ECE2500Y MEng Project Report

Camera System for Time-of-Flight 3D Imaging with Custom Coded-Exposure Image Sensor

Savo Bajic

Summary

Report on my work undertaken for Intelligent Sensory Microsystems Laboratory (ISML) lab, focusing on the development of the new T3.2 image sensor host board and other minor projects conducted to aid in the development of the T7 imaging sensor system such as subframe readout. My project began officially January 2023 and ran until September 2023, but the majority of my efforts began with the conclusion of the winter semester in April.

The outcome of my work is detailed and supported with the reasoning for design choices of note. Furthermore, I have identified areas of improvement for the future champions of this work.

Contents

1 Introduction **2**

2 Background and Motivations **3**

2.1 Coded Exposure Imaging 3

2.2 Time of Flight Sensing 4

2.3 T3.2 and T7 Image Sensors 4

3 Circuit Board for T3.2 **5**

3.1 Design Objectives 6

3.2 Schematic Design 7

3.3 Board Design 11

3.4 Validation 14

4 Drivers for T3.2 **16**

4.1 Peripheral Circuitry 16

4.2 T3.2 Interfacing 18

5 Subframe Readout for T7 **21**

5.1 BRAM + USB 3.0 vs. DRAM Hypothesis 21

5.2 Optimization 22

5.3 USB Investigation 22

6 Generative Masking for T7 **24**

7 Conclusion **26**

List of terms **27**

Appendices **28**

Appendix A Work Schedule **28**

Appendix B Layout Renders **29**

Appendix C Recommendations for Future Work **33**

1 Introduction

The Intelligent Sensory Microsystems Laboratory (ISML) from the University of Toronto has iterated upon its Time of Flight (ToF) Image Sensor design with the T3.2 sensor. To verify the sensor's design, characterize, and eventually use it, a custom Printed Circuit Board (PCB) is needed to host the required supporting circuitry for the sensor's operation. In addition to the design of this board to support T3.2 there is a need to prepare the drivers needed to control and eventually pull data from T3.2 and pass it onto the host computer for processing via a Field Programmable Gate Array (FPGA). This was my primary task.

Much of the work in this report is based off the previous efforts for the board and drivers prepared for the T3 sensor preceding T3.2, as well as earlier T3.2 PCB drafts. Modifications were made to accommodate the changes between the two sensors, as well as addressing issues identified in these earlier versions and incorporating improvements in supporting circuitry from related projects like T7.

In addition to the work done to advance the T3.2 project, there was some work undertaken to aid the development of ISML's T7 imaging sensor project. These forays into T7 focused on improving the high-speed performance of the sensor in burst imaging and subframe readout applications.

I completed the majority of the work for the PCB on my own, however I had the help of a few summer students that I led when it came to the programming stages of the project. In Appendix A I have prepared a rough timeline of how I spent my time during this project.

2 Background and Motivations

2.1 Coded Exposure Imaging

Coded Exposure Imaging (CEI) are a modern class of digital cameras that are an active area of research to ISML which expand on the functionality of traditional digital imaging. Their defining improvement over traditional digital imaging sensor is per-pixel control of exposure ("coding" or "masking" the image). This is generally done as a series of "subframes" which the resulting cumulative exposure is the frame. This enables different regions of the same sensor to capture light for different periods, which in conjunction with proper post-processing enables novel imaging techniques with applications in High Dynamic Range (HDR), medical, and control system imaging.

At ISML the CEI sensors being developed are multi-tap CEI pixels. Each pixel has multiple taps which are "exposed" based on the masking applied to the pixel array, thus masking pixels does not lead to light being lost, merely "sorted" into different taps during the exposure period. The T3 line of sensors has four-tap pixels while the T7 sensor has a two-tap pixel structure.

Augmenting these sensors with controlled illumination sources that are synchronized to the coding, further enhances their utility, allowing for advanced data to be collected visually, often at video rates of 30 frames per second or more. An example of this is demonstrated in Figure 1 where a CEI sensor with controlled illumination was able to determine both the normals and albedo (reflectivity) of all the surfaces in the scene.

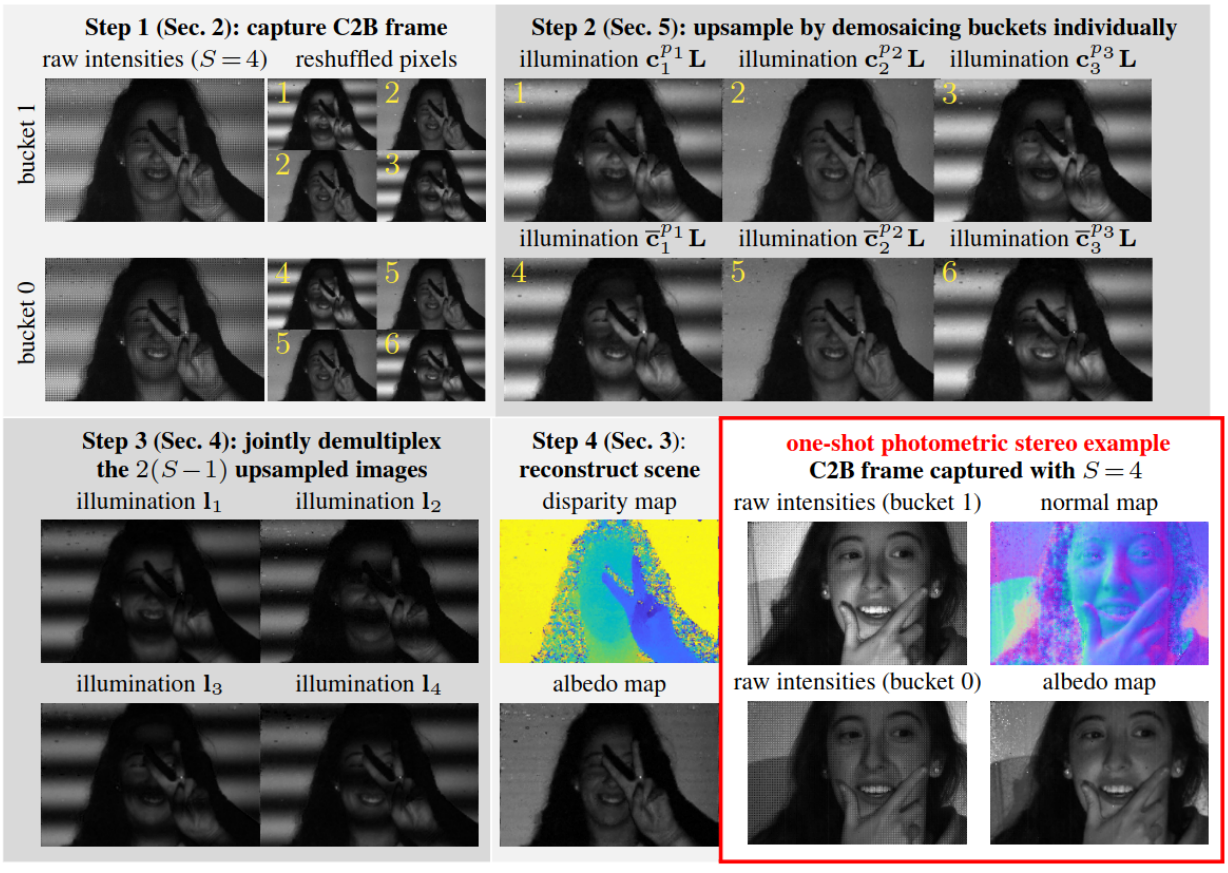

Figure 1: Example application of CEI and controlled illumination [1]

2.2 Time of Flight Sensing

Time of Flight (ToF) sensing is a method of measuring distance using waves. The basic principle is measuring the time between creating a wave and observing its reflection off an obstacle, the *"time of flight"*; the distance to said obstacle can be calculated given the wave's speed in the ambient medium. Sound is often used for this (e.g. echolocation for bats), but light can also be used with the correct sensing system.

The method used in T3.2 is called "pulse-based" ToF. Using the four buckets/taps which are all exposed 90 degrees out of phase with one another it can accurately estimate the time of flight of a pulse based on the relative amount of exposure between the four taps. Using four buckets it can also estimate the ambient lighting levels to better isolate the pulse in adverse lighting conditions. Their exposure process is shown graphically in Figure 2.

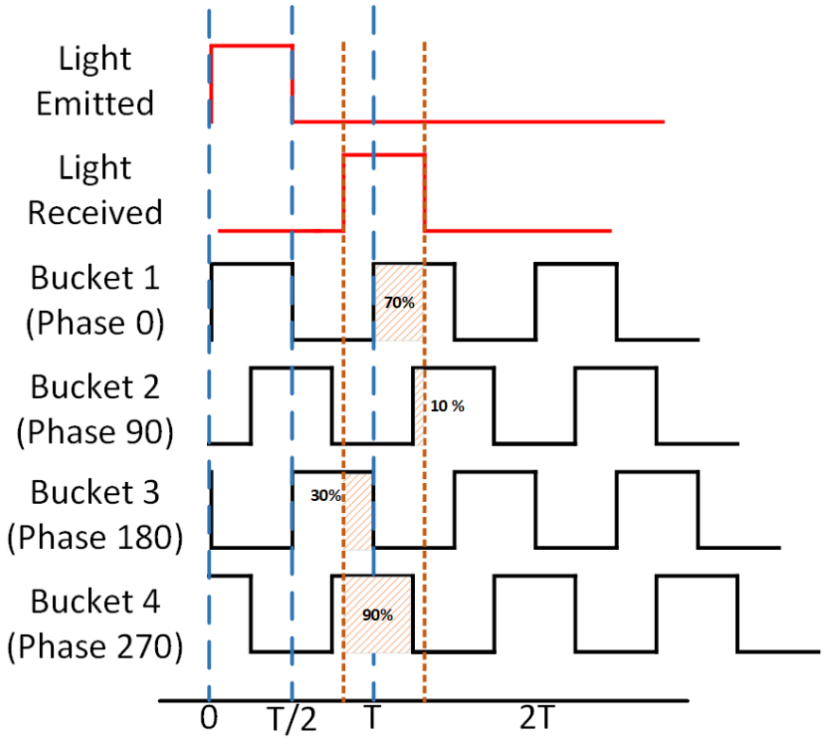

Figure 2: Demonstration of how the taps are exposed based on the pulse return time [2]

2.3 T3.2 and T7 Image Sensors

T3.2 is a four-tap pixel image sensor, with a resolution of 1032 by 44. Its primary focus is Time of Flight imaging with provisions for Coded Exposure Imaging. Due to it being a "line" image sensor it is dependant on an opto-mechanical assembly consisting of galvo mirrors to redirect light so it may properly scan a scene from top to bottom. Due to issues with the internal ADCs of T3, T3.2 was designed with 18 analog outputs for external ADCs as a contingency if this issue reoccurs.

