Real-Time Video Edge Detection on an FPGA Using a Sobel Filter

Chad Lucas
School of Electrical Engineering
and Computer Science
University of North Dakota
Grand Forks, USA
chad.lucas.ndus@edu.com

***Abstract*— Edge detection is an important operation in image processing. It is used in many fields, for example the defense sector for target tracking, the automotive industry for lane detection and in the health sector for medical imaging. Many of these systems are designed using software-based approaches. This project presents a hardware methodology using an FPGA based edge detection video pipeline. This implementation provides lower latency and faster processing due to the hardware's ability to provide parallel processing.**

Keywords - FPGA-ZYBO-z7, Edge Detection, Vivado, Vitis, Sobel Filter, AXI4-Stream

I. Introduction

Edge detection is an important application in image and video processing. It helps identify boundaries between objects and has numerous applications in various fields, such as medical imaging, automotive lane detection and computer vision for object detection. Software based approaches implement edge detection filters in sequential order, it must wait for an operation to finish before it begins the next process, instruction-by-instruction. Using a Field Programable Gate Array (FPGA) based hardware design, the performance over its software counterparts cannot be matched. FPGA architecture allows multiple operations to execute in parallel, which makes it an ideal choice for real-time image processing applications that require high throughput and low latency.

This project will utilize a Zybo z7-20 FPGA to implement a real-time Sobel edge detection video pipeline. A Test Pattern Generator (TPG) IP with a custom Sobel filter written in Verilog is used to create a synthetic video stream which is sent to HDMI for display. At a later phase we will replace the TPG with a MIPI-CSI-2 IP to process real time video data from a Digilent PCAM 5 camera module. This project will focus on the preliminary phase of the pipeline with the TPG and custom Sobel filter.

II. Theoritical

An image can be represented as a 2-D function f(x,y), where x and y correspond to mapped coordinates, and its amplitude of f at any coordinate pair (x,y) is called its intensity [1]. A pixel is a sample of the image intensity mapped to an integer value and an image is made up of these two-dimensional arrays [2]. For example, a Pixel [0,0] would correspond to the top left corner of the image. The position of x is mapped to the horizontal direction, and the y value is mapped to the vertical direction as shown in Figure 1.1.

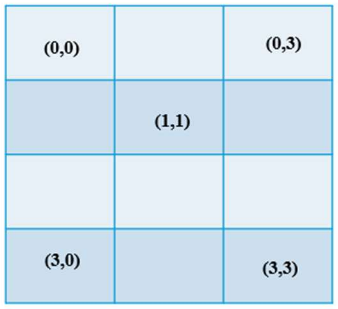


Figure 1.1 Showing the mapped pixel coordinates.

In a typical digital image, each pixel intensity is stored as red, green and blue (R, G, B) components. The components can be considered a vector R(x,y), G(x,y), B(x,y) where each channel encodes the intensity of that color at that location. Edge detection filters such as the Sobel filter rely on converting these sets of values into one single grey scale intensity value, where 0 = black and 255 = white. The larger value corresponds to a brighter pixel. Figure 1.2 shows a rgb image vs a greyscale.

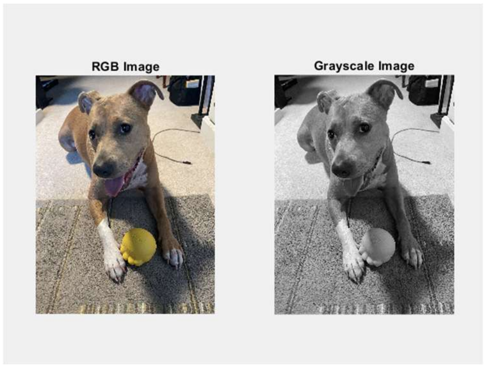


Figure1.2 RGB vs Greyscale

A. *Sobel Filter*

Edges are important sharp local changes in image intensity values [1]. The high-pass Sobel filter focuses on these high frequency gradient transitions to locate edges. The Sobel filter uses two 3x3 convolution masks to approximate the image gradient in both the horizontal and vertical directions. Each mask is placed at the center of the pixel and multiplied with the filters coefficients. Figure 1.3 shows the horizontal mask and its coefficient being multiplied by the corresponding pixel values.

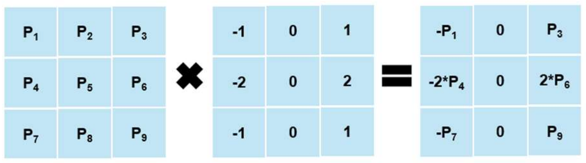


Figure 1.2: Sobel convolution operation.

