2012-82

Foveon F13 Camera

Authors: David Shelley

Abstract: Most high-fidelity digital cameras currently available obtain their images using technology where individual pixels can only acquire a single color. Since the acquisition of multiple colors is necessary to capture a full colored image, picture resolution is lost due to difficulties in interpolation between non-adjacent, same color pixels. The resulting unsharp images create the need to find a new way to obtain these images without requiring interpolation between adjacent pixels.

An innovative method to capture images with individual pixel cells consisting of three layers of photodetectors stacked vertically upon one another has been created to rectify this problem, but no complete camera system has been fabricated to date. This project integrates a stacked photo detector imager chip into a complete camera, allowing full functionality and control of the photo detector chip to a personal computer. A six layer printed circuit board is fabricated using Cadence Allegro, firmware is written and compiled using Xilinx ISE and software is written in Microsoft Visual Studio 2010.

Type of Report: MS Project Report

Foveon F13 Camera

Master's Project Report

Dave Shelley

November 21, 2012

Abstract - **Most high-fidelity digital cameras currently available obtain their images using technology where individual pixels can only acquire a single color. Since the acquisition of multiple colors is necessary to capture a full colored image, picture resolution is lost due to difficulties in interpolation between non-adjacent, same color pixels. The resulting unsharp images create the need to find a new way to obtain these images without requiring interpolation between adjacent pixels.**

An innovative method to capture images with individual pixel cells consisting of three layers of photodetectors stacked vertically upon one another has been created to rectify this problem, but no complete camera system has been fabricated to date. This project integrates a stacked photo detector imager chip into a complete camera, allowing full functionality and control of the photo detector chip to a personal computer. A six layer printed circuit board is fabricated using Cadence Allegro, firmware is written and compiled using Xilinx ISE and software is written in Microsoft Visual Studio 2010.

1. Introduction

High-fidelity color cameras have found their way into many applications, such as the medical, astronomy, and biology fields. As the demand for progress in each of these fields continues, so does the demand for higher-fidelity imaging. Higher-fidelity imaging can be defined as the ability to capture the most correct image representation possible, limited only by noise [1]. However all imagers fall short of this description due to the presence of errors in the imaging system. Errors can be caused by many different problems, including errors in the data from improper implementation, errors from the imager or algorithms for creating the image, or errors in methodology in how to capture images [1].

Current high-fidelity color cameras, known as color filter array sensors (CFA), suffer from the second kind of error mentioned: errors from the imager or algorithms for creating the image. They fall short of this description due to their inability to capture multiple colors of information at every individual pixel site. As a result, interpolation is required between similar color pixels, leading to a lack of sharpness between non-adjacent pixels. Color information at sites not captured at a given pixel can be derived through many different techniques, however no method leads to a true representation of the color information all of the time. A physical illustration of this technological limitation is seen in Figure 1 [2].

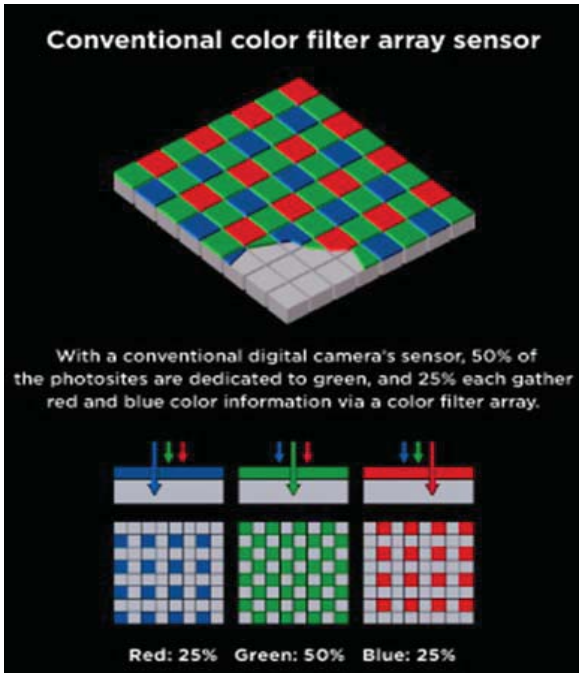

Figure 1: Conventional High-Fidelity Camera Layout

In order to combat this issue, an imaging sensor has been created that has the ability to capture multiple colors at a single vertical pixel cell. Because interpolation of colors between non-adjacent pixels is unnecessary, the full resolution of colors can be correctly represented, and the image becomes one step closer to meeting the "correct image" definition outlined above. This layered system approach is represented in Figure 2 [2].

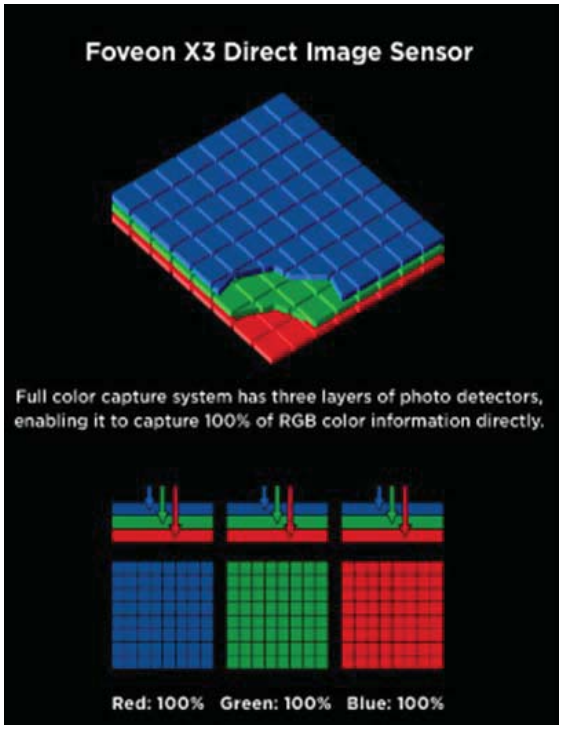

Figure 2: Vertically Stacked High-Fidelity Camera Layout

Due to the novelty of this technology, very few commercial devices have been created that utilize this concept. Of the commercial devices that are available, all are created to be point-and-shoot cameras; meaning none are applicable to the research and academia environment. As a result, this project implements a fully functional camera suitable for the lab environment utilizing this higher-fidelity imaging technique.

The project consists of many separate but highly integrated parts to achieve the desired result of a fully functional camera. A vertically stacked photodiode imager with all support electronics and interfaces is integrated on a six layer printed circuit board (PCB). All software and firmware required to integrate these components together is also constructed to provide the necessary glue logic (in the case of the firmware), and the necessary human interface (in the case of the software) to allow for full functionality. Details of each of these interfaces, support electronics, firmware and software can be found in the following section.

2. Project Flow

The methodology outlined in this paper follows a similar order to that usually taken in the design of a physical electronic device. That is:

A. Design Requirements & Theory
B. Apply Theory to Schematic
C. Layout Schematic to Printed Circuit Board
D. Validate Printed Circuit Board to Requirements defined in Step A

A. Design Requirements & Theory

To achieve the desired result of fabricating a full featured camera system, several components must be integrated together for the system to function properly. The main component of the system, the image capture chip, serves as the centerpiece of the project. All electronics integrated into the system support this chip in some way. These support systems include:

- A data capturing system for acquiring analog imaging data
- Reference signals to properly power the image capture chip
- A digital interface for transferring image data to a personal computer
- A digital interface for controlling the image capture chip

Due to the time constraints affiliated with completing the project, all components utilized to fulfill any of the support systems are common, off-the-shelf (COTS) components. A top level physical description of all components for the project can be seen in Figure 3.

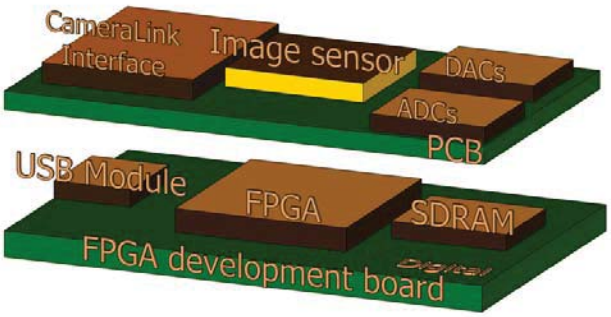

Figure 3: Top Level Camera Description

Each of these individual components is discussed in detail in the following sections.

Foveon F13

The paramount requirement for this project is to implement a high-resolution, high speed color camera incorporating a spectral image sensor with vertically stacked photodetectors in each pixel. The imaging capture chip chosen to meet this requirement is the Foveon F13 imaging sensor seen in Figure 4 [3].

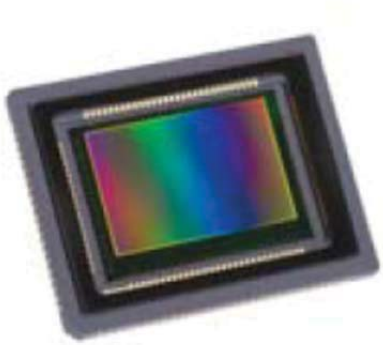

Figure 4: Foveon F13 Imaging Chip

The Foveon F13 has the ability to capture extremely high resolution images with the following key camera parametrics:

- 14.4 Megapixels (2688 Horizontal x 1792 Vertical x 3 layers)
- 40 MHz readout rate
- 3:2 Aspect ratio
- 7.8µm x 7.8µm pixel pitch
- 3 differential analog outputs for Red, Green, & Blue Channels
- Up to 5 frames per second at full resolution
- 2.5V LVCMOS Compatible Digital I/O
- Operates on custom voltages between -.5V and +2.5V

These aforementioned parametrics drive all subsequent support system requirements.

As mentioned above, digital command and control of the device occurs through LVCMOS interfaces. Control of the device consists of many different digital discrete lines (such as chip select lines, photodiode reset lines, photodiode enable lines, and photodiode clocking lines) and many different serial interfaces (such as serial clocks and registers for customization of the imagers performance). The bank of registers controlled by a serial

interface manages horizontal counters, vertical counters, horizontal binning, vertical binning, reset timing, and delay generation [2]. All of these digital imager interfaces are driven through an FPGA at 2.5V LVCMOS voltages.

The only exception to the digital I/O rule is the photodiode clocking lines. It is decided that this digital line is to be connected to a jumper network, allowing flexibility to the source of the clock signal that is driving the chip. Foveon F13 digital clock sources can be from the FPGA or from a Phased Locked Loop (PLL). Additional details specifically relating to the FPGA's digital interfaces can be found below in the section "Glue Logic and High Speed Digital Control Interface".

The second major interface that must be accommodated is the differential analog outputs for red, blue and green pixel information. These outputs are driven in parallel to allow readout of the three colors simultaneously. Each differential pair outputs a signal between .4V to 1.4V at a rate of 40MHz. With this in mind, an Analog to Digital converter (ADC) must be paired with the Foveon's operating characteristics to capture the analog output without any image degradation. This includes not loading the sensor's analog amplifiers, which distorts the analog input signal and provides sufficient resolution and operating frequency to most accurately digitally represent the input signal. Additional details on the ADC chosen to fulfill this goal can be found in the section "Analog to Digital Acquisition".

The last major Foveon F13 interface is the requirement to provide custom reference voltages to operate the chip properly. Necessary voltages can range between +2.5V and -.5V all while requiring not trivial amounts of current from each reference voltage. The common voltages of +2.5V, -.5V, 1V, .5V, 1.4V, .4V draw up to 100mAmps of current for proper functionality. This design challenge led to the implementation of complex reference voltage creation that could accommodate these stringent requirements.

For digital interfaces and photodiode reset lines, much higher current draw of 1A can occur. Luckily, for these interfaces a common voltage of 2.5V is required. Subsequently, a voltage regulator able to produce more than 1A at 2.5V is selected to accomplish this requirement. Additional details on the Foveon's required voltages can be found under the "Reference Voltage Creation" section below.

The Foveon F13 is also chosen because previous generations of Foveon imaging chips have been integrated in the Washington University Advanced Sensors Lab. Their proven track record of a robust and agile sensor provides additional rationale for selecting this sensor.

Analog to Digital Acquisition

As cited above, the Foveon F13 outputs three analog signals representing the image information from red, green, and blue pixel layers. Because the Foveon F13 provides its pixel information in the form of an analog signal, the information must be digitized as quickly as possible to retain data integrity, provide data retention, and perform any post-processing tasks in the digital domain. To correctly represent this analog pixel information, an adequate Analog-to-Digital Converter (ADC) must be selected.

The Foveon F13 outputs pixel information for the red, green, and blue planes at a 40 MHz readout rate, all while supplying a clocking signal to the ADCs. This equates to an ADC requirement of at least 40 Million Samples per Second (MSPS). Note that the Nyquist thorium does not apply in this case because we are not trying to capture an asynchronous signal and recreate it in the digital domain. The Foveon F13 provides a clocking signal so an equal sampling rate to the clocking signal is necessary.

Another key design trait of the Foveon F13 imager is the use of a differential signals for the analog outputs. This is beneficial because higher noise immunity can be achieved with differential signals. Since the ADC utilizing differential signaling only digitizes the difference between the positive and negative components of the differential pair, any common noise on both lines is filtered out. With this in mind, an ADC with differential inputs is required to accept the analog signal from the Foveon F13.

The ADC chosen for analog acquisition of the red, green, and blue color pixels is the Analog Devices AD9245. The AD9245 has the ability to capture analog signals with the following parametrics:

- 14 Bit signal resolution
- 40 MSPS
- 2.5v or 3.3v Operation
- Differential Input
- Flexible Analog Input allowing range of differential inputs

The 40 MSPS and differential input characteristics accommodate the two major features mentioned above, however the flexible input and output voltage range allows easy integration with different peripherals. This project does not require the use of one voltage level with which all other components must be compatible. Additionally, this ADC can accept a broad range of differential input voltages, not just balanced differential inputs. This is necessary because the differential output of the Foveon F13 is always positive between .4V and 1.4V. Lastly, the 14 bit signal resolution provides more than enough resolution to accurately represent any input voltage with high precision.

To provide additional noise immunity and prevent any sensor loading, complex analog buffering circuitry is developed between the Foveon F13 and the ADCs. This buffering circuitry consisted of an operational amplifier setup in a unity gain voltage follower configuration, followed by a low pas filter to remove any high frequency component that could undesirably skew the ADC measurement. A block diagram of this interaction can be seen in Figures 5, 6, and 7.

