ECE532: Digital Systems Design
Final Project Group Report
PanCam

Group 26: Andy Gong, Yulang Luo, Xianglin Wang

April 2025

Contents

1 Overview

This project aims to create and implement a panoramic camera on the Nexys Video FPGA board, called PanCam. The user will simply hold and rotate the FPGA board, while the gyroscope Pmod will automatically detect angle changes and trigger image captures by the camera Pmod. The process of image taking and the final result are shown to the user via HDMI display. Since this design captures individual images at set angles, an image stitching processor is used to smooth the transition between the edge of images

1.1 Background and Motivation

The group initially debated whether to pursue a more computation- and algorithm-intensive project, such as accelerating linear programming. However, after further discussion, we decided it would be more valuable to use this opportunity to deepen our understanding of FPGA development tools—specifically Vivado, AXI interfaces, and DMA data transfer. Additionally, we discovered a shared interest in landscape photography, which inspired the idea of building a panoramic camera system. This project naturally involves interfacing with multiple hardware components, capturing and processing image data, and managing real-time data flow—all of which align well with our goal of gaining hands-on experience in digital design and embedded systems on FPGAs.

1.2 Goals

The main goal of this project is for the device to achieve the functionalities of a basic panoramic camera. The device accepts an user button press to initialize the image capture sequence. Then, when the user rotates the device, the gyroscope captures the angular velocity, which is then processed by the soft-processor on the FPGA and converted to the current angle the camera is facing. When the camera has rotated for a set degrees away from the previous image, the camera image is captured and saved to memory, with the edge of each image boundary with neighbouring images sent to an on-board IP to smooth the image boundary transitions. During this process, the full image width is not used due to warping of high FOV images can affect the quality of final image. In the end, when all the captured images are stored in memory, the resulting images are shown to the HDMI display. Due to the image potentially being very wide, the HDMI display only shows a portion of columns at a time, with user joystick input able to pan the image left and right to show different portions of the images.

The individual images taken by the camera is similar to those presented in Figure 1, with the ideal final result shown in Figure 2.

Figure 1: Individual images that could be captured by the camera

An image showing the high level overview of the image stitching process is shown in Figure 3.

In addition to the main goals, some additional goals that we aim to achieve is to utilize the on-board OLED display to show simple instructions to the users, including useful information such as “how many more degrees to rotate until next image capture”. Furthermore, the image boundary stitching IP could be designed to

Figure 2: Ideal combined panoramic image

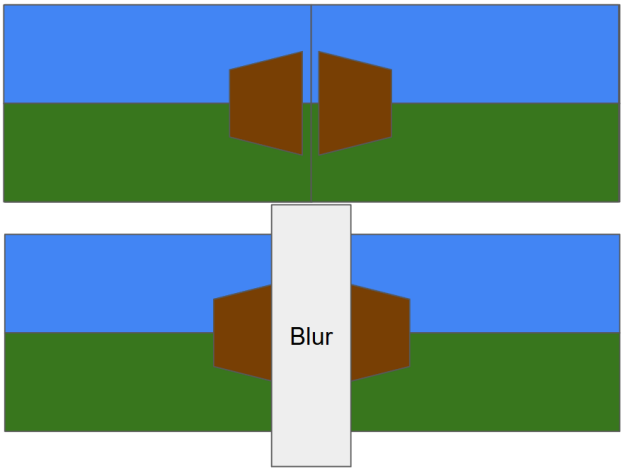


Figure 3: A diagram showing the idea behind image stitching, where a Gaussian blur is used to smooth out the intersection between images.

take into account the non-linear characteristics of the edges of images taken, and perform better smoothing of boundaries.

1.3 Block Diagram

Figure 4 shows the block diagram for the PanCam project. The design is split into four major components. The first major component is the HDMI data path used to display the camera input and the final result on a screen display. The second component is the camera input, where modules are required to program the OV7670 camera used and to read its image input to memory. The third component is the image processing data path, where the custom IP is used to process a data stream of columns of pixels by applying Gaussian blur on the input images' edges. The last component is the processor and control data path, consisting mostly the Microblaze processor, GPIO pins, and memory interface.

1.4 IP Descriptions

This section provide a high level overview of the IP blocks that are used in the project, separated into four major data paths.

1.4.1 HDMI Data Path

The HDMI data path transfers data asynchronously from the frame buffer to the HDMI port for display. It leverages IP from both Xilinx and Digilent. Our implementation integrates the output module from

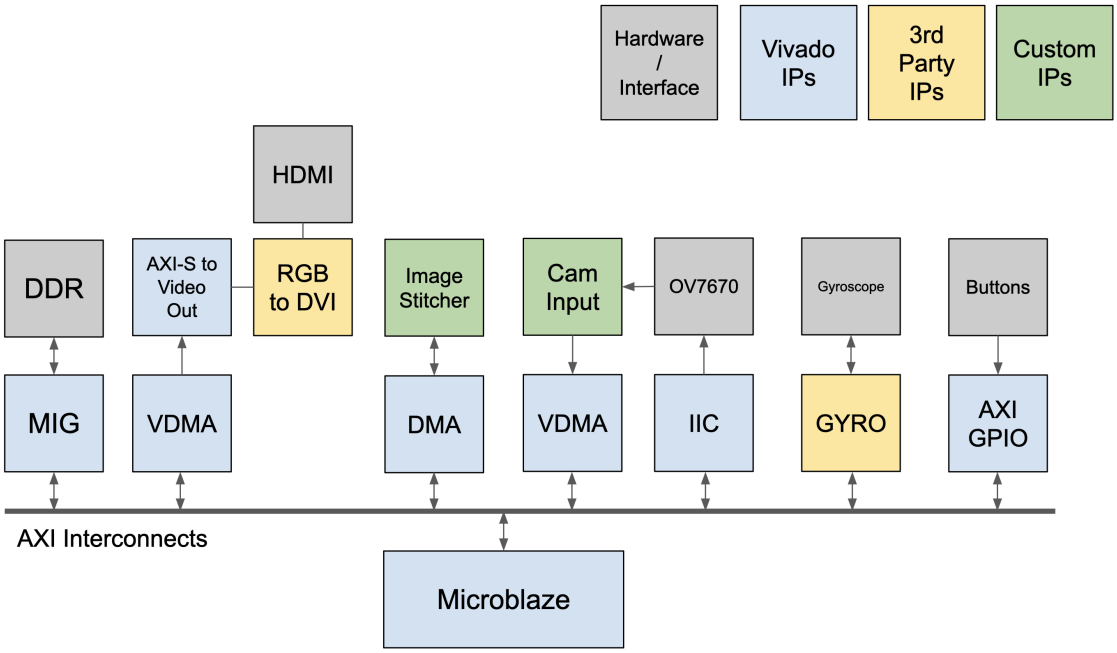


Figure 4: Block diagram of the PanCam project

the Digilent HDMI demo [1], whereas the original demo featured both input and output modules. In this configuration, a VDMA is used to move data from the frame buffer to the HDMI display in TMDS format.

