CECS 490B Final Report

By
Nicholas Bishop
Joshua Rodriguez
Carlos Verduzco

December 14, 2022

Team Name
Magnetic Flux

Team Members and Biography

Nicholas Bishop

My name is Nicholas Bishop and I am currently pursuing a bachelor's degree in Computer Engineering. I consider myself a jack of all trades when it comes to anything mechanical and engineering wise. Some of my hobbies include rebuilding classic cars, programming, and learning/experimenting with new ideas and concepts to further my knowledge of engineering. My goal is to become an embedded system engineer and I have high hopes that this project will help me get closer to that goal.

Carlos Verduzco

My name is Carlos Verduzco and I am currently pursuing a bachelor's degree in Computer Engineering. Some of my hobbies include playing guitar/piano, weightlifting, and programming. I am interested in anything hardware/software related when it comes to being able to interface with musical instruments such as an electric guitar or digital piano. A personal goal of mine is to build a custom made electric guitar with built in effects and anything else I might find interesting to add to it.

Joshua Rodriguez

My name is Joshua Rodriguez and I am currently pursuing a bachelor's degree in Computer Engineering. Some of my hobbies include jogging, reading, playing trading card games, and programming.

Project Overview

This project is about making a three dimensional game table by using wooden blocks to draw images on a tabletop. How this will be accomplished is by using square blocks that are recessed and flush with a tabletop. The blocks can then be raised at any given moment by the use of solenoids that are controlled by the microcontroller. The user can then play a game of pong vs another person or vs an AI using these features described. The amount of blocks that will be moveable to make a picture are a 8x8 grid of blocks, totaling 64 blocks that can be individually moved by a microcontroller. A seven segment display will be used to display the player's current score in the game of pong. The players will use a custom made controller each with 8 buttons on them. Buttons will be used for different functionalities such as moving the paddles, starting the game, launching the ball, and more. Below is an image of how the product was intended to look like at the start of this project and the final design.

The blocks closest to the scoreboard in this picture will be the area of blocks player one is controlling. Only three of these blocks for that row will be up at any given time. These three blocks represent the current position of player 1's paddle. When the player pushes the button on their controller to move the paddle for example right, the leftmost block that is up will go down, and the block next to right most block currently up, will move up. This simulates movement since the position of the paddle has shifted to the right. 00111000 -> 00011100 (0 are blocks that are down, 1s are blocks that are up). Player 2 will be controlling the eight blocks opposite of player 1s side for their paddle the same way. For the ball, since two rows are used for paddles, the remaining six rows will be the area for the ball to be in. The ball will be represented with a single block that is up in the middle of the board. To simulate movement, the block currently up will go down and a new block will raise up, indicating the new position of the ball. The direction the ball moves is determined by how it was hit with the paddle. If the middle block of the paddle hits it, then the block will go straight. If the right or left of the paddle hit the ball, then it will move diagonally across the board.

Project Implementation

Project Demo: https://youtu.be/d7wKTL70yRU

Software

Source Code Link: https://github.com/NickBishop97/Solenoid-Pong-Game

The software portion of the project consists of all the code needed for the pong game to be implemented. This would include several functions such as for updating the paddles, updating the 8x8 grid, receiving data from shift registers, and updating the position of the ball using SysTick. We start off by initializing all the preprocessor directives by defining values which will be used throughout the program. These would include all definitions for all the pins used on the microcontroller, values to set the clock and latch of all shift registers, and seven segment display values to display 0-9 on them.

Each of the .c files and their main functions will have a title and explanation for how it works in order to make this section of the report more readable. To see the code line by line, refer to the section of the report named "Complete Source Code".

Port_Init.c

This file contains the port initializations for those used on the TM4C microcontroller. The ports used and their functionality are listed here

```
PA2 Output - Clock pin shared between all solenoid shift registers
PA3 Output - Clock pin for Player 2 Shift Reg
PA4 Output - Latch pin for Player 2 Shift Reg
PA5 Input  - Data pin for Player 2 Shift Reg
PB0 Output - Latch pin for Shift Reg 1-2 for solenoids
PB1 Output - Latch pin for Shift Reg 3-4 for solenoids
PB2 Output - Latch pin for Shift Reg 5-6 for solenoids
PB3 Output - Latch pin for Shift Reg 7-8 for solenoids
PB4 Output - Data pin for Shift Reg 1-2 for solenoids
PB5 Output - Data pin for Shift Reg 3-4 for solenoids
PB6 Output - Data pin for Shift Reg 5-6 for solenoids
PB7 Output - Data pin for Shift Reg 7-8 for solenoids
PD2 Output - Clock pin for Player 1 Shift Reg
PD3 Output - Latch pin for Player 1 Shift Reg
PD6 Input  - Data pin for Player 1 Shift Reg
PE5 Output - Clock pin shared for all SSDs
PD7 Output - Data pin for Player 1 SSD
PE1 Output - Latch pin for Player 1 SSD
PE3 Output - Data pin for Player 2 SSD
PE4 Output - Latch pin for Player 2 SSD
```

We also make use of the SysTick timer available on our ARM TM4C Microcontroller. We set it to be used as an interrupt with an initial reload value of 5,000,000, corresponding to a 0.313 second delay based on the MCU running on a 16 Mhz clock. This means that the code inside our SysTick_Handler will execute at this rate when the reload value counts down to 0. This speed will be changed as the game continues, with more details explained below.