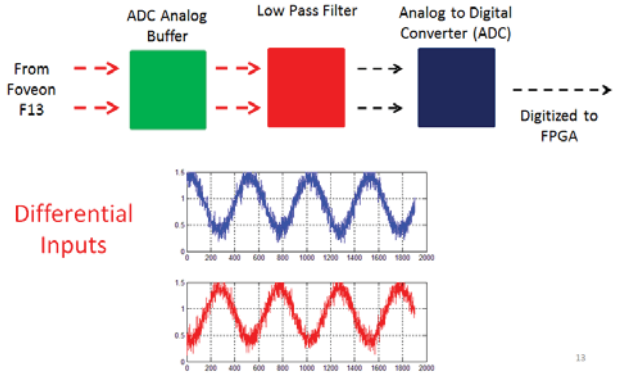

Figure 5: Foveon F13 Differential Output and Buffering

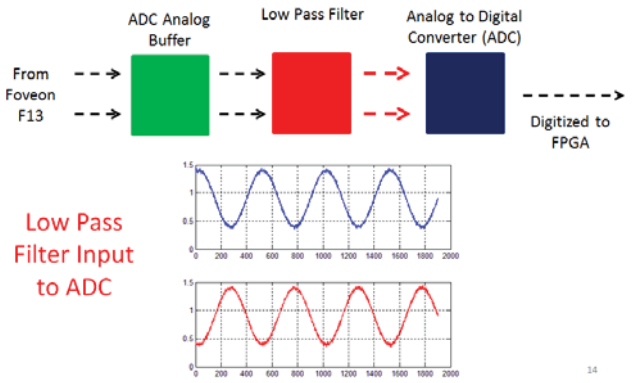

Figure 6: Effect of DC Blocking Capacitors

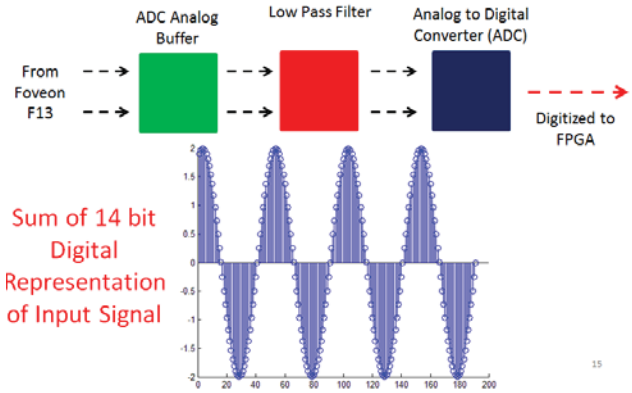

Figure 7: Sum of 14 Bit Digital Representation of Input Signal

As the Foveon F13 analog signal propagates from the left to right as seen in the figures above, the first step in the analog filtering scheme is the ADC Analog Buffer. This buffer serves to provide high input impedance to prevent loading of the Foveon analog output amplifiers, provide low output impedance to the low pass filter capacitors, and have high current drive to the ADCs. Since the low pass filter allows frequencies to pass that are below the RC time constant of 1/(Resistance*Capacitance), appropriately selected components are chosen to satisfy the ability to pass a 40 MHz signal. In this application, .1µF and 33Ω components are chosen because this leads to a 3dB cutoff point of approximately 48MHz. This combination is appropriate because the frequency of the input signal is 40MHz. Since the cutoff frequency is 48MHz, this filter implementation allows the input frequency to pass but not allow much higher frequency noise to contribute to the input signal, reducing input noise.

Last in this circuitry chain is the Analog to Digital Converter. The analog to digital converter accepts this filtered differential input and creates a 14 bit parallel output digital representation of this value. The ADC is configured to accept a differential voltage range of 0 to 2 volts, which falls well within the differential range of the Foveon's output of .4 to 1.4 volts. When sampled at 40MHz as driven by the Foveon F13, the ADC can be properly clocked to accommodate acquisition of all pixels at a 40MSPS rate. The 14 bit parallel output from all 3 ADCs is connected to an FPGA for post processing and data export on a personal computer. An out-of-bounds bit is also supplied to the FPGA when the input differential signal falls outside the 0 to 2 volt input range.

It is also decided that the ADC clock should be connected to a jumper network, allowing flexibility to the source of the clock signal that is driving the ADC. ADC digital clock sources can be from the FPGA, a Phased Locked Loop (PLL) shared with the FPGA, its own Phased Locked Loop independent from the FPGA, or from the Foveon F13. By accommodating different sources of this clock signal, different delays and frequencies can be derived outside of the signal provided by the Foveon F13. For

normal operation, the ADCs are driven by the Foveon F13.

The ADCs are powered by two independent power sources. These two power sources, consisting of 3.3V and 2.5V, are chosen for two reasons. First, the 3.3V power supply is chosen to power the analog circuitry within the ADC. This voltage selection allows the analog front end of the ADC to create a wide enough input span to accept the input voltages from the Foveon F13. Since the maximum 3.3V current draw is 55mA per ADC, it is driven by a dedicated voltage regulator. Secondly, the 2.5V power supply is chosen to power the digital logic within the ADC. This allows the digital outputs of the ADC to properly interface with the input voltages of the FPGA. Since the maximum 2.5V current draw is 5mA per ADC, it is driven by a shared voltage regulator which powers all digital logic on the board.

It should also be noted that the effective number of bits in this application is not equal to the actual number of bits in the ADC. With a sampling frequency of 40MHz, the effective number of bits in the ADC is approximately 12 bits, not 14. This is because the ADC is able to represent voltages below the noise floor of the input signal. Since signals below the noise floor do not contain usable information, these bits are effectively inadequate for contributing to the overall measurement. The effective number of bits is a calculation of the number of bits that reside above this noise floor and can subsequently produce useable information.

Reference Voltage Creation

The Foveon F13 requires many different voltages and currents to function properly. These voltages must be created external to the Foveon F13, as it does not derive them internally. Besides the digital reference voltage of 2.5V for all digital communication to other components, the F13 necessitates the creation of many different custom voltages, including -.5V +2.5V, +1V, +.5V, +1.4V, +.4V among others. Additionally, many of these custom voltages draw large amounts of current; therefore simple voltage regulators are not be sufficient for meeting this requirement.

To combat this complicated problem, a network of voltage regulators, charge pumps, Digital-to-Analog Converters (DACs), and Op-Amps are utilized to fabricate these non-trivial reference voltages. The regulators and charge pumps create the voltages at the corners of the voltage range of -.5V and +3.3V. The DACs are used to fabricate any custom voltage necessary between the -.5V and +3.3V input extremes. The Op-Amps serve as voltage followers where adequate current can be drawn from them without unnecessarily pulling down the DACs voltage rail. It also serves as a buffer between the Foveon F13 and the DACs to provide isolation from the power circuitry. The source of power for all this regulation circuitry comes from an external power supply providing 5V to a development board which distributes the 5V to the printed circuit board. The development board is discussed in the “Glue Logic and High Speed Digital Control Interface” section. The interconnectivity of these various devices can be seen in Figure 8.

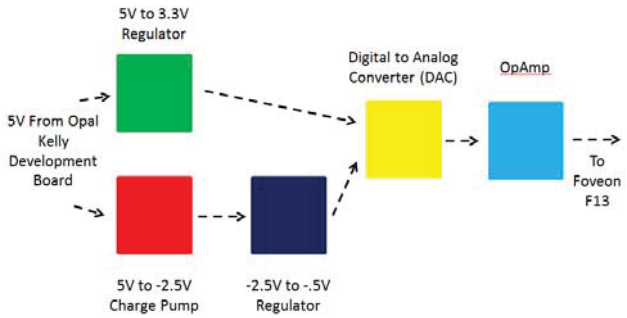

Figure 8: Custom Voltage Creation

Upwards of 20 different custom voltages are required to adequately power the Foveon F13 imager. To properly supply these voltages, the order of operation is as follows:

1. Input 5V from power supply
2. Use charge pumps to create -2.5V and regulators to create 3.3Vdc from 5V (5 pairs total)
3. Regulate -2.5V down to -.5V
4. Input each pump/regulator pair into dual rail inputs of DACs
5. Program DAC to output voltage between +3.3 to -.5V (20 channels total)
6. Since DAC cannot drive sufficient current for this application (limited to 50mA), send DAC output to Op Amp with unity gain allowing up to 200mA drive current per channel (20 total)

For instances of the Foveon F13 and other support electronics where more common voltages of 3.3V and 2.5V are needed, large drive voltage regulators are employed to provide up to 2Amps of current per rail. Utilizing Op-Amps driven by DACs does not provide sufficient current for this type of application. In many instances, the maximum current draw of a single rail is up to 1Amp. The two components requiring this large current draw are the photodiode reset lines as well as the entire 2.5V digital reference voltage rail for all digital circuitry.

High Speed Image Transfer Interface

Now that the Foveon F13 is powered properly and is outputting analog image information to the ADCs, a method of transferring this large quantity of digital data quickly to a personal computer is required.

Several common personal computer interfaces were considered due their large data bandwidth abilities. In order to find which interface is sufficient for this camera, it must be determined what type of bandwidth is needed to transfer these images in a reasonable amount of time. To find required bandwidth, we use the following equation assuming one full camera frame a second consisting of all three colors:

$$Bandwidth = \frac{14.4\ MegaPixels}{frame} x \frac{14\ bits}{Pixel} x \frac{1\ frame}{second} = \frac{202\ Mbits}{Second}$$

Equation 1: Required Throughput Bandwidth for 1 Frame/Sec

The camera can take up to 5 frames per second, therefore the maximum bandwidth necessary for this application becomes 1010Mbits/second, or approximately 1Gbit/second. With this maximum bandwidth requirement calculated above, we can explore the possibility of using different off-the-shelf interfaces.

USB 2.0 was considered, but is too slow to transmit multiple full resolution images in a reasonable amount of time. Since the maximum theoretical throughput of USB 2.0 is 480Mbits/second, it would require several

seconds to transfer 5 full resolution color images. This was deemed unacceptable because at 5 images per second, the interface cannot sustain the necessary throughput and image information is lost. Assuming no image retention within the camera is available, this interface is not fast enough.

Although many higher speed busses are becoming available which accommodate this kind of throughput (USB 3.0 at 5Gbit/sec, Thunderbolt at 10Gbit/sec, SATA 3.0 at 6Gbit/sec, 1Gbit Ethernet), these interfaces are still in their infancy and do not have many off-the-shelf components that could be integrated in a timely manner. Additionally, many of these interfaces require several chips and complicated interfacing logic to work properly. As this technology matures and off-the-shelf interface Intellectual Property (IP) becomes available, these options have the potential to be more beneficial in the future.

Because of the shortfalls mentioned above, a camera interface has been developed and currently exists that supports the camera's throughput needs. The CameraLink interface is designed for the purpose of standardizing scientific and industrial video products including cameras, cables and frame grabbers [4]. The CameraLink interface in its highest speed mode can sustain data throughput of 2.38Gbit/second. This interface is more than sufficient for 5 frames per second bandwidth needs of 1Gbit/second. Furthermore, many laboratory cameras currently use this interface with readily available software, thus allowing rapid prototype proof-of-concepts to be created.

Another advantageous feature of the CameraLink protocol is the integration of RS422 lines within its interface. The RS422 interface connects with a CameraLink compatible frame grabber on a personal computer to allow bidirectional communication between a computer and the camera for command and control purposes. This provides simplification in interfacing to the camera because control and response of the Foveon F13 as well as image data from the Foveon F13 can all be obtained within one convenient interface.

The CameraLink circuitry is illustrated in the block diagram of Figure 9.

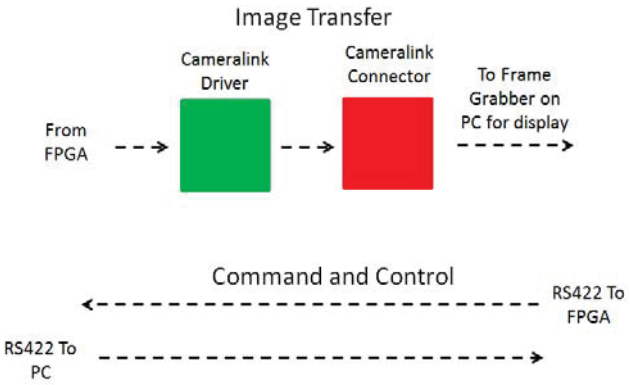

Figure 9: CameraLink Interface

As displayed above, firmware within the FPGA drives the CameraLink chip through a parallel 28 bit interface. The 28 bits received by the CameraLink chip are then clocked into the chip through a dedicated CameraLink clocking line from the FPGA. The chip serializes the incoming bits and sends them out through 4 differential pairs in the CameraLink connector alongside a single differential pair clock signal. These 5 differential signals are seen at the CameraLink receiver chip located within the frame grabber, are reassembled into their original 28 bit parallel configuration, and are saved within the frame grabber. This process is repeated until all 202Mbits of a single frame of data are sent across the interface. The entire

frame is retained and displayed on the PC for further processing.

For the CameraLink interface, 3.3V is provided to the chip for all digital logic as well as the driver voltage for all differential lines on the interface output. This allows the device to interface to the FPGA. This 3.3V is sourced from the 2A 3.3V regulator on the board.

3.3V digital logic voltage is chosen due to its compatibility with 2.5V digital logic. This is because with a digital logic voltage of 3.3V, the CameraLink chip recognized a high level input voltage as any value between 2V and 3.3V according to JEDEC Standard No. 8-C [5]. Next, 2.5V LVCMOS guarantees that the high level output voltage can be any value between 2V and 2.5V according to JEDEC Standard No. 64-A [6]. Therefore, since the FPGA would only be driving logic into the CameraLink interface over a very short distance and the minimum output logic high voltage is larger than the minimum acceptable input voltage of 3.3V logic, this engineering tradeoff is deemed acceptable.

It is also worth noting that for the beginning portion of the project, the RS422 interface within CameraLink is not utilized for Command and Control. In its place, the USB interface on the FPGA development board is utilized because it has been fully tested by the development board vendor, thereby allowing more attention to be given to other subsystems. During the system verification portion of the project, this allowed for easy insight into system behavior in the firmware without having to worry about whether the computer to camera RS422 interface is working properly. Once all systems have been validated in the camera, an easy transition can be made within the firmware to transition from USB to RS422 as the command and control interface.

Glue Logic and High Speed Digital Control Interface

Lastly, and perhaps most importantly after the Foveon F13, is the glue logic that ties all the support electronics together to allow the Foveon F13 to function properly. Glue logic tasks include properly driving the DACs to setup the Foveon F13's reference voltages, capturing digital image data from the analog to digital converters, formatting digital image data for CameraLink export, providing a PC interface through USB, and interfacing with the Foveon F13's digital interface for command and control. For all these tasks to be completed simultaneously in a single electronic component, a powerful device is required.

The component chosen to meet these requirements is a Field Programmable Gate Array (FPGA). An FPGA can meet all of the requirements described above simultaneously, while also providing flexibility. These devices provide many input and output pins to be universally assigned to any task through the construction of firmware resident on the module.

Due to the desire to use rapid prototyping materials whenever possible, a FPGA is chosen that is already integrated into an off-the-shelf development board. This allows more attention to be given to the development of the Foveon F13 support hardware rather than to the development of FPGA support hardware.

The XEM3050 development board by Opal Kelly provides a reconfigurable development

platform allowing the integration of several board components and interfaces required by this project, including:

- Xilinx Spartan-3 FPGA
- 110 I/O pins on 2 Samtec Connectors
- USB 2.0 Interface
- 64-MByte of SDRAM
- 1-MByte of SSRAM
- 1-MByte of Flash memory
- Provides 5V, 3.3V, 1.8V over expansion connectors
- Powered via 5V external power supply or on-board USB power
- Integrated PLL Source up to 150MHz

A picture of the development board can be seen in Figure 10.

Figure 10: Opal Kelly XEM3050 Development Board

The presence of the Xilinx Spartan-3 FPGA and the ability to interface to it through 110 I/O pins serves as an excellent basis for tying all of the project's components together. These I/O pins can be set to 2.5V digital LVCMOS logic, thus allowing compatibility with all support electronics. All 110 I/O pins are utilized in interfacing to the Foveon F13 and its support electronics.

Foveon F13 interfaces include all discrete control lines such as reset lines, counters, pixel row and column enable, imager power down. Additional Foveon F13 to FPGA connections include the serial control interface consisting of a serial data line, a serial clock line, and a serial enable line, as well as a dedicated clock line for shifting pixel data out of the analog differential outputs.

The ADC support electronics also consume a large portion of this I/O interface. 14 data bits and an overflow bit from each ADC encompass all the information sent to the FPGA. This information is processed and temporarily stored within the firmware until an entire frame's worth of image information is shifted out of the Foveon F13. A single digital line is sent from the FPGA to the ADC clock line should the FPGA clock be selected to drive the ADC via a jumper on the board.

The DAC support electronics only require 5 of the 110 I/O pins to operate since the 5 DAC components used to create 20 reference voltages are tied together serially in a daisy-chained shift register format. Serial DAC control data is shifted bit-by-bit out of the FPGA and is clocked after each bit is asserted on the serial line. This is repeated until all bits are shifted out to all DACs. Using this approach, only 5 development board pins are required to drive a virtually infinite number of DAC outputs.