1.4.2 Camera Data Path

The system utilizes an OV7670 camera module as the primary source of video input. Configuration of the camera is handled through the AXI IIC IP core provided by the Xilinx library, which interfaces with the camera's SCCB protocol. Once the raw image data is captured, it is routed through a custom-designed IP core for initial processing. The processed data is then transferred to the AXI VDMA module writing the data into a Frame Buffer on the DDR memory, preparing it for subsequent stages such as image processing or storage.

1.4.3 Gyroscope Data Path

The system incorporates a gyroscope module [2] to track the rotational movement of the board. This rotation data is used to determine when the panoramic camera system should pause and capture the next image for stitching. Once angular velocity data is retrieved from the gyroscope, it is processed by Digilent's custom gyroscope IP block and then passed to the MicroBlaze processor for further control logic and integration with the image capture system.

1.4.4 Image Processing Data Path

The image processing data path is responsible of applying filters to the edges of input images in an attempt to reduce possible misalignment that may occur at the boundary of consecutive images. This data path utilities a DMA module to transmit data to and from the image stitching IP block using AXI streaming interface with the memory. Within the image stitching IP, the block design is split into an input memory buffer that receives input from the AXI streaming interface and temporairly stores the image pixels for data reuse, and sends the pixels to be processed to 3 separate image processor cores that performs Gaussian blur

for each of the 3 colour channels. The result is sent to an output buffer that returns the filtered pixels in an output AXI streaming interface.

1.4.5 Processor and Control

Lastly, to connect all the modules together, a MicroBlaze IP is used as the software processor for this project, along with its associated blocks including an AXI Interconnect, AXI Uartlite, AXI Interrupt Controller, BRAM, and Clocking wizard. To connect to the DDR memory provided on the Nexys video board, a MIG was also included in the project. Additionally, AXI GPIO modules are added to allow the user to control the project with on-board buttons.

2 Outcome

This section provides the description of the project's outcome, as well as discussions regarding possible future steps that can be taken to improve the project.

2.1 Results

Overall, the project was implemented and tested in small increments, separated by hardware components. Individually, the project can interface with all the hardware components that is meant to be used.

The project is capable of capturing images from the OV7670 Camera and display them onto a monitor using HDMI interface. This is done through using VDMA IP blocks to read image input from the OV7670 camera to a specific range of memory on the DDR memory on the Nexys Video board called the Frame Buffer, and allowing the HDMI interface to use a separate VDMA IP to read data from the Frame Buffer, in order to display its content onto the screen. Figure 5 demonstrates this camera direct pass through to the HDMI display. Additionally, since the data is stored on the DDR memory, it can be accessed by the MicroBlaze processor and be routed to alternative address on the DDR memory as a image "capture" – storing a frame of image at any given time. It can be noticed in Figure 5 that the colour output has slight distortion, this is likely a white balance issue and should be resolved upon further adjustments.

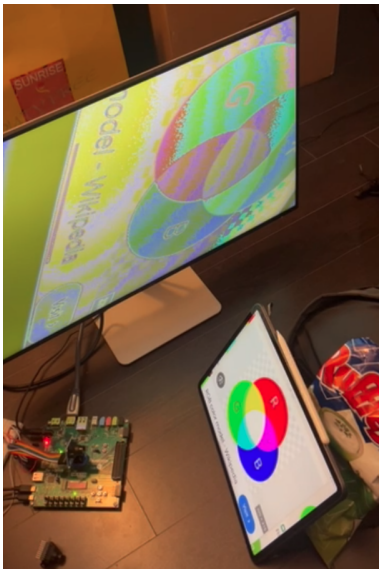


Figure 5: OV7670 Camera capturing an image and displaying the image on a display using HDMI

For the gyroscope, using Digilent gyroscope IP from [2], the Nexys Video board can receive from the gyroscope Pmod the angular velocity on all three axis of rotation at any given time on MicroBlaze. By filtering out small angular velocity values, and performing software integration on the angular velocity received at set time intervals, we were able to determine the relative angle of rotation of the board at any given time.

The image stitching IP was developed separately from the rest of the project. Since the first submodule, every single sub-module has been tested rigorously with verilog testbench, including varying hardware parameters. The IP is capable of receiving a stream of pixels, and outputting the average value of each pixel with its neighbouring pixels as the output of applying a Gaussian blur to the image, tested in verilog testbench shown in Figure 12.

Software was developed to combine the modules together following the software processing logic presented in Figure 13, where at set angle rotations, an image would be captured and split into three sections, a left-edge, centre, and right-edge. The centre of an image is unmodified and directly loaded onto a final display memory, and the edge of images would be blurred and stitched by the image stitching IP. The data path is shown in Figure 14, and the split of every input image is shown in Figure 6. The far edges of images were not used to prevent high FOV camera image warping near the edges.

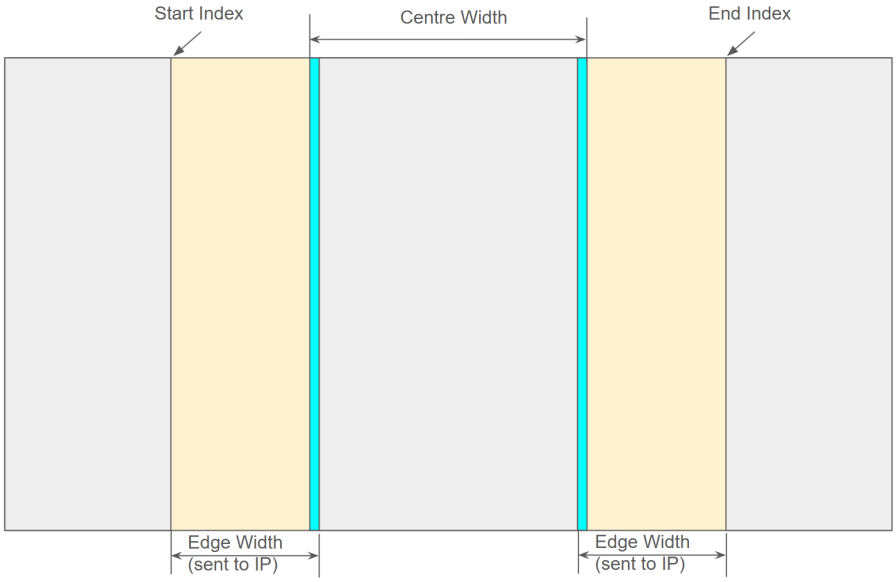


Figure 6: Input image split between edge and centre

All core goal of the project is achieved in their separate components, and several components have already shown to be able to work together and produce partial results.

