# DEVELOPMENT OF A SMART UNMANNED SYSTEM HYPERSPECTRAL IMAGER FOR GROUND FUEL CHARACTERIZATION

by

Nathaniel Bennett Sweeney

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2025

# DEDICATION

I dedicate this to all my family members for their love and support over the years, especially my father who I miss dearly.

# ACKNOWLEDGEMENTS

I would like first thank my advisor Dr. Ross Snider who has been an incredible mentor for the last three years starting with capstone, and will continue to be a great mentor to me over the next three years as I continue on. He is a terrific source of knowledge and experience and has been instrumental in my growth as a researcher and a person. I would also like to thank my other committee members: Dr. Brad Whitaker and Dr. Joe Shaw. Both of them have played a key role in my learning over the last few years, especially for the gaps of knowledge that I had going into this project.

I would also like to thank all of my colleagues at RCI who have been a great source of knowledge and friendship over the last year and a half. I am also thankful for all of the different members of the SMART FIRES project that I've worked alongside.

I'm incredibly thankful for all of my family and friends for supporting me over the years and bringing joy to my life. From baking to just having people to talk to about anything, I could not have done it without any one of them and I cannot thank them enough.

This material is based upon work supported in part by the National Science Foundation EPSCoR Cooperative Agreement OIA-2242802. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Computational efforts were performed on the Tempest High Performance Computing System, operated and supported by University Information Technology Research Cyberinfrastructure (RRID:SCR_026229) at Montana State University.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

# LIST OF TABLES

# LIST OF FIGURES

LIST OF FIGURES – CONTINUED

# ABSTRACT

Hyperspectral imaging and machine learning have been shown to be valuable tools for environmental monitoring but present a variety of issues. Hyperspectral imaging generates a significant amount of data that needs to be stored for gathering results, and most environmental monitoring occurs offline and in remote locations. This presents a need for some way to classify hyperspectral images in real time without the need to store the large data. Our solution is to create a custom imager that contains the necessary hardware for collecting hyperspectral images and implementing the image processing and classification onto the same system. This allows for the classification of hyperspectral images without the need to store any data for processing later. The development of both custom hardware for interfacing a complimentary metal-oxide-semiconductor (CMOS) sensor with a system-on-module field-programmable gate array and various peripherals alongside custom firmware and embedded software will allow for a singular embedded system capable of real-time classification of hyperspectral data in the field.

# INTRODUCTION AND BACKGROUND

## SMART FireS

The Sensors, Machine learning, and Artifical intelligence in Real Time Fire Science (SMART FireS) project is a NSF EPSCoR Research Infrastructure Improvement Track-1 project focused on gaining a stronger understanding of the process and effects of prescribed burns for helping with wildfire management in Montana. This project consists of four thrusts, each focusing on different areas of the project, with strong collaboration between them across a variety of research projects at five different universities across Montana. These thrusts are the Fire and Smoke Science (FSS) thrust, the Artificial Intelligence and Machine Learning (AIML) thrust, Smart Optical Sensors (SOS) thrust, and the Social Psychology, Economics, and Ethics (SPEE) thrust. This specific research project is part of the SOS thrust, with strong collaboration between our team and the AIML team.

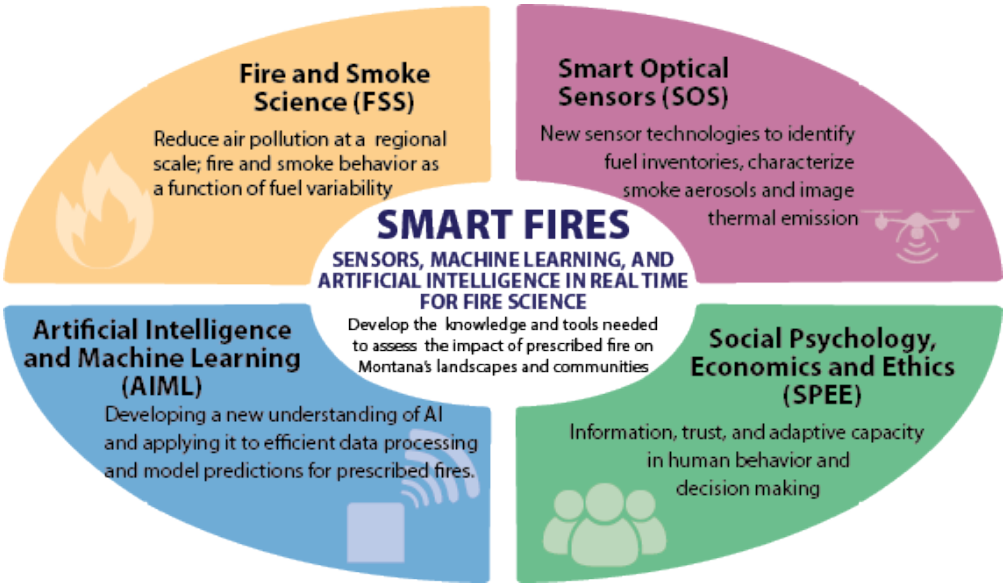


Figure 1: Graphic showing the four thrusts of the SMART FireS project.

## Wildfires

In 2024, the United States experienced 61,685 wildfires that burned 8,851,142 acres [1]. This equates to 143.49 acres burned for each fire, roughly 129 American football fields. This ranks as the 5th most acres burned per fire in the last 25 years, along with the 7th most for acres total burned and 19th most for number of fires[1]. The U.S. Forest Service and Department of the Interior estimate that over the last decade, the cost of wildfire suppression averages around $2.9 billion each year, with an expectation that this cost will rise as the U.S. feels more of the effects of rising global temperatures[4]. A striking example of this is the Los Angeles fires that occurred in early 2025, with an estimated cost of upwards of $250 billion[5]. The Office of Management and Budget predicts that the cost of wildfire suppression will increase by 40% in the mid-century and 76% in the late-century based on modeling predictions from ten different climate models [6]. These costs are only including wildfire suppression and not management costs.

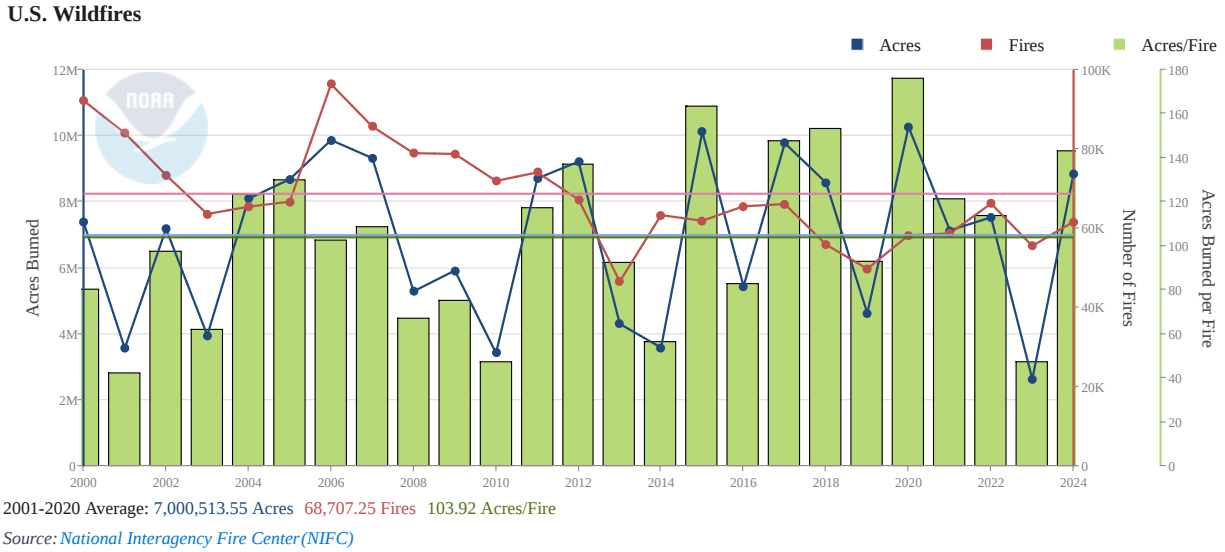


Figure 2: 2024 statistics for U.S. wildfires[1]

Although wildfires are a necessary process for the environment to help regulate itself,

only 15% of wildfires are naturally occurring, with the other 85% caused by human interactions such as unattended campfires, equipment malfunctions, or acts of arson [7]. This means that 85% of these wildfires are fully preventable with safer practices, and the other 15% can help be mitigated through the use of proactive measures such as prescribed burns.

A prescribed burn is a forestry management method that utilizes a controlled and low-intensity fire in a prescribed area at a prescribed time [2] to burn up excess ground fuel and to mitigate the effects of a wildfire if one goes through the prescribed area. When paired with manual thinning of the vegetation on a forest floor, it can have a massive positive effect on the recovery of the landscape, as shown in Figure 3. Prescribed burns are not a new concept, and indigenous people in North America and Australia used cultural burning to help maintain landscapes thousands of years prior to European colonization [8]. The Salish and Pend d'Oreille used these burns for a variety of reasons such as clearing brush, pruning berry patches, and maintaining areas [9].

While highly beneficial, thinning and prescribed burns are costly and time consuming. The need to conduct these prescribed burns before the risk of a large-scale wildfire grows too large requires tools capable of gathering valuable information. The development of these new tools would allow for more efficient prescribed burns by allowing for large-scale ground fuel characterization at a more frequent rate than is currently allowed.

## Hyperspectral Imaging

When defining an image and the 'colors' that can be used for the image, the color information can be encoded with a variety of different approaches. Multispectral imaging uses a series of different wavelengths to convey information. These wavelengths can vary in number and are often spaced about with uneven sampling. They can be as simple as a red, green, and blue image (RGB) that uses some superpixel of those three colors in order

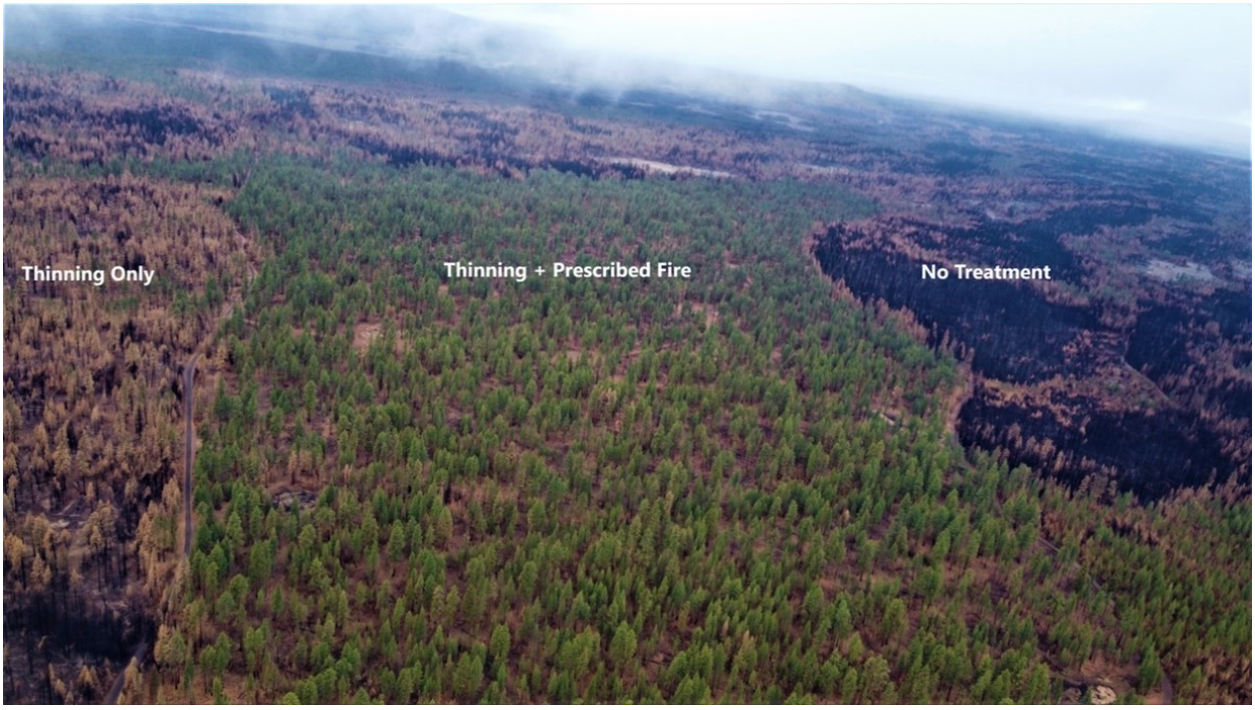


Figure 3: Example of different methods of mitigation from the Bootleg Fire in Fremont-Winema National Forest[2]

to convey a wide range of colors, or as complex as a sensor with multiple filters across a large range of wavelengths in order to gather very specific information in a non-contiguous set of wavelengths. Along with multispectral imaging there is hyperspectral imaging. While hyperspectral imaging is also just an image containing a variety of wavelengths, the hyperspectal image wavelengths are recorded in a contiguous band evenly sampled. While more wavelength bands can provide more information, the more bands that the image contains increases the size of the image in terms of data size. This larger data requires more computational power to process and more resources committed to storing the data.

The range of wavelengths being used can also vary depending on the application. Visible (VIS) hyperspectral imaging utilizes wavelengths of visible light (VIS) (to the human eye) from the range of 400 nm to 700 nm. Near infrared (NIR) hyperspectral imaging records infrared wavelengths in the range of 700 nm to 1000 nm. Once you get into the infrared (IR)

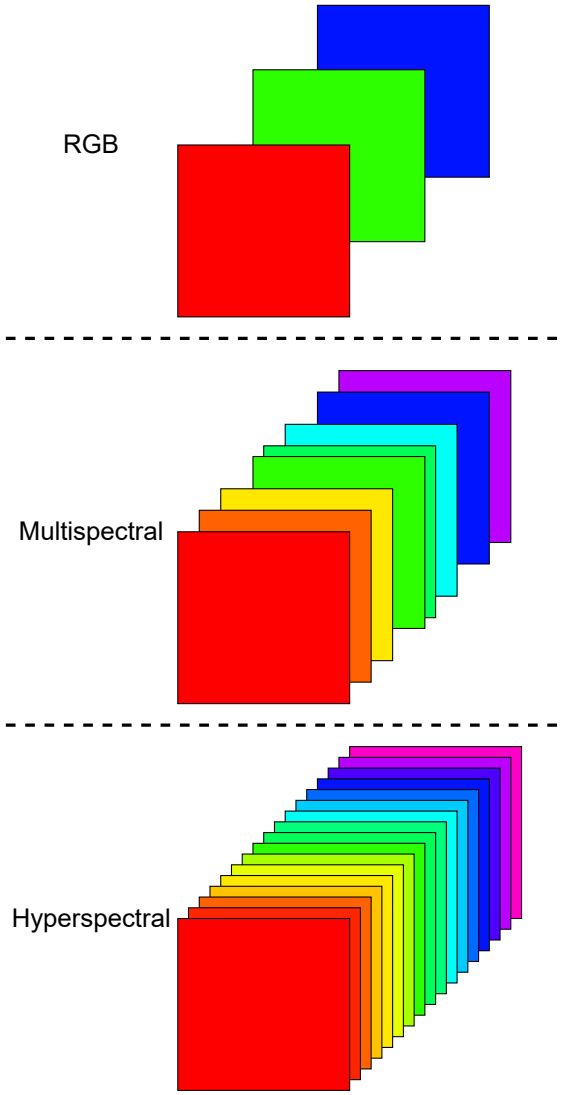


Figure 4: Difference between RGB, multispectral, and hyperspectral datacubes

wavelengths above 1 $\mu$m you get the short wave IR (SWIR) at 1 $\mu$m to 3 $\mu$m, medium wave IR (MWIR) at 3 $\mu$m to 5 $\mu$m, and long wave IR (LWIR) at 8 $\mu$m to 14 $\mu$m. The definition for all of these different wavelengths above 700 nm can also vary depending on the field that it is being applied to. The wavelength used can also introduce additional requirements as far as the materials that capture the photons. Silicon is a cheap material capable of capturing

photons in the visible to near infrared range, but once you go above the NIR range, you begin to require detectors to be made of materials such as InGaAs (Indium Gallium Arsenide).

Hyperspectral imaging has been applied to wildfires to both detect [10] and map [11][12] the forest floor both before and after wildfires. This can come in the form of hyperspectral data collected from satellites[13] like AVIRIS[14] or PRISMA[15], or from cameras mounted on drones[16]. When using hyperspectral imaging in applications that involve plant health, the target wavelengths for analysis can range from the lower end of the visible range all the way up to the SWIR wavelengths[15][17]. Analyzing the 680-750 nm range, also known as the 'red edge', can provide insight to the chlorophyll production of a plant [18]. The entire visible spectrum gives insight to the pigmentation of a plant, which is another useful feature when deciding the health of a plant [19]. Other The use of hyperspectral imaging for the characterization of ground fuel also allows for the identification of ground fuels without the need to move through hard to maneuver areas[20]. These characteristics can give us quantifiable insight related to mapping out wildfire fuel on the ground at a larger scale than with just human vision. This can also be done at a much more frequent rate than manual characterization, meaning it is easier to maintain up to date and frequently updated fuel maps.

Hyperspectral imaging has also been shown to be a useful remote sensing tool in a variety of other applications, such as precision agriculture for weed and pest detection [21], food sorting with detection of abnormalities such as bruises[22], vein detection in medical applications [23], and environmental monitoring of rivers and lakes [24].

Due to the wide range of applications that utilize hyperspectral imaging, there is a variety of commercially available hyperspectral cameras depending on the intended application use. Resonon is a local Bozeman company that designs advanced hyperspectral cameras and imaging systems. These cameras cover a wide range of wavelengths ranging from 300 nm all the way up to 2500 nm. The most comparable commercially available

cameras from Resonon to our application are the Pika L and the Pika XC2, which both operate in the 400 nm to 1000 nm range (VIS-NIR) but differ in their spectral and spatial characteristics. For our system, we are working alongside Resonon to develop a system that interfaces with optics developed by them for their products.

Table 1: Comparison of Commercially Available Hyperspectral Cameras from Resonon.

| **Camera** | $\lambda$ **Range [nm]** |
|---|---|
| Pika L | 400 - 1000 |
| Pika XC2 | 400 - 1000 |
| Pika IR | 900 - 1700 |
| Pika SWIR | 1000 - 2500 |

The optical layout for a hyperspectral camera starts with an objective lens that focuses light onto a plane with a slit in it, as shown in Figure 5. The slit is used to block all of the light except for a single line of spatial pixels. Because the sensor only receives a single line of spatial pixels, the system requires some sort of physical movement in order to gather full spatial images of a desired area. This can be done by moving what the system is looking at, such as a conveyor belt in food sorting applications, or by moving the camera itself, like in aerial or satellite-mounted hyperspectral cameras. After the light passes through the slit, it goes through a collimation optic to collimate the received light. After the collimator, the light hits some sort of diffraction element, such as a grating, which splits the incoming light into individual wavelengths across a desired spectrum. This split light then hits a final lens that focuses the light across the sensor in the system, usually a CMOS (Complimentary Metal-Oxide Semiconductor) sensor or a Charge-Coupled Device (CCD) Array. A CCD array gives a higher quality image with less noise, but is more expensive than its CMOS sensor counterpart, which also has the benefit of often having circuits built into the sensor for helping handle data from the sensor.