T7 is a two-tap pixel image sensor, with a resolution of 480 by 640 (the highest from ISML thus far). Its primary focus is Coded Exposure Imaging (CEI), with it's large resolution it has been a good candidate for many new techniques. Since it is a more conventional "square" image sensor, it depends on a simple optical lens to properly focus the scene before it.

3 Circuit Board for T3.2

The core of my project was the design and delivery of a host circuit board to allow ISML the use their T3.2 image sensor for experiments. This board would be responsible for properly interfacing the FPGA with the sensor, hosting the off-chip Analog to Digital Converter (ADC) array, and providing the appropriate power levels for T3.2. A summarized block diagram of the system is in Figure 3.

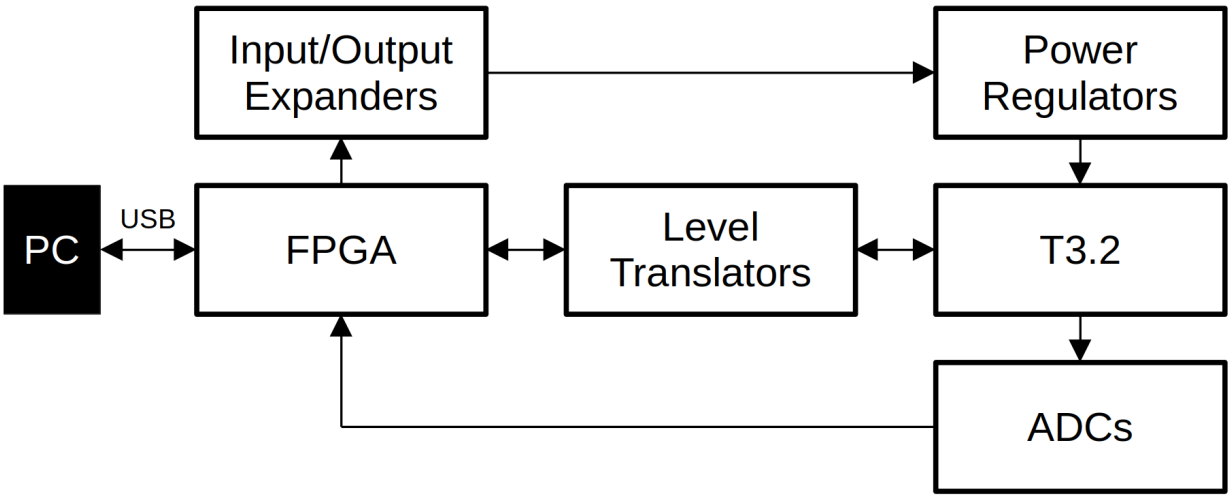

Figure 3: Simplified block diagram of board functionality

There was a design previously made for T3.2 in ISML, which was used as the basis for my design. Furthermore improvements made to the other host boards in the lab since T3 (namely T6's and T7's) were incorporated into the design.

3.1 Design Objectives

The major change for the host board from T3 to T3.2 was the inclusion of *"off-chip"* Analog to Digital Converters to serve as backups in the event the ADCs in the T3.2 chip (*"on-chip"* ADCs) failed to perform as required, as was the unfortunate case with T3. The remainder of the objectives aligned largely to the original ones for T3. Listed in no particular order, the design objectives for this PCB were:

- The new PCB had to fit within the existing enclosure and optical system for T3, shown in Figures 4 and 5
  - Location of the T3.2 sensor must fall under the current optical assembly (metallic structure in Figures 4 and 5)
  - The board was not to interfere with the vertical support to the upper left portion of the host board location and reference camera stand on the left
  - The resulting allowed dimensions were 110 mm wide, 170 mm long; with the center of the T3.2 chip placed approximately 40 mm in from both the top and right edges
- The PCB was limited to six copper layers, but could have components placed on both sides
- Handle all the FPGA-sensor connections with minimal delays and phase skew on data buses
- Host the newly introduced off-chip ADCs
- Allow for the digital adjustment of all the power lines feeding T3.2
- Allow for the insertion of external power on the power lines in the event of design failure or power monitoring is needed

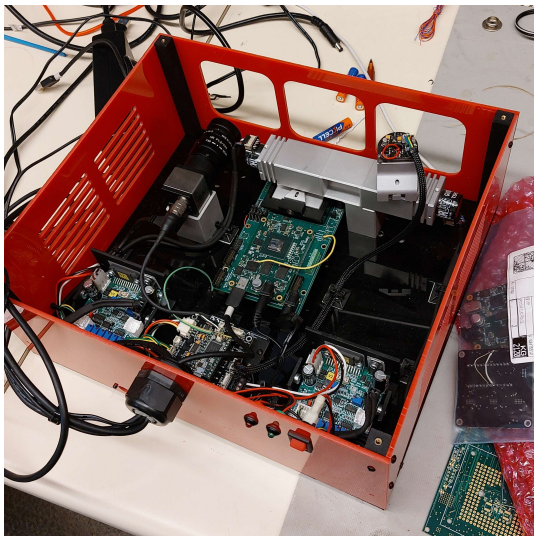

Figure 4: Overview of laser opto-mechanical assembly for T3 series

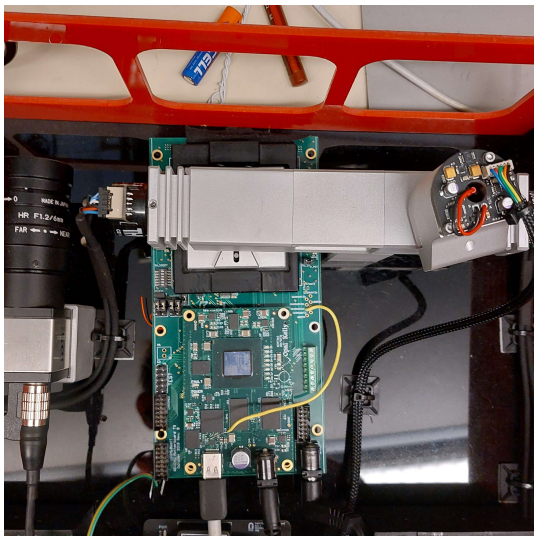

Figure 5: Location of host board under opto-mechanical assembly

3.2 Schematic Design

The first stage in designing the board for T3.2 was capturing the circuit design in schematics. At the highest level most of the connections were already correctly drawn in ISML's prior work on the draft T3.2 board, especially the connections directly to T3.2. My contributions to the schematic design were primarily optimizing and renovating subsystems with either improved designs taken from other boards or novel circuits for the group.

Part Selection

When I began the project the first task I was assigned was to check the Bill of Materials (BoM) of the existing design to identify if any parts out of stock (or critically low) from the preferred vendor so we could redesign the system accordingly. Luckily, only two components were out of stock and needed replacement as detailed in Table 1. These parts were both in the voltage reference generation subsystem for the on-chip ADCs.

Original	Replacement	Function
AD8403ARUZ10	AD5204BRUZ10	Four-channel 10 kOhm digital potentiometers
MAX38908ATD+	RTQ2520	High current linear voltage regulators

Table 1: Out of stock component replacements

These replacements were sourced from equivalents of the originals used in T7 where their performance in ISML systems was satisfactory. The new potentiometers were functionally equivalent to the originally selected ones. The replacement voltage regulators however were only rated for 2 A of current compared to the original ones' 4 A along with a slightly worse PSRR. Consulting with group members it was determined that these were acceptable performance compromises and they wouldn't severely affect the performance of the final board.

Input and Output

Given that the FPGA needed to not only communicate with T3.2 and its supporting circuits but also the off-chip ADCs, it ran out of pins to directly connect to everything, to remedy this input/output limitation two methods were employed. For "slow" signals such as chip selects or resets a pair of input/output expander chips (PCA9555DBQR) were added to the circuit allowing the FPGA to operate up to 32 pins with just two of its own at the expense of some latency and bandwidth due to the inter-chip communication. The exact connections to these are shown in Figure 6 from the top level schematic.

The use of the expansion chips did free up a significant portion of the pins needed for all the "fast" connections needed to reach the FPGA directly, these lines were the data lines to/from either the off-chip ADCs or T3.2 itself. However the FPGA was still a few pins short, so multiplexers (MUX) were selected to allow the FPGA to switch between sets of signals to connect directly to itself to avoid the latency and bandwidth limitations inherent by the expander chip solution.

The fact that there are essentially two sets of fast signals that are functionally mutually exclusive (off-chip ADCs vs. T3.2 readings) lends to easy multiplexing where the FPGA selects between one entire set of inputs or the other. The design I inherited from the previous student made use of this fact, it had a 48-channel 2:1 multiplexer circuit to accommodate multiplexing the 8 data lines for each of the six off-chip ADCs (Figure 7). These 48 ADC signals were only being multiplexed with 12 signals from T3.2, so in reality only 12 signals needed to be multiplexed and the

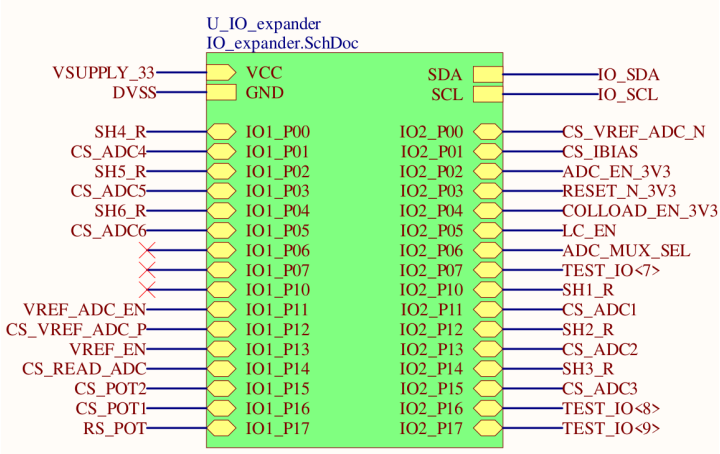

Figure 6: Top level connections to input/output expanders, "IO SCL/SDA" are to the FPGA

remaining 36 could be directly connected to the FPGA (Figure 8). This would reduce the size of the BoM in addition to removing the parasitic effects a multiplexer would impart on connections unnecessarily.

