Name: John Huang, Tommy Jablonski
TA: Zach Knot

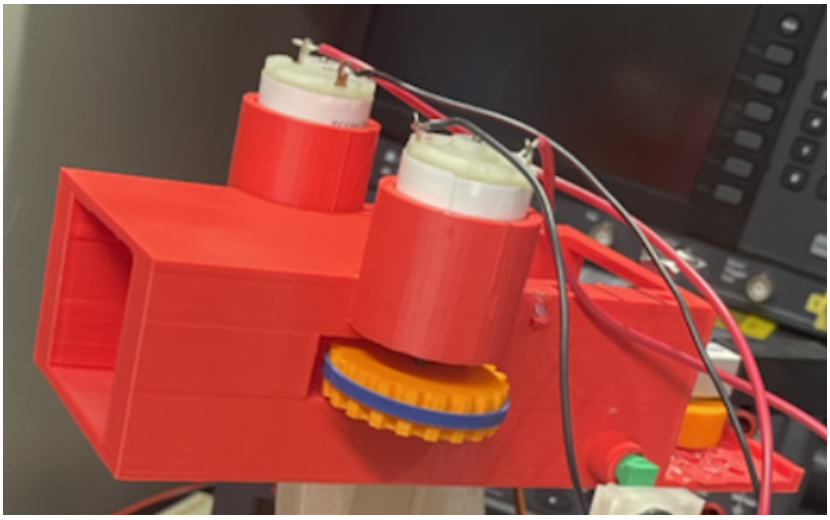
ECE 350 Technical Report: Ping-Pong Ball Turret

Project Design & Specifications

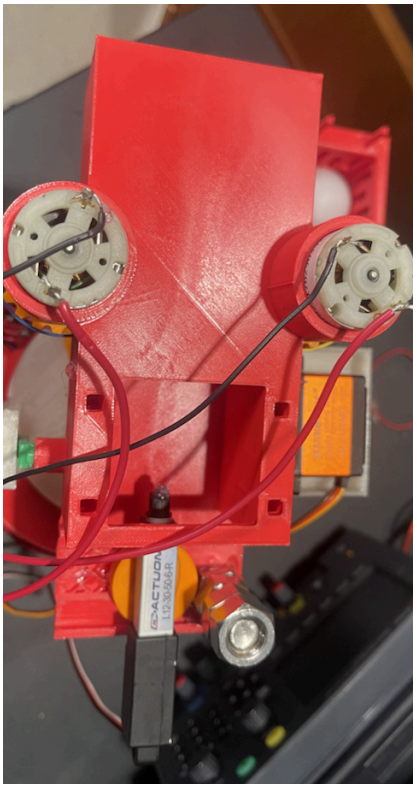
The project the team created was a bi-directional ping pong ball turret. The turret can pan horizontally 180, and pivot vertically 60 degrees in either direction. It can fire a Ping-Pong ball from the chamber which will travel approximately ten feet. The turret design is split into four parts: turret body, rotating base, stand, and turret controller.

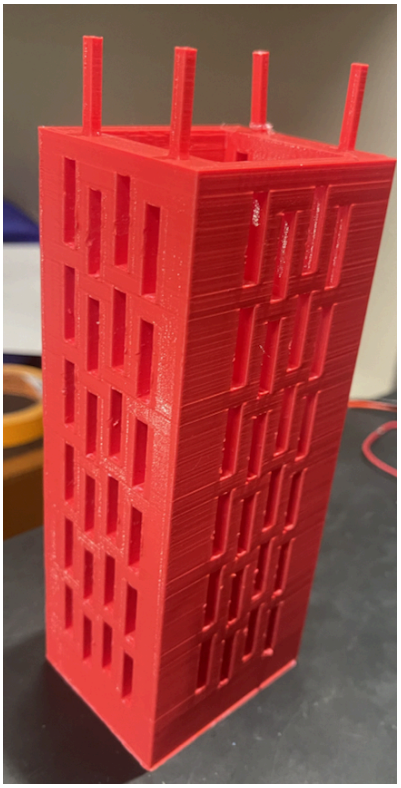
I. Turret Body

The turret body contains the mechanism to fire the Ping-Pong balls. The length of the body is approximately four Ping-Pong balls long, and the width of the body is one Ping-Pong ball wide. The body contains two evenly-spaced hollow circular extrusions for holding two DC motors. A gear is friction fit to the shaft of each DC motor and extends into the turret body itself. This is pictured in the figure below.

