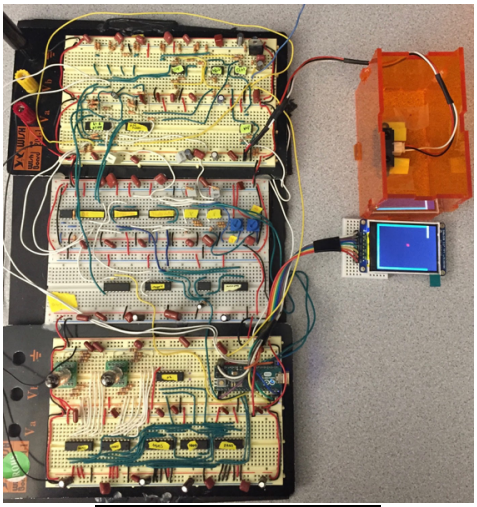
ES52 Final Report:
Single-Player Analog Pong

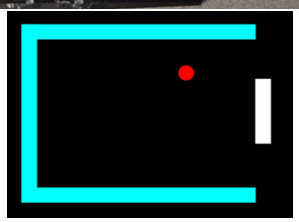
Enxhi Buxheli, Austin Sams, Adham Meguid

December 11, 2017

Abstract

Pong is a table tennis video game that is one of the earliest video games ever created. The game consists of two rectangles (paddles) controllable by a human or a computer and a small square (ball). The paddles are on either end of the screen (arena) and the player or computer can only move the paddles vertically. The ball starts in the middle and moves towards one of the two paddles, reflecting whenever it hits a wall or a paddle. If a player fails to hit the ball back, then the other player gains a point. One of the earliest versions of *Pong* was implemented using solely analog circuitry. In this project, we decided to create a single-player version of the analog *Pong* game in which the player controls a paddle using their hand's distance from an infrared sensor and there are three walls which the ball can bounce off of. Effectively, the player gets a point for each bounce off of their paddle, and their score is the number of bounces before they miss the ball.





Contents

1 Introduction

Please note: we have chosen to intersperse our "Results" portion of this report with images and outputs throughout section 2.2, as we felt this is where they best fit in. We have also moved most of our digital circuitry discussion and code discussion until after the appendix as well, for logistical difficulties.

1.1 Ideal Project Behavior

In our version of Pong, we used both analog and digital circuitry in order to bring about Endless Single-Player Pong. Our analog circuitry in large part determines what is displayed on our digital displays! For example, the ball and paddle position are determined by analog

circuitry which is then fed through a microcontroller to display gameplay in a fun and colorful way for the user.

The ball's position is controlled by integrator's and flip-flops which can cause a change in velocity to the ball each time that the ball hits a wall. This flipping of velocity is used to bounce of walls and the logic behind that is controlled with comparators. These comparators have a ton of hysteresis to prevent multiple switches of the ball's velocity. The paddle's position is determined by the user's position of their hand above the infrared (IR) sensor. When the user lowers their hand, the paddle lowers with it. When the user raises their hand, the paddle rises with it!

Every time the ball hits the paddle, the user's score goes up by a point and if you're good enough, you might even get the "NEW HIGH SCORE!!" message. All of this was possible through the combination of digital and analog circuitry.

1.2 Power Supply Scheme

During our planning phases for this project, we decided upon using dual supply. We chose dual supply because single supply design requires experience and caution, as it can easily lead to careless mistakes. For example, when using Op Amps and Comparators with single supply you need to take input common mode range and output swing into careful consideration. Input common mode range defines the range of input voltages an op amp or comparator can receive relative to its Vcc and Vee. Output swing defines the range of output voltages an op amp or comparator can supply releative to its Vcc and Vee. In single-supply, you are more likely to exceed one of these two specs, as you have a smaller voltage range you can work as an input to your IC. If you exceed one of these specs, unexpected behavior can occur (outputs could be clipped, for example) and leave the circuit-designer puzzled.

However, when performing initial tests reading voltages into the Arduino, we discovered that the Arduino cannot read negative voltage. As a result, we could not use negative voltages for the arena or the ball; we needed to use single supply while having "negative"

values. We achieved this behavior by creating a +2.5V pseudoground, as discussed later. We needed +5V for our digital logic, but we also needed +9V to overcome input common mode range limitations in our comparators. To compensate, we use a power suppply set at +9V, send that voltage directly to our comparator Vcc pins, and then use a voltage regulator to create +5V for the rest of our circuit. The voltage regulator circuit is discussed in Section #2, Subsystem #1.

2 The Design

2.1 Systems Breakdown

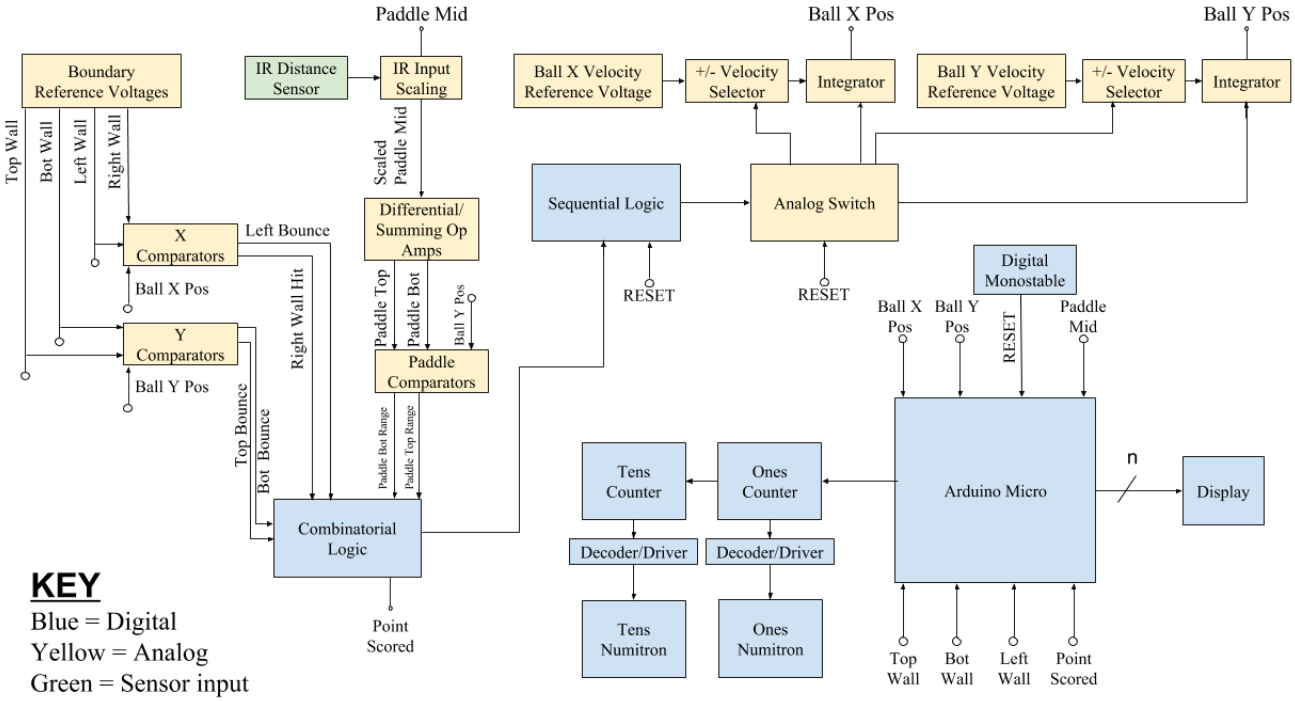


Figure 1: Analog Pong Block Diagram.

To play a game of Analog Pong, we first need to construct a playing field, an arena. The *Arduino Micro* block will be reading in voltages and rendering them on the Thin-Film Transistor *Display* block. We thus create an arena by using voltage dividers to set the

boundary wall voltages of our arena, one at the top of the display, one at the bottom, one to the left, and one 'invisible' wall that will determine where our paddle lies. The 'top' of the display was chosen to be represented by +4V, as was the 'invisible' paddle reference wall. The bottom and left sides of the display were chosen to be +100 mV. We use voltages ranging from 0 to +5V, because the Arduino can only read in positive voltages, and because a significant portion of the digital circuitry in our schematic is characterized at +5V, or only +5V tolerant. These references are fed into the *Arduino Micro*, which uses the voltages to display corresponding rectangular walls on the *Display* block.

Having set up our arena, we can now look towards our ball. The bulk of our circuit begins with the balls' X and Y velocities, represented by the *Ball X Velocity Reference Voltage* and *Ball Y Velocity Reference Voltage* blocks in Figure 1 respectively. These blocks each contain a potentiometer as a voltage divider that allows their output to range from values as high as +5V to as low as +2.5V. The +2.5V is a pseudo ground, which we use as the 'zero' for all arena referencing because the Arduino cannot read negative voltages. These reference voltages can be switched between their positive value and their 'negative value' (0 to +2.5V) using an analog switch, as shown in the *+/- Velocity Selector* blocks. This switching will occur whenever a ball hits a wall in order to simulate a ball bouncing off a wall and reversing its velocity. The outputs of the velocity selectors go into separate integrators for the Ball's X velocity and the Ball's Y velocity, shown by the *integrator* blocks. These integrators will sum up the ball's velocities over time, and since the integral of velocity is position we can get the ball's X and Y coordinates from the output of the integrators. These outputs are sent to the *Arduino Micro* block, so that it can read in the voltages and display the corresponding position on the *Display* block.

Once the ball is in motion, eventually it will contact one of our arena references. When it does, either our *X Comparators* block or *Y Comparators* block output will change. This change in output will cause a change in our *Combinational Logic* block, which will in turn impact our *Sequential Logic* block on the rising edge. We use sequential logic because we

want the corresponding change to be based upon the current state. Effectively, when the ball hits an arena wall we want to modify the current state by reversing one of our two velocities. We then want to *maintain* this new state until another rising edge, so that after a velocity reversal our ball continues to have the reversed velocity until it hits another wall. This sequential logic feeds into the *Analog Switch* block to determine whether one of the ball's reference velocity voltages should be flipped.

The player uses the paddle by moving their hand up and down above an infrared distance sensor, shown in the block *IR Distance Sensor* . Our sensor in its current setup has an output that ranges from +1.2V to +3.2V, so it needs to be scaled to 0V to +5V. We scale the output in the *IR Input Scaling* block, which then outputs the scaled voltage to the *Arduino Micro* block as the paddle's midpoint for rendering. However, getting the ball to bounce off of the paddle conditionally is trickier, as the ball could be at the correct X position but be above or below our paddle. For this reason, we use the *Differential/Summing Op Amps* block to obtain the top and bottom voltages of the paddle's range. We can then use these voltages in conjunction with the ball's Y position at the *Paddle Comparators* block to determine whether the ball is in the Y range of the paddle. Finally, we feed this information into more *Combinational Logic* to determine whether the ball is at the required X position for bouncing as well. If so, and if the ball is in the paddle's range, the ball's X velocity is reversed. If not, the ball continues on as normal.

The *Arduino Micro* block is also in charge of keeping score. Every time the ball bounces off of the paddle, the *Arduino Micro* block receives a signal from the *Combinational Logic* block, increments an internal code variable for the counter (for later high score storage), and in turn sends a rising clock pulse to the *Ones Counter* block. This counter increments, cascading over to the *Tens Counter* block if needed. The counter outputs are send through *Decoder/Driver* blocks so that they can be displayed on the two Numitrons to show the player their current score. We found that 2 Numitrons were sufficient, as 99 points takes a significantly long time to achieve. However, the Arduino's internal variable counts past 100,

and the Numitrons cleanly reset to 0, so if a user wanted to they could claim a score as high as 32,767, Arduino Micro's max signed integer value.

Finally, at the end of a game, the *Digital Monostable* block counts for 5 seconds before queuing the Arduino to send a signal to reset any blocks that need it so that we are ready for a new game.

2.2 Subsystems

2.2.1 Subsystem #1: The Voltage Regulator

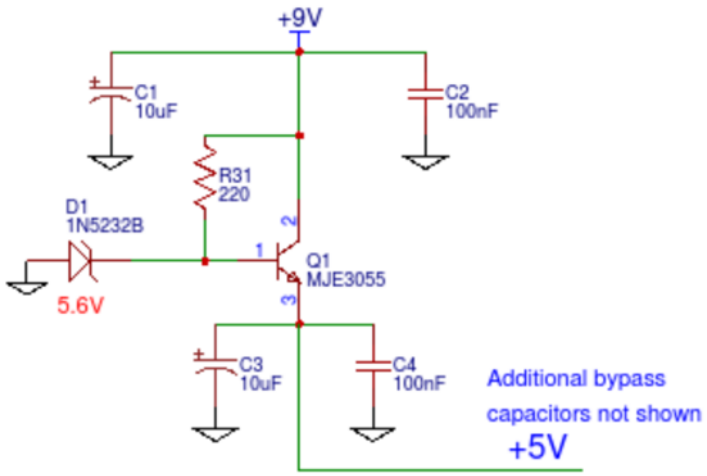


Figure 2: The schematic of the 9V to 5V voltage regulator used in our circuit.

While the majority of our ICs and references required a 5V supply, there were a few parts that required a higher voltage in order to operate correctly, including the Arduino Pro Micro, and both LM399 comparator packages (U3 and U5). Rather than use two power supplies, one set to +9V and another set to +5V, it seemed prudent to build a 9V to 5V regulator which would allow us to power both the circuitry that required the higher voltages and the rest of the boards with a single power supply. The +9V supply is connected to the collector of a MJE3055 power BJT and in parallel to the base of the BJT through a 220Ω resistor. When the +9V power supply is turned on, the 5.6V 1N5232 Zener Diode becomes reverse

biased due to the +9V flowing through the 220Ω resistor, causing the zener to act like a stable 5.6V voltage source. Apart from connecting the +9V power supply from the collector of the MJE3055 to the base in order to reverse bias the zener diode. The 220Ω resistor also serves to protect the zener diode by limiting the amount of current it receives. Due to the BJT rules, the voltage drop that occurs between the base and the emitter is 0.6V, giving us a stable 5.0V output from the emitter of the MJE3055. Bypass capacitors, both electrolytic and non electrolytic were placed in parallel with both the +9V voltage supply and the newly created +5V voltage supply in order to ensure stability. Below is an image of the +9V to +5V voltage regulator in action.