The CameraLink support electronics consume the last of the remaining digital pins on the development board. This 30 pin interface consists of 28 image data outputs from the FPGA to the CameraLink driver, as well as a clock line and a chip enable line. A balanced differential input is also available for dedicated

CameraLink RS422 communication to a frame grabber on a PC.

The remaining pins on the I/O connectors on the development board serve as the means to distribute power to all components on the camera. A separate power connector installed on the board allows a 5V power supply to provide all the power the camera needs. Power can also be drawn from the onboard USB 2.0 connector, however this project requires more current beyond what USB 2.0 can provide at 500mA. The 3.3V and 1.8V regulators on the development board are not be used due to their limited current sourcing.

Of additional interest is the inclusion of the USB 2.0 interface. As previously mentioned, an Opal Kelly USB 2.0 Application Programming Interface (API) and firmware IP is provided with the development board, allowing an easy and proven method to transfer control information from the PC to the development board. The use of the RS422 lines in the CameraLink will serve as the High Speed Digital Interface in the future, but due to the time constrains of this project, the USB 2.0 interface is used at this time.

In addition to hardware, firmware must be created that drives the digital logic of the Foveon F13 and its support electronics. Supplemental firmware must also be written to perform the internal conversion necessary to format the digitized input data from the ADC into a format which the CameraLink interface can understand.

This also holds true for the FPGA to Foveon F13 interface. The firmware must be programmed to accept command inputs from the USB 2.0 interface (and RS422 interface in the future) and drive the Foveon F13 digital lines appropriately. This means asserting chip select lines on the DACs, ADCs, CameraLink, and F13 chips, providing unique clocking signals to different components, populating and shifting out register information to the Foveon F13 and DACs as well as providing relevant discrete control data to all devices. This firmware is written using the Xilinx ISE development platform using the Verilog hardware description language.

Lastly, software is written to provide a means to control the camera over the USB 2.0 interface for the early stages of the project. This provides a human interface to the camera through a personal computer. With such interface, custom voltages can be relayed through software, down to the FPGA, and shifted out over registers to the DACs. This interface also provides the capability to determine status of the FPGA's register states to be sent to the DAC and the Foveon F13. The software is written using the Microsoft Virtual Studio 2010 using the C++ programming language.

Engineering Interfaces

Several supplemental features are provided on the PCB as a means to more easily test the board's functionality as well as provide sufficient board flexibility.

The first supplemental feature is the jumpers residing on the board to provide flexibility for the following system parameters:

1. All FPGA Bank Voltages
   - 3.3V
   - 2.5V
2. Foveon F13 Clock Source

- FPGA Directly
- Phase Locked Loop (PLL)

3. Photodiode Reset Voltage source
   - Dedicated Regulator
   - DAC driven Op Amp
4. ADC Clock Source
   - PLL
   - FPGA Directly
   - F13 Delayed

System flexibility in these areas allows features to be customized depending upon camera application without needing the PCB to be altered to accommodate these changes.

The second supplemental feature is the addition of voltage test points to the PCB. This allows for more rapid verification and test of the system because many hard to reach signals are available on the surface of the board with convenient oscilloscope hooking locations. Available voltage test points include:

1. All Reference voltages
   - DAC sourced
   - Voltage regulator sourced
2. Key serial control/clock lines between the FPGA and Foveon F13

Now that requirements and theory have been derived for all system components, a holistic picture of the entire camera system can be conveyed. This diagram can be found in Figure 11.

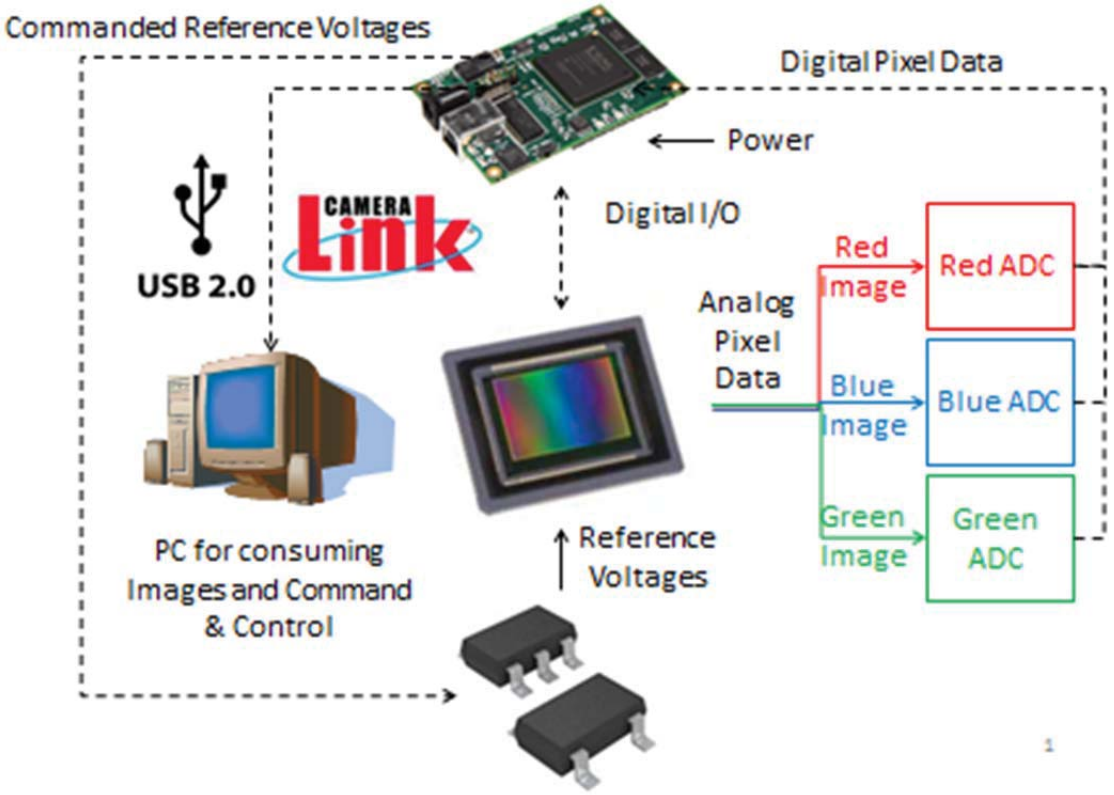

Figure 11: Top Level Project Block Diagram

B. Apply Theory to Schematic

Once the design requirements and theory from the previous section is established, a schematic is created. This schematic encompasses all Foveon F13 interfaces, all support electronics and all external interfaces required to make the top level design theory as seen in Figure 3 a reality. The schematic and the PCB layout are both completed using Cadence Allegro PCB Designer. The schematic can be seen in Figure 12.

Much like the requirements and theory section, the schematic is segregated into several groups as defined by the component's purpose.

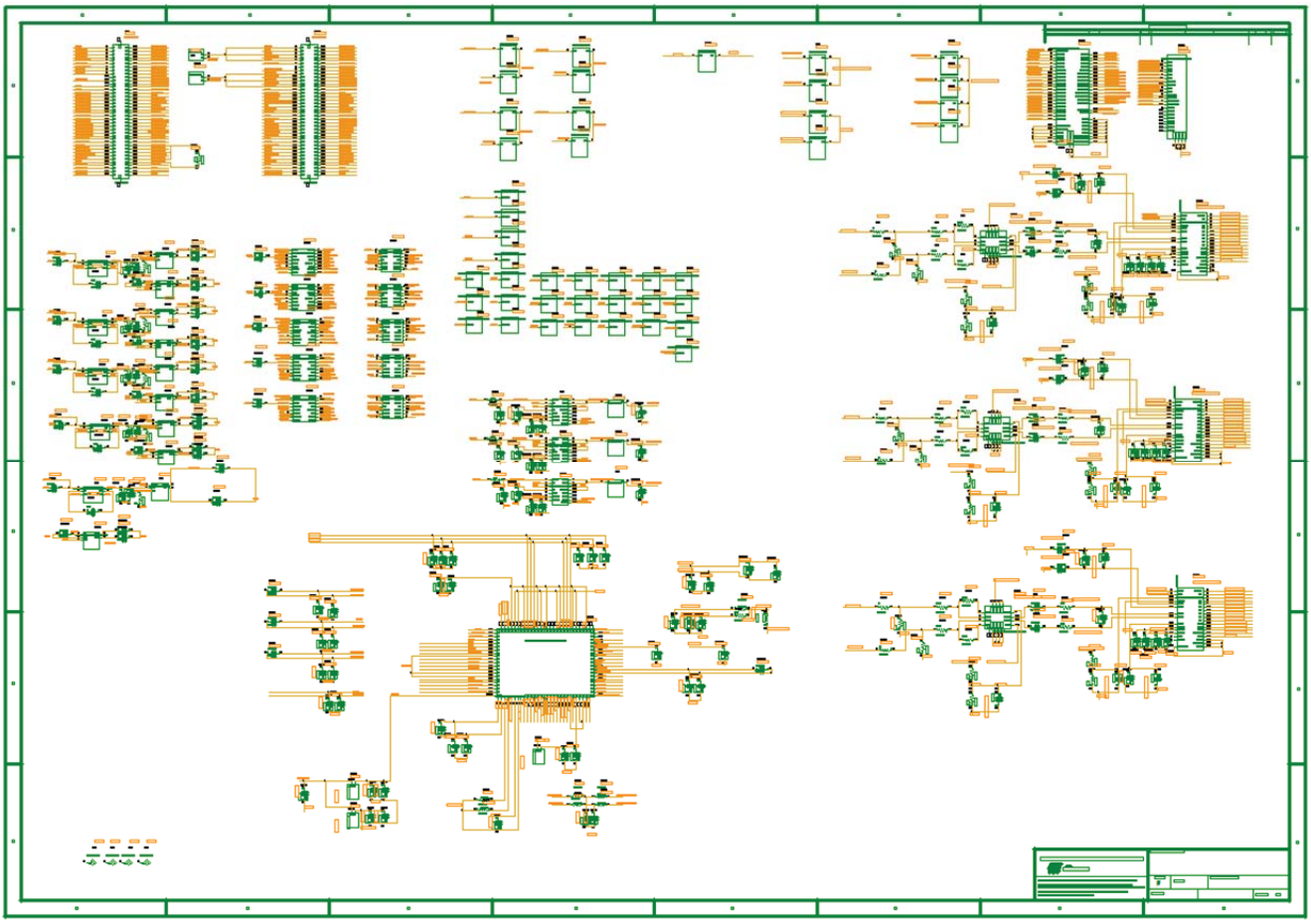

Figure 12: Complete PCB Schematic

Foveon F13

The Foveon F13 serves as the centerpiece of all other electronics in the schematic. The schematic accommodates the Foveon F13 by integrating a 100 pin socket in which the F13 is seated to interface with all support electronics. This socket also provides flexibility that the F13 can be removed and stored in a safe electrostatic discharge (ESD) protective environment during transport.

High and low frequency capacitor networks are added to the inputs of all analog

lines to filter undesired frequency components of input signals. A top level schematic view and a lower level component view can be found in Figures 13 & 14 respectively.

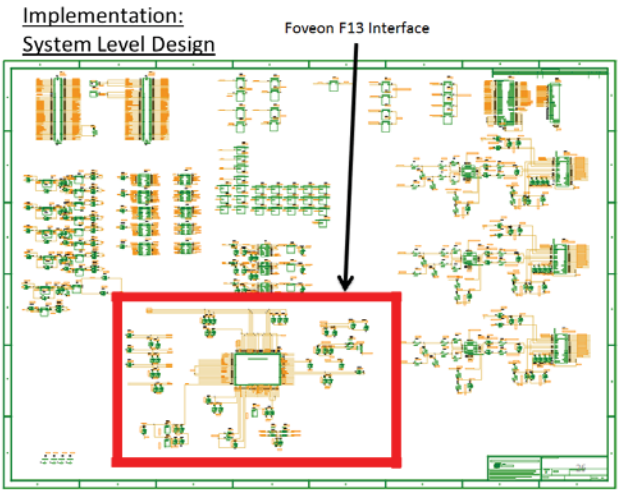

Figure 13: Top Level Schematic: F13

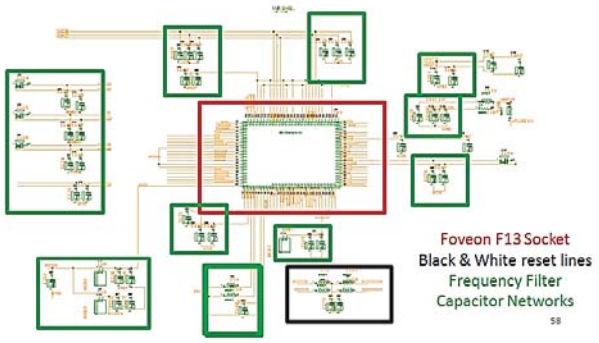

Figure 14: F13 Component View

Analog to Digital Acquisition

The analog to digital interfaces are identical for the red, green and blue color acquisition paths. The complicated differential circuitry path can be seen in the lower level component view. Also of note is the derivation of various ADC reference voltages. For this implementation, the ADC uses its internal reference voltage of 1V with an acceptable span of 0 to 2V. This setup can be seen in Figures 15 and 16.

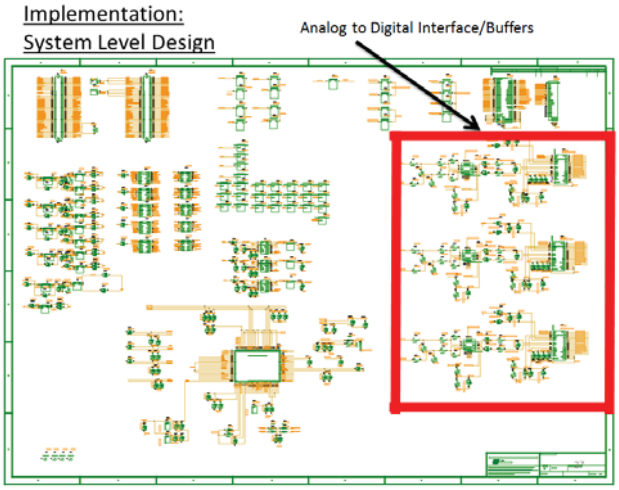

Figure 15: Top Level Schematic: ADC

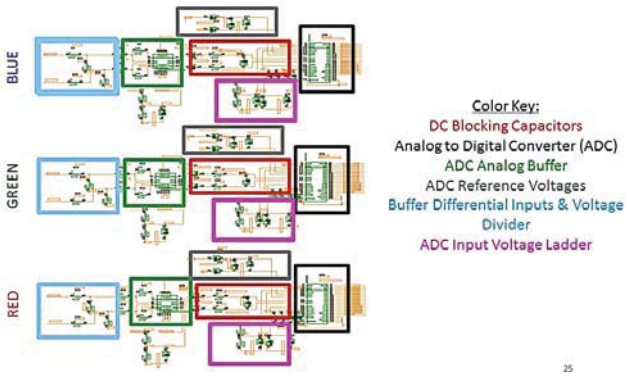

Figure 16: ADC Component View

Reference Voltage Creation

Reference voltage creation consists of 5 instances of the charge pump, regulators, DAC,

and OpAmp circuitry described above. Three dedicated, high current, voltage regulators provide the voltage necessary for 2.5V digital logic, 3.3V ADC power and 2.5V photodiode reset voltage. The input power for the custom voltage circuitry, as well as the dedicated high current regulators, comes from the Opal Kelly Development board through the J2 and J3 connectors. This setup can be seen in Figures 17 and 18.