When applied to environmental monitoring, specifically the characterization of ground fuel on a forest floor, hyperspectral imaging has shown the ability to provide insight on the fire

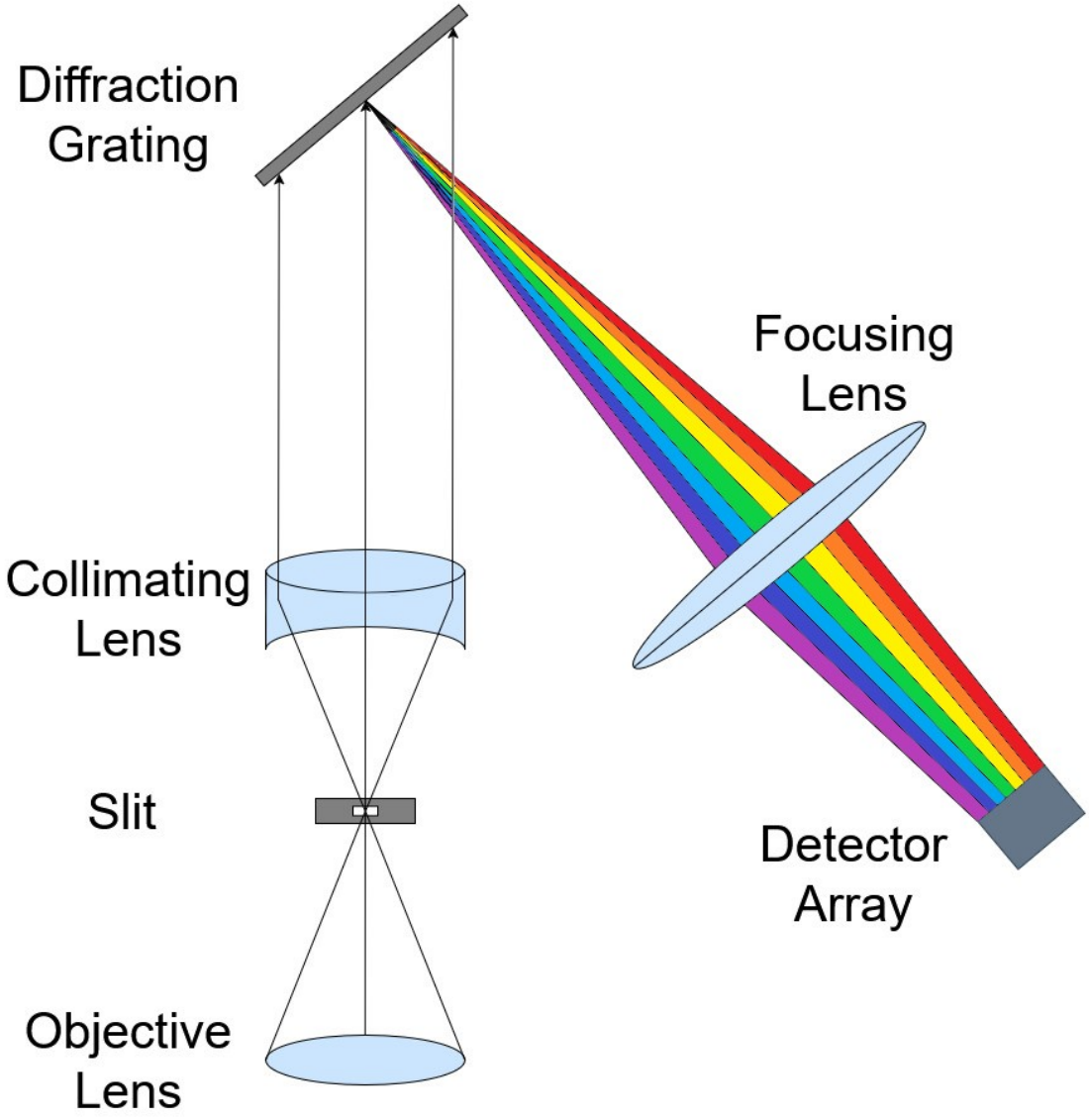


Figure 5: Typical Optical Front End for Hyperspectral Imaging

risk of the ground fuel [25]. However, in such a time-critical and highly variable environment, the drawbacks of hyperspectral imaging such as big data and long post-processing times are emphasized. The conditions can vary highly between days, resulting in gathered data becoming obsolete if weather conditions change rapidly before results are collected. This shows a need to accelerate the process of gathering data and obtaining results from that data. This acceleration can be accomplished by implementing both the processing and the

computation for classification into a single advanced system built using a custom piece of hardware such as a field programmable gate-array.

## Field-Programmable Gate Arrays

Field-Programmable Gate Arrays (FPGAs) are integrated circuits created with an array of components all connected by a reprogrammable interconnect, allowing for the user to implement custom hardware designs using some combination of the system components. These components are usually made up of lower-level hardware such as complex logic blocks, which can contain registers and lookup tables (LUT), and digital signal processing (DSP) slices, which are usually made up of a multiplier, adder, and a flexible arithmetic logic unit (ALU). This set of different blocks and the interconnect between them is often referred to as the FPGA "fabric" or programmable logic (PL).

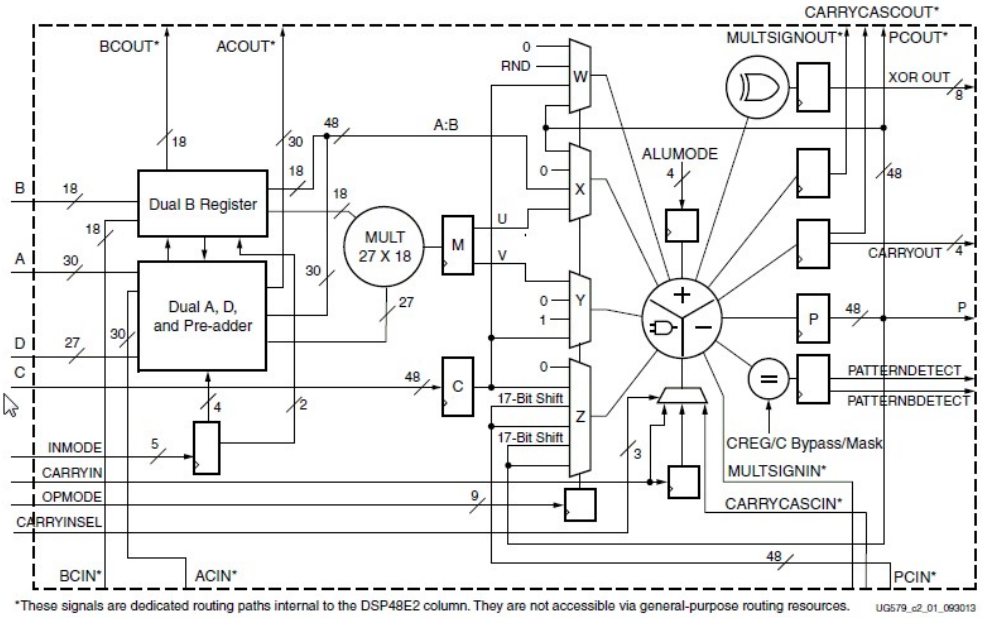


Figure 6: DSP48E2, the DSP Slice Found in Xilinx Ultrascale+ Architectures. Image from Xilinx[3]

The nature of the FPGA fabric allows for the streaming of data without the need to

buffer and store it. Paired with the ability to implement high speed computations on data using the DSP slices and complex logic blocks in the fabric, this makes FPGA's an ideal piece of hardware for applications that require fast signal processing in a high-speed and high-bandwidth environment. Another huge benefit for FPGA's that allow them to excel in real-time applications is the parallel nature of the hardware. Every element of the FPGA can be active at the same time and running in parallel, which means that by increasing the amount of hardware resources, you can also increase the speed at which the processing is done. All of these parallel components can also run off a single clock, making it easy to synchronize them together.

The programming of the FPGA is handled through the use of a Hardware Description Language (HDL) such as VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) or Verilog/SystemVerilog. While these can be somewhat similar to some more traditional scripting or scientific programming languages, HDLs are often much lower level with Register Transfer Level (RTL) coding, and really are more seen as a description of what physical hardware components are doing at any given clock cycle. There are some high-level synthesis (HLS) tools that can be used to generate HDL from something such as C or C++ code, but these aren't always the most resource efficient, and are seen more as time efficient. The mapping from sequential languages to parallel computations can also be somewhat problematic as they don't often translate as well as some people might imagine.

System-on-Chip and System-on-Module FPGAs

While most of the benefits and reasons for using an FPGA in an application are rooted in the versatility and efficiency of the programmable logic, some systems can benefit from having a functioning computer system alongside the programmable logic. For some applications, the implementation of a softcore processor focused on a specific application is all that's

necessary. This is a CPU implemented in the FPGA fabric using a combination of DSP slices, LUTs, and RAM. While this can do some more specific work, it's less capable of running more general purpose software written in high level languages like C, Python, or Matlab. For systems that would benefit from the capability of running software like that, you need a more general purpose CPU akin to those found in mobile phones. When there is a general purpose CPU implemented into the silicon alongside the programmable logic, the processor is referred to as a hard processor system (HPS). A System-on-Chip (SoC) FPGA is a FPGA with both programmable logic and a HPS together on the same silicon die.

This HPS is capable of running general purpose operating systems, usually in the form of some custom Linux kernel. This allows you to compile and run software on the FPGA that doesn't translate well to the FPGA fabric. This also allows for a simple interface for interacting with the PL. While it can be beneficial for the inclusion of a HPS into the system, it increases the complexity of the system due to the interconnect between the PL and the HPS. This complexity can come in the form of timing closures or device driver implementations, software that tells the hardware how to interact with software on the system.

Another step up in the complexity of the system is the inclusion of external dynamic random access memory (DRAM). DRAM allows the user to store and retrieve data based on a memory address. This memory is larger, cheaper, and consumes less power than its static random access memory (SRAM) counterpart in the FPGA fabric. This DRAM allows the FPGA and HPS to store data without using RAM implemented into the FPGA fabric which often consumes more power for less space. This external DRAM can be shared between the HPS and PL or each of the two can have their own independent memory devices. The need for external memory requires a PCB that includes a SoC FPGA and is what is referred to as a System-on-Module (SoM) FPGA. The SoM FPGA can be thought of as a more out of the box solution for implementing designs on a SoC FPGA, with the designers creating a

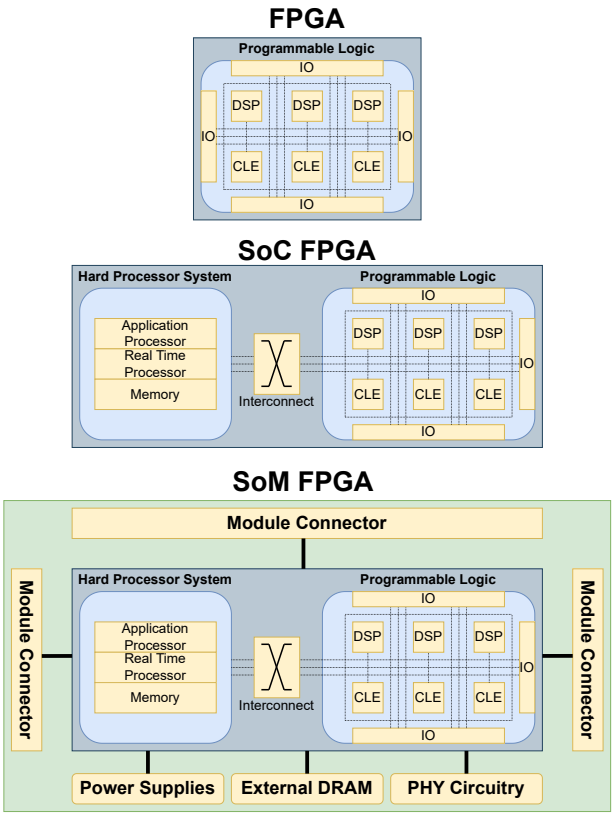


Figure 7: Comparison of different types of FPGA's

board with the connections between the SoC and external circuitry such as DRAM and PHY layers for peripherals. While this helps to kickstart the design of a system by allowing faster use of peripherals such as Ethernet, it still increases the overall complexity of the system significantly.

With a basic understanding of both hyperspectral imaging and FPGA's, one can get a better picture of how the two might fit together. I will elaborate more on the current methodology for hyperspectral imaging and then discuss how our proposed methodology will accelerate the workflow of both processing the data and classifying it as well.

## Current Hyperspectral Data Processing Methodology

The current methodology for hyperspectral image processing can be split into two steps: the data collection and the postprocessing/machine learning stages. Both of these steps can take a significant amount of time to accomplish.

### Data Collection

Hyperspectral images can be collected with a variety of devices, but airborne systems that utilize systems such as satellites and unmanned drones are able to collect significant amounts of data when compared to ground-based systems. While satellites are able to gather information on larger areas, they lack the finer spatial resolution, have less frequent passes over areas, and can have the ground blocked by clouds. Drone-based systems aren't able to cover as much area, but have much finer spatial resolution and can collect data over an area more frequently. Drones also allow the attachment of external storage devices to contain the data collected during a flight. One downside to the aerial systems is that in forested areas with a thick canopy, the aerial imager would not have the ability to properly characterize the ground fuel. One goal of the project is to deploy the imager on a ground vehicle in order to gather data from beneath the canopy.

The large size of the hyperspectral images lead to the requirement of large storage devices in terms of data storage. Using the assumption of a 200 color band system with 8-bit precision and only 1 M pixels, a full image would be 190 MB. If that was increased to 400 bands with 12-bit precision, the resulting image would be around 600 MB. For comparison, a similarly sized RGB image would only be 36 MB. This significant data size presents an issue for both the storing and the processing of the data, as it takes more than a single image to create a meaningful dataset.

### Postprocessing and Machine Learning

Once the data has been collected, it can either be used to train a new model that hasn't been created yet, or it can be used for classification or inferencing with a model that has already been trained. If you are trying to train a new model, you'll likely need more data than if you were simply trying to do inferencing over an area. In the case of real-time inferencing, the goal is to avoid storing any data and instead process it in real-time.

This postprocessing normally entails a conversion from radiance to reflectance with either a reflection tarp or a downwelling irradiance sensor. This reflectance data can then be used as training data if a model doesn't exist, or it can be fed into a model that has already been trained in order to gather results.

## Proposed Methodology

The various drawbacks to the current methodology leave room for improvements in the pipeline of data collection to classification. Our proposed methodology seeks to streamline the system and combine all parts of the hyperspectral imaging pipeline into a single system. This system will require the ability to stream in large amounts of data, do processing on the wavelength vectors of the hyperspectral images, and classify them used an embedded machine learning model, all while functioning in 'real-time'. In the case of a smart hyperspectral imager, the 'real-time' timing means being able to keep up with the frame rate of the CMOS sensor and to process all of the data in a frame before the same line in the next frame.

Since the proposed machine learning model for the ground fuel characterization is a 3D convolutional neural network (CNN), there is a significant amount of data that needs to be stored in order to hold the weights for the convolutional kernel, which is the reason why a SoM is particularly useful. The amount of weights necessary for the model wouldn't fit in RAM in the fabric, so the external DRAM allows for the storage of the weights that can

be grabbed when doing the necessary computations on the data using the kernel. Beyond storing the weights, the other math for accelerating the model can simply be implemented with DSP slices and LUT's for nonlinear functions.

The implementation of the system onto an FPGA will allow for the high speed streaming of large amounts of data due to the nature of the FPGA's fabric. Instead of capturing and storing hyperspectral images for classification, the system will stream in a certain amount of data before feeding it into a classification model. Old data will then be overwritten by new data that requires a circular buffer of hyperspectral data. This will also require the creation of a custom printed circuit board (PCB) to allow for the interfacing between the FPGA and a CMOS sensor used for capturing the hyperspectral images. This circuit board will also require a variety of peripherals connected to the FPGA that allows verification of the system.

Alongside the streaming of the data, the FPGA must be capable of processing the images and then utilizing an embedded machine learning model for the inference of the hyperspectral data. Implementing the processing into a pipeline capable of being parallelized will allow the system to keep up with the throughput of the data capture. The FPGA is also capable of implementing a machine learning model into the fabric, but will require some distillation of the model, a process that requires both pruning and quantization. Pruning is the removal of nodes or parameters through a variety of methods, while quantization is the decrease in the size of the numbers in the machine learning model. Most machine learning models utilize double precision floating point, which is a 64-bit number split up into a sign bit, 11 exponent bits, and 52 bits for the mantessa (the precision of the number). FPGAs struggle with using floating point numbers, which requires the conversion of the floating point to a fixed point value. A fixed point value can be any number of bits and has a fixed radix point in front of trailing fractional bits. The more fractional bits, the higher the precision of the number. Quantization involves the change from double precision floating point to fixed

point, and then a decrease in the fractional length until the classification error falls below a certain threshold.

The use of a SoM FPGA will also be beneficial in allowing the use of software that isn't ideal for the FPGA fabric alongside external memory capable of storing the results or weights for the embedded machine learning model. The use of a Linux operating system will also allow the implementation of simple userspace interfaces for interacting with the hardware. The FPGA also allows the system to come in a compact package with high performance. The HPS and the fabric will also allow us to partition the problem into different components that function better using one type of hardware as opposed to shoehorning the problem into a fabric only solution or a CPU only solution.

Using FPGA's for various parts of hyperspectral imaging is not a novel concept. FPGA's with hyperspectral imaging have historically been used for compression[26], real-time target detection[27], and acceleration of machine learning models for classification[28]. While these systems are useful for accelerating certain processes in the hyperspectral imaging pipeline, our methodology seeks to create a single embedded system capable of both collecting large amounts of data and accelerating the inferencing of the data.

## Broader Impacts and Applications

Along with its usage in ground fuel characterization, the real-time inferencing performed by the hyperspectral imager can be applied to a variety of commercial applications simply by replacing the inferencing model embedded into the FPGA fabric, assuming that the applications also operate in the same visible to near-infrared range that ground fuel characterization operates in. High speed sorting that utilizes hyperspectral imaging is a market that would heavily benefit from the real-time inferencing that our system is capable of, such as in applications of food sorting.

# 17

# CMOS SENSOR PRINTED CIRCUIT BOARD REVISION 1

In order to create a smart hyperspectral imager, there needs to be hardware capable of connecting a CMOS sensor to a FPGA. While FPGAs often utilize development boards for creating prototypes, development boards don't have a CMOS sensor integrated into the boards, nor do CMOS sensors have a simple interface connection for plugging into a board. This leads to a requirement that a custom PCB must be developed in order to create a system prototype for the hyperspectral imager. The focal point of this board is the connection between a CMOS sensor capable of capturing wavelengths in the visible to near-infrared range and a FPGA capable of implementing all of the requirements in the system ranging from the streaming of data and the embedding of a machine learning model into the fabric. Alongside the connection between these two pieces, there needed to be a variety of peripherals for the FPGA or the CMOS sensor to interact with. These will allow for easier debugging, verification, and interfacing with components necessary for a field prototype later down the road.

Due to the nature of the physical connections for the FPGA and the CMOS sensor, these will remain constant for the system prototype. A change in the sensor would require the board to be redesigned depending on the pin layout of the new sensor and the required connections to the FPGA, while a change in the FPGA would require different physical connectors for mounting the FPGA onto the board with the sensor. Theoretically you could replace the FPGA with another FPGA that has the same module connectors, but it would also require specific pins on the two boards to match such as the power pins or the reserved pins for specific peripherals interfacing with the HPS on the FPGA.

## Sony IMX425LLJ-C

The first component that was selected for the system was the CMOS sensor used for capturing the hyperspectral data. It's important to note that for hyperspectral imaging, the sensor only needs to be sensitive over the range of wavelengths that the system is targeting. Since this system operates in the visible to near-infrared range, we can get way with an easily accessible and cheap CMOS sensor with silicon as the main substrate. We then want to find a sensor that has a high number of pixels, a high pixel area, high electron well depth, high data throughput, a good frame rate, and an analog to digital converter (ADC) with large bit lengths.

A higher number of pixels allows for both better spatial resolution and spectral resolution when the system is operating. The more rows that you have on the sensor active area, the more spatial pixels that the system is able to capture. The higher number of columns allows the light at the target wavelengths to be spread across a larger area. This in turn will increase the spectral resolution by allowing each column to contain a shorter range. As an example, if your CMOS sensor only has 500 columns for the VIS-NIR (400 nm to 1000 nm), your spectral resolution would be 1.2 nm per pixel. If increased to 1000 pixels per row, the spectral resolution increases to 0.6 nm per pixel.

The pixel area, electron well depth of a sensor, and the frame rate are all related to how well a sensor is capable of capturing photons. A higher pixel area is capable of capturing more electrons, while the electron well depth is the amount of electrons that the sensor is capable of capturing in a single pixel before beginning to saturate. The frame rate is the number of frames (in seconds) that the sensor is capable of capturing. A higher frame rate will lead to less time that photons are collected in the pixels before being emptied. All of these parameters have an impact on the strength of the signal captured along with the noise generated by the sensor for the system.

The ADC bit length determines the size of numbers that an analog signal can be represented as. This output from the ADC can be referred to as a digital number (DN) and depends on how many bits the ADC hardware is capable of generating. A bit length of 8 bits means that the analog signal is represented by $N \in 2^8$, or 0 to 255, while a bit length of 12 bits allows the signal to be represented with $N \in 2^{12}$, or 0 to 4096. The higher bit length results in a better resolution for the system, which is $Resolution = \frac{R_{sig}}{2^B}$, where $R_{sig}$ is the range of values that the analog signal can be represented by and $B$ is the bit length of the ADC.