The Sobel kernel G_x (horizontal) and G_y (vertical):

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 1.3 Sobel filter kernels

After the convolution operation the sliding window value is further processed, and the edge magnitude is summed. We use max of the absolute value instead of the square root to provide faster processing and reduce latency.

- $G_x = sum\ of\ kernel * pixel$
- $G_y = sum\ of\ kernel * pixel$
- $|G| \approx \max(|G_x|, |G_y|)$

The ideal gradient magnitude is given by $\sqrt{G_x + G_y}$, however this is expensive to implement directly in hardware and so the max absolute value is more efficient.

III. System Architecture

A. *Hardware Platform*

The edge detection video pipeline is implemented on a Xilinx Zynq-7000 Soc-Based FPGA development board (Zybo Z7-20) as seen in Figure 1.4. The board combined both a dual-core ARM processing system (PS) with programmable logic (PL). The PS is used for system configuration and control. For example, setting up the test pattern parameters or the HDMI format sizing. The PL implements the logic behind the video datapath. It is responsible for controlling the Sobel Filter core as well as the AXI4-Stream to Video Out IPs.

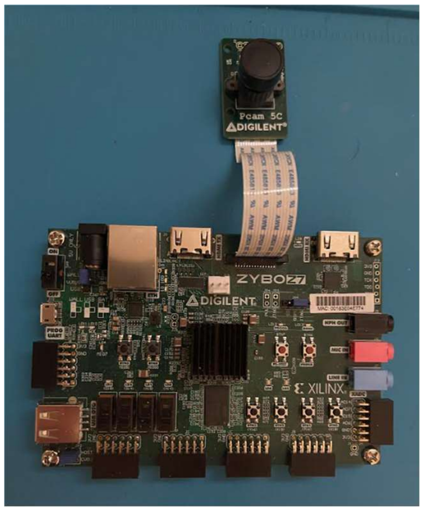


Figure 1.4 Zybo z7-20 w/ Pcam5 camera module

B. AXI4-Stream Protocol

The AXI4-Stream communication protocol is the main driving factor for our data. It provides a reliable and robust packet delivery process, allowing pixels to be processed each clock cycle. The protocol relies on a handshake between the TREADY and TVALID signal to both be high at the same time. If TREADY is true and TVALID false, the protocol will wait until it sees both set too high to stream data through as shown in Figure 1.5.

In this project pixel data is represented by a 24-bit RBG value with 8-bits per channel color. The signal is carried on the TDATA bus and waits for the handshake, also to verify the start of the video frame (SOF), the TUSER is also set too high. When the last pixel in the frame is reached the TLAST signal goes high to indicate end of line (EOL). This creates a safe and reliable datapath for high throughput for video pipelining.

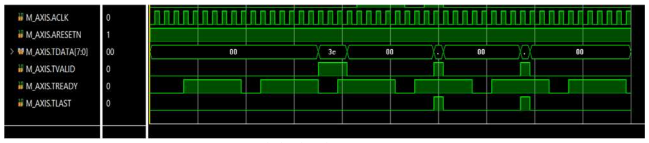


Figure 1.5 AXI4-Stream Handshake between TVALID & TREADY, showing TDATA is allowed to passthrough as both are held high.

C. *Video Pipeline*

Our video pipeline is transmitted by our AXI4-Stream protocol which allows for the video signal to be transformed and processed by the Test Pattern Generator, custom Sobel filter, Video Timing Controller (VTC), AXI4-Stream to Video

Out and then outputted by our RGB2DVI (HDMI) IPs as shown in the block diagram in Figure 1.6.

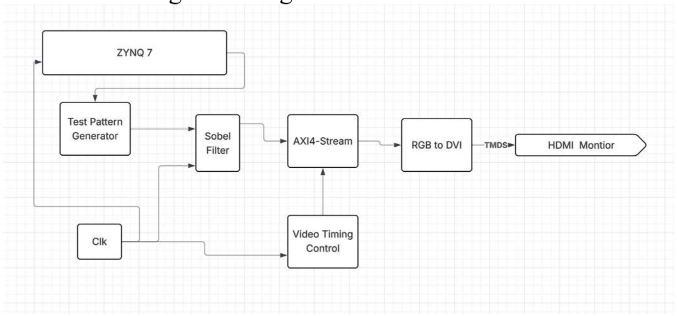


Figure 1.6 Video Pipeline system architecture.

Our TPG will act as a stand in for our OV54640 camera sensor. It allows for the creation of a continuous stream of frames of several different pattern choices. We will use the color bar for our project as is shown in Figure 1.6. This will allow us to test and verify the Sobel edge-detection pipeline in real time before connecting an external sensor.