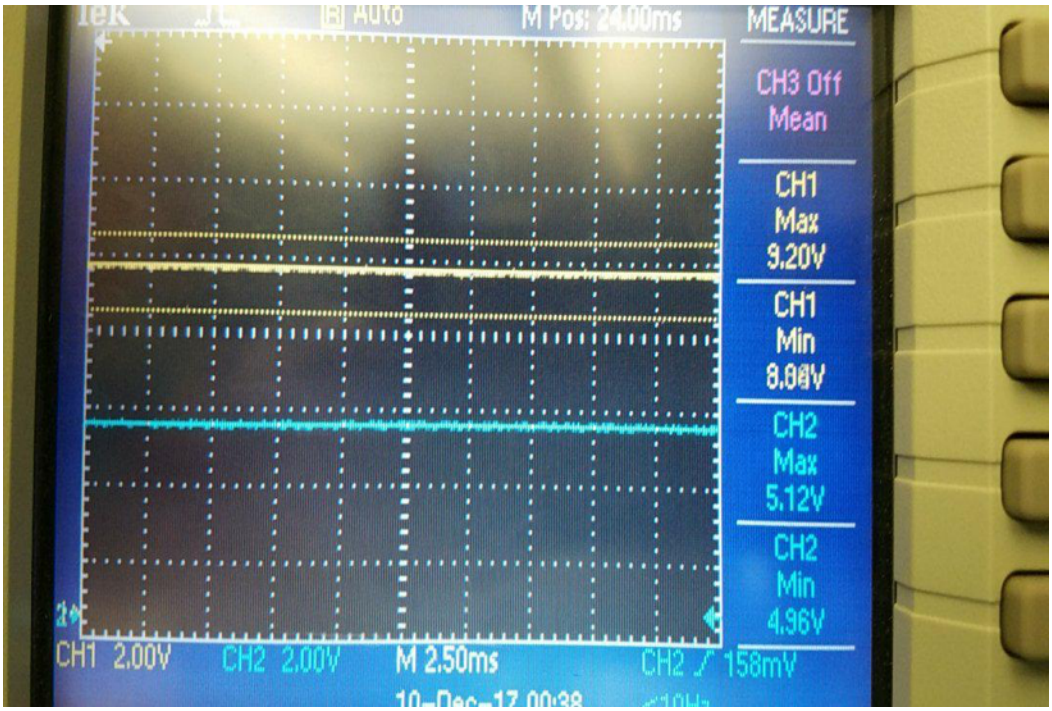


Figure 3: Image demonstrating the operation of the voltage regulator.
Yellow = 9V input from PSU
Blue = Regulated 5V output

2.2.2 Subsystem #2: Voltage References for the Arena

In order to build our pong arena, boundary conditions were needed. In deciding what voltages to use for our walls, we decided on bounding our choice to the Arduino's analog

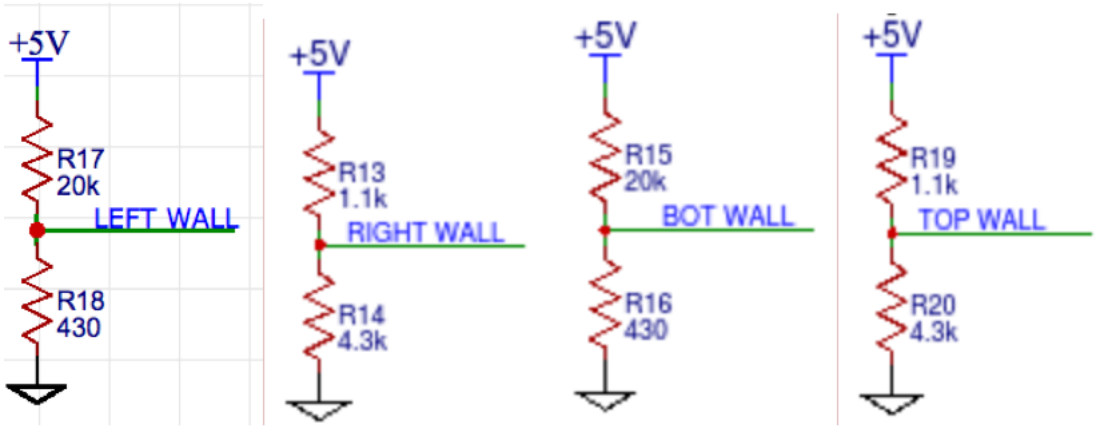


Figure 4: The schematic of our arena wall references. Please note that "Right Wall" is equivalent to "paddle's X position reference" and will be used interchangeably.

input voltage range of 0 to +5V. This choice reduced the amount of needed circuitry, as we would not have to build a circuit to shift the wall references into the input range of the Arduino. That being said, we could not just make our walls 0V and 5V, as there would be a period of time where the ball's position (which was being fed into the Arduino to map its current position in the arena) would continue integrating beyond those boundaries before the comparator output a "HIGH" signal and inverted the direction of integration. In addition, we needed to decide whether to have the game-over zone's reference to be in line with the paddle's X position reference voltage (i.e. the same value), or to have the game-over zone's reference voltage to be slightly greater than the paddle's X position reference voltage. Having the game-over zone and paddle's X position reference voltages be exact same would cause our game to rely solely on combinatorial logic in deciding whether the ball would bounce off of the paddle or end the player's game, while having them separated by some amount would allow both a buffer for signals to propagate through our circuit and lower the chance of a false Game Over, it would also risk complicating how far back to place the Game Over area on the display. After some debate and several iterations, we eventually settled upon the separated paddle and game-over zone design with the bottom and left wall's references being +100mV and the top wall and paddle/game-over zone's reference to be at +4V. We

found that +100mV was enough of a buffer to allow the ball to continue integrating while the circuit switched the direction of integration without it reaching territory that would be dangerous to the Arduino. Since the paddle's circuitry was expected to be fairly complicated, we decided to give it a larger buffer by making it +4V rather than +4.9V. For simplicity and a square arena, we matched the top wall's reference voltage to the paddle's X position reference voltage. As for the game-over reference, we decided to have the Arduino end the game once it read that ball's X position had reached a value of +5V. A drawing of the final arena with accompanying reference voltages can be seen below.

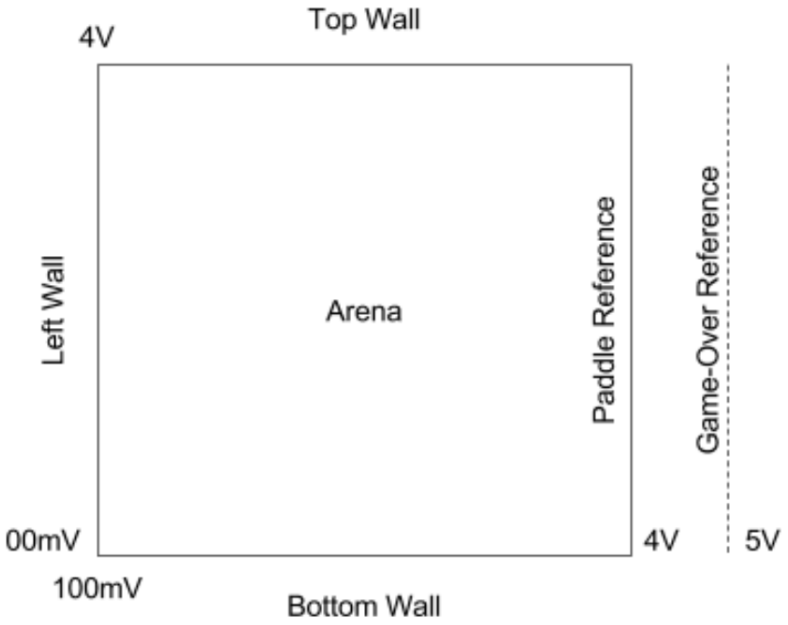


Figure 5: An image depicting the arena and associated references.

2.2.3 Subsystem #3: Shifter and Scaling Circuit for IR Distance Sensor

Prior to shifting and scaling the output of the IR distance sensor, it was necessary to first characterize its output in response to varying distances. To characterize the IR distance sensor we affixed a yardstick to a stable surface and aligned it so it was just above the sensor. Then, measurements of the sensor's output in relation to our hand were taken at incremental distances. A summary of the results can be seen in the table below.

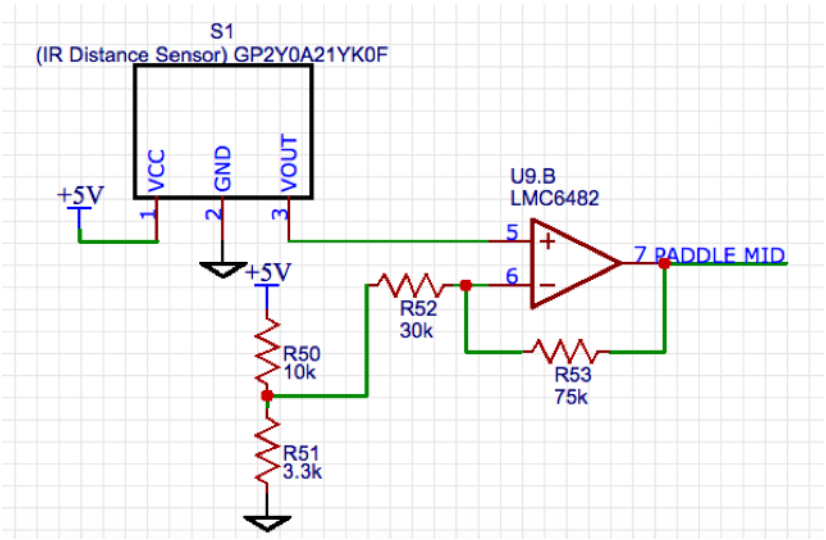


Figure 6: Schematic of the shifter and scaling circuit used to change the output of the IR distance sensor from 1.2V - 3.2V to 0V - 5V.

Distance from sensor	Voltage output
1 inch	3V
2 inches	3.4V
3 inches	3.2V
4 inches	2.3V
5 inches	1.9V
6 inches	1.6V
7 inches	1.3V
8 inches	1.25V
9 inches	1.2V
10 inches	1.2V

Figure 7: A table summarizing the characterization of the IR distance sensor.

The sensor had a fairly predictable increase in voltage as the distance to the sensor decreased, however, we found that getting too close (less than 2.5 inches away) to the sensor resulted in the voltage actually decreasing. To prevent this phenomenon from causing

confusion with the player, we decided to construct a box around the sensor to prevent the player from getting too close to the sensor. This effectively capped the maximum voltage output of the sensor at 3.2V. Although the 1.2V to 3.2V range provided by the IR sensor was sufficient for the Arduino, as we could have scaled the input in code to match the arena size, the creation of paddle references that spanned the entirety of the arena necessitated the design and creation of a shifting and scaling circuit. As characterization of the IR sensor determined its effective range was 1.2V - 3.2V, the shifter and scaling circuit was required to first subtract 1.2V from the IR sensor input and the multiply the resulting difference by 2.5, or, in other words: $2.5(Vsensor - 1.2V)$.

While we were unable to find documentation of the output resistance of the IR distance sensor, this proved to not be an issue, as the signal was fed directly into the non-inverting input of the LMC6482, U9.B. Next, the 1.2V reference was created via a voltage divider and then fed into the inverting input of U9.B. As the voltage divider had a thevenin output impedance of 2.48kΩ, this was done through a 30kΩ resistor to avoid thevenin loading issues with the gain of the U9.B. A 75kΩ resistor was then connected from the inverting input of U9.B to the U9.B's output giving the circuit a gain of 2.5 - exactly what we needed! Below is an image depicting the functionality of the shifter and scaling circuit.

2.2.4 Subsystem #4:Differential/Summing Op Amps and Creation of Top/Bot Paddle Signals

Clipper bounds: In order to make sure the paddle both stayed within the bounds of the arena and maintained a constant length, a clipping circuit for both the top paddle position and the bottom paddle position was designed.

Bottom Paddle Position Clipper:

As we decided our paddle length would be 1 volt (approximately 25% of the total arena height) and the top wall of the arena to be +4Vs, the bottom paddle position should never exceed a Y position of +3Vs, lest the paddle size shrink. To achieve this clipping reference,

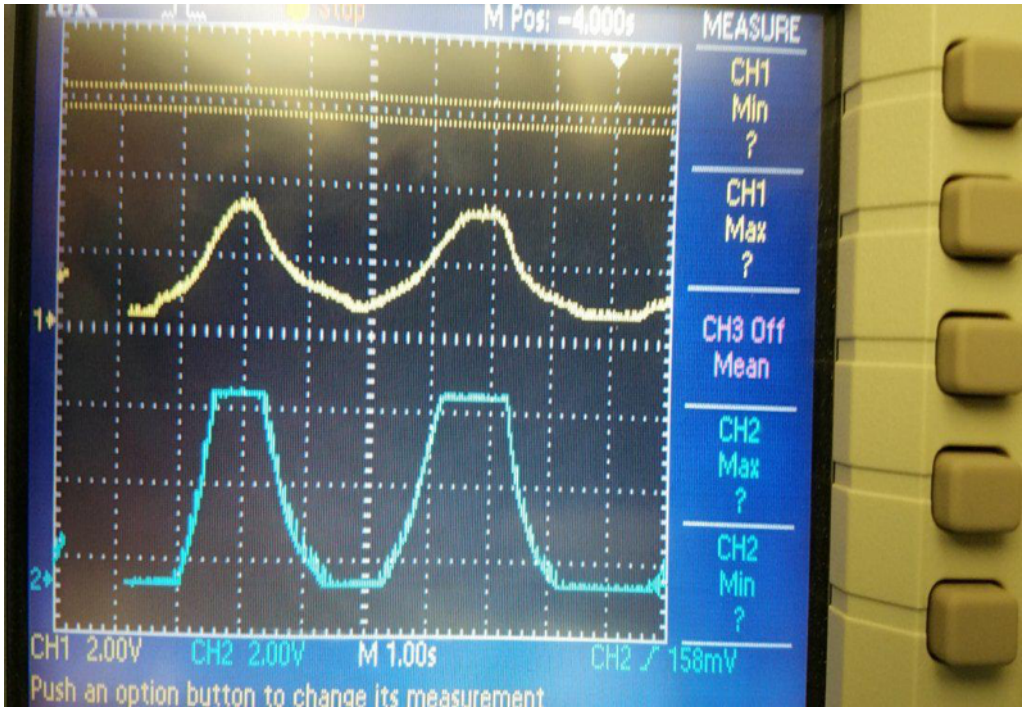


Figure 8: Picture demonstrating the shifter and scaling circuit.
Yellow = Raw IR Sensor Output
Blue = Shifted and Scaled Signal

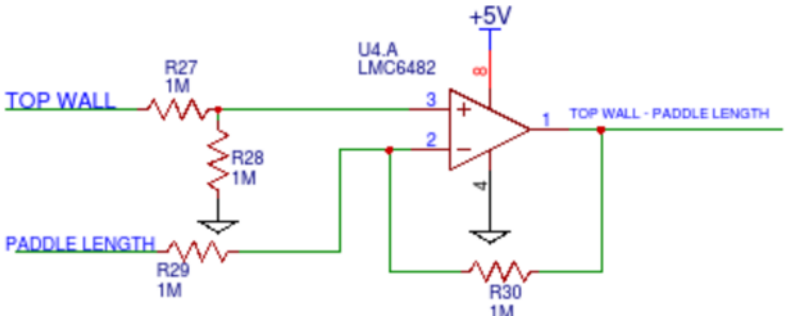


Figure 9: Schematic of the differential op amp used to clip the maximum voltage the bottom paddle position could reach.

1/2 of a LMC6482 op amp, U4.A, was employed in a differential configuration, subtracting the total paddle length (set by a voltage divider to be +1V), from the top wall reference. In order to prevent thevenin issues and to make sure the differential op amp was a unity gain, 1M ohm resistors were used. Such large resistors could be utilized due to the output of the newly created clipping reference going directly into another LMC6482 op amp U2.B, thus no worries about driving the next circuit arose. The resulting output (seen below) was as

expected, at +3V.

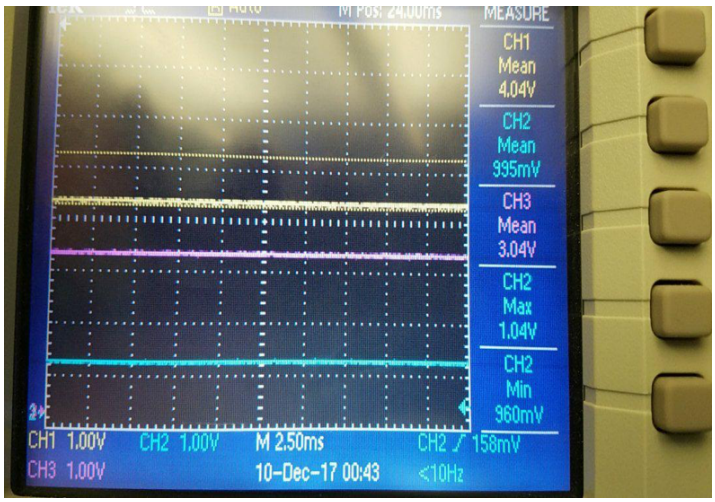


Figure 10: An image of the bottom paddle position clipping reference.
Yellow = Top Wall Reference
Blue = Total Paddle Length
Purple= resulting clipping reference for the bottom paddle position

Top Paddle Position Clipper:

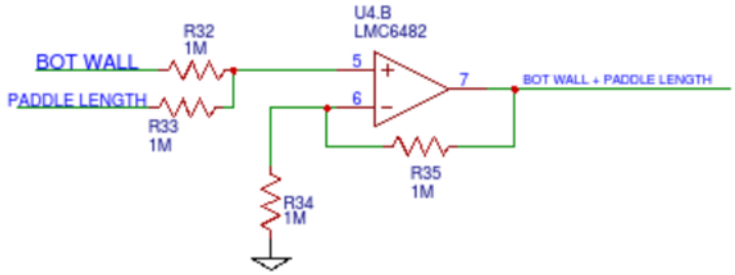


Figure 11: Schematic of the summing op amp used to clip the minimum voltage the top paddle position could reach.

As stated previously, that the paddle length should be a static 1V. Taken with the bottom wall reference of 100mV, the top paddle position should never go below a Y position of +1.1Vs, lest the paddle size shrink. To achieve this clipping reference, another LMC6482 op amp, U4.B, was employed in a summing configuration, adding the total paddle length (set by a voltage divider to be 1V), to the bottom wall reference (set by another voltage divider to be +100mV). Similar to the bottom paddle position clipper reference circuit, 1M

ohm resistors were used were used to both ensure unity gain and prevent thevenin loading issues. Again, no worries about driving the next circuit arose as this output was connected to another LMC6482 op amp, U1.A. The resulting output (seen below) was as expected, at +1.1V.