With all of these different parameters in mind, the CMOS sensor chosen for the system is the Sony IMX425LLJ-C. This decision was made after consulting with Resonon, who is handling the hyperspectral optics front end of the system. The IMX425LLJ-C has 1.76 M pixels (1604x1100), a 9 $\mu m$ by 9$\mu m$ square pixel area, and an electron well depth of 80,000 $e^-$. This electron well depth was not present in the datasheet for the CMOS sensor and is based on an assumption. The sensor has a sensitivity range of 400 nm to 1000 nm, with the peak sensitivity around 580 nm and a minimum relative sensitivity of around .08 at 1000 nm as shown in Figure 8. The sensor also has a variable bit length of 8, 10, or 12 bits on the ADC output and a variable frame rate depending on different parameters in the sensor's configuration. These configuration registers are written to or read from with simple serial communication interfaces such as serial peripheral interface (SPI) or inter-integrated circuit (I2C). The sensor can be configured to have a different number of low voltage differential signaling (LVDS) outputs (2,4,8) for the data which, alongside the bit length of the ADC, impacts the frame rate of the system. Using 8 LVDS channels alongside a 12-bit ADC results in a frame rate of 180.6 frames per second with a data throughput of 4.752 Gbps.

The system also has two different operating modes: a scalable low-voltage signaling with embedded clock (SLVS-EC) mode and a non-embedded clock (SLVS) mode. While the SLVS-EC mode is capable of much higher data throughput from the sensor, it comes with a

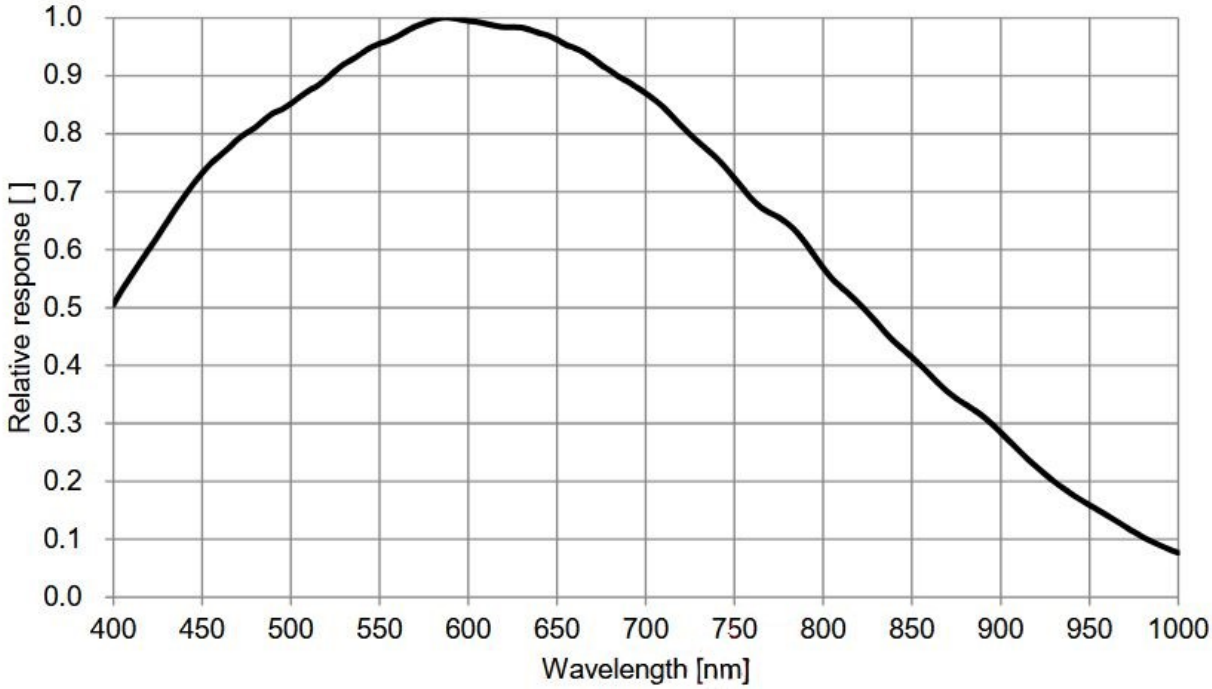


Figure 8: Spectral sensitivity of Sony IMX425LLJ-C CMOS sensor

Table 2: Frame rate and data rate of IMX425LLJ-C with varying SLVS channels and 12-bit ADC.

| **SLVS Ch. #** | **Frame Rate [FPS]** | **Data Rate [Gbps]** |
|---|---|---|
| 8 | 180.6 | 4.752 |
| 4 | 96.6 | 2.376 |
| 2 | 50.1 | 1.188 |

higher degree of difficulty for implementing the clock extraction from the data. As a result, the system will currently utilize the sensor on the regular SLVS operating mode, which has a clock line for the data as opposed to embedding the clock in the data. This also requires that the data lines be equal length on the physical board connection, as opposed to SLVS-EC that doesn't require the traces to be equal lengths [29].

Real-Time Constraints

With the time constraint for real-time processing stemming from the data throughput of the system, there is some control over what actually qualifies as 'real-time'. Since the

CMOS sensor allows us to configure the number of LVDS output channels and the ADC bit depth, we have three different options of data throughputs with nine different frame rates (three for each data rate). This allows us to modify the throughput of the system depending on the latency of the processing pipeline from input to inference. We can also develop hardware capable of extracting the clock from the data, which would give us faster frame rates and a higher data throughput. The decision for which data throughput to use depends on the latency of the accelerated pipeline for the inferencing process.

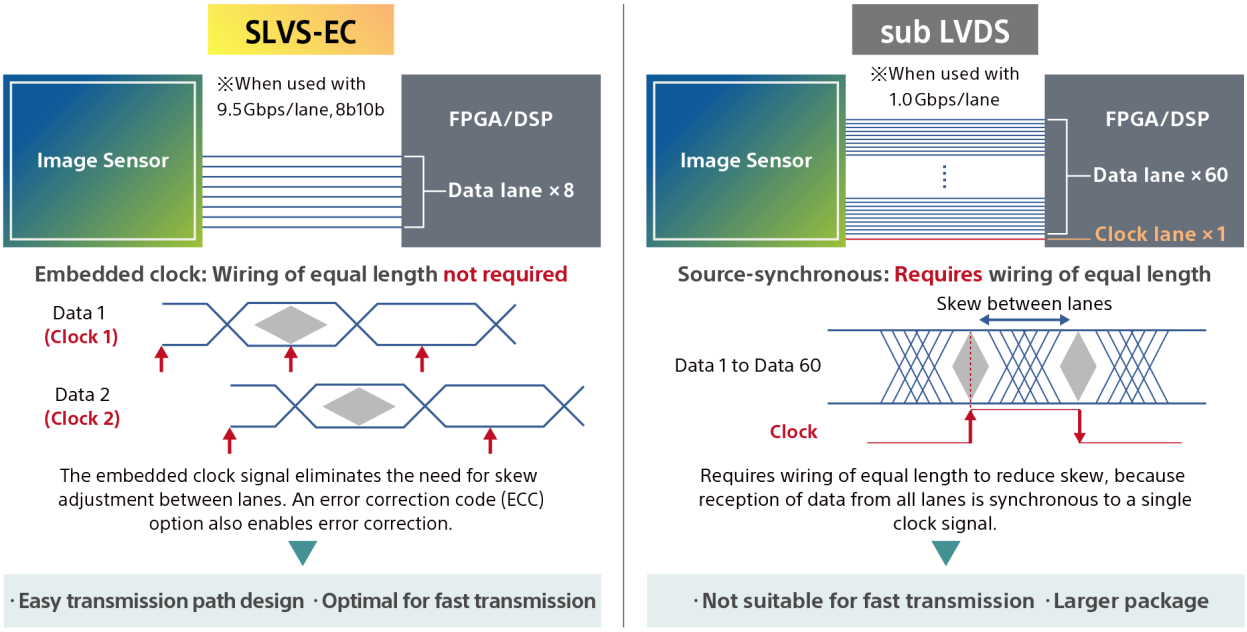


Figure 9: Diagram for the EC and non EC operating modes

## Enclustra Mercury+ XU1 SoM

After picking out the CMOS sensor for the system, we needed to determine the FPGA that would get used for the system prototype. While for most commercial products you would want a FPGA that has similar resources to the requirements of the system, the development of a working prototype wants as many resources as possible to avoid dealing with low level optimization while just trying to get a working system. This helps to avoid timing closure

issues that would crop up if you start using too many resources or even running out of resources entirely. When discussing the amounts of resources in an FPGA, one usually refers to the number of DSP slices and complex logic elements. Alongside these resources come a variety of different additions that a FPGA is capable of having. These include having a HPS for the system like in SoC FPGAs or external DRAM connected to the board such as SoM FPGAs. FPGAs can also include peripherals such as video codec units, GPUs, varying numbers of application processor units (APUs) or real time processors (RPUs), and various high speed interfaces such as USB, Ethernet, or PCIe. While it can be beneficial to have a chip that has all the different bells and whistles, this comes at a price. This means there must be a balance between the amount of resources and the price of the board. Purchasing a board also often requires a development board alongside the FPGA. This allows the development of projects on the FPGA chip with known working hardware as opposed to developing only on a custom circuit board.

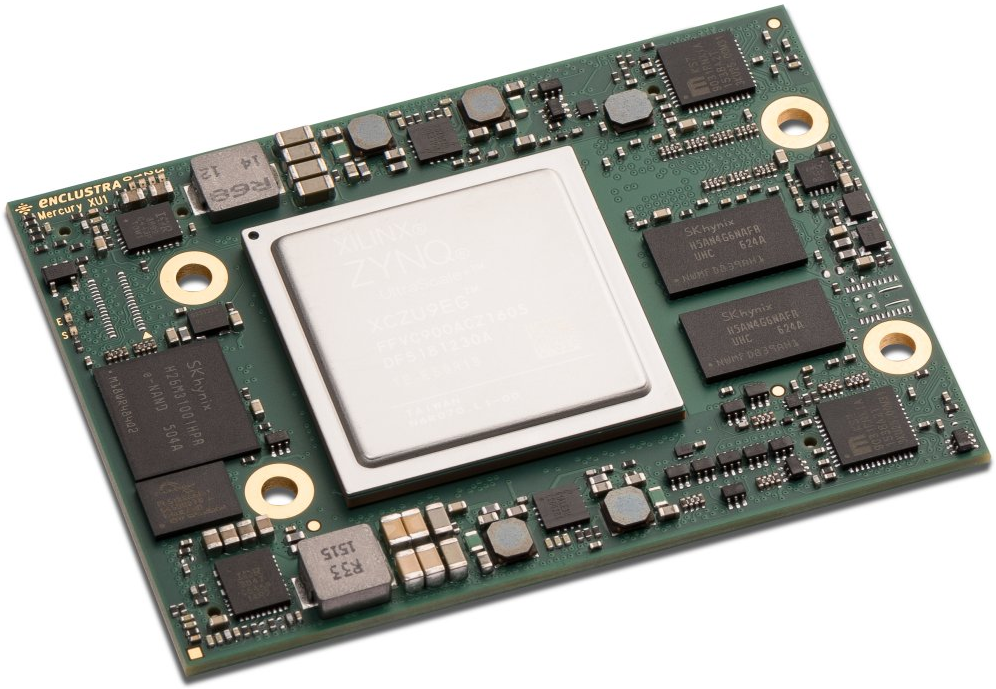


Figure 10: Enclustra Mercury+ XU1 SoM

When deciding which FPGA to choose, we had a few pieces in specific that we were looking for in a board. The first being we wanted to use a SoM FPGA. The external DRAM would allow for storage of data used in the embedded machine learning model or calibration values without using SRAM in the fabric of the FPGA. This SoM would also include a HPS that would allow us to run Linux on the system. Our next requirement for the board was to be a good value relative to the number of DSP slices in the chip. This would allow us the most room to implement the processing of the vectors and the embedded machine learning model custom accelerators. Beyond that, any extra peripherals on the board were a bonus for the project and could allow us to experiment with the system further down the road. Since we wanted to also use a board with good software tools, we were also looking at either an Altera or Xilinx FPGA, who are the two biggest market shares in the FPGA market.

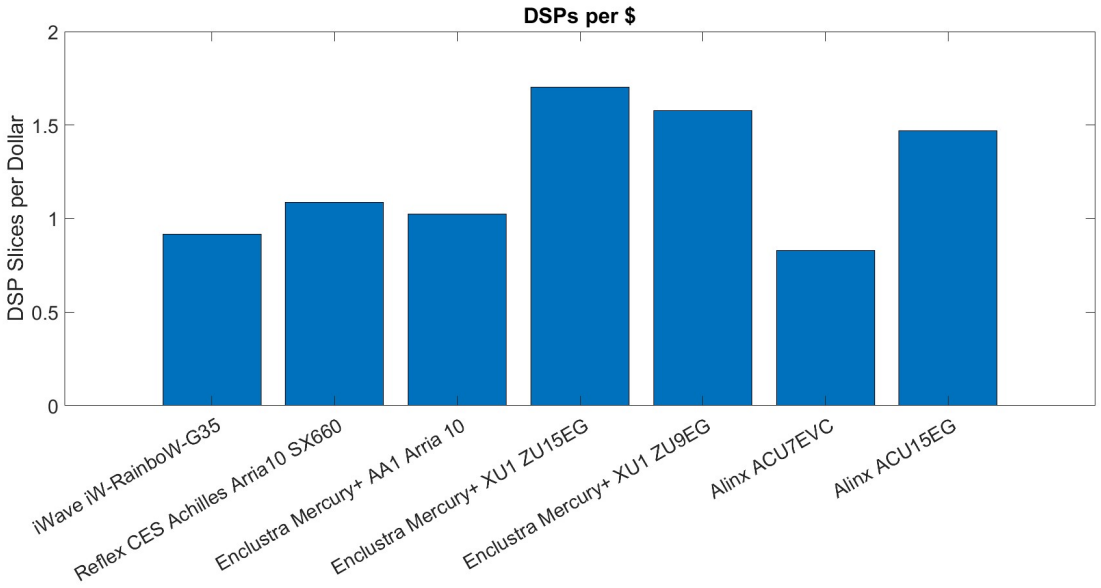


Figure 11: Comparison of the number of DSP slices per dollar for various SoM FPGA's

After looking at a variety of vendors and their chips, we landed on the Enclustra Mercury+ XU1 SoM FPGA. This stems from the amount of resources available in the programmable logic along with the value in terms of DSP slices per dollar shown in Figure 11. It contains a Xilinx MPSoC with a four core ARM Cortex A53 APU, a two core ARM

Cortex R5 RPU, and an ARM Mali-400MP2 GPU, and several high speed peripherals. This FPGA comes in a handful of packages with different amounts of resources depending on the specific FPGA chip used in the package and as shown in Figure 11. The model we chose used a Xilinx Zynq Ultrascale+ MPSoC ZU15EG, which contains 747,000 logic elements, 3,528 DSP slices, and 4GB of external DRAM. This high amount of resources gives us the best chance at prototyping a full system without beginning to run into resource limitations, especially in terms of DSP slices.

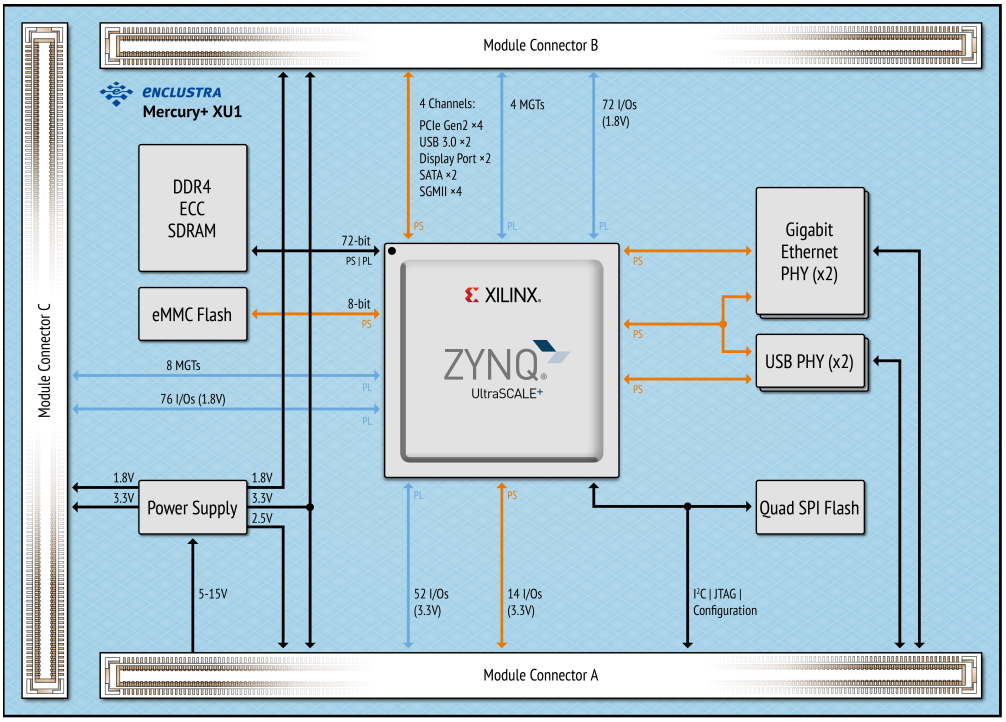

Figure 12: Diagram of the peripherals on the SoM board

Alongside the Mercury+ XU1, we also got the Mercury ST1 development board. This board contains physical connectors for a variety of peripherals that include a mini DisplayPort connector, a HDMI connector, a RJ45 Ethernet jack, various USB connectors, a micro SD card holder, and a SFP+ connector. These connectors allow for interfacing with the FPGA in a variety of ways alongside having working hardware to debug issues that may arise when implementing projects onto the CMOS development board. This board also

works with any Enclustra Mercury boards, allowing us to change the FPGA that we use down the road without having to buy new development boards.

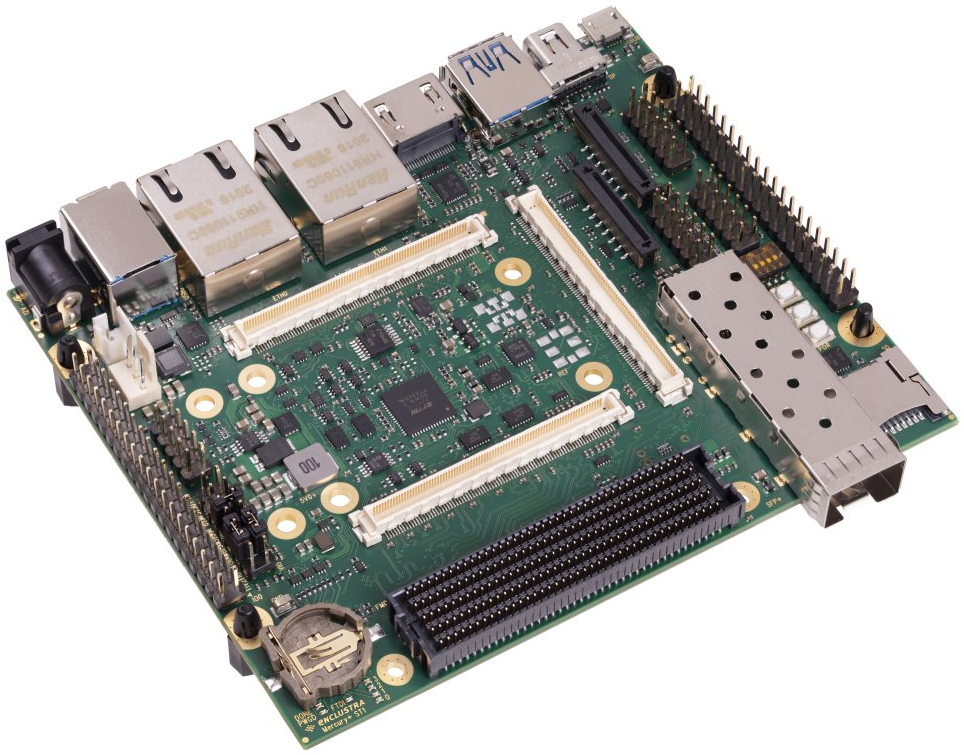

Figure 13: Enclustra ST1 baseboard

## Circuit Board Peripherals

Now that we have our primary components selected for the development board, we then needed to determine the peripherals for the board that we want for development of the imager prototype. These would allow us to implement and verify different parts of the system before scaling down the board to a more compact board intended to function as a field prototype. Alongside these peripherals, there will be smaller components intended for the functionality of the board and for configuring the board. We decided that the development board should include a DisplayPort connector, an Ethernet jack, a USB connector, micro

SD card reader, a GPS, an inertial measurement unit (IMU), a JTAG header, configuration pins, and a handful of general purpose IO (GPIO) connectors.

