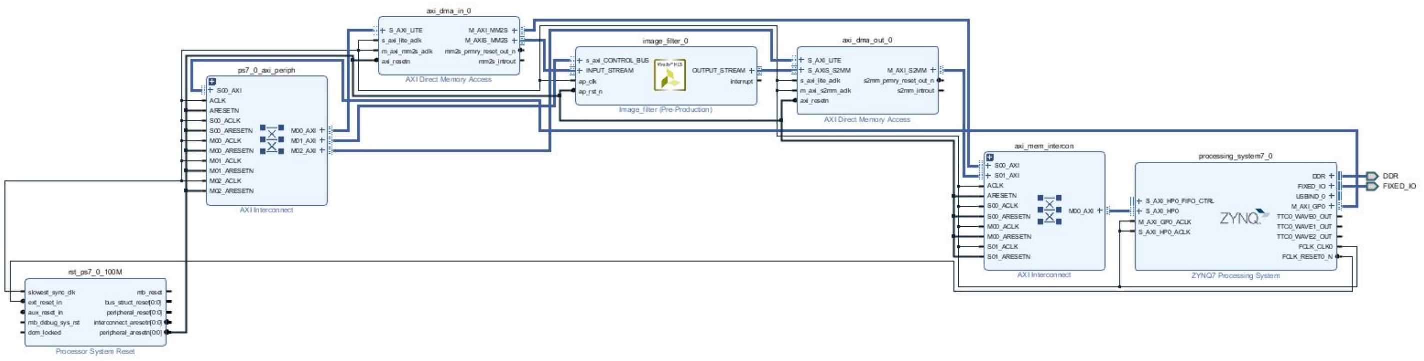
Objective

After completing this lab, we will be able to learn:

- Use Vivado-HLS API to generate a Video edge detection RTL IP with AXI-Stream interface from System-C code.
- Use Opencv-hls library to help develop the project.
- **[Additional]** Build a PYNQ host system program and verify the design functionality on the Zedboard.
- **[Additional]** Demonstrate on a vehicle recording dataset.

System diagram

This lab intends to develop a **Video edge detection** through the AXIStream interface.



Interface

This design utilize the **AXI-Stream interface** to transfer the data and **AXILiteS interface** to transfer parameters. The following table shows the usage for the mmio.

Addr. (+Base addr.)	bit	Usage
0x00	0	ap_start
0x00	1	ap_done
0x00	2	ap_idle
0x00	3	ap_ready
0x00	7	auto_restart
0x04	0	Global interrupt
0x14		Num. of rows
0x1C		Num. of cols

Notice that the mmio can be set in the hls offset #pragma.
The original provided code sets the rows and cols as 'ap_stable', so these parameters will be load from ROM instead of setting from the IP input.

C code

```
1  void image_filter(AXI_STREAM& INPUT_STREAM, AXI_STREAM& OUTPUT_STREAM) {
2
3  #pragma HLS INTERFACE axis depth=10000 port=INPUT_STREAM bundle=VIDEO_IN
4  #pragma HLS INTERFACE axis depth=10000 port=OUTPUT_STREAM bundle=VIDEO_OUT
5
6  const int rows = MAX_HEIGHT;
7  const int cols = MAX_WIDTH;
8
9  RGB_IMAGE  img_0  (rows, cols);
10 RGB_IMAGE  img_1a (rows, cols);
11 RGB_IMAGE  img_1b (rows, cols);
12 RGB_IMAGE  img_2a (rows, cols);
13 RGB_IMAGE  img_2b (rows, cols);
14 RGB_IMAGE  img_3  (rows, cols);
15
16 hls::AXIvideo2Mat(INPUT_STREAM, img_0);
17 replicate_by2<RGB_IMAGE,RGB_PIXEL>(img_0, img_1a, img_1b, rows, cols);
18 grad_vertical<RGB_IMAGE>(img_1a, img_2a, rows, cols);
19 grad_horizontal<RGB_IMAGE>(img_1b, img_2b, rows, cols);
20 add_with_color<RGB_IMAGE,RGB_PIXEL>(img_2a, img_2b, img_3, rows, cols);
21 hls::Mat2AXIvideo(img_3, OUTPUT_STREAM);
22 }
```