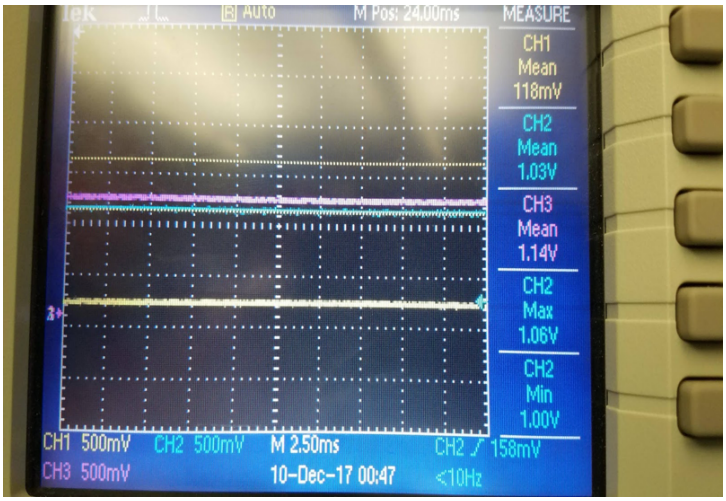


Figure 12: An image of the top paddle position clipping reference
Yellow = Bottom Wall Reference
Blue = Total Paddle Length
Purple= resulting clipping reference for the top paddle position

Creation of Paddle Positions:

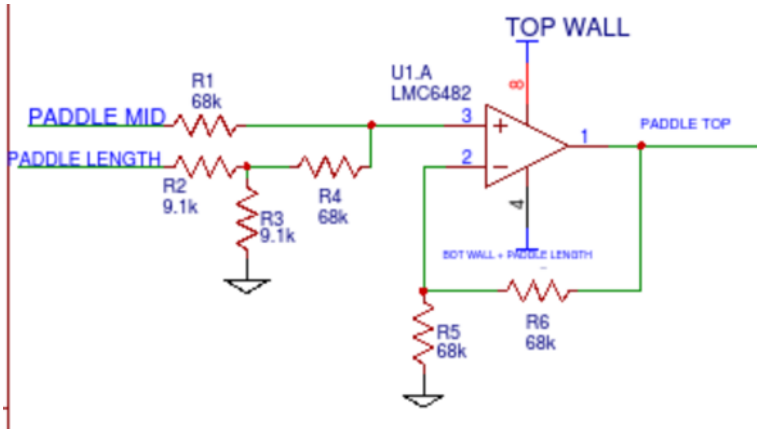


Figure 13: Schematic of the summing op amp used to create the top paddle position.

After setting the clipping references and shifting and scaling the voltage from the IR distance sensor, it was time to create the top and bottom positions for the paddle. This

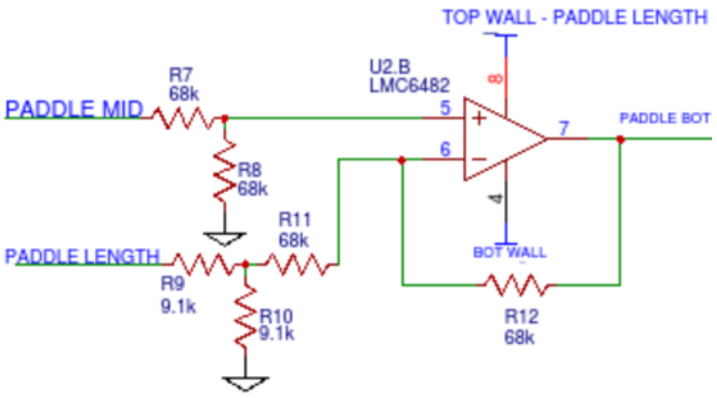


Figure 14: Schematic of the differential op amp used to create the bottom paddle position.

was achieved by taking the shifted/scaled input voltage from the IR distance sensor (the paddle's midpoint position), and adding/subtracting half of the total paddle length (500mV) using LMC6482 op amps (U1.A and U2.B, respectively). The clipping references and wall references were then set to the corresponding Vcc and Vee of each LMC6482 op amp in order to achieve a top paddle position that varied from +1.1V to +4V and a bottom paddle position that varied from +100mV to +3V depending on the voltage input from the IR distance sensor. Note that because U1A and U2B each required a different Vcc and Vee from each other that two separate LMC6482 op amp packages were required. Considerations of thevenin loading issues were very important in the creation of the circuit, as there were essentially three stages being combined - the voltage divider for the full length of the paddle, the voltage divider for 1/2 the length of the paddle, and the and the resistors used to give the LMC6482 op amps unity gain. The voltage divider for the total paddle length had a thevenin output impedance of 876Ω, while the voltage divider half the paddle length had a thevenin input impedance of 9.1kΩ. Similarly, the voltage divider for half the paddle length had a thevenin output impedance of 4.5kΩ and the input impedance for the differential/summing op amps was 68kΩ. Thus, all thevenin loading issues were properly considered and addressed. As can be seen from the image below, the clipping/paddle boundary creation circuit properly created signals for the top and bottom positions of the paddle that mirrored the shifted/scaled input from the IR distance sensor (the paddle midpoint) but that also maintained a separation of

1V.

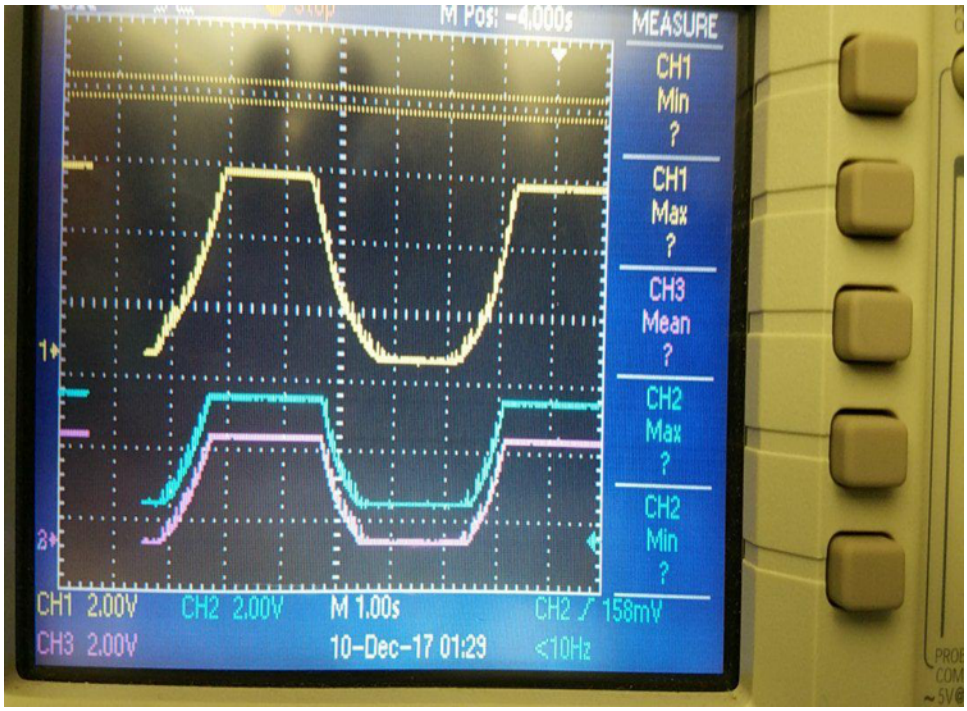


Figure 15: An image showing the top, bottom, and middle paddle positions after the shifting/scaling and clipping circuit.
Yellow = shifted/scaled voltage input from the IR distance sensor (paddle midpoint)
Blue = Top paddle position
Purple= Bottom paddle position

2.2.5 Subsystem #5: Comparators

Once all of our boundary conditions were agreed upon, set, and built, it was time to set up the comparators that would test whether the ball was within the arena or length of the paddle. If the ball was not within these ranges, a signal needed to be sent to reverse the direction of integration or to end the game.

The arena comparators consist of four LM339 comparators, or one full package, shown in the schematic above as U5. In order to prevent common mode input range issues, U5 was powered by a +9V supply, allowing all of the comparators to accurately distinguish when a trip point had been exceeded and output the corresponding voltage. In a comparator, if the non-inverting input voltage exceeds the inverting input voltage, the BJT inside of

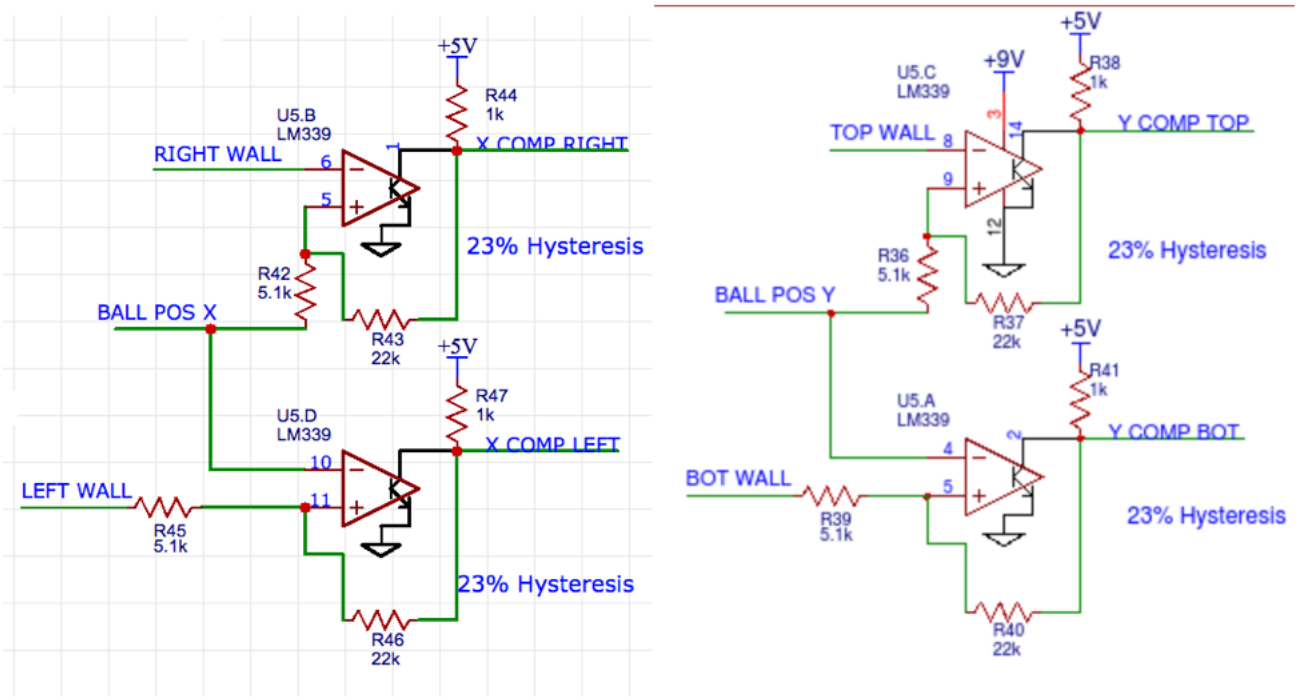


Figure 16: Schematic of X and Y comparators for the arena boundaries.

the comparator goes "OFF" and thus the resulting output is pulled high to the voltage connected. Similarly, if the inverting input voltage exceeds the non-inverting input voltage, the BJT goes "ON" and thus the resulting output is GND, or whatever lower voltage is connected to the bottom of the BJT. Since we wanted the comparators to test whether the ball's position was within a range of values (100mV to 4V in both the X and Y directions) we set up the comparators in a "window" configuration, where the ball position was in the inverting input in the comparator with the lower voltage reference wall and in the non-inverting input in the comparator with the higher voltage reference wall. Thus, if the ball's position was less than the higher voltage wall (the right wall in the X direction and the top wall in the Y direction), but greater than the lower voltage wall (the left wall in the X direction and the bottom wall in the y direction) both BJTs in the comparator would be "ON". This configuration does allow for the outputs of each comparator to be independent of each other - that is, their output voltages could be set to be different if so desired - but for the purpose of our circuit, it was actually more advantageous to have all of the outputs be

the same if the BJT was turned "OFF" by the ball's position exceeding any of the thresholds. In our system, the comparator outputs eventually act as the clock for a flip flop, therefore if the ball exceeds the bounds of the arena, a HIGH output from the comparator to simulate a rising clock edge was necessary. Therefore, we connected each output of U5 to +5V along with a 1kΩ pull up resistor. After some testing it became clear that while the comparators could accurately distinguish when the ball had exceeded the bounds of the arena, noise was causing the comparators to trigger multiple times. If the comparator was triggered an even number of times, it would essentially negate the signal to invert direction of the ball, causing it to be stuck slightly outside of the area's bounds. An depicting the multiple triggers can be seen below.

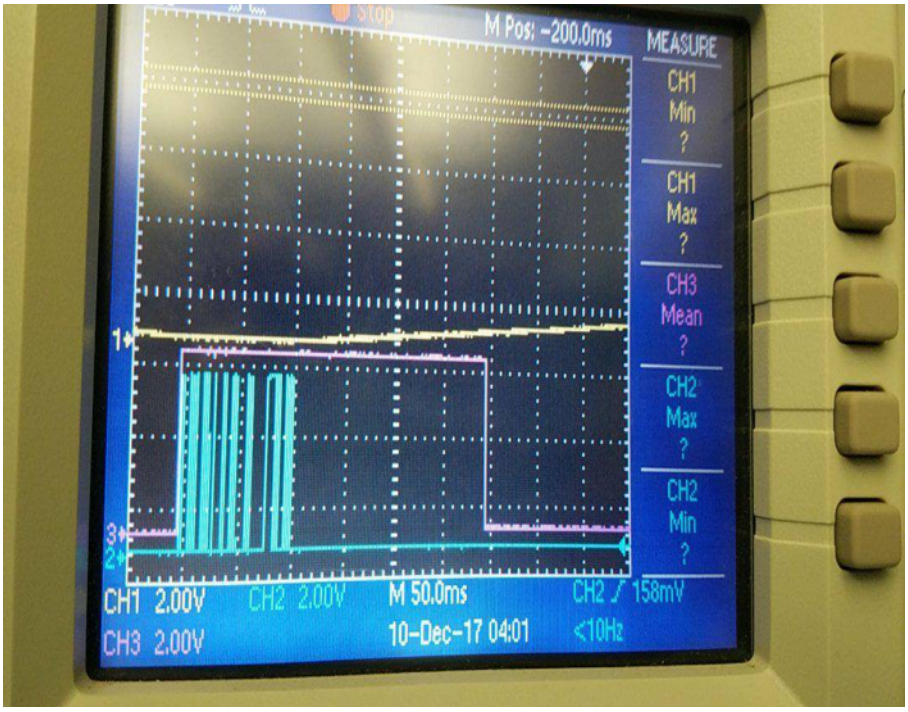


Figure 17: An image showing the output of the front wall comparator without hysteresis
Yellow = Ball X Position
Blue = Front Wall Comparator Output
Purple= Flip Flop Output

The solution for this multiple trigger problem? Hysteresis. By connecting the non-inverting input of the comparator to the output, thereby feeding back into the non-inverting

input some of the output's voltage, we were able to raise and lower the thresholds as the ball's position exceeded the arena's bounds and then returned to the playing area. Finding the correct amount of hysteresis was no trivial matter however; the LM339 comparators can turn ON and OFF on the order of 200 nanoseconds, making the changes sometimes too fast for oscilloscopes used in the lab to capture. Thus, finding resistor values that both yielded enough positive feedback and avoided thevenin loading issues with the voltage dividers proved challenging. After trial and error, we discovered that roughly 23% hysteresis was required to prevent multiple triggers on the static arena walls. While the path to this discovery was a long one, it was well worth it as can be seen in the images below.