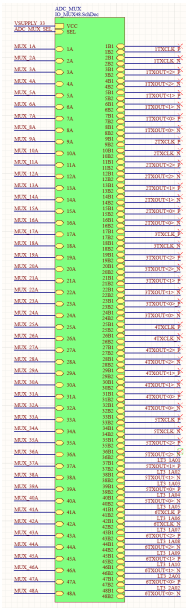

Figure 7: Original 48-channel multiplexer in top level schematic

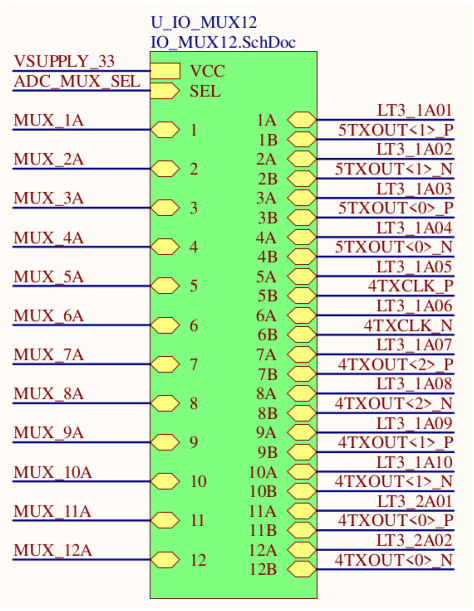

Figure 8: Revised 12-channel multiplexer in top level schematic

Level Translation

T3.2's digital domain operates at about 1.2 V, the FPGA however generally operates at 3.3 V which if directly connected to T3.2 would cause damage to T3.2. To enable them to communicate safely, level translators are used which convert between the two levels. The primary level translators selected for use with T3.2 are the SN74AVC20T245DGVR chips which have 20 channels each and are rated for up 100 Mb/s converting between 3.3 V and 1.2 V which is the target communication

speed for T3.2's use. Two of these are used to send data from the FPGA to T3.2, and one to return data from T3.2 to the FPGA.

In addition to these level translators, there are two additional level translators of a different model - SN74AVCH4T245PWT. These have only four channels, but are rated for up to 5 V. One is used to convert three signals FPGA to T3.2, the other is used to allow the FPGA to communicate with the reference voltage sources which operate at 5 V.

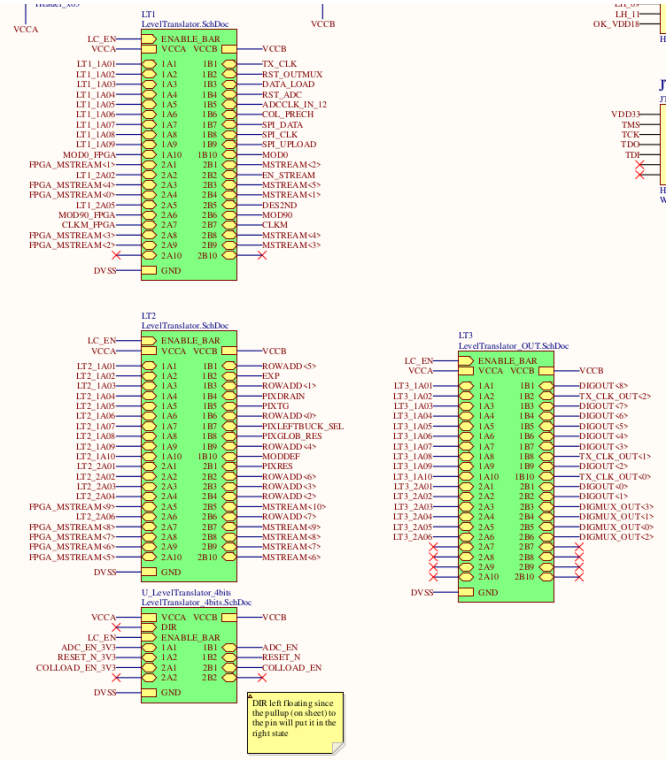

Figure 9: Level translators between the FPGA and T3.2

Power Regulators

Eight voltage regulators are needed for T3.2 to be operational in the analog domain: five for the various power rails and three to act as voltage references for the ADCs. All these regulators for T3.2 can all have their output voltages adjusted using the digital potentiometers that form their feedback networks, in addition to being entirely enabled/disabled as desired via the FPGA. This allows for safe start-up procedures to be done automatically and the voltages to be adjust quickly and accurately as desired. These analog supply voltage regulators are all RTQ2520 chips, and are used as per their reference design (Figure 10) with minor variations on the feedback topology used.

This is in addition to the six fixed regulators used to supply digital power to the components on the board. Separate 3.3 V regulators for the FPGA, level-shifters, and off-chip ADCs, 1.8 V for the FPGA, and separate 1.2 V regulators for the FPGA and T3.2.

These regulators were all selected based on those used previously for T3 or T7.

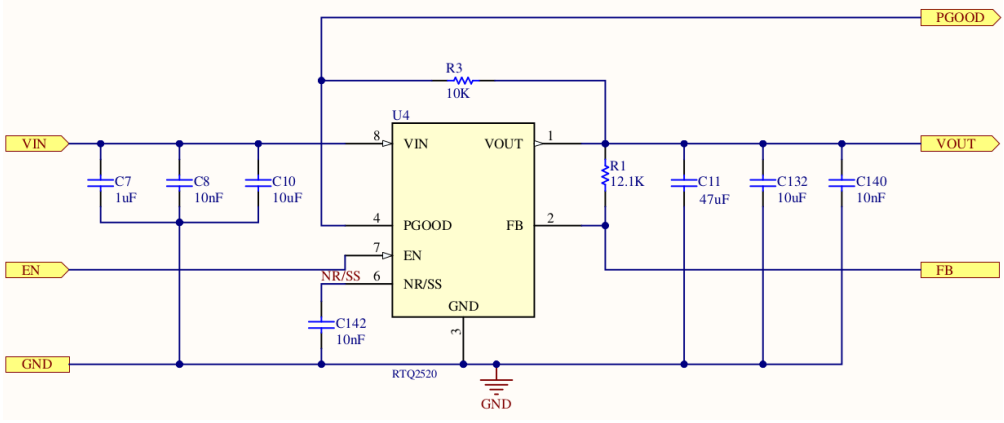

Figure 10: Design for the RTQ2520 chip used on the T3.2 board

Off-chip Analog to Digital Converters

The main change in the system design from T3 to T3.2's board is the introduction of these Analog to Digital Converters. Their purpose being to allow the group to read the analog signals generated by the pixels directly if the on-chip ones fail to meet performance requirements allowing the pixels to still be used and evaluated. 18 analog signals are fed out of T3.2 and must be sampled at rates exceeding 10 MS/s for full utilization of T3.2, so six of the LM98722 Analog Front End chips are used which can sample 3 channels at rates of 15 MS/s each.

3.3 Board Design

I contributed significantly to the board's design and layout, although there was a completed layout from previous work, the numerous changes to the circuit's design as well as improved layout practices prompted a complete redesign almost from scratch.

Floorplan

The board was laid out to satisfy the physical requirements placed on it, the full allowable size was allocated (110 mm by 170 mm) and T3.2 was placed such that its center was the required 42 mm from both the top and right edge. The FPGA was placed along the bottom in keeping with ISML design patterns. The remaining systems were then placed based on functionality, for example the level translators and the off-chip ADCs were placed between the FPGA and T3.2 while the voltage regulators were generally placed in proximity to the devices they were intended to supply. In Figures 11 and 12 the floorplan is annotated by function, un-annotated versions are available in Appendix B.

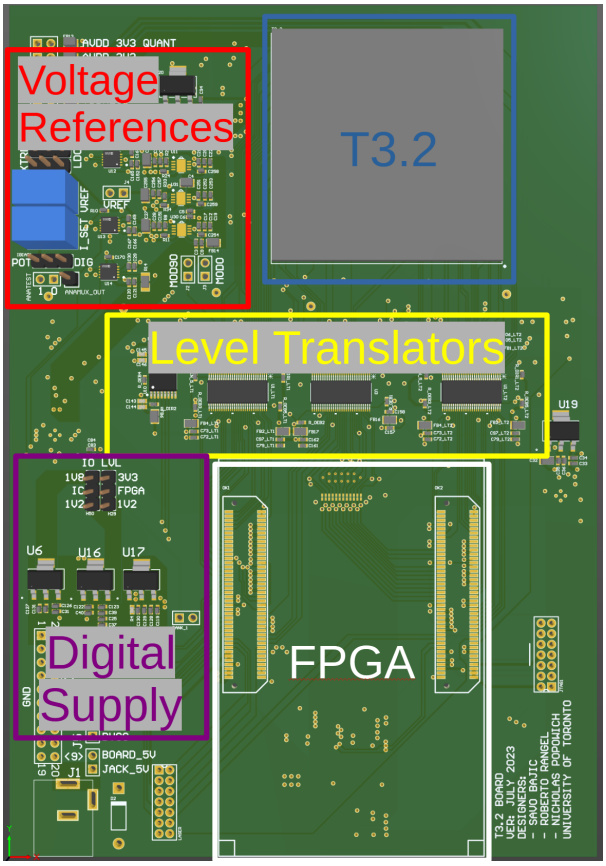

Figure 11: Floorplan for the top

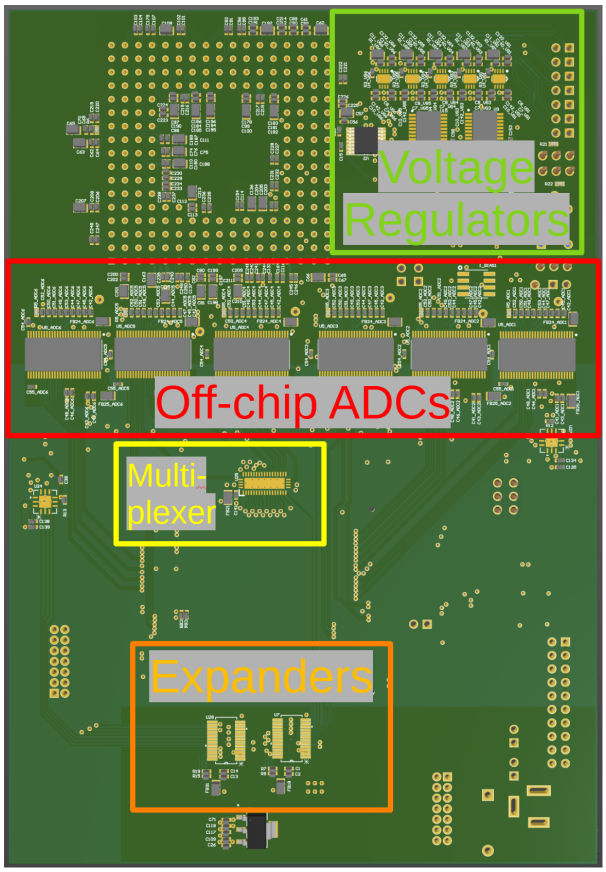

Figure 12: Floorplan for the bottom

The FPGA, level translators, and T3.2 sensor were all placed on the top layer to allow most of their traces to be run on the top layer alone, omitting the need for vias which would complicate routing and impede signal transmission. The off-chip ADCs would only fit nicely on the bottom as a result. To further preemptively improve the routing stage, many of the components on the top were shifted to the right of the board with T3.2 to try and equalize all the trace lengths.