Unfortunately, the Nexys video board provided for this project became nonfunctional right as the group wanted to perform the final integration of the entire project. With only days to spare and no new Nexys Video board available for use, an attempt was made to migrate the project onto a Zedboard provided for us. However, due to the new board having completely different interfaces and the lack of time for us to adapt to this new working environment, the final project was not able to have all of its components integrated.

2.2 Future Work and Improvements

Given additional time, there are several stretch goals we would like to pursue to enhance the user experience and potentially improve the quality of the panoramic images.

2.2.1 OLED Display

Currently, the control logic in the software loop is not immediately intuitive to users. To address this, we plan to utilize the onboard OLED display on the Nexys Video board to provide user guidance. This could include prompts on how to start the capture sequence, when to rotate the camera, and how to pan left or right through the final stitched image. Providing real-time visual feedback would make the system more user-friendly and self-guided.

2.2.2 Joystick Interface

The current panning mechanism relies on push buttons, which are not ergonomically ideal. Replacing them with a joystick would offer a smoother and more comfortable control experience, especially when navigating across the wide panoramic output.

2.2.3 Improved Image Stitching Algorithm

At present, the image stitching algorithm applies a Gaussian blur along the boundaries of adjacent images. While simple and efficient, this method only works well when lens distortion is minimal and the field of view is narrow. To support wider panoramic views, a more advanced stitching algorithm could be implemented—one that compensates for lens distortion through non-linear pixel mapping. This would allow for fewer input frames to cover the same panoramic width, reducing mechanical rotation error and potentially improving the quality and consistency of the final output.

2.2.4 Camera Image Format

The current camera configuration does not produce accurate color output. To address this, further testing with different configuration settings is necessary. In addition to selecting the correct color format, there are various camera registers related to white balance, saturation, and other image parameters. Tuning these settings is important to ensure visual consistency across frames—particularly when images are stitched side by side in the panoramic view. Proper calibration can significantly improve the overall appearance and quality of the final output.

2.2.5 Integration

Because of the earlier board failure, full system integration and testing could not be completed. Although all individual components have been developed and verified through unit tests, we now require a functional board that is compatible with our existing test setup. The next critical step is to integrate all modules and perform system-level testing on a working board to ensure correct functionality and performance.

3 Project Schedule

3.1 Intended and Actual Milestones

The intended and the actual milestone achieved, separated by milestone weeks, are included in Table 1.

Table 1: Description of the weekly milestones for this project

Original Milestone	Final Milestone
Milestone 1	
• Complete the block diagram for the image stitching IP including processor and data-reuse router. • Access the Nexys Video HDMI sample project. • Research existing OV7670 camera projects using VGA. • Look for existing gyroscope SPI interface source code and evaluate whether a similar approach can be applied to our project. • Begin planning and preparing function prototypes for Microblaze functionalities required.	• Generated bitstream for HDMI interface sample project. • Researched OV7670 sample project and datasheet. • Designed IP core with 3 parallel channels and 3x3 Gaussian blur. • Developed a pipelined data buffer for efficient streaming in column-major order. • Pseudo code written for final MicroBlaze control logic
Milestone 2	
• For custom IP, finish creating basic 3 by 3 processing units and create unit tests for this module. • Implement HDMI on Nexys Video board and display a custom image/shape. • Begin implementing camera input and control on the board. • Show data being read from the SPI interface connecting the gyroscope or joystick. • Create supporting functions that can interface with the PMOD testing.	• HDMI output successfully implemented on Nexys Video board, capable of showing custom patterns. Resolved VDMA and frame buffer alignment issues. • Created and tested 3×3 Gaussian blur processing unit (custom IP). • Started custom AXI streaming interface development. • Camera input and control in progress on board. • Initial gyroscope testing initiated using Digilent PMOD library.

Milestone 3	
• Complete IP data routing module, test the functionality of routing module. Integrate routing and processing sections of the module together. • Display camera input live through HDMI • Read accurate angle data of board rotations from the gyroscope through MicroBlaze UART. • Create supporting functions that can interface with the PMOD testing in Microblaze.	• Completed and documented flexible input buffer for custom IP core. • Output buffer design and integration in progress. • Started integration of camera module with HDMI display. • Developed custom video input block and format converter directly connecting to HDMI output. • Gyroscope successfully connected and readable via MicroBlaze UART. • SPI integration with MicroBlaze functional and tested.
Milestone 4	
• Completed integration of the IP core and can interface with live camera image input. • Camera image input should now be routed to the IP core for "edge stitching" • Gyroscope angle data should be associated with the image taken by the camera, and should be used by the camera control to decide when to take pictures. • Support the routing of camera image to the IP core, and use gyroscope data to control the camera.	• Camera input successfully displayed on HDMI from writing camera input to Frame Buffer. • Camera image visible through HDMI, however, colours are incorrect, camera seems to not be programmed by SCCB interface. • Designed and tested custom memory access logic for frame storage. • Custom IP core fully tested in simulation and packaged. • IP integration with MicroBlaze and DMA in progress.

Milestone 5	
• Integration of all modules together to achieve the base functionality of the device design, automatic image capture and stitching. Displaying the final result on the HDMI screen. • Begin implementing joystick image-panning on the HDMI screen. • Attempt to improve the image stitching IP to include linear transformation stretching of the images to make a smoother image stitching attempt. • Add interfacing to the OLED display, to show the angle data and instruct the user to rotate for subsequent camera images.	• Successfully programmed the OV7670 camera using AXI IIC Vivado ip through the camera's SCCB interface. • Camera now displays recognizable colors with ongoing RGB tuning. • Integrated AXI DMA and began testing memory write to IP core. • Used a color bar test to confirm functional camera programming and output path.
Milestone 6	
• Final testing of the integrated modules, and add the more advanced image stitching processor to the integrated design, and get ready for the final presentation.	• Completed full software integration for all modules. • Panoramic image stitching pipeline implemented in software. • Gyroscope integrated for system control and image capture flow. • Optimized memory layout and data flow between IP and display, and added ILA debugging support for IP and camera interface. • Nexys Video board broken, hardware and integration test blocked by the broken board. • Tested project migration to Zedboard.

3.2 Milestone Discussions

Building a high-level architectural plan for the project and conducting background research were the main goals of the first milestone. The team stayed on track in the early phases, and implementation started in the second milestone. However, issues with the camera module and a misguided design approach caused the project to encounter a major delay starting from Week 3.