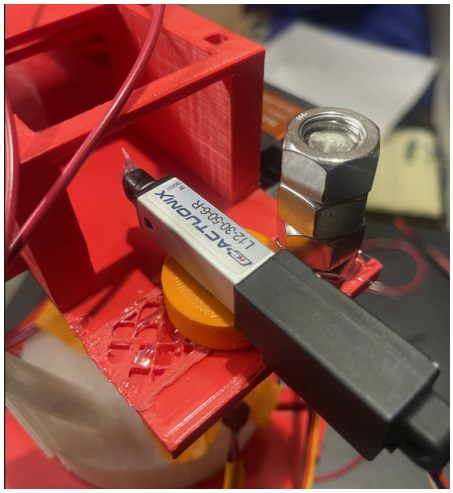


The DC motor housings are spaced such that the edge of each gear is spaced forty millimeters apart (the diameter of a Ping-Pong ball) inside of the housing. As a ball rolls down the chamber, each motor spins counterclockwise, thus turning the gears and expelling a ball when it comes in contact with the gears. At the base of the housing, there is a forty by 40-millimeter cutout for dropping balls into the turret body. Surrounding the perimeter of this cutout, there are four small square holes evenly spaced. These holes fit into four legs of a square magazine, which contains a forty by 40-millimeter cutout for holding Ping-Pong balls, and is the height of four Ping-Pong balls. When operating the turret, the magazine snaps into place using the legs and the cutout on the body itself and feeds Ping-Pong balls into the turret body via gravity. Both the cutout and the magazine are pictured below.

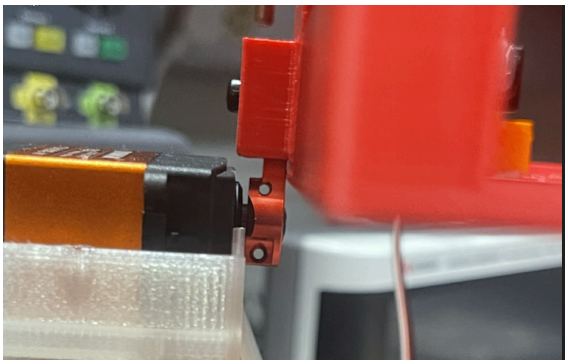


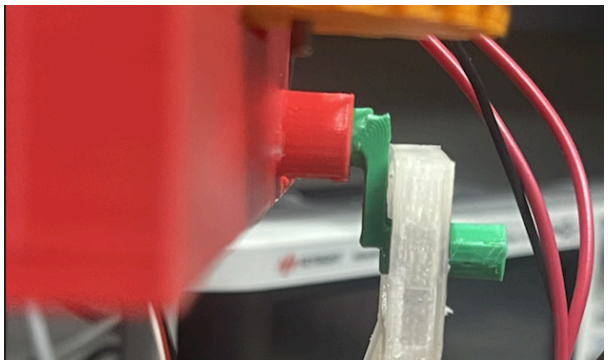


At the bottom of the base, there is a 40-millimeter-long extrusion that holds a disk to raise and mount the linear actuator using hot glue. As a signal to fire the ball is received by the FPGA, the linear actuator fully extends, to about thirty-five millimeters, and pushes a Ping-Pong ball down the chamber of the body. It is dimensioned such that the arm of the linear actuator pushes the ball to exactly the center of the gear. This ensures that in cases where the turret body is vertical, the ball will reach the firing mechanism despite the resistance from the force of gravity. The linear actuator and its support are pictured below.