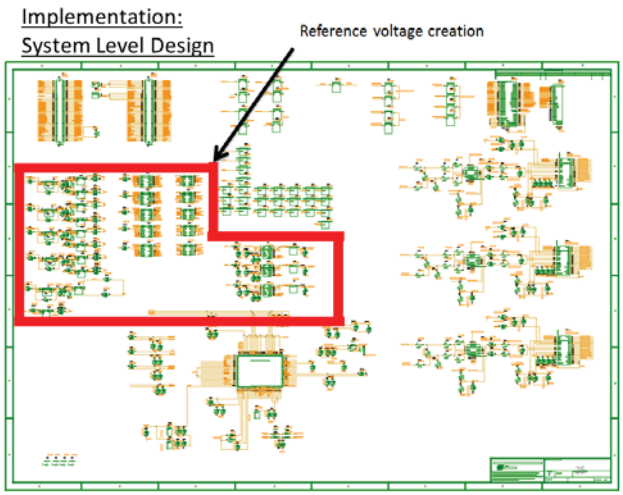

Figure 17: Top Level Schematic: Reference Voltage

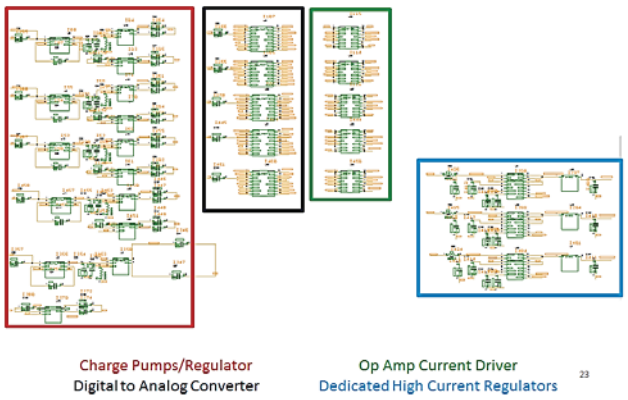

Figure 18: Reference Voltage Component View

High Speed Image Transfer Interface

As previously mentioned, the CameraLink interface consists of two simple components to accomplish its goal: A CameraLink driver interface and an industry standard CameraLink connector. This leads to the straight forward implementation into the schematic as seen in Figures 19 and 20.

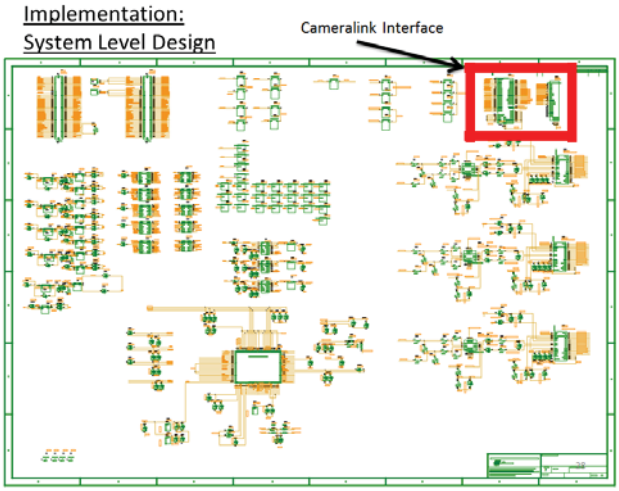

Figure 19: Top Level Schematic: CameraLink

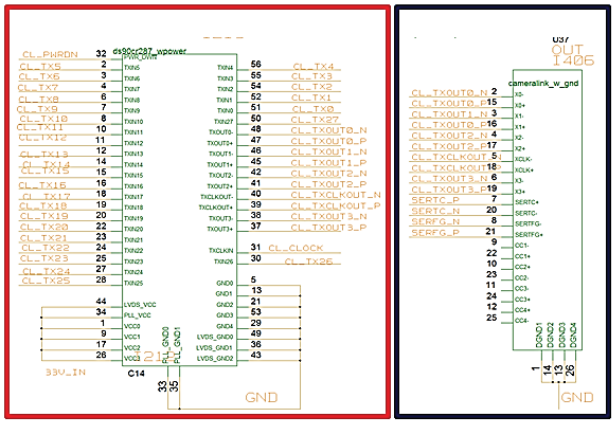

Figure 20: CameraLink Component View

Glue Logic and High Speed Digital Control Interface

Because of the use of an off-the-shelf FPGA Development board, the firmware glue-logic as well as the USB 2.0 interface remains on the development board. To interface to these features, two 80 pin Samtec connectors are provided on the development board. Of these 160 available pins, 110 pins are assigned to digital I/O, with the rest assigned for power transfer between the development board and the PCB daughter board. A JTAG interface is also provided at this interface by the development board but is not routed to external pins on the PCB, because an eternal JTAG header is already made available on the daughterboard. Moreover, the RS422 connection from the CameraLink interface is supported at these connections by defining two pins to be the positive and negative terminals of the balanced differential pair and by providing a 100Ω resistor between the two terminals to allow sufficient current to be driven between the leads. Several ferrite beads are installed at the power output from the Opal Kelly development board to suppress high frequency noise on the input voltages. Figures 21 and 22 depict the development board PCB daughterboard interface.

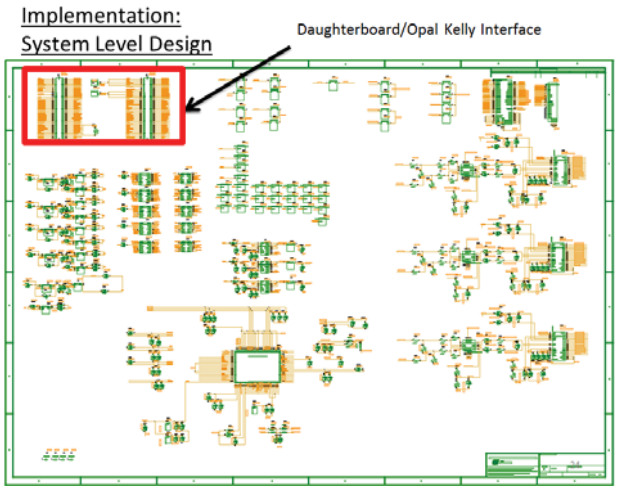

Figure 21: Top Level Schematic: Development Board Interface

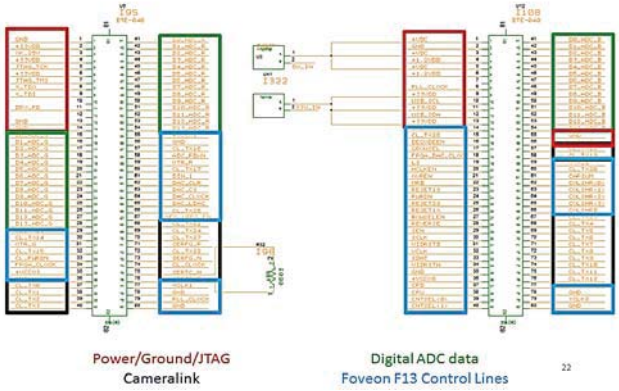

Figure 22: Development Board Interface Component View

Engineering Interfaces

The final component to be instantiated within the schematic is the engineering components. The jumpers that allow design flexibility for clock and voltage sources are implemented by allowing a jumper to select one of multiple inputs to a single output. This employment can be thought of as a manual multiplexing technique. The test point

components accomplish their goal by connecting serially with signals on the board. Their goal is not to influence board behavior by connecting external components, but rather to gain insight into board signals that would not usually be available without this kind of application. Jumper schematic location and configuration can be found in Figures 23 and 24. The voltage test point schematic location and configuration can be found in Figures 25 and 26.

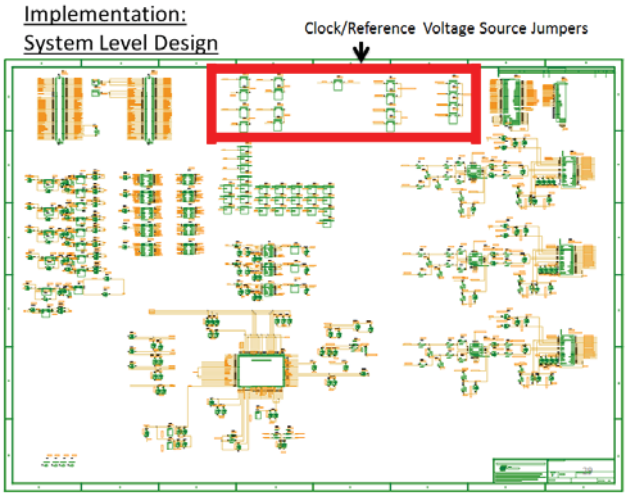

Figure 23: Top Level Schematic: Jumper Interface

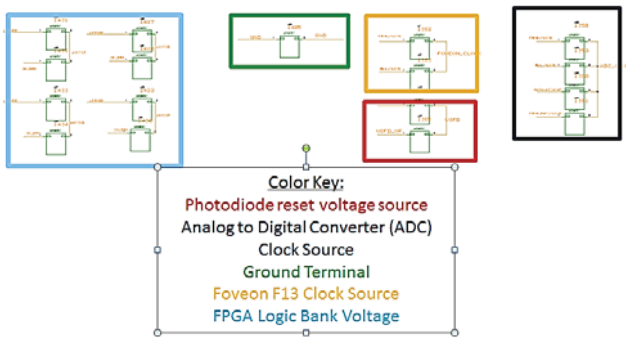

Figure 24: Jumper Component View

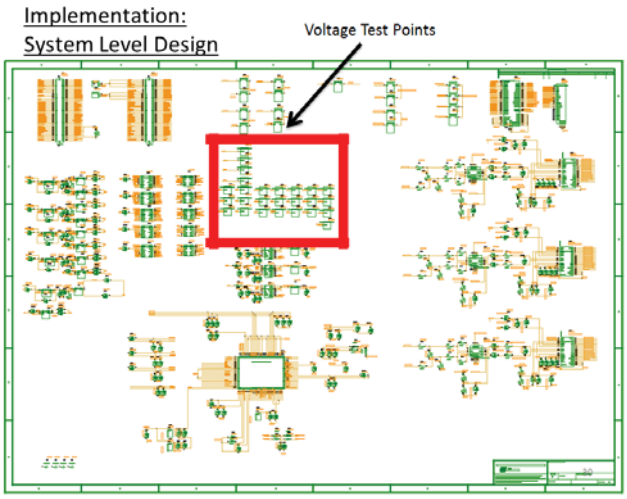

Figure 25: Top Level Schematic: Test Point Interface

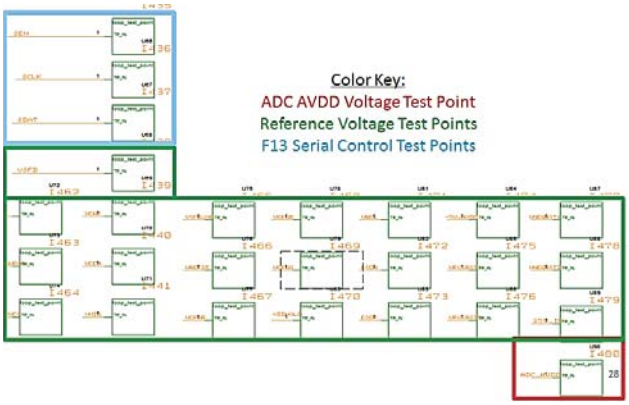

Figure 26: Test Point Component View

C. Layout Schematic to Printed Circuit Board

Once a schematic meeting all requirements has been solidified, the layout stage of the PCB can begin. This part of the process is not as simple as it may seem, due to several underlining physical PCB requirements which have not yet been addressed. These requirements stem from the desire to provide a

noise-free environment in which to capture images, thus coming as close to a high-fidelity imager as possible. Taking these steps allows the device to mitigate the effects of noise as much as possible.

To curtail this noise, a six layer PCB board is fabricated, which consists of 4 signal layers, a ground layer, and a power layer. The division of these layers to perform unique tasks helps reduce this error by keeping electromagnetic interference (EMI) minimized.

Supplementing the use of a 6 layer board is the requirement to make analog signal traces as short as possible from source to destination. By keeping these distances small and trying to digitize the signal as quickly as possible, the least amount of analog signal corruption can occur.

Another requirement to minimize noise in the PCB is to always run analog signals perpendicular to other analog or digital signals, thus preventing cross-talk between the two traces. This is always emphasized between adjacent layers of the board as well as the analog portion of the ADC circuitry. Ensuring this portion of the imager is as short and untainted as possible ensures the noise in the system is kept to a minimum.

Other physical system requirements are device size and packaging. The PCB is to maintain the approximate size of the Opal Kelly form factor of 75mm long by 50mm wide. This allows camera packaging of all circuitry into one enclosure to easily occur. Furthermore the PCB is also to mate with the two Samtec connectors on the bottom of the Opal Kelly development board.

Additionally, logical packaging of various camera subsystems is enforced. This means that ADC components are collocated with other ADC components, reference voltage components are collocated with other reference voltage components etc. Lastly, mounting holes are provided to mechanically secure the daughterboard to the Opal Kelly development board.

By meeting the above steps in conjunction, noise due to physical implementation can be minimized.

6 Layer PCB Stackup

A six layer PCB is utilized in the implementation of the daughterboard. These six layers consist of a top and bottom layer where both device connections and signals reside, two sub layers for additional signal routing, a power layer for power distribution and a ground layer. A visual depiction of the PCB stackup can be seen below in Figure 27.

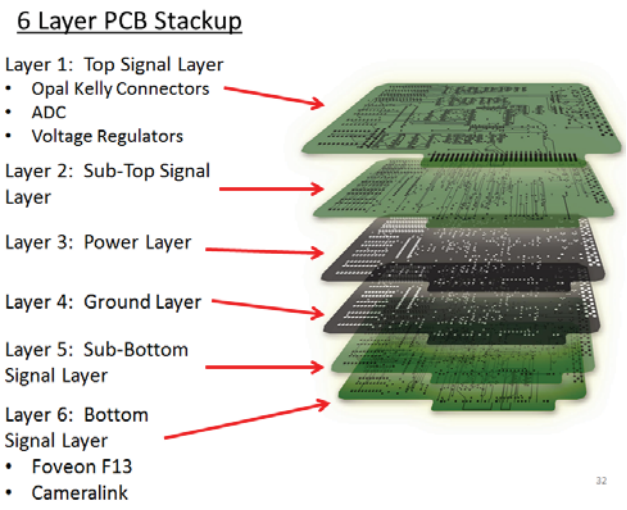

Figure 27: 6 Layer PCB Stackup

Layer 1: Top

The top layer of the board consists of most reference voltage components, the Opal Kelly/Daughterboard Samtec interface connectors, and a portion of the ADC circuitry. The reference voltage portion residing on this side of the board is the positive and negative voltage regulators, the charge pumps, 3 of the 5 DACs, 5 OpAmps, and a dedicated 3.3V high current regulator. The remaining portion of the reference voltage circuitry resides at this exact location on the other side of the board.

The ADC portion residing on this side of the board is the ADCs themselves. Note that since close proximity to other analog ADC front end components is desired, the front end components reside at the exact same location on the other side other the board. A component and signal layout of the top layer can be seen in Figures 28 and 29 respectively.