Originally, the project planned to allow the HDMI display to show a live-feed of camera video input by milestone 3 as a proof of a working camera interface, and to confirm that the camera is recording data in the correct format that can be recognized by the HDMI output interface. Throughout week 2 and week 3, significant effort was spent developing and testing a method to connect the camera's input directly to the HDMI output port via AXI streaming. However, this approach quickly revealed some fundamental issues

– the camera input and HDMI output have significantly different data format and timing requirements, making it difficult to directly port the camera input to the HDMI output. Later it was realized that a direct connection from the camera input to the HDMI output was not only impractical, but also unnecessary – since the idea behind camera input was to make the image available in memory to be processed by other hardware and software logic, and a direct pass through would not store the image in memory and thus not useful for the remainder of the project. This misstep costed us roughly a week of lost progress. After milestone 3, a new approach was devised: instead of forcing the HDMI output to take the camera input, we use VDMA block to write the camera input to the same memory space as the HDMI output buffer. This way, the image is stored in a region of memory accessible by other software and hardware, and it allows the HDMI output to directly display image captured from the camera.

The updated method successfully displayed live camera output on the HDMI display in Week 4. Nevertheless, there was still some obvious color distortion in the output. After further investigation, it was discovered that the camera module itself was the cause of the problem because its SCCB interface was unresponsive, pointing to a hardware issue. After a replacement OV7670 module was ordered and installed, the issue was fixed, evident by the new camera being able to display a test pattern.

Overall, the delay caused by the initial approach on the camera to HDMI pass through and the delay caused by the faulty camera not responding to SCCB interface programming has brought a roughly 2 weeks delay in the development of the overall project. However, this delay was accounted for during the planning stage of the project, where the goals of this project have been split into core goals and reach goals, and the later milestones reserved for such reach goals. With this delay in mind, the group would be performing the components' integration for Milestone 6 – two weeks before the end of project, and reallocate the times initially planned for the reach goals to the integration of core goals.

Under normal conditions, this would have allowed the project to finish on time. However, between Milestones 5 and 6, the Nexys Video board stopped working, as the programming port of the board suddenly became non-responsive. The failure happened just as final integration was about to begin. The issue was reported to the teaching team at the start of Milestone 6. At that point, only one week remained for the full system integration.

The team explored migrating the project to another FPGA board, the ZedBoard as no other Nexys Video board was available for use. However, differences in architecture, constraint files, IP cores, and HDMI support made the transition too complex. There was not enough time to complete the migration. As a result, software integration and module development were finished, but final hardware testing and full system integration could not be completed.

4 IP Block Descriptions

This section provides detailed descriptions of the IPs used for the project, separated by the major data paths.

4.1 HDMI Data Path

This data path outputs RGB data from the frame buffer to the HDMI display. The input is stored in off-chip DDR memory as a tightly packed sequence of 8-bit green, blue, and red channels. The data path is derived from the Digilent Nexys Video Board HDMI Demo [1]. During the initial implementation attempt of the demo, an error occurred in the VDMA start sequence due to a software check on the frame buffer's starting address alignment. Since the frame buffer was defined as a C global array, the solution was to apply the alignment attribute in the compiler. Later, the team also discovered that enabling the misaligned transfer option in the VDMA block provided an alternative fix. The HDMI data path's functionality was validated by generating arbitrary patterns across all color channels in the frame buffer and confirming that the display accurately reproduced the memory content. Later, after the camera is implemented, the HDMI data path was also capable of directly presenting the camera's image capture onto the HDMI display.

4.1.1 Video DMA

In order to free CPU from the burden of streaming videos and to enable faster HDMI display, the group uses a DMA to move data asynchronously from frame buffer to HDMI. Xilinx has a IP that's called Video DMA. Compared to Xilinx's regular DMA block, VDMA has the benefit of continuously looping over a frame buffer to ensure a non stopping stream without any intervention from the CPU such as calling DMA start again. This looping nature makes it work seamlessly with the video timing controller. The VDMA's output formats is AXI stream. This IP is added by the Digilent HDMI Demo [1], and we modified this IP to enable misaligned transfer, and disabled the input port which was originally reserved for HDMI input.

4.1.2 Video Timing Controller

The Video Timing Controller is in generation mode, given by the Digilent HDMI Demo [1]. It generates the vertical, horizontal sync, and blanking signals for the VDMA. Both the Video Timing Controller and the Video DMA are driven by the same dynamic clock.

4.1.3 AXI-S to Video Out

Both the AXI stream from VDMA and the synchronization signals from the Video Timing Controller drive this module, provided by the Digilent HDMI Demo [1]. It concatenate all color channels and pass on the synchronization signals.

4.1.4 RGB to DVI encoder

This module is created by Digilent [1]. It converts the parallel RGB on the dynamic pixel clocks to TMDS signals for outputting to HDMI. The output to HDMI is in the format of TMDS. All signals has positive and negative wire pair. It has clk, red, green, and blue signals. Data are transmitted in 8b10b encoding.

4.1.5 Dynamic Clock Generator

The dynamic clock in the Digilent demo [1] was used for runtime adjustable display resolutions. Since for a fixed frame rate, higher resolution would require more data to be transmitted. Both the clock to VTC and VDMA and the clock to RGB to DVI encoder will need to be adjusted. Since changing resolution is not required in this project, the software would initialize a clock speed and never change it again throughout runtime.

4.2 Camera Data Path

The camera data path is designed to receive, process, and stream image data from the OV7670 camera module to the HDMI output and image processing pipeline. It is composed of three major stages: camera configuration via IIC/SCCB, data conversion through a custom IP, and memory write using the AXI VDMA. Each stage has been customized to accommodate the camera's data format and ensure compatibility with the rest of the system. An image of the Camera Input IPs is shown in Figure 7.

4.2.1 AXI IIC

The first stage of the data path involves configuring the OV7670 camera module. An AXI IIC module from the Xilinx IP library is used to communicate with the camera's SCCB interface, enabling register-level configuration to define output formats and operational modes. Initially, a custom 3rd party Verilog-based controller was used to program the camera [3], but it failed to reliably write to the SCCB interface. The team

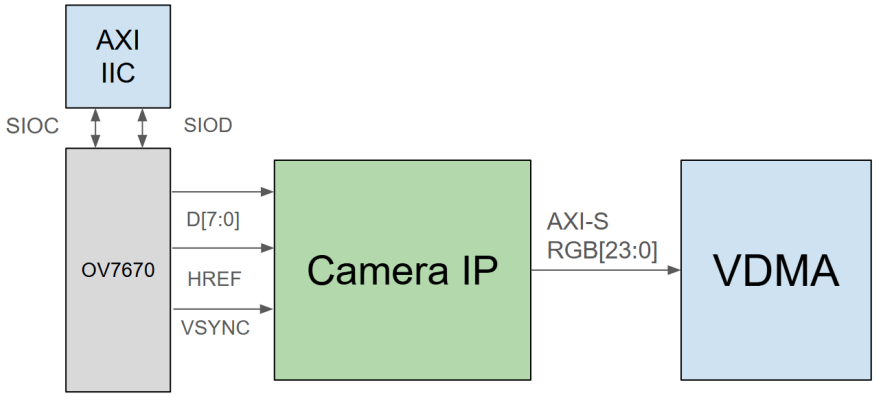


Figure 7: Camera IPs

subsequently adopted the AXI IIC IP core, which handles IIC transactions in hardware, while register writes are performed through software. The specific register settings and their intended configurations are referenced in the camera datasheet [4]. The current configuration ensures a stable image output, although some color distortion remains. This approach not only provides reliable camera initialization but also eliminates the need to rerun synthesis whenever updates to the camera configuration are required. The RGB565 format was selected for image output due to its balance between data size and color accuracy compared to lower-bit alternatives.