The DisplayPort connector allows us to start by developing a passthrough component for the CMOS sensor, which would enable verification for the functionality of the sensor and to be the first step in developing an imager. The Ethernet and USB connectors were chosen for interfacing other devices for functions such as data transfer or for setting up external servers for the root file system or TFTP server of the FPGA. The SD card reader allows us to use an SD card to contain the boot image and file system for the FPGA that will start on bootup and for storing data that we would want to access on another device. The GPS and IMU are for developing the interfaces for these two, as a future field prototype will require both the GPS and IMU data for determining where the data is actually located when the imager is slightly off axis. The JTAG and configuration pins are for configuring and booting the board through different modes, and the GPIO pins can be used for debugging purposes or simpler interfaces with hardware. Several of these decisions were also influenced by the FPGA having dedicated pins that connect directly to the HPS, allowing for easy integration between the connectors and the HPS without using fabric resources for interface controllers.

## Schematic Design

Once all the parts were selected for the board, the schematic for the board then had to be created prior to the routing of the components. This involves reading many pages of datasheets for components to determine where every pin needs to be connected for allowing for full functionality. This includes having to find specific pins for the FPGA that can function as clock pins, differential pair pins, high data throughput pins, or reserved pins on the FPGA. Going component by component from the most critical (CMOS sensor) to the least critical (GPIO headers) allows for the most choices for using pins as opposed to trying to fit in connections for a critical component with limited options.

Another helpful boon for the schematic design is by using reference designs from other schematics like the Mercury ST1 baseboard. This allows for having a known schematic for different components that can work and leads to a lower likelihood that there is an issue with the component on the first run of the board. Minor modifications must be made depending on different parts that may be used due to price or availability reasons.

This step in the design process ultimately boils down to reading reference designs and dozens of pages of datasheets in order to simply connect one point to another.

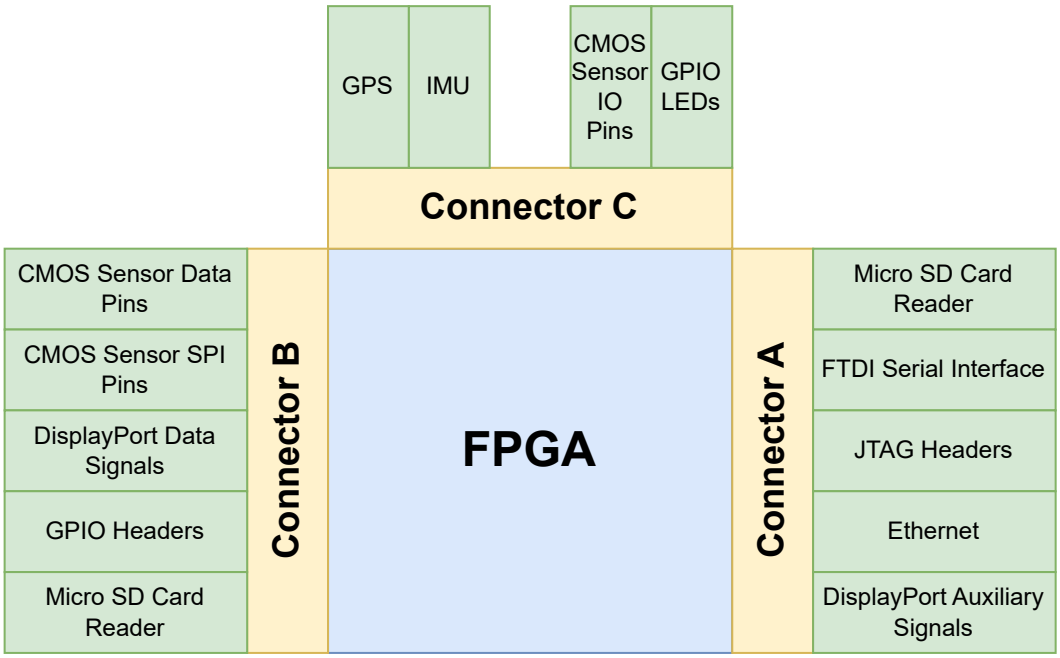


Figure 14: Top level overview of the first revision of the PCB

## Circuit Board Design

Once all of the schematic connections are complete, the process turns from pages of reading into a puzzle of fitting together different components and their traces without overlapping and while maintaining signal integrity for the system. Similarly to how the

schematic was done starting with more important components, it's even more imperative that the routing of the most critical and complex components is done first. Often these components like the CMOS sensor and the DisplayPort contain traces that require length and impedance matching on differential pairs. This ensures that data for the same clocks arrive to their destination at the same time synchronously. This can also lead to a significant increase in the space that the routing takes up due to the curves that the traces have to follow in order to get the lengths matched. Often after beginning to route a component on a circuit board, you'll realize that there are better pin connections that would fit for the space on the board with where you want to place the component. This was the case with pretty much every component on the board, as I went through and redid the schematic for almost all of the components after beginning the routing.

The circuit board was created as a 6000 mil (6 in.) by 4000 mil (4 in.) board of FR4 material with four layers. The outer two layers were one oz. copper pours reserved for the signal connections and allowed for an easier time with routing signals that would have overlapped had the signals only been kept to a single layer. The inner two layers were 2 oz. copper pours with one layer reserved for a common ground between all of the components on the board and the other reserved as a power layer that would be split between polygons of different voltage levels. This required placement of components that were similar operating voltage levels near each other to avoid having multiple split planes of the same power level.

Due to the difficult nature of some of the components, namely the Sony CMOS sensor being a 234 pin line grid array packed into a 24.5 mm by 21.4 mm package, we decided to opt against manually assembling the board and instead had Out Of The Box Manufacturing print and assemble the circuit board for us to minimize the chance that the board doesn't function due to an assembly error.

Routing Considerations

While some pins on components for a circuit board can be connected with minimal considerations, other pins might require specific layouts to account for impedance, timing, or temperature characteristics. Differential pairs require pairs of traces to be equal lengths to allow for the positive and negative values on the traces to sync up properly when clocked in with each other. These traces must also be separated from one another to minimize crosstalk between traces carrying high speed data. Higher speed connections require isolation for the traces to prevent noise from interfering with the data being sent over the trace. Traces also require impedance matching that depends on the component and is affected by the trace width, the trace gap, and the physical characteristics of the layer that the trace is on such as the material and the weight of the layer.

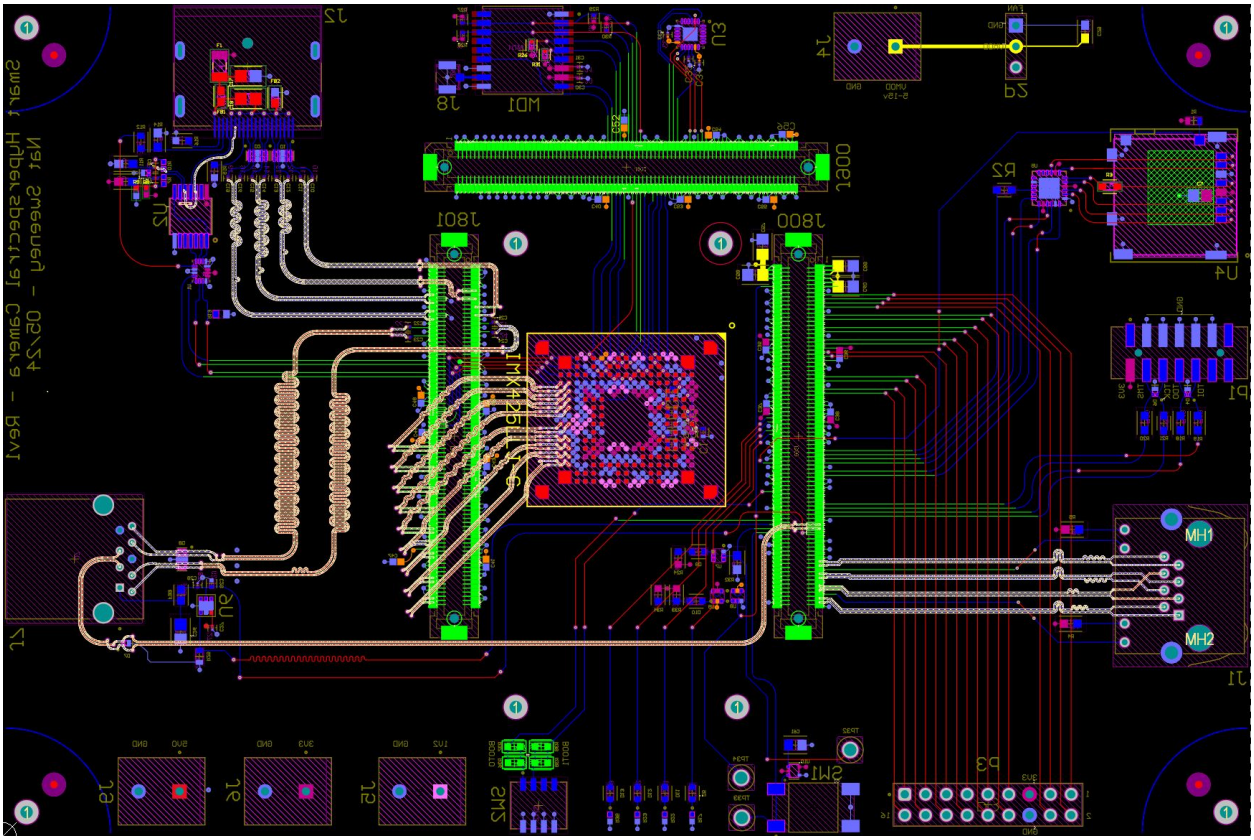


Figure 15: Full view of the first revision of the circuit board

CMOS Sensor

The physical connection for the Sony IMX425LLJ-C comes in the form of a 226 pin line grid array (LGA), with data pins, clock pins, signal pins, three levels of power pins, and separate digital and analog ground pins. Many of the pins are also designated as 'No Connect' (NC) pins, and should be left open on the board and not connected to ground for stability or thermal reasons. To start with the routing, I placed the CMOS sensor in the center of the board on the opposite side of the connectors used for mounting the FPGA onto the board. I then started by branching out the data and data clock pins to their designated connectors determined in the schematic stage. I started with the pair that had to go the furthest difference to minimize that, which would in turn reduce the amount of length matching required by the other 7 pairs of data outputs and the clocks. Since the CMOS sensor has a 30-60 Ω single ended output impedance, the trace impedance will need to be calculated in Altium in order to get the proper trace width and spacing between pairs.

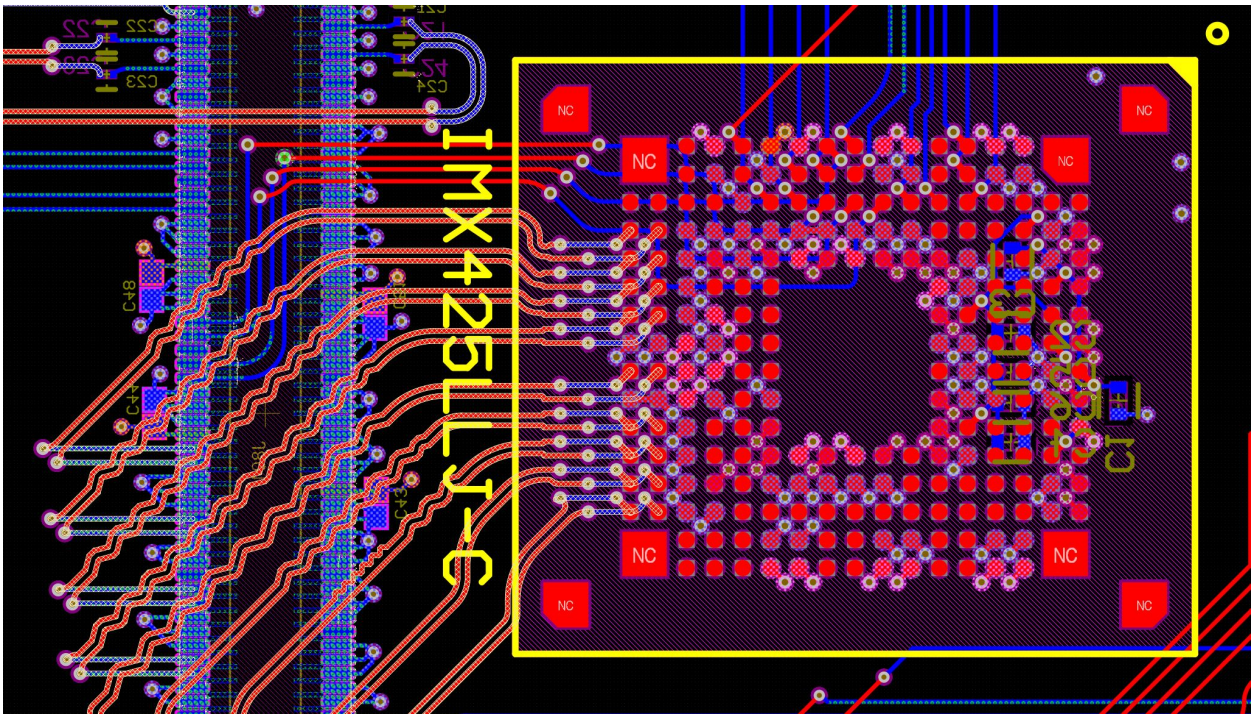


Figure 16: Close view of the Sony IMX425LLJ-C routing

After some different attempts with different turns, I was running into an issue where the mounting holes for the FPGA were causing some unnecessarily long traces in order to have a large enough keepout around the hole. After getting the data pins connected in a functional manner, I decided to return to the schematic and see if I could change which connector pins were connected to the data pins of the sensor that would avoid going by the mounting holes. This also required that the new pins for the sensor had all of the pins for the sensor on the same IO bank for the connectors to avoid any unnecessary noise. Eventually I found a good solution that shortened the trace length for the data pins significantly along with decreasing the amount of length matching that would be required by rotating the sensor and using a different connector than what was originally chosen. This did result in the traces for the signal and IO pins of the sensor to be longer, but this is fine considering none of these pins are high speed connections and are meant for serial communication and configuration of the sensor. I then did the connections for the pins that required a connection to bypass capacitor.

After fanning out all of the necessary connections from underneath the sensor, I then began connecting the voltage and ground connections. This involved using a 'dogbone' technique in order to connect the pins to vias that go to the proper power plane while keeping enough clearance between the vias and adjacent pins. This involves placing a via diagonally adjacent to the pin and connecting it with a thin trace, hence the name due to the resemblance to a dogbone.

DisplayPort

The next component to be done was the DisplayPort connector. This peripheral allows for the displaying of video and graphical data onto a monitor, similar to HDMI. The reason this was chosen over a HDMI connector is due to the royalty fee associated with HDMI connectors as oppposed to DisplayPort connectors. Similarly to the CMOS sensor, this

connector had several differential pairs that needed to be length matched. The HPS on the FPGA also has dedicated pin connectors for the DisplayPort, meaning that there isn't an option to change which pins are connected to which in order to clean up the routing. This meant placing the DisplayPort connector as close as possible to the necessary pins to avoid the differential pairs 'blocking' other pins from being connected, similarly to Tron. The connector also needs extra room for additional circuitry such as a LVDS transceiver and level shifter for the AUX and hot-plug detect (HPD) pins. The differential pairs also need to be connected to AC coupling capacitors and flyback diodes for electrostatic discharge (ESD) protection. The impedance matching for the DisplayPort connector is also 50 Ω single ended, meaning that the same trace width could be used for both the sensor and the DisplayPort since the 100 Ω differential pair impedance works for both.

USB

A universal serial bus (USB) type A host consists of three differential pairs with dedicated pins for the processor system (PS) to USB connections. The USB connector also required 5 V for functionality, meaning that the USB connector had to be placed away from the other components to allow for a power plane at that voltage level. This meant that the path for the traces going to the dedicated pins were much longer and led to some difficulty with routing other traces for connections for some of the minor components on the board.

The USB connector was primarily included to allow for interfacing with a downwelling irradiance sensor. After doing some reading into datasheets for the sensors, I believe that the data from a downwelling irradiance sensor can be gathered through serial communication like I2C or SPI. This means that the next revision of the board could forego the USB connector and instead interface with a downwelling irradiance sensor with a serial communication driver. While this would take up some fabric resources, it would reduce the complexity of

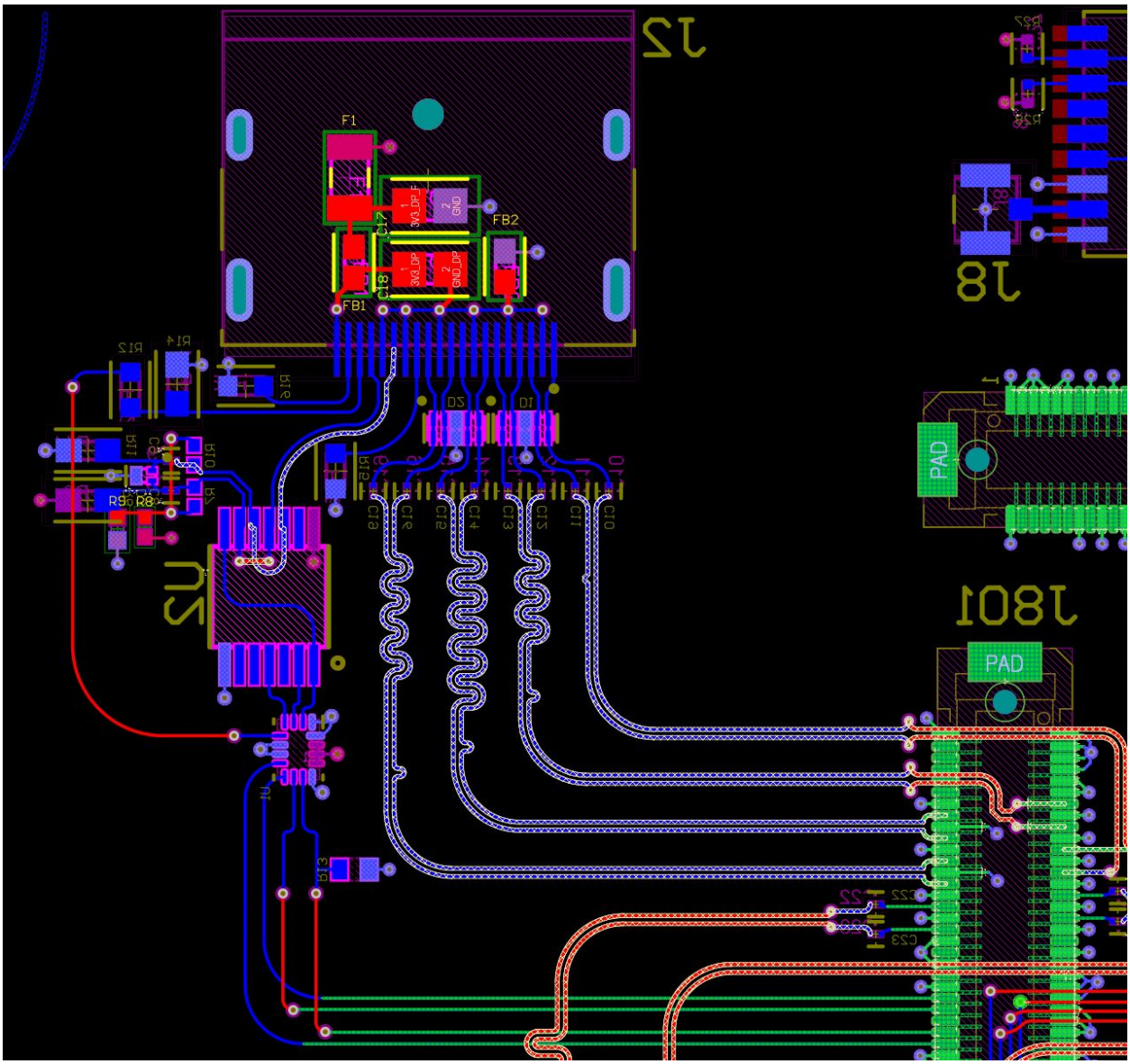


Figure 17: Close view of the DisplayPort connector routing

the board and allow for more compact routing and a decrease in power usage for the board.