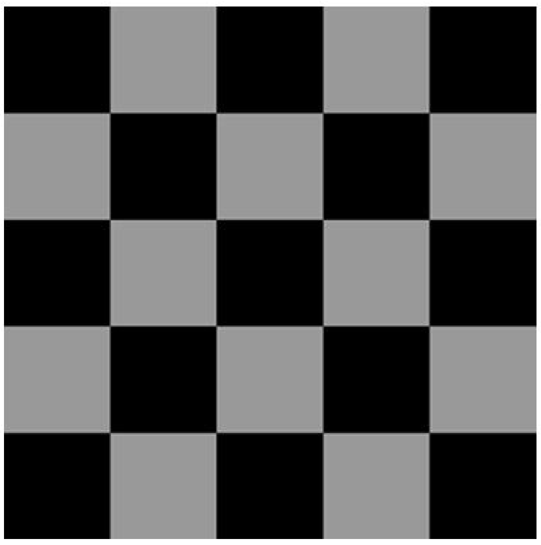
- Line 16: Transform the AXI input into the hls:Mat format. In this function it handles the last, user and keep signal in the AXI_Stream structure datatype. It will check if the start of line (sof) is transmit every rows*cols cycle, if end of line (eof) is transmit after each row transmission. Also, it will transform the ap_axiu<32,1,1,1> data type into three unsigned char by calling the hls::AXIGetBitFields() function. (More detail can be found in the vivado-hls library)

- Line 17: copy img_0 to img_1a and img_2a
- Line 18~19: Filter the image by calling the hls::Filter2D function. It requires the kernel input and output steaming datatype and anchor.
- Line 20: If the vertical grad value is greater than 100 then marked it red, if the horizontal grad value is greater than 100, marked it green.

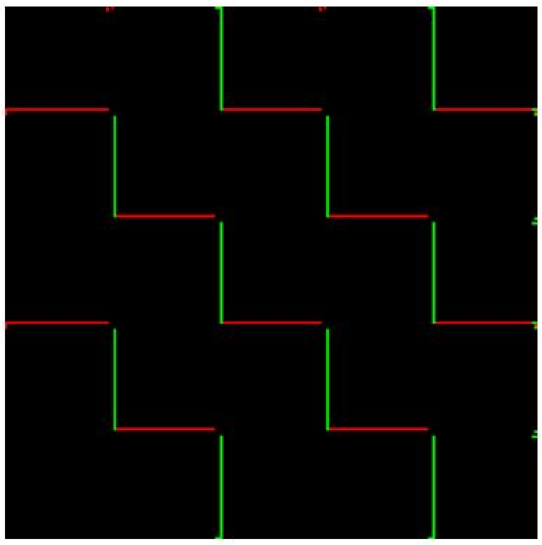
Notice that we have modified the 'hls::AXIvideo2Mat' and 'hls::Mat2AXIvideo' function to 'myAXIvideo2Mat' and 'myMat2AXIvideo' in order to met the DMA transmittion protocal (will be shown in hte 'Image transmission protocals' parts).

An exmaple

The following figure shows the orignal figure and the figure after edge detection.