Since both SIOC and SIOD wires are driven by a logic 0, a pull-up resistor is required physically to allow the camera to be programmed.

4.2.2 Camera Input IP

The second stage handles image reception and format conversion. The OV7670 outputs RGB565 pixel data across an 8-bit data bus over two consecutive clock cycles. The timing and structure of this output is illustrated in Figure 8.

A custom camera input IP module was developed to receive the camera output, reconstruct RGB565 pixels, and convert them to RGB888 format. This IP is clocked using the pixel clock (pclk) provided directly by the OV7670 camera. The pclk signal serves as the synchronization clock, where each rising edge corresponds to a valid data byte on the din input bus. This clock is derived internally by the camera from the external xclk input, which is supplied by the board and operates at 25MHz. Over two pclk cycles, one full RGB565 pixel is transmitted.

The module implements a 2-stage conversion using an internal counter to track whether the incoming byte corresponds to the first or second part of the pixel. On the first cycle, the red component is captured and stored in the upper byte of the 24-bit output. On the second cycle, the green and blue components are extracted, expanded to 8 bits using left shifts, and stored in the remaining two bytes. This results in a 24-bit RGB888 pixel. This conversion is necessary because both the HDMI display controller and the custom image processing pipeline are designed to operate with tightly packed 24-bit RGB888 pixel data.

Once the full RGB888 image format is stored as a 24-bit data, it is then output to the VDMA IP using AXI Streaming format. Since new pixels arrive every pclk cycle, the **tready** signal is ignored by this IP and new data is sent every clock cycle.

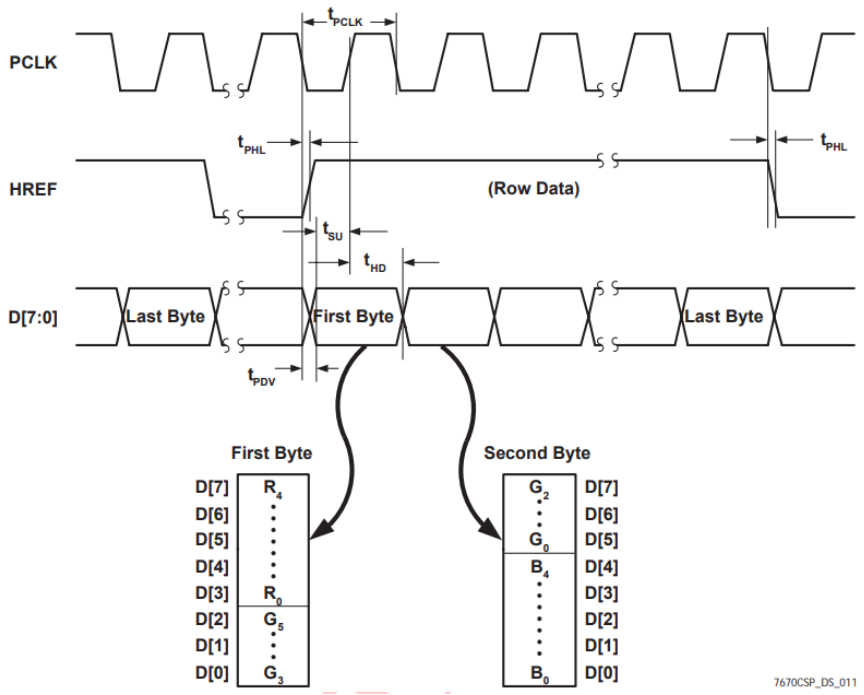


Figure 8: Timing diagram of OV7670 RGB565 data format [4]

4.2.3 VDMA Configuration

To ensure compatibility with the RGB888 pixel stream produced by the custom IP, the Xilinx AXI VDMA was reconfigured to accept unaligned input and to operate with a 24-bit input data width. The default configuration of the VDMA expects 32-bit aligned data, which is incompatible with the tightly packed 24-bit RGB888 format. By adjusting the VDMA's data width to 24 bits and enabling unaligned transfers, the system is able to stream camera data correctly from the custom IP through the VDMA to the HDMI display pipeline without data loss or misalignment.

4.3 Gyroscope Data Path

The system uses the ST L3G4200D gyroscope module from Digilent [2] to measure angular velocity, which is then integrated in software to estimate the board's rotation angle. Digilent provides a compatible gyroscope IP core [5] that facilitates data acquisition over the SPI protocol. In addition, Digilent offers a sample reference project [5] where the gyroscope is initialized via software, and angular velocity data is transmitted over UART. Our software implementation builds upon this sample project, extending it to perform real-time integration of angular velocity in order to calculate the board's orientation. This integrated angle data is then used to determine when to trigger image capture within the panoramic stitching pipeline.

4.4 Image Processing Data Path

The image stitching IP, at a high level, receives a stream of pixel values from AXI streaming interface in the column-major order across the full height of the image. As an output, the IP will be sending a stream of processed image pixel values that was passed through a Gaussian blur filter, also in a column-major order.

All the data input and output to and from this IP is directly routed to a DMA IP block connecting the IP to the DDR memory.

4.4.1 Image Stitching IP

The high level overview of the IP is shown in Figure 9.