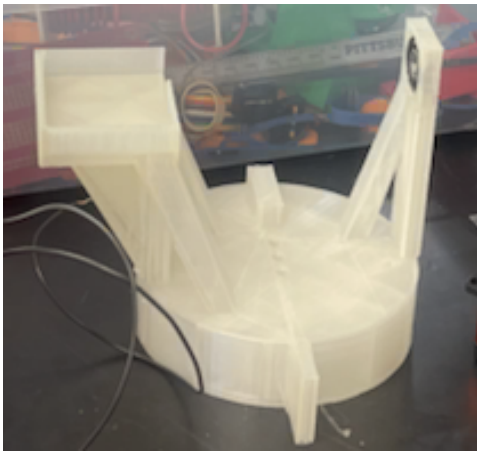
A thin rectangular bar runs through the turret body two millimeters from the ceiling. This bar holds the ball in the chamber in cases where the turret is pointed downwards to prevent premature firing. On the left side of the housing, there is a square extrusion that contains screw holes to hold an arm that is attached to a servo motor. Because of this attachment, while the servo motor rotates its arm, the turret will turn with it vertically. On the right side of the housing, there is a hollow circular extrusion to hold an arm that will connect to a bearing. This is intended to decrease the force on the servo motor from holding up the turret body. The cut-outs are pictured below.

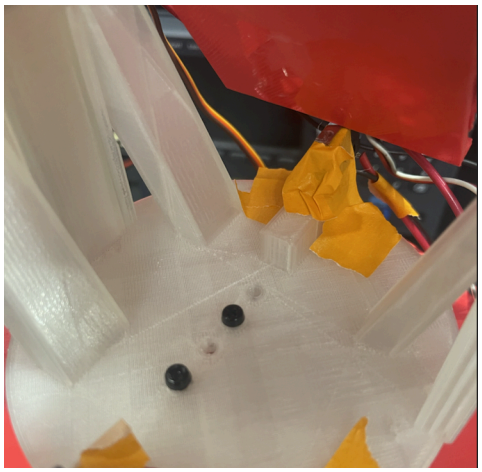




II. Rotating Base

The rotating base is a circular plate 100 millimeters in diameter. On the right side of the plate, there is a support to hold a servo motor, which is hot glued in place. The arm of the motor extends into the turret body, which is used to rotate the turret vertically. On the left side of the plate, there is a square extrusion with a cutout to hold a bearing by friction. An arm runs from the turret body through the center of the bearing in order to relieve the weight on the servomotor. The arm is designed such that the axis of rotation of the servo motor lines up with the center of the bearing. At the center of the base, there are holes that are used to screw the lever arm of a lower servo motor in place. The holes ensure that the center of rotation is about the center of the plate. Thus, when the arm of the lower servo motor rotates, the plate rotates with it, in the horizontal direction. The rotating base is pictured below.

