Mickey McMahon

EPHX 601

Professor A. Bean

12 May 2023

Frog-Eye Demonstration Final Report

Abstract Summary

This project attempts to demonstrate the utility of accelerated image processing through the functional implementation of dynamic vision sensing on an FPGA board. Our goal is to explore the improvements in timing and efficiency of data transmission by asynchronous imagers over that of conventional shutter cameras. The project includes a four-by-four pixel array composed of infrared detecting circuits implemented on a custom board. By illuminating a pixel with an infrared laser, a user is able to draw different shapes. Using an analog-to-digital converter, image data is transferred to the FPGA's internal memory, which is accessible by a Python Overlay API. The overlay includes algorithms that reconstruct the 'image' of the laser's motion, as well as predict its path.

Background and Motivation

A standard camera captures data using a shutter, whereby images are taken at a constant framerate. While useful for still image and video applications, shuttering proves less effective at capturing high-speed motion. Because the captured frame is an integration of data over the period of the shutter, an object moving appreciably faster than the framerate of the imager will appear blurred [1]. Dynamic vision sensing addresses this issue by transmitting data on a per-pixel basis only when the imager detects a local change in brightness. In other words, pixels are read independently and asynchronous, similar to the operation of the eye's retina [2]. Changes in a pixel's brightness are detected by feeding the output of a photodetector to a comparator circuit, in which the illumination of a pixel beyond a threshold value causes the circuit to emit a signal. Because the sensor avoids reading unnecessary data, it exhibits higher temporal sampling and lower latency, enabling the image to be amalgamated in a more continuous manner. This allows for more accurate object tracking [3]. Dynamic vision sensing is used in a variety of embedded systems that require high-speed motion detection, including robotics, autonomous vehicles, drones, and visual sensor networks, among others [4]. This project investigates the advantages of dynamic vision sensing through the construction of a four-by-four pixel array that captures and transmits data from an infrared laser to a PYNQ-Z2 FPGA board at a constant framerate, emulating a conventional shutter camera. Although unable to be implemented due to time constraints, the physical design of the board includes space for several comparator circuits which can be used for a contrasting readout method intended to mimic a dynamic vision sensor. In addition, this project is motivated by the use of machine learning algorithms to predict motion. Robotic systems, for example, use image data to make statistical generalizations predicting the motion of objects around them to more safely interact with their environments. This project serves as an example of the utility of path prediction through the design of an algorithm that reconstructs and predicts the path of the laser using linear regressions.

Description of the Project

The pixel array is implemented using photodiodes capable of detecting infrared light. A user holding a laser with a compatible spectral range is able to shine the laser at the array to draw lines and shapes. Upon illumination, a photodiode emits a signal that is fed to an amplification circuit. The amplified signal is then fed to a peak detector and track-and-hold circuit responsible for storing the signal's maximum voltage in a charged capacitor. The voltage across these capacitors represent the intensity of light detected by each pixel over a brief period of time. Next, the output of the 16 track-and-hold circuits are fed to a 16-input analog multiplexer. The multiplexer is responsible for selecting which pixel to read. The multiplexed data is fed to an 8-bit ADC and a voltage translator that steps down the voltage of the digital signal before being read into the PMOD ports of the FPGA. Concurrently, the FPGA is responsible for writing out multiple control signals to parts of the pixel circuit. The control signals include a clock and chip-select for the ADC, bits for the select input of the multiplexer, and bits to 'reset' the track-and-hold circuits. For the purposes of this paper, the process of reading all 16 pixels will be termed an 'event'. Once an event has concluded, the FPGA uses the reset signals to flip switches that flush the track-and-hold capacitors of their voltage, preparing the circuit for another event. This readout method emulates that of a standard imager, as the FPGA captures data from all 16 pixels at a constant frame rate irrespective of the local intensity of each pixel. The internal configuration of the FPGA is specified and implemented in Vivado and managed by a Python API. The Python API uses a Jupyter notebook as the interface for managing data transfer between the board's programmable logic and an external computer. For this project, data read into the FPGA is stored on the board's internal memory. With the overlay, a user is able to execute an algorithm that uses the stored data to reconstruct the image and predict its path of motion.

System-Level Requirements

- The user must be able to illuminate a 4x4 array of pixels using a hand-held laser
- The system must be capable of captures images of the laser's motion at a constant framerate, subject to the following specifications:
    - Each 'pixel circuit' should comprise of the following:
        - Photodiodes (infrared photodetectors)
        - An amplifier
        - A peak detector to locate the maximum light intensity over a window of time
        - A track-and-hold circuit to store the voltage corresponding to the maximum light intensity in capacitor
        - A switch parallel to each capacitor that allows the capacitors to be flushed after an 'event'
    - Aside from each pixel, the circuit must include the following components
        - A 16-input multiplexer able to select from each of the 16 pixels
        - An ADC capable of converting the analog signal from a pixel into a digital one able to be read into the FPGA
        - A voltage translator that steps up/down voltages between the 3.3V FPGA pins and the 5V I/O pins of other components
- The project board must include connectable and/or switchable power supplies for the laser and circuit board
- The system should be configured such that the PMOD ports on the FPGA are able to:
    - Write out control signals for the multiplexer, ADC, and switches
    - Read in data from the ADC

- The user should be able to start and stop reading/writing from the FPGA using a Python overlay
- After starting and stopping reading/writing, the user should be have access to the following data displays:
    - A graph that reconstructs an image of the laser's path
    - A graph that predicts the laser's path of motion
    - A numeric coordinate representing a prediction of the laser's next most likely location

Component Requirements

- Circuit board
    - Compact size, able to house relevant components without occupying too much space
    - Must include all aforementioned components as well an expansion header for potential window comparators and voltage testing
- Laser
    - 850 nm-1100 nm wavelength
    - Collaminated signal
    - Compatible spectral range with photodiodes
    - Hand-held
- Photodiodes
    - 850 nm-1100 nm wavelength
    - Active area on the scale of ~mm
    - Constant output current (~mA)
- Multiplexer
    - 16-inputs
    - Short active delay (~ns)
- ADC
    - 8-bit output
    - Short active delay (~ns)
- Voltage Translator

    - Must step voltages up/down from 3.3V and 5V
    - Short active delay (~ns)
- FPGA
    - Must be capable of reading in/writing out data from PMOD ports
    - Must be able to store data on internal memory
- Python overlay
    - Must allow the user to execute code that reads/writes data at a constant framerate
    - Must be able to reconstruct a laser's path of motion over a period of time
    - Must be able to find the most probable line of motion from multiple frames

Project Design and Build

Figure 1 displays the high-level relationship between relevant project components in a block diagram. A laser emits an infrared signal detected by the 4x4 pixel board. The pixel board transfers analog signals from each signal to the ADC and subsequently the FPGA. The FPGA sends out control signals to different components on the circuit board, as well as stores the read data for processing and display.

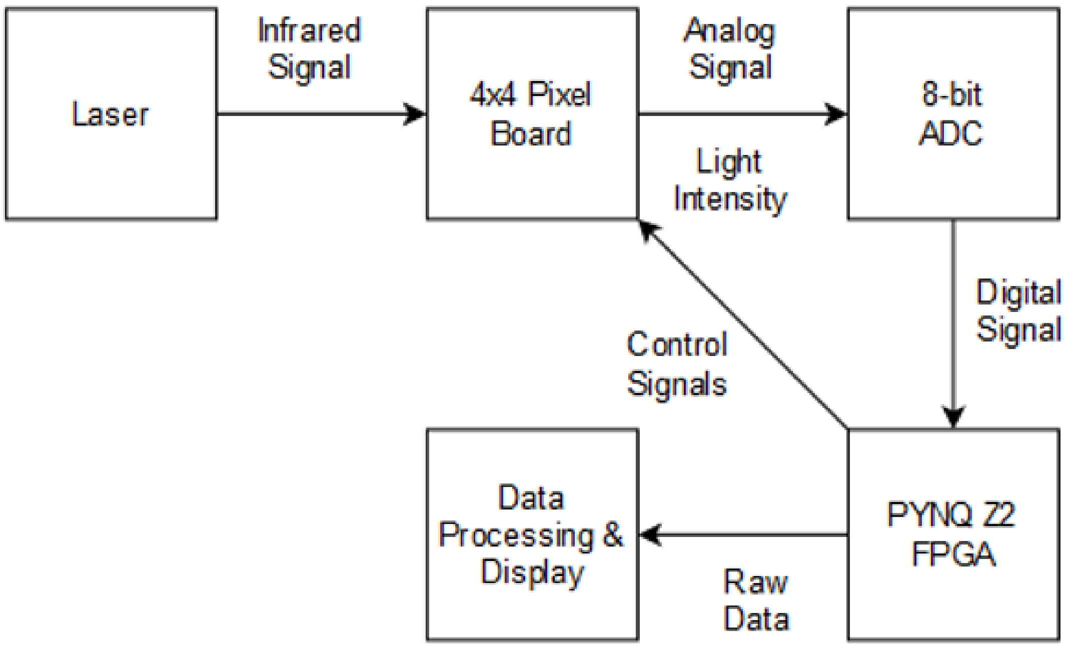


Figure 1: Project Block Diagram

Figure 2 illustrates a high-level representation of our circuit diagram with building blocks not explicitly considered in Figure 1. Each ‘pixel circuit’, which refers to the circuit components that comprise each pixel, is composed of a photodiode, amplifier, and peak detector/track-and-hold circuit. The output of each ‘pixel circuit’ is fed to the 16-input multiplexer, which controls the data selected to be converted by the ADC. In addition, the voltage translator responsible for translating the voltages of the control signals and read data to and from the FPGA are shown.

