ECE532 - Digital Systems Design: Final Report

Autonomous Search and Rescue Robot Using FPGA-Based Audio Localization

Rozilyn Marco
Joshua Piruzza
Jake Sprenger
Balaji Venkatesh

2024-04-12

Table of Contents

1. Overview

Our project was an audio-guided search-and-rescue robot with a live video stream to a base station. As shown in Figure 1, we split this project into four subsystems spanning two FPGAs. We were able to design and build a functional prototype.

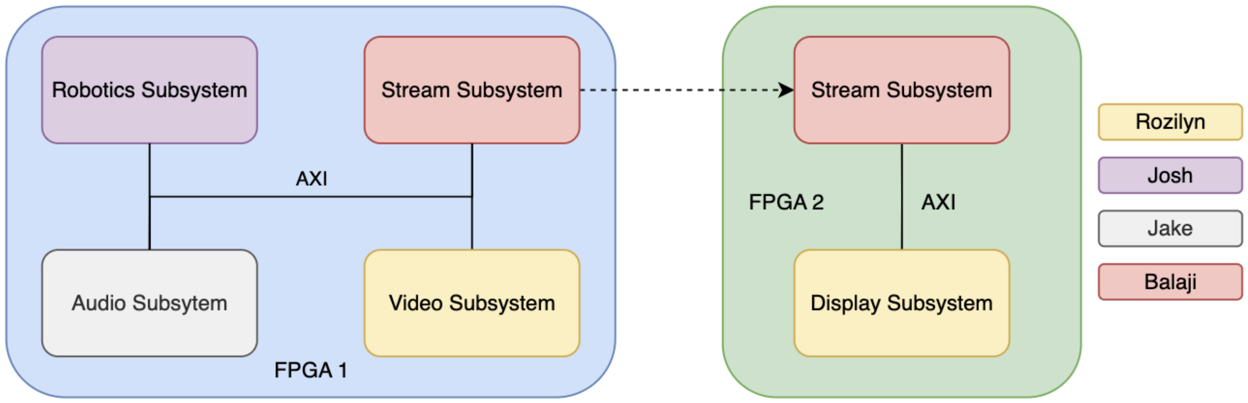


Figure 1 - Project Overview

1.1. Motivation

A search and rescue robot equipped with audio localization and video streaming capabilities can significantly enhance search operations in disaster-stricken or inaccessible areas. Audio localization allows the robot to detect and move towards sounds of distressed individuals, even when they are obscured by debris or in visually impenetrable locations. Video streaming to a base station enables real-time situational awareness, allowing human operators to make informed decisions, strategize rescue efforts, and identify hazards or victims' locations without being physically present. This combination of technologies increases the efficiency and safety of search and rescue missions, potentially saving more lives by rapidly locating and assessing the condition of victims.

1.2. Goals

Our goals were to:

- ☑ Build and control a robot
- ☑ Move the robot to find a speaker in a two dimensional space
- ☑ Stream live video and metrics from the robot back to the base station
- ☑ Display video and metrics on a monitor through VGA

1.3. Block Diagrams

1.3.1. FPGA 1

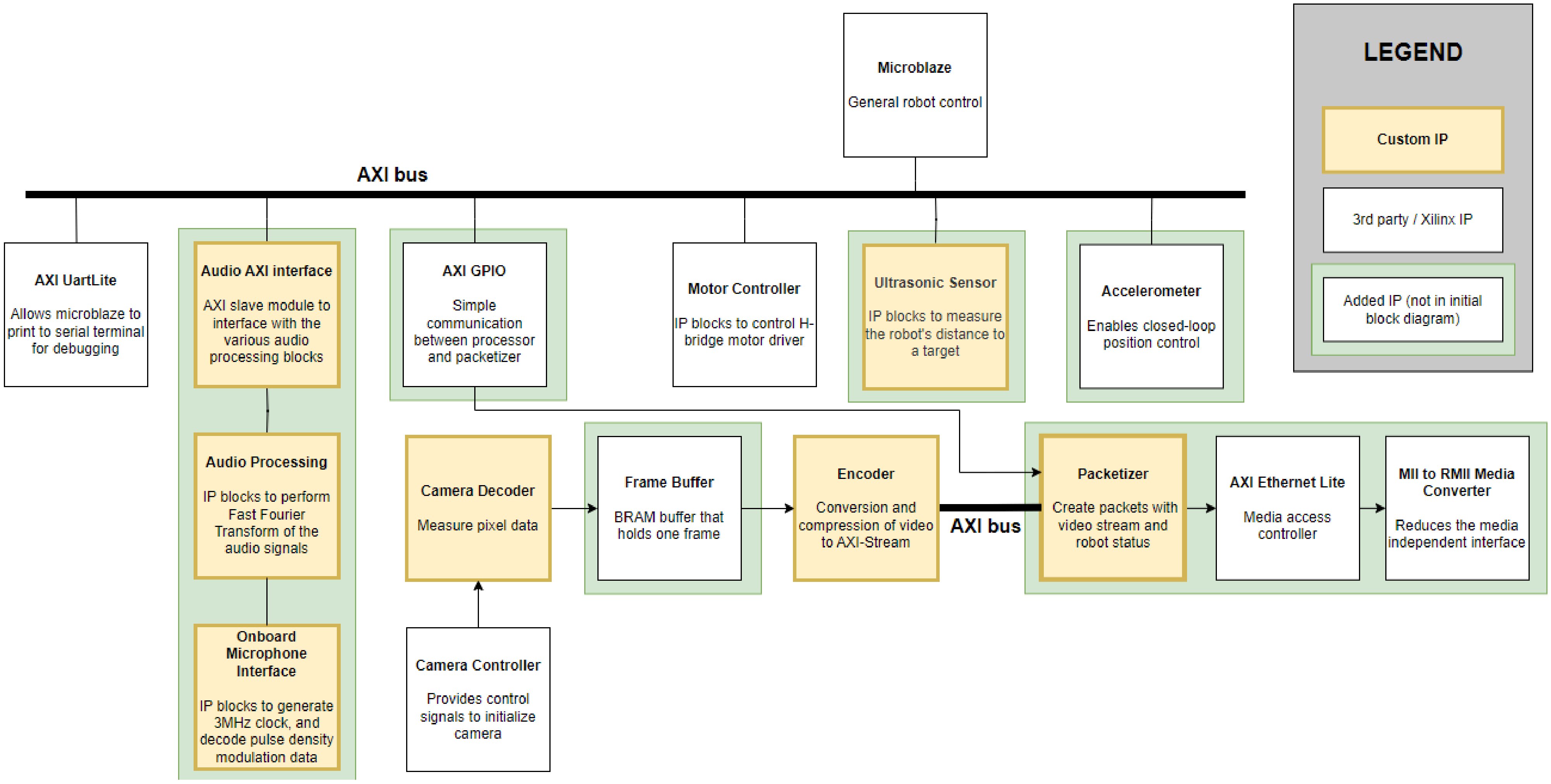

LEGEND
Custom IP
3rd party / Xilinx IP
Added IP (not in initial block diagram)
Microblaze
General robot control
AXI bus
AXI UartLite
Allows microblaze to print to serial terminal for debugging
Audio AXI interface
AXI slave module to interface with the various audio processing blocks
AXI GPIO
Simple communication between processor and packetizer
Motor Controller
IP blocks to control H-bridge motor driver
Ultrasonic Sensor
IP blocks to measure the robot's distance to a target
Accelerometer
Enables closed-loop position control
Audio Processing
IP blocks to perform Fast Fourier Transform of the audio signals
Onboard Microphone Interface
IP blocks to generate 3MHz clock, and decode pulse density modulation data
Camera Decoder
Measure pixel data
Camera Controller
Provides control signals to initialize camera
Frame Buffer
BRAM buffer that holds one frame
Encoder
Conversion and compression of video to AXI-Stream
AXI bus
Packetizer
Create packets with video stream and robot status
AXI Ethernet Lite
Media access controller
MII to RMII Media Converter
Reduces the media independent interface