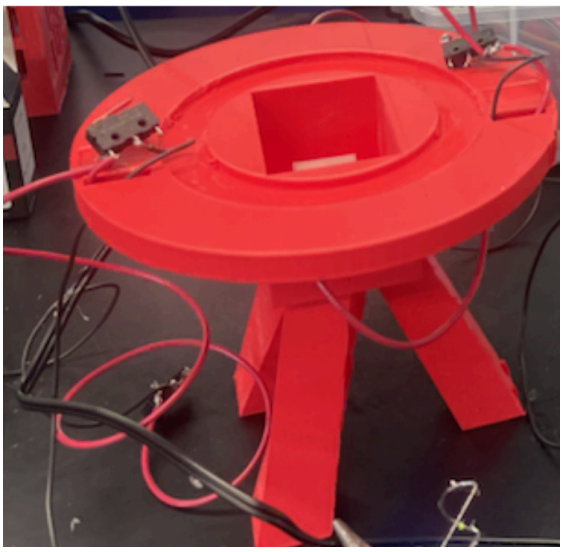


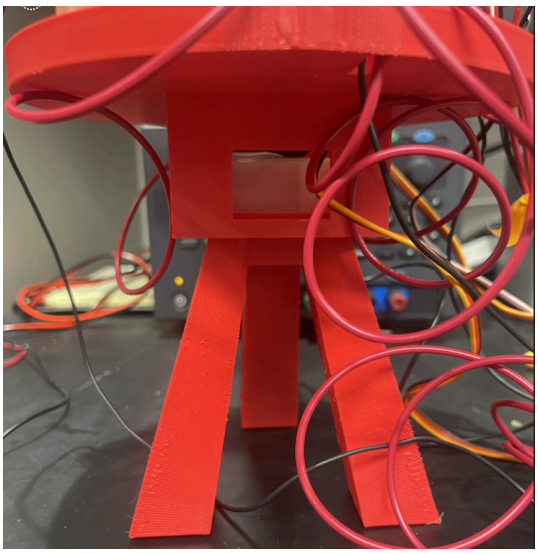


Also on the rotating base, there is a vertically-oriented limit switch on the front and back of the base. As the turret body rotates vertically, it will hit the limit switch, in which our FPGA does not allow any more power to be driven to the servo to turn in that direction. This helps ensure that the turret body does not hit the front or back side of the base as it rotates vertically. On the front of the rotating base, there is a square extrusion. This extrusion is intended to hit limit switches on the stand as it turns horizontally, thus limiting the range of motion in the horizontal direction to 180 degrees.

III. Stand

The stand is a 140-millimeter circular plate that holds on the rotating base. The stand contains a square cut-out at the center to hold the servo which turns the rotating plate. There is also a circular groove to hold a bearing on which the rotating plate rests. This bearing is intended to take the stress off of the arm of the servo. On the left and right sides of the stand, there is a one-millimeter square groove to hold the horizontal limit switches. The switches function the same as the vertical limit switches but limit motion in the horizontal direction. Three square legs extend from the bottom of the stand. The stand is pictured below.

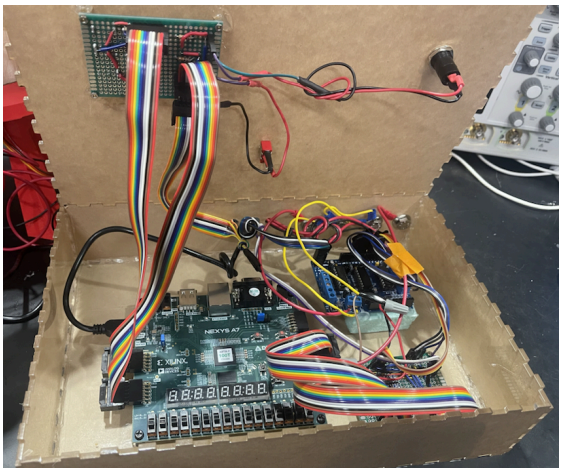




IV. Controller

The controller is a laser-cut box with buttons for movement to the left, right, up, and down directions, a fire button, and a switch to enable fully automatic firing. The fire button, when held down, extends the linear actuator to push the ball down the chamber and spins the DC motors to fire the ball. The up and down buttons power the servo motor in the stand to spin either clockwise or counterclockwise. The right and left buttons power the servo motor in the rotating base to spin either clockwise or counterclockwise. Inside the controller, the FPGA is connected

to a PCB on the top of the controller which routes the inputs into the JC or JD Pmod headers, as described in the Inputs and Outputs section. The PWM signals from the JA Pmod header are routed into a level converter to output 5V PWM signals. All of the digital outputs from the JB Pmod header are wired into pins to be wired directly into the motor controller. The controller, and the connections, are pictured below.



Inputs

There are a total of 10 inputs into the FPGA. All of the inputs are digital since they all connect to an individual switch. Four of the inputs are for controlling the movement of the turret and are connected to the JC Pmod header. Another four of the inputs are for the limit switches placed on the turret and the last two inputs are for the single and auto-firing inputs on the controller. These six inputs are connected to the JD Pmod header. The wiring of these inputs is discussed in the Circuit Diagrams section. Since these inputs are all individual switches they can be read as digital inputs.

The digital inputs from the JC and JD Pmod headers are handled by storing them in registers using the custom instruction LDI (Load Digital Input). The details of the custom instructions are discussed in the Changes to Processor section. After the inputs are stored in registers they can easily be manipulated and read using assembly code.

Outputs

A total of eight digital outputs are emitted from the FPGA. Four of these outputs are PWM signals to control the two continuous servo motors, two DC motors, and one linear actuator. Only four PWM signals are required since the two DC motors use one PWM signal because they must be in sync to properly fire the Ping-Pong ball. The other four signals are digital logic output signals that are used to initialize the motor controller being used to power the two DC motors.