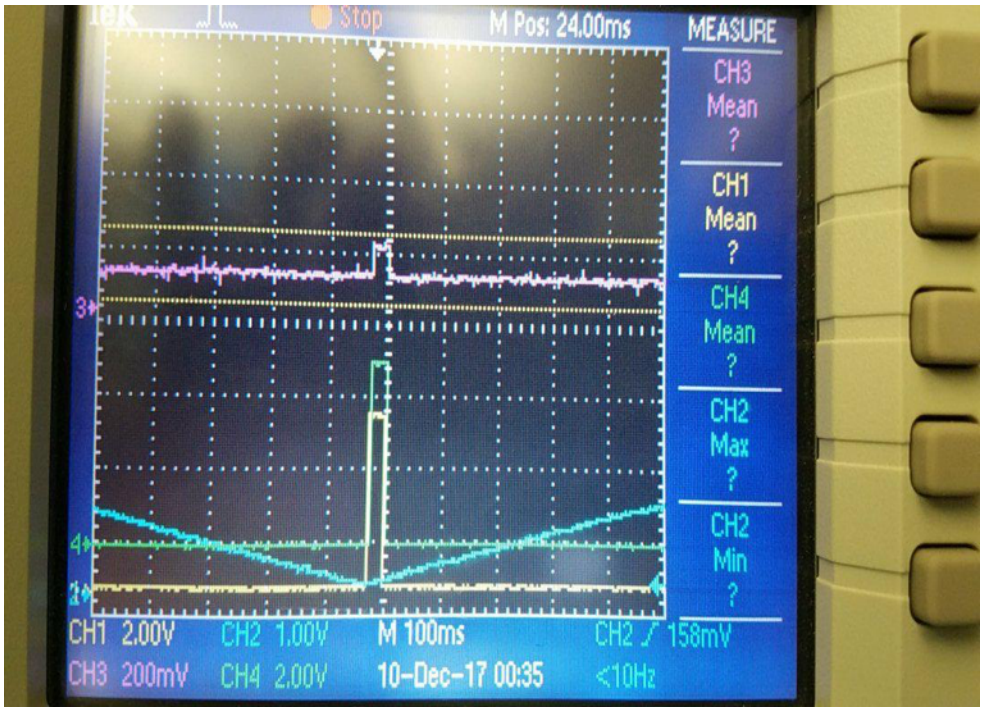


Figure 18: An image showing the the output of the front wall comparator after the addition of hysteresis.
Yellow = Front Wall Comparator Output
Blue = Ball X Position
Purple= Hysteresis

Similar to the wall comparators, the paddle boundary comparators were configured to be a "window," continuously testing to see whether the ball was within the range of the paddle. If the ball exceeded the range of the paddle, the corresponding comparator would send a

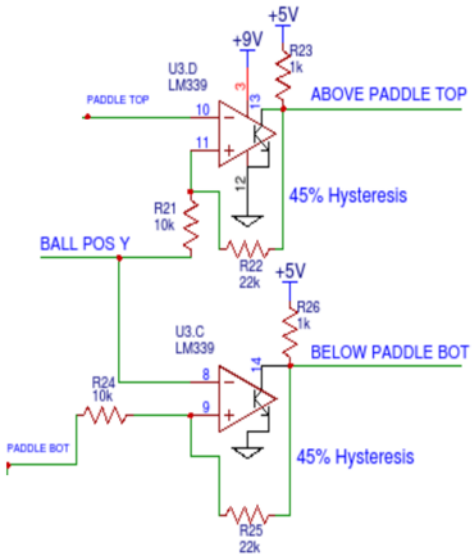


Figure 19: Schematic of the paddle boundary comparators

"HIGH" signal to the digital logic circuitry. Unlike the wall comparators, the paddle range was dynamically changing the with the player's input to the IR sensor. This caused much higher levels of noise, as the IR sensor can only read distance at a certain speed, and the speed of the player's hand changed quite frequently. Thus, much higher levels of hysteresis (roughly 43%) were required to prevent multiple triggers for the paddle comparators.

2.2.6 Subsystem #6: Pseudoground

Early on in our prototyping phase, we wanted to test the Arduino's ability to read the voltage references we created with our dual supply system. It was at this point that we realized that the Arduino cannot read negative voltages. We could not use negative voltages for the arena or the ball, so we needed to use single supply while having "negative" values. We created this 2.5V pseudoground reference to accomplish just this goal.

The pseudoground reference works by first using a voltage divider from our +5V line with two equal-valued resistors, R48 and R49. This voltage divider has 1/2 the input voltage at the intersection between R48 and R49, or +2.5V. However, simply passing this +2.5V to a rail would be inadequate for a number of reasons. First of all, the +2.5V pseudoground will

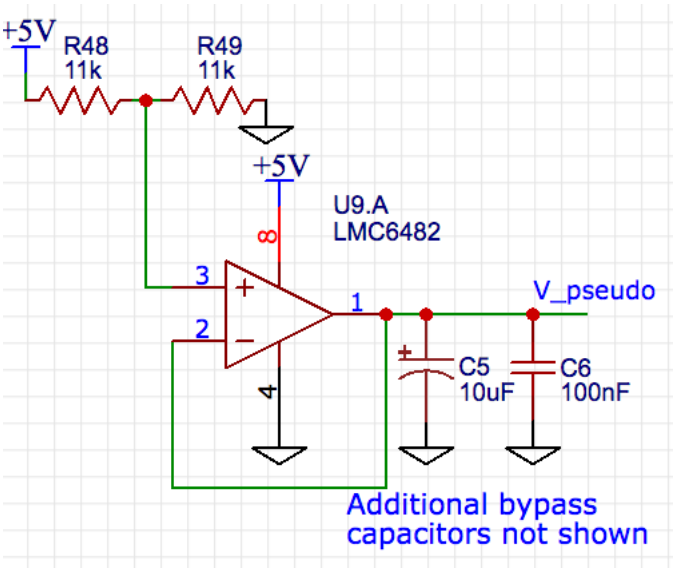


Figure 20: Schematic of the circuit used to create a stable +2.5V pseudoground.

be used as an input to numerous other IC's. Failure to buffer the output could mean that the voltage divider's output collapses, and starts outputting less than +2.5V to other circuits. Second of all, its thevenin output resistance may not be ten times less than whatever it's driving. For this reason, we use a unity gain buffer, U9.A, to ensure a constant output voltage of +2.5V, and effectively zero thevenin output resistance. Finally, we add bypass capacitors to the pseudoground rail, as is proper practice for all supply rails.

2.2.7 Subsystem #7 X/Y Velocity and Ball Pos:

The X and Y Ball positions are inherently based upon their velocities. We set the velocity of the ball in the X and Y direction with R59 and R54, respectively. These are two potentiometers configured as voltage dividers, ranging from +5V to our +2.5V pseudo ground. Going to pseudoground instead of ground allows us to "negate" the X and Y velocity while still using positive voltages. For example, "negating" +2.6V would provuce +2.4V. Remaining in positive voltage ranges is critical for the Arduino, as discussed above. From there, the voltage divider output goes to a circuit that Professor Abrams taught us about. This circuit works with a switch at + input of the op amp U12.A or U13.A. The switch is

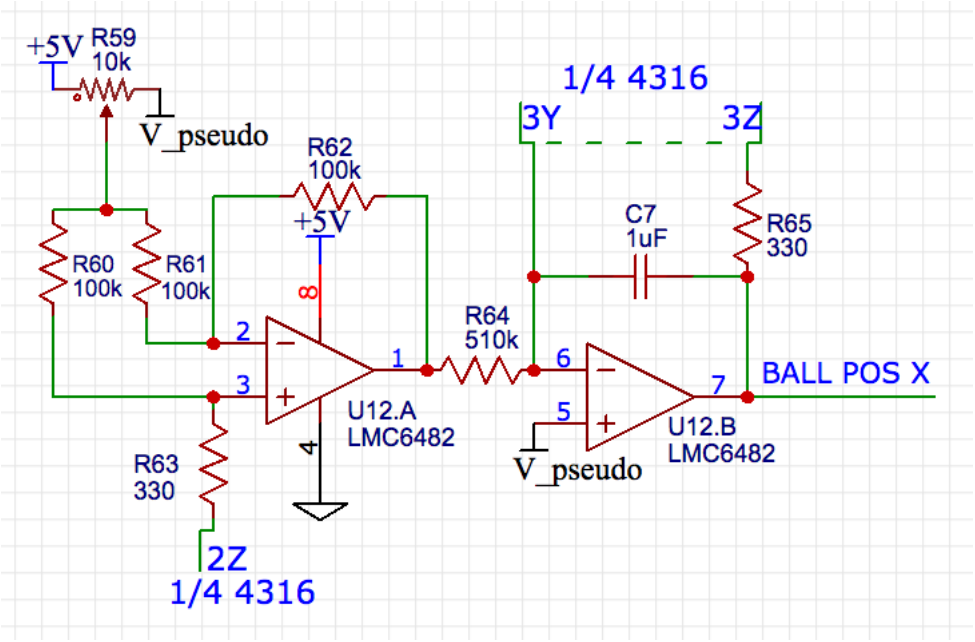


Figure 21: Schematic of the circuit used to determine ball's X position and alter its velocity.

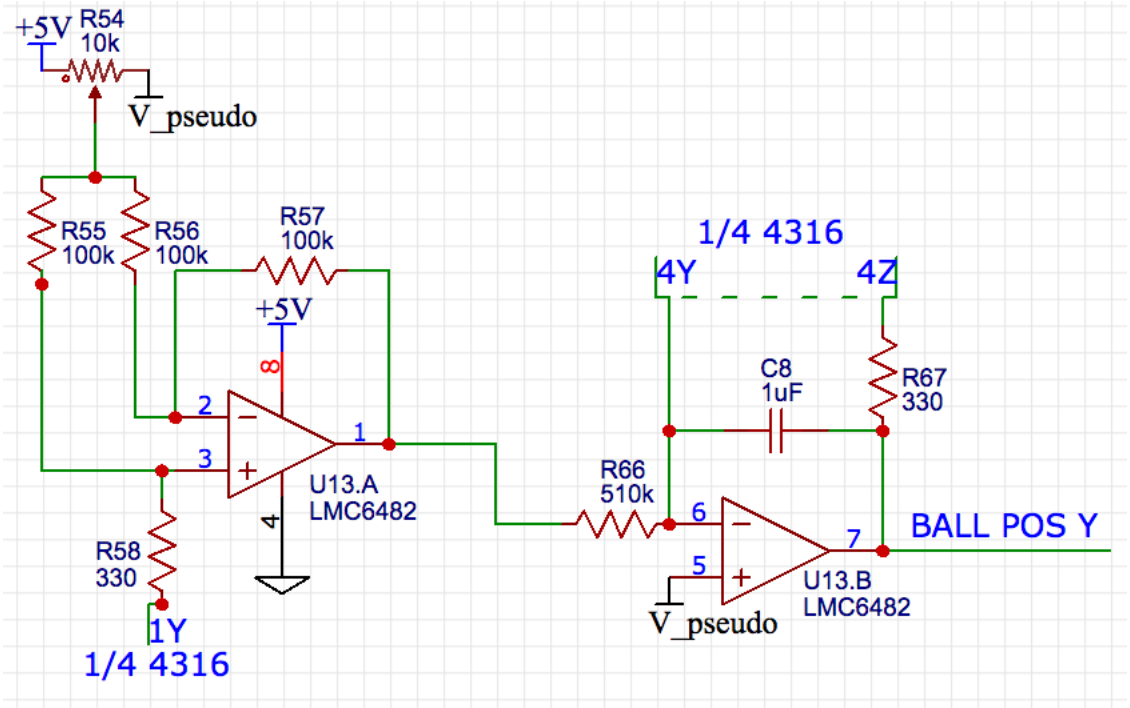


Figure 22: Schematic of the circuit used to determine ball's Y position and alter its velocity.

an analog switch (U11) which is shown below. When the switch is open, no current flows through R58 or through R63 (current limiting resistors), and the input voltage at the + input of U12.A or U13.A will be the positive reference voltage. Because an op amp with

negative feedback does its best to keep V-= V+, the voltage at V- is equal to V+. However, this circuit has a gain of 1, therefore Vout = the positive reference voltage. If the switch is closed to pseudoground, however, now we have an op amp configured in an inverting configuration. With an op amp in an inverting configuration, Vout = -Vin = -Vreference. But because we use pseudoground instead of ground, we flip over +2.5V instead of +5V, our desired behavior.

The below scope output shows the circuit functioning as expected for the X direction:

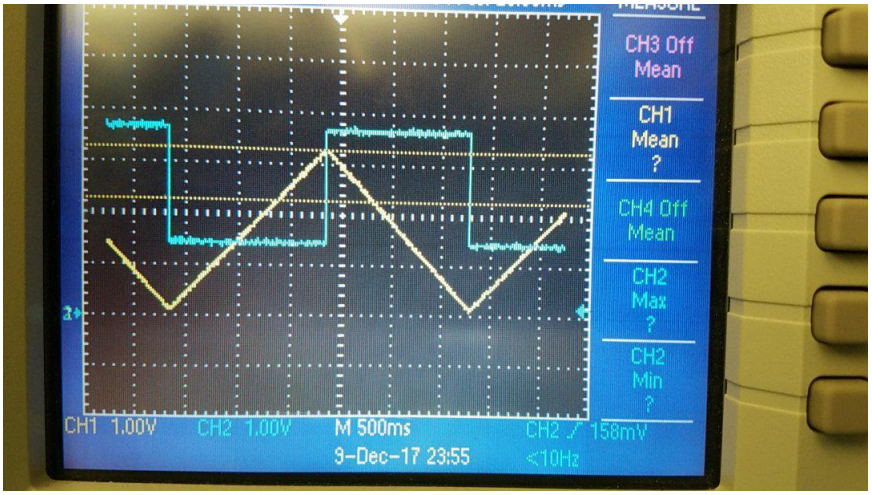


Figure 23: Scope output of X velocity and position with the velocity signal going between an inverted and non-inverted state.
Yellow = X position
Blue = X velocity

The output of 12.A and 13.A simply go into op amps as inverting integrators, U12.B and U13.B respectivecly. The R64/R66 and C7/C8 were chosen empirically such that the ball seemed to be able to reach a max speed that was challenging, but mid and low speeds that were reasonable. The output of U12.B and U13.B are our X and Y Ball positions, respectively.

2.2.8 Subsystem #8 Combinational Logic and Flip Flops:

One of the most important parts of pong is the actual reflection of the ball off of the walls. With the X and Y position circuits discussed in subsystem #7 being fed into the

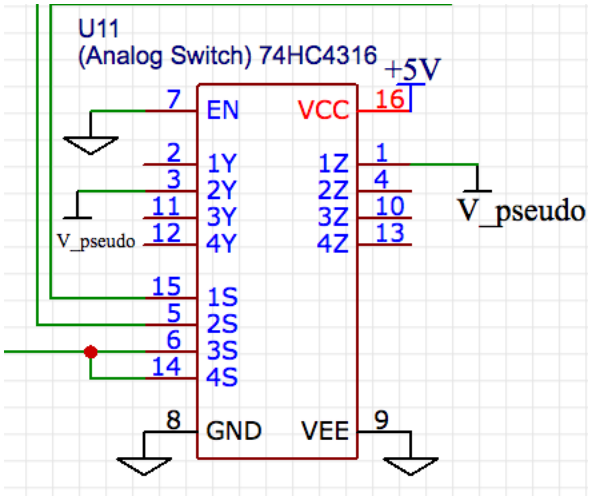


Figure 24: Analog Switch Connections

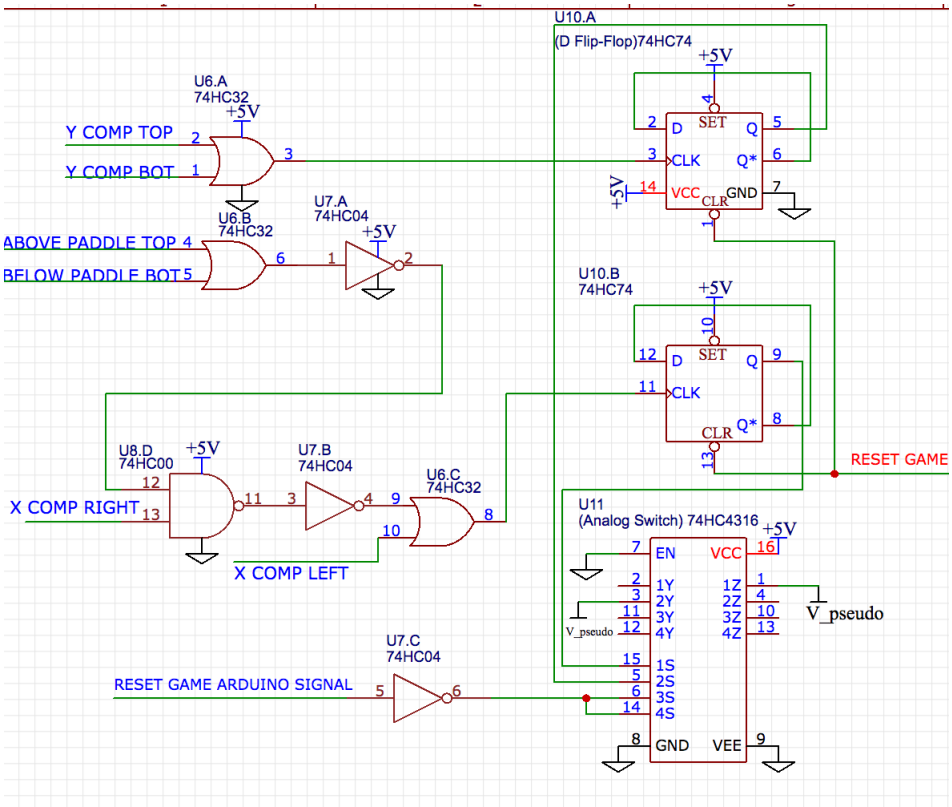


Figure 25: Schematic of the circuit used to determine whether to activate the analog switches of U11.

comparators in subsystem #5, we need to actually interpret and act upon the values output by the comparators. In order to do this, we need combinational logic. Rather than explain the logic in words, below is the series of truth tables that led to the combinational logic

in the above schematic (U6-U8). We use the signals inside of our schematic for ease of understanding:

Truth Tables:

Y COMP TOP	Y COMP BOT	OUTPUT
L	L	L
H	L	H
L	H	H
H	H	X

OR gate

ABOVE PADDLE TOP	BELOW PADDLE TOP	OUTPUT
L	L	H
H	L	L
L	H	L
H	H	L

NOR = OR with INVERTER (since using OR's already)

PADDLE OUT	X COMP RIGHT	OUTPUT
L	L	L
H	L	L
L	H	L
H	H	H

AND = NAND with INVERTER (since using NAND's already)

X COMP LEFT	BELOW PADDLE TOP	OUTPUT
L	L	L
H	L	H
L	H	H
H	H	X

OR gate

Figure 26: The truth tables for the comparator outputs from subsystem #5. Each table is followed by the gate we chose to use to implement the logic.