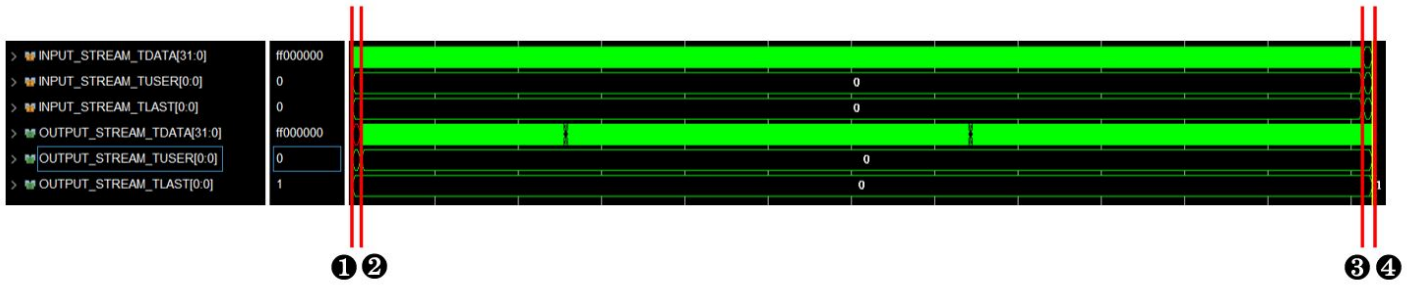
Original



After

Cosim waveform

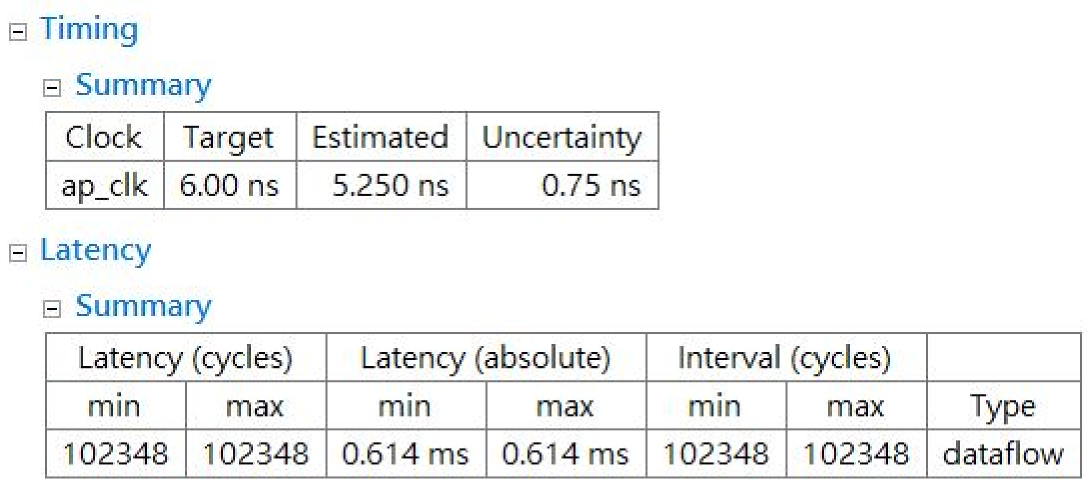
The whole waveform is shown in the figure below (Notice that this waveform is **the modified one**, refer to 'Image tranmission protocals' part).



❶ Input stream TUSER is pull high: the first pixel is input.
❷ Output stream TUSER is pull high: the first pixel is output.
❶~❸: Data input in streaming format.
❸ Input stream TLAST is pull high: the last pixel is input.
❷~❹: Data output in streaming format.
❹ Output stream TLAST is pull high: the last pixel is output.

Performance and Analysis

We can see that it **consumes 102348** for calculation and the **estimated clock rate is 5.250s** from the report below:



⊟ Timing

⊟ Summary

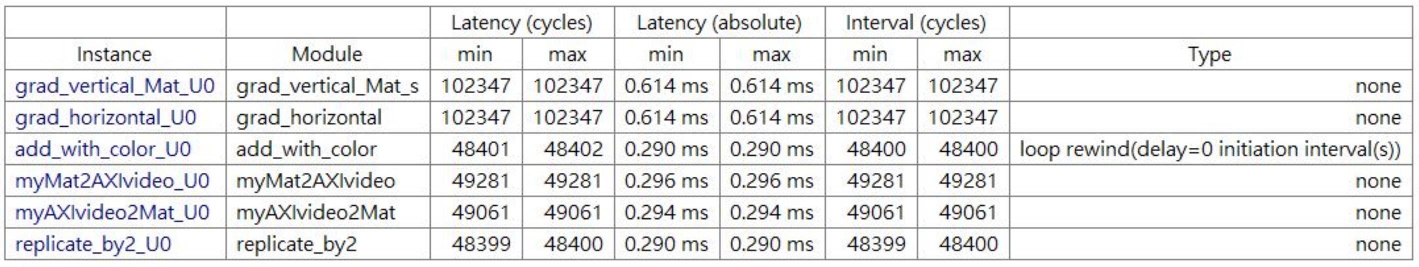
Clock	Target	Estimated	Uncertainty
ap_clk	6.00 ns	5.250 ns	0.75 ns

⊟ Latency

⊟ Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
102348	102348	0.614 ms	0.614 ms	102348	102348	dataflow

The following figure shows the detail latency of each block.