I. PWM Signals

The PWM signals are generated using behavioral Verilog modules. This module first clocks the 10 MHz system clock down to ~1.2 KHz using a 13-bit counter. This slow clock is then used to increment an 8-bit counter whose value is compared to an inputted duty-cycle value. When the 8-bit counter has a lower value than the 8-bit duty-cycle value then the outputted PWM signal is high and is low vice-versa. This ultimately creates a 50 Hz PWM signal whose duty cycle is controlled by an 8-bit binary number. This module is sufficient for controlling the servo motors and DC motors because they don't require PWM signals whose duty cycles change during operation.

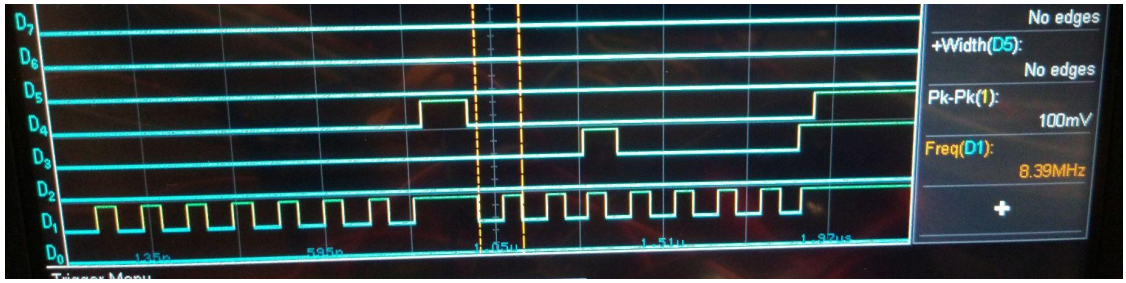
Another behavioral module was created to handle the timing of the linear actuator which, for auto-fire, required a changing PWM signal to extend and contract the arm at regular intervals without any of the physical inputs having changed. This module simply creates a clock with a ~13-second period and 50% duty cycle from the system clock to toggle an 8-bit wire between the decimal values 12 and 24. This is fed into an instance of the PWM module as the duty cycle which outputs a 50 Hz PWM signal that toggles between a pulse width of 1 ms and 2 ms. These are the pulse width's required by the linear actuator to go between full extension and retraction.

All of these PWM signals are controlled via memory-mapped I/O. The RAM addresses of 1000 to 1003 (inclusive) are special 32-bit registers separate from the normal memory (allows for Vivado not to generate the entirety of the RAM) and are wired as the inputs to three PWM modules and one of the linear actuator controller modules. The outputs of those modules are then wired to the first four pins of the JA Pmod header.

II. Digital Logic Signals

The other four outputs are digital logic signals that are used to initialize the motor controller which uses an SN74HC595N (3-state, 8-bit shift register) to control the two H-bridges on the board. By reading the signals outputted by an Arduino Uno R3 (which the motor controller was originally made for), and the Arduino library used to control the motor controller, a total of four signals were deemed to be required for initialization.

The four signals are data, enable, clock, and latch signals. Each of the 8 motor outputs available on the motor controller has a number between 0 and 7 (inclusive). The 8 ports are joined in pairs and numbered from 1 through 4. Within each pair, the ports are labeled either A or B. In order for motor port 3A, which has an associated value of 5, to be turned on, an 8-bit, one-hot encoding is created where the 6th-bit position contains a 1. This encoding can be joined with any other ports that want to be turned on. If motor port 4A (which has a value of 0), should also be turned on, then the 1st-bit position should contain a 1. This results in the following 8-bit number 00100001. To send this motor controller, the enable and latch signals are turned off, and the clock signal is toggled such that there are 8 positive edges. On each positive edge, one bit of the 8-bit encoding is read by the motor controller in reverse order. After all, values are read, the latch signal is turned on, but the enable signal is kept off. The described signal can be seen in the figure below.