The decoupling capacitors needed around T3.2 were placed on the bottom so they could remain close to the pins without impeding the optical assembly going around the sensor on top.

Routing

Routing began following the functional organization of components on the board. Of the six available layers, the two outermost layers were allocated for local connections to component pads, the two innermost for long signal runs, and the two intermediate ones for ground and power.

The most important data traces on the board would be the data buses between the FPGA and T3.2, specifically the mask stream into T3.2 and the digital output from it. To ensure that these signals were minimally degraded the traces were kept to the top layer for the majority of their runs except when passed to the multiplexer which was mounted to the bottom. This led to an iterative process of reassigning connections to the FPGA, multiplexer, and level translators to prevent any trances needing to cross over one another.

Once these main connections were formed, other relatively high speed and fan-out buses like the SPI bus used for interacting with many of the supporting circuits were laid out. Eventually progressing to slower and less critical traces. The final result of the data routing is demonstrated in Figure 13.

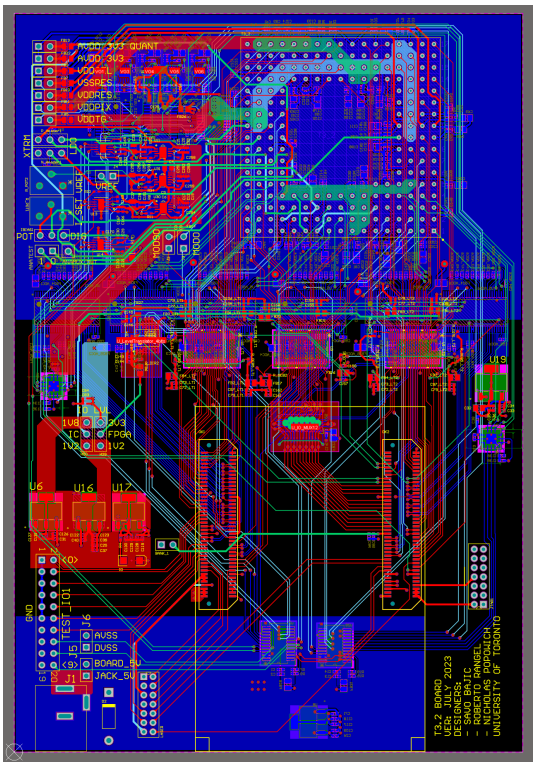

Figure 13: The data connections of the board (power planes have been hidden)

An unfortunate side-effect of having all the high speed buses on a single layer with long parallel trace lengths as observable in Figure 13 is the increased cross-coupling between the individual traces which may introduce glitches as the data rate is increased.

One important metric that was verified in reference to the high speed buses prior to commit-

ting the board to production was the bus skew. This is the relative delay between the fastest and slowest trace that belongs to a bus based on the speed of electric signals and the trace lengths of each bus element. If this is allowed to get too large then the signals may arrive across different clock cycles at the destination and be misinterpreted. The values for relative delays in the two main digital buses are summarized in Table 2.

Observing the results the maximum skew across both buses was approximately 150 ps which was determined to be negligible due to this representing about 3% of the 5 ns clocking period to be used. Furthermore the delay of bus elements to the bus clocks was found to be at most 208 ps which was determined to be just as negligible so no corrections were needed.

Bus Element	Trace Len. (thou)	Est. Delay (ps)	Delay to Bus Min. (ps)
CLKOUT<0> *(DIGOUT)*	3625	508	208
CLKOUT<1> *(DIGOUT)*	1902	266	-33
CLKOUT<2> *(DIGOUT)*	2225	312	12
DIGOUT<0>	3284	460	160
DIGOUT<1>	2987	418	119
DIGOUT<2>	2184	306	6
DIGOUT<3>	2184	306	6
DIGOUT<4>	2140	200	0
DIGOUT<5>	2423	339	40
DIGOUT<6>	2290	321	21
DIGOUT<7>	2231	312	13
DIGOUT<8>	2667	373	74
CLKM *(MSTREAM)*	4973	696	207
MSTREAM<1>	4161	583	94
MSTREAM<2>	4060	569	80
MSTREAM<3>	4443	622	133
MSTREAM<4>	4548	637	148
MSTREAM<5>	4599	644	155
MSTREAM<6>	3492	489	0
MSTREAM<7>	3687	516	27
MSTREAM<8>	3706	519	30
MSTREAM<9>	3679	515	26
MSTREAM<10>	3542	496	7

Table 2: Trace delays of critical signals

3.4 Validation

Once the boards arrived at the lab (as in Figure 14) and final assembly was conducted, they were put through some introductory tests to ensure the design was minimally viable prior to the installation of any T3.2 chips or FPGAs. These were semi-formal tests that can be quickly summarized.

1. "Smoke Test" - Apply power and monitor board to ensure none of the components are getting excessively hot
    - Off-chip ADCs and their voltage regulator were found to heat up quickly when power was applied
    - Remedied by disconnecting the ADCs from power. This was acceptable since they could be reconnected later if T3.2's internal ADCs were found inadequate.
    - No other parts were found to heat up when power was applied
2. Voltage rail verification - Inspect all voltage regulators to be outputting the expected values
    - All fixed regulators met their expected outputs
    - Varying regulators met their expected default outputs based on the power-on settings of their potentiometers and feedback circuitry
3. Data path verification - Checking continuity between critical points
    - All critical data paths (e.g. DIGOUT, MSTREAM) met expectations

    NOTE: This test is generally performed by automated testing equipment at the end of PCB manufacture by vendors to ensure the produced boards meet specification. This test is mostly checking that solder joints (especially those done by hand) are valid.

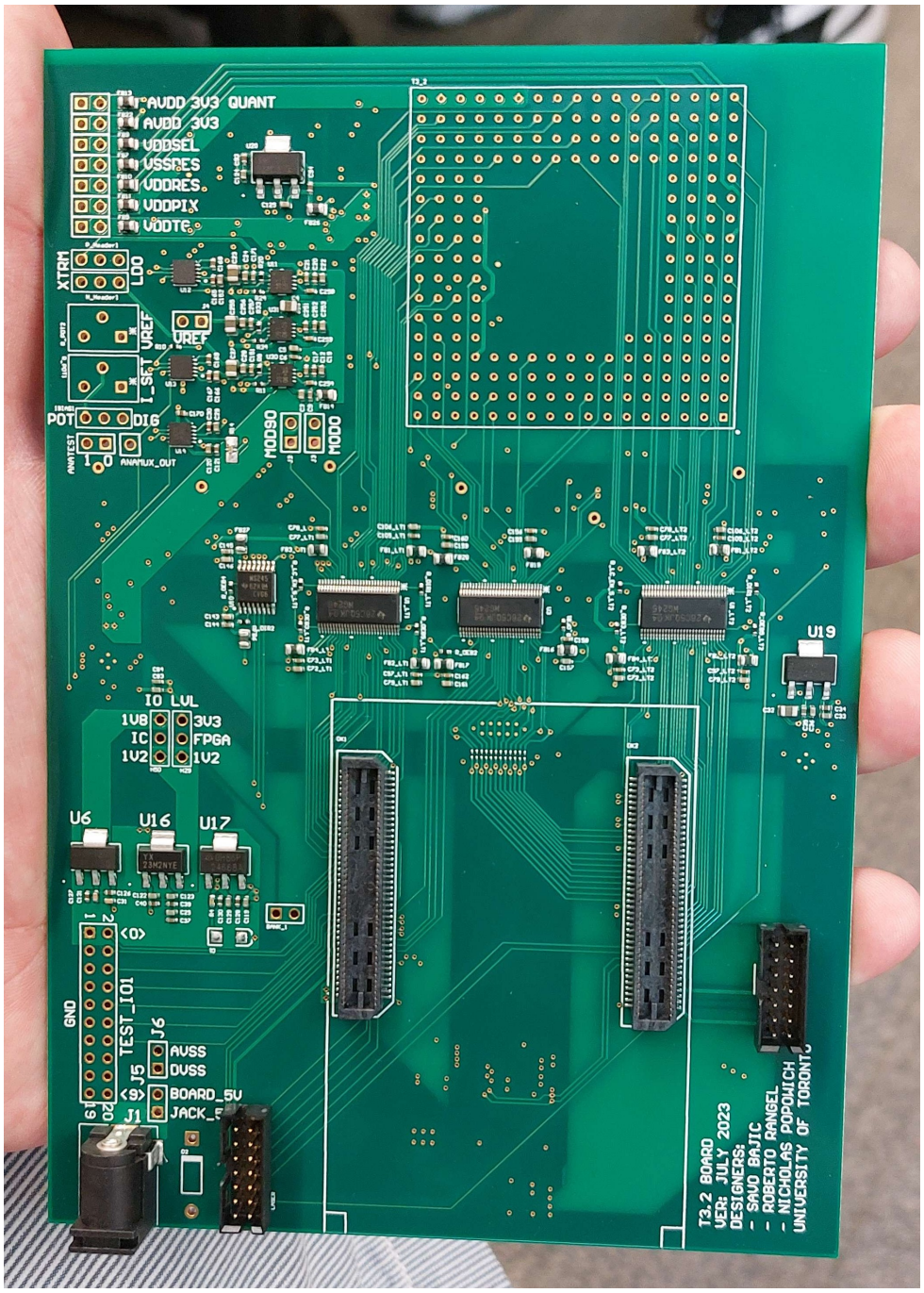

Figure 14: Boards prior to manual installation of headers, potentiometers, and sensor socket

4 Drivers for T3.2

With the hardware's basic functionality verified on arrival, it came time to develop the software drivers needed to properly manipulate the system into ultimately getting the T3.2 sensor operational. This was to be conducted in two stages coinciding with the installation of the FPGA first to build up and test the drivers needed for the peripheral circuitry, and then a T3.2 chip once we were confident the board was ready for it and it would not damage the sensor.