Ethernet

The RJ45 Ethernet connection is the most straightforward component on the system as far as routing goes. It contains four differential pairs and two additional pins. With the dedicated Ethernet pins on the FPGA connectors, these were just a nearly straight

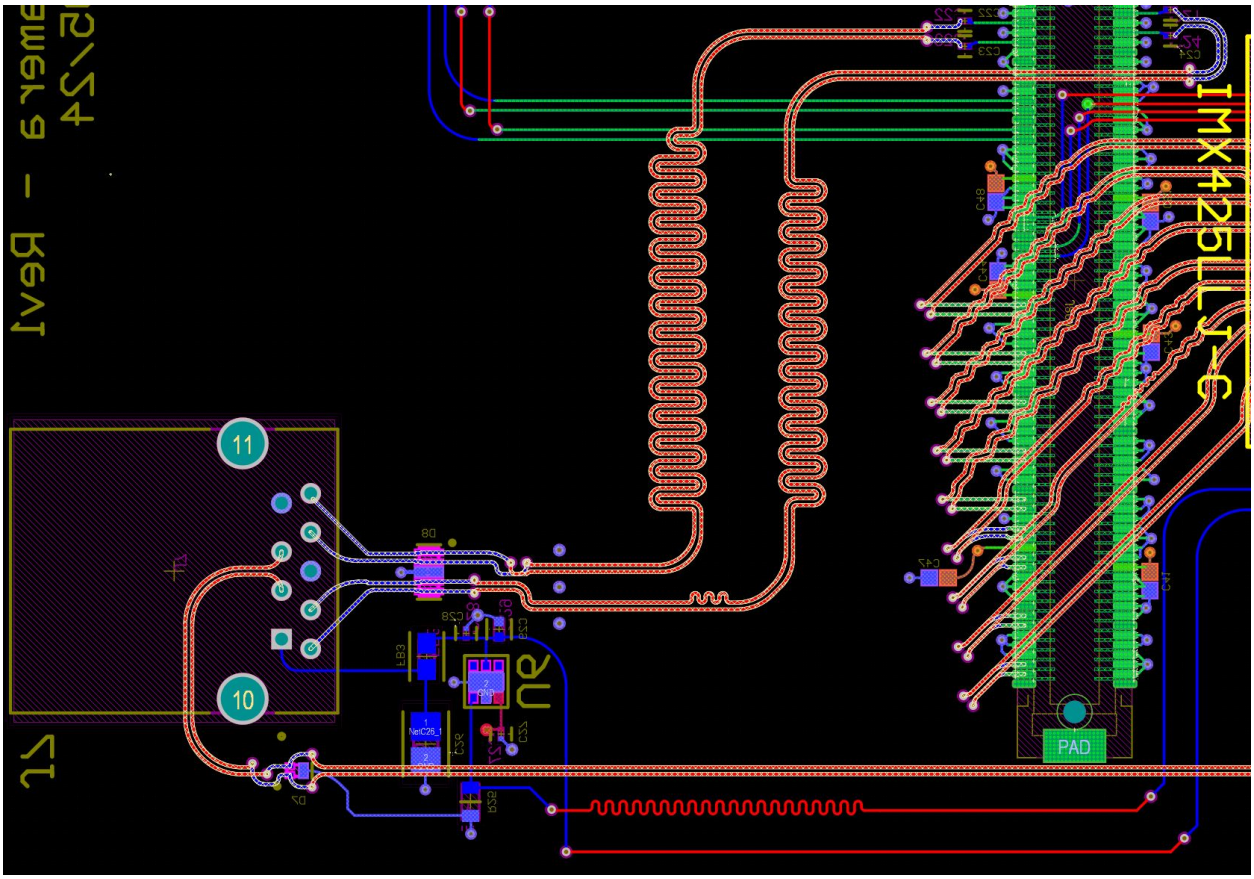


Figure 18: Close view of the USB 3.0 connector routing

line connecting the component to the connectors. Ethernet is also a 100 Ω differential pair impedance, meaning it used the same trace width and gap as the CMOS sensor and DisplayPort.

SD Card

The only additional circuitry for the micro SD card reader was the addition of a level shifter. In turn, this level shifter ended up being unnecessary due to the same voltage being used for both ports of it. A good reason to change that and properly use the level shifter is to allow for the abiltiy to run the SD card at the highest speed possible, which requires the high speed handshaking between the host and card to have a shift down in voltage.

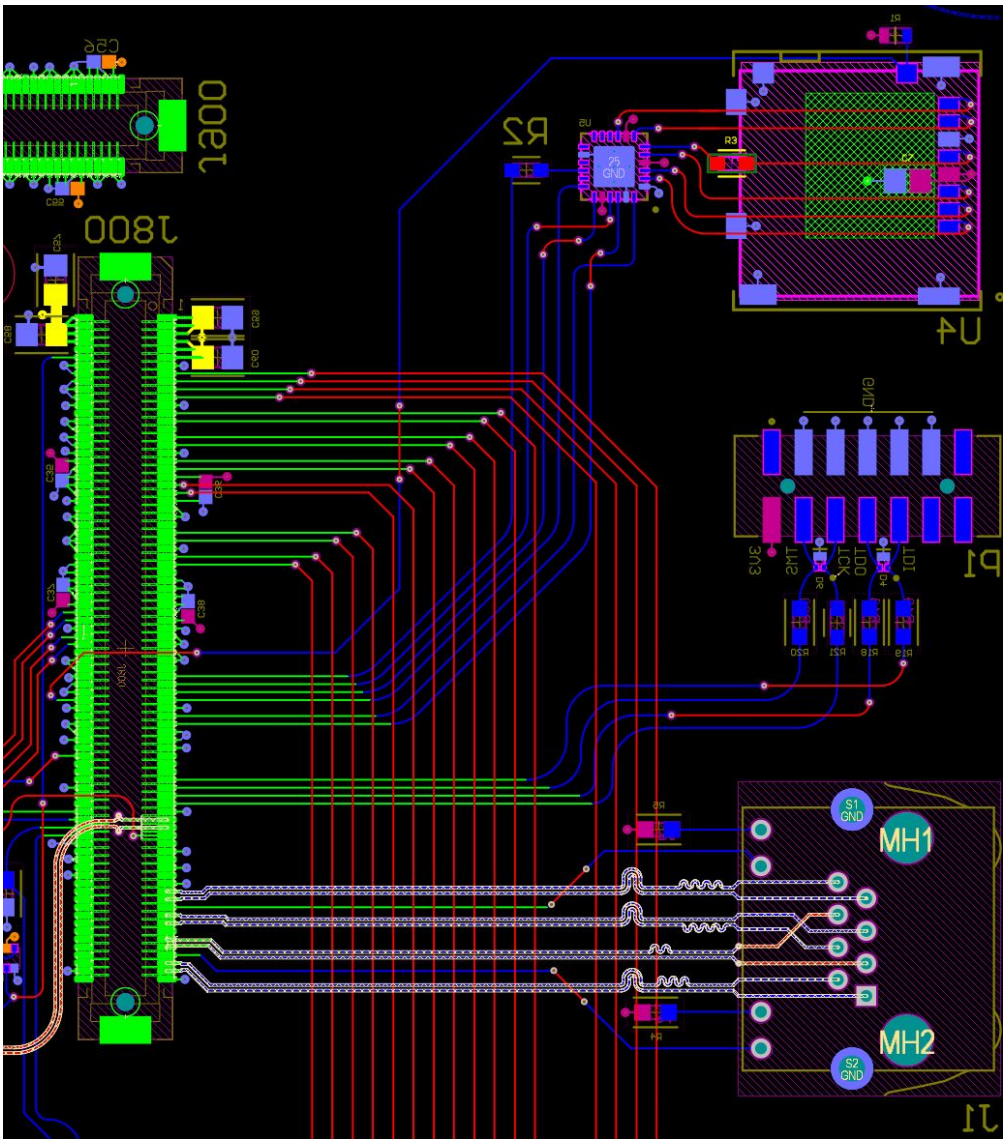


Figure 19: Close view of the peripherals (Ethernet, SD Card, JTAG) connected to FPGA connector A

GPS and IMU

The GPS and IMU were the first components other than the sensor that didn't have any dedicated pins for the connectors, which led to them being easier to slot into free areas on the connectors. For the GPS we are using a ublox MAX-M10S-00B and for the IMU we used a TDK ICM-20948. Both of these components were chosen due to the low power and easy interfacing to the FPGA. These were able to fit in next to each other with direct

connections to the connectors and no additional IC attached to them. The pin connections for these were moved around several times while I was changing some of the pin connections on the CMOS sensor.

JTAG, GPIO, and Configuration Pins

The final connections made were for the JTAG connector, the GPIO pins, and the connections for the board configuration. The dedicated JTAG pins were all connected directly to a pin header, plus the addition of ESD protection on the relevant pins for the JTAG connection. After all of these connections were made, I found a span of free connections on the FPGA and routed them to a set of headers alongside a pin for voltage and ground. These were done last due to them having no requirements for length matching or exact pins. I also made the connections for the boot mode configuration pins on the FPGA and connected them to a set of dip switches that short to ground when closed and are at the configuration voltage when open.

Power

The power requirement for the board is only four different voltage levels alongside the voltage level for the module power. The Mercury+ XU1 accepts a range of 5-15 V for the module power and the peripherals require 1.2 V, 1.8 V, 3.3 V, and 5 V. In an effort to allow more time to prototype a power sequencer, one was developed externally instead of building it into the board, the input for all of these power planes were external wire screw terminals that go to respective power planes on one of the internal layers of the board. While this did simplify the power on the board, it did lead to a requirement that a power subsystem be developed afterwards. This system would have to handle the sequencing for the power due to the CMOS sensor having specific requirements for timing on the power levels for the chip.

Problems

In general, it's expected that there are always going to be some issues with custom circuit boards on the first run of the board, and this board was no exception unfortunately. While there is still some functionality with the board, other issues with it require a second run of the board in order for the imager to achieve full functionality for a field test in the summer. The primary issue on the board is the inability to boot from the SD card that contains the Linux image and the root file system. Further investigation into this issue shows that the SD card clock is not being generated by the FPGA when the board boots. This was almost certainly caused by a power sequencing issue that I missed due to me not focusing on a power system for this revision of the board. By pulling the VCC_IO pins high too fast (the requirement is simply >0 ms), the IO bank for that connector is held in an unknown state. By adding in a sequencer based off of another signal on the board, the SD card clock should be able to start being generated.

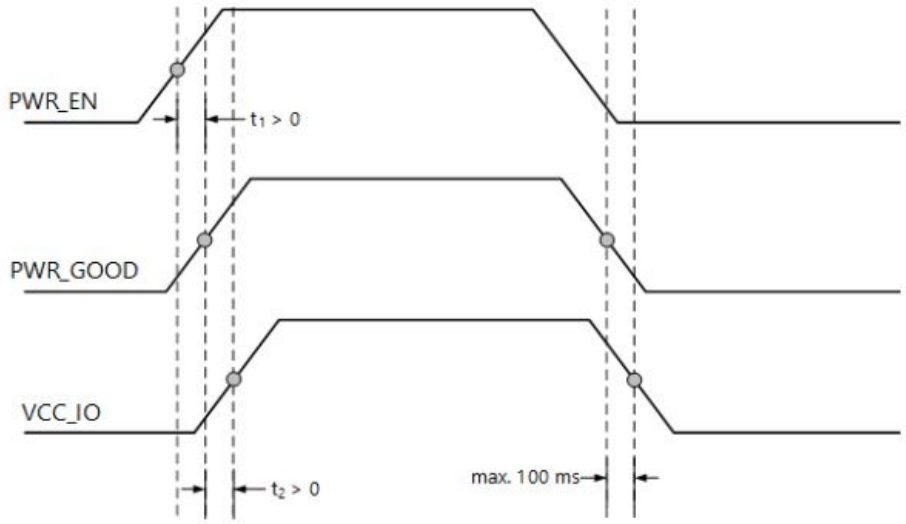


Figure 20: Timing diagram of the FPGA that shows the sequencing requirements for the VCC_IO pins

Along with a major error like the VCC_IO pins, there is a variety of small errors that were missed during the review process. The mounting holes in the corners were not adjusted,

leading to mounting holes incapable of actually holding a screw. The status LED's for certain pins on the FPGA were also placed in the center of the board, which is directly under the FPGA when it is plugged in to the board.

The external power system, while not necessarily a problem, would lead to a higher likelihood of failure for the power system if brought into the field and would require a very intricate case for holding the different PCB's and the wires connecting the power system prototype to the power wire terminals.

# FPGA PERIPHERAL DEVELOPMENT

Once the hardware was developed, I moved onto doing the development for the FPGA. The main focus of this was on the development of a 'passthrough' project for allowing verification of the CMOS sensor connections and firmware. This passthrough would pipe the data from the Sony CMOS sensor out through a DisplayPort connection to a monitor connected to the board. Since the custom PCB shares several connections with the baseboard due to the dedicated pins for the HPS, the development of this could be done prior to the board arriving after ordering it for manufacturing.

The development of the peripherals for the Mercury+ XU1 show how more advanced FPGAs leads to a much more complex development process. The passthrough system for the XU1 would involve register transfer level (RTL) code written in VHDL, block diagrams for connecting IP blocks in Vivado, and software for the embedded Linux kernel such as device trees and device drivers. This leads to multiple points of failure if and when something goes wrong, which can be extra difficult to debug if there isn't any meaningful error message to point in the direction of the root of the error.

## Test Pattern Generator Display Port Passthrough

The initial development of the passthrough began prior to the arrival of the CMOS sensor PCB, so the initial project was developed using a test pattern generator (TPG) in lieu of the CMOS sensor. This IP block in Vivado generates the video signal for a test pattern, which can then be connected to the DisplayPort on the HPS block. The functionality of the DisplayPort on the HPS can be configured to either use a 'live input' of streaming data in real-time from the PL or a 'non-live input' that instead reads data from memory through a DMA interface[30]. These inputs can either be in the form of native video or through the AXI-Stream interface, a standard high speed high bandwidth data transmission

interface. The DisplayPort controller subsystem is also capable of advanced processes like graphics upsampling and downsampling, chroma keying, video blending, audio mixing, and color space conversion.

Even though the TPG outputs data through an AXI-Stream interface, I converted it to a native video interface using a conversion block in the fabric. This allowed for an easier time plugging in the CMOS sensor block which outputs its data through a native video interface as opposed to an AXI-Stream one. Since the conversion from AXI-Stream to native video only changes the data, the TPG must also be connected to a video timing controller (VTC). This IP block generates the horizontal and vertical sync for the TPG data, which is something that is also replaced when the CMOS sensor is connected to the system since it generates its own HSYNC and VSYNC signals alongside the data. These were all then run off of a 74.25 MHz clock, which is capable of generating 1280x720 HD resolution at 60 Hz, or 1920x1080 at 30 Hz.

Once this block design was generated and connected to the top level of the fabric, the bitstream was generated after undergoing the synthesis and implementation of the design into the FPGA fabric. This bitstream is the binary code that is used to upload a design to the fabric of a board through some serial connection to the board. However, since we want to use the SoC of the FPGA, we would need to export this bitstream to a Xilinx support archive (XSA) file in order to use it to generate a custom Linux image using Petalinux. Once the XSA was created, a Petalinux project was created and used this XSA as the initial hardware configuration. This helped to autogenerate most of the device trees for the system that then just have to be modified for additional changes to the system that we needed to make.

In our case, the only changes we needed to make were to configure the DisplayPort to function with a live input since it doesn't come with that behavior by default. This can be accomplished by adding an additional clock to the device tree node, which can then be

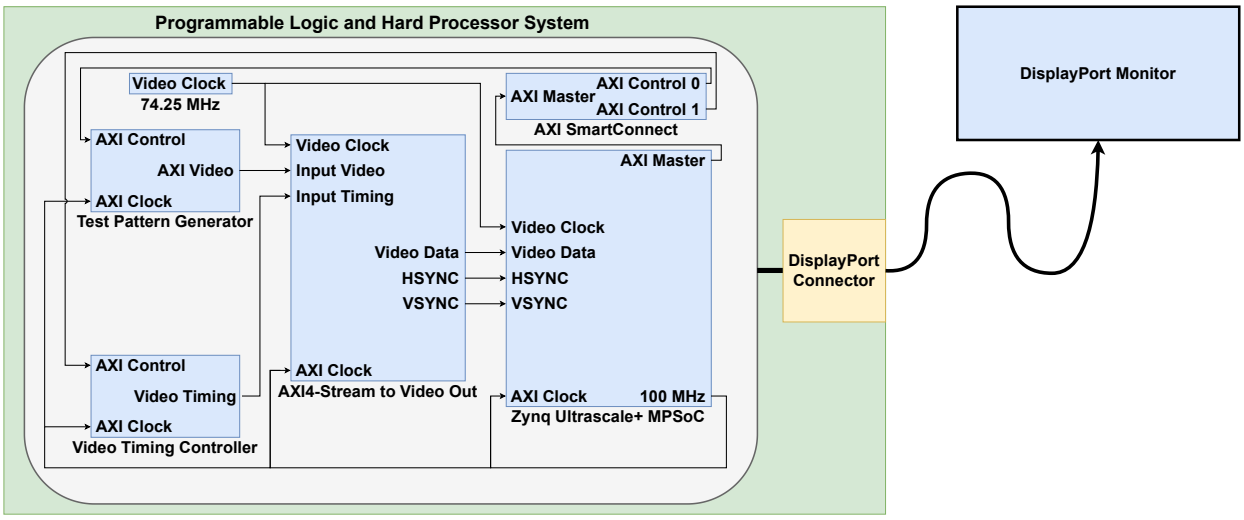


Figure 21: Block diagram of the live input hardware

enabled after the system boots up or during the system boot. After the live input is enabled, the DisplayPort should switch over from the default interface showing the console to the data generated from the fabric.

This all seems very simple with minimal straightforward changes to the system in order for everything to function. However, the path to getting a working DisplayPort took a significant amount of time due to a myriad of issues ranging from minimal or incorrect documentation, toolchain issues, and significant time between changes due to the build process of the system. Much of the documentation for accomplishing something will often leave off pertinent info about different subsystems that need to be configured a certain way, leading to a process of chasing down errors during compilation or on boot. A great example of this is certain device tree bindings referencing different nodes on a device tree without stating what those nodes look like (or even what they are sometimes).

Often these problems can be alleviated by using a reference design that often comes with the FPGA board provided by the company that creates the board. These often have the peripherals for the development board functioning at a basic level and can be modified for more specific applications. However, when we first started the development of the

FPGA peripherals, the Enclustra reference designs were two years out of date. With FPGA development, you often want to stay on the same version of software due to changes between yearly versions being significant enough to cause issues on projects if you update. If we were to stick with the older versions of Vivado and Petalinux, it would be 4-5 years out of data by the time the end of the grant rolled around. By using the newer version of the software, it required that we start the project from scratch. While this was slower for the development process, it created a much deeper understanding of the tools from the debugging I had to do with the device tree and fabric.

At the end of 2024, Enclustra finally released their board support packages (BSP) for their boards running on the 2024 version of all of the development tools. This allowed us to have a working reference design to start projects from, which also coincided with the same time that I finally managed to get the DisplayPort output of the development board working.

## CMOS Sensor Passthrough

Once the DisplayPort was functioning correctly, the implementation of the CMOS sensor could start. Since the CMOS sensor is capable of outputting native video and generates its own timing signals and data, the TPG, VTC, and AXI-Stream to native video converter could be taken out of the block design and instead replaced with IP blocks to handle the CMOS sensor. This required creating an IP block that connects the top level FPGA pins that the CMOS sensor is attached to to the internal signals of the block. While it would be ideal for the sensor to simply get a power and clock and output valid data through the LVDS channels, the sensor uses synchronization signals and blanking periods in between transfers of valid pixels. This requires a finite state machine (FSM) for collecting data from the sensor and only outputting valid data.

The state machine for the CMOS frame buffer is a simple eight state FSM for handling

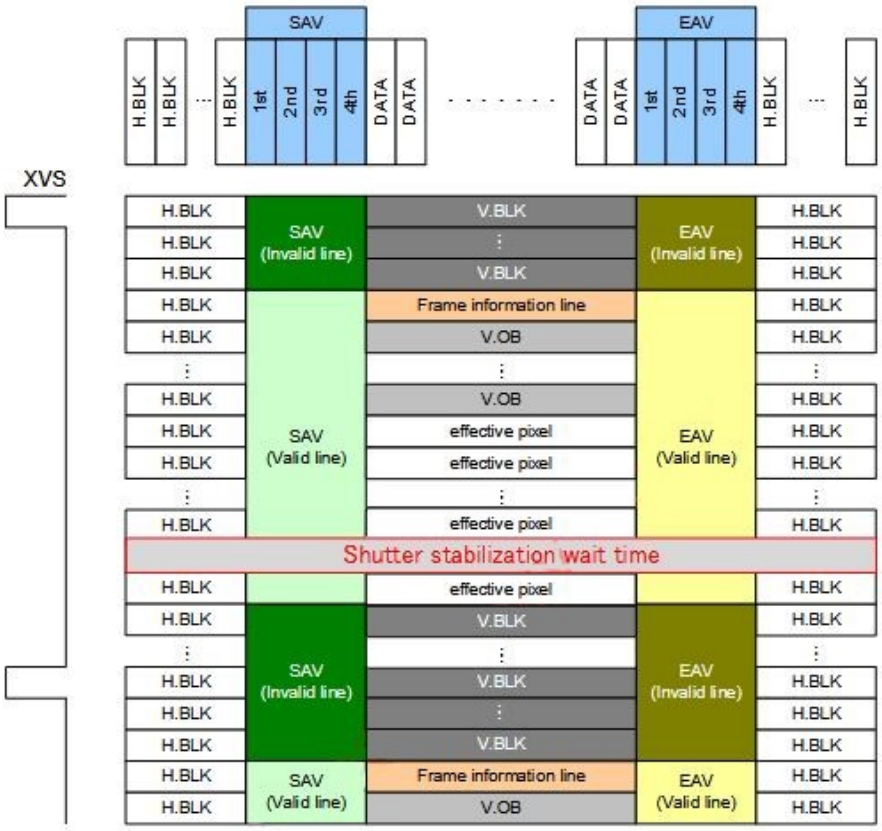


Figure 22: Output of the CMOS sensor data

the different types of data that the CMOS sensor is capable of outputting. The FSM starts in a waiting state for the vertical sync (VSYNC) signal to go high, which indicates that the sensor is beginning to output a full frame. Once the VSYNC is asserted, it moves on to a state that waits for the horizontal sync (HSYNC) signal to go high, which indicates the beginning of a new line in the frame. This signal is held high for the duration of the row and goes low shortly after the horizontal blanking period begins.