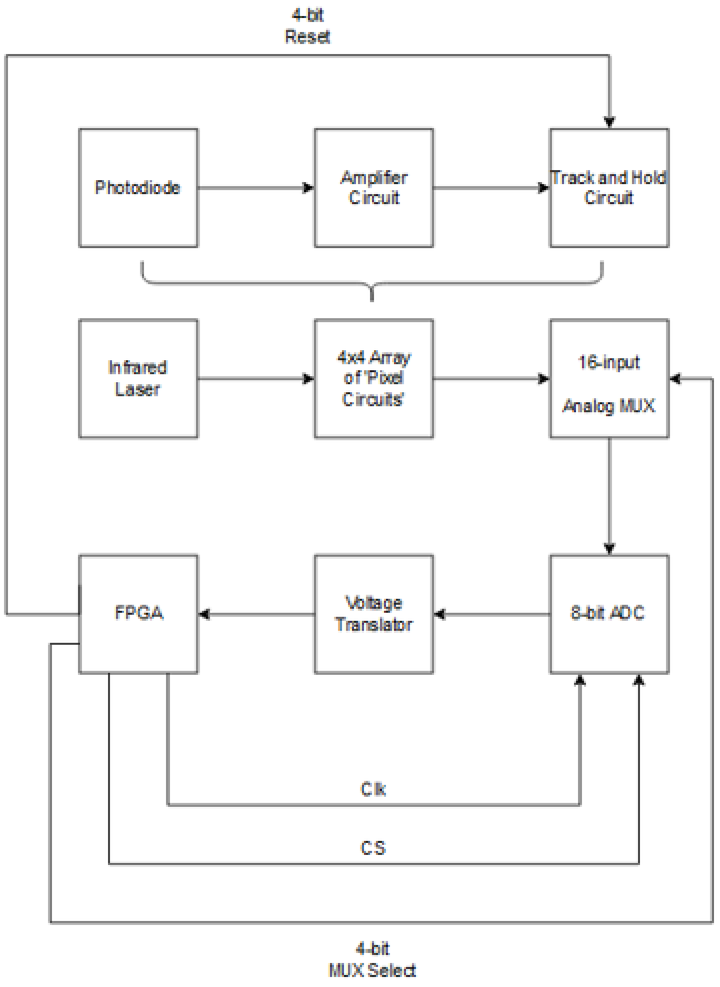


Figure 2: Circuit Block Diagram

Figures 3-5 are snippets of the circuit board schematic. Pictured in Figure 3 is the ‘beginning’ of the 16 pixel circuits, each with a photodiode as an input to an amplifier circuit. Figure 4 is a schematic for the peak detector and track-and-hold circuits that store the waveform from the amplifier in a capacitor to be inputted to the multiplexer. Notice that parallel with each capacitor is a switch with a reset input. This allows the FPGA to clear each capacitor after being read. The schematic in Figure 5 shows the connections between the multiplexer, ADC, voltage translator, and PMOD ports on the FPGA, as well as the control signals and read inputs for the relevant ports. Notice the FPGA is connected to the multiplexer with 4 bits for the select signal. It is also connected to the ADC with a clock and chip-select signal, and each column of pixels (Figure 4) is connected to one of 4 reset bits.