Though it may seem that the truth tables and the resulting combinational logic is all we need to achieve the desired behavior, we have not yet accounted for feedback. That is, after as a comparator signals the analog switch to flip the ball's velocity in the X or Y direction, the compator's output will revert to its initial state as soon as the ball's velocity flips and it moves away from the wall. To counteract this undesired behavior, we use the flip flops U10.A and U10.B shown in the image at the beginning of this subsystem section. Configured with Q*'s input being fed into D for both U10.A and U10.B (U10.A for the Y direction, U10.B for

the X), and with the comparator logic outputs as inputs to the CLK pin, we ensure that the flip flops reverse upon every rising clock edge. In so doing, we also ensure that analog switch U11 will only reverse the ball's velocity upon a rising edge; that is, the ball's velocity will only reverse upon first contacting the wall. It will not reverse again upon the comparator's output going low. The timing diagram below demonstrates this idea:

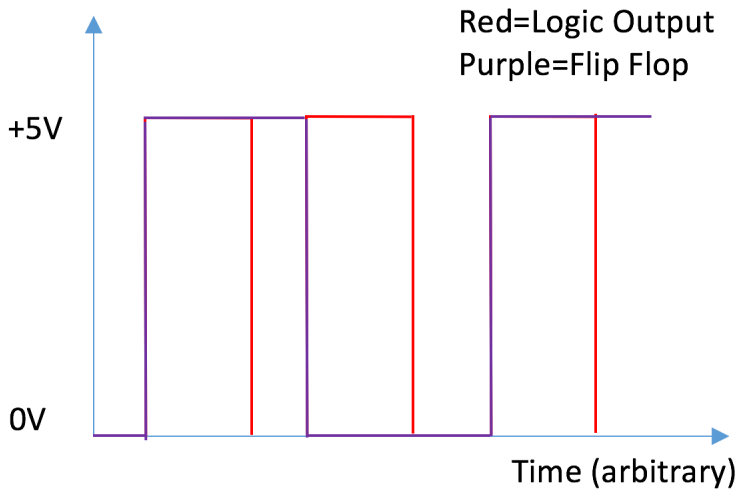


Figure 27: Timing diagram showing how flip flops only respond on rising logic edges.

2.3 Design Narrative

Over the course of the project, our design underwent a few changes. The type of sensor we chose to use changed, our power system was re-worked a few times, and how we planned to interface our circuits together was a continuous discussion. At times it seemed as though our project had a life of its own, with everything falling perfectly into place and working from the get-go, while at other times it seemed like the project never was going to get off of the ground. Coming into the project the consensus was that the ball?s integrative motion would be one of the most difficult parts of the project. We all thought we had a good handle on comparators and micro-controllers, which seemed straightforward initially by comparison, but were worried about inversing the ball based off a signal seemed like it was going to be a challenge. This ended up being almost the exact opposite of our expectations. Without a

doubt, the comparators which were responsible for the bouncing of the ball off of the arena walls and the paddle were a major problem. On our first iteration of the circuit, we used LM311s as a result of our disposition to using parts that we had become familiarized with through lectures and labs. When we were trying to debug our circuit and the comparators, we realized that the LM311s didn't have the proper "Input Common Mode Range"for the functionality that we desired with a Vcc of +5V and a Vee of 0V. The LM311s could only output Vcc - 1.5V which was far below the inputs we were putting into the comparators, and could not output 0V if the BJT in the LM311 was "ON,"being only capable of between Vee + 200mV and Vee + 300mV depending on the temperature of the IC. Upon this realization, we decided to switch to using LM339s which are overall better comparators and come in quad packages which helped to consolidate some of the circuitry. Instead of having 6 LM311s, we could just use two LM339 packages capable of a total of 8 different comparator outputs! We ultimately did end up changing out input voltage to +9V because we were operating at the limits of the comparator by using just +5V and we wanted to ensure that no problems would arise in our comparators which are critical for the functioning of our Pong game as a whole. Common mode input range woes, multiple triggers of the comparator outputs were a major issue. While we had learned that comparators can be prone to multiple triggers to do noise and the hysteresis could be a solution, we failed to realize the importance of preventing multiple, unwanted signals in a circuit. This was especially critical for us, as we were using our comparator outputs as clock inputs for flip flops. After battling with stuck balls for hours, we were finally able to find a range of hysteresis values that would prevent the multiple triggering problems. These problems taught us three very important things: hysteresis is crucial in preventing multiple triggers, it's dangerous to use outputs that are prone to multiple triggers as clocks, and it's very important to be sure that the part you're using is capable of doing what you want it to on the voltage supplies you're running it off of.

2.4 Final Schematic

We have broken our schematic into three parts for ease of viewing. The first part is the arena, the second is the ball/paddle position, and the third is the digital circuitry. All signals are labelled in all CAPITAL letters.

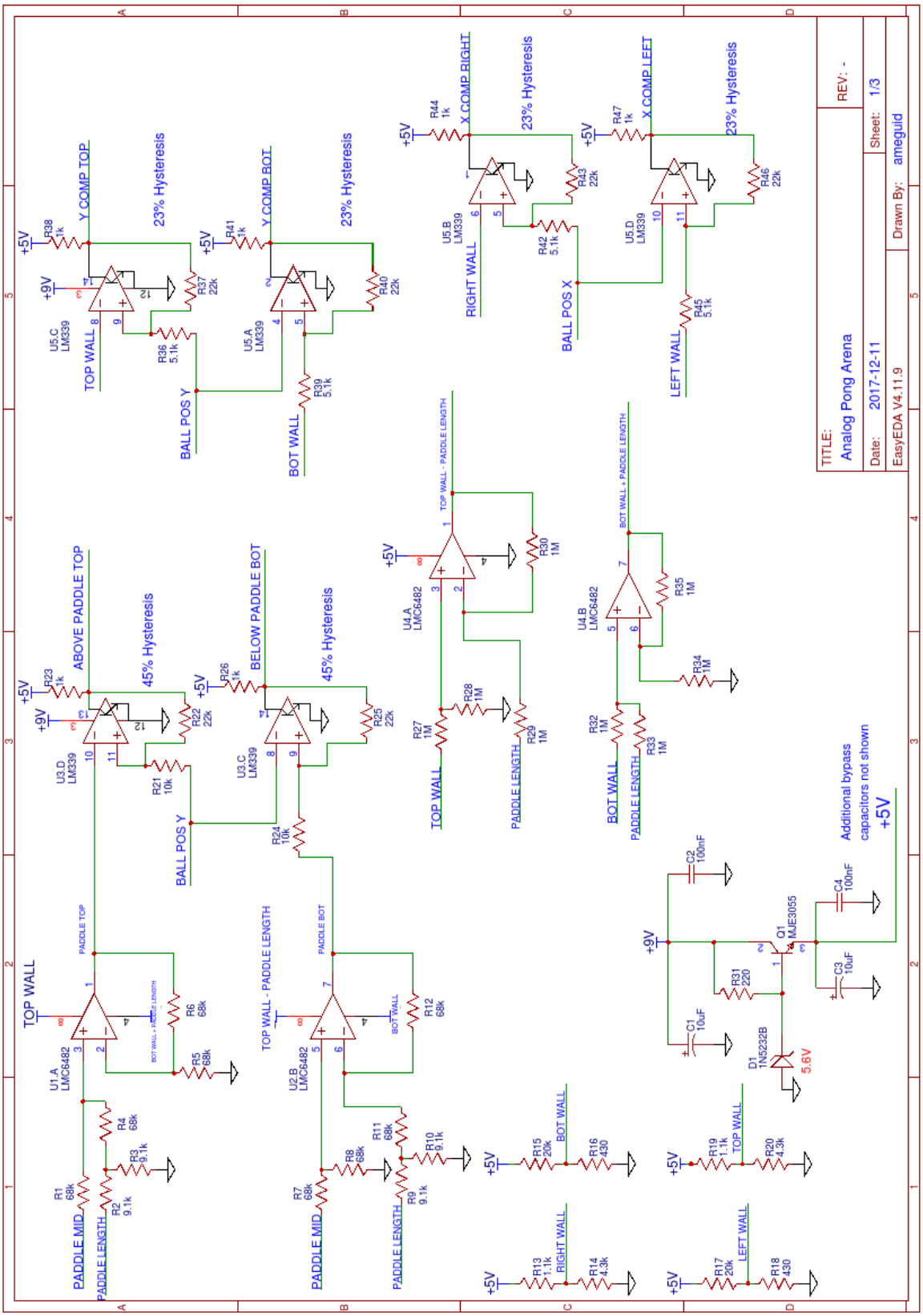


Figure 28: Schematic Page 1

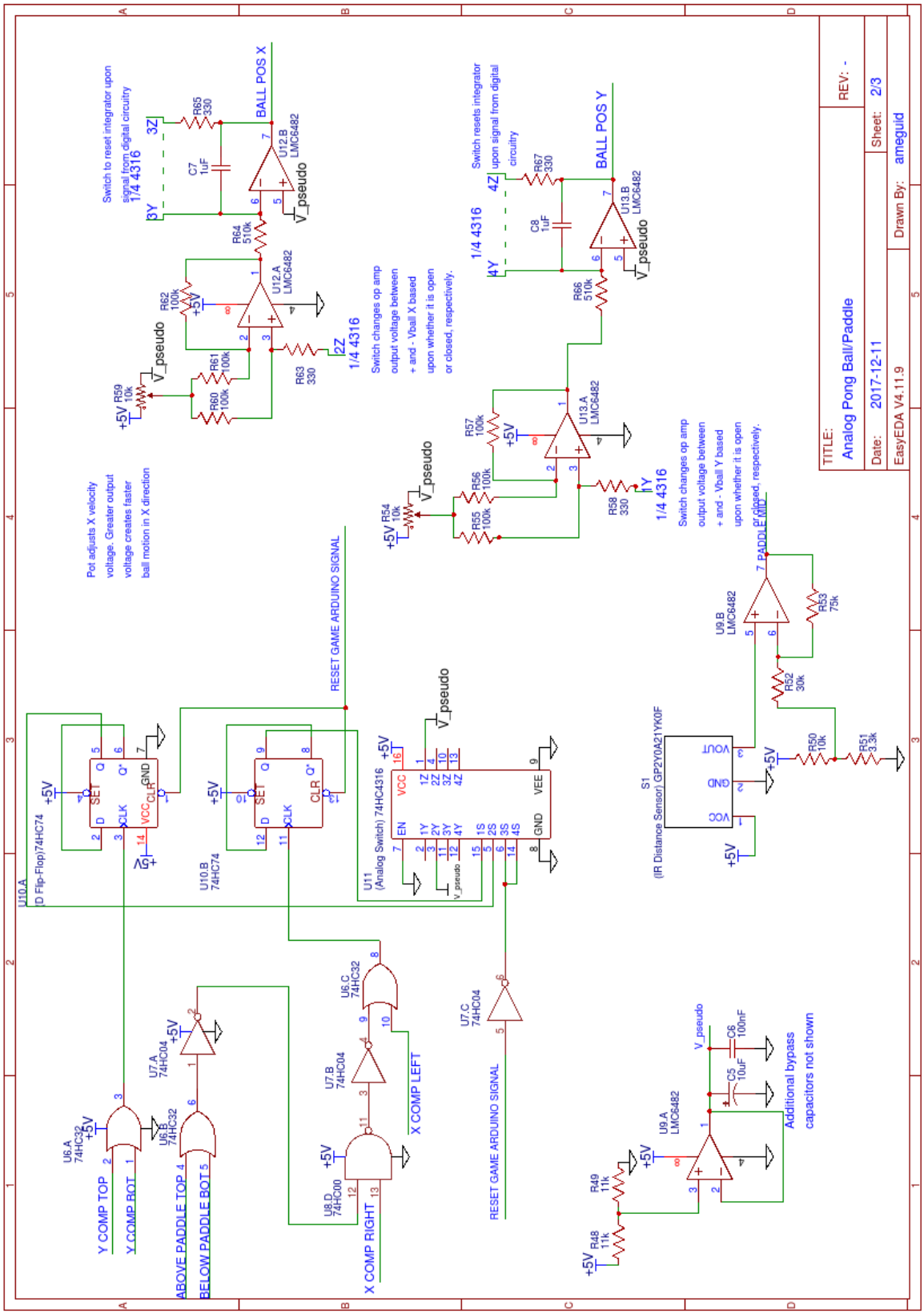


Figure 29: Schematic Page 2

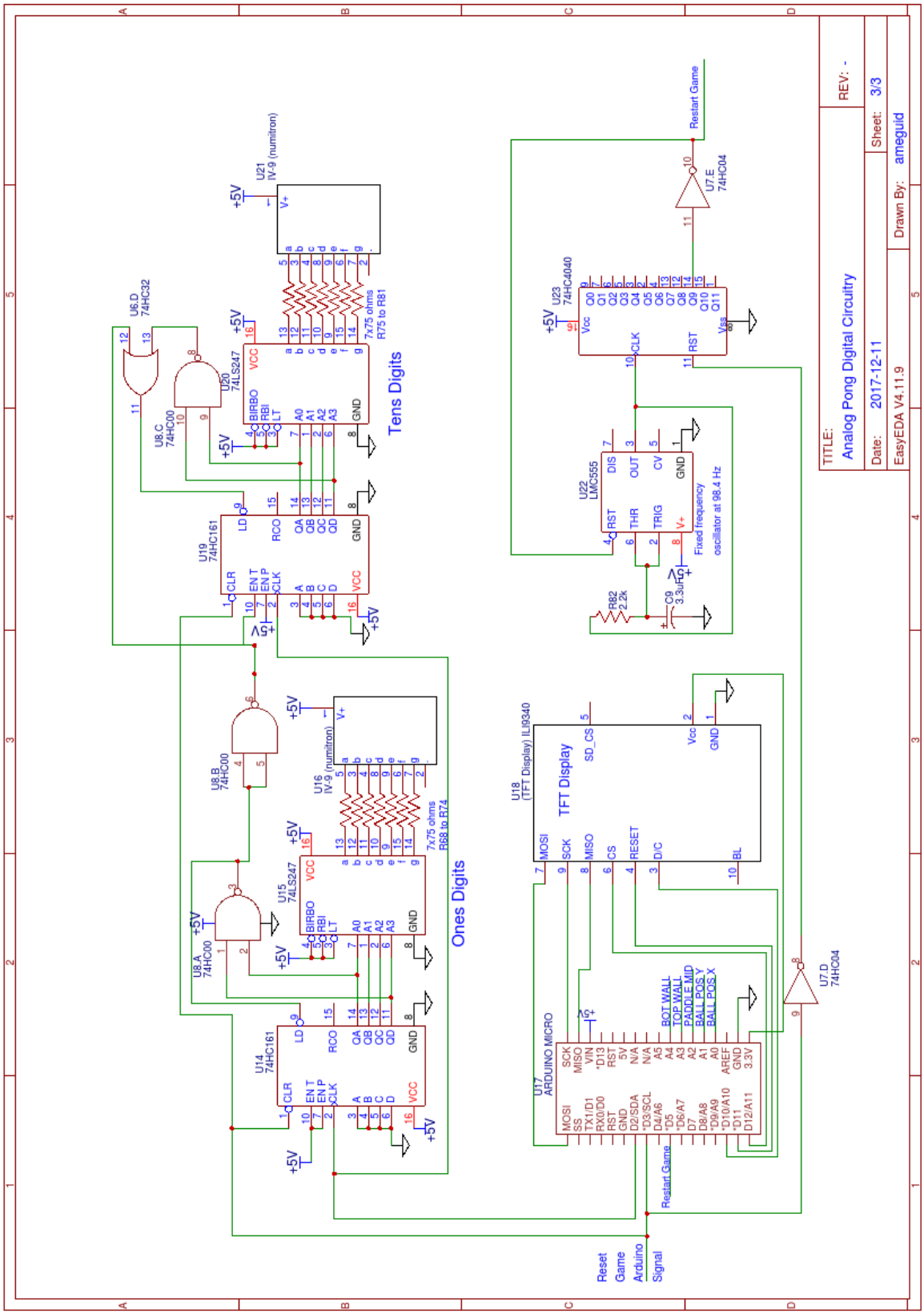


Figure 30: Schematic Page 3

3 Conclusion

The project went without a hitch for the most part. We were able to see move the balls and display movement on them while we were still testing them with switches to serve as the rising clock edges and bounce. The brunt of the problems came when we tried to transfer that functionality over to the comparators. This is where we encountered troubles from a lack of hysteresis and it was one of the biggest learning experiences of the whole final project. We learned that we should (1) make sure that we are using components with the proper input common mode range so that the inputs can actually be compared by the comparator, (2) should always have some hysteresis on our comparators because they are so fast at switching [sometimes so fast that even the oscilloscope couldn't pick up on the switches], (3) rising clock edges can be very dangerous, and (4) that no matter how good you are at planning, little ruts in the road can throw you far off track. Had we not had any problems with hysteresis, we would've loved to been able to try and add more functionality such as having the ball speed increase as the score increases to increase the difficulty of the game, make a circuit that could perform high score storage, adjust the size of the paddle to modify difficulty, increase the highest score, and potentially even add a two-player mode [this would be just for fun as the majority of circuitry would be repetitive].