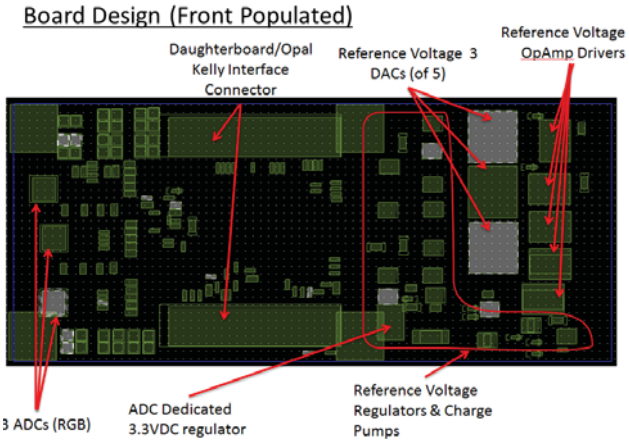

Figure 28: Top Layer Component Layout

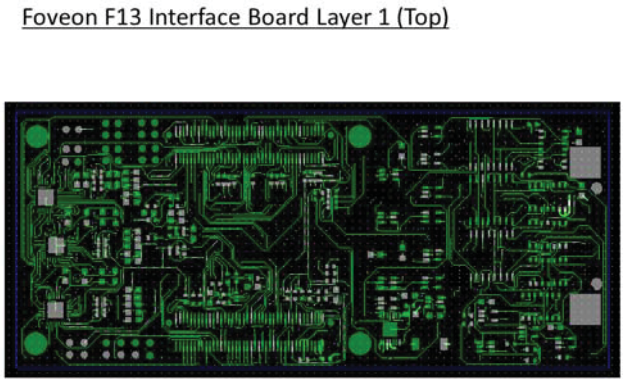

Figure 29: Top Layer Signal Layout

Layer 6: Bottom

The bottom layer of the PCB board consists of a portion of the ADC circuitry, all jumpers, all voltage test points, the Foveon F13 interface socket, a portion of the reference voltage circuitry, and all CameraLink components.

The ADC portion residing on this side of the board is all resistance networks, low pass filters and buffers. Also of note is the nearness of the buffers to the Foveon F13. This closeness satisfies the desire to have all analog components as close as possible to one another.

Secondly, all clock and reference voltage select jumpers are positioned near one another so in the event that the complete camera is packaged within an enclosure, all jumpers can be readily accessed at one location. The voltage test points are positioned around the board wherever open space is available while retaining neat packaging wherever possible. In this case, test points reside either above, below or to the right of the F13 interface socket.

The remaining reference voltage circuitry is positioned right above the other

components on the top side of the board. These components include the last 2 DACs of 5, as well as two 2.5V high current regulators for photodiode reset and digital logic. Most biasing capacitors and resistors for the reference voltage circuitry are also found on this plane.

Lastly, the CameraLink chip and connector are found on the far right side of this layer. The CameraLink connector is made flush with the right side of the board to allow the connector to stick out to external interfaces when the camera is packaged within an enclosure. The connector is also screwed down to the board to prevent trace and pin cracking due to stresses incurred during cable mating and removal. A component and signal layout of the bottom layer can be seen below in Figures 30 and 31 respectively.

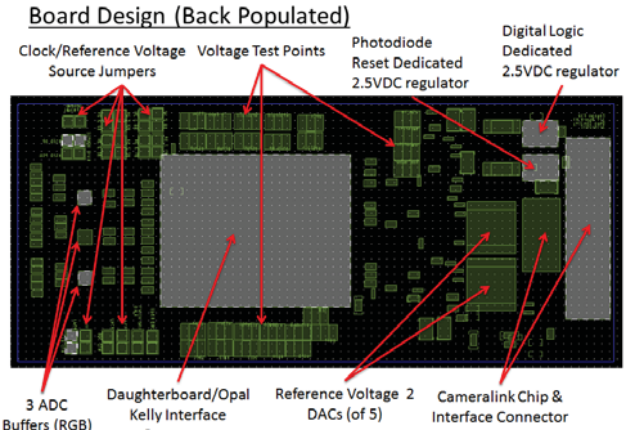

Figure 30: Bottom Layer Component Layout

Foveon F13 Interface Board Layer 6 (Bottom)

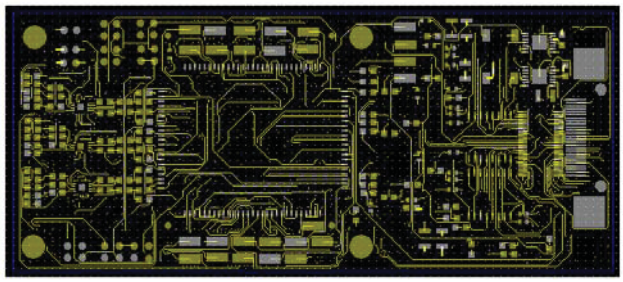

Figure 31: Bottom Layer Signal Layout

Layer 2: Sub-Top & Layer 5: Sub-Bottom

Layers 2 and 5 serve as the signal layers for most of the board's routing between components. Sub-Top Layer 2 consists of signals to be routed from Layer 1 Top level components, and Sub-Bottom Layer 5 consists of signals to be routed from Layer 6 Bottom level components. Also of note is that signals on a given layer cross perpendicular to signals on adjacent layers whenever possible. Lastly, corners are mitered whenever possible, as it is good design practice and helps prevent reflections in high frequency signals. Trace layout for the Sub-Top and Sub-Bottom layers can be seen in Figures 32 and 33.

Foveon F13 Interface Board Layer 2 (Sub-Top Signal Layer)

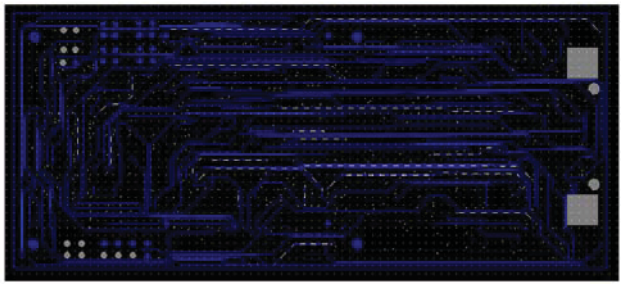

Figure 32: Sub-Top Signal Layer

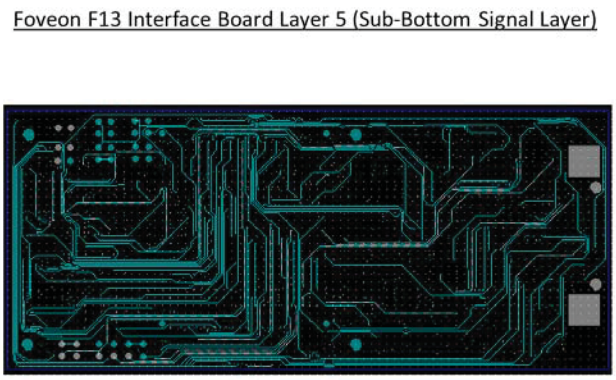

Figure 33: Sub-Bottom Signal Layer

Layer 3: Power

Layer 3 serves a very important role in this PCB's design: the power distribution plane. A power plane or a wide power trace is chosen to distribute power within the board for a number of reasons.

First, a power plane is chosen due to the desire to have a strict control over impedance of a given input power. If a small trace is chosen to power two components on opposite ends of the board, a slight voltage variance would occur due to the voltage drop in the trace between the two components. A large power plane or trace minimizes this impact.

Secondly, utilizing a power plane provides a small loop area from power source to load and then to ground. Since electromagnetic interference (EMI) is directly proportional to the loop area that power signal must travel, minimizing this area also minimizes EMI.

Another reason to utilize a power plane or wide trace is the ability to diminish crosstalk between traces. If a signal is close to its reference voltage plane, crosstalk is minimized [7].

Because of all the aforesaid reasons, a power plane is highly desired in this PCB application. However, many different reference voltages exist and only one designated power plane is available in this six layer board design. Therefore a single power plane must be shared with all reference voltages. The reference voltages traces are made as wide as possible to take advantage of the benefits mentioned. This allows the power plane to both prevent noise from outside the board from propagating internal to the board, as well as preventing noise from within the board from being distributed to other components. Figure 34 describes the power plane distribution network.

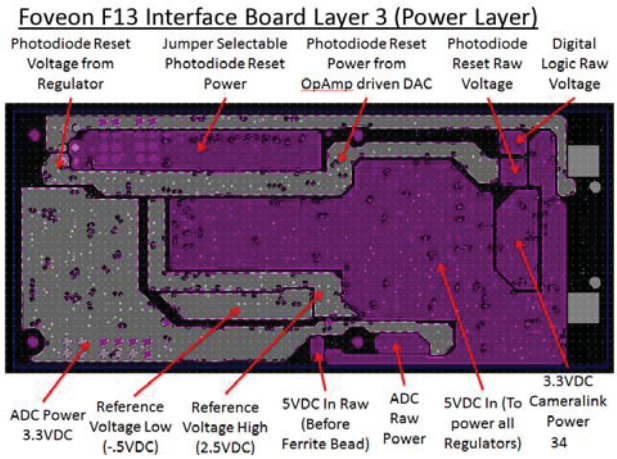

Figure 34: Power Layer

Layer 4: Ground

Much like the layer 3 power plane, the ground plane serves an important role in the PCBs design. In this case, a plane covering the entire board is chosen over several wide ground traces for multiple reasons.

First, a single ground plane is chosen to simplify circuit design. This is because a ground connection can be made right under a

component's pad location, reducing the number of traces required to connect the component to ground. This also has the added benefit of allowing additional room for other traces.

Another added benefit of a ground plane is the desire to control the path impedance of a component to ground. Like the power plane, when a component has a very short distance to travel to connect to a ground plane, the impedance in the connection is minimized. When this is performed for all components, it assures that the ground reference is as similar as possible to all components. Similarly, when the ground plane is very large, there is a very small impedance change in the ground plane from one side of the board to the other, thereby diminishing noise.

One last reason to implement a ground plane is the ability of the plane to act as a decoupling capacitor when placed next to a power plane. In this situation, the two planes act like a capacitor preventing noise from being coupled from one circuit to another through a power supply [8]. It is because of this reason that the layer 3 power plane and the layer 4 ground plane are adjacent to each other in the board stackup. The ground layer physical layout can be seen in Figure 35.

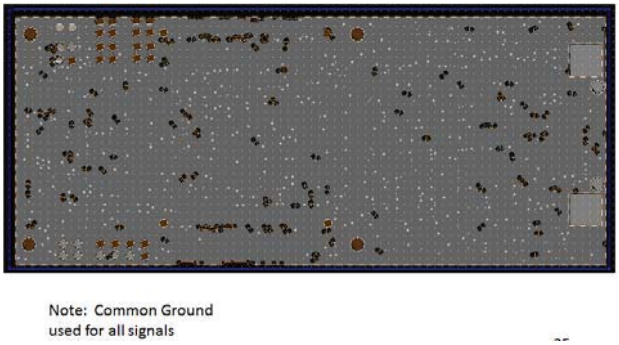

Figure 35: Ground Layer

D. Validate Printed Circuit Board

Using the design theory and implementation described above, the daughter card is fabricated and populated with all card components, including the ADC, reference voltage, CameraLink and Foveon F13 circuitry. The card is populated by hand using both a soldering iron and a hot air soldering station. Both the soldering iron and station are set to 667 degrees Fahrenheit for all soldering tasks. The populated board can be seen below in Figure 36 (Top layer) and Figure 37 (Bottom layer). Note that the circuitry identified in the populated Figures below is the same circuitry identified in the board theory of Figures 28 and 30.

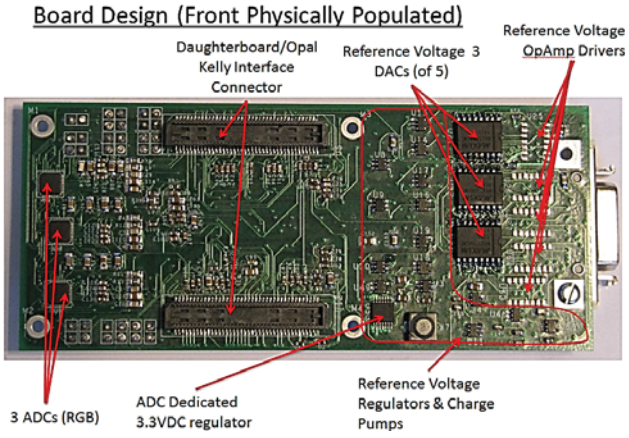

Figure 36: Populated Top Layer

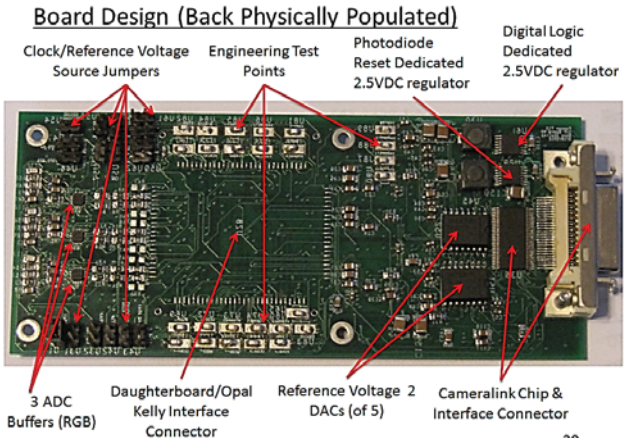

Figure 37: Populated Bottom Layer

Since this project is very complex, it proved difficult to confirm all aspects of the design within the time allocated. Consequently, the scope of this project does not entail all testing that must be conducted on this camera to validate full functionality, but rather tests all analog components of the camera as well as a several digital interfaces. The components that are tested include:

- All reference voltages are driving the correct voltage
- All ADC circuitry is functioning as expected
- CameraLink interface can correctly transmit images

This leaves several components untested and requires the Washington University Advanced Sensors Team to validate these aspects before the camera can be utilized for its desired application. Also, since programming the Foveon F13 through the firmware requires the construction of complex software and firmware to properly configure the device, the Foveon F13 is left uninstalled for all of the testing mentioned below. Lastly, at the time of testing, the OpAmps used for high-current voltage following of the DACs were not available from the component supplier. Subsequent testing of the device requires the installation of these components.

Reference Voltage Validation

The first of the analog components to be tested is the reference voltage circuitry. This consists of testing the three standalone high current drive voltage regulators as well as the twenty DAC driven voltages. To setup the DAC driven voltages, both software and firmware are written.

Software to control the DACs is written in C++ and can be found in Appendix A: C++ DAC Programming Code. This code takes the 20 voltages desired by the end user and transfers them down to the FPGA for further processing. This is accomplished by utilizing the Opal Kelly Front Panel IP provided with the development board to send information from a PC to the development board over the USB interface.

In this code, the software sends the FPGA DAC configuration data such as the voltage value to be programmed into each of the four channels of each of the five DACs. The program is coded to serially send configuration information to the FPGA in the following order:

A. DAC#5-Channel A
B. DAC#4-Channel A
C. DAC#3-Channel A
D. DAC#2-Channel A
E. DAC#1-Channel A

When an entire channel has been sent to the FPGA for processing, the firmware immediately shifts the configuration information out to the DACs. Once complete, the software is able to

send the next channel set of data to the FPGA. Steps A through E are repeated three more times for Channels B, C and D of the 5 DACs.

Firmware to accept the software commands and drive the DAC configuration logic is written in Verilog and can be found in Appendix B: Verilog Firmware DAC Programming Code. This code takes the commands of a single channel as provided by the software and shifts the information out to the DACs to be configured.

Programming is accomplished by commanding the FPGA to configure four channels of the five DACs in serial. This consists of driving the Clear, Chip Select, Load DAC, Clock and Data lines in a particular order to correctly program the DACs. Upon power up, the DAC's Clear line is driven low and the Chip Select is driven high. The chip enable line on the -.5 negative voltage regulator is also driven high at startup from its normally pulled low position (to prevent negative transients from damaging the DAC at startup). At this point, the devices are ready to be programmed. Since the DACs are configured in a daisy chain fashion, data shifted and clocked out of the FPGA propagates through DAC1, DAC2, DAC3 and DAC4 until it reaches its correct destination in DAC5. Once this has been accomplished for all five DACs, the firmware strobes the "Load DAC" line and the DAC configures its output voltage for that channel to the commanded voltage. This is completed four times until all four channels of all five DACs have been properly programed.