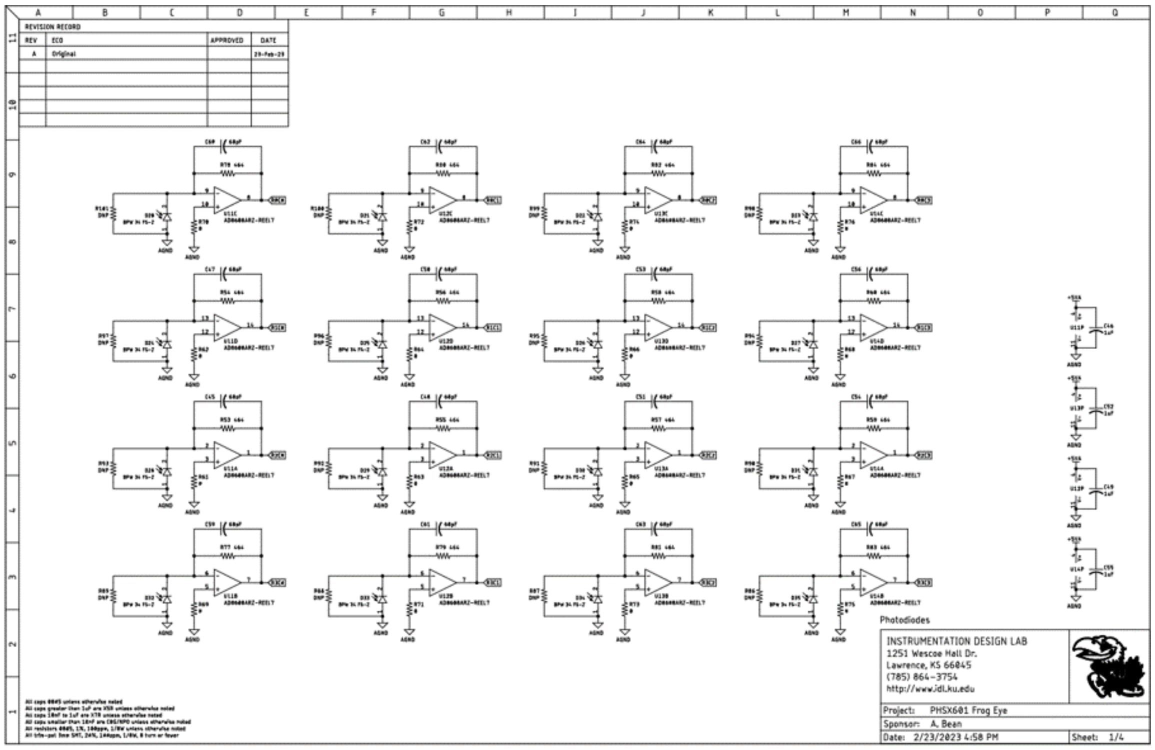


Figure 3: Circuit Schematic #1

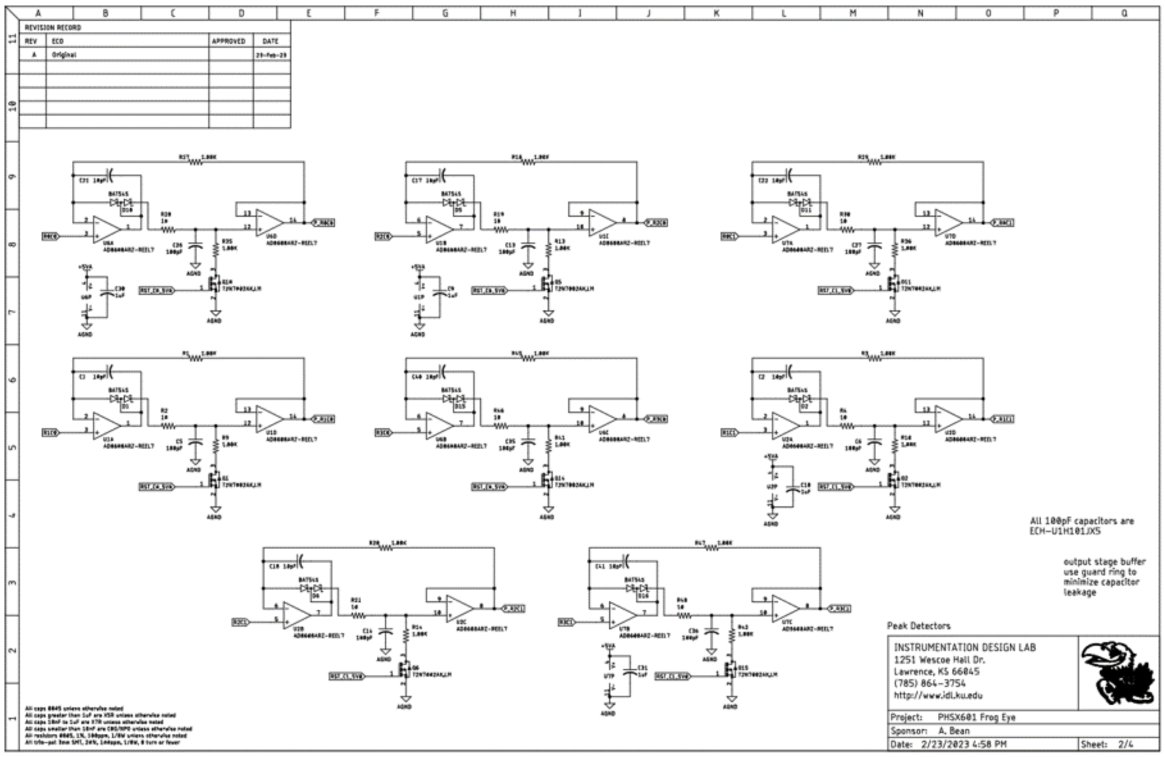

REVISION RECORD
Peak Detectors
All 100pF capacitors are ECH-U1H101JX5
output stage buffer use guard ring to minimize capacitor leakage
INSTRUMENTATION DESIGN LAB
1251 Wescoe Hall Dr.
Lawrence, KS 66045
(785) 864-3754
http://www.idl.ku.edu
Project: PHSX601 Frog Eye
Sponsor: A. Bean
Date: 2/23/2023 4:58 PM
Sheet: 2/4

Figure 4: Circuit Schematic #2

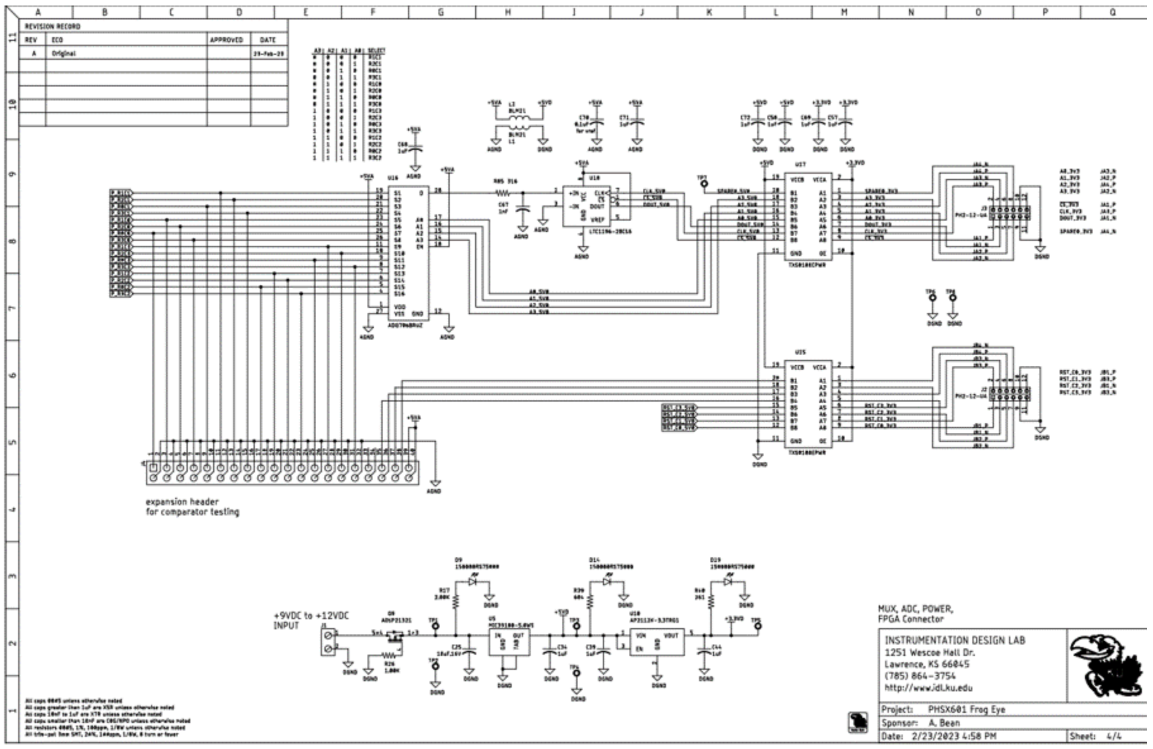


Figure 5: Circuit Schematic #3

Figure 6 is the complete 3D model of the circuit schematic. The 4x4 pixel array is in the bottom left-corner.

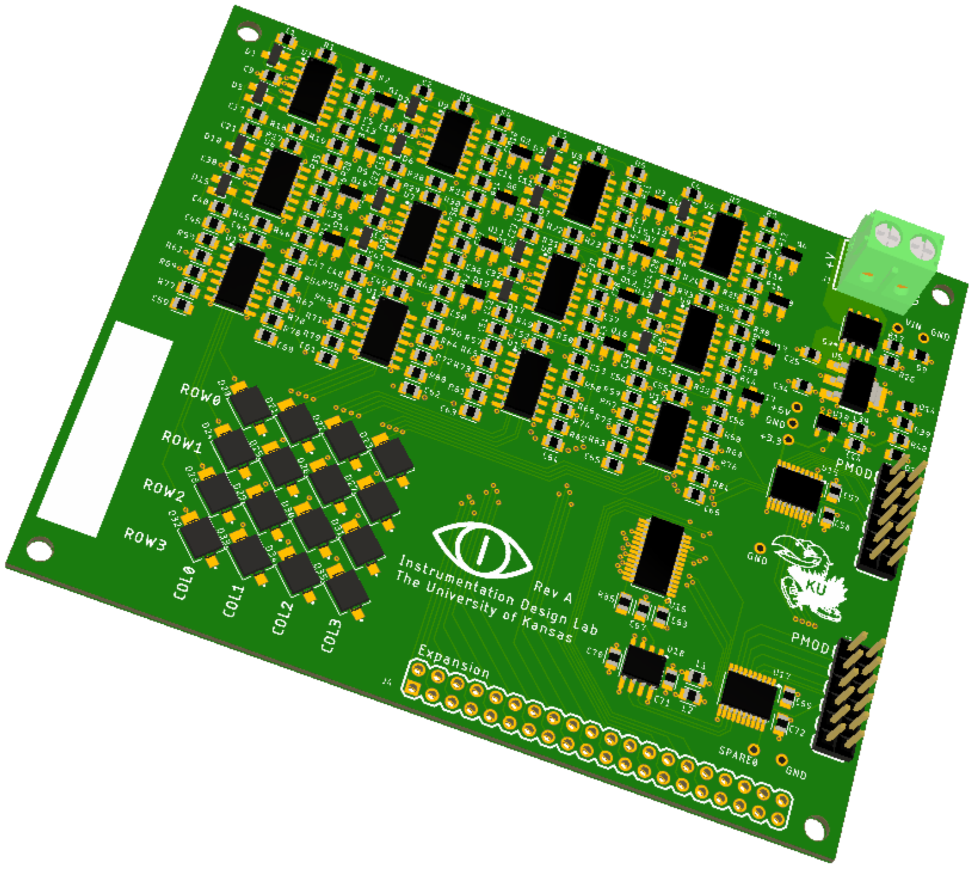


Figure 6: Circuit Board 3D Model

Figure 7 is a screen-capture of the hardware FPGA configuration as designed in Vivado. The block diagram is a high-level representation of the relationship between the ZYNQ processor/programmable logic and the board's external interfaces. Exporting this block diagram and uploading it to the Jupyter notebook is what allows the user to control the board's configuration as well as manage the board's data transfer/storage using the Python overlay. The block diagram specifies the following external interfaces:

- miso, mosi, and mosi_copy - used for bidirectional serial communication between the board and the ADC, essentially our 'read' signals.

- Btns_4bits, leds_4bits, sws_2bits - GPIO interfaces for controlling the FPGA's buttons, LED's, and switches. The buttons are used for starting and stopping reading and writing to/from the FPGA, while the LED's and switches are purely for demo purposes.
- control - GPIO peripheral, a 12-bit binary 'word' used to send out the mux-select and reset signals.
    - Bits 3:0 - mux-select signal.
    - Bits 8:5 - 4-bit reset signal, controls the column to reset. For instance, setting bit 5 'hi' resets C0.
    - Bit 4 and Bits 11:9 - empty.
- sclk - SPI output, 3.3 MHz clock for the ADC.
- csbar - SPI output, chip-select for the ADC.