4 Bibliography

We sought out only the assistance of course staff in the creation of this project, and have credited their circuits where they have arisen (such as the velocity flipping circuit in subsystem #7).

We would like to credit: https://en.wikipedia.org/wiki/Pong for helping us to define Pong in our abstract.

5 Appendix

Name	Quantity	Reference Numbers
(Analog Switch) 74HC4316	1	U11
(D Flip-Flop)74HC74	1	U10
(IR Distance Sensor) GP2Y0A21YK0F	1	S1
(TFT Display) ILI9340	1	U18
1.1k	2	R13,R19
100k	6	R55,R56,R57,R60,R61,R62
100nF	3	C2,C4,C6
10k	3	R21,R24,R50
10k Potentiometer	2	R54,R59
10uF	2	C1,C3,C5
11k	2	R48,R49
1k	6	R23,R26,R38,R41,R44,R47
1M	8	R27,R28,R29,R30,R32,R33,R34,R35
1N5232B	1	D1
1uF	3	C7,C8
2.2k	1	R82
20k	2	R15,R17
220Ω	1	R31
22k	6	R22,R25,R37,R40,R43,R46
3.3k	1	R51
3.3uF	1	C9
30k	1	R52
330Ω	4	R58,R63,R65,R67
4.3k	2	R14,R20
430Ω	2	R16,R18
5.1k	4	R36,R39,R42,R45
510k	2	R64,R66
68k	8	R1,R4,R5,R6,R7,R8,R11,R12
74HC00	1	U8
74HC04	1	U7
74HC161	2	U14,U19
74HC32	1	U6
74HC4040	1	U23
74LS247	2	U15,U20
75k	1	R53
9.1k	4	R2,R3,R9,R10
ARDUINO MICRO	1	U17
IV-9 (numitron)	2	U16,U21
LM339	2	U3,U5
LMC555	1	U22
LMC6482	6	U1,U2,U4,U9,U12,U13
MJE3055	1	Q1

Figure 31: Our Bill of Materials for the Entire Project.

Software

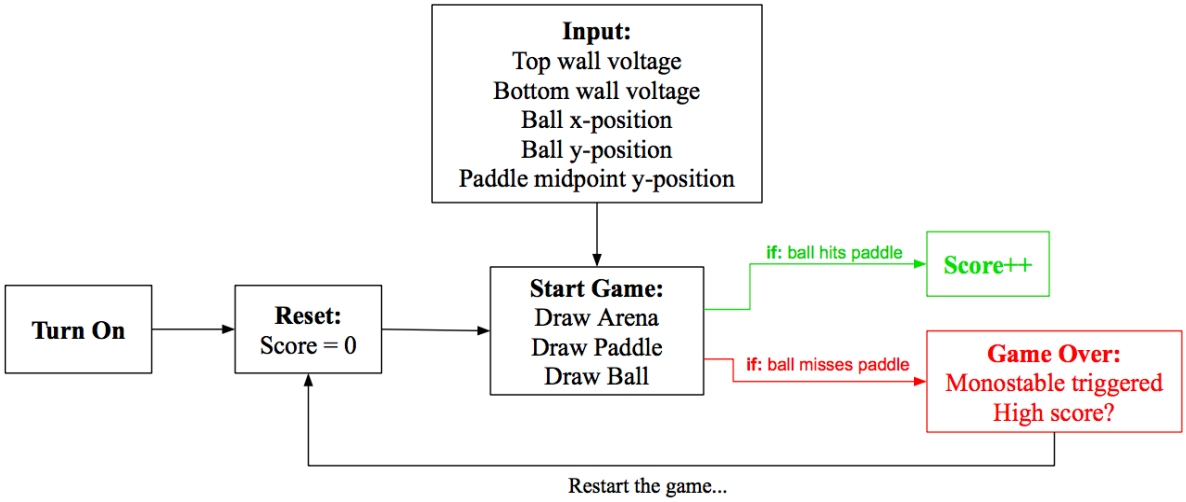


Figure 28
Software Flow-Chart

When the Arduino is first connected to power, and prior to the running of the setup code, I declared handful of global variables that were used throughout the different functions of the circuit and to drive the display. The primary purpose of the Arduino Micro is to display the analog circuitry's output onto a 2.2" TFT color display so the user has useful feedback for proper gameplay. Some of the functionality of the game, such as scoring is controlled by the microcontroller, but we'll touch on that a bit later.

The header of the document was where I setup my pinouts for all parts of the microcontroller that would be interacting with the analog parts of the circuit and outputting to the display and digital circuitry. The Adafruit_ILI9340 library provided me with the functions

necessary to manipulate the TFT display. Below I show how the display was connected to the Arduino.

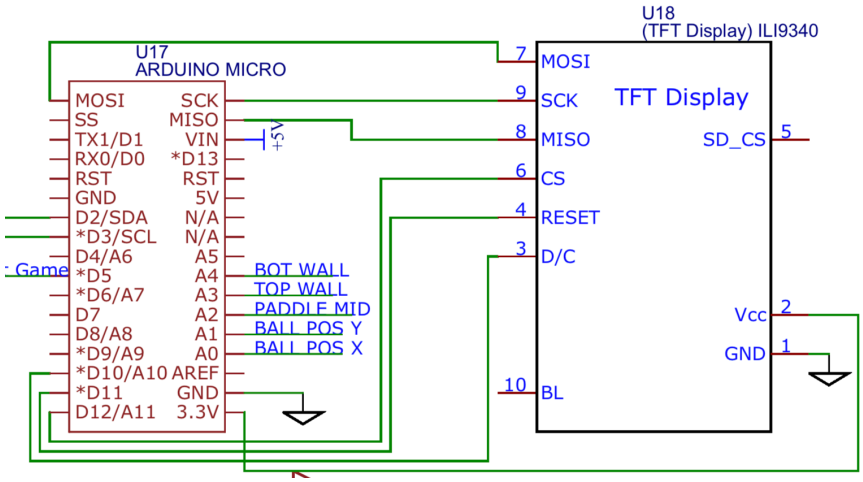


Figure 29
Pinout from the Arduino to the TFT 2.2" 240x320 pixel ILI9340 Display

In the setup code, all of the digital input and output pins were declared which allows the program to use them later to write and read data from. I did not initialize the analog pins because that is unnecessary in Arduino sketches. The setup code is also where I set up the display for output using the function tft.begin(); which comes from the Adafruit_GFX library.

In the setup code, I used one of the variables that I declared earlier in my global variable and constant declarations – pause. Pause is a boolean which allows for the stopping of gameplay in the instance of a reset or a game over. The function drawPaddle(), drawBall(), and gameOver() are all conditioned on this boolean so if pause is true, the game stops and the paddles and balls are not drawn. This is indicative of a reset or game over. If pause was false, the gameplay would resume and the user can move the paddle and score points by hitting the ball with the paddle. Additionally, in the setup code, I reset the high score to 0 to ensure that there wouldn't be a random integer that took the place of my variable highScore which stores the high

score in it. I then restart the game which draws the borders of the Pong arena and gets the display ready for its input values.

My loop code was pretty simple with only two true lines in it…

(1) drawPaddle(!pause);

(2) drawBall(!pause);

The simplicity of this loop is a direct result of the use of functions and it does a great job of consolidating the code.

My first function was initially used in the setup code – restartGame(). This function as previously stated, is responsible for setting up the display for gameplay. The arena is drawn with walls of the thickness I declared in my constant declaration in the header of the sketch. This function also reset the user's current score to 0 and outputs a *reset* signal to the rest of the circuit to reset. This include sending a ACTIVE LOW signal to the score counters the Numitrons use to display the score, the D Flip-Flops which are ACTIVE LOW to reset their OUTPUT to LOW, and through an inverter, U7.C, to the Analog Switch which is ACTIVE HIGH. Sending the reset signal to all of these components resets the integrators and as a result, the ball's position.

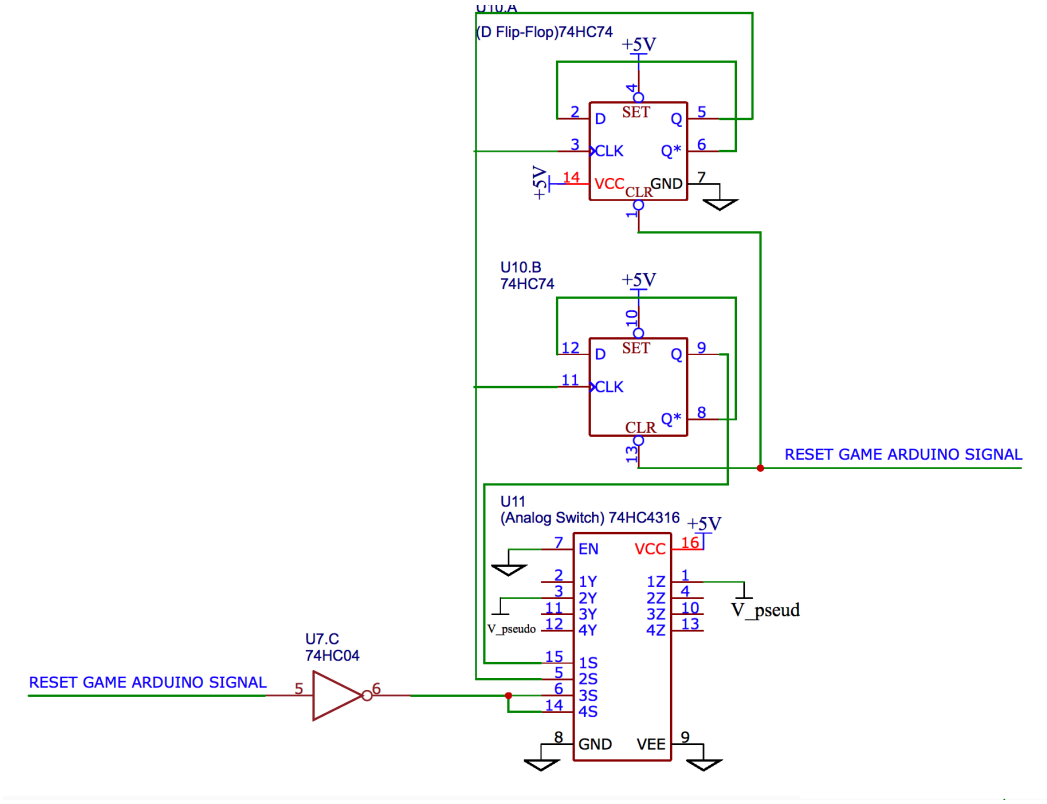


Figure 30
Arduino RESET signal interfacing with ball's position

My second function drawPaddle. It is used to draw the paddle that the user controls using the IR sensor. In order to ensure that the paddle that is drawn on the display is equal in size to that of the actual voltages, I use the analogRead() function to read in what the top and bottom wall's reference voltages are. I then take these values and multiply them by (5.0/1023.0) in order to convert these values from the Arduino sensor read values into actual voltages.

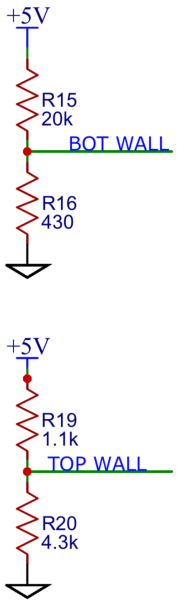


Figure 31
Bottom and Top Wall Reference Voltage read by the Arduino to map the paddle's height and position along with the ball's y-position mentioned below.

I then use these voltages to resolve what the height of the paddle is going to be in terms of # of pixels by taking 240, the height of the display, and multiplying it by (1/(topWall-botWall)) in order to find what proportion of the screen that the paddle is going to be mapped to. Next, I read in the paddle's midpoint y-position using analog read from a clipping op amp's output which ensures that the paddle stays within the bounds of the screen.

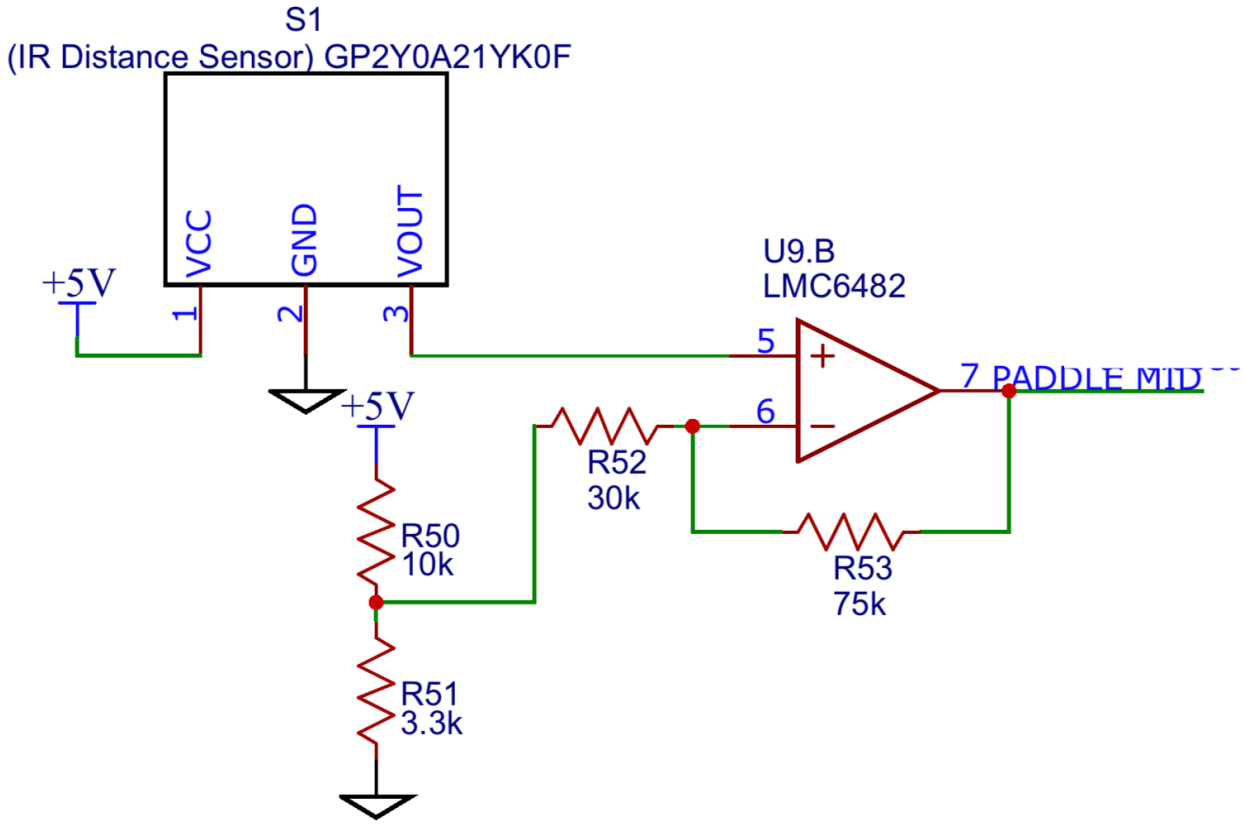


Figure 32
Schematic showing the paddle's midpoint position that was read by the Arduino.