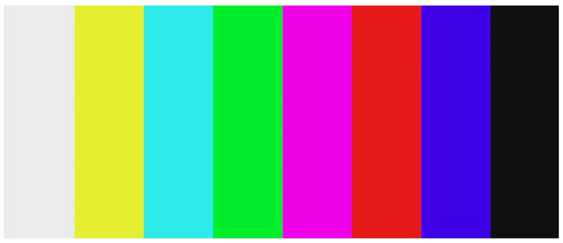


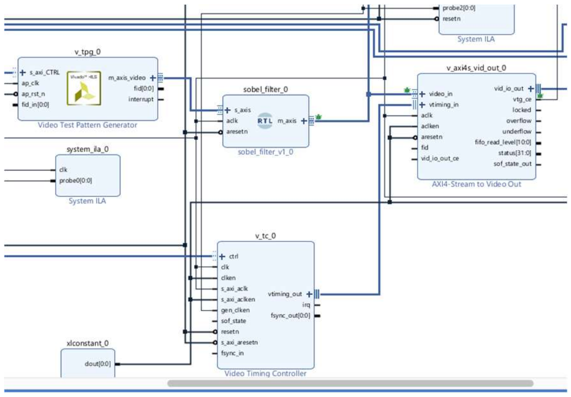
Figure 1.6 TPG color bar pattern.

To allow for pixel data to propagate through the data bus the timing verification is crucial. The design uses two main clocks, a 100 MHz clock that drives the AXI4-Stream and the 74 MHz pixel clock that is used by the HDMI output. The VTC aligns both clock domains so both the horizontal and vertical signals sync up and frames are displayed properly. If timing is not matched it can cause irregularities, distortion or even a blank screen.

D. *Sobel Filter Design*

The custom Sobel Filter was designed in Verilog and integrated inside our Vivado video pipeline as shown in Figure 1.7. To ensure proper functionality several important top-level requirements were needed. The filter needs to use the AXI4-Stream master-slave input/outputs - needed to be able to maintain correct frame and line timing and output one pixel per clock.

The internal workings of the filter include 2-line buffers which hold the previous rows of pixels which allow us to create a 3x3 window to perform the convolution operations on each clock cycle. This parallelism design provides a streamlined low latency option that is far superior to sequential software systems.



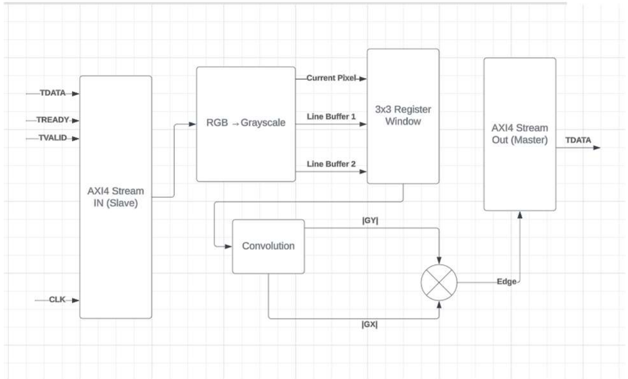


Figure 1.7 Custom Sobel Filter in Vivado video pipeline & Sobel architecture.

IV. Simulation and Verification

The custom Sobel filter IP block was simulated in Vivado. The testbench generated a small 24-bit test image. The AXI4-Stream properties were tested to make sure TVALID and TREADY, when high would allow pixels to stream on the TDATA bus. We also tested TUSER to activate the SOF and TLAST EOL, to verify proper pixel transmission. Waveform analysis confirmed that the pixel data propagated through the Sobel pipeline, demonstrating both edge computation and AXI protocol compliance as shown in Figure 1. 9. First, we will run a simple convolution test to make sure the G_x and G_y are calculated accurately. We will use a simple 3x3 pixel array:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 255 & 255 \\ 0 & 255 & 255 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 510 \\ 0 & 0 & 255 \end{bmatrix}$$

$$= 765$$

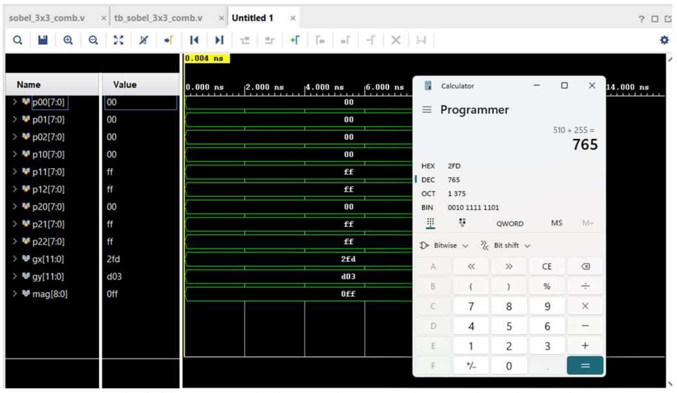


Figure 1.8 Sobel filter convolution and summation testbench results.