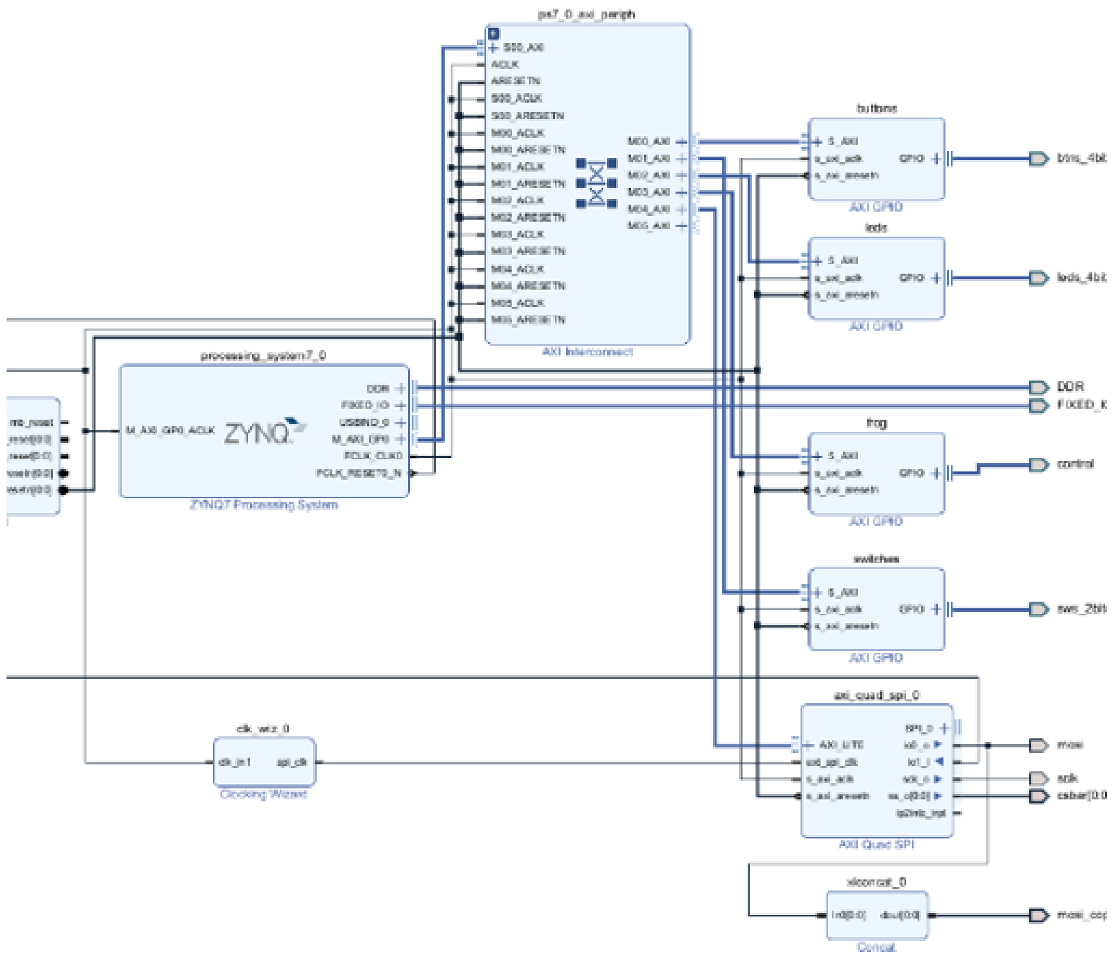
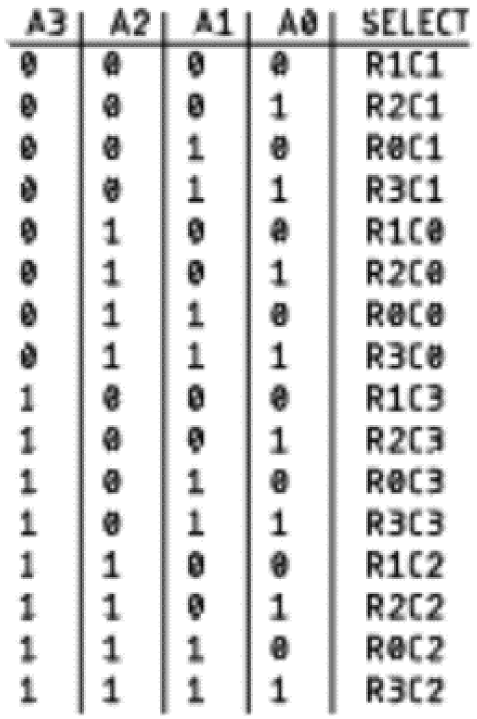


Figure 7: Vivado Block Diagram

Figures 8 and 9 illustrate the layout of pixels on the circuit board, their labels, and the control signals used to select each pixel from the multiplexer. For instance, to read the pixel in the top-left corner (R0C0), the FPGA writes out the control signal “0x006”.

	C0	C1	C2	C3
R0	0x006	0x002	0x00E	0x00A
R1	0x004	0x000	0x00C	0x008
R2	0x005	0x001	0x00D	0x009
R3	0x007	0x003	0x00F	0x00B

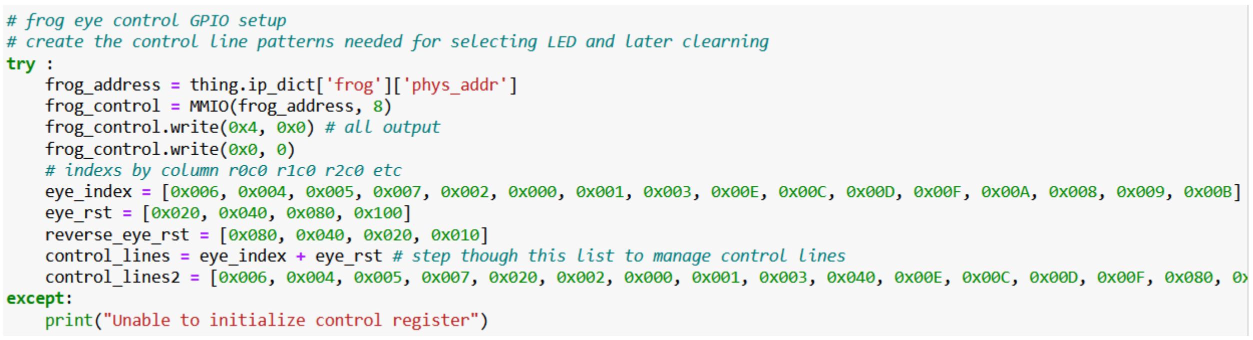
Figure 8: Pixel Array Labels



A3	A2	A1	A0	SELECT
0	0	0	0	R1C1
0	0	0	1	R2C1
0	0	1	0	R0C1
0	0	1	1	R3C1
0	1	0	0	R1C0
0	1	0	1	R2C0
0	1	1	0	R0C0
0	1	1	1	R3C0
1	0	0	0	R1C3
1	0	0	1	R2C3
1	0	1	0	R0C3
1	0	1	1	R3C3
1	1	0	0	R1C2
1	1	0	1	R2C2
1	1	1	0	R0C2
1	1	1	1	R3C2

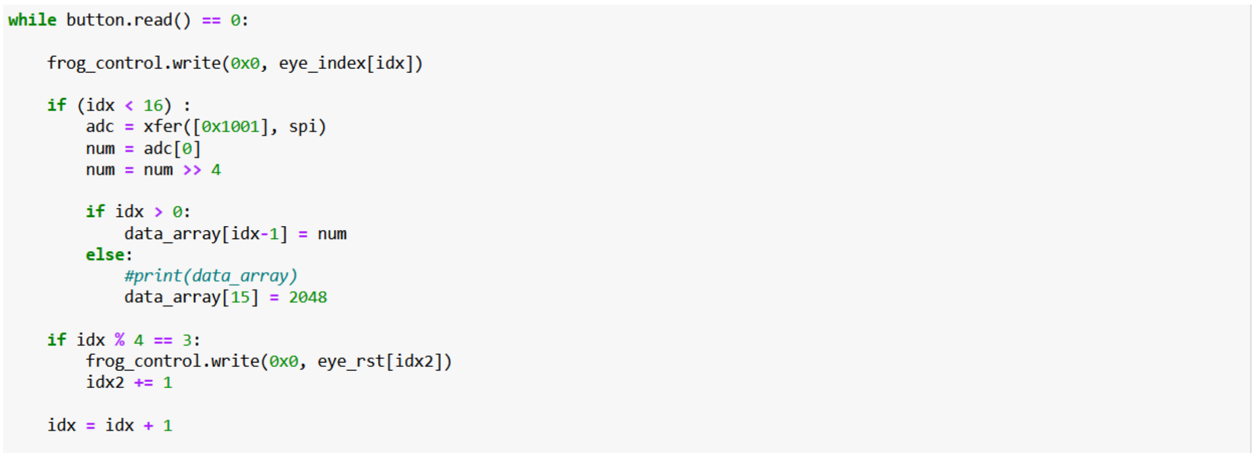
Figure 9: Pixels and Their Associated Select Signals