Once HSYNC goes high, the sensor will output one of four different sync codes that depend on the number of bits used in the A/D conversion. These sync codes each come through as four smaller sync codes. The next state of the FSM buffers in the data from the sensor, checking at each new input whether the four sync codes are valid. If the sync

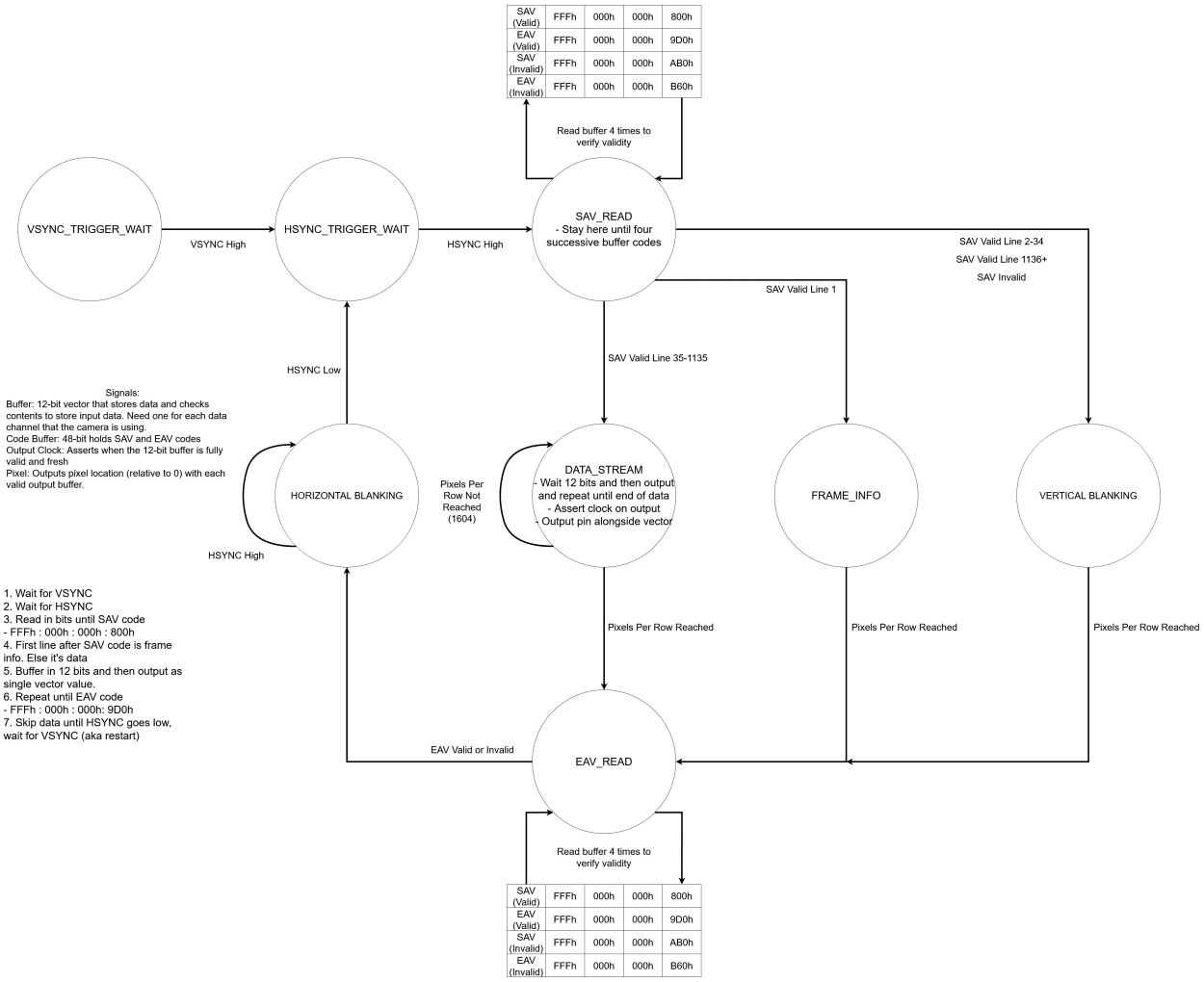


Figure 23: State diagram of the CMOS sensor frame buffer

codes match one of the two expected start of line codes (SAV), the FSM moves into one of three states depending on the current line of the frame and whether the SAV code is valid or invalid. While the SLVS-EC operating mode of the sensor gives the line number at the start of each line, we have to manually keep track of the line number for each frame.

If the SAV valid comes through as invalid, which means that the code matched the 'Invalid SAV' code and not that it didn't match one of the four codes, then that means that the data for that line of the frame is a vertical blanking period. This means that the data is not valid sensor data and we wait until we've reached the number of pixels in a row and

then jump to the next state. The FSM also jumps to this state if the sensor sends a valid SAV code and the line number is like two through 34, or beyond line 1136 before the frame ends since these rows also contain invalid data. If the sensor asserts a valid SAV code and it's line one, then the data from this row is frame information and should not be streamed out as valid data. If the sensors sends a valid SAV on lines 35 through 1135, then we finally have valid pixel data that we can then begin buffering before outputting as a single pixel and asserting an output clock.

All three of these previous states check whether they've reached the total number of pixels in the row and once they have they all jump to the same state that reads in codes checking for an end of line code (EAV). If by this point the total number of lines for a frame has been reached, then the FSM jumps to the beginning state and waits for a VSYNC assertion to begin collecting a new frame. If the end of frame isn't reached, the FSM will wait for the HSYNC signal to go low before waiting for it to go high again to collect another row of data.

Since the number of LVDS channels is configurable, this frame buffer should be able to handle eight different LVDS channels at most. However, the DisplayPort live interface doesn't have the data bandwidth for handling this much data, so the CMOS sensor will be configured to just use two LVDS channels for the passthrough verification. This will get increased once we are no longer interested in the seeing the data and want to make full use of the data throughput of the sensor.

The development of both of these peripherals is not fully complete due to the issues discovered with the first revision of the PCB that led to a shift into the development of a second revision to allow for full verification of the CMOS sensor passthrough.

# SYSTEM RADIOMETRIC ANALYSIS

## Radiometric Analysis

In an ideal world, the signal collected by a sensor would consist of just the data representing the target. Unfortunately for us, we live in a less-than-ideal world where the signal collected by a sensor consists of varying types of noise and background to accompany the target. This noise can vary depending on the type of sensor, the components used in creating the sensor, the use case of the sensor, or just noise caused by the environment where the sensor is being used. One important metric for a remote sensing system is the signal-to-noise ratio (SNR), which is the strength of the target signal relative to the noise in the system. This metric can be collected by gathering multiple different measurements over time and dividing the mean of the signal by the standard deviation of the signal. This can also be done by gathering the strength of the signal based off of data collected by the imager and specifications of the sensor and dividing that by the total theoretical noise of the signal caused by the various components and environmental factors. Usually this is calculated using real data collected by the imager. However, in lieu of real data due to the system not being fully functional, we can use software to gather a theoretical signal collected by the imager.

## Modtran Simulation

To substitute the lack of at-sensor radiance gathered with the imager, we can use software called Modtran in order to gather radiance values that reflects more realistic measurements that account for different types of scattering and absorption in the atmosphere depending on environmental and atmospheric conditions. The input parameters shown in Table 3 were chosen to gather the direct transmitted irradiance from the sun with atmospheric conditions typical of the continental U.S. The observer height of zero is for

a ground-based hyperspectral imager, meaning we can assume that the scattering in the atmosphere between the ground and the sensor is minimal. The solar zenith angle was also selected to closely mimic what it would be during a typical midday data collection.

Once we run this Modtran simulation, we need to do a few conversions in order to convert the transmitted solar irradiance as a function of wavenumber ($cm^{-1}$) into reflected radiance as a function of wavelength ($nm$). Once we have the direct solar irradiance, we can scale the units from $\frac{W}{cm^2 \mu m}$ to $\frac{W}{m^2 nm}$. We can then multiply by the reflectance of the ground (in this case, 0.2) to get our reflected spectral irradiance ($\frac{W}{m^2 nm}$). We then divide this irradiance by $\pi$ for the projected solid angle of the heimisphere that the scattered light goes towards, giving us our reflected radiance ($\frac{W}{m^2 srnm}$) at the sensor. This assumes the sensor is on the ground. If we wanted to model the sensor in the air, we would need to multiply our reflected radiance by the atmospheric path transmittance. With this reflected radiance, we can then calculate the optical flux of the system.

Table 3: Modtran simulation parameters.

| **Parameter #** | **Value** |
|---|---|
| Model Atmosphere | Mid-Latitude Summer |
| Type of Atmospheric Path | Slant Path to Space or Ground |
| Mode of Execution | Direct Solar Irradiance |
| CO2 Mixing Ratio | 419.0 ppmv |
| Aerosol Model Used | Rural - VIS = 23 km |
| Ground Altitude Above Sea Level | 1.5 km |
| Path Type | Observer Height, Zenith Angle |
| Observer Height | 0 km |
| Solar Zenith Angle at H1 | 45 degrees |
| Initial Wavelength | 400 nm |
| Final Wavelength | 1000 nm |
| Wavelength Increment | 0.37 nm |
| Day of Year | 121 |

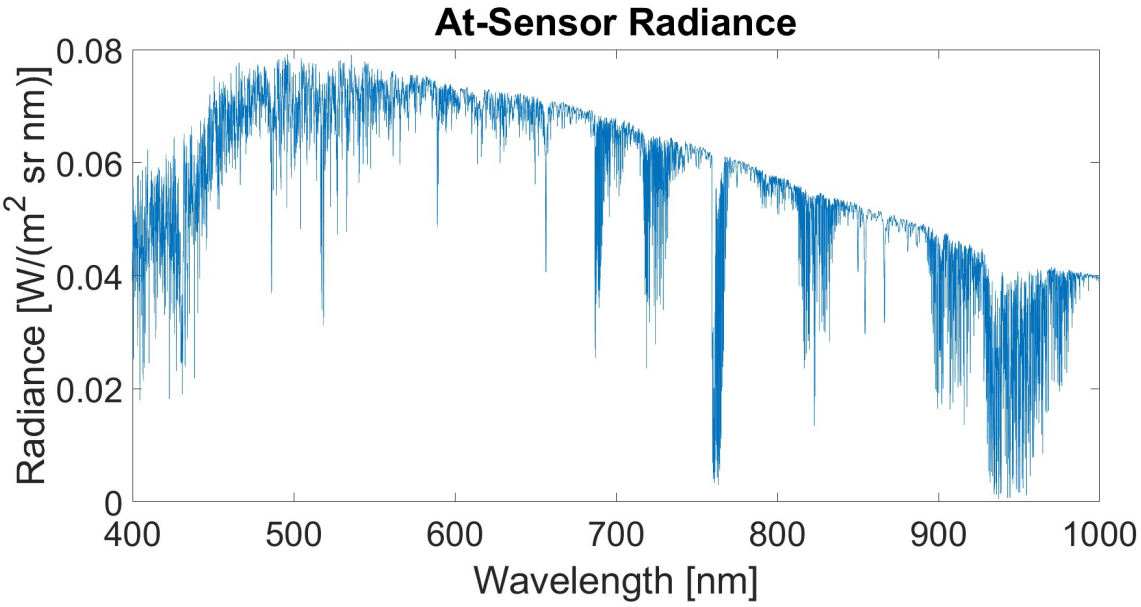


Figure 24: At sensor radiance of the system from Modtran simulation

Optical Flux and Signal-to-Noise Ratio

For hyperspectral cameras, the theoretical optical photon flux at each pixel is dependent on radiance at the sensor, the area of the pixel in the sensor, the efficiency of the CMOS sensor, and the f-number (f/#) of the system. Since the radiance at the sensor and the efficiency of the sensor both depend on the wavelength range the system is targeting, the SNR of the system is also a function of wavelength and isn't a singular number. There are several sources that can create noise in a hyperspectral system, but in most cases only a handful of them create a significant enough amount of noise to be included in a theoretical SNR calculation. Some of the more significant sources of the noise in a hyperspectral system include the shot noise, read noise, dark current noise, and quantization noise [31]. The shot noise is the variance of the system caused by the physical nature of photons and is the energy of the signal divided by the energy of the photon at a certain wavelength. The read noise and the dark current noise of a system are both sources of noise created by the CMOS sensor or CCD used in the system and are often provided by the manufacturer. The quantization

(or digitization) noise is the noise associated with the uncertainty when the analog signal collected by the CMOS sensor is converted to a digital number for the hyperspectral system to use.

Since our system isn't fully complete and we don't have access to some of the details for the optical frontend of the system being developed by Resonon or missing values in the CMOS sensor datasheet, we have to make a handful of assumptions about certain parameters in order to gather an accurate SNR estimate. These assumptions include the f-number of the system, the electron well depth of the CMOS sensor, the dark current of the CMOS sensor, and the read noise of the CMOS sensor. With these assumptions based on other similar components or consulting with Resonon, we have all of the necessary information for doing a theoretical calculation based on Equation 2 for the optical flux at each pixel of the detector. For the flux calculation, the at-sensor radiance comes from our previous Modtran simulation, where we take the solar irradiance and convert it into the reflected solar radiance at a low-altitude imager. The area of the detector pixels and the quantum efficiency come from values given in datasheet of the CMOS sensor. Once we have the optical flux calculated, we can calculate the SNR of the system using Equation 3.

$$L_{photon}(\lambda) = L_{spectral}(\lambda)\frac{\lambda}{hc}\Delta\lambda \tag{1}$$

$$\Phi(\lambda) = \frac{L_{photon}(\lambda)A_D\pi\epsilon(\lambda)}{4(f/\#)^2} \tag{2}$$

The SNR of a system at a high level is the mean of the target signal divided by the standard deviation of the signal. When applied practically, the equation for SNR can become a bit more complicated than that. For the SNR of a hyperspectral imager, we need the optical

Table 4: Flux and SNR Parameters, Values, and Sources

| Symbol | Parameter | Value | Source |
|---|---|---|---|
| $\Phi(\lambda)$ | Optical Flux | See: Figure 25 | Theoretical Calculation |
| $L_{spectral}(\lambda)$ | At Sensor Radiance | See: Figure 24 | Modtran |
| $L_{photon}(\lambda)$ | Photon Radiance | See: Equation 1 | Physics |
| $A_D$ | Detector Pixel Area | $9x9$ $\mu$m | CMOS Sensor Datasheet |
| $\epsilon(\lambda)$ | Detector Efficiency | See: Figure 8 | CMOS Sensor Datasheet |
| $\Delta\lambda$ | Pixel Wavelength Range | 0.375 nm | $\lambda$ Range |
| $f/\#$ | F-number | 2.4 | Assumption |
| $h$ | Planck's Constant | $6.626x10^{-34}$ Js | Constant |
| $c$ | Speed of Light | $3x10^{8}$ $\frac{m}{s}$ | Constant |
| $i_{dark}$ | Dark Noise | 125 $\frac{e^-}{s}$ | Assumption |
| $\Delta t$ | Integration Time | 0.0056 s | CMOS Sensor Configuration |
| $N_{read}$ | Read Noise | 6.6 $e^-$ | Assumption |
| $WellDepth$ | Electron Well Depth | 80000 $e^-$ | Assumption |
| $B$ | ADC Bit Depth | 12 bits | CMOS Sensor Configuration |

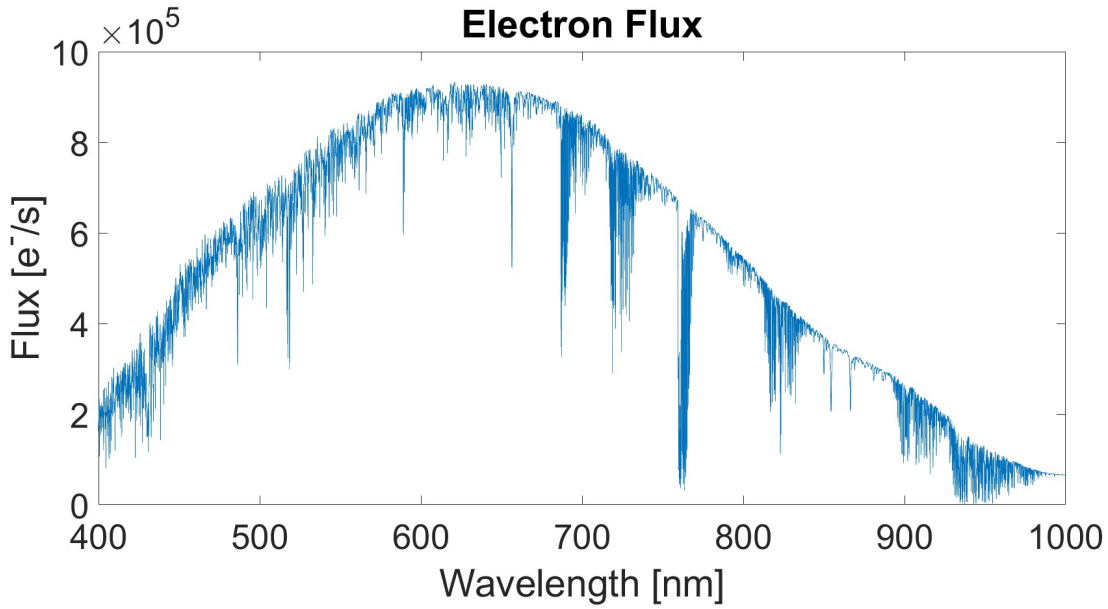


Figure 25: Optical flux theoretical calculation from Equation 2

flux at each pixel of the detector and a variety of different noise sources determined by the characteristics of the sensor and imager. For a hyperspectral imager, the noise sources that we need are the signal shot noise, dark signal shot noise, read noise, and quantization noise. All of these noise sources can be viewed as independent of one another and independent on

a pixel by pixel basis. The numerator of the SNR equation is simply the energy of each pixel and the inverse photon energy. This simplifies down to units of $e^-$ and represents the mean of the signal as a function of $\lambda$. For signal and dark current shot noise we are assuming they are Poisson processes, so the mean of the signal is equal to variance of the signal. This means that the variance of the photon signal is the same value as the mean of the signal found in the numerator of the SNR equation. Since standard deviations always have the same units as the signal and since variance is the square of the standard deviation, this photon signal noise has units of $e^{-2}$. The same assumptions are made for the dark signal noise, with the value being the same as the dark signal, also in units of $e^{-2}$. Read noise on the detector is typically reported in the sensor datasheet as a root mean-square-error (RMSE) value. This RMSE value can be thought of as the standard deviation, with the square of it (mean-square-error (MSE)) being the variance. The quantization noise is derived from the standard deviation of a uniform probability density function, so it is also RMSE value. To be able to add these to the two variances previously calculated we need to convert them from standard deviations to variances by squaring them. Both of these noise sources also come in terms of $e^-$ for the standard deviations, which causes the variance to have units of $e^{-2}$. Once we have all of these independent variances (which we have assumed to be true), we can add them in quadrature and take the square root in order to get the standard deviation in the denominator as a function of wavelength. This will also make the units in the denominator $e^-$, leading to the units in the numerator and the denominator cancel each other out to give us our unitless SNR as a function of wavelength.

$$SNR(\lambda) = \frac{\Phi(\lambda)\frac{\lambda}{hc}}{\sqrt{\Phi(\lambda)\frac{\lambda}{hc} + i_{dark}\Delta t + N_{Read}^2 + (\frac{WellDepth}{2^B\sqrt{12}})^2}}, \tag{3}$$

With the light spread out across a range, the strength of the photons isn't high enough