As we see in Figure 1.8 the convolution successfully calculates the G_x and the G_y values.

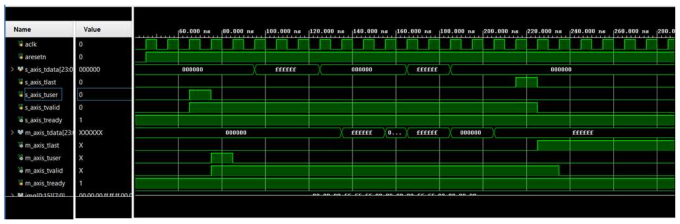


Figure 1.9 AXI4-Stream pixel transmission.

V. FPGA Implementation

A. Vivado Block Design

The video processing pipeline was designed in Vivado using the IP Integrator as seen from the block design in Figure 2.0. The design uses several AXI4-Stream IPs, a custom-built Sobel Filter, Test Pattern Generator as well as several clocking and reset blocks.

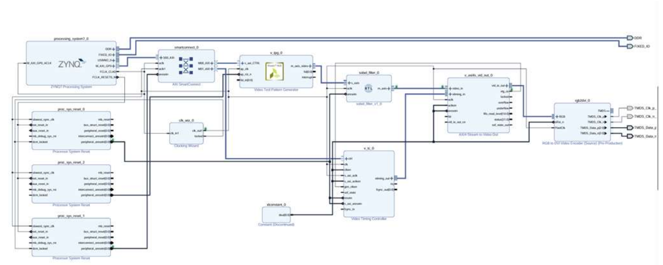


Figure 2.0 Edge detector pipeline implemented in Vivado.

The driver in our system is the main ZYNQ 7 (PS), which drives the internal clock and reset operations. The processing system allows us to sync up our AXI clocks which run at a different frequency than our video clock. It also handles the I^2C communication between the camera module for our future work. The PS allows us to run our bare-metal custom C code to initialize and configure our video pipeline components such as our TPG and timing controller.

The Sobel filter output is not entirely ready to be streamed to our HDMI monitor; it is only a continuous series of bits at this point. The AXI4-Stream to Video out subsystem provides the heavy lifting for signal transformation. It takes in the TDATA, the AXI handshake as well as TUSER and TLAST and generates the important pixel data such as HSYNC and VSYNC – which are parameters for the frame sizing – for accurate HDMI transmission.

The signal is finally processed by the RGB to DVI Encoder (Source). This IP block is the interface between the FPGA's internal video pipeline and the HDMI displays. To make the signal suitable for display it needs to be converted to a TMDS-encoded serial data. The TMDS is comprised of 4 channels, 3 differential signal data channels and 1 clock channel. Each channel can carry a 10-bit stream [4].

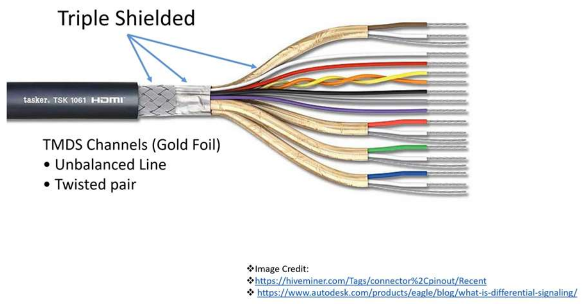

Triple Shielded
tasker. TSK 1061 HDMI
TMDS Channels (Gold Foil)
• Unbalanced Line
• Twisted pair
❖Image Credit:
❖https://hiveminer.com/Tags/connector%2Cpinout/Recent
❖ https://www.autodesk.com/products/eagle/blog/what-is-differential-signaling/

Figure 2.1 HDMI cable showing the TMDS channels.

The RBG to DVI, takes the 24-bit RGB, Hsync, Vysnc and pixel clock data and encodes the information on the final leg of our video pipeline, onto the 10-bit TMDS data channels for monitor output.

B. Results

Figure 2.2-3 Shows the proper edge detection from our source image created by our TPG. We see clear vertical white lines at the edges of where the color bars transition in the frame. This resolution was at 720p at a frame rate of 60Hz.