Figures 10-11 are snippets of code from the Jupyter notebook that demonstrate read and write functionality. In Figure 10, note the array labeled “eye_index” and “eye_reset”. These are arrays that store the values of the 12-bit control signal in the sequence we intend to step through the pixel array and the reset signals. Our project is designed such that we read C0 from top to bottom, then C1, C2, and C3. In other words, the FPGA reads data from R0C0 -> R3C0, then from R0C1 -> R3C1, and so on until the last pixel is read. This is performed using the loop in Figure 11 that steps through the eye_index array as an argument for the ‘write’ function. The write function is also responsible for setting the chip-select for the ADC low. In other words, data is only read from the ADC after the multiplexer switches its input. After reading each column, the control signals for resetting the columns are written out. Reading data from the ADC is performed by the xfer function, also found in Figure 11.



```
# frog eye control GPIO setup
# create the control line patterns needed for selecting LED and later clearning
try :
    frog_address = thing.ip_dict['frog']['phys_addr']
    frog_control = MMIO(frog_address, 8)
    frog_control.write(0x4, 0x0) # all output
    frog_control.write(0x0, 0)
    # indexs by column r0c0 r1c0 r2c0 etc
    eye_index = [0x006, 0x004, 0x005, 0x007, 0x002, 0x000, 0x001, 0x003, 0x00E, 0x00C, 0x00D, 0x00F, 0x00A, 0x008, 0x009, 0x00B]
    eye_rst = [0x020, 0x040, 0x080, 0x100]
    reverse_eye_rst = [0x080, 0x040, 0x020, 0x010]
    control_lines = eye_index + eye_rst # step though this list to manage control lines
    control_lines2 = [0x006, 0x004, 0x005, 0x007, 0x020, 0x002, 0x000, 0x001, 0x003, 0x040, 0x00E, 0x00C, 0x00D, 0x00F, 0x080, 0x
except:
    print("Unable to initialize control register")
```

Figure 10: Control Signal GPIO Setup, Jupyter Notebook



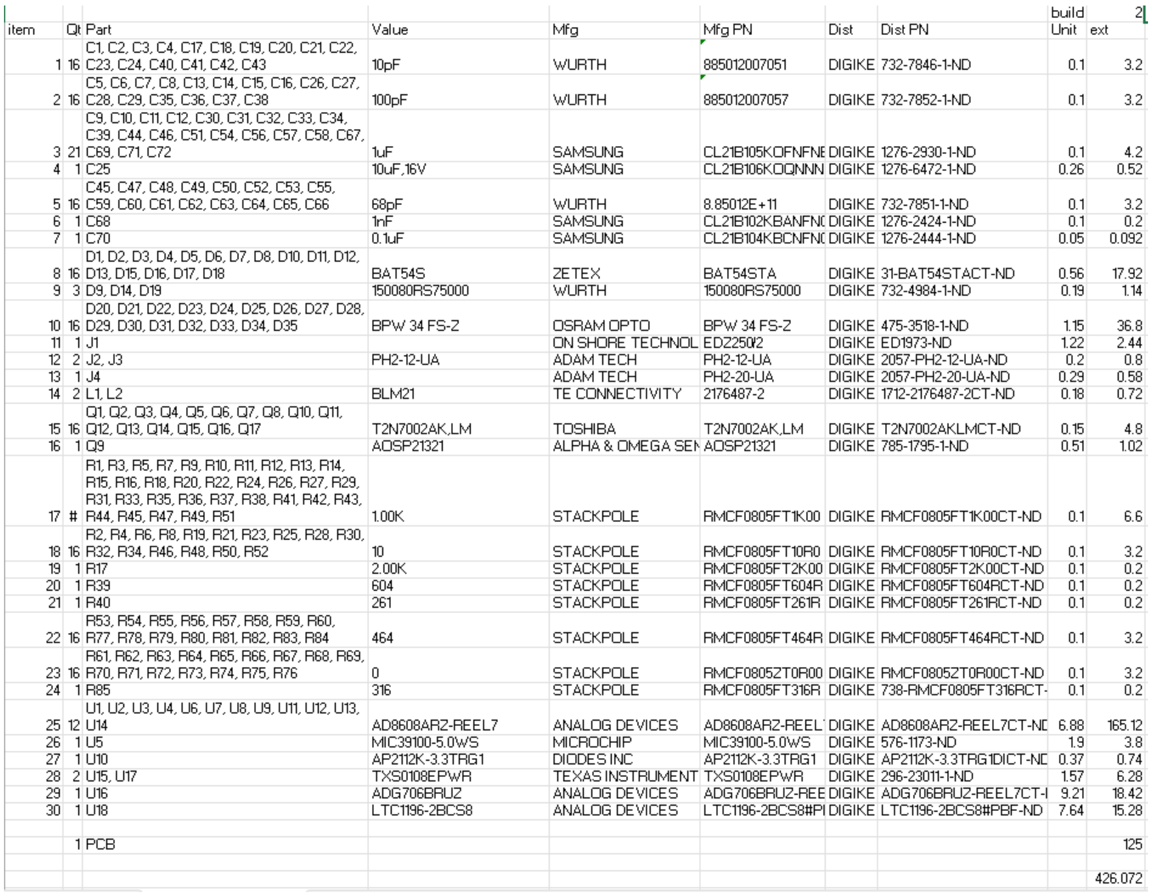
```
while button.read() == 0:

    frog_control.write(0x0, eye_index[idx])

    if (idx < 16) :
        adc = xfer([0x1001], spi)
        num = adc[0]
        num = num >> 4

        if idx > 0:
            data_array[idx-1] = num
        else:
            #print(data_array)
            data_array[15] = 2048

    if idx % 4 == 3:
        frog_control.write(0x0, eye_rst[idx2])
        idx2 += 1

    idx = idx + 1
```

Figure 11: Control Signal Write and ADC Read, Jupyter Notebook

Reconstructing the image of the laser's motion and predicting its path are done by storing each 'event' or 'frame' of data in an array. The likeliest location of the laser for one frame is found using a weighted average of the intensities, stored in the aforementioned array, across the pixels in 2D space. A prediction for the laser's path of motion is performed via a linear regression of the most recent 100 frames of data collected. Shown in Figure 11, the loop for reading and writing signals is ended upon the press of a button on the FPGA board. Once a button has been pressed, two graphs are shown for the user: the image of the laser's motion, and the frames used for predicting the laser's path along with its linear regression.

Parts List

See Figure 12 for a list of parts ordered and their prices. Total cost for construction of the circuit board was $426.072. In addition, a mounting board, a 9-12V and 5V power source for the circuit and battery, respectively, a wave generator, and the PYNQ Z-2 board were supplied by the physics department.