I then use the Arduino map function to make sure that the sensor values that I am getting from the top and bottom walls are the values from which the paddle can move between. I then constrain the yPaddle's position just to ensure that if the sensor value is a little off that the paddle still remains on the screen. I drew the paddle using a white rectangle of the same thickness as the arena borders' for consistency and the same height as the difficulty that was calculated earlier using the top and bottom wall voltages. One of the problems that I ran into while testing this code was that the paddle wouldn't move smoothly across the screen. After some research, I came up the YouTuber, Dejan Nedelkovski. Reference: https://www.youtube.com/watch?v=jPU4iv378ig. He helped me figure out that add a cushion

around the paddle would make the movement of the paddle appear smooth as it wrote over the black space you had put around it so I have a cushion around the paddle that I draw.

The next function that I created was the drawBall() function which read in the ball's x and y position from the integrators. I then mapped the ball using the top and bottom wall analog read values that I had acquired through the drawPaddle() function to map the y position of the ball and I used the known values of the voltages to create the left and right wall limits for the ball so it would be map the x position of the ball properly.

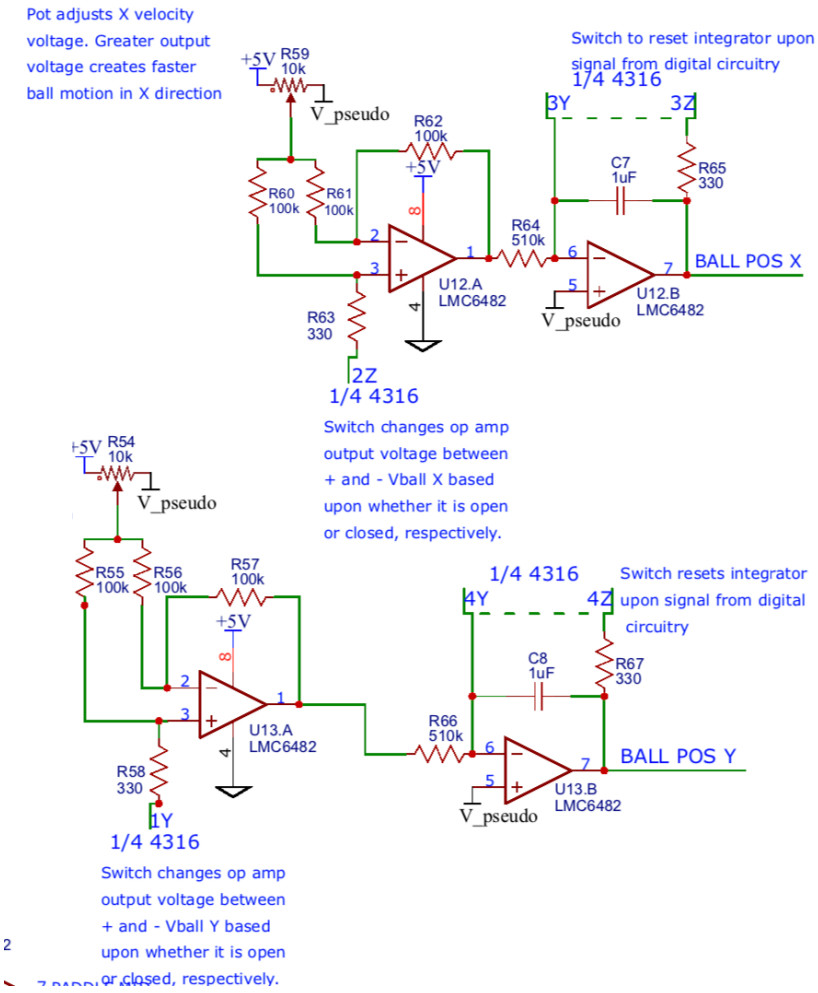


Figure 33
Schematic shows the ball's x and y position that was read in by the Arduino

I chose a radius of 7 for the ball because that was the most aesthetically pleasing where it wasn't too small for the user to squint to see it, but also not proportionality large for the small display we had. I also used a cushion for the ball because I was having issues with trailing from the ball as well. In the drawBall function, I also included the functionality of the ball, such as incrementing the score by one on a paddle hit which can be found the the if look inside of the drawBall function. We found that in testing, the ball would tend to double count on a paddle hit so to resolve this I used the timers current time to ensure that there was only one increment to the score on a paddle hit. If there was a paddle hit, I would send out a HIGH counter signal which was a rising clock edge to the 74HC161 counters which would result in the score increasing and the Numitrons displaying the new current score of the user.

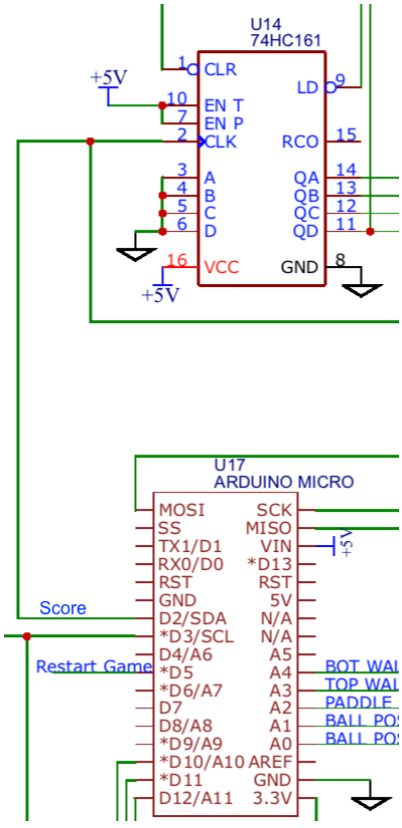


Figure 34
Schematic showing the score output signal that is sent to the clock of the ones digit of the Numitron. The tens digit is not show, see final schematic for full view.

Additionally, the drawBall function would calculate if the ball had gone beyond the paddle at which point pause was declared as true stopping the drawing of the new position of the ball and paddle and the gameOver function would take over.

The gameOver() function would start off by clearing the screen and comparing the user's current score with the previous high score. If the user had beat the previous high score, a congragulations message would be displayed. If not, the screen would clear and a monostable would be triggered causing it to go HIGH. GAME OVER would then be displayed along with the high score and a countdown of the amount of time left on the monostable. The countdown would continue for 5 seconds until the output of the monostable would go low again at which point pause would become false and the game would restart just to repeat again. See the section on monostables for a further explanation.

```
/*
  ES52 Final Project: Analog Pong

  Name: Pong_Display.ino
  By: Adham, Austin and Enxhi.
  Date: November 30th, 2017
  --- Credit to Adafruit for TFT initialization code ---

  This is an Arduino sketch which takes in analog signals from analog
  circuitry - which control the position and velocity of the ball, position
  and movement of the paddle, and reference voltages for the arena boundaries
  - and displays these signals on a TFT 2.2" 240x320 ILI9340 display produced
  by Adafruit.

  The display shows gameplay in a user-friendly manner through a colorful and
  beautiful display. Additionally, input from the analog circuitry is used to
  register the scoring of a point, which occurs when the ball hits the paddle,
  triggering a HIGH signal to digital pin 2 incrementing 74HC161 counters
  outputting their count to 2 Numitrons thus displaying scores from 0 - 99 points.

  mru: Enxhi Buxheli 12/9/2017 - cleaning up the code
*/

// Adding libraries
#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9340.h"

// Header guard to prevent dual imports from GFX and ILI9340 libraries
#if defined(__SAM3X8E__)
    #undef __FlashStringHelper::F(string_literal)
    #define F(string_literal) string_literal
#endif

// I/O Pin Constants
#define _sclk SCK   // Display: Serial clock pin
#define _miso MISO  // Display: MISO pin
#define _mosi MOSI  // Display: MOSI pin
#define _cs 12      // Display: CS pin
#define _dc 10      // Display: DC pin
#define _rst 11     // Display: RESET pin

#define ballX A0        // Ball: x-position
#define ballY A1        // Ball: y-position
#define paddleY A2      // Paddle: y-position of midpoint
#define wallTop A3      // Wall: top
#define wallBot A4      // Wall: bottom
#define counter 2       // Counter: score signal output to counters
#define reset 3         // Game: Reset signal
#define restart 5       // Restart: signal from the monostable that triggers the game over

#define PADDLEBOUNCE 4  // DEBUG PADDLE: test of game prior to full functionality

// Program CONSTANTS
const boolean debug = false; // DEBUG TESTING IF TRUE
int thickness = 10; // Constant thickness of boundaries and paddle

// Program Variables
int topWall;        // Variable that stores analog read of the top wall
int botWall;        // Variable that stores analog read of the bottom wall
int difficulty;     // The paddle's height
```

```
int yPaddle;        // Midpoint of paddle's y-position
int score;          // Current game score
int highScore;      // High score storage
boolean pause;      // Pausing of the game on a game over

long lastMillis;    // DEBUG SCORING: time of last hit
long curMillis;     // DEBUG SCORING: time of current hit

// Hardware SPI - display runs faster than software equivalent
Adafruit_ILI9340 tft = Adafruit_ILI9340(_cs, _dc, _rst);


void setup() {
  // Initialize our I/O pins
  pinMode(counter, OUTPUT);
  pinMode(reset, OUTPUT);
  pinMode(_cs, OUTPUT);
  pinMode(_dc, OUTPUT);
  pinMode(_rst, OUTPUT);
  pinMode(restart, INPUT);

  // DEBUG testing game functionality
  if (debug == true){
    pinMode(PADDLEBOUNCE, OUTPUT);
    digitalWrite(PADDLEBOUNCE, LOW);
  }

  // Setup serial connection
  Serial.begin(9600);

  // Begin use of display
  tft.begin();
  Serial.println("TFT initiated");

  // Restart game: draw arena borders
  restartGame();

  // On complete reset of the game, high score reset to 0
  highScore = 0;

  pause = false; // Allows game to start, if true, game stops
}

void loop(void) {
  // Draw the movement of the paddle throughout the game while pause is false
  drawPaddle(!pause);

  // Draw the movement of the ball throughout the game while pause is false
  drawBall(!pause);
}



/*********************************************************
 * void restartGame()
 *
 * Function to restart the game when turned on or when
 * reset button pressed upon a "GAME OVER" or whenever
 * the user feels like pressing the reset button. This
 * also resets the counters and the ball's position.
 *********************************************************/
```

```
void restartGame(){
  Serial.println("GAME RESET");

  // Send a reset signal so the score and ball position resets
  digitalWrite(reset, LOW);   // reset signal: resets counters and ball position
  delay(50);                  // delay in place for numitron reset [too fast o.w.]
  digitalWrite(reset, HIGH);  // reset is Active Low
  score = 0;                  // reset score to 0 in code

  // Clear screen [LEFT-RIGHT] to reset it for gameplay
  tft.setRotation(2);
  tft.fillScreen(ILI9340_BLACK);

  // Rotate the display so (0,0) is in top left corner
  tft.setRotation(3);

  // Drawing arena with chosen thickness of pixels
  tft.fillRect(0, 0, tft.width()-50, thickness, ILI9340_CYAN);                       // top wall
  tft.fillRect(0, 0, thickness, tft.height(), ILI9340_CYAN);                         // left wall
  tft.fillRect(0, tft.height()-thickness, tft.width()-50, tft.height(), ILI9340_CYAN);// bottom wall
}


/*********************************************************
 * void drawPaddle()
 *
 * Function to draw the paddle as the user input from the
 * IR sensor changes. This function allows for the smooth
 * movement of the paddle by compensating for the slow
 * refresh rate of the TFT display with a cushion around
 * the paddle.
 *********************************************************/
void drawPaddle(boolean){
  // Reference voltages to be used in mapping
  topWall = analogRead(wallTop);
  botWall = analogRead(wallBot);

  // Changing the analog read values into voltages
  float topWallVolts = topWall * (5.0 / 1023.0);
  float botWallVolts = botWall * (5.0 / 1023.0);

  // Draws a paddle which is modified by the changes in arena size
  difficulty = 240 * (1 / (topWallVolts-botWallVolts));

  // Read in analog paddle position
  yPaddle = analogRead(paddleY);
  yPaddle = map(yPaddle+difficulty/2, botWall, topWall,  0, 239); // maps analog values to screen
  yPaddle = constrain(yPaddle+difficulty/2, 0, 240-difficulty/2); // constrain to the screen

  // Drawing the actual paddle
  tft.fillRect(tft.width()-50, yPaddle-difficulty/2, thickness, difficulty, ILI9340_WHITE);

  // Compensate for slow refresh rate by having a black cushion around the paddle
  tft.fillRect(tft.width()-50, yPaddle+difficulty, thickness, 10, ILI9340_BLACK); // below paddle
  tft.fillRect(tft.width()-50, yPaddle-difficulty, thickness, 10, ILI9340_BLACK); // above paddle
}

/*********************************************************
 * void drawBall()
 *
 * Function to draw the ball throughout the game as it
```

```
 * moves around the screen and bounces. The ball is
 * confined by the limits of the screen and the boundaries
 * of the arena. It has a black cushion around it so the
 * display shows smooth movement of the ball without a
 * trail.
 ******************************************************/
void drawBall(boolean){
  // ball position values to be mapped
  int xBall = analogRead(ballX);
  int yBall = analogRead(ballY);

  int radius = 7; // most aesthetically pleasing ball radius

  // ball mapped to the limits of the display - map(val, lowVal, highVal, lowDisp, highDisp)
  xBall = map(xBall, (0.1/5.0)*1023, (4.0/5.0)*1023, 2*thickness+radius-3, 319);
  yBall = map(yBall, botWall, topWall, 2*thickness+radius+1, 239-5*thickness-radius);

  // drawing ball - function: tft.fillCircle(x, y, radius, color);
  tft.fillCircle(xBall, yBall, radius, ILI9340_RED);

  // compensate for slow refresh rate by having a black cushion around the paddle
  for (int k = 1; k<=5;k++){
    tft.drawRect(xBall-radius-k, yBall-radius-k, (radius+k)*2, (radius+k)*2, ILI9340_BLACK);
  }

  curMillis = millis(); // stores the current time to avoid double counting of the score

  // increment the score by 1 when the ball hits the paddle
  if ((yBall < yPaddle+(difficulty/2) && yBall > yPaddle-(difficulty/2)) && ((xBall > (tft.width()-50-
thickness)) && (xBall < tft.width()+thickness-50)) && curMillis > lastMillis+250){
    digitalWrite(counter, HIGH);      // incrementing counter
    digitalWrite(counter, LOW);
    score++;

    lastMillis = millis();
  }

  // adding paddle functionality when debugging
  if (debug == true){
    curMillis = millis();

    // increment score by 1 when ball hits the paddle
    if ((yBall < yPaddle+(difficulty/2) && yBall > yPaddle-(difficulty/2)) && ((xBall > (tft.width()-50-
thickness)) && (xBall < tft.width()+thickness-50)) && curMillis > lastMillis+250){
      digitalWrite(PADDLEBOUNCE, HIGH); // FAKING BOUNCE
      digitalWrite(counter, HIGH);      // incrementing the score
      digitalWrite(counter, LOW);
      score++;

      lastMillis = millis();
    }
    else{
      digitalWrite(PADDLEBOUNCE, LOW); // FAKING BOUNCE
    }
  }

  // display GAME OVER when ball passes paddle
  if(analogRead(ballX) > 920 && analogRead(ballX) < 950){
    pause = true;
    gameOver(pause);
  }
```

```
}


/*******************************************************
 * void gameOver()
 *
 * Function to clear the display in order to display the
 * GAME OVER message to the user which tells them that
 * they lost the game. This message is sent out when the
 * user fails to hit the ball with the paddle.
 *******************************************************/
void gameOver(boolean){
  Serial.println("GAME OVER");

  // Rotate screen to clear screen [LEFT-RIGHT] to prepare for GAMEOVER message
  tft.setRotation(2);
  tft.fillScreen(ILI9340_BLACK);
  Serial.println("Screen reset");

  // Figure out if high score has changed
  if (score > highScore){
    highScore = score;

    // Rotate the display so (0,0) is in the top left corner
    tft.setRotation(3);

    // Display a congragulations message for 3 seconds
    tft.setTextColor(ILI9340_YELLOW);
    tft.setTextSize(5);
    tft.setCursor(25,25);
    tft.println("CONGRATS!");
    tft.setCursor(25,70);
    tft.println("NEW HIGH");
    tft.setCursor(45,115);
    tft.println("SCORE!!!");
    delay(3000);

    tft.setRotation(2);
    tft.fillScreen(ILI9340_BLACK);
  }

  // Rotate the display so (0,0) is in the top left corner
  tft.setRotation(3);

  // Display GAME OVER message
  tft.setCursor(25, 85);            // Aesthetics of message
  tft.setTextColor(ILI9340_GREEN);  // Aesthetics of message
  tft.setTextSize(5);               // Aesthetics of message
  tft.println("GAME OVER");

  // Display HIGH SCORE
  tft.setCursor(35, 135);           // Aesthetics of message
  tft.setTextColor(ILI9340_RED);    // Aesthetics of message
  tft.setTextSize(3);               // Aesthetics of message
  tft.print("HIGH SCORE: ");
  tft.println(highScore);

  // send out reset signal to start monostable
  digitalWrite(reset, LOW);
  digitalWrite(reset, HIGH);
```

```
  // stay in gameover until the monostable triggers a reset
  while(digitalRead(restart) == HIGH){
    for (int j = 5; j > 0; j--){
      tft.setCursor(5, 165);          // Aesthetics of message
      tft.setTextColor(ILI9340_CYAN); // Aesthetics of message
      tft.setTextSize(2);             // Aesthetics of message
      tft.print("Game will restart in... ");
      tft.print(j);
      delay(1000);
      tft.fillRect(285, 165, 320, 50, ILI9340_BLACK);
    }
  }
  // allow for the game to restart from game over
  pause = false;
  restartGame();
}
```

Score Keeping:

The user's current score is displayed on a Numitron display which increment by one each time the ball bounces off of the paddle. The Numitrons display the user's score throughout gameplay through the use of two 74HC161 3-bit binary counters with asynchronous clear and two 74LS247 Display/Decoder Drivers that decode the binary counters' count into the 7 segments that light up on the Numitrons' 7 different segments. The output of the 74LS247s go through seven 75\si{\ohm} current limiting resistors each to ensure that each segment of the Numitron never exceeds its maximum current threshold of 20mA.