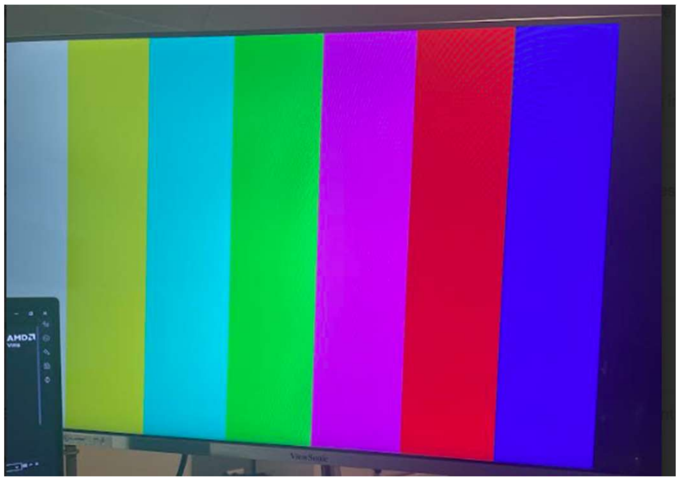


Figure 2.2 Source signal from our TPG at 720p, 60Hz fps.

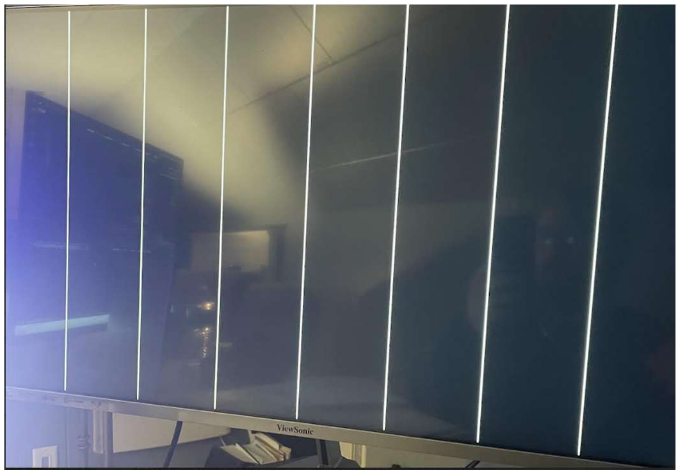
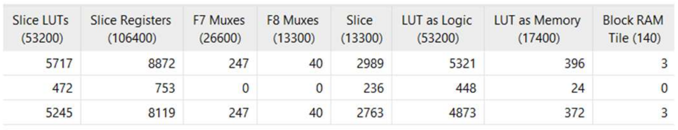


Figure 2.3 Sobel edge detection filter showing white vertical lines where the source signal colors transition.

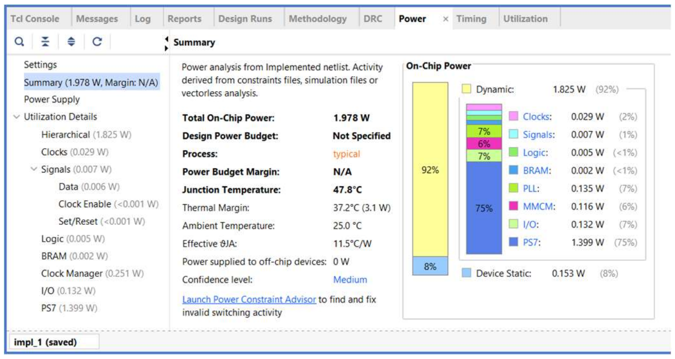
C. *Resources Utilization and Timing*

From our utilization report [Figure 2.xxx], we see our hardware accelerated video processing system uses only a small fraction of available programmable logic resources. 20% of all available LUT's were used as well as only around 8% of all slice registers and 2% of block ram. This leaves a vast number of available resources for our future camera extensions.

The power summary estimates show total on-chip power of around 1.98W, with most of the consumption made by the internal processing system (PS7). This is expected since the PS handles the ARM cores, clocking operations and all interconnections.



Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Slice (13300)	LUT as Logic (53200)	LUT as Memory (17400)	Block RAM Tile (140)
5717	8872	247	40	2989	5321	396	3
472	753	0	0	236	448	24	0
5245	8119	247	40	2763	4873	372	3



Settings
Summary (1.978 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (1.825 W)
Clocks (0.029 W)
Signals (0.007 W)
Data (0.006 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (0.005 W)
BRAM (0.002 W)
Clock Manager (0.251 W)
I/O (0.132 W)
PS7 (1.399 W)

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

Total On-Chip Power:	**1.978 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**47.8°C**
Thermal Margin:	37.2°C (3.1 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	1.825 W	(92%)
🟪	Clocks:	0.029 W	(2%)
🟦	Signals:	0.007 W	(1%)
🟩	Logic:	0.005 W	(<1%)
🟦	BRAM:	0.002 W	(<1%)
🟩	PLL:	0.135 W	(7%)
🟥	MMCM:	0.116 W	(6%)
🟩	I/O:	0.132 W	(7%)
🟦	PS7:	1.399 W	(75%)
🟦	Device Static:	0.153 W	(8%)

Figure 2.4 Resource and power utilization.