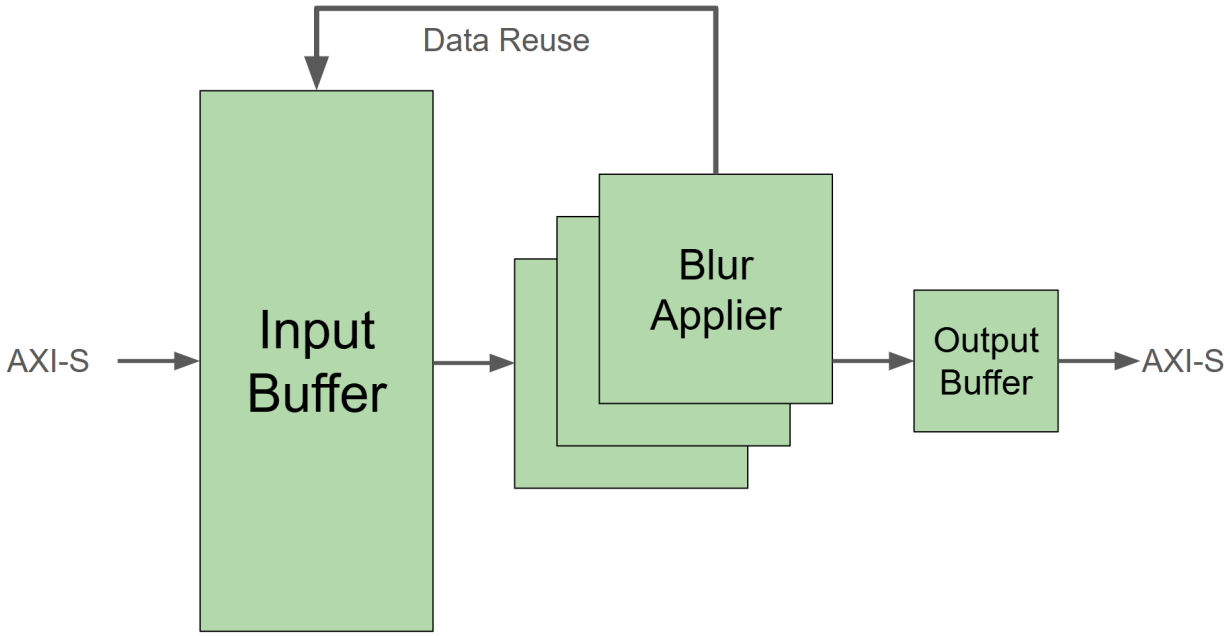


Figure 9: High level overview of the image stitching IP

The input data structure of this IP is in the form of 32-bit values, with the highest 24 bits containing an RGB888 formatted pixel value, and the last byte is ignored. Upon the data being received by the IP, the pixel values are then split into three separate independent layers of registers and data paths corresponding to the three RGB colour channels. The IP will also be waiting for a **tlast** signal to be sent by the AXI Streaming interface to indicate the that the last pixel of the last column to be processed has been delivered to the IP.

As shown Figure 9, the IP has three separate sub-modules: the Input Buffer, the Blur Applier (the Image Processor), and the Output Buffer.

The Input Buffer's block diagram is shown in Figure 10, where data sent to the IP via AXI Streaming is received. Within the Input Buffer, there are $N \times M$ registers arranged in the format of M shift registers each having length N, with the input to the registers being at the bottom and the output on the top. Here, N is the total column height of the column of image to be sent to the IP, and M is the width of the filter (typically 3). When a pixel is sent to the IP, it is received by the bottom right register and all the data is shifted "up" by one space. The data input for any subsequent columns of shift registers are being input from the blur applier. Overall, the blur applier will simply send the rightmost column's output from the input buffer back to the middle column's input after a delay by M cycles, and the middle column's output sent back to the left most column's input M cycles later. This circular structure of the input buffer allow for each pixel to be passed through a data pipeline where each data can be used in all 9 related filter operation in a 3×3 filter, and more for larger filters. This is because every column of data input is sent to the input side of the input buffer M times, allowing each column to be used as an input to all M columns of the image filter.

To ensure proper data alignment of the data from different columns, delays are added to the input buffer once a whole column of data has been received from the AXI-S interface, where the bottom right register is filled with M zeros after a full column's input. This is to ensure that the M cycle delay used by the Blur Applier will still allow the first row of all columns to be sent to the Blur Applier at the same cycle. Finally,

since this pipeline require constant data flow to perform filter computations, when all the column inputs has been received by the IP, the **tlast** signal instructs the input buffer to continuously "flush" out the last column input to produce the output of the final column.

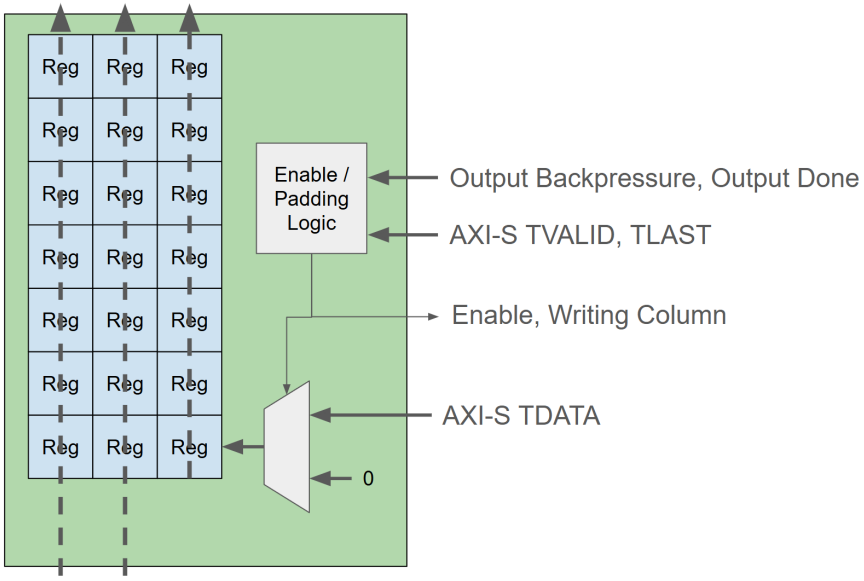


Figure 10: Input Buffer module overview of the IP

The blur applier is shown in Figure 11. This is the module that performs the Gaussian blur computation on the columns of pixel input. Since an Gaussian blur filter averages the pixel value of all the pixels within the filter, it is equivalent of multiplying every pixel by a factor of $\frac{1}{M^2}$ in fixed-point format and summing them together. In this module, the data is a flow of M columns through the device in a shift-register style. After each cycle, a pixel is shifted "upward" by one register, while its value is multiplied by a fixed-bit representation of the filter value $\frac{1}{M^2}$ and stored in a temporary register. Then the value of each row is summed in the next cycle, while the sum of each row is then summed together in the next cycle to the filter output. This separation of multiply and accumulate operation is to ensure that the data path has no one single large computation that may severely limit the maximum allowed clock frequency.

As previously mentioned, the Blur Applier will output each column input after an M cycle delay, allowing the input buffer to reuse the data of each pixel.