								build	2
item	Qt	Part	Value	Mfg	Mfg PN	Dist	Dist PN	Unit	ext
1	16	C1, C2, C3, C4, C17, C18, C19, C20, C21, C22, C23, C24, C40, C41, C42, C43	10pF	WURTH	885012007051	DIGIKE	732-7846-1-ND	0.1	3.2
2	16	C5, C6, C7, C8, C13, C14, C15, C16, C26, C27, C28, C29, C35, C36, C37, C38	100pF	WURTH	885012007057	DIGIKE	732-7852-1-ND	0.1	3.2
3	21	C9, C10, C11, C12, C30, C31, C32, C33, C34, C39, C44, C46, C51, C54, C56, C57, C58, C67, C69, C71, C72	1uF	SAMSUNG	CL21B105KOFNFNE	DIGIKE	1276-2930-1-ND	0.1	4.2
4	1	C25	10uF,16V	SAMSUNG	CL21B106KOQNNN	DIGIKE	1276-6472-1-ND	0.26	0.52
5	16	C45, C47, C48, C49, C50, C52, C53, C55, C59, C60, C61, C62, C63, C64, C65, C66	68pF	WURTH	8.85012E+11	DIGIKE	732-7851-1-ND	0.1	3.2
6	1	C68	1nF	SAMSUNG	CL21B102KBANFN(	DIGIKE	1276-2424-1-ND	0.1	0.2
7	1	C70	0.1uF	SAMSUNG	CL21B104KBCNFN(	DIGIKE	1276-2444-1-ND	0.05	0.092
8	16	D1, D2, D3, D4, D5, D6, D7, D8, D10, D11, D12, D13, D15, D16, D17, D18	BAT54S	ZETEX	BAT54STA	DIGIKE	31-BAT54STACT-ND	0.56	17.92
9	3	D9, D14, D19	150080RS75000	WURTH	150080RS75000	DIGIKE	732-4984-1-ND	0.19	1.14
10	16	D20, D21, D22, D23, D24, D25, D26, D27, D28, D29, D30, D31, D32, D33, D34, D35	BPW 34 FS-Z	OSRAM OPTO	BPW 34 FS-Z	DIGIKE	475-3518-1-ND	1.15	36.8
11	1	J1		ON SHORE TECHNOL	EDZ250/2	DIGIKE	ED1973-ND	1.22	2.44
12	2	J2, J3	PH2-12-UA	ADAM TECH	PH2-12-UA	DIGIKE	2057-PH2-12-UA-ND	0.2	0.8
13	1	J4		ADAM TECH	PH2-20-UA	DIGIKE	2057-PH2-20-UA-ND	0.29	0.58
14	2	L1, L2	BLM21	TE CONNECTIVITY	2176487-2	DIGIKE	1712-2176487-2CT-ND	0.18	0.72
15	16	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q10, Q11, Q12, Q13, Q14, Q15, Q16, Q17	T2N7002AK,LM	TOSHIBA	T2N7002AK,LM	DIGIKE	T2N7002AKLMCT-ND	0.15	4.8
16	1	Q9	AOSP21321	ALPHA & OMEGA SEN	AOSP21321	DIGIKE	785-1795-1-ND	0.51	1.02
17	#	R1, R3, R5, R7, R9, R10, R11, R12, R13, R14, R15, R16, R18, R20, R22, R24, R26, R27, R29, R31, R33, R35, R36, R37, R38, R41, R42, R43, R44, R45, R47, R49, R51	1.00K	STACKPOLE	RMCF0805FT1K00	DIGIKE	RMCF0805FT1K00CT-ND	0.1	6.6
18	16	R2, R4, R6, R8, R19, R21, R23, R25, R28, R30, R32, R34, R46, R48, R50, R52	10	STACKPOLE	RMCF0805FT10R0	DIGIKE	RMCF0805FT10R0CT-ND	0.1	3.2
19	1	R17	2.00K	STACKPOLE	RMCF0805FT2K00	DIGIKE	RMCF0805FT2K00CT-ND	0.1	0.2
20	1	R39	604	STACKPOLE	RMCF0805FT604R	DIGIKE	RMCF0805FT604RCT-ND	0.1	0.2
21	1	R40	261	STACKPOLE	RMCF0805FT261R	DIGIKE	RMCF0805FT261RCT-ND	0.1	0.2
22	16	R53, R54, R55, R56, R57, R58, R59, R60, R77, R78, R79, R80, R81, R82, R83, R84	464	STACKPOLE	RMCF0805FT464R	DIGIKE	RMCF0805FT464RCT-ND	0.1	3.2
23	16	R61, R62, R63, R64, R65, R66, R67, R68, R69, R70, R71, R72, R73, R74, R75, R76	0	STACKPOLE	RMCF0805ZT0R00	DIGIKE	RMCF0805ZT0R00CT-ND	0.1	3.2
24	1	R85	316	STACKPOLE	RMCF0805FT316R	DIGIKE	738-RMCF0805FT316RCT-	0.1	0.2
25	12	U1, U2, U3, U4, U6, U7, U8, U9, U11, U12, U13, U14	AD8608ARZ-REEL7	ANALOG DEVICES	AD8608ARZ-REEL	DIGIKE	AD8608ARZ-REEL7CT-NI	6.88	165.12
26	1	U5	MIC39100-5.0WS	MICROCHIP	MIC39100-5.0WS	DIGIKE	576-1173-ND	1.9	3.8
27	1	U10	AP2112K-3.3TRG1	DIODES INC	AP2112K-3.3TRG1	DIGIKE	AP2112K-3.3TRG1DICT-NI	0.37	0.74
28	2	U15, U17	TXS0108EPWR	TEXAS INSTRUMENT	TXS0108EPWR	DIGIKE	296-23011-1-ND	1.57	6.28
29	1	U16	ADG706BRUZ	ANALOG DEVICES	ADG706BRUZ-REE	DIGIKE	ADG706BRUZ-REEL7CT-I	9.21	18.42
30	1	U18	LTC1196-2BCS8	ANALOG DEVICES	LTC1196-2BCS8#PI	DIGIKE	LTC1196-2BCS8#PBF-ND	7.64	15.28
	1	PCB							125
									426.072

Figure 12: Frog Eye Bill of Materials

Simulations

To visualize timings pertinent to the project, two simulations were performed in VHDL in Vivado. Figure 13 is a simulation that examines the delays of each component in the circuit (found in their datasheets) with relevance to a particular stimulus from the FPGA. The first two signals in the diagram are a 12 MHz clock and chip-select signal as 'seen' by the FPGA. The following two signals are those same control signals as seen by the ADC. Due to the delay of the voltage translator, the control signals from the FPGA do not arrive to the ADC until a fraction of a second later. Similarly, the delay of the voltage translator causes a delay of a ns for data from the ADC to be read into the FPGA. The reason this diagram is necessary is because it was important for us to examine whether the circuit delays would complicate the times for which we should be writing out control signals from the FPGA. This diagram led us to the conclusion that the circuit delays should not be a problem, since clocking the ADC at its maximum frequency (12 MHz) results in a delay of less than a microsecond, which is magnitudes of order shorter than the time with which we switch the multiplexer (once every ~330us, 3000 Hz).

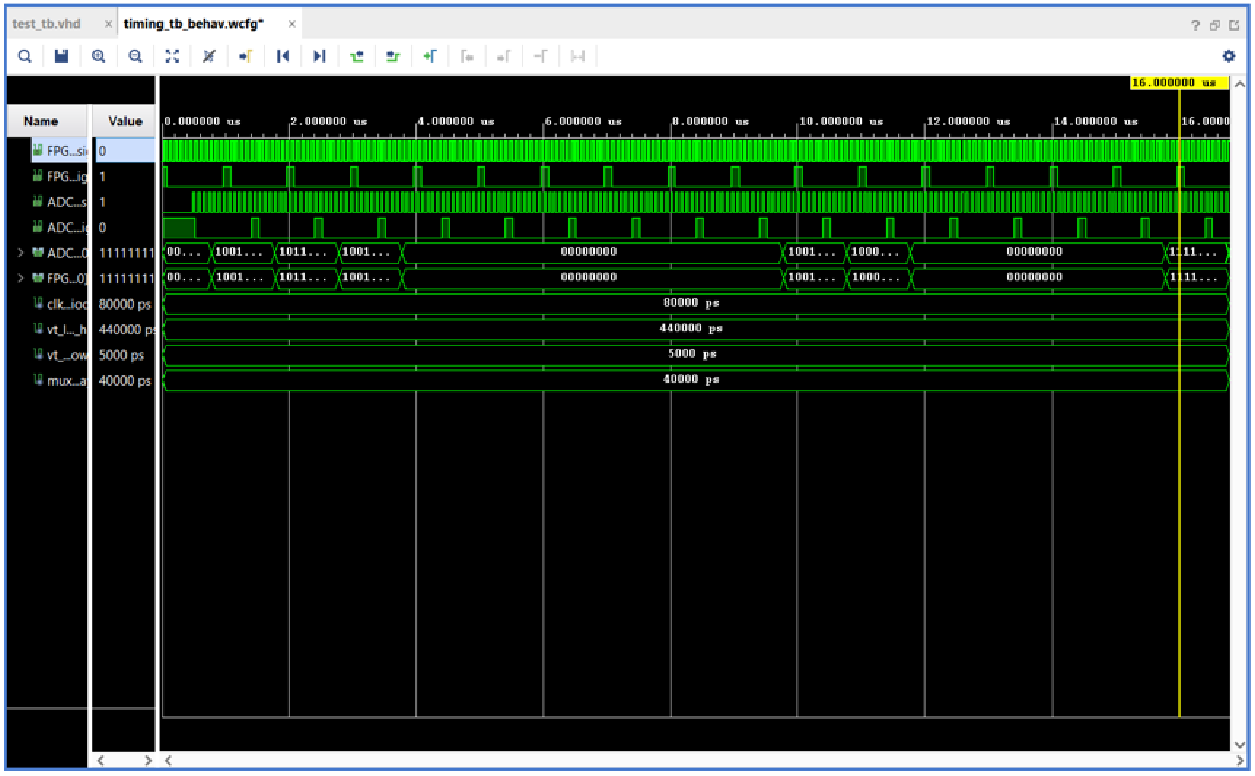


Figure 13: Timing Diagram, Circuit Delays

Figure 14 is a simulation of the control signals using the timing information gathered in the "Read Timing" test under the "Tests" section. The simulation includes the clock, chip-select, resets, and mux-select bits. Notice that the chip-select only changes value upon a change in the mux-select signals. The total time for 1 event to occur is about 6.7ms. This timing information for each signal in the diagram, sans the clock, should have an uncertainty of about ~100us, since the information is based on an average of read-times across many events rather than physical measurements.