The majority of this work was done in Verilog for the the FPGAs, with some Python code prepared for the host computer.

4.1 Peripheral Circuitry

I use *"peripheral circuitry"* to encompass any circuitry on the board other than the FPGA or T3.2. In summary this would be the following:

- Input/output expanders
- Level translators
- Multiplexer
- Off-chip ADCs
- Potentiometers
- Voltage regulators/references

The majority of the peripheral circuitry uses Serial Peripheral Interface (SPI) to communicate with the FPGA, making an SPI driver for the board a requirement. As part of this protocol the chips must receive a "chip select" signal which is when a dedicated pin on said chip is pulled low, as mentioned in Section 3.2, all of these lines were moved from connecting directly to the FPGA to the input/output expander chips. These are controlled by the FPGA using a second protocol: Inter-Integrated Circuit (I2C), which in turn necessitated its driver on the FPGA.

Input/Output Expander

Due to the fact that all peripheral circuitry depended on the correct operation of the input/output expanders to enable or select the appropriate chips, this was the first driver developed. It was composed of two parts, an I2C interface for the FPGA pins, and a wrapper module which would read in the desired state for the expanders' pins and convert that into commands for the I2C driver to enact. Due to I2C only being used for the expander chips, it was fine to abstract it away like this. A diagram of this system is portrayed in Figure 15.

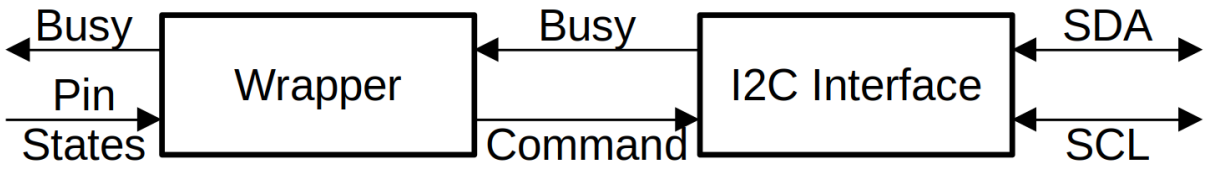

Figure 15: Block diagram of the FPGA's I2C driver system

A benefit of this approach is that should there be a future revision of the T3.2 board (*perhaps for a T3.3 sensor!*) where off-chip ADCs are no longer needed freeing up FPGA pins, the expander driver can be removed entirely and the connections that were previously made to it can be routed to FPGA pins directly.

The actual behaviour of each module was also relatively simple owing to the division of function. The interface module simply idles waiting for a command, then it will assert that it is busy as it executes the command, before returning to idle until its next command is received. The commands follow a simple format: 7-bit address, 8-bit target register address, 16-bits data to write to register, all padded with a leading zero to make it 32 bits wide.

The wrapper idles by monitoring the pin states it is supplied to the last set it enacted via the interface, when there is a change between clock cycles it reacts by asserting the busy flag and generating the required command for the interface module. It then waits until the interface completes the transaction before deasserting its own busy flag and returning to idle.

The busy flag is very important for ensuring that the correct sequencing can occur in the system. If the FPGA had the connections done directly to pins, code in the FPGA can safely assume that by the next clock cycle that pin is in the expected state. However with the expander chips this assumption is far from the truth since the I2C transaction takes approximately 90 microseconds to complete, about 9000 clock cycles at 100 MHz. So for modules that need to be sure that the pin is in a desired state before progressing (e.g. an SPI driver selecting a chip), it can monitor the state of the busy flag to know when the pin has reached the desired state.

After some testing and tuning to increase the communication speed to the maximum supported by the input/output expanders the driver was complete and latency was measured to be the aforementioned 90 microseconds. With this working the level translators, multiplexer, potentiometers, and voltage regulators could all be enabled and/or selected.

SPI System

The SPI driver was based off the design from T7, modified to work with the input/output expander via the I2C driver for chip selection. Unlike the I2C driver which receives commands from a wrapper and is thus "hidden" from the user, the SPI driver is configured on the FPGA to receive instructions directly from the user (or rather our host-side driver software) using the OpalKelly FrontPanel system. These instructions contain both the target device address as well as the information to transfer.

The modification needed to adapt T7's driver to T3.2 was to add additional states internally to respect the time it takes the I2C/expander driver to toggle the chip select pins as needed. This was done by monitoring its busy state to ensure that it was asserted following a requested change in the chip select pins and then advancing to next state once "busy" was eventually deasserted following a completed pin configuration. The resulting modified system is shown in Figure 16.

One limitation of the driver presently employed on T3.2 is that it does not record the data it is provided during the transaction, it merely serves to command other chips.

Voltage Regulator Drivers

The voltage regulators controlled using a combination of both the input/output expander and the SPI drivers. The voltage regulators used only received signals to enable or disable them directly (via the expander), their output levels were indirectly controlled by adjusting the settings of the digital potentiometers placed in their feedback networks.

The driver for these made use of the existing SPI driver interface, so no Verilog code was prepared for this to work on the FPGA, it was all developed as part of the Python driver for the T3.2 camera. It took the form of a function which received a target channel and the desired

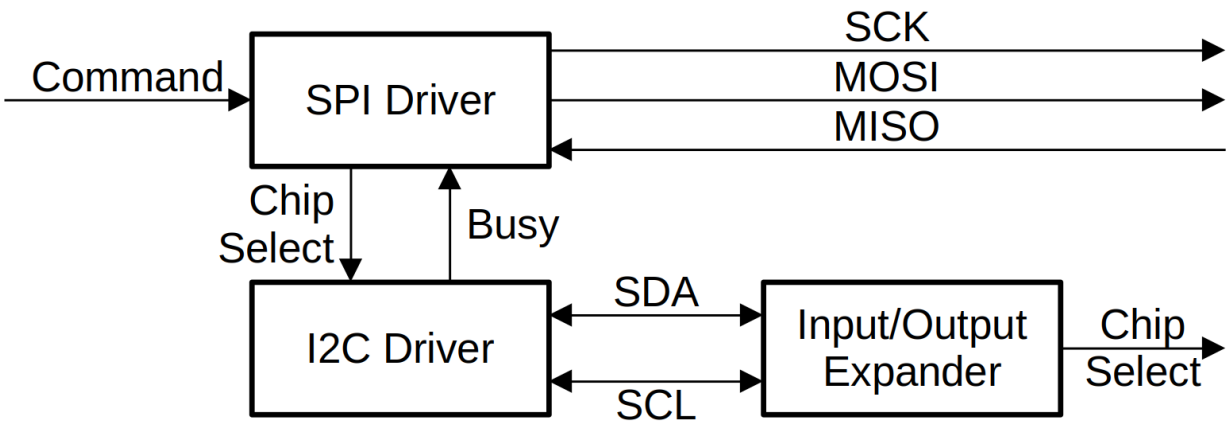

Figure 16: Block diagram of the FPGA's SPI driver system

voltage to set it to. It would then derive the required SPI command needed for that given voltage channel to approximate the voltage based on the analytical model of the feedback circuit and push it through the SPI pipeline.

This worked for the analog domain power supplies, however the potentiometer used for the analog reference voltages failed to respond as expected to the instructions. I believe this has to do with the SPI protocol configuration and the driver not being designed to handle different phase ("PHA") chips. This minor issue with the references lead to them simply being bypassed and shorted to their respective extremes to allow development to continue for other parts of T3.2.

Off-chip ADC Drivers

There was no development of drivers for the off-chip ADCs. This was skipped so efforts could be focused on getting T3.2 operational as originally envisioned using its on-chip ADCs - only returning to work with these off-chip ADCs if it was deemed necessary.

4.2 T3.2 Interfacing

Once all the drivers for the peripheral circuitry were in a reasonable state, work on the drivers to interact with T3.2 began. With T3.2 largely resembling T3 in terms of their digital system, the hope was that minimal modifications would be needed to port the existing code base for T3 to work with T3.2.

The plan for developing and verifying T3.2 was to start at the bottom of the data flow as portrayed in Figure 17, configuring the sensor, and gradually working up through the other stages to be certain that there were no unexpected issues between the stage being worked on and the FPGA.

Each step in developing drivers for T3.2 was as much a test of the drivers working, as it was a test of T3.2 behaving as designed to, since it had yet to be used by the lab.

Configuration

The first step in operating the sensor is to configure its control registers to have the sensor work in a given manner. This would allow the sensor to be set into its various test modes which would allow the driver for T3.2 to be gradually developed knowing the expected behaviour of the sensor.

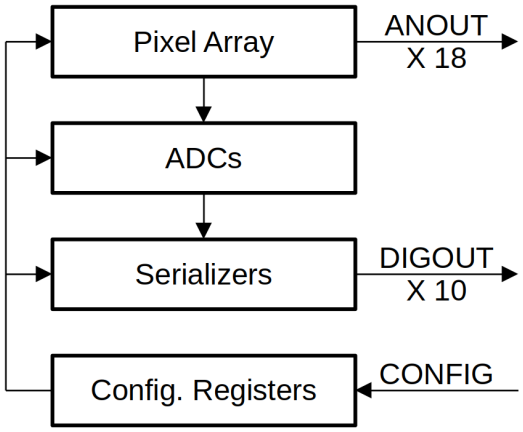

Figure 17: Summary of the main data flows in T3.2

Due to some of the internal differences relating to the analog systems in T3.2 from T3, the control registers were different so a new driver needed to be prepared.