Once configured by both software and firmware, all reference voltages can be checked. For this test, the output pin of all regulators, charge pumps and DAC are tested to comply with their intended voltages. Table 1 describes the outcome of this testing:

Name	Voltage Value	Correct Output
5x 3.3V Reg	3.3V	Yes
5x -2.5 Charge Pumps	-2.5	Yes
5x -.5 Negative Regulators	-.5	Yes
DAC5 CH A: VPOSRST1	2.5	Yes
DAC4 CH A: VREL	-.5	Yes
DAC3 CH A: CVPUR	.9	Yes
DAC2 CH A: VIDPOS	2.5	Yes
DAC1 CH A: VCAP	2.5	Yes
DAC5 CH B: VNEGRST1	-.5	Yes
DAC4 CH B: ESDN	-.5	Yes
DAC3 CH B: VCPUB	.9	Yes
DAC2 CH B: VIDNEG	-.5	Yes
DAC1 CH B: VCEH	2.5	Yes
DAC5 CH C: VPOSRST2	2.5	Yes
DAC4 CH C: -.5VDD	-.5	Yes
DAC3 CH C: VCPUG	.9	Yes
DAC2 CH C: VREF	2.5	Yes
DAC1 CH C: VCEL	-.5	Yes
DAC5 CH D: VNEGRST2	-.5	Yes
DAC4 CH D: ESDP	2.5	Yes

DAC3 CH D: VIDHOLD	1	Yes
DAC2 CH D: VSFD_OP	2.5	Yes
DAC1 CH D: VMID	1	Yes
High Current ADC VDD	3.3	Yes
High Current Digital Logic	2.5	Yes
High Current Photodiode Reset	2.5	Yes

Table 1: Probed Voltages

It should be noted that a problem was found in the High Current Photodiode Reset line schematic during this testing. The ground pad of this regulator is incorrectly tied to the voltage output pin of the device caused by a typographical oversight on the schematic. To solve the problem and test the regulator, the trace between the ground pad and the chip is severed and the pin properly drives the requested voltage.

ADC Validation

The second of the analog components to be tested is the ADC circuitry. This assessment determines if the front end ADC circuitry and ADC themselves are functioning properly. Before testing can begin on the ADC, firmware is written to configure both the ADC buffers and the ADCs. This firmware can be found in Appendix C: Verilog Firmware ADC Programming Code.

The FPGA is configured by firmware to command the ADC buffer and ADC power down signals to a low state from their internally pulled high values. The FPGA is also configured by firmware to drive an output clock with a frequency of 10MHz. Both a clock from the FPGA and a clock from the development board PLL are tested to ensure the ADCs can be clocked by both sources through a selection jumper. Once this occurs, both the buffer and the ADC are ready to accept differential signals and convert them to their binary equivalent value.

Since the Foveon F13 was not installed at the time of testing, a signal generator takes its place as the source of an analog image signal. For this testing, a differential input signal generator was not available, so a single ended signal generator is connected to the positive input of the ADC buffer of each color. The negative input of the ADC buffer of each color is allowed to float.

To validate the output of each ADC, all 14 pins of each ADC are probed to determine if the binary output by the pin matches the expected value of the input signal. It is worth noting that only the most significant bit (MSB) of the ADC is tested for correctness in voltage and timing. The MSB is chosen to determine proper functionality due to its relative ease of validation. When the input square wave transitions from a low to a high state, the MSB transitions from a low to a high state as well.

Each of the three ADCs is tested to ensure all 14 output pins are driving an appropriate voltage for a given input signal. Figure 38 and 39 represent the Blue ADC MSB and least significant (LSB) when clocked by the 40MHz signal from the PLL. This represents the same frequency the ADCs will be clocked by the Foveon F13 when connected.

Although not shown here, the Red and Green ADCs provide the same correct output for both MSB and LSB bits. The red signal on the plots below represents the 40MHz clocking signal. The yellow signal on the plots below represents the input signal to the ADC buffer front end. The blue signal on the plots below represents the MSB (on Figure 38) and LSB (on Figure 39) value output from the ADC. The 7 sample delay can also be seen between the yellow input rising edge and the blue output rising edge on Figure 38.

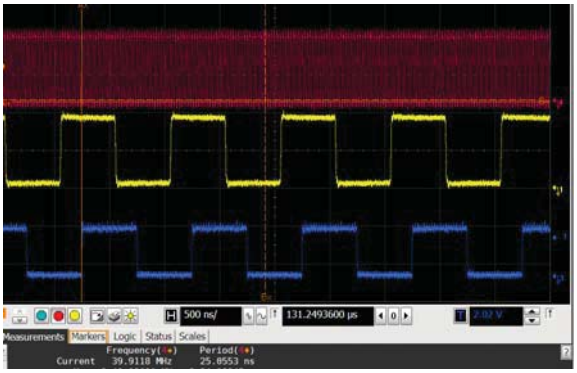

Figure 38: Blue ADC MSB Oscilloscope Plot

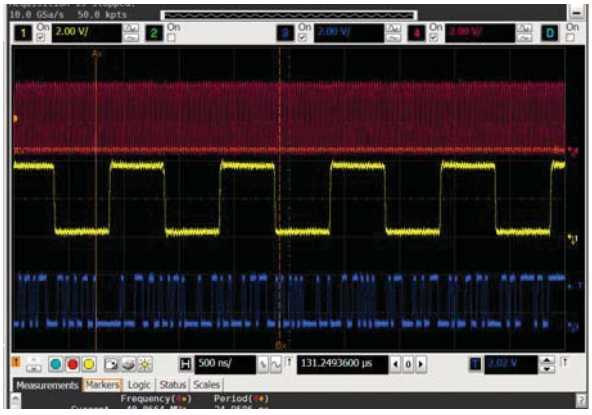

Figure 39: Blue ADC LSB Oscilloscope Plot

Lastly, to corroborate the findings of the aforementioned ADC test, a special test is conducted to validate the differential voltages as they propagate through the low pass filter. In this scenario, a 10MHz sinusoidal input is driven into the positive terminal of the ADC buffer. The negative terminal is left floating. This test shows that the circuitry is behaving as a low pass filter, as well as shows any attenuation that the signal may encounter as it travels through the ADC front end.

Like the other ADC test, the input is only driven by half of a differential driver, therefore a smaller output voltage than when differentially driven is expected. Under complete differential driving circumstances, the circuit is expected to show a gain of one throughout the entire front end circuitry. The validation of this behavior for the Blue ADC can be seen in Figure 40. The yellow signal is the input sinusoidal signal. The red and blue signals represent the negative and positive inputs to the ADC respectively.

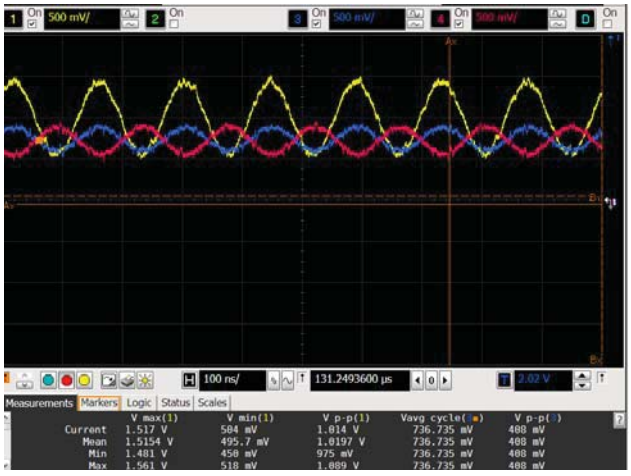

Figure 40: Blue ADC Differential Oscilloscope Plot

CameraLink Validation

The last component of the system to be tested is the CameraLink component. The

CameraLink test demonstrates that many of the digital pieces of the camera are functioning properly. These include the FPGA, the CameraLink driver, the CameraLink cable pin-out and the framegrabber software installed on the destination PC.

The Foveon F13 is not installed for this portion of the testing so an adequate substitute must be used to take its place. To accomplish this task, firmware is written which simulates a frame of data coming from the ADCs. This firmware can be found in Appendix D: Verilog CameraLink Interface Code.

The firmware is programed to perform several different tasks which must occur in correct sequence to output a frame of data. First, the firmware creates a synthetic frame of data which takes the form of a gradient image. This means that a frame of data is created where the pixel greyscale cycles from white all the way to black in small greyscale steps. This synthetic image consists of 1024x1024 pixels. Therefore with an 8 bit greyscale definition, the gradient picture cycles from white to black 4 times.

Now that a synthetic image has been created in firmware, the FPGA must package the frame into something the CameraLink chip can understand. In this portion of the firmware, the FPGA formats the synthetic image into 28 parallel output bits where 24 of the bits represent 3, 8-bit pixels, and 4 status bits. The four status bits pertain to frame and line end indicators. When a full 28 bit payload is assembled in the firmware, the data is asserted on the 28 pins shared between the FPGA and the CameraLink chip. Once the data is sent, the CameraLink clock line is toggled so the CameraLink chip can capture in values on the 28 input pins.

At the CameraLink chip, the data is repackaged into four serial streams and sent out the CameraLink connector to the framegrabber on the destination PC. The framegrabber then reconstructs the differential payload into its original 28 bit configuration. This process is repeated indefinitely until all 1024x1024 pixels are sent. The frame status bit of the last CameraLink transmission has an end-of-frame designator associated with it. The framegrabber acknowledges this end-of-frame marker and plots the received image on the PC. The successful transmission of the synthetic gradient image can be seen in Figure 41.

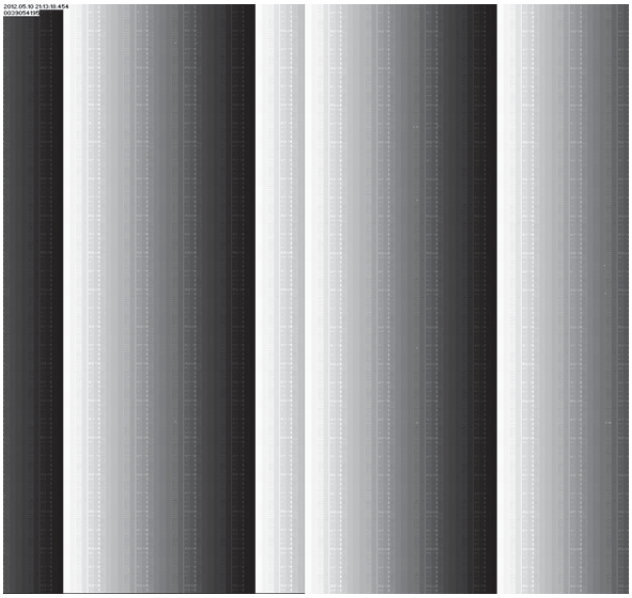

Figure 41: CameraLink Gradient Output

During CameraLink testing, it was found that the positive CameraLink connector output for differential driver 2 was in fact not driven by driver channel 2 but rather by differential driver 0. The negative terminal of differential driver 2 is correctly joined to the connector. As a result, the data sent by the CameraLink driver to the framegrabber does not properly represent the

gradient fabricated in the firmware. It is possible that both pixel data, as well as end-of-frame data, could be corrupted by this incorrectly assigned driver. This causes data to deviate from the ideal synthetic gradient as well as incorrect interpretation of end-of-frame or end-of-line markers. This leads to the incorrect asymmetrical image captured in Figure 41. Assigning the positive differential 2 driver output to the connector output fixes the identified problems.

Although a small problem was found within the PCB, the validation of the signal path of the CameraLink interface was adequately demonstrated. It is shown that the FPGA can correctly fabricate a synthetic image, this image can be packaged into a CameraLink compatible input, the CameraLink chip can (almost) send a correct differential image signal to the framegrabber, and the framegrabber can disassemble the differential input and fabricate an image.

Conclusion

In summary, it was shown that the foundation of a vertically stacked photodiode imager camera was completed. This foundation consists of the design, fabrication, and test of a vertically stacked photodiode image sensor upon a printed circuit board comprising of all necessary support electronics and interfaces. Additionally both firmware and software components were written to properly interface the daughterboard PCB with both a personal computer and an FPGA.

Additional progress will be required to achieve a fully functional vertically stacked photodiode imager. These tasks include completing the integration of the Foveon F13 with the FPGA and converting the dedicated USB interface to the RS422 interface embedded within the CameraLink protocol.

The Foveon F13, and subsequently the firmware, will need to be fabricated to allow for configuration information to be passed between the FPGA and the Foveon F13. This includes features such as pixel grouping, reset reference voltages, clock delay time and video vs. single frame configuration information. Additionally, although the ADC circuitry has been validated for proper functionality, it will need to be rechecked once the Foveon F13 provides the differential input rather than the signal generator used for this test.

Lastly, the RS422 interface will need to be substituted for the USB interface for all Foveon F13, ADC and DAC control. This will allow the USB interface to go unused and have all control and image data transferred through the single CameraLink interface cable.

Although a small amount of work remains to be completed on this camera, the Washington University Advanced Sensors team now has a strong platform to continue developing next-generation imaging sensors due to the rigorous engineering discipline involved with this project. With the large strides taken in this imager's development, this project can make a significant contribution to the advancement of the medical, biological and astronomy fields.

WORKS CITED

[1] Rick Perley, High Fidelity Imaging. NRAO-Socorro: Twelfth Synthesis Imaging Workshop 2010 June 8-15 Available: http://www.aoc.nrao.edu/events/synthesis/2010/lectures/SISS2010-HiDy.pdf

[2] Foveon X3 F13 14.4 Megapixel CMOS Direct Image Sensor Preliminary Sensor Datasheet Rev G, Foveon Inc. April 2008

[3] Foveon Inc, 2010 , Foveon X3® 14.1 Megapixel DSLR/Pro Image Sensor Available: http://www.foveon.com/article.php?a=222

[4] Specifications of the Camera Link Interface Standard for Digital Cameras and Frame Grabbers, Version 1.1 Automated Imaging Association, Jan 2004

[5] Interface Standard for Nominal 3 V/3.3 V Supply Digital Integrated Circuits, JEDEC Standard 8-C, June 2006

[6] Standard for Description of 2.5 V CMOS Logic Devices with 3.6 V CMOS Tolerant Inputs and Outputs, JEDEC Standard 64-A, October 2000

[7] Douglas Brooks, “Splitting Planes For Speed and Power”, Printed Circuit Design, a CMP Media publication, December, 2000

[8] Leroy Davis, Capacitor Data IC By-Pass Design Information, 26 February 2012, Available: http://www.interfacebus.com/Design_Capacitors.html

NOTE: All source code in the following appendices can be found on the Washington University Advanced Sensors server.