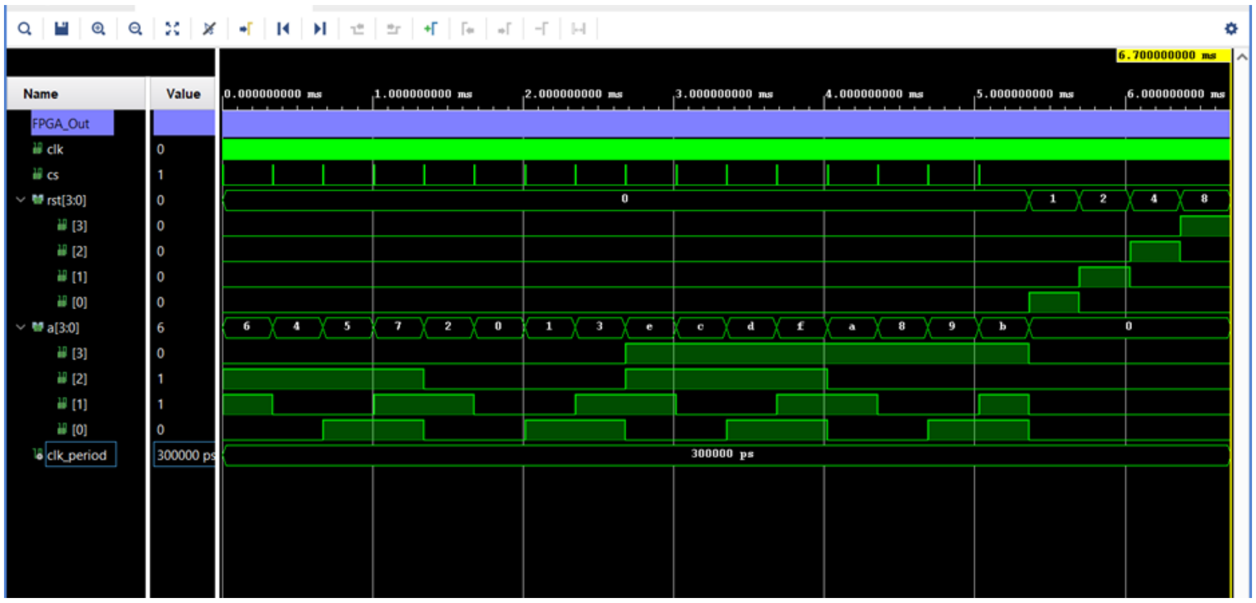


Figure 14: Timing Diagram, Control Signals

Tests

The following tests were performed to verify project functionality.

Control Signals: For the control signals, an oscilloscope was attached to each write port to verify the control signal matched the correct pin and expected waveform. For example, for the ADC clock, the oscilloscope was attached to PMOD-A Pin 2 and examined on the oscilloscope. We looked to ensure that the waveform matched a 3.3 MHz square wave with a 50% duty cycle as specified by the SPI peripheral in Vivado. This was repeated for each write-out pin. In addition, a multimeter was attached to the end of each circuit pin to verify control signals were being sent across the lines and to the write circuit pin. For example, if we only wrote out one reset signal, we expect to see a non-zero voltage at only one pin on the circuit-side. This was also repeated for each write-out signal.

Pixel Circuits: Because the output of each pixel circuit is mapped to a particular location on the expansion header, it was possible to test that a pixel emitted a signal upon illumination by attached multimeter probes to the expansion header. Each pixel was tested and without an infrared laser to verify expected behavior. For example, if a laser were illuminating only the pixel in R0C0, then the holes in the expansion header corresponding to that pixel should be the only pair with a non-zero voltage across them.

ADC: ADC behavior was tested using two methods:

- Writing out control signals to only read from one pixel at a time
- Writing out control signals to constantly read from all 16 pixels and display the most recent 'event'

The former test was necessary to verify that an analog signal from an illuminated signal was being properly selected and outputted from the ADC. For example, if we illuminated the pixel in R0C0, and wrote out the appropriate control signals to only read from R0C0, then we expect to see a digital signal on our Python overlay that was non-zero when illuminated, and zero when non-illuminated. This was repeated for each pixel. The latter test was necessary to verify we could accurately read a complete event. For example, if we illuminated the pixel in R0C0, and wrote out the appropriate control signals to read from all 16 pixels, then we should expect to see the pixel R0C0 read out the largest value (meaning its exposed to the most light), with only the pixels around it experiencing a spillover/offset error. This was also verifiable by illuminating particular pixels over a predetermined path for a short period of time. For example, if we illuminated the pixels in C0 from top-to-bottom, in our excel file, we expect to see that the pixel with the maximum intensity varies from pixel R0C0 to pixel R3C3 sequentially.

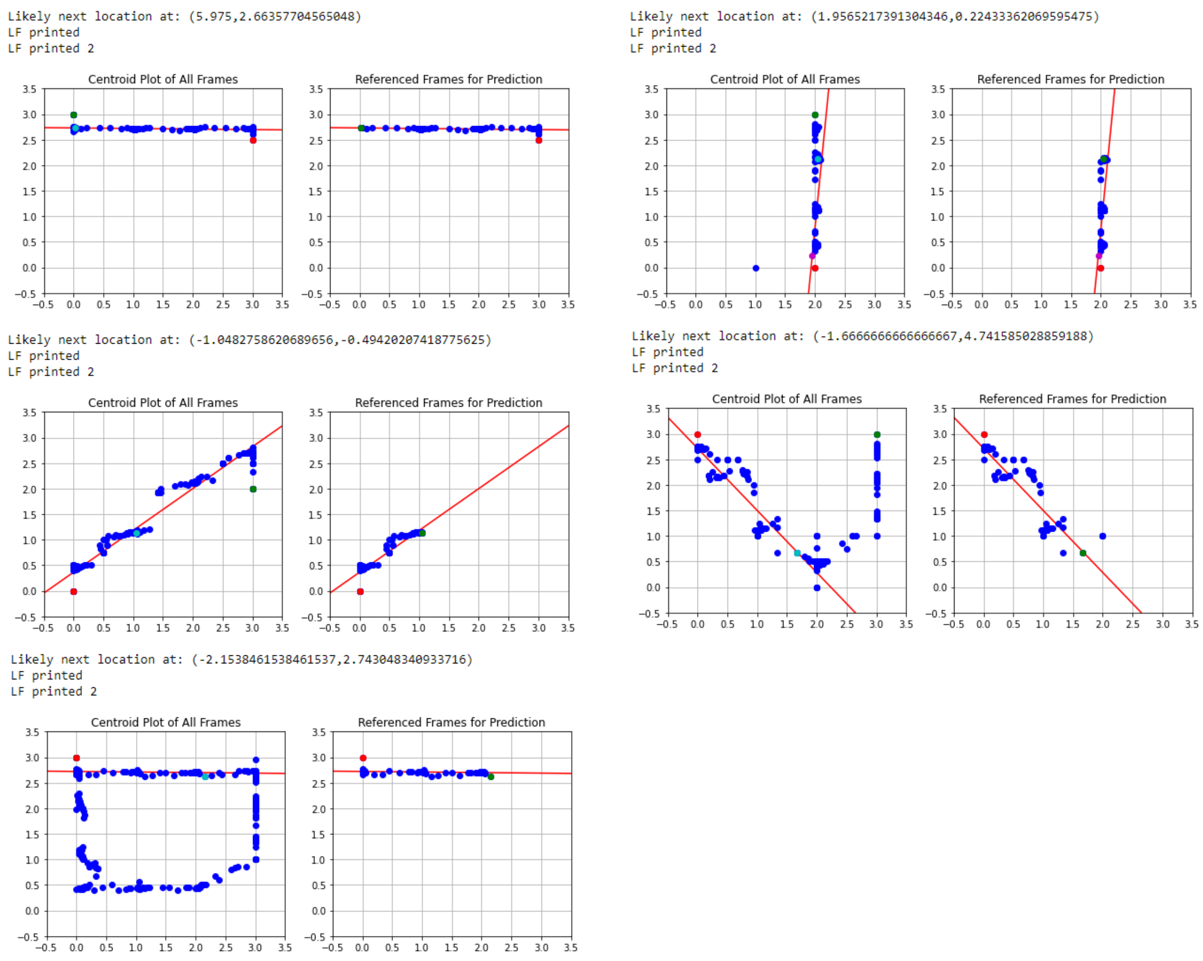
Read Timing: The timing of our read data was tested by inserting time-markers into our Python code and averaging the time-per-read over 200 events. After repeating this process multiple times, we found that a good approximation for the total time to read one event was approximately ~6.7ms, allowing for 150 events to be read per-second.