Figure 1 - FPGA 1 Block Diagram

1.3.2. FPGA 2

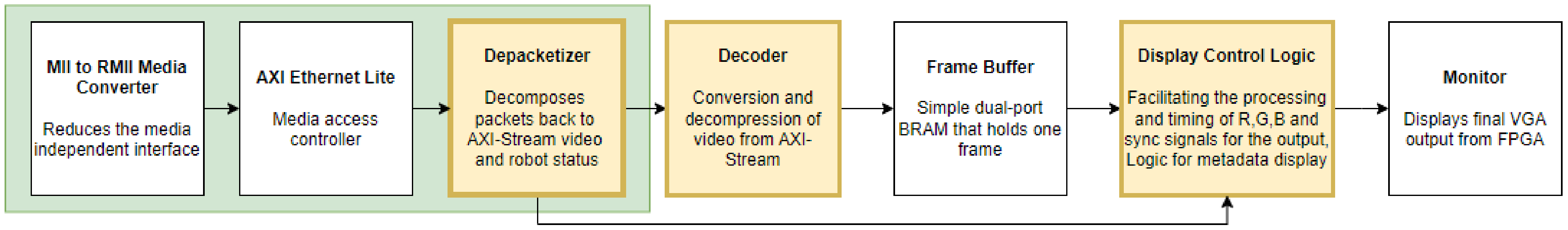

MII to RMII Media Converter
Reduces the media independent interface
AXI Ethernet Lite
Media access controller
Depacketizer
Decomposes packets back to AXI-Stream video and robot status
Decoder
Conversion and decompression of video from AXI-Stream
Frame Buffer
Simple dual-port BRAM that holds one frame
Display Control Logic
Facilitating the processing and timing of R,G,B and sync signals for the output, Logic for metadata display
Monitor
Displays final VGA output from FPGA

Figure 2 - FPGA 2 Block Diagram

1.4. IP	**Description**
Xilinx IP	
Microblaze	Top-level robot control
AXI UartLite	Allows Microblaze to print to serial terminal for debugging
AXI GPIO	Allows Microblaze to communicate with Packetizer
BRAM Generator	Creates memory buffers for video data
Accelerometer Control	Interfaces with the onboard accelerometer
AXI Ethernet Lite	Media access controller for ethernet
MII-RMII Converter	Reduces the media independent interface
Third-Party IP	
Camera Control	Controls OV7670 camera through I2C
Debouncer	Handles camera reset switch debouncing
Camera Capture	Takes video data from OV7670 camera and puts it in BRAM
PMOD HBridge	AXI Slave controlling PMOD HBridge PWM and direction
Custom IP	
Microphone Interface	Generates 3MHz clock and decodes pulse-density modulation data
Audio Processor	Performs Fast Fourier Transform on audio signals
Audio Interface	AXI slave interfacing with audio processing blocks
Ultrasonic Control	AXI slave interfacing with the ultrasonic sensor
Video Encoder/ Decoder	Encodes/decodes video data from/to memory to/from words for transmission
Packetizer/ Depacketizer	AXI-Lite masters interfacing with the AXI Ethernet Lite Constructs/deconstructs packets from/to words and sideband data
VGA Driver	Displays video data from BRAM through VGA - heavily modified from 3rd-Party IP to display status info, so considered custom

2. Outcome

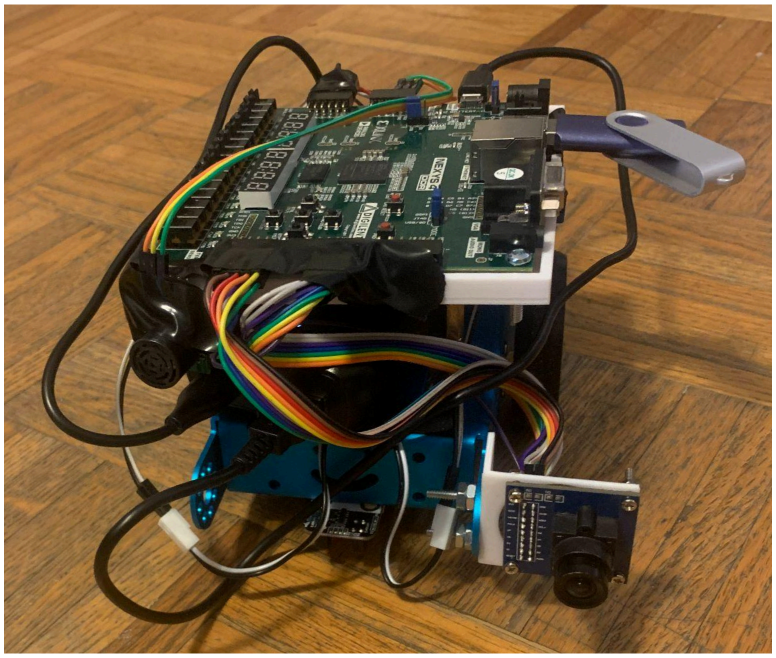


Figure 3 - Robot

2.1. Features

One key difference between the initially proposed system and the final implementation was the “stream” subsystem. Initially, we expected to use a wireless protocol to transmit the video stream back to the base station. However, after discovering the WiFi module was not implementable without the involvement of a Microblaze processor, we changed our system to use a wired ethernet cable. Our goals involved creating most of our design in hardware, so the use of the Microblaze was not desirable.

Other than that, our final system included **more** features than initially proposed. Added features included the ultrasonic sensor and the accelerometer. We were able to add more features since our project timeline ensured that key items were done and integrated early, which allowed new features to be introduced based on observations made during testing.

2.2. Functionality

Overall, the project worked really well. We achieved the key project goal: locate a speaker in a 2-D search space, purely guided by sound. Furthermore, we were able to integrate all of our subsystems, which meant that we could drive, while controlled by sound, stream video back to a base station, and display the live video feed with additional information on a VGA monitor. This achieved all of our goals as mentioned in section 1.2.

However, there were some limitations with the project, and items that could be updated for increased performance. First, there were some issues with the video feed synchronizing between the base station and the robot. It took a few CPU resets before the display synchronized properly. Second, there were large limitations with the hobby motors we used for the robot. They were very unreliable, and the lack of closed-loop control made the robot guidance difficult. We implemented a PID controller which used the onboard accelerometer to mitigate this issue, but noticed accelerometer drift over time, which often confused the robot. In the future, a gyroscope module might give more precise position control; additionally, better quality motors might give a more predictable response.