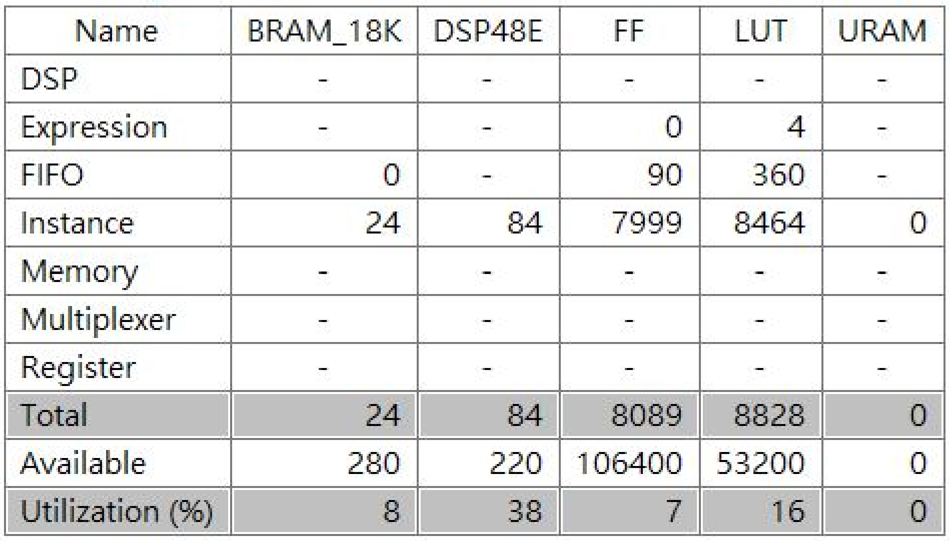
		Latency (cycles)		Latency (absolute)		Interval (cycles)		
Instance	Module	min	max	min	max	min	max	Type
grad_vertical_Mat_U0	grad_vertical_Mat_s	102347	102347	0.614 ms	0.614 ms	102347	102347	none
grad_horizontal_U0	grad_horizontal	102347	102347	0.614 ms	0.614 ms	102347	102347	none
add_with_color_U0	add_with_color	48401	48402	0.290 ms	0.290 ms	48400	48400	loop rewind(delay=0 initiation interval(s))
myMat2AXIvideo_U0	myMat2AXIvideo	49281	49281	0.296 ms	0.296 ms	49281	49281	none
myAXIvideo2Mat_U0	myAXIvideo2Mat	49061	49061	0.294 ms	0.294 ms	49061	49061	none
replicate_by2_U0	replicate_by2	48399	48400	0.290 ms	0.290 ms	48399	48400	none

Notice that the latency report in summary is the maximum of all block due to the ‘HLS dataflow #pragma’.

Since the fucntion add with color has three procedures, we add the loop rewind pragma to further reduce the latency.

Utilization

The utilization estimation is shown below:

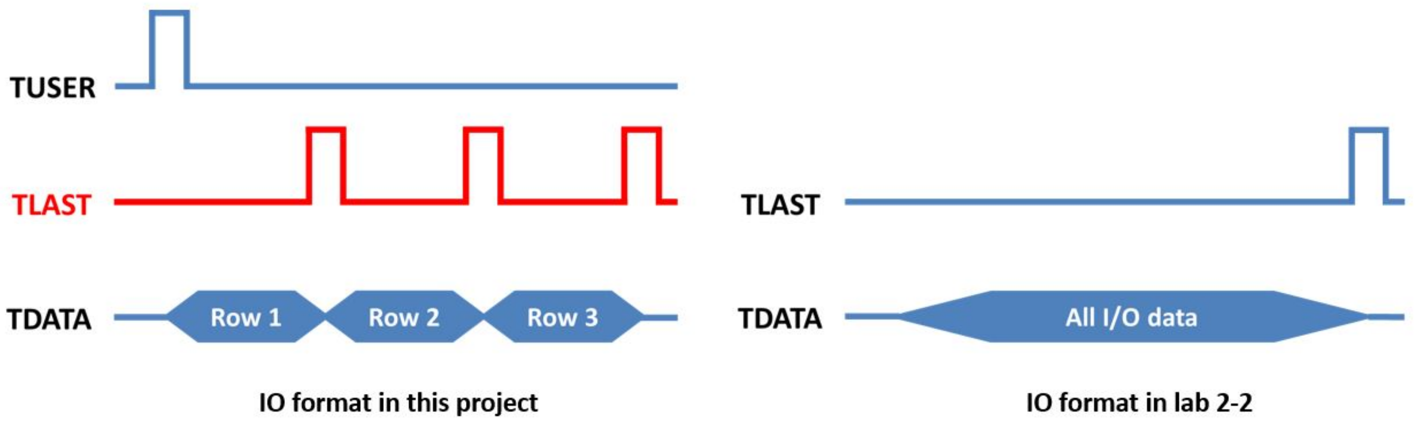


Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	4	-
FIFO	0	-	90	360	-
Instance	24	84	7999	8464	0
Memory	-	-	-	-	-
Multiplexer	-	-	-	-	-
Register	-	-	-	-	-
Total	24	84	8089	8828	0
Available	280	220	106400	53200	0
Utilization (%)	8	38	7	16	0

Since this design use the HLS-streaming datatype for the image flow, it use FIFO as implementation. Compared with full array, which will map to BRAM, it saves large amount of BRAMs.