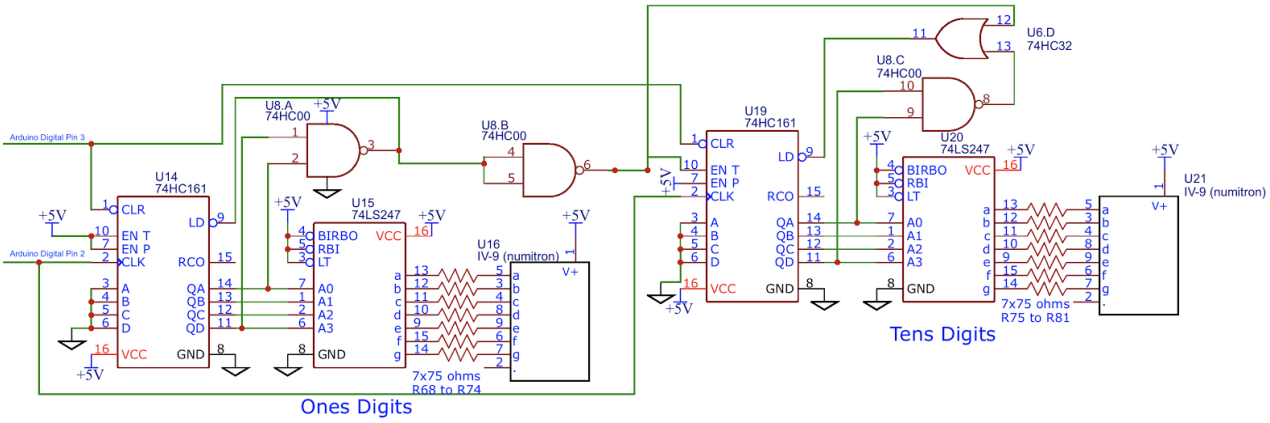


Figure X
Schematic of Numitron Display

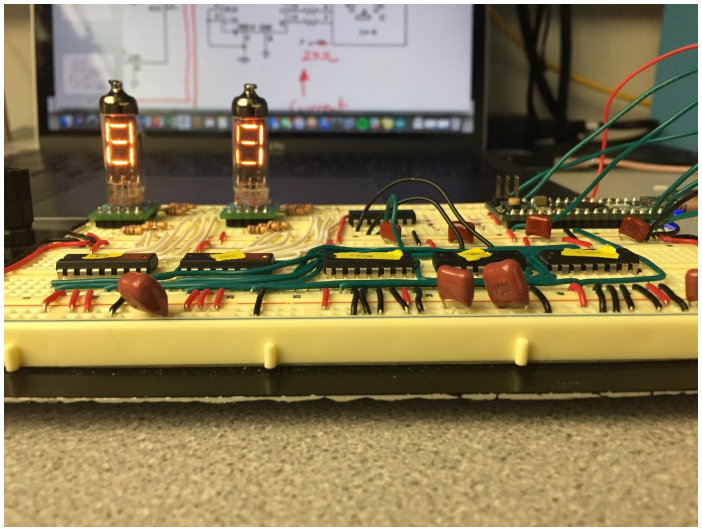


Figure X
An image showing the Numitron display at a count of 88 to display full functionality.

Each time the user scores by hitting the ball with the paddle, the microcontroller sends a HIGH signal from its digital pin 2, quickly followed by a LOW signal. This counter output coming from digital pin 2 of the microcontroller is connected to both the ones and tens digit CLOCKs of the 74HC161 counters which causes the ones digit counter, U14, to increment by one on a HIGH CLOCK signal. Once counter U14 reaches a count of 9, NAND gate U8.A is used to send a LOW signal to the active LOW LOAD pin of the counter which loads the value 0 into the counter thus starting the ones digits again at 0. The NAND gate output of U8.A is also connected to NAND gate U8.B which acts as an inverter sending a HIGH ENABLE signal to the tens digit counter when the the ones digit is rolling back to zero. This results in the tens digit 74HC161 counter, U19, to increment by one every time the ones digits roll back to 0.

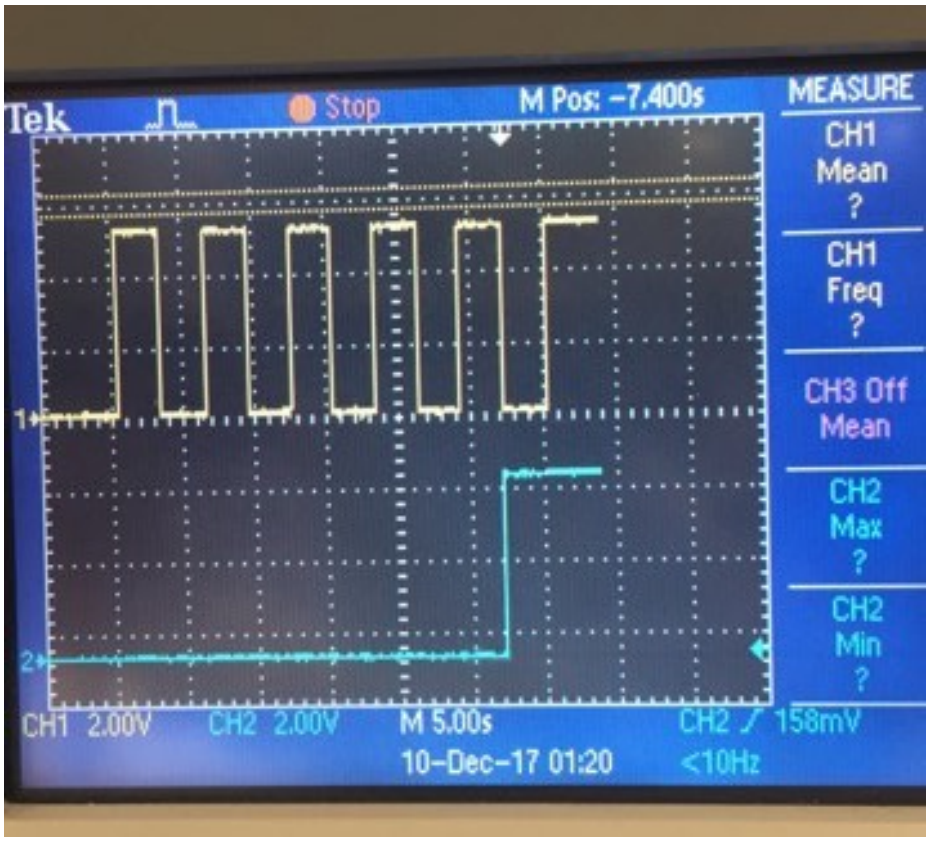


Figure X
An image showing counter output of 74HC161s as they count from 0-11
***Yellow** = Ones digit counter probed at the binary value for 1*
***Blue** = Tens digit counter probed at the binary value for 1*

In order to avoid too much repetitive circuitry, we used just two Numitron displays, counters, and display decoder/drivers. To avoid overflow of the counters which would result in garbage values being displayed on the Numitron, we used OR gate U6.D with inputs coming from NAND gates U8.B and U8.C. The output of this OR gate is connected to the active LOW load of the tens digit 74HC161 counter - U19. Shown below is the truth table for OR gate U6.D whose results show that when both the tens and ones digits are 9, they will both load a value of 0 on the next rising edge of the clock. The Numitrons count from 0-99 and roll back to 0 on overflow, but the Arduino stores the value of the current score and uses that to calculate if the user got a new high score.

U8.B Output (Ones NAND)	U8.C Output (Tens NAND)	U6.D Output (OR gate)
0	0	0
0	1	1
1	0	1
1	1	1

Figure X
A table showing the truth table of the OR gate whose inputs are the outputs of the ONEs and TENs digit NAND gate which go LOW when the divide by 10 counters try to increment past 9. Highlighted are the values that result in the TENs digit counter loading a value of 0.

We connected the two 74HC161s, which are synchronous counters with asynchronous clears, to the same Arduino clock signal for the incrementation of the user's score to allow for seamless synchronous counting. Additionally, when a reset signal the is sent from the Arduino when the function gameOver() in the code is called upon a user missing the ball with their paddle, since the 74HC161s have asynchronous clear, we have connected their active LOW CLR inputs to the reset signal. The asynchronous clear is the primary reason for choosing the 74HC161s over the 74HC163s which has a synchronous clear. The asynchronous clear allows for us to reset the counters to 0 at any point without having to depend on a clock signal to reset them.

Resets:

Upon a restart of the Arduino Micro or a Game Over, we wanted the game to easily reset. On the software side of things, we used a restartGame() function in order to reset gameplay from a full on reset, but we wanted to add a monostable so a player of our Pong game wouldn't have to continuously restart the game if they wanted to play again.

After a couple of test runs with the game fully running, we tried out a couple different time values. First, we tested the monostable at 10 seconds which after a couple runs felt much too long. Next, we tried a 5 seconds monostable which, after a few gameplays, felt pretty reasonable. Just to make sure that we had found the ``right'' value we tried a step below at 2.5 seconds just to see if approximately 5 seconds was the right amount of time before the game restarted. At 2.5 seconds, the user barely had any time to react to seeing the high score and preparing for the next round of Pong so we reverted back to the 5 second monostable depicted below.

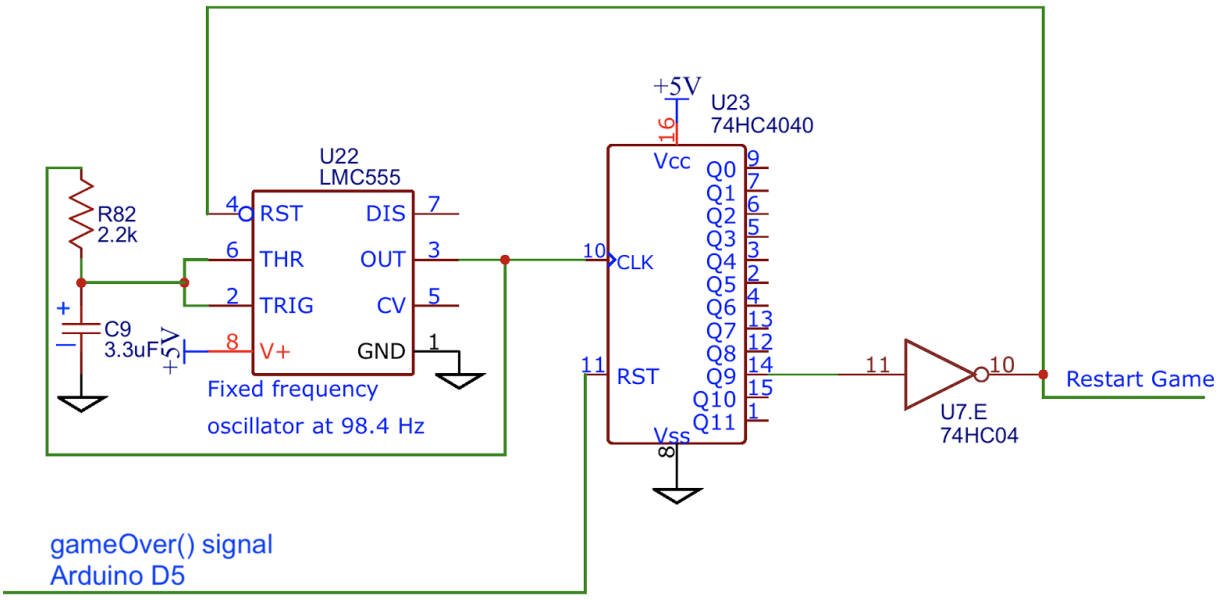


Figure X:
Schematic of the monostable with the LMC555 operating at frequency of 98.4Hz

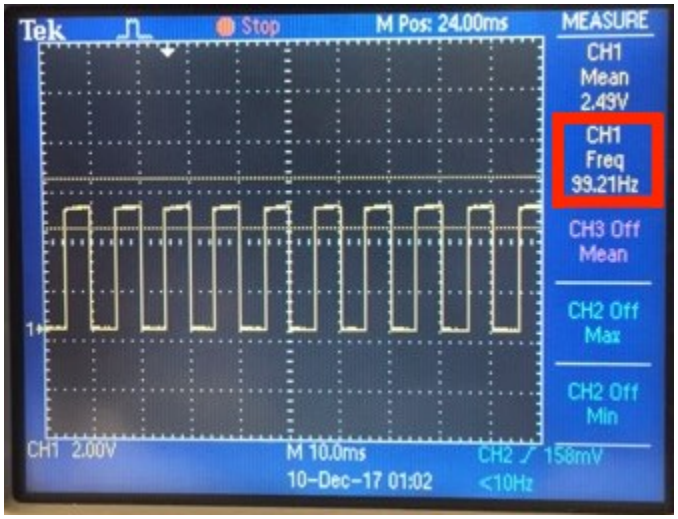
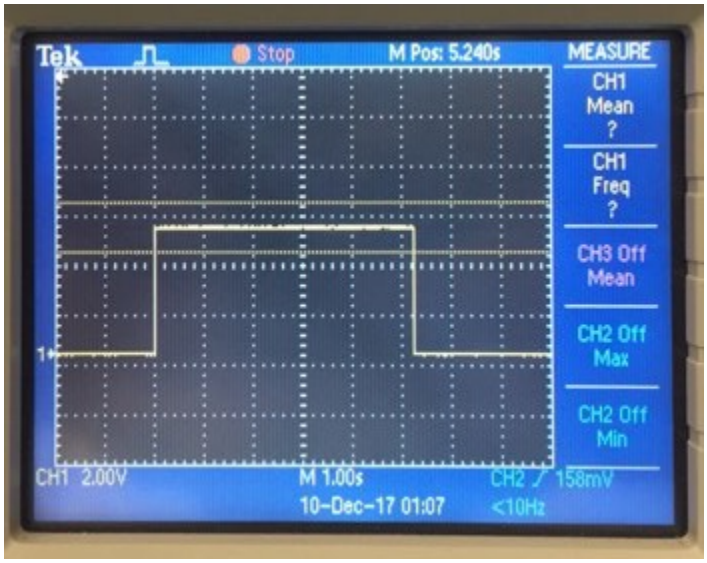


Figure X
An image showing the clock output of the LMC555 timer. Boxed in red is the measured frequency.
***Yellow** = LMC555 OUTPUT pin 3*

The monostable is triggered on a ``*Game Over*'' at which point the LMC555 starts to send out clock signals to the 74HC4040. We calculated that to get an approximately 5 second long monstable, we could use a approximately 100Hz clock with Q9 being the output pin of the 74HC4040 which ultimately triggers a restart of the game that is sent to the Arduino at which point the Arduino executes the restartGame() function resetting gameplay.

The choice to use an approximately 100Hz clock to get an approximately 5 second output of the monostable was made because the output of the 74HC4040 was greatly simplified at these values. Since the output of the 74HC4040 occurs at a clock cycle on the order of $2x$, the math and circuitry would be simplified if I just used the Q9 output of the 74HC4040 to get $29 = 512$ clock cycles. Taking the number of clock cycles until an output and just dividing by approximately 100 made it easy to get a HIGH output for approximately 5 seconds.



1100Hz29 clock signals = 5.12 seconds

Figure X
An image showing the HIGH output of Q9 (pin 14 of 74HC4040) for approximately 5 seconds which corresponds to the predicted output shown by the calculation above.
***Yellow** = 74HC4040 Q9 output*