2.3. Next Steps

If the team had additional time for this project the system each subsystem could have been improved in multiple ways. For the streaming subsystem a next step would be to route the ethernet packets through an on-board router to be able to send the video captured by the robot wirelessly. In addition to this, at the beginning of the project we created a compression block that was able to compress the video packets in simulation. If we had more time we would like to add this submodule into our design to improve the frame rate. For the robotics subsystem a next step would be to create a proper closed loop controller to be able to directly control the movement of the robot. We attempted this with the onboard accelerometer but the final design needed more tuning and a higher quality sensor. For the audio subsystem a next step would have been to utilize a higher quality microphone to be able to have better data collection as well as to complete a better search for the speaker. For the camera subsystem, a next step would be to have additional meaningful data on the display as well as implement operator controls such as a zoom feature or brightness settings. If someone had to take over the project the team would suggest working on both the closed loop controller as well as the wireless streaming of the video as those are the most crucial next steps for the design.

If we could start over one of the biggest changes would be a larger focus on the closed loop control of the robot. Through our testing we found that the open loop controller was not enough to have full control over the robot. If we knew this from the beginning, we could have requested a gyroscope PMOD that would have been able to help us better control the robot.

3. Project Schedule

The original schedule is shown in section 3.1 below, with the final updated schedule shown in section 3.2. In summary, the key milestones were met, but each subsection was delayed at some point in the project. For the Audio subsystem, there were delays early in the project while the microphones were allocated between the various groups. The Robot subsystem remained on time, which allowed new features to be added while others caught up. The Stream subsystem was initially delayed due to attempting to use the WiFi module; however, the milestones and scope were adjusted dynamically to account for this lost time. Finally, the Video subsystem was generally on-time, with small changes being made to account for the changes to the data stream.

Overall, the milestones and schedule for our project was well-planned. We planned to spend a long time integrating and testing, which was valuable as the semester went on. We were able to make sure that systems were integrated early, and we validated key aspects of the design before moving on to new challenges.

3.1. Original Schedule

	Audio	**Robot**	**Stream**	**Video**
1	Research the DSP slice functionality and create a plan of how to use hardened blocks with custom IP to do real-time FFTs.	Come up with preliminary drawings for a vehicle that is able to mount the FPGA and peripherals.	Research WiFi card, camera, H.262 compression, and DCT on FPGAs.	Show RGB, hsync, vsync signals working with the 640x480 VGA standard.
2	Perform FFT of a hard-coded waveform (benchmark accuracy of algorithm and debug any errors) in Vivado testbench.	Create a model for the vehicle and start printing and assembling the vehicle, as well as start coding the interface with the motor.	Test WiFi module and choose Option A or B. In some way, send a frame.	Display a static frame.
3	Interface with on-board microphone and do real-time FFT with real data. Play a pre-recorded “frequency of interest” and	Have a fully assembled vehicle that is able to move in a straight line for a predefined	Implement DCT, and finalise internet connection.	Display video or sequence of frames through a frame buffer.

	validate that the algorithm measures it.	amount of time		
4	Start testing with audio filtering interfacing with the motor to find and drive towards a speaker.		Complete JPEG compression.	Display output from system video decoder.
5	Finalise testing and creating a vehicle that is able to detect a sound emitting to a speaker and drive towards the speaker.		Complete H.262 compression.	Display text along with video output.
6	Improve audio subsystem using more mics or digital filtering - depends on previous results.	Measure distances with the ultrasonic sensor and stop before crashing.	Complete H.262 decompression.	Incorporate status info in VGA output
7	Prepare for final demo + Buffer week			

3.2. Actual Schedule

	Audio	**Robot**	**Stream**	**Video**
1	Same as original.	Same as original.	Same as original Found that WiFi was unworkable with provided module	Same as original.
2	Same as original.	Same as original.	Tried TEMAC module for internet, did not work Created DCT test in meantime.	Same as original.
3	Real time FFT test moved to next milestone since onboard microphone was different from expected.	Driving test was moved to next milestone.	Continued TEMAC issues, camera issues.	Same as original.
4	Attempted integration of audio	Completed driving test.	Switched to AXI Ethernet Lite,	Started working on colour conversion

	and robot, Vivado issues.		issues remained.	and finished implementation of frame buffer.
5	Integration complete, some work on localization algorithm.	Communication with ultrasonic sensor	Switched to RGB as YUV did not work. Due to this and high bandwidth found with Ethernet Lite, compression was abandoned.	Implemented camera and display system. Created colour conversion blocks, but no longer needed as YUV camera did not work.
6	Continued work on localization algorithm.		Finalized video transmission.	Added status information.
7	Same as original.			

4. Description of the Blocks

4.1. Audio Subsystem Blocks

The purpose of the audio subsystem is to output the spectrum of an audio input signal. The audio input signal comes from the Nexys 4 DDR board's onboard microphone, encoded in pulse density modulation (PDM) format. The spectrum output is accessible over an AXI interface, so that the processor can access the data for use in its search-and-rescue algorithms.

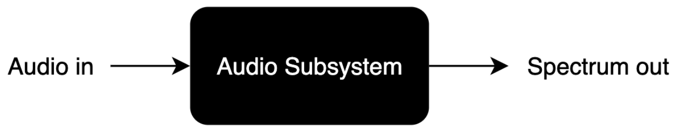


Figure 4 - Black-box diagram of system

This more detailed block diagram shows the major components in the Audio subsystem.

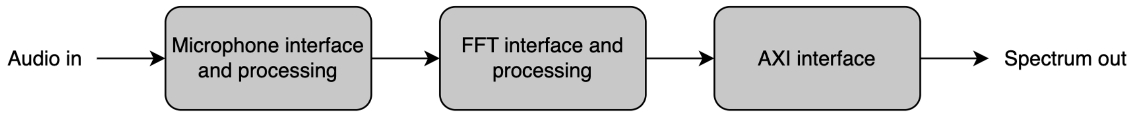

Audio in
Microphone interface and processing
FFT interface and processing
AXI interface
Spectrum out

Figure 5 - Three main IP blocks

4.1.1. Microphone Interface and Processing IP Blocks

The IP block diagram for the microphone interface and processing are found in the figure below. All blocks are custom implementations, written in Verilog.

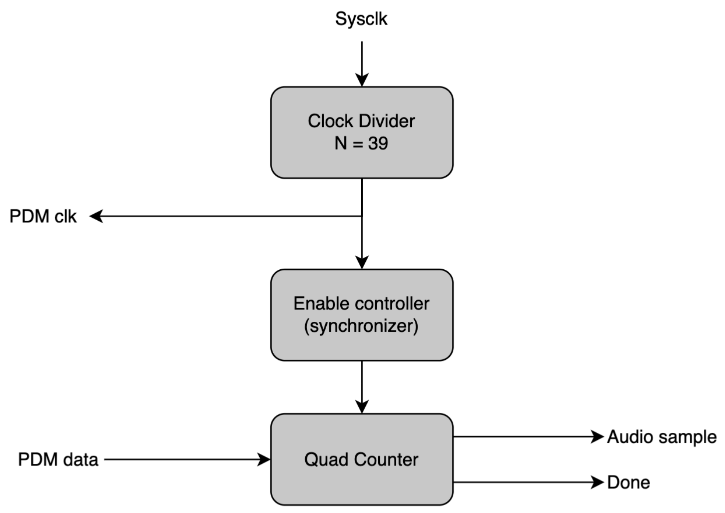


Figure 6 - IP blocks for microphone interface and processing

The onboard MEMS microphone requires several custom Verilog interface modules. First, the microphone requires a clock input in the range of 1 - 3.3 MHz. This is much lower than the system clock (100 MHz), so a module was created to generate an acceptable clock for the microphone. Two options would be a simple clock divider (using a counter and flip-flops), or a phase-locked loop (PLL). We chose a clock divider, due to its simplicity, and the low output speed requirements. We divide the 100 MHz system clock by 39 to generate a 2.564 MHz clock for the microphone. This specific frequency was chosen because it lies reasonably in the required range; with no fixed constraints, it is sensible to not target the microphone's maximum possible speeds. The clock division is done in the **Clock Divider** block.