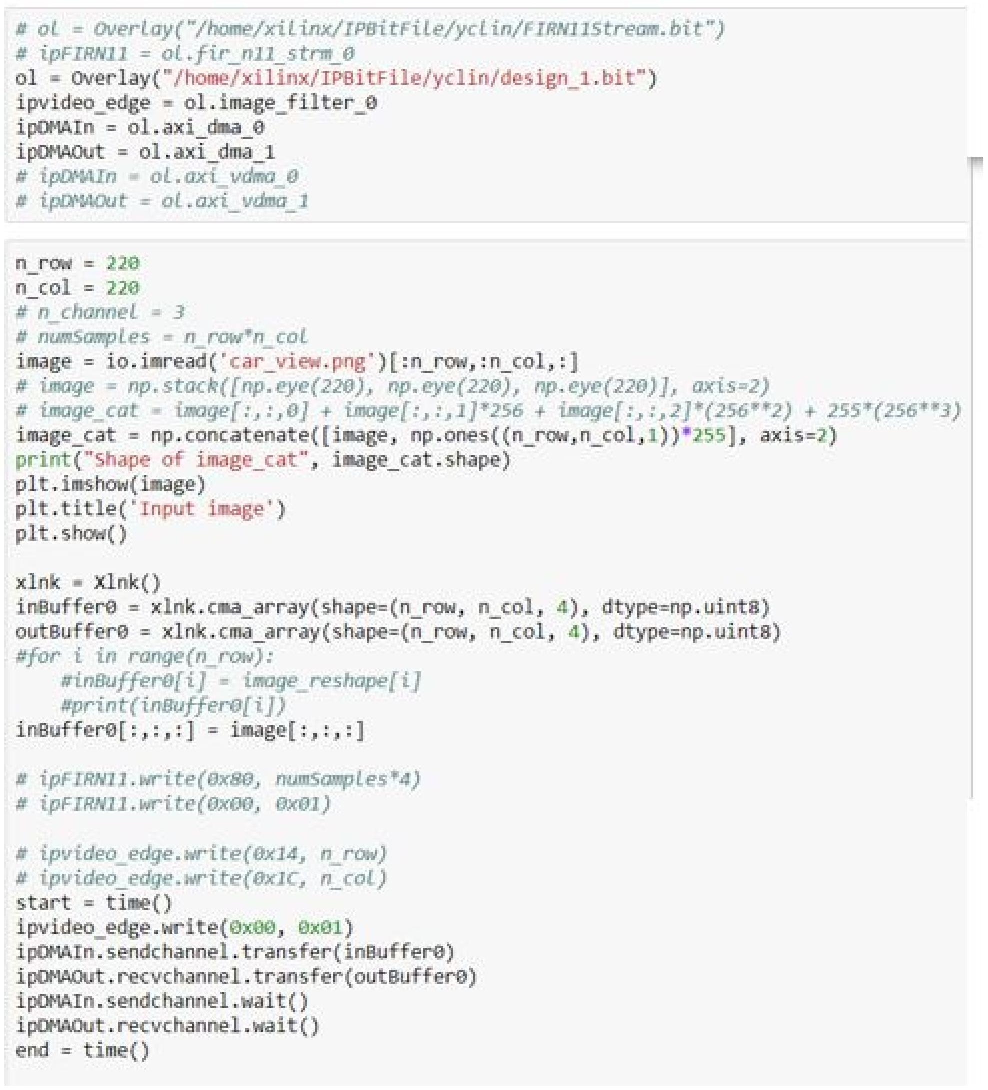
Image tranmission protocals

The following figure show the differnet between the transmission protocals of Lab2-2 and this project.



In Lab2-2, ‘TLAST’ bit indicates the end of the 1D data stream. While in this project, the provided ‘AXIvideo2Mat’ function, ‘TUSER’ still indicates the first pixel, but ‘TLAST’ is pulled high after every row transmission.
In order to simulate the transmit protocal in PYNQ host program, we modified the ‘IplImage2AXIvideo’ function provided in “hls_opencv.h” called in the Tester program. Also, we re-write the ‘AXIvideo2Mat’ funciton to avoid detecting the ‘TUSER’ bit from DMA (actually the DMA has no ‘TUSER’ port).

Python host program



```
# ol = Overlay("/home/xilinx/IPBitFile/yclin/FIRN11Stream.bit")
# ipFIRN11 = ol.fir_n11_strm_0
ol = Overlay("/home/xilinx/IPBitFile/yclin/design_1.bit")
ipvideo_edge = ol.image_filter_0
ipDMAIn = ol.axi_dma_0
ipDMAOut = ol.axi_dma_1
# ipDMAIn = ol.axi_vdma_0
# ipDMAOut = ol.axi_vdma_1
```

```
n_row = 220
n_col = 220
# n_channel = 3
# numSamples = n_row*n_col
image = io.imread('car_view.png')[:n_row,:n_col,:]
# image = np.stack([np.eye(220), np.eye(220), np.eye(220)], axis=2)
# image_cat = image[:,:,0] + image[:,:,1]*256 + image[:,:,2]*(256**2) + 255*(256**3)
image_cat = np.concatenate([image, np.ones((n_row,n_col,1))*255], axis=2)
print("Shape of image_cat", image_cat.shape)
plt.imshow(image)
plt.title('Input image')
plt.show()

xlnk = Xlnk()
inBuffer0 = xlnk.cma_array(shape=(n_row, n_col, 4), dtype=np.uint8)
outBuffer0 = xlnk.cma_array(shape=(n_row, n_col, 4), dtype=np.uint8)
#for i in range(n_row):
    #inBuffer0[i] = image_reshape[i]
    #print(inBuffer0[i])
inBuffer0[:,:,:] = image[:,:,:]

# ipFIRN11.write(0x80, numSamples*4)
# ipFIRN11.write(0x00, 0x01)

# ipvideo_edge.write(0x14, n_row)
# ipvideo_edge.write(0x1C, n_col)
start = time()
ipvideo_edge.write(0x00, 0x01)
ipDMAIn.sendchannel.transfer(inBuffer0)
ipDMAOut.recvchannel.transfer(outBuffer0)
ipDMAIn.sendchannel.wait()
ipDMAOut.recvchannel.wait()
end = time()
```

Notice that if the input image is 3 channel, we have to pad a dummy channel, since the IP reads a 32-bit data, which consists of dummy bits and RGB.

Encountered bugs

- The IP's IO port seems to be the same as this design, but the PYNQ host program would stuck at recvchannels.
- Cosim deadlock problem (increase FIFO depth).

Learnt

- The transmission protocal of image and 1D data is not the same.
- Dig into the C library to study the function of each functions (ap_axiu<W,1,1,1>, hls::AXIVideo2Mat, hls::Mat, ...).
- Debugging by comparing ILA waveform and cosim waveform
- More understanding on PYNQ module (MMIO, Xlnk, Overlay, ...)
- Using the OpenCV-HLS library to develop a design.