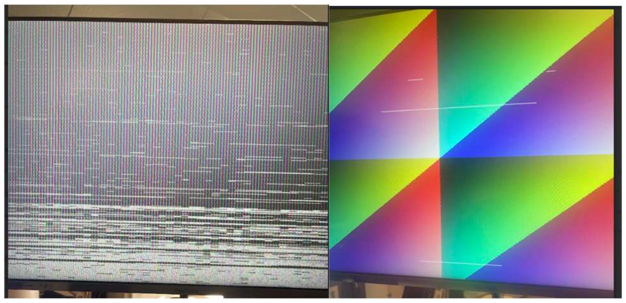
VI. Future Work

A. *MIPI-CSI-2 Camera Integration*

The Test Pattern Generator IP block was replaced with a Digilent PCAM5 camera module which uses the Omnivision OV5640 image sensor. This configuration replaces the static test pattern with a real-time image sensor, the full block diagram can be seen in Appendix A. Several more communication protocols as well as timing configuration added complexity to our initial video pipeline. The camera module uses the I^2C protocols to initialize and power up the camera as well as the MIPI CSI-2 receiver core that controls the pixel throughput. The Zynq processing system (PS), was used to setup and control and edit registers through a bare-metal C application in Vitis. Compared to the TPG, the new camera configuration is significantly more complex, as correct operation depends on the sensor's registers being properly and sequentially and clock timing to capture and lock onto MIPI-Rx data.

B. *Challenges*

Several configuration changes were explored to try and create a stable video streaming signal, from adjusting the Pcam 5 module through the MIPI-CSI-2 receiver block. First adjusting the frame rate as well as the resolution, then configuring the proper initializing registers. Another design was implemented that added the pixel buffer onto another on-chip memory block. However, none of these configurations were successful in displaying a proper frame on the HDMI output as shown in Figure 2.xxx.



Although a stable video pipeline wasn't established in this initial phase, the experiments provide insightful and helpful data. The sensitivity of the MIPI timing protocols along with the strict sequential sensor register configuration provides a good starting point for future implementation fixes.

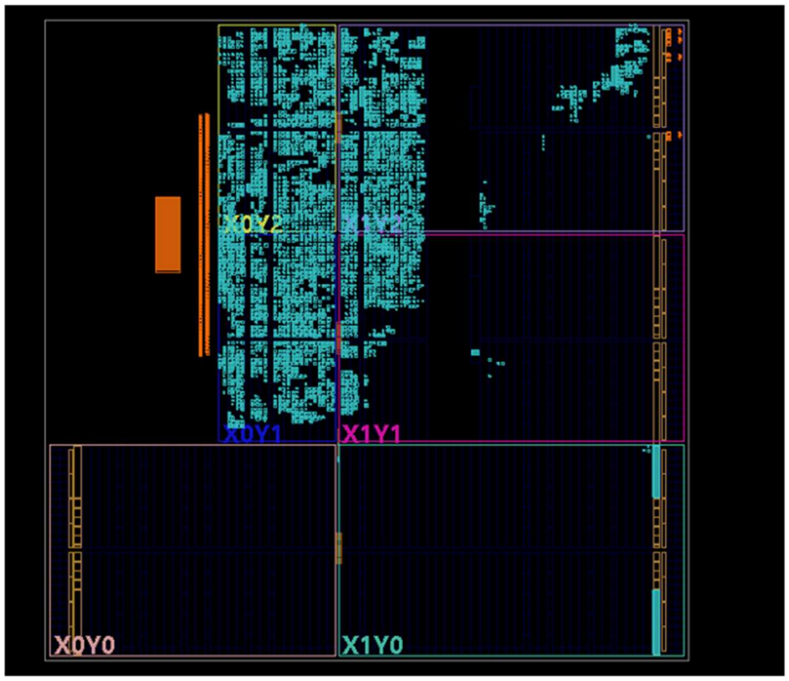
VII. Conclusion

This project implemented and designed a real-time Sobel edge detection video pipeline on a Digilent Zybo -Z7-20 FPGA. This edge detection hardware accelerator creates speed and low latency that cannot be matched by its software equivalent. We created a parallel processing Sobel Filter that takes in a 24-bit RGB pixel value and processes it to find the images edges every clock cycle. A static test pattern was generated which was controlled by the systems timing controller and then sent through the data bus using AXI-Stream protocols.