Appendix A: C++ DAC Programming Code

```
// DAC code

#include <iostream>
#include <cstdlib>

#include "okFrontPanelDLL.h"

using namespace std;
const int width = 452;
const int height = 562;

// DAC pins
int A  = 0x0;   //(DAC/A)
int B = 0x1;    //(DAC/B)
int C = 0x2;    //(DAC/C)
int D = 0x3;    //(DAC/D)


void pause() {
    cout << endl << "Press Enter to continue..." << endl;
    cin.get();
}
int convertVoltageToInt(float Voltage)
{   int Result;
    float new_Voltage;
    float VBIAS = -2.5;
    float VREF = 5.8;
    Result = (Voltage - VBIAS) * 256 / VREF;
    return (Result);
}

void programDac( int inputAddress, double inputVoltage1, double inputVoltage2, double inputVoltage3, double inputVoltage4, double inputVoltage5, okCUsbFrontPanel *xem)
{   int Status;
    int VoltageToInt1 = convertVoltageToInt(float(inputVoltage1));
    int VoltageToInt2 = convertVoltageToInt(float(inputVoltage2));
    int VoltageToInt3 = convertVoltageToInt(float(inputVoltage3));
    int VoltageToInt4 = convertVoltageToInt(float(inputVoltage4));
    int VoltageToInt5 = convertVoltageToInt(float(inputVoltage5));

    int SetValue1 =  (1 << 8) + (inputAddress << 10) + VoltageToInt1;
    int SetValue2 =  (1 << 8) + (inputAddress << 10) + VoltageToInt2;
    int SetValue3 =  (1 << 8) + (inputAddress << 10) + VoltageToInt3;
    int SetValue4 =  (1 << 8) + (inputAddress << 10) + VoltageToInt4;
    int SetValue5 =  (1 << 8) + (inputAddress << 10) + VoltageToInt5;

    xem->UpdateWireOuts();


    cout<< "\n   Input voltage = " << inputVoltage1 << "\n";
    cout<< "   Input address = " << inputAddress <<"\n";
    cout<< "   Voltage converted to int = " << VoltageToInt1 << "\n";
    cout<< "   d_count = " << xem->GetWireOutValue(0x22) << "\n";
    cout<< "   Value to be set = " << SetValue1 << "\n";

    do {xem->UpdateWireOuts();
    Status = xem->GetWireOutValue(0x20);
    } while (!(Status & 0x01));  // Wait for idle to go high

    xem->SetWireInValue(0x01,SetValue1); // 12-bit address/config/data input to DAC state machine
    xem->SetWireInValue(0x02,SetValue2); // 12-bit address/config/data input to DAC state machine
    xem->SetWireInValue(0x03,SetValue3); // 12-bit address/config/data input to DAC state machine
    xem->SetWireInValue(0x04,SetValue4); // 12-bit address/config/data input to DAC state machine
    xem->SetWireInValue(0x05,SetValue5); // 12-bit address/config/data input to DAC state machine
    xem->UpdateWireIns();

    xem->ActivateTriggerIn(0x40, 0); // send program signal
    xem->UpdateWireOuts();
    cout<< "   Trigger Idle =  " << xem->GetWireOutValue(0x20) << "\n";

    // see outputs after DAC was programmed
    do {xem->UpdateWireOuts();
    Status = xem->GetWireOutValue(0x20);
    cout<< "   Idle =  " << xem->GetWireOutValue(0x20) << "\n";
    cout<< "   add_d_in = " << xem->GetWireOutValue(0x21) << "\n";
    cout<< "   d_count = " << xem->GetWireOutValue(0x22) << "\n";

    } while (!(Status & 0x01));

        xem->UpdateWireOuts();
}

int main(int argc, char* argv[]) {
    //set up opal kelly library
    int x;

    if (!okFrontPanelDLL_LoadLib(NULL)) {
        cout << "ERROR: FrontPanel DLL could not be initialized." << endl;
        return(-1);
    }
    okCUsbFrontPanel *xem;
    xem = new okCUsbFrontPanel;

    if (okCUsbFrontPanel::NoError != xem->OpenBySerial()) {
        cout << "ERROR: Could not open FPGA." << endl;
```

```
        return(-1);
    }

    cout << "Device firmware version: " << xem->GetDeviceMajorVersion() << "." << xem->GetDeviceMinorVersion() << endl;
    cout << "Device serial number: " << xem->GetSerialNumber() << endl;
    cout << "Device device ID: " << xem->GetDeviceID() << endl;

    if (okCUsbFrontPanel::NoError != xem->ConfigureFPGA(string("./DAC/dac1.bit"))) {
        cout << "FPGA configuration failed.\n" << endl;
        return -1;
    } else {
        cout << "Configuration done." << endl;
    }

    if (xem->IsFrontPanelEnabled()) {
        cout << "FrontPanel support is enabled." << endl;
    } else {
        cout << "ERROR: FrontPanel support is not enabled." << endl;
        return -1;
    }

    //Call programDac
    x = 250;

    //Program All A Channels on DACS
    programDac(A, 2.5, -.5, .9, 2.5, 2.5, xem);
    // Set VPOSRST1 to 2.5 Volts (A) DAC5
    // Set VREL to -.5 Volts (A) DAC4
    // Set VCPUR to .9 Volts (A) DAC3
    // Set VIDPOS to 2.5 Volts (A) DAC2
    // Set VCAP to 2.5 Volts (A) DAC1

    programDac(B, -.5, -.5, .9, -.5, 2.5,  xem);
    // Set VNEGRST1 to -.5 Volts (B) DAC5
    // Set ESDN to -.5 Volts (B) DAC4
    // Set VCPUB to .9 Volts (B) DAC3
    // Set VIDNEG to -.5 Volts (B) DAC2
    // Set VCEH to 2.5 Volts (B) DAC1

    programDac(C, 2.5, -.5, .9, 2.5, -.5,  xem);
    // Set VNEGRST1 to -.5 Volts (B) DAC5
    // Set VPOSRST2 to 2.5 Volts (C) DAC5
    // Set -.5_AVDD to -0.5 Volts (C) DAC4
    // Set VCPUG to .9 Volts (C) DAC3
    // Set VREF_2.5 to 2.5 Volts (C) DAC2
    // Set VCEL to -0.5 Volts (C) DAC1



    programDac(D, -.5, 2.5, 1, 2.5, 1,  xem);
    // Set VNEGRST1 to -.5 Volts (B) DAC5
    // Set VNEGRST2 to -.5 Volts (D) DAC5
    // Set ESDP to 2.5 Volts (D) DAC4
    // Set VIDHOLD to 1 Volts (D) DAC3
    // Set VSFD_OP to 2.5 Volts (D) DAC2
    // Set VMID to 1 Volts (D) DAC1

   pause();
 return(0);
}
```

Appendix B: Verilog Firmware DAC Programming Code

File: DAC.v

```
`timescale 1ns / 1ps

module DAC1
 ( input  wire [7:0]  hi_in,
   output wire [1:0]  hi_out,
   inout  wire [15:0] hi_inout,

   output wire        i2c_sda,
   output wire        i2c_scl,
   output wire        hi_muxsel,

        input  wire        clk1,
        //input  wire                   button,                       // provide reset
   output wire [3:0]  led,
        output wire                   ldac_not1,
        output wire             cs_not1,
        output wire                   s_out1,
        output wire                   sclk_dac_1,
        output wire                   clr_not1,
        output wire             neg_vreg_en

   );

//-------------Internal Constants---------------
parameter SIZE=2;
parameter S0=2'b00, S1=2'b01, S2=2'b10;

//-------------OK Wires---------------
wire[15:0]  ep01wire;           // 16-bit data in (used for address and data) (din)
wire[15:0]  ep02wire;           // program
wire[15:0]  ep03wire;           // program
wire[15:0]  ep04wire;           // program
wire[15:0]  ep05wire;           // program

wire [15:0]  ep20wire;          // idle
wire [15:0]  ep21wire;          // din
wire [15:0]  ep22wire;          // d_count


//-------------OK Trigger---------------
wire[15:0] ep40trig; //active high, programming mode (program)


//-------------Target interface bus-------------
wire        ti_clk;
wire [30:0] ok1;
wire [16:0] ok2;
```

```
//-------------Internal Variables---------------

wire program;
wire [59:0] din;
reg s_out, cs_not, ldac_not, clr_not;
wire clk_out;
reg idle = 1'b0;                                //signal sent from state machine in state 0
reg  [SIZE-1:0] state;           //Seq part of FSM
wire [SIZE-1:0] next_state; //combo part of FSM
reg [5:0] d_count;
reg [59:0] add_d_in;
reg clkdiv = 1'b0;
reg [2:0] count = 3'b011;
wire reset;
reg [3:0] ledDebug;
//-------------Assignment statements---------------
assign i2c_sda = 1'bz;
assign i2c_scl = 1'bz;
assign hi_muxsel = 1'b0;
assign neg_vreg_en = 1'b1;  //<<4-28-12 DPS>> Added to drive the Negative Voltage regulator Enable bit High.

assign led = ~ledDebug;
//assign reset = ~button; //when pressed, button = 0. So reset = 1 when button is pressed
assign clk_out = clkdiv; //confirm if correct
assign program = ep40trig;
//assign program = ep02wire[0];
assign din = {ep01wire[11:0],ep02wire[11:0],ep03wire[11:0],ep04wire[11:0],ep05wire[11:0]};
//assign din = ep01wire[11:0];
assign s_out1= s_out;
assign cs_not1= cs_not;
assign ldac_not1 = ldac_not;
assign clr_not1 = clr_not;
assign sclk_dac_1= clk_out; //need an output clock as well!
assign ep20wire[0] = idle;
assign ep21wire[11:0] = add_d_in;
assign ep22wire = d_count;


//-------------Clock divider to generate 10 MHz clock from clk1 (set at 48MHz)---------------
always @(posedge clk1) begin
   count <= count - 1;
   if (count == 3'b000) begin
    count <= 3'b011;
         clkdiv <= ~clkdiv;
        end
end


//-------------DAC State Machine---------------

assign next_state=fsm_function(state, program, d_count);

//-------------Function for Combinatorial Logic---------
function [SIZE-1:0] fsm_function;
 input [SIZE-1:0]  state;
 input program;
 input [5:0] d_count;

 case(state)
        S0:if (program == 1'b1) begin
                        fsm_function=S1;
                end else begin
                        fsm_function=S0;
        end
        S1:if(d_count == 4'b000000) begin
                        fsm_function=S2;
                end else begin
                        fsm_function=S1;
                end
        S2: fsm_function=S0;

        default: fsm_function=S0;
 endcase
endfunction

//-------------Sequential Logic-------------------------
always @ (negedge clkdiv)
begin : FSM_SEQ
 if (reset == 1'b1) begin
        state <= S0;
 end else begin
        state <= next_state;
 end
end

//------------Output Logic-----------------------
always @ (negedge clkdiv)
begin : OUTPUT_LOGIC
 if (reset == 1'b1) begin
        cs_not <= 1'b1;
        add_d_in <= 60'd0;
        s_out <= add_d_in[59];
end
else begin
```

```
 case(state)
        S0: begin
          idle <= 1'b1;
          d_count <= 6'b111011;
          cs_not <= 1'b1;
          add_d_in <= din;
          s_out <= add_d_in[59];
          ldac_not <= 1'b1;
          clr_not <= 1'b1;
          ledDebug <= 4'b0001;
        end
        S1: begin
          idle <= 1'b0;
          cs_not <= 1'b0;
          d_count <= d_count-1;
          s_out <= add_d_in[59];
          add_d_in <= add_d_in << 1;
          ldac_not <= 1'b1;
          clr_not <= 1'b1;
          ledDebug <= 4'b0010;
        end
        S2: begin
          idle <= 1'b0;
          cs_not <= 1'b1;
          s_out <= 1'b0;
          ldac_not <= 1'b0;
          clr_not <= 1'b1;
         end
        default: begin
          d_count <= 6'b111011;
          cs_not <= 1'b1;
          s_out <= add_d_in[59];
          ldac_not <= 1'b1;
          clr_not <= 1'b1;
          idle <= 1'b0;
        end
 endcase
end
end

//-------Instantiate the okHostInterface and connect endpoints to the target interface.---------
wire [17*3-1:0]  ok2x;
okHost okHI(
        .hi_in(hi_in), .hi_out(hi_out), .hi_inout(hi_inout), .ti_clk(ti_clk),
        .ok1(ok1), .ok2(ok2));

okWireOR # (.N(3)) wireOR (ok2, ok2x);


okWireIn     wi00(.ok1(ok1),                   .ep_addr(8'h00), .ep_dataout(ep00wire));
okWireIn     wi01(.ok1(ok1),                   .ep_addr(8'h01), .ep_dataout(ep01wire));
okWireIn     wi02(.ok1(ok1),                   .ep_addr(8'h02), .ep_dataout(ep02wire));
okWireIn     wi03(.ok1(ok1),                   .ep_addr(8'h03), .ep_dataout(ep03wire));
okWireIn     wi04(.ok1(ok1),                   .ep_addr(8'h04), .ep_dataout(ep04wire));
okWireIn     wi05(.ok1(ok1),                   .ep_addr(8'h05), .ep_dataout(ep05wire));
okWireOut    wo20(.ok1(ok1), .ok2(ok2x[ 0*17 +: 17 ]), .ep_addr(8'h20), .ep_datain(ep20wire));
okWireOut    wo21(.ok1(ok1), .ok2(ok2x[ 1*17 +: 17 ]), .ep_addr(8'h21), .ep_datain(ep21wire));
okWireOut    wo22(.ok1(ok1), .ok2(ok2x[ 2*17 +: 17 ]), .ep_addr(8'h22), .ep_datain(ep22wire));
okTriggerIn  trigIn40 (.ok1(ok1),                                .ep_addr(8'h40), .ep_clk(clkdiv), .ep_trigger(ep40trig));
//okWireOut    wo21(.ok1(ok1), .ok2(ok2x[ 1*17 +: 17 ]), .ep_addr(8'h21), .ep_datain(ep21wire));



endmodule
```