Initialization Signals: D1: Clock, D2: Enable, D3: Data, D4: Latch

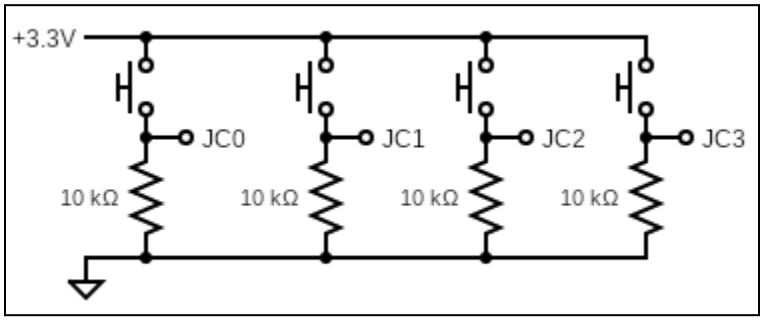
The logic for this was recreated into assembly code by manually recreating the function that the Arduino used which was written in C into the MIPS assembly instruction set for the processor to run. This provides the option for the assembly code to dynamically power, un-power, and reverse the DC motors, but this feature remains unused in the project.

In order to recreate the function in assembly two custom instructions were added: writeH (write high) and writeL (write low). These are analogous to Arduino's digitalWrite() method and allows the assembly code to write a digital output to the 8 pins on the Pmod header. More details on these instructions can be found in the Changes to Processor section.

Circuit Diagrams

I. Inputs

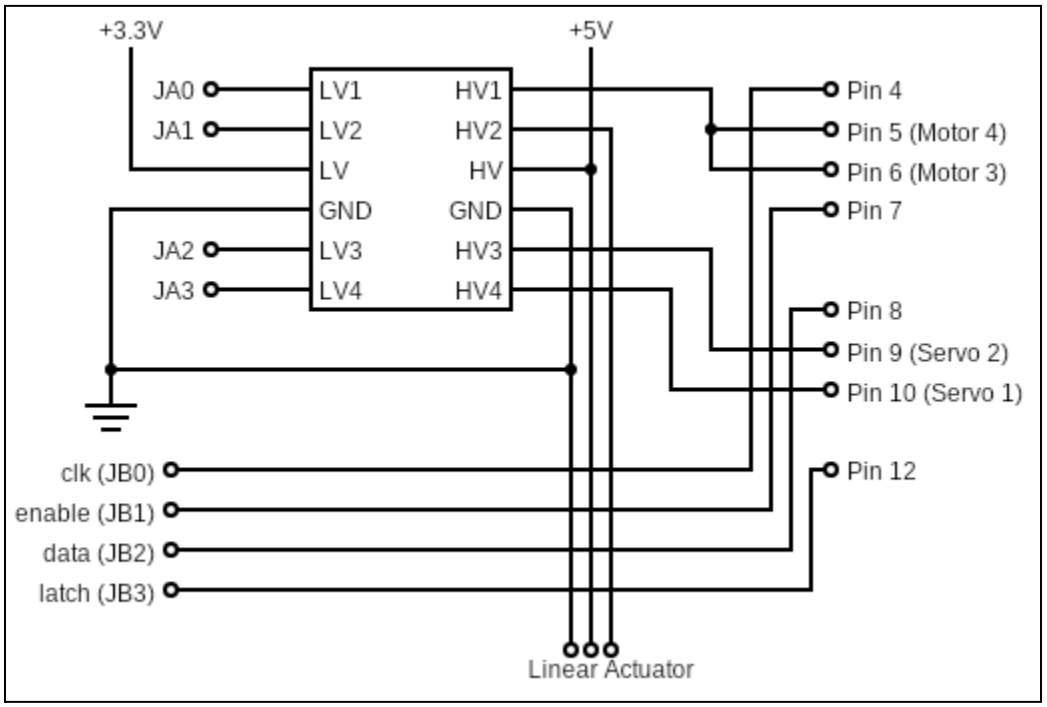
All of the 10 inputs were wired in the same configuration. All of the inputs connected to a given Pmod header were wired in parallel with individual 10 kOhm pull-down resistors and the switch pulling up to 3.3V. In reality, the wiring of the movement buttons became convoluted due to the buttons requiring to be in a specific arrangement of up, down, left, and right rather than in a linear line. A protoboard mounted to the top of the controller contains all of the wirings for the 10 inputs and utilizes male pins for easy troubleshooting.