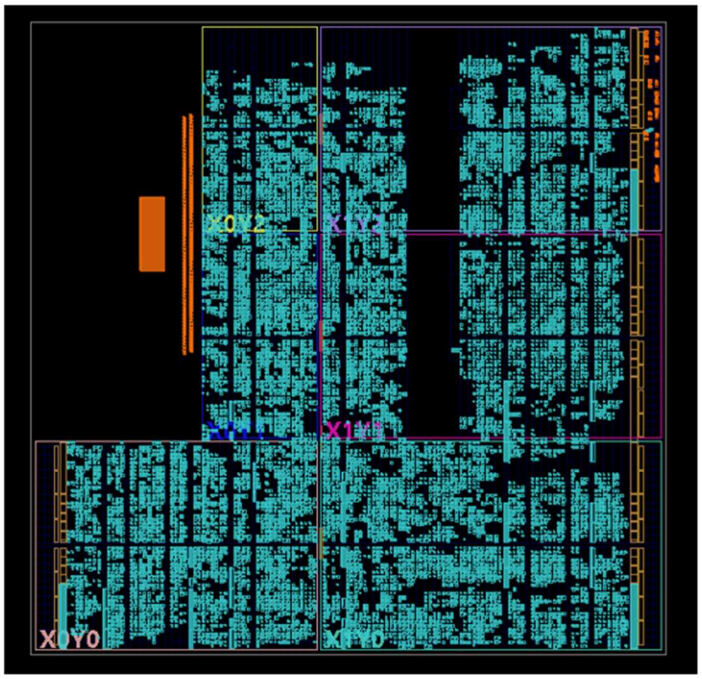
Preliminary work demonstrated a replacement to the Test Pattern Generator with a Pcam5 camera, however configuration complexity made the integration prone to frame rate issues as well as timing conflicts.

The implementation and results demonstrated the hardware designed Sobel filter, with real-time image processing could be built using minimum resources and provide high performance and low latency. This level of performance is critical in high-speed autonomous vehicles, precise robotic automation and embedded vision applications.

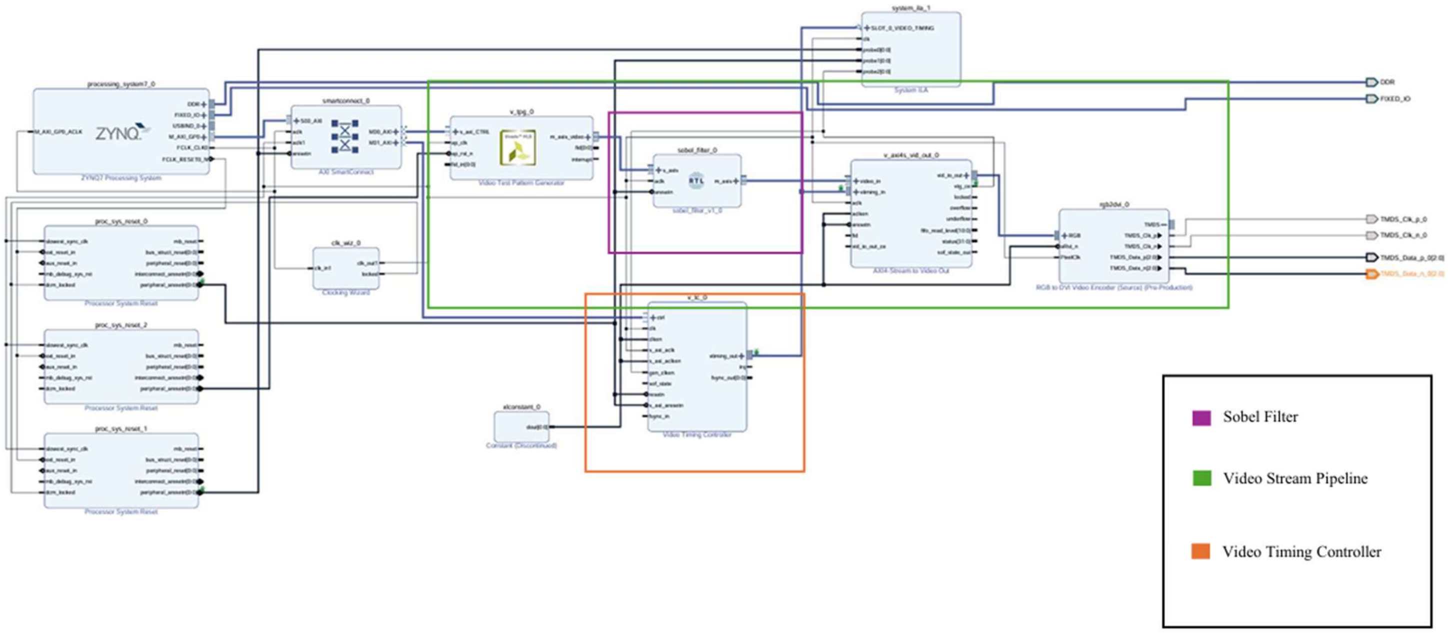
Appendix A: Floor plan comparison and block diagram