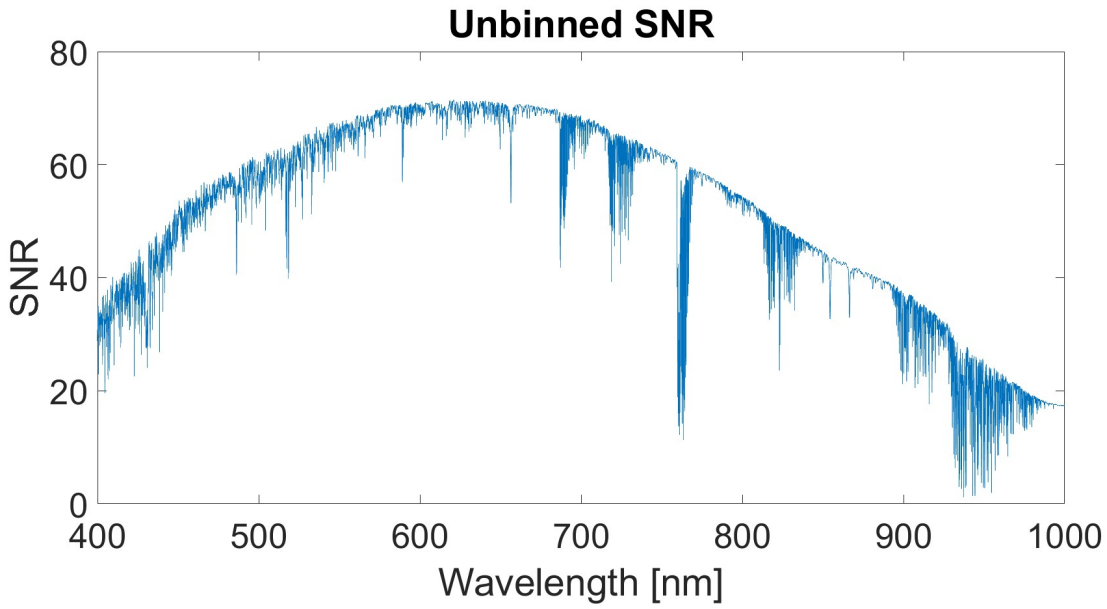


Figure 26: System SNR with no binning from Equation 3

for a strong signal in some cases. To counteract this, we can add adjacent wavelengths in order to increase the strength of the signal. This binning will increase the SNR of the signal, with a reduction in the optical resolution. Both of these increase or decrease as a factor of K which is equal to the binning value of how many adjacent wavelengths get added together as shown in Equation 4. Due to the custom adder tree developed by Nicholas Margavio, an REU student at MSU during summer 2024, the binning value does not have to be constant across all wavelengths and can be increased or decreased arbitrarily. This allows us to have higher and lower SNR or optical resolution depending on the wavelength.

$$SNR(\lambda_K) = \frac{\Sigma_1^K \Phi(\lambda_k)\frac{\lambda_k}{hc}}{\sqrt{\Sigma_1^K \Phi(\lambda_k)\frac{\lambda_k}{hc} + i_{dark}\Delta t + N_{Read}^2 + (\frac{WellDepth}{2^B\sqrt{12}})^2}} \tag{4}$$

Since we also have control over the bit depth of the ADC on the CMOS sensor and the frame rate of the system by changing the number of LVDS channels and the bit depth, we have some control over our system's SNR. If we wanted to increase our SNR at the cost

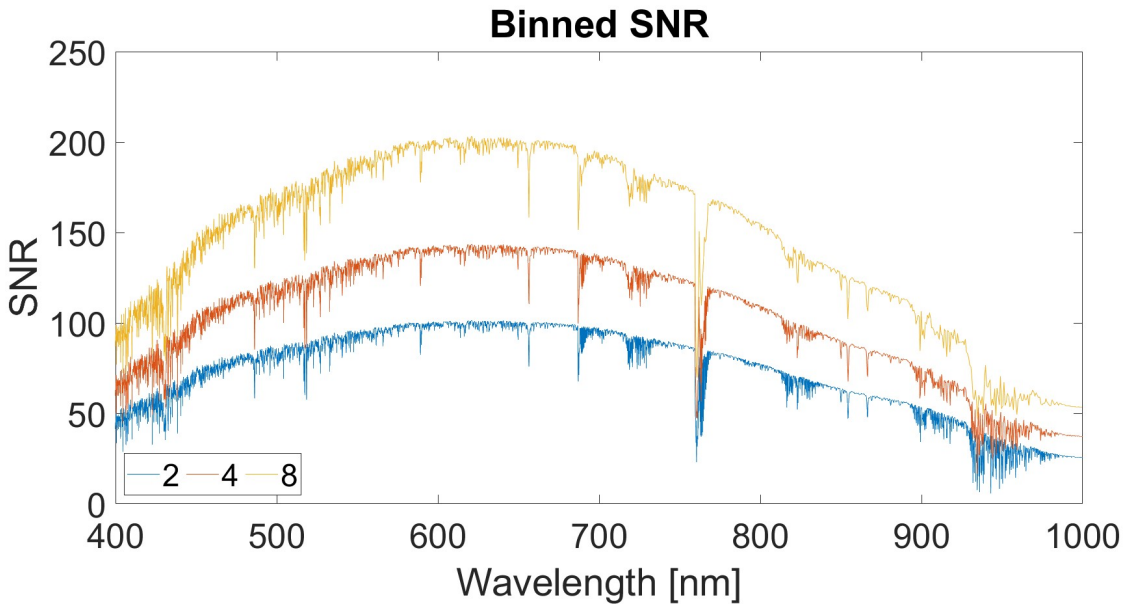


Figure 27: System SNR for different flat binning values from Equation 4

of data throughput, we can decrease the data throughput of the system, or inversely we can calculate the theoretical SNR before determining which data throughput would be best (in terms of SNR). The data throughput of the system is also not arbitrarily chosen, but influenced by the latency of the data processing pipeline for the hyperspectral images. Figure 28 shows the differences in the SNR by changing the frame rate of the system. The three frame rates chosen come from using a 12 bit ADC bit depth with varying LVDS channels found in Table 2.

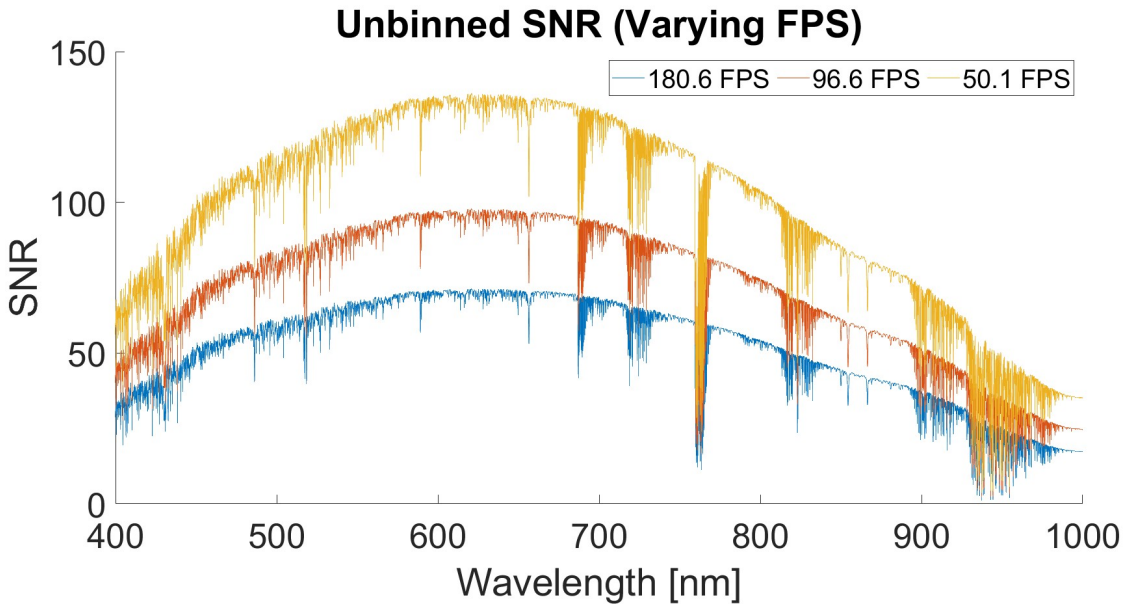


Figure 28: Unbinned system SNR for varying frame rates

# CMOS SENSOR PRINTED CIRCUIT BOARD REVISION 2

Due to the issues relating to the IO voltage banks and the inability to boot the system from the SD card, a second revision of the PCB needed to be completed in order to have a fully functional camera prototype. While it was a disappointment, the need for a second revision allowed me to make some changes to minor problems on the board along with adding additional components in order to allow the camera to perform better in the field as opposed to just functioning in a lab environment.

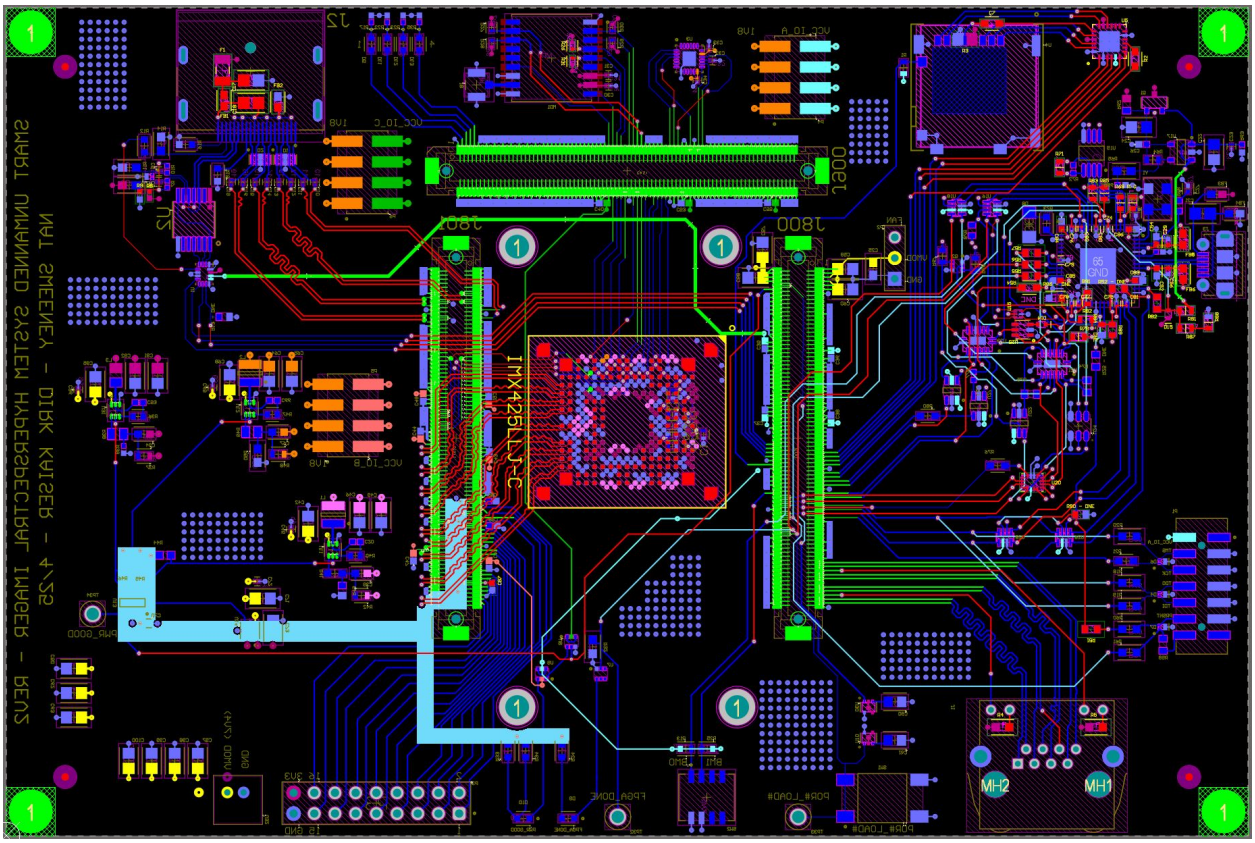


Figure 29: Full view of the second revision of the circuit board

## Modifications

The primary change to the second revision of the board was the inclusion of a power sequencing system on the board alongside a more sophisticated power setup for the board. On the original board, the VCC_IO pins on each of the module connectors were connected to the power planes for 3.3 V and 1.8 V directly since we didn't want to have a configurable IO voltage for the connectors. However, by connecting these pins to the power planes directly, we failed a step in the sequencing of the FPGA module where the VCC_IO pins needed to go high after the PWR_GOOD signal and do so within 100 ms[32]. By violating this sequencing step, the FPGA module holds these pins in an unknown state. This is believed to be the reason why the FPGA module was failing to generate a clock for the SD card reader. For the new board, an individual power plane was created for each of the module connectors with a delay circuit used to connect the planes to their designated voltage plane. By using the PWR_GOOD signal as the enable for this delay, the I/O banks will get pulled high with the proper timing.

In the first revision of the board, a USB 3.0 connector was included to allow for the interfacing between the FPGA and a downwelling irradiance sensor. After some investigation into a downwelling irradiance sensor, it was determined that a serial interface such as SPI or I2C would allow for the collection of data from a downwelling irradiance sensor. This would allow for the removal of the USB 3.0 interface, and in turn would free up considerable space on the circuit board along with the removal of the 5 V power plane that only the USB 3.0 connector utilized. This would also simplify the power sequencing on the board and allow for a longer life for the camera in the field.

An additional modification on the second revision of the board was the inclusion of a serial interface utilizing a USB 2.0 interface and a FTDI chip. The FTDI chip allows for the conversion from USB data to different forms of serial communication such as UART, I2C,

SPI Flash, or JTAG. This allows for an external computer to connect to the FPGA using the UART interface, or for some other external device such as a downwelling irradiance sensor to communicate over some other form of serial communication. This subsystem would take up considerable area on the board due to the amount of integrated circuits (ICs) such as level shifters and multiplexers required for interfacing with the FPGA. Since the USB 2.0 interface also runs off of 3.3 V, it isn't necessary to add in a 5 V plane in the same way that a USB 3.0 connector needed.

After a first attempt at routing the FTDI subsystem on a four layer board with two of the layers being signal layers, the switch from a four layer board to a six layer board was done. This would allow for a ground plane beneath each of the outer signal layers to help shield differential pairs and other high speed traces from noise, and for the inclusion of another signal layer on the interior of the board. Since this hidden layer would be next to the power plane and isn't separated by a ground plane, the traces being routed on this layer were limited to slower speed traces that weren't susceptible to noise generated by the power sources. All of the layers were also switched to 1 oz. pours as shown in Figure 30.

With the inclusion of new subsystems on the second revision, several of the minor parts of the first revision such as the GPIO and status LEDs were removed until it was determined which pins would be available after routing the new subsystems. This would also allow for fixing mistakes in the placement of these components on the first revision of the circuit board, such as the status LEDs for the FPGA being placed in a spot that cannot be seen when the FPGA is plugged in.

## Schematic Design

The schematic design for the second revision of the board was limited to the additional FTDI subsystem, modifying the power sequencer to match the signals from the FPGA, and the modification of the GPIO connections for the GPIO header and LEDs. For the FTDI

| # | Name | Material | Type | Weight | Thickness | Dk | Df |
|---|---|---|---|---|---|---|---|
| | Top Overlay, To... | | Overlay | | | | |
| | Top Solder, Top... | Solder Resist | Solder Mask | | 0.4mil | 3.5 | |
| 1 | Sensor Layer | | Signal | 1oz | 1.4mil | | |
| | Dielectric 4 | PP-006 | Prepreg | | 8mil | 4.1 | 0.02 |
| 2 | GND1 | CF-004 | Plane | 1oz | 1.378mil | | |
| | Dielectric 2 | PP-006 | Core | | 15mil | 4.1 | 0.02 |
| 3 | Hidden Layer | CF-004 | Signal | 1oz | 1.378mil | | |
| | Dielectric 1 | FR-4 | Prepreg | | 8mil | 4.2 | |
| 4 | PWR | CF-004 | Signal | 1oz | 1.378mil | | |
| | Dielectric 3 | PP-006 | Core | | 15mil | 4.1 | 0.02 |
| 5 | GND2 | CF-004 | Plane | 1oz | 1.378mil | | |
| | Dielectric 5 | PP-006 | Prepreg | | 8mil | 4.1 | 0.02 |
| 6 | Component Layer | | Signal | 1oz | 1.4mil | | |
| | Bottom Solder,... | Solder Resist | Solder Mask | | 0.4mil | 3.5 | |
| | Bottom Overlay,... | | Overlay | | | | |

Figure 30: Layers of the second revision of the PCB

subsystem, the fanout from the FTDI chip needed to be fed into multiplexers for the serial communication signals that share a common pin (I2C, UART, SPI Flash) and those need to be connected to level shifters to shift from the 3.3 V logic that they are initially running off of into the 1.8 V logic level being used by the FPGA.

## Circuit Board Design

The majority of the work for the second revision of the PCB came from the routing of the FTDI circuit and the rearrangement of the power planes in order to properly sequence the VCC_IO banks. The FTDI subsystem had a significant number of components required for functionality along with a large number of traces for connecting these components. Luckily, the FTDI circuit does not contain any higher speed data lines, so there were less considerations required for the integrity of the signals when compared to something like the DisplayPort connector or the data lines from the Sony CMOS sensor.

### Serial FTDI Subsystem

The FTDI subsystem consists of 16 ICs, a USB 2.0 connector, and a significant number of resistors, capacitors, diodes, and ferrite beads. For the layout of this subsystem I started

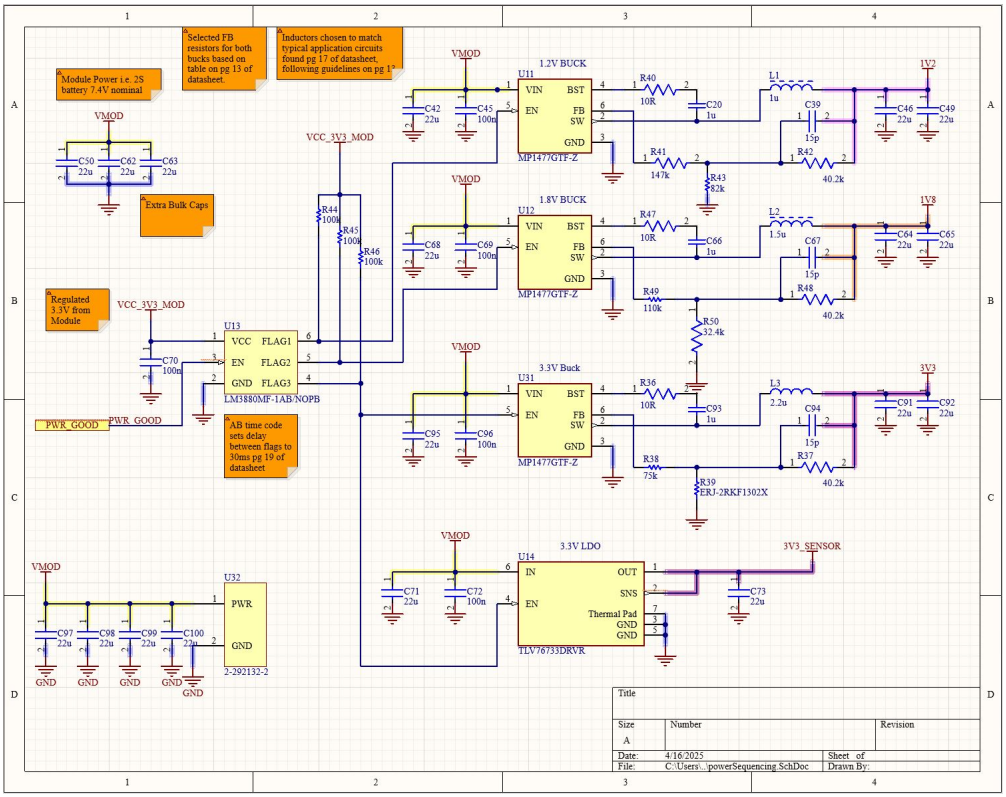

Figure 31: Schematic of the power sequencer subsystem

by placing the USB connector and its connection to the FTDI chip as far away from the dedicated pins on the FPGA connector that the subsystem would connect to. This would allow for the most room possible in placing the ICs that needed to be between the FTDI chip and the connector. I then placed all of the necessary power components for this subsystem as close to the USB connector and away from the serial communication traces as I could to minimize the little noise there might be. I then placed the EEPROM component connected to the FTDI chip, which will allow for testing of the FTDI chip and the USB connector.