The MEMS microphone outputs data in Pulse Density Modulation (PDM) encoding format. To convert this data into regular integer format, several IP blocks were created. First, four counters (found in the **Quad Counters** block) sample the incoming data. These counters need to be precisely interleaved in time to achieve the sampling frequency necessary for our project. The counters are controlled by the **Enable controller** IP block, which enables each counter at the right moment in time after a system reset.

4.1.2. FFT Interface and Processing IP Blocks

The blocks for the FFT interface and processing are shown in the figure below. Most blocks are custom, except for the Xilinx FFT IP core.

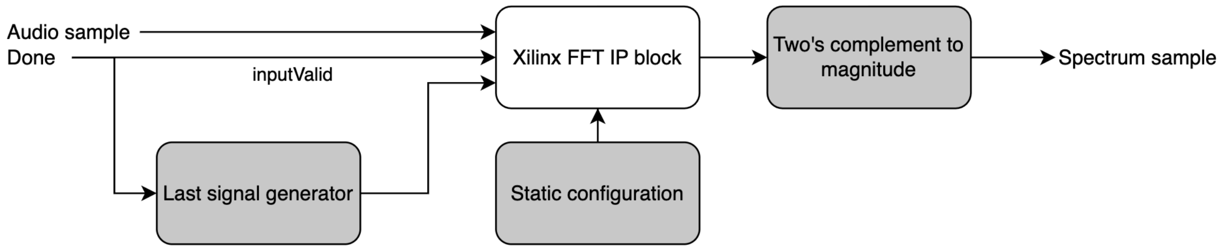


Figure 7 - FFT interface and processing IP blocks

Audio samples from the previous blocks are streamed into the Xilinx FFT core. The IP core is configured over an AXI-like interface, which is done in the **Static configuration** block. This block generates the data for the configuration, as well as performing the handshaking to confirm that the core is set up. The FFT core needs an indication of when the "last" sample in the frame has been received, so a custom IP block generates this signal using a counter. Lastly, the IP core outputs its data in two's complement format, however we are really only interested in the magnitude of the number to perform our audio localization. So, the streamed output passes through a custom block which converts from two's complement to magnitude, before entering the next IP blocks.

A constraint for simulation is needed to make the Xilinx FFT block work properly. In the functional simulation, if the data changes exactly with the clock edge, the Xilinx FFT block breaks. So, a constraint is needed to slightly delay the clock to the FFT core during simulation. This issue is not present in the timing simulation since the data changes .

4.1.3. AXI Interface IP Blocks

The blocks for the AXI interface are found below.

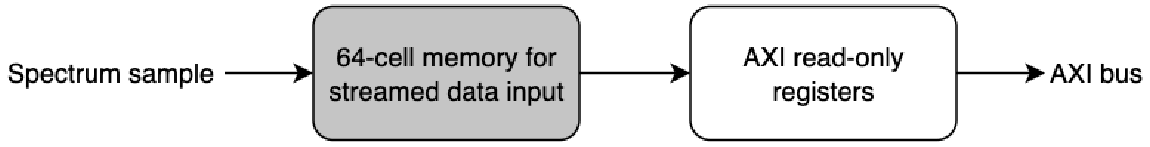


Figure 8 - AXI interface block diagram

The spectrum samples are streamed out of the FFT core and through the two's complement to magnitude conversion block. A custom block takes in the streamed audio samples, and once an entire frame is received, stores it in memory. The frame is stored un-interrupted by the next frame's streamed samples, until it is replaced in a single clock cycle upon the next frame's completion. An autogenerated AXI interface retrieves specific samples from memory upon receiving a READ command from the master (MicroBlaze).

4.2. Robotic Subsystem Blocks

4.2.1. H-Bridge Interface

The H-bridge interface was third party IP that was utilized in this project. It worked by sending a PWM and direction signals to the bridge modules that were able to control the speed and direction of each DC motor on the robot. To interface with the module, software was developed to send AXI messages to control the speed, direction and enable of the motor. The control of the motor was done with an open loop control that would turn off and on the motor after specific time intervals.

4.2.2. Ultrasonic Sensor IP Block

The ultrasonic IP block was custom made and interfaced with the sensor through PWM signals. It constantly requested distance data from the ultrasonic sensor and sent the data into a register that could be read by the AXI master. The high level block diagram can be seen below.

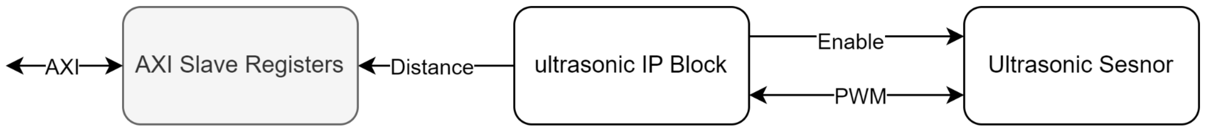


Figure 9 - Ultrasonic IP Block Diagram

The ultrasonic sensor could be accessed using multiple different techniques. The team chose to utilize the PWM communication mode as it was easier than communication over analog signals or UART. The system worked by setting the start stop signal to high which would initiate the ultrasonic sensor to send a pulse to find the closest object. As the PWM signal was set high until the pulse was detected by the ultrasonic sensor, a counter was utilized to count how many clock cycles the PWM signal was set high. Once the PWM signal was set back to low the distance was measured and inputted into the AXI register. This AXI register was read-only to the master and was able to be read at any time by the microblaze. An example of the distance measurement sequence can be seen below.