Read Data and Path Prediction: Image reconstruction and path prediction algorithms were verified by matching expected behavior with the algorithm's results. This was performed by illuminating the pixel array using a predetermined path - a box, line, triangle, or dot, amongst other shapes. For the image reconstruction graph, we expect to see a rough drawing of the same shape visualized on the computer. For the path prediction algorithm, we expect to see a reasonable linear regression using the most recent 100 frames of data. A similar shape is not apparent or the predicted line strays from the data indicates either an issue with the code or reading from the ADC.

Results

Figure 15 shows the results of several trials of the image reconstruction and path prediction algorithms. These are examples of what a user sees after running the project and executing the Jupyter code. For each pair of graphs, the left graph represents the plot of every frame centroid used to reconstruct the image, while the right graph only represents the frames used to plot the linear regression, which is shown in red. R0C0 is located at coordinate (0,3). In the images, the green dot represents the starting point of the laser, or where it was first detected. The red dot represents the last point the laser is detected. The cyan dot is the point where the prediction algorithm begins reading the data. Recall the prediction algorithm only uses the most recent 100 frames of data to predict the next location of the laser. The magenta dot is the predicted location, though in some instances this appears off the board and is not visible. However, the coordinates for the predicted location are available above each graph. In these displays, each pixel is represented as a 2x2 grid with a particular pixel at the center. This is necessary to account for the laser being located between rather than directly above certain pixels.

As shown, with the laser, a user is able to draw particular shapes and patterns, including lines and boxes, which are able to be accurately reconstructed. In each image, the linear regression appears reasonable based on the available data. Uncertainties can likely be accounted for by the steadiness of a user's hand and spillover from the laser. For instance, a user will never be able to draw a perfect line with their hand alone, and the laser's signal is not collimated enough to shine over only one pixel at a time, meaning at any given location, a laser can illuminate several pixels, which changes the location of a frame's centroid.



```
Likely next location at: (5.975,2.66357704565048)
LF printed
LF printed 2
```

```
Likely next location at: (1.9565217391304346,0.22433362069595475)
LF printed
LF printed 2
```

```
Likely next location at: (-1.0482758620689656,-0.49420207418775625)
LF printed
LF printed 2
```

```
Likely next location at: (-1.6666666666666667,4.741585028859188)
LF printed
LF printed 2
```

```
Likely next location at: (-2.1538461538461537,2.743048340933716)
LF printed
LF printed 2
```

Figure 14: Trials and Results of Image Reconstruction and Path Prediction

User Manual

The following components are required to access and run the project:

- An external monitor/desktop
- Project board
- Laser apparatus, including laser power source
- 9-12V board power source (either battery or function generator)
- 5V battery (power source for the laser)

In addition, the following software tools are required.

- Vivado
- Jupyter notebook
- TeraTerm

1. Connect the ethernet cable and USB power source from your desktop to the FPGA.
2. From the GitLab, download and unzip the frogeye_spi.zip file of the Vivado project to your preferred destination.
3. Navigate to the project from your chosen filepath "/frogeye_spi" and open the project in Vivado.
4. Inside the project, go to "File -> Export Block Design", keep the .tcl file named "design_1", and click "Ok". Now, in the main "/frogeye_spi" folder, you should see an updated "design_1.tcl" file.
5. Now, two more files must be located and copy-pasted in the main project folder.
    a. The first .hw file, labeled "design_1.hw", can be found in the "project_1.hw" folder.
    b. The second .bit file, labeled "design_1_wrapper.bit", can be found in the "project_1.srcs" folder, at the location "sources_1/bd/design_1". This file needs to be renamed "design_1.bit" before being pasted.
6. Once copied, power on the FPGA board, and open the program "Tera Term" to verify that serial communication with the FPGA has been established. If the board has not been connected via TeraTerm before, the user must configure the terminal settings and locate the board's emulated serial port. To do this, select "Serial Port Setup" from the Setup menu, and select the highest numbered port available in the drop list for Ports. Set the

speed to “115200” and the data to “8 bit”. In the Terminal Setup menu, ensure that the terminal ID is set to “VT1000” and “Local echo” is unselected. Once settings are configured, restart TeraTerm. It will take about 30 seconds-1 minute for the connection to be established. You will know the connection with the board is complete once the four LED’s on the FPGA begin to blink, and TeraTerm outputs a wall of colorful text.

7. In an empty web browser, search “Pynq006:9090”. This is the web address for the Python API Overlay, which allows users to control data transfer between the board and the computer using Jupyter Notebook. In the event the website asks for a password, the password is always “Xilinx”.
8. Once the notebook is open, navigate to “Files -> frogeye_spi”. It may take a moment for the “Files” page to refresh and fill with projects, so be patient.
9. In the “frog_eye_spi” folder, upload the .bit, .tcl, and .hw files from the main project folder using the “upload” button in the top-right corner. This step is necessary to ensure the firmware configuration is up-to-date with the Vivado block diagram.
10. Once uploaded, upload the python code file “frog_eye.ipynb” from the GitLab, and run the first five cells of code sequentially.
    a. The first block of code imports necessary python libraries.
    b. The second block of code creates variables for the AXI peripherals.
    c. The third block of code initializes and debugs GPIO ports.
    d. The fourth block of code performs necessary memory mapping and declares variables for writing out the FPGA control signals.
    e. The fifth block of code defines functionality for reading and writing to/from the board.
11. Power on the circuit board and the laser by flipping the first and last switch on the project board’s switch board. The switch board should already be connected to active power sources, whether a battery or function generator. If connected to a function generator, ensure that the generator’s output is a 9-12V DC.
12. With the board fully powered, run the final cell of code in the notebook This code begins writing signals from the FPGA and reading in from the circuit’s ADC. The user can now shine the powered infrared laser on the pixel array. Once concluded, the process can be stopped by clicking on the four LED buttons on the FPGA.

a. Below the previously executed block of code, the user should now be able to see a diagram that reconstructs the laser's image and predicts its path.

Recommendations

Due to time limitations, we were unable to implement the faster readout method that selectively reads only pixels that undergo a non-negligible change in illumination. One recommendation for future iterations of this project is the installation of window comparator circuits near the expansion header of our circuit board. The board was deliberately designed to include space for comparator circuits that only emit a signal upon a change in voltage. Using these circuits as the select inputs to a series of multiplexers would ideally permit the functionality described above. Thus, the project board would emulate dynamic vision sensing from a hardware perspective. Contrasting the timing information of events from both our implementation and the dynamic vision sensor would accomplish the goals we set out to accomplish at the start of the semester - namely, investigating the benefits in efficiency and speed that accompany dynamic vision sensors over standard cameras. Another recommendation for future projects is investigating the current drawn by the circuit board. During tests in which the board was powered with 9V batteries, we discovered the source's battery life was drained incredibly quickly. As such, the batteries aren't a reliable power source. We recommend future experiments examine the current drawn by the board and uncover methods for reducing the board's power consumption, or locate more reliable power sources. A final recommendation is to enhance the sophistication of the path prediction algorithm. Currently, the algorithm is only able to use linear regressions for prediction purposes. It is likely possible to incorporate algorithms that detect particular shapes and patterns, such as circles and squares, and is subsequently able to predict the location of the laser more accurately than our algorithm.

References

[1] IEEE Spectrum.
https://spectrum.ieee.org/dynamic-vision-sensors-enable-high-speed-maneuvers-with-robots

[2] Tech Insights.
https://www.techinsights.com/blog/image-sensor/dynamic-vision-sensors-brief-overview-image-sensor-techstream-blog

[3] Hindawi. https://www.hindawi.com/journals/complexity/2021/8973482/

[4] Key applications of the dynamic vision sensor, ResearchGate.
https://www.researchgate.net/figure/Key-application-scenarios-of-the-Dynamic-Vision-Sensor-DVS_fig2_332388971

Relevant Datasheets:

LTC1196 ADC:
https://www.analog.com/media/en/technical-documentation/data-sheets/119698fb.pdf

ADG706 MUX:
https://www.analog.com/media/en/technical-documentation/data-sheets/ADG706_707.pdf

TXS0108EPWR Voltage Translator:
https://www.ti.com/lit/ds/symlink/txs0108e.pdf?ts=1677778008883&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTXS0108E