Configuration is accomplished using a serial interface, not unlike SPI, where data is shifted in bit-by-bit and deserialized. However there is no chip select nor a constant output pin for the stream from T3.2. Instead there is a signal to load in the deserialized data stream and an output pin can be enabled by writing to the right configuration register. Due to the similarities of this protocol to SPI, its driver was used as the basis for this with modifications made to pulse the data load signal at the end of transmission.

To verify if the configuration was operating correctly, the test configuration stream used was intended to connect one of the test outputs (DIGMUX<0>) to the output of the configuration shift register. Thus if the system was working correctly then the output would return the previous command register setting as the new one was clocked in.

It took some time but there were two issues with the driver that I rectified and with help of others resulting in a working configuration driver. Firstly, we were originally looking for a response from the wrong output pin on T3.2 so it was not possible to verify the response. Secondly, the order of bits fed into T3.2 was reversed.

Reading the Serializers

The first stage in reading actual image data is to operate the serializers on-chip and then deserialize the data correctly on the FPGA's end. These were 100 MHz Double Data Rate (DDR) serializers and thus output data on both edges of the clock signal, effectively outputting 200 Mb/s at max speed. The most basic test was to supply a clock to the output stage with test mode enabled for the serializers in the configuration register. The expected result was a repeating 14-bit sequence "11110000110010" on all output channels.

Unfortunately the tests did not achieve this outcome. With test mode supposedly enabled, the digital outputs were all holding high as they were clocked. Other tests have been attempted to try and diagnose the issue but none have had any meaningful results. The following tests were conducted:

- Clock speed - It was possible that the clock speed was too fast for the serializers or the level translators. It was found when directly probing the clock input to T3.2 it was partially attenuated by the level translator which was operating at its maximum rated speed. Decreasing the clock rate did not help.
- Configuration register - The wrong bit might be set in the configuration register so test mode is not actually enabled. A proposed test is to set the entire control register to all 1's to guarantee test mode, but this has not been attempted as of writing.
- Serializer design issue - The serializer design in T3.2 is different to the one used for T3, it could be possible that the design doesn't function. Tests in Cadence strongly support the design as valid though.
- Our chips are faulty - Perhaps there is some damage on the test chips. Shouldn't affect them all equally though, and they all respond to the configuration streams.

The root cause is yet to be determined as my attention was shifted to helping with T7 for the final weeks of my project period. This is where my driver development for T3.2 paused since it was difficult to assess the on-chip ADCs without functioning serializers.

5 Subframe Readout for T7

On T7 there are two operating modes for the on-chip ADCs: normal 12-bit operation, and a 1-bit mode where the ADC is used like a comparator. The reasoning for this is that it can enable Flux to Digital Conversion (FDC) thanks to more frequent frame readouts. Flux to Digital Conversion is an alternative way of measuring the luminosity of a scene where instead of recording the value of each pixel at the end of exposure (essentially taking the integral of light striking it), the time it takes a pixel to cross a threshold is monitored (providing the derivative, or light flux) which is then extrapolated to estimate the true "brightness" values that the pixel would have at the end of exposure even if it might become saturated. This is shown graphically in Figure 18.

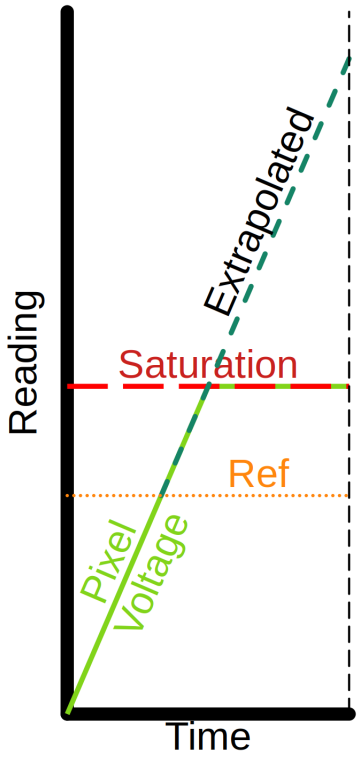

Figure 18: Basic demonstration of FDC extrapolation

The critical part of this process is the high frequency frame scans, T7 being capable of up to 2000 of these "subframes" a second. The problem with the code that achieved the 2000 subframes a second is that it was not fed live to the host computer, a short period of video was recorded to the Dynamic Random Access Memory (DRAM) on the FPGA and then uploaded to the host in a burst while the camera stops recording (because the DRAM is busy dumping the footage).

5.1 BRAM + USB 3.0 vs. DRAM Hypothesis

My goal was to help create a system that could stream the full 2000 subframes a second to a computer live. On paper this was well within the realm of possibility: each 480 by 640 frame at 1-bit of pixel depth would require $480 * 640 * 1 = 307200$ bits, at 2000 a second that would be just under 615 Mb/s, or 76.8 MB/s. This corresponds to approximately a quarter the verified 300 MB/s bandwidth achieved with the FPGAs employed in the ISML lab.

The proposed idea was simple, remove the DRAM that was preventing the easy streaming of

data through the system and replace it with something that can offload the data quickly. This was deemed to be an ideal application of using the onboard Block Random Access Memory (BRAM) of the FPGA and streaming the data at full speed to the USB without storing it for a notable amount of time. As long as the USB 3.0 connection would be able to move data off the FPGA faster than the data was generated (which it seemed to be able to), then the buffers would never overflow and the camera could run at full speed.

There was work done prior to my involvement as I was taking over for an outgoing master's student. These initial prototypes did eliminate the DRAM component, however they were only able to sustain about 100 subframes a second throughput.

5.2 Optimization

There were two angles to approach hastening the streaming: the host computer's processing (done in Python) or the FPGA's code. The Python client was struggling to keep up with displaying the data on screen with all the processing it had to do, but if it was only required to save the data to the drive it, was able to maintain the required throughput when data was available. I looked for ways to improve the system speed by speeding up the existing transfer speeds from the FPGA so more data could be made available to the host and by performing some operations on the FPGA to save processing time on the host.

Speeding up the transfer rate required a few tricks, the main one was the reorganization of how memory was used inside the FPGA. Originally two separate buffers were used to hold the outgoing data to form a bus width of 320 since that was the amount of data deserialized per half-row. Changing this to a single narrower, but deeper, buffer that was implemented in BRAM improved the system's performance and provided some much needed timing slack for the other optimizations I was trying to implement.

I intended to use the pipelining and parallelization of FPGAs to perform some of the common data manipulations that the host computer was performing to reduce the post-processing time for the stream frames. The operations I had a specific interest in performing was re-organizing the order column data was fed into the deserializer to match reality. Unfortunately any attempt to do so that deviated from how it was handled data prior to my involvement led to timing issues the corrupted the readout portion results.

In the end, the sum result of the work done to speed up this system achieved a throughput of approximately 250 subframes a second, which was only about a tenth of what was desired. There was no clear bottleneck in the system at this point.

5.3 USB Investigation

Due to the disappointing results from the optimization, I tasked two students to try and replicate the results for the high transfer speeds advertised of the FPGAs - without working in the framework of a camera (so they could just stream dummy generated data). This way one could observe what connection configuration yields the best transfer rate and then gradually build up the camera system around it so as to not disturb the communication setup.

The result of their work was that they were able to both achieve transfer rates of around 300 MB/s as advertised, however this was only feasible if there was no gap in the supplied data. Their results showed that the intermittent generation of frame data combined with limited buffer space available on the FPGA meant that the transfer was constantly starting and then stopping transfers as it consumed all the data. The repeated start/stops adding time overhead to the trans-

fer ultimately limiting the throughput.

The only way forward indicated by this was to have a larger buffer that doesn't need the transfer to start/stop transferring as frequently, as was the case with the original DRAM system. This however would likely need significant rework of that memory module to make it more streamlined so it can read and write to memory in quick succession.

6 Generative Masking for T7

Generally the pixel masks for CEI sensors from ISML are generated by the host computer and uploaded to the FPGA pixel-by-pixel and stored in memory on the FPGA for each subframe before uploading the masks to the sensor during exposure.

As part of the lab's efforts into adaptive masking algorithms with spatially varying burst imaging, I worked on an alternative approach to handling masking. The theory was that as long as the subframe number is known, the size of burst imagining tile a pixel belonged to, and the position (column/row) of the pixel in the array, it can be mathematically determined whether it is needed to be exposed or not, rather than depending on a mask for each of the subframes.

This would allow much less of the memory to be allocated for buffering the mask data since only one "tile size map" would be needed for any number of subframes with a bit depth based on the maximum tile size, rather than buffering several masks worth of pixels. It also allows for the FPGA to easily adjust its masking internally based on some algorithm in the future.

By limiting the valid tile sizes to powers of two, the mathematical expression for whether or not a tile should be exposed can be expressed using the following formulas where s is the subframe number, t the power of two for the tile size of a given pixel, c and r the pixel's column and row in the image, e being whether of not to expose that pixel.

$$a = (s >> t == c\%t + t * (r\%t))$$
$$b = (s\%t == 0)$$
$$e = a\&b$$

In the equations, a is true when it will be the that pixel's turn to be exposed within the tile; for example in a 2 by 2 tile, the top right pixel is the second to be exposed. b is true when the that given tile size is meant to be exposed based on the largest tile size, for example if the tile size for a pixel is 1 but the maximum is 4, then this will only be true every fourth subframe.

To implement this tile map parsing logic nicely on an FPGA a series of counters which would rollover at specific values are used in place of modulo operation to reduce logic complexity and ease timing since finding the modulo of something is essentially counting up to that number but restarting a second count periodically. E.g. $5\%3 = 2$ which is the same as clocking a counter with a maximum of 2, five times: $1, 2, 0, 1, 2$. To select what value to modulo by, one simply selects the counter with the corresponding limit.

This was successfully implemented and tested using Verilog, with a sample result waveform shown in Figure 19. It was eventually developed so that an arbitrary image size and masking column spacing could be applied and the hardware would handle it.

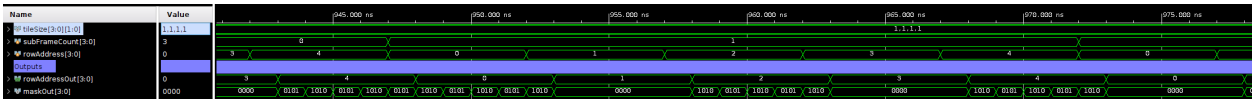

Figure 19: Logic simulation of a 2 by 2 tile array on a 20 column, 5 row test sensor being parsed

Although the tile map parsing logic was completed, there was a change in priorities prior to the completion of the tile size map generator, so it is not yet complete. I believe it would not take

long to complete the effort if effort was once again dedicated to it.