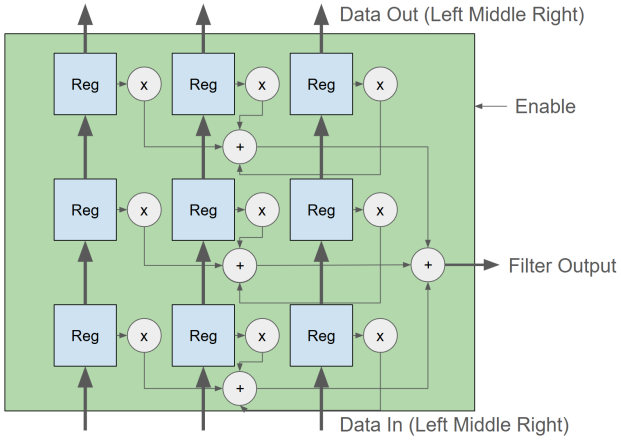
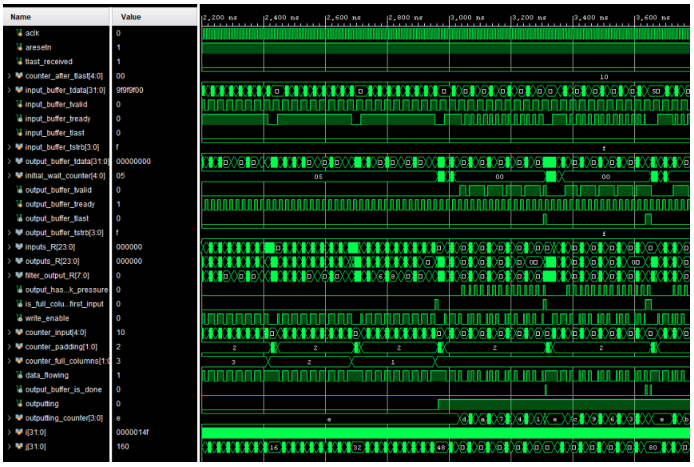


Figure 11: The Blur Applier module of the IP

The Output Buffer receives the pixel output of the Blur Applier, and sends the data through AXI Streaming

interface. For this module, it will initialize a signal indicating if any "back-pressure" exists for the data pipeline, where a data is valid to be output but the AXI Streaming interface is not yet ready to receive such data. In this case, the entire pipeline would halt until this data congestion is resolved. This is to prevent data being lost at the interface, since the IP is designed to require a consistent stream of data to compute correct output.

The functionality of this IP has been verified rigorously through software simulation. A stream of varying input has been generated and was sent to the IP from an AXI Streaming interface testbench, and the IP output is read from a separate AXI Streaming interface testbench. To verify both the device's ability to produce correct output, as well as its ability to correctly interface with AXI Streaming, the input valid and the output ready signals are set to randomly turn on and off during the transfer of a set of data. The testing waveform is shown in Figure 12a, and a printout of the data received by the IP's AXI output streaming interface is shown in Figure 12b, matching the expected output from the sequence of pixel values sent to the IP.



(a) IP Testing Waveform

```
ONE COLUMN DONE
Output: R= 97, G= 97, B= 97
Output: R= 98, G= 98, B= 98
Output: R= 99, G= 99, B= 99
Output: R=100, G=100, B=100
Output: R=101, G=101, B=101
Output: R=102, G=102, B=102
Output: R=103, G=103, B=103
Output: R=104, G=104, B=104
Output: R=105, G=105, B=105
Output: R=106, G=106, B=106
Output: R=107, G=107, B=107
Output: R=108, G=108, B=108
Output: R=109, G=109, B=109
Output: R=110, G=110, B=110
```

(b) IP output printout

Figure 12: Two images side by side

4.4.2 DMA

In order to connect the Image Stitching IP to the data that has to be processed, a Xilinx DMA IP from Vivado is used to provide the IP with AXI streaming data originated from the MIG and converting the IP's AXI Streaming output to the MIG.

4.5 Processor and Control

The MicroBlaze soft processor is responsible for controlling all AXI peripherals in the system. It also runs the main control loop. At startup, MicroBlaze initializes all AXI peripherals, configures the camera via the IIC interface, and allocates required image buffers on the stack. It then enters the main loop illustrated in Figure 13.

The program begins execution upon a user button press. In each loop iteration, the system captures an image from the camera. Once captured, the camera's VDMA is paused, and the image buffer is copied to a per-frame storage buffer. Meanwhile, a DMA is configured to receive the output from the image stitching module.

Simultaneously, another DMA is responsible for sending edge regions from the current and previous frames to the image stitcher for processing. Once the stitching DMA completes, the result is written back to memory.

A blocking angle integration function is then called to track rotation. The loop continues until a predefined rotational threshold is reached.

After the final frame has been processed, all stitched images are stored in a wide, row-major memory region. A strided VDMA then reads this memory to display the panoramic result on an HDMI output. Upon further user input, the base address of the VDMA is shifted left or right, allowing the HDMI display to pan horizontally across the panoramic image.

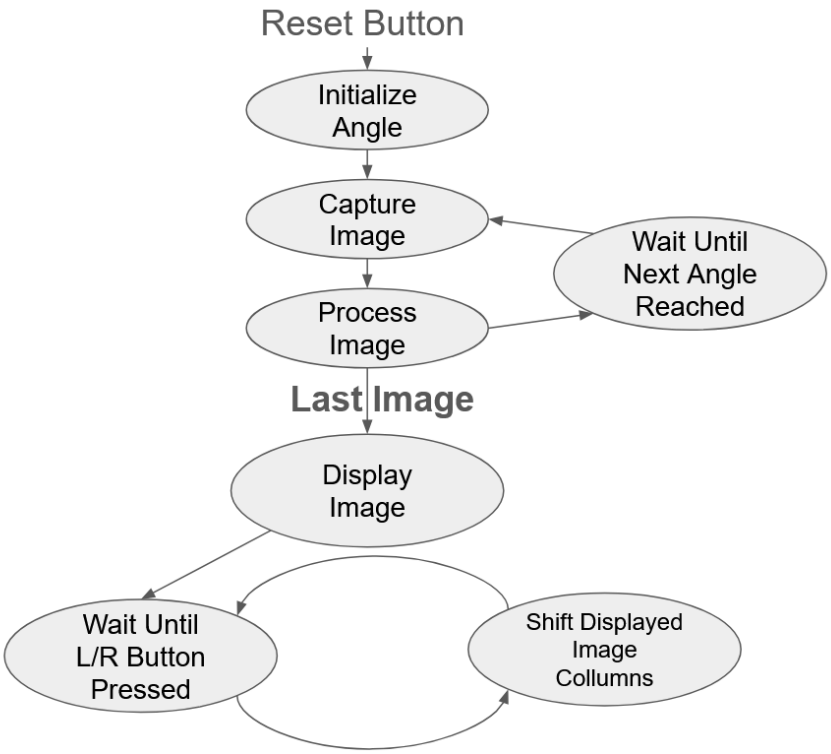


Figure 13: Software Loop Running on MicroBlaze

The memory data path of the project is presented in Figure 14, where each memory transfer between a hardware component / IP (shown in yellow boxes) are done via DMA or VDMA blocks.