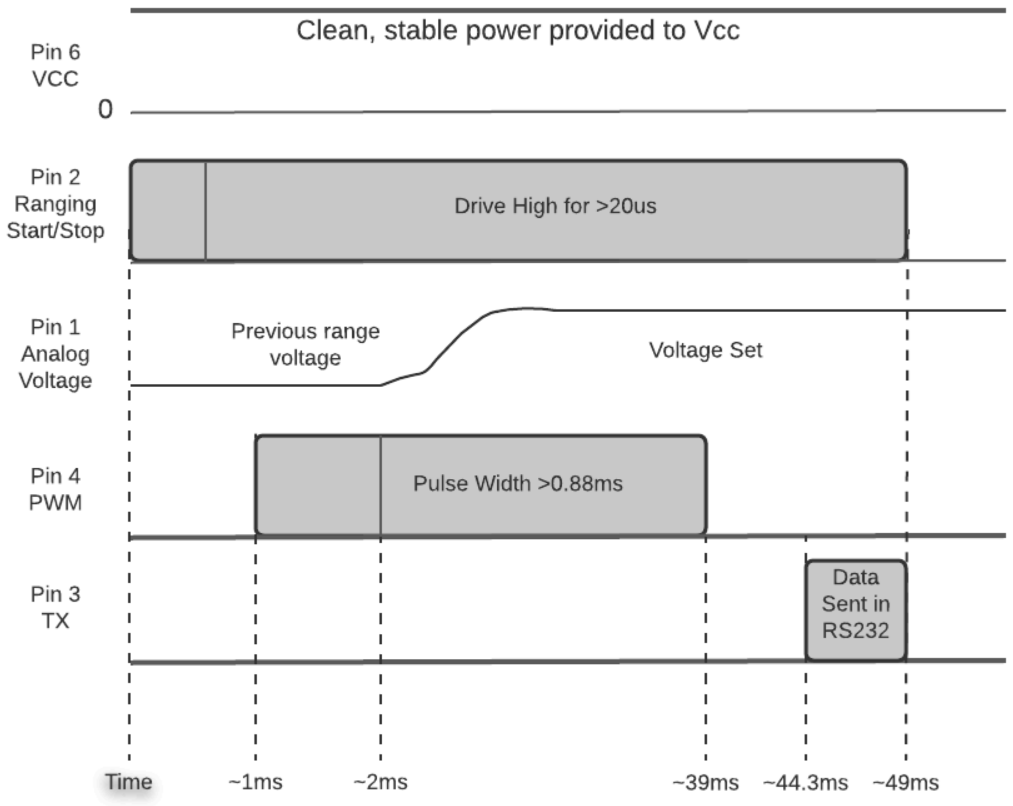


Figure 10 - Ultrasonic signals

4.2.3. Accelerometer Interface

The accelerometer was interfaced through third party IP that communicated with the on-board accelerometer through SPI messages. The block outputs the x, y axis acceleration of the board. To interface with the block we utilized AXI GPIO blocks to communicate with the microblaze. This allowed the system to have live acceleration data.

4.3. Microblaze Development

For the microblaze development the team created two key functions that were able to search for a speaker in a 2-D space as well as a closed loop controller to have better control over the robot.

4.3.1. Searching Algorithm

For the search algorithm the team created a system where the robot would take four audio samples to determine the quadrant the speaker was located in. The full sequence to find the speaker can be seen in Appendix A.

Once the quadrant had been determined the team developed a “sweep” that utilized the ultrasonic sensor to sweep an area to search for the speaker. This acted similar to a radar device that was able to scan a section of the quadrant for the speaker.

4.3.2. Closed Loop Controller

Additionally, a PI controller was created in the Microblaze to overcome the limitations that were found controlling the robot with an open loop controller. The PI controller attempted to control the robot to have no velocity in the y direction so the robot was able to drive in a straight line. This controller was not fully tuned but a barebones version of the controller was developed and tested. Appendix B shows a full diagram of the system.

4.3.3. Additional Functionality

The Microblaze code had additional functionality to be able to have better control over the robot. Utilizing the ultrasonic sensor the team tested and found a distance threshold that the robot would have to turn off its motor so it would not run into objects. This is utilized in all of the different modes and the microblaze would turn off the motors if it detected an object in front of it.

Additionally the project utilized the push buttons and LEDs to initialize tests as well as indicate what test mode the robot was in. This was all done by sending AXI messages to AXI GPIO that was connected to the LEDs and push buttons.

4.4. Streaming Blocks

The streaming blocks form a datapath between the video memories on both FPGAs. The following block diagram displays their layout.

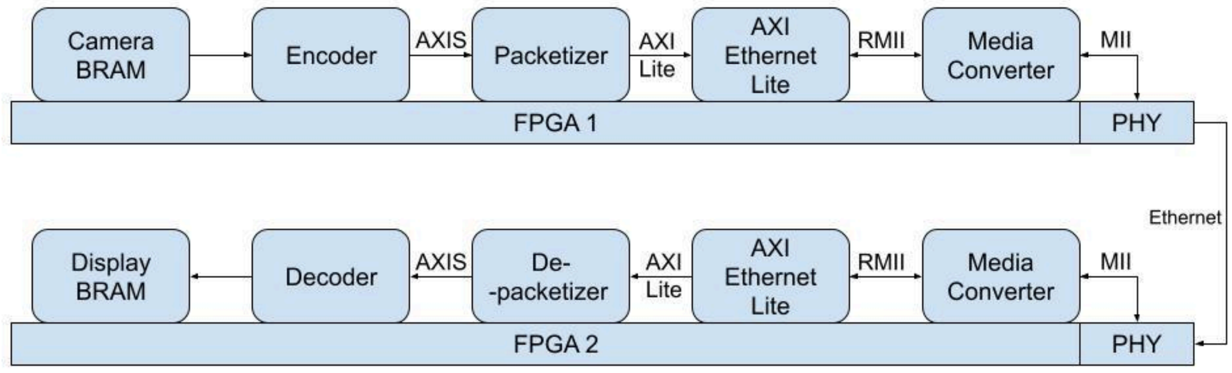


Figure 11 - Streaming Modules

4.4.1. Encoder and Decoder

The encoder is a custom module that takes sets of two 12-bit pixels (RGB444) from the BRAM and encodes them into a 32-wide AXI Stream (with some padding). The decoder does the opposite. In addition to the default AXI Stream signals, tlast is used to signal the end of a row, and tuser is used to signal the start of a frame. When integrating the design into the robot design, we decided to combine the encoder and decoder with the packetizer and depacketizer respectively for ease of use and updating. Within the module, there is simply an internal AXI Steam instead, and the logic for tlast and tuser was substituted in.

4.4.2. Packetizer and Depacketizer

The packetizer and depacketizer are both custom AXI-Lite Master modules designed specifically to interface with the AXI Ethernet Lite. The packetizer takes the video data from the AXI Stream and positions it to be sent using the AXI Ethernet Lite. After building the frame, it indicates to the AXI Ethernet Lite that the packet should be sent and then waits for the Ethernet Lite to indicate that another packet can be loaded. Research shows that this is a unique implementation compared to the more typical LWIP+Microblaze. The depacketizer waits for received packets and takes them apart back into words to be sent over the internal AXI Stream. Both these modules keep track of line numbers to help with synchronization. Together, they have the same effect as the AXI Stream configuration of the TEMAC.

4.4.3. AXI Ethernet Lite, Media Converter, Onboard PHY

The AXI Ethernet Lite and Media Converter are Xilinx modules. The AXI Ethernet Lite is an AXI Lite slave that serves as an Ethernet MAC. The Media Converter reduces the media interface to be compatible with the PHY onboard the Nexys 4 DDR.

4.5. Camera and Display Blocks

The first purpose of these blocks is to capture pixel data from the OV7670 Camera PMOD on the robot board and store them in BRAM. Secondly, they generate the VGA display signals using the output of the streaming subsystem on the base station board. A 640x480 resolution display is generated by double-double scanning the stored pixels. The third party blocks were sourced from the following GitHub repository:
https://github.com/laurivosandi/hdl/tree/master

4.5.1. Camera Capture

This block was third party sourced and was the interface between the camera and the block RAM for storing the frame pixel data. It receives the signals from the pclk, vsync, href, and data pins of the camera as input and generates the address, pixel, and write enable signals for the BRAM block. The addresses were incremented by one each clock cycle that the vsync signal was not high and the output pixel data is generated using a shift register d_latch where the MSB are extracted for each colour signal component.