I believe that this approach could have benefit for other pattern-based, or predictable mask behaviours such as a map to define the number of subframes to expose a pixel for in high dynamic range photography/videography applications.

7 Conclusion

Reflecting on the status of the T3.2 system it is unfortunately far from complete, held up by the issues related to the serializers and a an overall lack of time invested into the system due to changing priorities for the Intelligent Sensory Microsystems Laboratory Image Sensor group. I believe that if given more time either a solution to the serializer issue could be found or work could begin on bypassing them entirely using the off-chip ADCs. I am confident that the circuit board I designed is able to support these efforts as it stands.

Regarding my work on T7, I believe that the efforts into subframe readout held value, even if they failed to reach the performance that was desired. They revealed that an effort to revise the DRAM module rather than attempting to eliminate it entirely with BRAM might hold the key needed for maximizing our effective data throughput. As for generative masking, I think it holds the most promise in the near future as it heavily relates to the group's current focus on burst imaging applications and adaptive masking algorithms.

I've made a compiled list of my recommendations for future work in these areas based on my experience in Appendix C.

References

[1] M. Wei, N. Sarhangnejad, Z. Xia, *et al.*, "Coded two-bucket cameras for computer vision," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 55–73, ISBN: 978-3-030-01219-9.

[2] N. Gusev, *System-level design of coded-exposure cameras for computational imaging applications*, 2019.

Acronyms

ADC Analog to Digital Converter. 4–7, 9–11, 14, 16, 18, 20, 21, 26, 28, 33

BoM Bill of Materials. 7, 8, 28

BRAM Block Random Access Memory. 22, 26

CEI Coded Exposure Imaging. 3, 4, 24

DDR Double Data Rate. 19

DRAM Dynamic Random Access Memory. 21–23, 26

FDC Flux to Digital Conversion. 21

FPGA Field Programmable Gate Array. 2, 5–9, 11, 12, 14, 16–19, 21, 22, 24

HDR High Dynamic Range. 3

I2C Inter-Integrated Circuit. 16, 17

ISML Intelligent Sensory Microsystems Laboratory. 2–5, 7, 11, 21, 24, 26

PCB Printed Circuit Board. 2, 6, 14, 28, 33

PSRR Power Supply Rejection Ratio. 7

SPI Serial Peripheral Interface. 12, 16–19

ToF Time of Flight. 2, 4, 28

Appendices

Appendix A Work Schedule

In Table 3 is an approximate breakdown of how I spent my time during this project.

Month	Principle Tasks
Jan/Feb	Updating T3.2 regulators Checking T3.2 BoM for out of stock parts
April	T3.2 schematic review T3.2 PCB redesign
May	T3.2 PCB redesign
June	Design for manufacture review Machined enclosure components Started subframe readout
July	Board in production, started prep for T3.2 board Worked on T7 ToF tests T7 subframe readout Oversaw USB read speed experiments
August	T3.2 arrival and verification T3.2 drivers T7 generative masking T7 ADC verification

Table 3: Approximate work schedule

Appendix B Layout Renders

The following appendix showcases figures of the un-annotated layout renders.