After placing the USB connector, the FTDI chip, the EEPROM, and the FTDI power, I then fanned out the dedicated pins on the connector to different sections of the free space on the board. I then placed the multiplexers as close to the FTDI chip as I could while still allowing for room to route around and in between traces. When doing the connections for this subsystem, I tried to use the outer layers for routing when possible while only

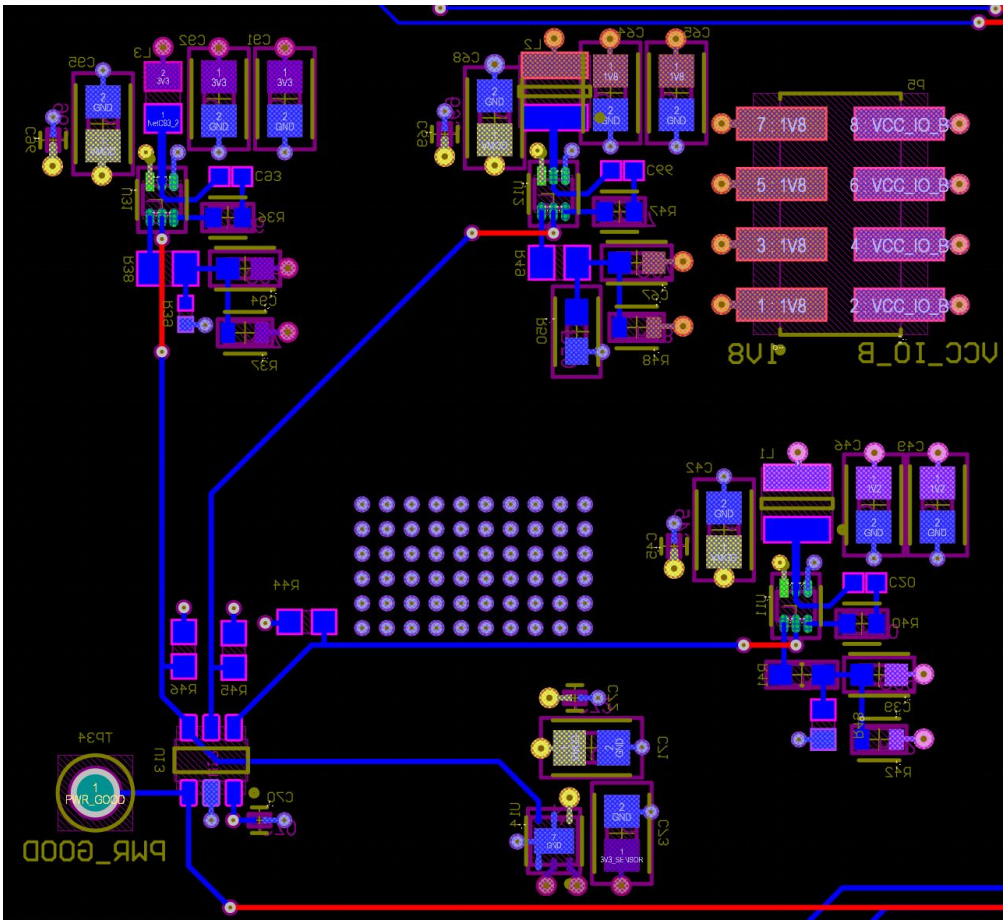


Figure 32: Layout of the power sequencer system on the camera

utilizing the hidden layer when crossing over traces that are on both layers. After connecting the multiplexers, I just fanned out the signals to the various level shifters that were than connected to the separated dedicated pins on the FPGA connector.

With the placement of each of these components, I also placed any bypass capacitors, ground vias, or power vias necessary for the functionality of the chips as close as possible to the physical pads on the footprints of the components.

Power Sequencing

Since the power sequencer was already prototyped prior to the design of the second revision of the PCB, it simply had to be placed on the open space where the USB 3.0

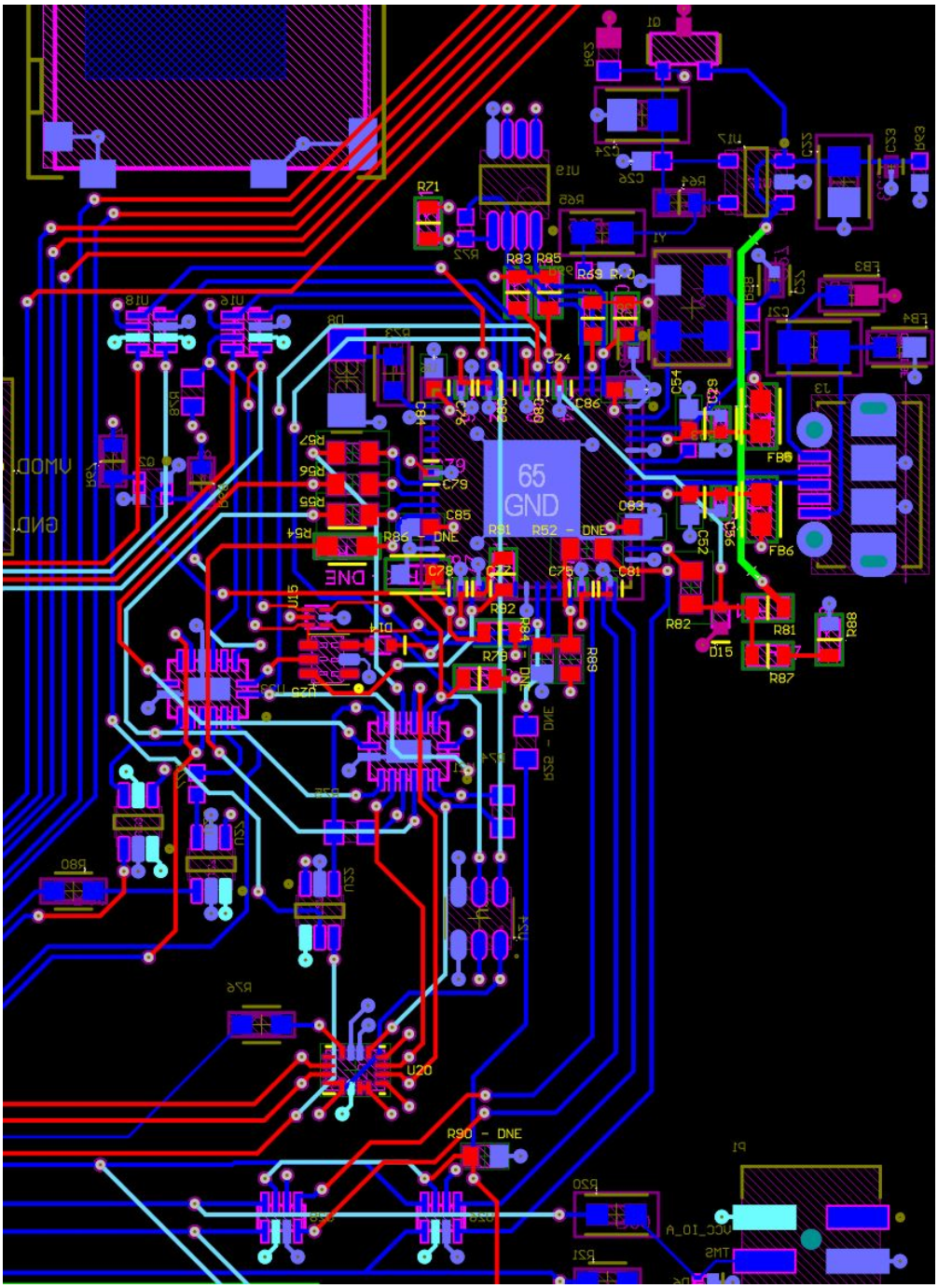


Figure 33: Close view of the routing for the FTDI subsystem

connector used to be and connected to the four power planes that it generated power for. The only change from the prototype power sequencer was the addition of a 3.3 V buck converter for the peripherals, leaving the 3.3 V linear regulator to only power the CMOS

sensor. This will help to reduce the noise on the analog power for the CMOS sensor and keep the max current draw from approaching the max amount of that the linear regulator can source.

The sequencer was then connected to the PWR_GOOD signal, which will cause the CMOS sensor and VCC_IO power planes to power up in the proper sequence relative to the FPGA module. With how the power sequencer was set up for the CMOS sensor, this means that the 1.8 V plane and IO planes will go high around 30 ms after the PWR_GOOD signal, which matches the timing requirement of $> 0$ s and $< 100$ ms.

Power Planes

With the second revision of the PCB, each of the FPGA module connectors were given their own power plane underneath the connector instead of being directly connected to one of the main power planes. These VCC_IO planes could then be connected to the 1.8 V power plane to supply the logic level voltage for the different peripherals connected to the FPGA. The connection for the IO planes to the 1.8 V plane was done through a set of headers that can be shorted together using caps. This allows for the changing of the voltage in a lab environment for both testing and debugging purposes.

Minor Changes

Since the stackup was changed for the second revision of the board, the differential pairs had to be modified to fit the new required trace widths and trace gaps to match the 100 Ω output impedance for each of the circuits. The new differential pairs were now 10 mil. widths with an 8 mil. gap between the traces. The status LEDs were also moved to the bottom of the board instead of directly underneath the FPGA when it's plugged in. I also redid the connections for the GPS, IMU, and GPIO pins.

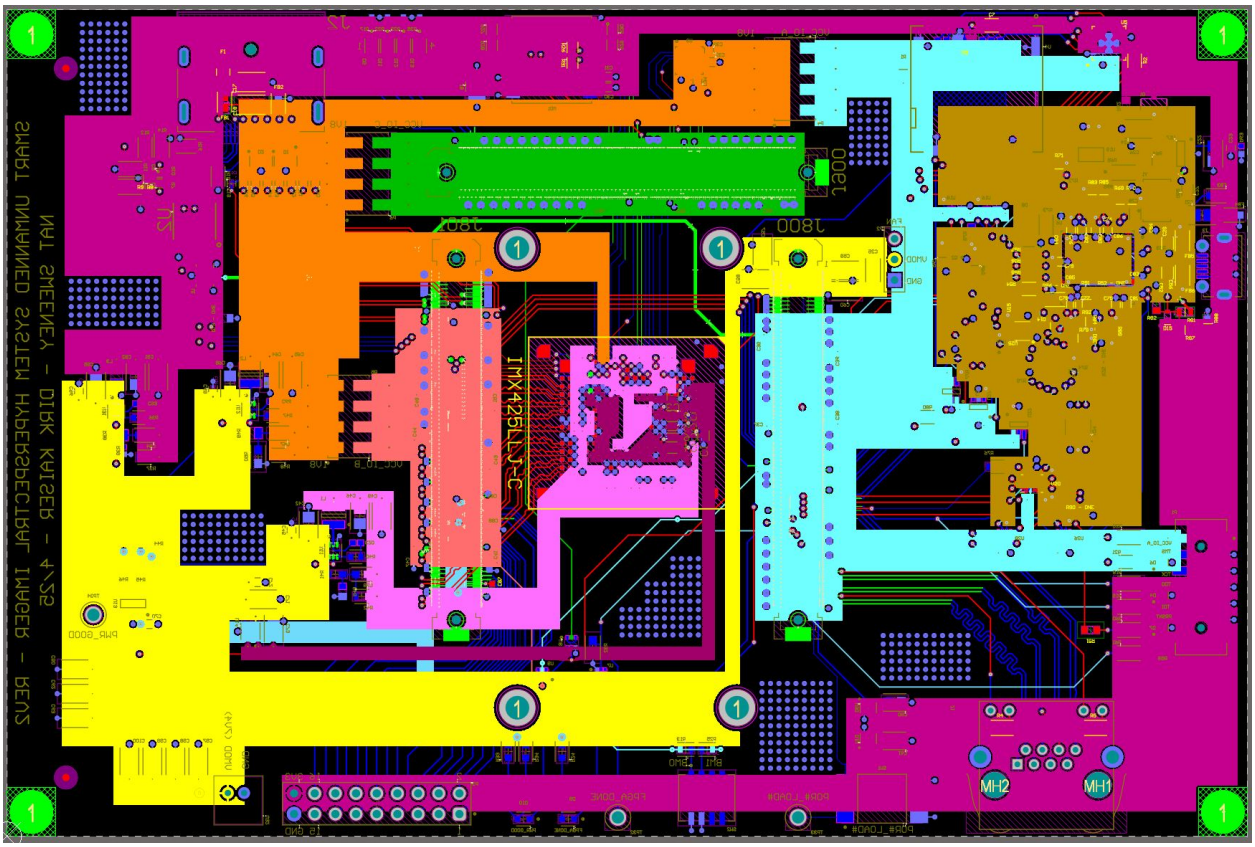


Figure 34: Power plane layout of the second revision.

# CONCLUSION AND FUTURE WORK

The development of the custom circuit boards and the DisplayPort passthrough on the FPGA allow for the implementation of the real-time data processing of the hyperspectral image data alongside the embedding of the trained machine learning model. This tandem will allow for the real-time inference of hyperspectral image data for use in characterizing ground fuel to assist with prescribed burns. The first revision of the board allowed for basic verification of peripherals on the circuit board, while the second revision of the circuit board will allow for the use of the imager in the field instead of just in a lab environment. The DisplayPort passthrough allows us to verify the physical connections between the circuit board and the CMOS sensor along with the handling of the image data streaming out of the sensor.

Once we have the second revision of the board printed and assembled, we will be able to finish the CMOS sensor passthrough on the circuit board. Once we have verified hardware connections and data handling, we will proceed to do a calibration of the CMOS sensor. This utilizes an integrating sphere to output a known radiance across a whole scene, which allows us to gather gain and offset values for each pixel on the CMOS sensor. These values can then be saved in memory and applied to the pixels as data is streamed from the sensor to the fabric.

With the second revision of the board we can also continue the peripheral development on the board. The GPS and IMU will be necessary for connecting results from the system to a real location. The Ethernet jack development will allow us to transmit files and connect to the camera through an external computer, which will prove useful when doing the calibration of the system.

Once we have the functioning and calibrated CMOS sensor, we can integrate the camera with the hyperspectral optics for the system and begin the embedding of the machine learning

model into the fabric of the FPGA. This will involve a distillation process to reduce the overall size of the model and the creation of accelerators in the fabric to allow the model to gather results from certain operations faster.

# REFERENCES CITED

[1] NOAA National Centers for Environmental Information. Annual 2024 wildfires report. Technical report, NOAA NCEI, 2024.

[2] USDA. Prescribed fire in the northwest. Technical report, U.S. Department of Agriculture, 2024.

[3] Xilinx. *UltraScale Architecture DSP Slice*. Xilinx, August 2021. Available at `https://docs.amd.com/v/u/en-US/ug579-ultrascale-dsp`.

[4] Sarah Farmer and Jenni Moore Myers. Economic risks: Forest service estimates costs of fighting wildfires in a hotter future. Technical report, USDA Forest Service, 2024.

[5] Roger Vincent. Estimated cost of fire damage balloons to more than $250 billion. *Los Angeles Times*.

[6] Office of Management and Budget. The federal government's budget exposure to financial risk due to climate change. Technical report, Office of Management and Budget, 2024.

[7] National Parks Service 022. Mar 2022.

[8] Clare E. Boerigter, Sean A. Parks, Jonathan W. Long, Jonathan D. Coop, Melanie Armstrong, and Don L. Hankins. Untrammeling the wilderness: restoring natural conditions through the return of human-ignited fire. *Fire Ecology*, 20(1), August 2024.

[9] The history of fire and the human use of fire in the northern rockies. `http://fwrconline.csktnrd.org/Fire/FireOnTheLand/History/TraditionalCulture/`. Accessed: 2025-01-27.

[10] Kathiravan Thangavel, Dario Spiller, Roberto Sabatini, Stefania Amici, Sarathchandrakumar Thottuchirayil Sasidharan, Haytham Fayek, and Pier Marzocca. Autonomous satellite wildfire detection using hyperspectral imagery and neural networks: A case study on australian wildfire. *Remote Sensing*, 15(3), 2023.

[11] Leigh B.and Hudak Andrew T.and Robichaud Peter R.and Morgan Penelopeand Bobbitt Michael J. Lewis, Sarah A.and Lentile. Mapping ground cover using hyperspectral remote sensing after the 2003 simi and old wildfires in southern california. *Fire Ecology*, 3(1):109–128, Jun 2007.

[12] Sarah A. Lewis, Andrew T. Hudak, Roger D. Ottmar, Peter R. Robichaud, Leigh B. Lentile, Sharon M. Hood, James B. Cronan, and Penny Morgan. Using hyperspectral imagery to estimate forest floor consumption from wildfire in boreal forests of alaska, usa. *International Journal of Wildland Fire*, 20(2):255–271, 2011.

[13] Christopher William Smith, Santosh K. Panda, Uma Suren Bhatt, and Franz J. Meyer. Improved boreal forest wildfire fuel type mapping in interior alaska using aviris-ng hyperspectral data. *Remote Sensing*, 13(5), 2021.

[14] D.A. Roberts, P.E. Dennison, M.E. Gardner, Y. Hetzel, S.L. Ustin, and C.T. Lee. Evaluation of the potential of hyperion for fire danger assessment by comparison to the airborne visible/infrared imaging spectrometer. *IEEE Transactions on Geoscience and Remote Sensing*, 41(6):1297–1310, 2003.

[15] Riyaaz Uddien Shaik, Giovanni Laneve, and Lorenzo Fusilli. An automatic procedure for forest fire fuel mapping using hyperspectral (prisma) imagery: A semi-supervised classification approach. *Remote Sensing*, 14(5), 2022.

[16] Igor Viana Souza, Francisca de Cássia Silva da Silva, Antonio Carlos Batista, Gil Rodrigues dos Santos, Maria Cristina Bueno Coelho, and Marcos Giongo. Using drones for estimating fuel material in cerrado grasslands. *Ciência Florestal*, 34(3):e73469, Aug. 2024.

[17] Michael J. Falkowski, Paul E. Gessler, Penelope Morgan, Andrew T. Hudak, and Alistair M.S. Smith. Characterizing and mapping forest fire fuels using aster imagery and gradient modeling. *Forest Ecology and Management*, 217(2):129–146, 2005.

[18] M. Dockray D. N. H. Horler and J. Barber. The red edge of plant leaf reflectance. *International Journal of Remote Sensing*, 4(2):273–288, 1983.

[19] Lili Guadarrama, Carlos Paredes, and Omar Mercado. Plant disease diagnosis in the visible spectrum. *Applied Sciences*, 12(4), 2022.

[20] Lisa-Natalie Anjozian. Naked eyes and hyperspectral images build fuel maps in the southern appalachian mountains. *Fire Science Brief*, 117, 2010.

[21] Billy G. Ram, Peter Oduor, C. Igathinathane, Kirk Howatt, and Xin Sun. A systematic review of hyperspectral imaging in precision agriculture: Analysis of its current state and future prospects. *Computers and electronics in agriculture*, 222:109037–, 2024.

[22] Janos C. Keresztes, Mohammad Goodarzi, and Wouter Saeys. Real-time pixel based early apple bruise detection using short wave infrared hyperspectral imaging in combination with calibration and glare correction techniques. *Food control*, 66:215–226, 2016.

[23] Henry Ndu, Akbar Sheikh-Akbari, Jiamei Deng, and Iosif Mporas. Hypervein: A hyperspectral image dataset for human vein detection. *Sensors (Basel, Switzerland)*, 24(4):1118–, 2024.

[24] Riley D. Logan, Madison A. Torrey, Rafael Feijó-Lima, Benjamin P. Colman, H. Maurice Valett, and Joseph A. Shaw. Uav-based hyperspectral imaging for river algae pigment estimation. *Remote Sensing*, 15(12), 2023.

[25] Philip E. Dennison, Kraivut Charoensiri, Dar A. Roberts, Seth H. Peterson, and Robert O. Green. Wildfire temperature and land cover modeling using hyperspectral data. *Remote Sensing of Environment*, 100(2):212–222, 2006.

[26] Julián Caba, María Díaz, Jesús Barba, Raúl Guerra, and Jose A. de la Torre and Sebastián López. Fpga-based on-board hyperspectral imaging compression: Benchmarking performance and energy efficiency against gpu implementations. *Remote Sensing*, 12(22), 2020.

[27] Bin Yang, Minhua Yang, Antonio Plaza, Lianru Gao, and Bing Zhang. Dual-mode fpga implementation of target and anomaly detection algorithms for real-time hyperspectral imaging. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6):2950–2961, 2015.

[28] Lucas Amilton Martins, Felipe Viel, Laio Oriel Seman, Eduardo Augusto Bezerra, and Cesar Albenes Zeferino. A real-time svm-based hardware accelerator for hyperspectral images classification in fpga. *Microprocessors and Microsystems*, 104:104998, 2024.

[29] High-speed interface slvs-ec. `https://www.sony-semicon.com/en/technology/is/slvsec.html`. Accessed 2025-02-18.

[30] Xilinx. *Zynq UltraScale+ Device Technical Reference Manual*. Xilinx.

[31] Rand Swanson. Signal-to-noise ratio (snr) in hyperspectral cameras. Technical report, Resonon, 2023.

[32] Enclustra. *Mercury+ XU1 SoC Module User Manual*. Enclustra.