4.5.2. Camera Controller

This third party block sent the camera configuration registers to the camera using an I2C-like bus. It differs from I2C as it implements a simplified state machine, fixed timing logic controlled by the divider signal, and a non-standardized acknowledgment signal.

4.5.3. Debouncer

This third party block used user input to restart camera initialization by sending a signal to the resend port of the Camera Controller module.

4.5.4. Block Memory Generators

Two identical Xilinx Block Memory Generator IPs were utilized with the Simple Dual Port configuration in the Standard Alone Mode with a 12-bit port width and 26144 Port Depth (18-bit addressing). The first one functioned as the interface between the camera and the streaming subsystems and the second as the interface between the streaming and the display subsystems.

4.5.5. VGA Driver

The base of this block was third party sourced and was customized to adjust to our project's RGB signal configuration and to add logic for the text display output. The Nexys 4 DDR board uses 4 bits for each of the RGB output signals to the VGA port and the original code used a RGB565 configuration. The code was adapted to RGB444 by removing the zero-padding, least-significant bits from the vga_red, vga_green, and vga_blue signals and adjusting the size of these output signals accordingly.

New input logic signals for the ultrasonic sensor and frequency amplitude values were added to the original code. Depending on the values of these signals, the text displaying logic would print "SEARCH" to the screen if the ultrasonic sensor threshold was not reached and "FOUND" if it was. The amplitude of the frequency of interest was displayed on a sliding scale of initially "0" symbols then replaced by "1" symbols as the amplitude increased. The value of these logic signals determine which pixel-setting logic was used to achieve the text display itself. This logic utilizes the horizontal and vertical pixel counters when setting specific pixel values. Constants for text sizing and on-screen positioning were defined for easy adjustments and for use in the pixel setting logic blocks.

This block additionally generates the remaining VGA output signals, hsync and vsync.

5. Description of the Design Tree

The final block diagrams of the robot and base station systems are shown in the figures below.