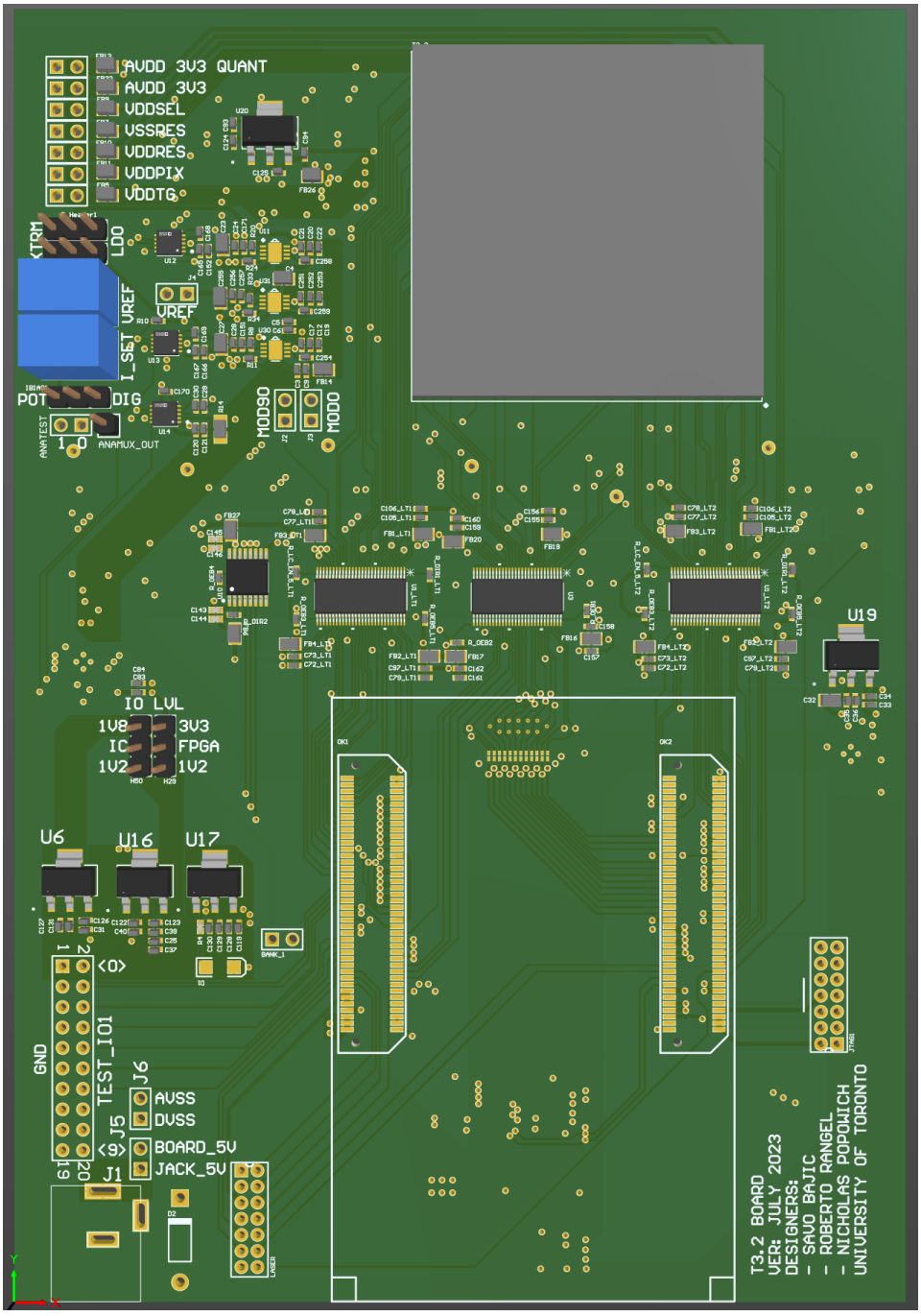

Figure 20: Render of board top

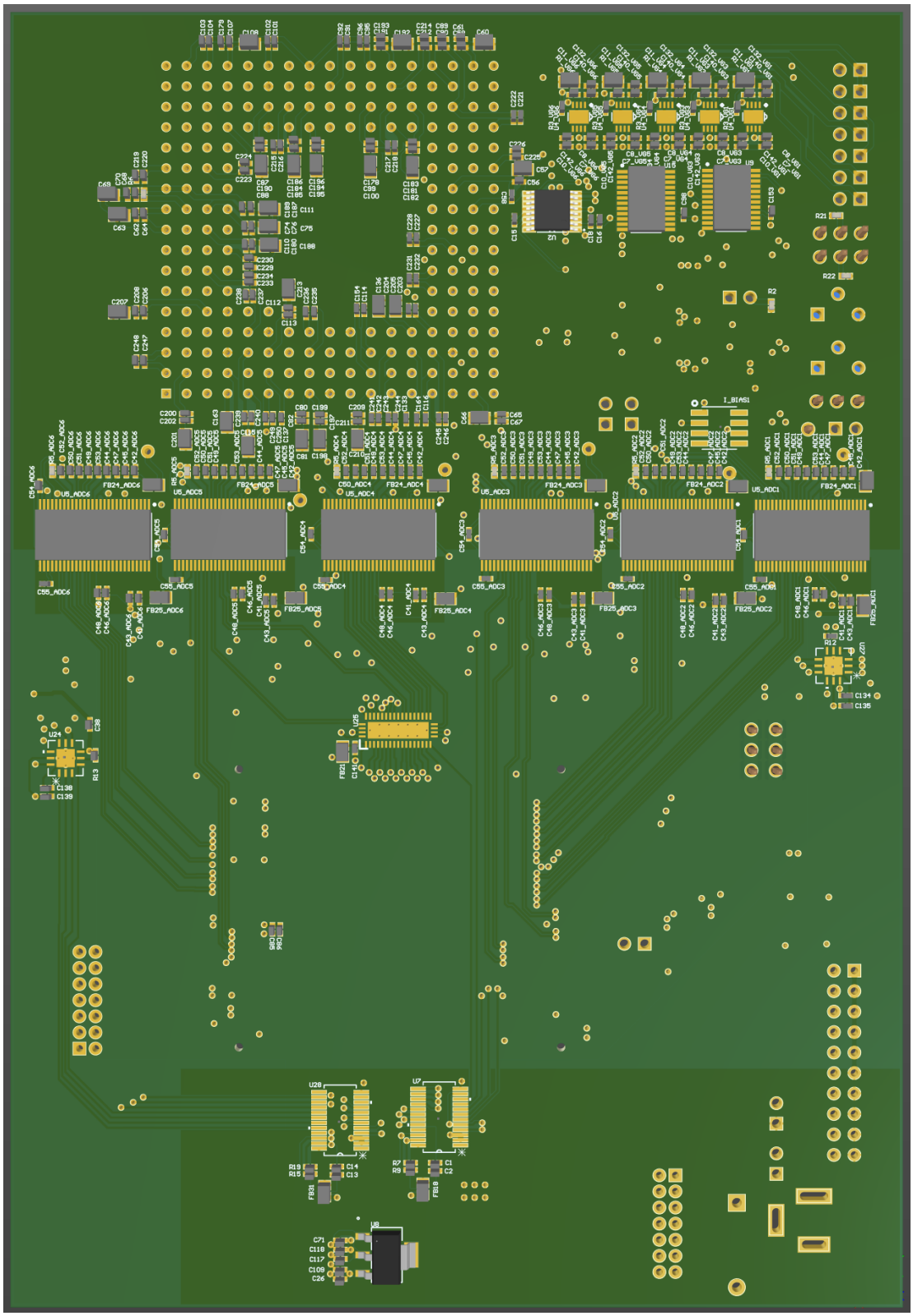

Figure 21: Render of board bottom

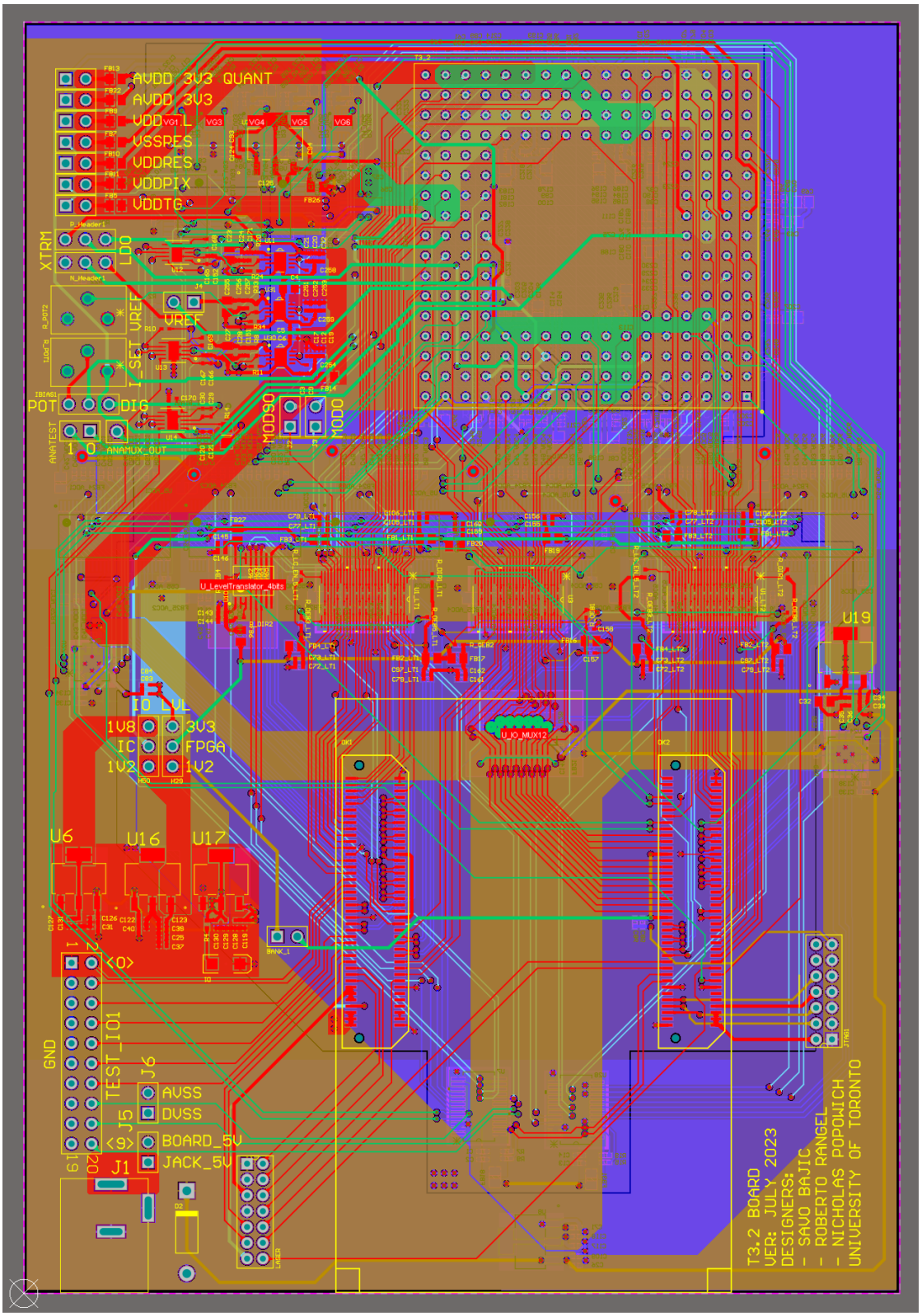

Figure 22: Routing of board with power planes

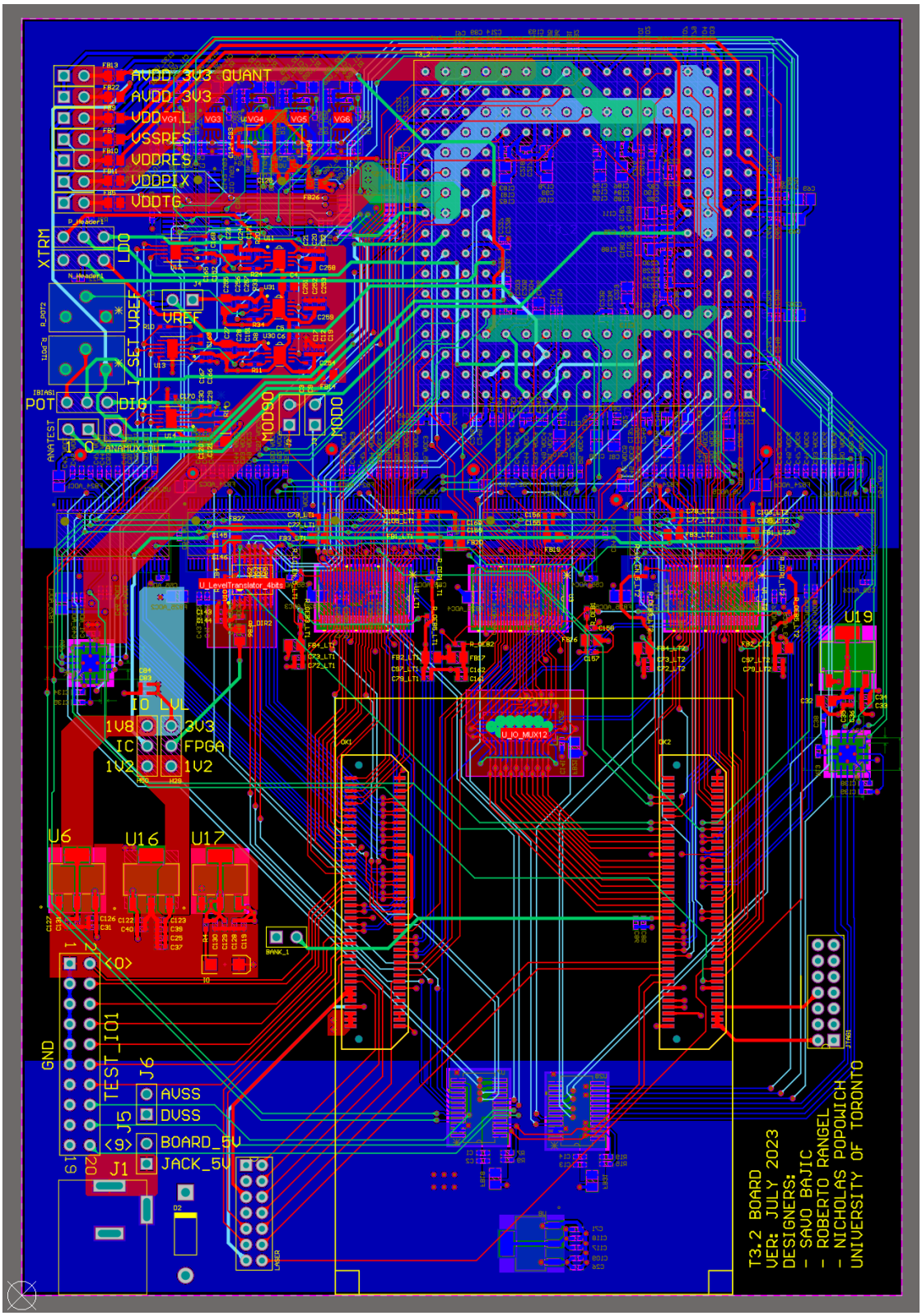

Figure 23: Routing of board without power planes (mainly data connections)

Appendix C Recommendations for Future Work

This is a summary of my key findings and advice for future work in the areas I worked on, I hope that these make their way into the hands of the future stewards of these projects. They are listed in no particular order.

- Add resistors to pull voltage regulators and potentiometers into disabled states on power up for safer startup sequencing (i.e. ensure potentiometers are set to the right value before enabling the voltage regulators, then the level translators)
- Have a more powerful or efficient regulator for the off-chip ADCs or other high load lines
- Make sure that the sensor packaging is properly marked, if not then modify the marking on the PCB to prevent incorrect insertion as was the case for two T3.2 sensors. Their health is yet to be assessed since they were incorrectly installed and powered.
- Change the potentiometers used for the reference circuits or invest time into an improved SPI driver which can handle varying polarity and phase settings.
- Redo the feedback circuits for voltage regulators to make more straightforward use of the potentiometers. Currently they are used in parallel with some resistors which makes only a small portion of the potentiometer's range actually used in addition to complicating the voltage setting math. Consider applying the rheostat scheme shown in Figure 24.
- Get regulators that can regulate to values below 0.5 V for the low-side regulators (VSS)
- Investigate getting faster level translators between FPGA and sensors
- Invest in improving the DRAM interface for more effective buffering in and out of it, especially to the USB interface
- T7 readout system is very timing sensitive, it might be worthwhile to harden it timing-wise to allow for more experimentation on that platform

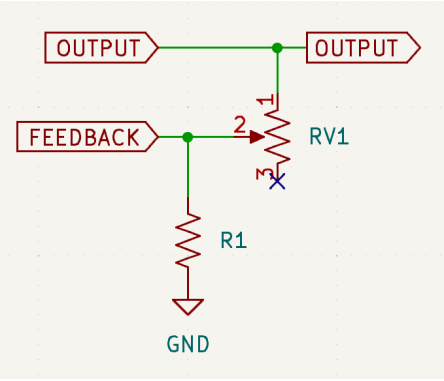

Figure 24: Rheostat feedback configuration for a generic voltage regulator