4.5.1 MicroBlaze

The MicroBlaze soft processor as well as its associated blocks, provided by Xilinx in Vivado, was instantiated as part of the HDMI Demo project [1]. Although MicroBlaze can typically be instantiated using Vivado's Block Automation, in our design it was included directly from the reference project. The MicroBlaze subsystem includes several key components: BRAM for instruction and data storage, an interrupt controller, and AXI interconnects for communication with peripherals. This processor is responsible for executing the control logic and managing all AXI peripherals in the system.

4.5.2 Memory Interface Generator (MIG)

The MIG IP is used to interface with the onboard DDR3 memory on the Nexys Video board. Like the MicroBlaze processor, the MIG in our design was not instantiated manually, but rather included as part of the original HDMI Demo project [1]. It enables high-throughput access to off-chip memory, which is essential for buffering large image frames during panoramic capture and stitching.

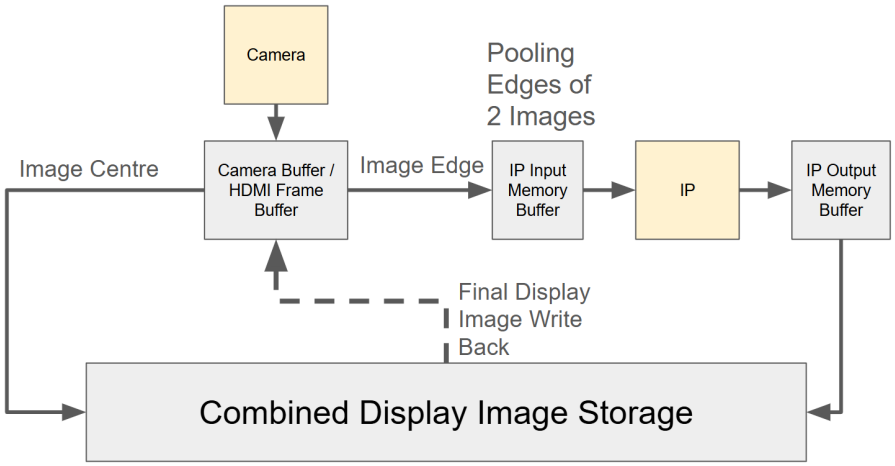


Figure 14: Memory data path of the project

4.5.3 AXI GPIO

The Xilinx AXI GPIO core was added manually using Vivado. It is used to interface with buttons. In our system, it plays a role in user input handling, such as triggering the start of image capture or panning the panoramic display.

4.5.4 Clocking

The system's primary clock is provided externally and used to drive the MIG, which in turn generates the internal ui_clk. This clock serves as the source for most internal logic, including the MicroBlaze processor and AXI peripherals. This was initially included as a part of the Digilent HDMI Demo [1].

To support the external camera, a separate 25 MHz clock was generated using a Clocking Wizard. This clock is used to drive both the camera module and its configuration interface via IIC. Additionally, the camera provides its own 25 MHz pixel clock back to the FPGA. This pixel clock is used to drive the camera-side VDMA as well as the custom camera input IP, ensuring proper timing and synchronization of incoming video data.

5 Design Tree Description

The project's design files have been uploaded to GitHub at the following repository:

https://github.com/AG2048/ece532-project

Within this repository:

- `README.md` is the README file providing a high-level overview of the project, as well as how to navigate the repository and how to run the project.
- `image_processor/` is the directory storing the verilog and testbench files used during the development of the Image Stitching custom IP.
- `OV7670_RGB565/` is the directory storing the verilog file of the Camera Input IP.

- `packaged_IP/` is the directory storing the packaged version of some of the IPs used in this project, namely the Image Processor and the Gyroscope IP.
- `final_software/` is the directory containing the final main c file.
- `vivado_proj/` is the directory storing the entire vivado project for the final version of the project. This project is a modified version of the Digilent HDMI Demo [1].

6 Advice for Future Students

6.1 Source control and collaboration

In this project, the team primarily worked with custom Verilog for IP blocks, C code for the MicroBlaze processor, and Vivado project files. Git is well-suited for version-controlling Verilog and C code. However, tracking the entire Vivado project in Git leads to excessive noise due to frequent changes and the complexity of determining which files should be ignored. Initially, we tried uploading zipped project directories—including intermediate files—to Google Drive, but this approach proved problematic. Moving entire Vivado projects across machines can introduce bugs, particularly due to differences in IP configurations and Vivado's sensitivity to absolute versus relative paths. After researching best practices, we found that many recommend using Git to track only the Vivado TCL scripts, which can regenerate the entire project. We strongly encourage this approach for managing Vivado projects effectively.

6.2 ILA and unit test

The team learned the importance of integrating ILA (Integrated Logic Analyzer) cores early in the development process. Hardware bugs are common after initial implementation, and having ILAs already in place can save significant build time. Additionally, inserting ILA blocks early on encourages designers to double-check that all clocks and resets are properly connected. Since hardware debugging is inherently time-consuming, it's generally more efficient to implement features in software when possible. For instance, we faced numerous challenges configuring our camera. Initially, the IIC interface was implemented entirely in hardware, meaning each new configuration required a full rebuild and long wait times. Later, we switched to using an AXI IIC core, allowing us to configure the camera through software, which greatly streamlined the debugging process.

6.3 Project Prioritization

In addition to the unexpected board failure, another key factor that affected our final results was time and priority management. The team devoted a significant amount of time to resolving issues with the camera, which delayed system integration by approximately a week. When the camera began displaying incorrect colors, all group members focused exclusively on diagnosing and fixing the problem. In hindsight, a more effective strategy would have been to allow some members to proceed with integration in parallel, or to temporarily accept the incorrect color output and revisit it later. Compared to full system integration, color inaccuracy was a relatively minor issue that should not have blocked overall progress.

7 Video

A video of the team explaining and demonstrating system components are included in the following link:

https://youtu.be/m2OMyKpjHQ8

References

[1] Digilent Inc., "Nexys Video HDMI Demo," https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-video-hdmi-demo/start, accessed: 2025-04-06.

[2] Digilent Inc, "Pmod GYRO Reference Manual," https://digilent.com/reference/pmod/pmodgyro/reference-manual, 2023, accessed: 2025-04-06.

[3] A. Nair, H. Ghosh, and T. Patel, "ECE532 - Animate With Action," https://github.com/nairadi/ECE532/blob/master/project_1/video_in/video_in.srcs/sources_1/new/I2C_OV7670_RGB444_Config.v, 2018, accessed: 2025-04-06.

[4] OmniVision Technologies, "OV7670/OV7171 CMOS VGA (640x480) CameraChip Sensor," 2006, accessed: 2025-04-07. [Online]. Available: https://web.mit.edu/6.111/www/f2016/tools/OV7670_2006.pdf

[5] Digilent Inc., "Digilent Vivado Library," 2025, accessed: 2025-04-07. [Online]. Available: https://github.com/Digilent/vivado-library