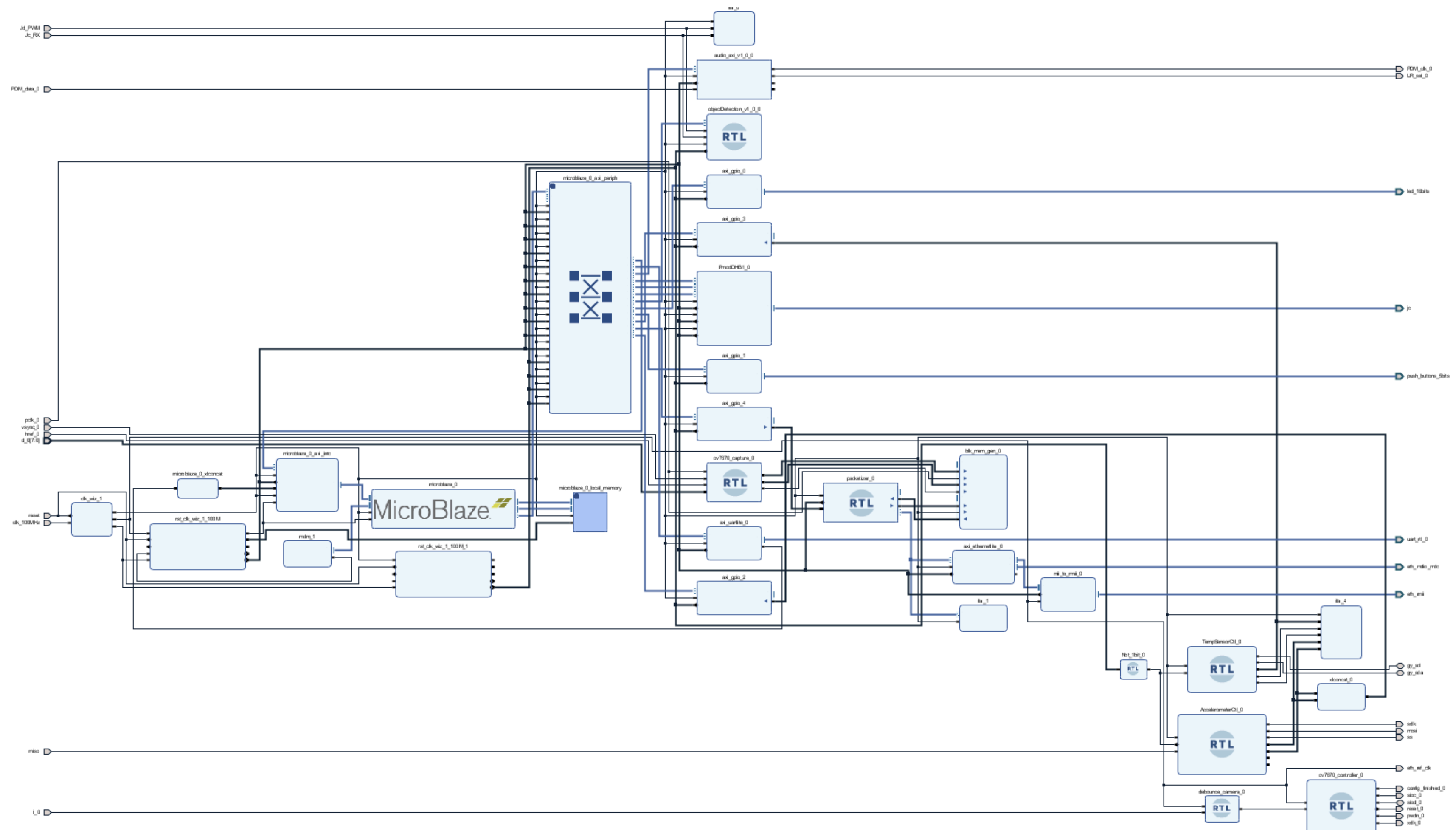


Figure 12 - Robot system

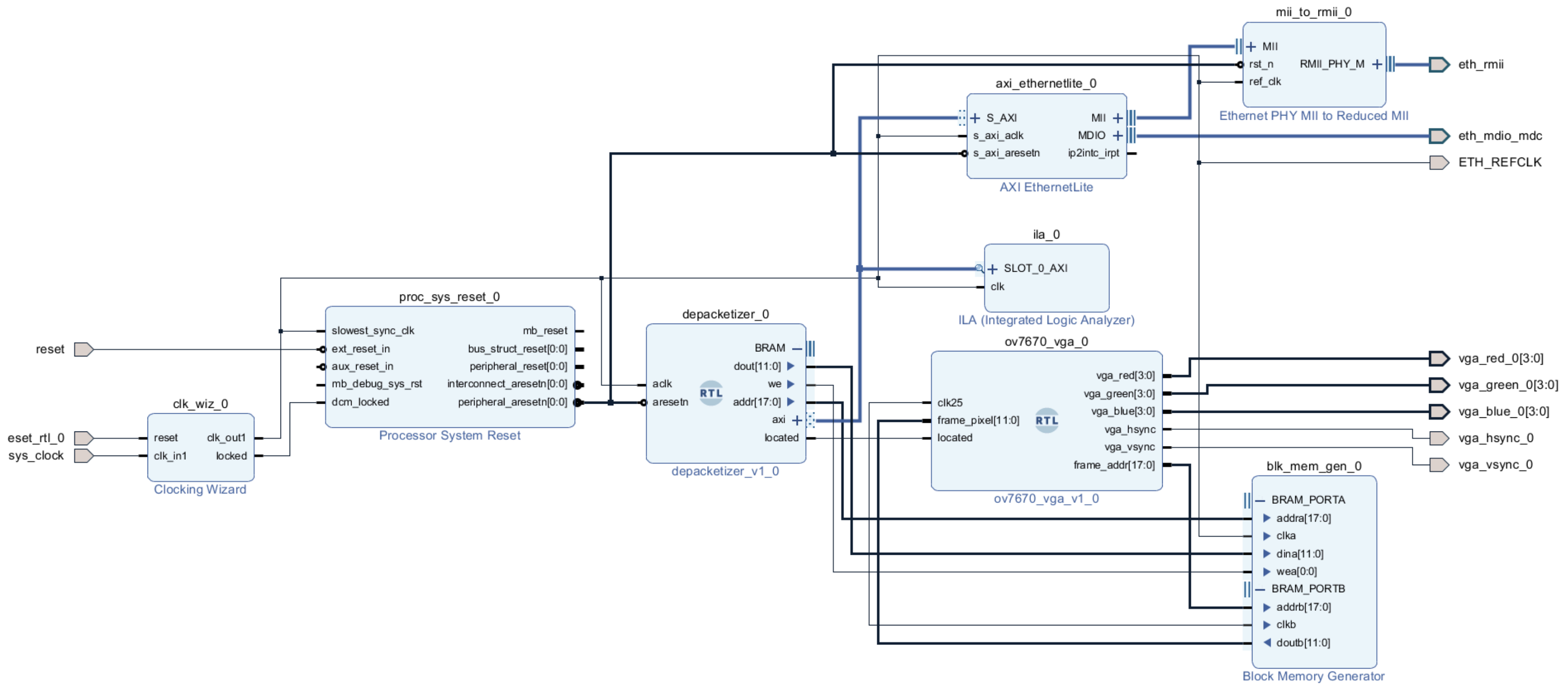


Figure 13 - Base station system

Both projects are uploaded to Github. The link to access our repository is https://github.com/ece532-ProjectGroup4/ECE532. The github repository contains all the source code, with information as to how to access each component.

6. Tips and Tricks

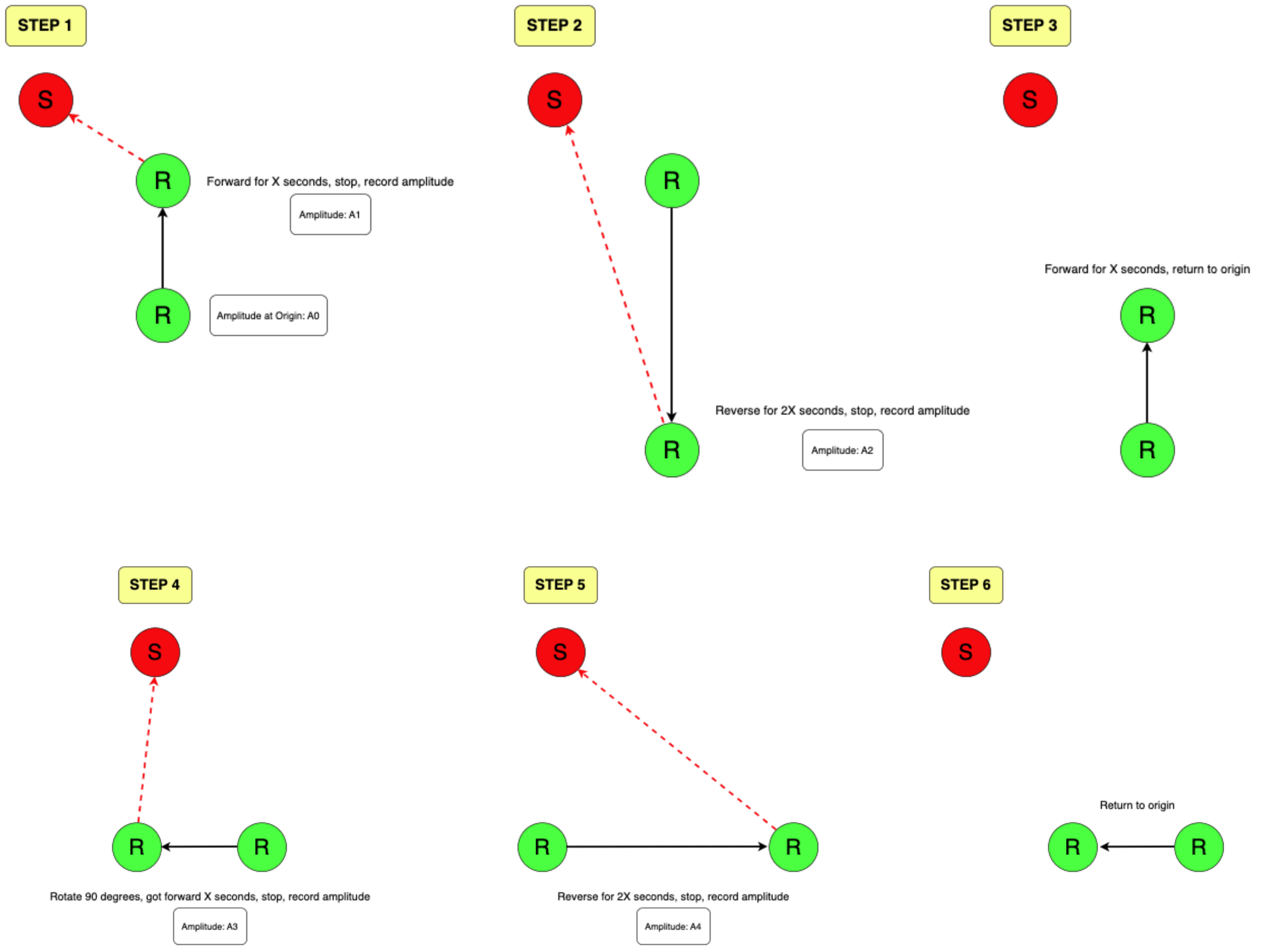
Below are some tips and tricks the team found useful in their completion of this project:

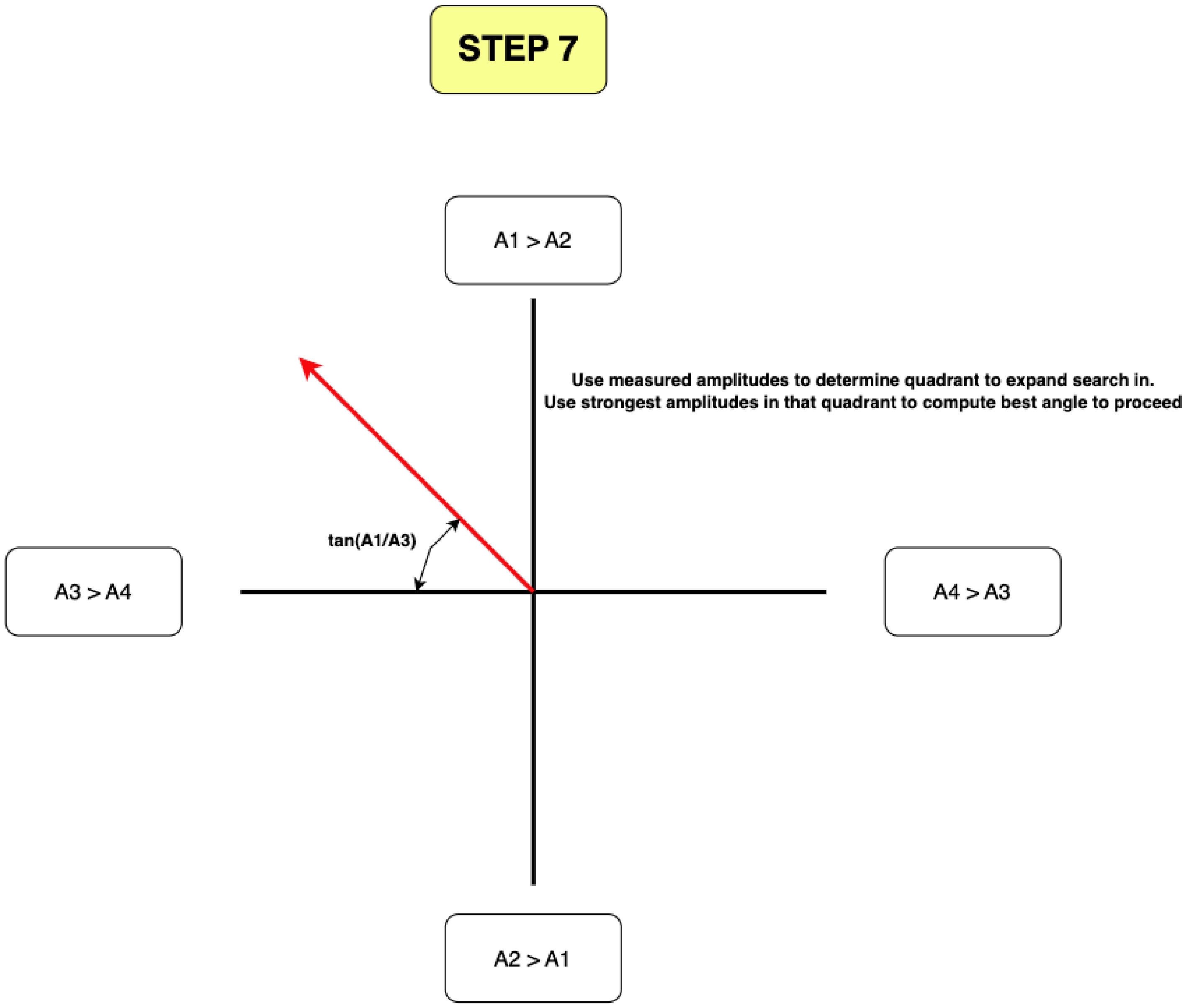
- Include the ILA in the block diagram early, and connect it to as many signals as possible. When issues arose in testing, it saved a lot of time by already having the ILA in the block diagram - we did not need to re-generate the bitstream to debug our circuits
- Similarly, including the LEDs, switches, and pushbuttons in the block diagram allowed us to easily display relevant data/signals/states on the FPGA. This was a very simple way to check key signals, and these outputs were software-defined by the MicroBlaze, which meant that we could easily change the functions based on what we were debugging.
- We found that when utilizing the on board accelerometer there was a slight drift that had to be accounted for when measuring the acceleration of the board. To remedy this we would take a measurement form the board when there was no motion to use as an offset when measuring the acceleration.
- Because of non-idealities in the motors the team utilized, we found that it is important to create a closed loop controller if a team wants full control over a driving robot.

7. Video

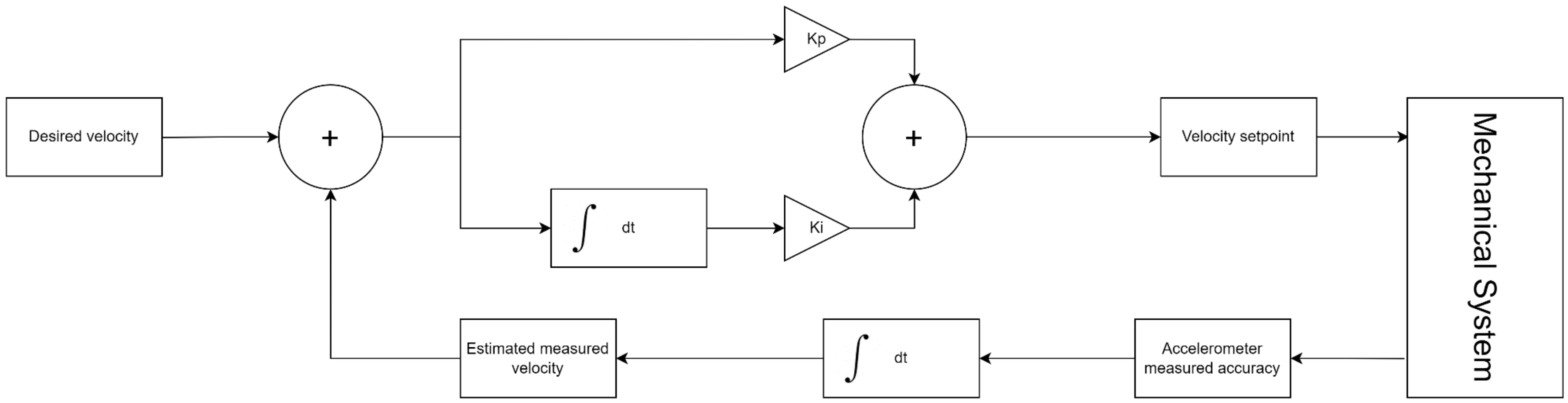
Our team did not create a single summarizing video, however, in the “videos” folder in our git repository, you can find videos of the robot successfully driving.

Appendix A


STEP 1
S
R
Forward for X seconds, stop, record amplitude
Amplitude: A1
R
Amplitude at Origin: A0
STEP 2
S
R
Reverse for 2X seconds, stop, record amplitude
Amplitude: A2
R
STEP 3
S
Forward for X seconds, return to origin
R
R
STEP 4
S
R
R
Rotate 90 degrees, got forward X seconds, stop, record amplitude
Amplitude: A3
STEP 5
S
R
R
Reverse for 2X seconds, stop, record amplitude
Amplitude: A4
STEP 6
S
Return to origin
R
R


STEP 7
A1 > A2
Use measured amplitudes to determine quadrant to expand search in.
Use strongest amplitudes in that quadrant to compute best angle to proceed
tan(A1/A3)
A3 > A4
A4 > A3
A2 > A1

Appendix B


Desired velocity
+
Kp
+
Velocity setpoint
Mechanical System
∫ dt
Ki
Estimated measured velocity
∫ dt
Accelerometer measured accuracy