JC Pmod Header Input Circuit Diagram

II. Outputs

The four PWM signals were wired into a 4-input Logic Level Converter to convert from 3.3V to 5V. This was necessary mainly for the two continuous servos and one linear actuator since 3.3V wouldn't be enough to sufficiently power them. The other four digital logic signals for initialization were wired directly to the input pins of the motor controller. A diagram is shown below which mirrors the layout implemented on the protoboard in the controller.



FPGA Output Circuit Diagram

Changes to Processor

I. Load Digital Input (LDI)

LDI is an I-Type instruction that writes a value into $rd and adds the immediate (N) with the value of $rs to determine what value is written. A 16-bit wire was added as an input into the processor which corresponds to each of the 8 pins on both the JC and JD Pmod headers. In the Wrapper, these 16 inputs were all debounced using two DFFs one wired into another, both of which are clocked on a slow clock in the hundreds of Hz range. In the execute stage of the processor pipeline, the inputs are put into a 16 to 1 MUX, whose select bits come from the first four bits (using [3:0]) of the result of the ALU which adds together the immediate (N) and $rs. The single-bit output of the MUX is appended with 31 leading zeros to create a 32-bit value that is passed into the O-reg of the X/M stage latch. The opcode of LDI, 01110, is also added into the bypassing and stall logic as an opcode that reads from $rs and writes to $rd so that the assembly code can aptly utilize the branching instructions.

II. Write High and Low (WriteH & WriteL)

These two instructions are both I-Type instructions but don't utilize the $rd value. The $rs and immediate values are added together in the execute stage using the ALU. In the memory stage of the pipeline, the first three bits are used to create an 8-bit one-hot encoding of the output pin selected. This encoding is passed as input into a register file made of eight 1-bit registers (DFFs) whose data is determined by the opcode of the two commands. The outputs of this register file are wired as outputs of the processor which is then wired to the 8 output pins of the JB Pmod header. The register file is also clocked on the negative edge of the clock, as is the main register file and RAM so that the X/M latch has time to propagate its new values. The opcodes of these two instructions (01101 = WriteH, 01100 = WriteL) are also put into the logic for bypassing and stalling, but only as instructions that read from $rs.

Assembly Code

The assembly code is divided into two stages: initialization and main loop. The initialization stage outputs the initialization signals required to set up the motor controller to correctly drive the motors. The main loop of the program repeatedly checks the movement, firing, and limit switch inputs to decide what to output on the PWM signals that control the servo and DC motors.

The core of the initialization stage is a function called latch_tx. This function is a mirror of the one found in the Adafruit Motor Shield v1 library and implements the process described in the Digital Logic Signals section above. This function is first called initially with data 00000000 to stop all movement of the motors and then called again with 00100001 to drive motor ports 3A and 4A.

The main loop first checks if either firing mode is activated. If the fire button is turned on, a value of 1 is written to memory address 1001 which is wired to the linear actuator controller module. A value of 1 tells the module to simply fully extend the linear actuator. If the auto switch is turned on, a value of 2 is written to address 1001. This tells the linear actuator controller module to repeatedly switch between full extension and retraction of the arm to repeatedly fire the Ping-Pong balls as described in the PWM Signals section above. If neither switch is on a value of 0 is written and the arm fully retracts.

The main loop then checks for all inputs related to the panning motion. The custom instruction LDI is used to store the left and right movement buttons as well as the left and right limit switches. If either the left or right movement buttons (but not both) are pushed, the corresponding limit switch is checked for activation. If no activation of the corresponding limit switch is detected then the value for clockwise or counterclockwise movement is written to the memory address 1003 which controls the bottom servo. When no movement should happen a value of 0 is written to the memory address. This logic is the same for the tilt movement except that the inputs are from different pins and the output is written to memory address 1002.