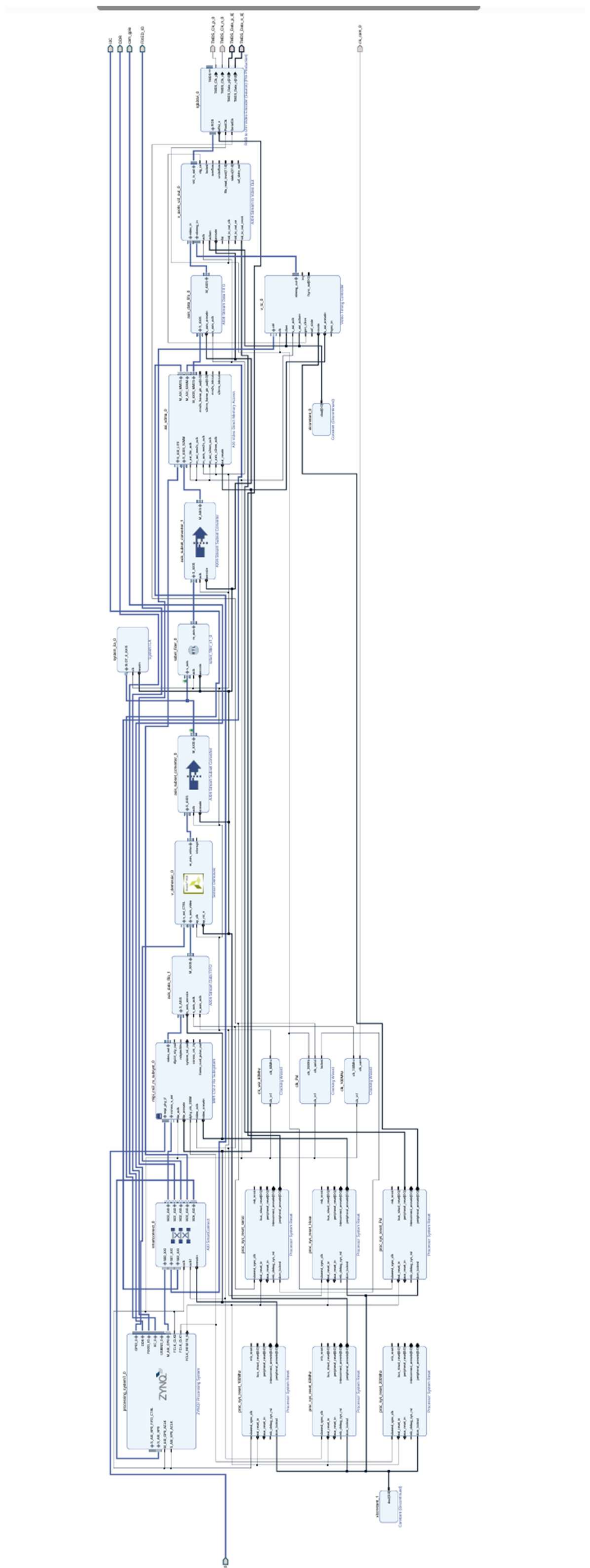


Floor plan of TPG



Floor plan of camera extension





References

[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. Boston, MA, USA: Pearson, 2018, p. 13.

[2] R. Jain, R. Kasturi, and B. G. Schunck, Machine Vision. New York, NY, USA: McGraw-Hill, 1995.

[3] AMD (Xilinx), "AXI4-Stream to Video Out LogiCORE IP Product Guide," PG044, v4.0, Oct. 2017. [Online]. Available: https://docs.amd.com/r/en-US/pg044_v_axis_vid_out

[4] "Transition-minimized differential signaling," Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Transition-minimized_differential_signaling (accessed Nov. 25, 2025).