Appendix C: Verilog Firmware ADC Programming Code

File: ADC2.v

```
`timescale 1ns / 1ps

module DAC1
 ( input  wire [7:0]  hi_in,
   output wire [1:0]  hi_out,
   inout  wire [15:0] hi_inout,

   output wire        i2c_sda,
   output wire        i2c_scl,
   output wire        hi_muxsel,

        input  wire        clk1,
        //input  wire                   button,                         // provide reset
   output wire [3:0]  led,
        output wire                   ldac_not1,
        output wire             cs_not1,
        output wire                   s_out1,
        output wire                   sclk_dac_1,
        output wire                   clr_not1,
        output wire             neg_vreg_en,
        input wire D1_ADC_R,
        input wire D13_ADC_R,
        input wire D1_ADC_G,
        input wire D1_ADC_B,
        input wire OTR_R,
        input wire OTR_G,
        input wire OTR_B,
        output wire ADC_PDWN,
        output wire FPGA_CLOCK
   );

//-------------Internal Constants---------------
parameter SIZE=2;
parameter S0=2'b00, S1=2'b01, S2=2'b10;

//-------------OK Wires---------------
wire[15:0]  ep01wire;           // 16-bit data in (used for address and data) (din)
wire[15:0]  ep02wire;           // program
wire[15:0]  ep03wire;           // program
wire[15:0]  ep04wire;           // program
wire[15:0]  ep05wire;           // program

wire [15:0]  ep20wire;          // idle
wire [15:0]  ep21wire;          // din
wire [15:0]  ep22wire;          // d_count


//-------------OK Trigger---------------
wire[15:0] ep40trig; //active high, programming mode (program)


//-------------Target interface bus-------------
wire        ti_clk;
wire [30:0] ok1;
wire [16:0] ok2;


//-------------Internal Variables---------------

wire program;

wire D1_ADC_R_in;
wire D13_ADC_R_in;
wire OTR_R_in;
wire ADC_PDWN_out;

wire [59:0] din;
reg s_out, cs_not, ldac_not, clr_not;
wire clk_out;
wire temp_clock;
reg idle = 1'b0;                                //signal sent from state machine in state 0
reg  [SIZE-1:0] state;          //Seq part of FSM
wire [SIZE-1:0] next_state; //combo part of FSM
reg [5:0] d_count;
reg [59:0] add_d_in;
reg clkdiv = 1'b0;
reg ADC_clock = 1'b0;
reg [2:0] count = 3'b011;
wire reset;
reg [3:0] ledDebug;
//-------------Assignment statements---------------
assign i2c_sda = 1'bz;
assign i2c_scl = 1'bz;
assign hi_muxsel = 1'b0;
assign neg_vreg_en = 1'b1;  //<<4-28-12 DPS>> Added to drive the Negative Voltage regulator Enable bit High.
assign ADC_PDWN = 1'b0; //<<5-7-12 DPS>> Added to drive the ADC power down bit low.
assign led = ~ledDebug;
//assign reset = ~button; //when pressed, button = 0. So reset = 1 when button is pressed
assign clk_out = clkdiv; //confirm if correct
assign temp_clock = clkdiv;
assign program = ep40trig;
//assign program = ep02wire[0];
assign din = {ep01wire[11:0],ep02wire[11:0],ep03wire[11:0],ep04wire[11:0],ep05wire[11:0]};
//assign din = ep01wire[11:0];
```

```
assign s_out1= s_out;
assign cs_not1= cs_not;
assign ldac_not1 = ldac_not;
assign clr_not1 = clr_not;
assign sclk_dac_1= clk_out; //need an output clock as well!
assign FPGA_CLOCK = temp_clock;
//assign FPGA_CLOCK = temp_clock;
assign ep20wire[0] = idle;
assign ep21wire[11:0] = add_d_in;
assign ep22wire = d_count;

assign D1_ADC_R_in = D1_ADC_R;
assign D13_ADC_R_in = D13_ADC_R;
assign OTR_R_in = OTR_R;
//-------------Clock divider to generate 10 MHz clock from clk1 (set at 48MHz)---------------
always @(posedge clk1) begin
   count <= count - 1;
   if (count == 3'b000) begin
    count <= 3'b011;
         clkdiv <= ~clkdiv;
        end
end

//-------------OTR FLAG VERIFICATION---------------
always @ * begin
   if (OTR_R_in == 1'b1) begin
                ledDebug[0] = 1'b1;
        end else begin
                ledDebug[0] = 1'b0;
        end
end


//-------------LSB VERIFICATION---------------
always @ * begin
   if (D1_ADC_R_in == 1'b1) begin
                ledDebug[1] = 1'b1;
        end else begin
                ledDebug[1] = 1'b0;
        end
end

//-------------MSB FLAG VERIFICATION---------------
always @ * begin
   if (D13_ADC_R_in == 1'b1) begin
                ledDebug[2] = 1'b1;
        end else begin


                 ledDebug[2] = 1'b0;
         end
end
//-------------DAC State Machine---------------

assign next_state=fsm_function(state, program, d_count);

//-------------Function for Combinatorial Logic---------
function [SIZE-1:0] fsm_function;
 input [SIZE-1:0]  state;
 input program;
 input [5:0] d_count;

 case(state)
        S0:if (program == 1'b1) begin
                        fsm_function=S1;
                end else begin
                        fsm_function=S0;
        end
        S1:if(d_count == 4'b000000) begin
                        fsm_function=S2;
                end else begin
                        fsm_function=S1;
                end
        S2: fsm_function=S0;

        default: fsm_function=S0;
 endcase
endfunction

//-------------Sequential Logic--------------------------
always @ (negedge clkdiv)
begin : FSM_SEQ
 if (reset == 1'b1) begin
        state <= S0;
 end else begin
        state <= next_state;
 end
end

//------------Output Logic-----------------------
always @ (negedge clkdiv)
begin : OUTPUT_LOGIC
 if (reset == 1'b1) begin
        cs_not <= 1'b1;
        add_d_in <= 60'd0;
        s_out <= add_d_in[59];
```

```
end
else begin
 case(state)
        S0: begin
          idle <= 1'b1;
          d_count <= 6'b111011;
          cs_not <= 1'b1;
          add_d_in <= din;
          s_out <= add_d_in[59];
          ldac_not <= 1'b1;
          clr_not <= 1'b1;
          //ledDebug <= 4'b0001;
        end
        S1: begin
          idle <= 1'b0;
          cs_not <= 1'b0;
          d_count <= d_count-1;
          s_out <= add_d_in[59];
          add_d_in <= add_d_in << 1;
          ldac_not <= 1'b1;
          clr_not <= 1'b1;
          //ledDebug <= 4'b0010;
        end
        S2: begin
          idle <= 1'b0;
          cs_not <= 1'b1;
          s_out <= 1'b0;
          ldac_not <= 1'b0;
          clr_not <= 1'b1;
         end
        default: begin
          d_count <= 6'b111011;
          cs_not <= 1'b1;
          s_out <= add_d_in[59];
          ldac_not <= 1'b1;
          clr_not <= 1'b1;
          idle <= 1'b0;
        end
 endcase
end
end

//-------Instantiate the okHostInterface and connect endpoints to the target interface.---------
wire [17*3-1:0]  ok2x;
okHost okHI(
        .hi_in(hi_in), .hi_out(hi_out), .hi_inout(hi_inout), .ti_clk(ti_clk),
        .ok1(ok1), .ok2(ok2));


okWireOR # (.N(3)) wireOR (ok2, ok2x);
okWireIn     wi00(.ok1(ok1),                  .ep_addr(8'h00), .ep_dataout(ep00wire));
okWireIn     wi01(.ok1(ok1),                  .ep_addr(8'h01), .ep_dataout(ep01wire));
okWireIn     wi02(.ok1(ok1),                  .ep_addr(8'h02), .ep_dataout(ep02wire));
okWireIn     wi03(.ok1(ok1),                  .ep_addr(8'h03), .ep_dataout(ep03wire));
okWireIn     wi04(.ok1(ok1),                  .ep_addr(8'h04), .ep_dataout(ep04wire));
okWireIn     wi05(.ok1(ok1),                  .ep_addr(8'h05), .ep_dataout(ep05wire));
okWireOut    wo20(.ok1(ok1), .ok2(ok2x[ 0*17 +: 17 ]), .ep_addr(8'h20), .ep_datain(ep20wire));
okWireOut    wo21(.ok1(ok1), .ok2(ok2x[ 1*17 +: 17 ]), .ep_addr(8'h21), .ep_datain(ep21wire));
okWireOut    wo22(.ok1(ok1), .ok2(ok2x[ 2*17 +: 17 ]), .ep_addr(8'h22), .ep_datain(ep22wire));
okTriggerIn  trigIn40 (.ok1(ok1),                                .ep_addr(8'h40), .ep_clk(clkdiv), .ep_trigger(ep40trig));
//okWireOut    wo21(.ok1(ok1), .ok2(ok2x[ 1*17 +: 17 ]), .ep_addr(8'h21), .ep_datain(ep21wire));



endmodule
```

Appendix D: Verilog CameraLink Interface Code

File: Camera_Link_Top.v

```
`timescale 1ns / 1ps

module Camera_Link_Top(
        //Clock input from PLL
        input wire clk, //max 85 MHz

        //CL outputs
        output wire clo_clk,            //named Camera Link Clock
        output wire clo_fval,           //CL_TX25
        output wire clo_lval,           //CL_TX24
        output wire clo_dval,           //CL_TX26
        output wire [7:0] clo_a,        //Starting at bit 7:  CL_TX5, CL_TX27, CL_TX6, CL_TX4, CL_TX3, CL_TX2, CL_TX1, CL_TX0
        output wire [7:0] clo_b,        //Starting at bit 7:  CL_TX11, CL_TX10, CL_TX14, CL_TX13, CL_TX12, CL_TX9, CL_TX8, CL_TX7
        output wire [7:0] clo_c,        //Starting at bit 7:  CL_TX17, CL_TX16, CL_TX22, CL_TX21, CL_TX20, CL_TX19, CL_TX18, CL_TX15
        output wire clo_pwrdn_n
        );


//-------------------------------------------------------------------------
// Internal Variables
//-------------------------------------------------------------------------

//Pins to Top
wire i_clock;
wire i_reset;

//Top to Pins
//wire i_clo_fval;
//wire i_clo_lval;
//wire i_clo_dval;
//wire [7:0] i_clo_a;
//wire [7:0] i_clo_b;
//wire [7:0] i_clo_c;
//wire i_clo_clk;

//Top to CameraLink
wire i_dataValidIn;
wire i_fEndIn;
wire i_lEndIn;
wire [0:+24] i_dataIn;
wire i_readyOut;

//Gradient Generator to Top
wire i_dataValidOut;
wire i_fEndOut;
wire i_lEndOut;
wire [0:+24] i_dataOut;
```

```
wire i_enable;

//------------------------------------------------------------------------
// Assignment Statements
//------------------------------------------------------------------------
assign i_clock = clk;
assign i_dataValidIn = i_dataValidOut;
assign i_fEndIn = i_fEndOut;
assign i_lEndIn = i_lEndOut;
assign i_dataIn = i_dataOut;
assign clo_pwrdn_n = 1'b1;
assign i_reset = 1'b0;
assign i_enable = 1'b1;

//------------------------------------------------------------------------
// Instantiate CameraLink Out Module
//------------------------------------------------------------------------
CameraLinkOut #(
        .counterWidth(12),
        .betweenFramesDelay(2640),
        .betweenLinesDelay(190)
) CameraLinkOut (
        .clk(i_clock),
        .rst(i_reset),

        .dataValidIn(i_dataValidIn),
        .fEndIn(i_fEndIn),
        .lEndIn(i_lEndIn),
        .dataIn(i_dataIn), //bus [23:0]
        .readyOut(i_readyOut),

        .clo_clk(clo_clk),
        .clo_fval(clo_fval),
        .clo_lval(clo_lval),
        .clo_dval(clo_dval),
        .clo_a(clo_a), //bus [7:0]
        .clo_b(clo_b), //bus [7:0]
        .clo_c(clo_c) //bus [7:0]
);
//------------------------------------------------------------------------

//------------------------------------------------------------------------
// Instantiate Gradient Generator Module
//------------------------------------------------------------------------
GradientGenerator #(
        .rows(1024),
        .columns(1024),

        .rowCounterWidth(10),
        .colCounterWidth(10)
) GradientGenerator (
        .clk(i_clock),
        .rst(i_reset),

        .enable(i_enable),   //connects to readyOut of next block
        .dataValidOut(i_dataValidOut),
        .fEndOut(i_fEndOut),
        .lEndOut(i_lEndOut),
        .dataOut(i_dataOut) //bus [23:0]
);
//------------------------------------------------------------------------

endmodule
```

File: GradientGenerator.v

```
`timescale 1ns / 1ps

module GradientGenerator #(
        parameter rows = 1024,
        parameter columns = 1024,
        parameter rowCounterWidth = 10,
        parameter colCounterWidth = 10
)(
        input wire rst,
        input wire clk,

        input wire enable,
        output wire dataValidOut,
        output wire fEndOut,
        output wire lEndOut,
        output wire [23:0] dataOut
);
/* Instantiation Template
GradientGenerator #(
        .rows(1024),
        .columns(1024),
        .rowCounterWidth(10),
        .colCounterWidth(10)
) instanceName (
        .clk(),
        .rst(),

        .enable()   //connects to readyOut of next block
        .dataValidOut(),
        .fEndOut(),
        .lEndOut(),
        .dataOut() //bus [23:0]
);
*/

reg [rowCounterWidth-1:0] rowCount,rowCountD;
reg [colCounterWidth-1:0] colCount,colCountD;

//assign values to output wires
assign dataValidOut = 1;
assign lEndOut = (colCount == columns-1);
assign fEndOut = (colCount == columns-1) & (rowCount == rows-1);
assign dataOut[7:0] = colCount[7:0]; //A
assign dataOut[15:8] = rowCount[7:0]; //B
assign dataOut[23:16] = colCount[7:0]; //C

always @ * begin

  //default values:
       rowCountD = rowCount;
       colCountD = colCount;
       //count up when enabled:
       if(enable) begin
               colCountD = colCount + 1;
               if(colCount == columns-1) begin
                       colCountD = 0;
                       rowCountD = rowCount+1;
                       if(rowCount == rows-1) begin
                               rowCountD = 0;
                       end
               end
       end
end

always @(posedge clk) begin
       if(rst) begin
               rowCount <= 0;
               colCount <= 0;
       end else begin
               //latch DFFs on clock
               rowCount <= rowCountD;
               colCount <= colCountD;
       end
end
endmodule
```

File: CameraLinkOut.v

```
`timescale 1ns / 1ps

module CameraLinkOut #(
        parameter counterWidth = 12,
        parameter betweenFramesDelay = 2640, //66 us @ 40 MHz
        parameter betweenLinesDelay = 190 //4.75 us @ 40 MHz
)(
        input wire rst,
        input wire clk, //max 85 MHz

        input wire dataValidIn,
        input wire fEndIn,
        input wire lEndIn,
        input wire [23:0] dataIn,
        output wire readyOut,

        //CL outputs
        output wire clo_clk,            //named Camera Link Clock
        output wire clo_fval,           //CL_TX25
        output wire clo_lval,           //CL_TX24
        output wire clo_dval,           //CL_TX26
        output wire [7:0] clo_a,        //Starting at bit 7:  CL_TX5, CL_TX27, CL_TX6, CL_TX4, CL_TX3, CL_TX2, CL_TX1, CL_TX0
        output wire [7:0] clo_b,        //Starting at bit 7:  CL_TX11, CL_TX10, CL_TX14, CL_TX13, CL_TX12, CL_TX9, CL_TX8, CL_TX7
        output wire [7:0] clo_c         //Starting at bit 7:  CL_TX17, CL_TX16, CL_TX22, CL_TX21, CL_TX20, CL_TX19, CL_TX18, CL_TX15

        );
/* Instantiation Template
CameraLinkOut #(
        .counterWidth(12),
        .betweenFramesDelay(2640),
        .betweenLinesDelay(190)
) instanceName (
        .clk(),
        .rst(),

        .dataValidIn(),
        .fEndIn(),
        .lEndIn(),
        .dataIn(), //bus [23:0]
        .readyOut(),

        .clo_clk(),
        .clo_fval(),
        .clo_lval(),
        .clo_dval(),
        .clo_a(), //bus [7:0]
        .clo_b(), //bus [7:0]


        .clo_c() //bus [7:0]
);
*/

reg ready;
//reg valid, validD;
reg fval, fvalD, lval, lvalD, dval, dvalD;
reg [7:0] a, aD, b, bD, c, cD;

assign clo_clk = clk;
assign readyOut = ready;
assign clo_fval = fval;
assign clo_lval = lval;
assign clo_dval = dval;
assign clo_a = a;
assign clo_b = b;
assign clo_c = c;


//counter for timing
reg [counterWidth-1:0] count, countD;

//state machine
parameter idleState = 0,
          betweenFramesState = 1,
                        lineValidState = 2,
                        betweenLinesState = 3;
reg [4:0] state, stateD;

always @ * begin
        //defaults
        //validD = valid;
        fvalD = fval;
        lvalD = lval;
        dvalD = dval;
        aD = a;
        bD = b;
        cD = c;
        stateD = state;
        ready = 0;
        countD = count;

        case(state)
        idleState: begin
                fvalD = 0;
                lvalD = 0;
                dvalD = 0;
```

```
                if(dataValidIn) begin
                        ready = 1;
                        if(fEndIn & lEndIn) begin
                                //frame End marker, we'll start a new frame next
                                stateD = betweenFramesState;
                                countD = 0;
                        end
                end
        end
        betweenFramesState: begin
                fvalD = 0;
                lvalD = 0;
                dvalD = 0;
                if(count == betweenFramesDelay-1) begin
                        stateD = lineValidState;
                end else begin
                        countD = count + 1;
                end
        end
        lineValidState: begin
                if(dataValidIn) begin
                        ready = 1;
                        fvalD = 1;
                        lvalD = 1;
                        dvalD = 1;
                        aD = dataIn[7:0];
                        bD = dataIn[15:8];
                        cD = dataIn[23:16];
                        if(lEndIn) begin
                                if(fEndIn) begin
                                        stateD = betweenFramesState;
                                        countD = 0;
                                end else begin
                                        stateD = betweenLinesState;
                                        countD = 0;
                                end
                        end
                end else begin
                        dvalD = 0;
                end
        end
        betweenLinesState: begin
                lvalD = 0;
                dvalD = 0;
                if(count == betweenLinesDelay - 1) begin
                        stateD = lineValidState;
                end else begin

                        countD = count + 1;
                end
        end
        endcase
end

always @(posedge clk) begin
        if(rst) begin
                fval <= 0;
                lval <= 0;
                dval <= 0;
                a <= 0;
                b <= 0;
                c <= 0;
                count <= 0;
                state <= idleState;
        end else begin
                fval <= fvalD;
                lval <= lvalD;
                dval <= dvalD;
                a <= aD;
                b <= bD;
                c <= cD;
                count <= countD;
                state <= stateD;
        end
end
endmodule
```