Challenges

One of the largest challenges of the project was dimensioning and designing how the mechanical parts of the turret would work in unison. Because 3D printers are not precise to fractions of a millimeter, designing the model in CAD required introducing roughly one and a half millimeters of excess room everywhere on the model. However, sometimes the printer was precise, sometimes it was not. The solution to this was to create a parametrized CAD model, where each dimension is defined in terms of a variable relative to one another. When a print did not turn out as expected, changing the model was as simple as altering one parameter. This helped save time in subsequent prints when a dimension did not turn out as expected, or the print itself failed due to design reasons.

Another large challenge was in soldering the PCB. Because there were ten inputs, it was a very hefty soldering job, one beyond the scope of what the team had done before. Because the entire functionality of the project relied on the PCB which routes the inputs and outputs to the respective PMod headers, the solder job required had to be precise and without error. (Iterative testing, as described below, helps explain how the team ensured the PCB was soldered correctly.) Ensuring there were no bad connections on the board, and that the wiring diagram on the PCB matched the one designed by the team, proved to be a time-consuming, challenging process.

Testing

Because of the mechanical nature of the project, testing required ensuring both the motors worked when given input on the controller and that the turret fit together with the motors in such a way that both fires a ball and rotates. Primarily, using an oscilloscope the digital signals were tested to make sure the timing was as expected. These signals are described in detail in the Inputs and Outputs sections. Once the team was confident that the signals & timing used were as expected, the turret was tested mechanically. This entailed rotating the turret horizontally and vertically via the inputs on the controller. Part of this testing was ensuring the limit switches worked as expected. Because the limit switches are wired in parallel, a multimeter was first used to ensure the soldered connections between switches were good connections. Secondly, the turret was rotated to contact each limit switch, which the team tested to see if the servo motors would stop rotating, in accordance with the assembly code.

It was also tested to ensure that the firing mechanism worked as expected. The turret was fired at an angle above the origin, below the origin, and on the origin. This was to ensure that the linear actuator was pushing the ball directly into the firing mechanism. This was also done to test the bar that runs through the turret body–ensuring that a ball is *not* fired while the turret is tilted down without pressing the fire button. While soldering the PCB, a multimeter was used to ensure each soldered connection was a good connection. This was done iteratively throughout the soldering process. If any bad connections were detected, it would require unsoldering the component and resoldering. This thorough testing process helped ensure that the mechanical component of the turret worked, as well as the controller of the motors functioned properly as well.

Future Improvements

There are a few future improvements that can be made to the design to improve wire management. Chiefly, the limit switches on the rotating base have no designated cutout to hold them in place. This means that they are taped in place, and the wires have no designated holes to go through. In the stand, for example, there are designated cutouts to feed the horizontal limit switches' wires through, thus 'hiding' the wires below the stand and to the controller. A simple improvement to the limit switches is twofold: creating an encasing to hold the switches in place without tape and creating a cutout in the rotating plate to feed the wires through. This change would reduce the number of wires visible to the user. Secondly, there is a counterweight hot glued at the back end of the turret body. This is due to the fact that the axis of rotation is not about the center of mass of the turret body. Another simple improvement would be to weigh the DC motors, and turret body, and move the axis of rotation of the turret body to be centered about the center of mass, thus reducing the need for a counterweight. Finally, quick disconnects for all output wires to the turret could be added to help wire management. This could be done in a variety of ways, but possibly using Molex connectors could be a clean solution.

New Features

There are primarily two features that would be added to the project given more time. Firstly, the initial goal was to implement the bi-directional turret with a SEN14607 infrared camera which used heat tracking to aim at a target. Implementing this camera would require first creating a housing mechanism on the turret body for holding the camera in place. Secondly, the communication protocol (I2C) used by the camera would have to be integrated with our controller design. The assembly code would also be altered to include logic on how to rotate the servo motors in accordance with the output read by the camera. This would likely require heavy testing.

The second feature that could be added is a 'special' feature which occurs if all four movement buttons are pressed at once. Currently, pressing each button moves it, and if the left and right buttons (or up and down buttons) are pressed at the same time, the servos do not move. However, there is a case, where the up, down, left, and right buttons are pressed at the same time, which could be used to add a special feature. This feature could take the form of playing a unique noise or firing sound.