ShiftRegsInit.c

This file contains the code for receiving and sending data to the shift registers used for the solenoids, seven segment displays, and controller buttons.

We have **write_reg** which is a function for sending data to shift registers and in turn powering solenoids attached to them. This function is used for the 74HC595 Serial In- Parallel Out Shift Registers controlling the solenoids. How this function works is that the latch for all shift registers are set LOW so that no data is sent into the solenoids yet. With there being 4 sets of two 8 bit shift registers daisy chained together, we need to feed data into each set 16 times, hence the for loop that iterates 16 times. In the for loop, we start with the clock being low, then we feed data into each SER or data pin of the first shift register in the daisy chained sets. CLOCK_HIGH will then shift the current value of Data into Bit 0 of the shift register, as well as shift the previous bits in any of the pins to the left by 1. So if the shift register were currently holding 0000 0000 0011 1101 and we send a 0 to the Data pin, it will now hold 0000 0000 0111 1010. Latch High will then store the values from the shift register into the latch/storage register which then sends values to the solenoids determining which ones will activate or deactivate.

Next we have the function **p1_button_write_reg** which is used for receiving data from the 74HC165 Parallel In - Serial Out shift register corresponding to the player 1 controller and returning it as an 8-bit value which can then be used for determining which buttons were pressed and what action to perform based on this information. When Latch is low, data from the shift register cannot be shifted in, so we set it to low then high with a small delay in between. This allows data to be received from the buttons connected to the shift register. We then enter a for loop where we set the current value from the data pin of the shift register to controller_value, and shift it to the left by the current value of i, the variable iterating the for loop. For example, in the first iteration, controller value will simply equal 0000 0001. However the next iteration we shift the value being received by one. Now controller values equals 0000 0011. This will make it so that eventually the entire 8 bits being received a bit at a time will be an 8 bit value. The data pin we are using is active low, so the controller value will equal 1111 1111 when no button is pressed, with certain bits turning to a 0 when a button is pressed. This value will then be shifted to the right by 6. The reason for this is because port PA6 is used to receive data with it corresponding to bit position 0100 0000. Shifting to the right by 6 makes it equal to a normal 1 bit value Finally, this value is returned and set equal to a variable which will be used in another function for analyzing and determining the next action based on the button press information. The function **p2_button_write_reg** works the exact same way, except it is for receiving data from the shift register connected to the player 2 controller.

The function **ShiftReg_SSD** acts the same as write_reg, with a few values changed and it being used for the seven segment displays instead of solenoids.

Controller.c

This file contains the code for determining which actions to perform based on which button a player pressed on their controller and setting flags used in other related functions.

The function **p1_Button_Data** is used for analyzing the information received from p1_button_write_reg. This information is used to determine which button was pressed by a player, and then setting flags which are used in other functions. We will be going through each statement in order. When no button is pressed for a certain amount of time, the game will enter standby mode. In this mode, both paddles will be controlled by an AI, allowing a game of pong to be played on the screen. Once a button is pressed, standby mode is exited and will revert to waiting for the player to select the game settings. If a game has started between two players or one player against and an AI and SW1 is pressed, a flag will be set which causes the entire game to pause. Pressing SW1 again unpauses the game. A secret mode was added called professor mode, where an AI activates to control the player 1 paddle and track the ball no matter where it is. This mode is activated by pressing SW6 and SW3 simultaneously. Pressing SW2 will deactivate this setting. If at any point the game needs to be restarted for some reason, pressing SW8 will revert everything back to its initial values and allow the player to select the game settings again. Pressing SW7 will set a flag for moving the paddle to the right, while SW6 will set a flag for moving the paddle to the left. Finally, pressing SW3 will set a flag for launching the

ball when the player is currently holding. The function **p2_Button_Data** acts similarly, but only allows player 2 to control their own paddle or launch the ball.

The function **pauseMode** simply disables the SysTick Timer which will stop the ball, and in turn stop the players from being able to move paddles or perform other functionalities until unpaused. Finally, the function **Restart** will set all values to their initial states, causing the game to restart as if the reset button on the microcontroller were pressed.

Paddles.c

This file contains the code for setting values of where the paddle will be located as well as tracking the current position of the player paddles.

With the flags set from the functions in **Controller.c**, we can move onto the function **updatePaddle**. Statements in this function are repeated several times with the only difference being which button was pressed by the player for moving left or right, so the following explanation will be about the button for moving the paddle left on the player 1 controller. Starting with the first if statement, we have flags for player1.leftpress which should be one when the left button is pressed for player 1, while pl.leftone should be 0 since this will be the first occurrence of the button being pressed since the previous read. ((value1)&P1_PADDLE_LEFTBOUND)==0 means that the paddle has not reached the left wall boundary which prevents the paddle from going off the 8x8 grid. If the requirements are met, we enter the if statement and shift the position of the paddle to left by 1, as well as set p1.leftone to 1. The next if statement is entered if the button is still being pressed and hasn't been released yet. This increments a counter which will be used for a delay in the next if statement. The final if statement is similar to the first one, but the difference is that our counter needs to equal a predefined value we set earlier in order for the paddle to move left. This essentially creates delay between when the paddle can move while the paddle is still being pressed. The reason for this design is because we want the paddle to move instantly when it is first pressed, but we do not want the button to continue moving instantly otherwise it'll be on the other side of the paddle row in an instant.

The function **PaddleCoordinates** is used to set a value corresponding to the current position of the middle of a paddle. This is used in order for the AI to be able to go towards the ball based on where the ball and paddle are currently located.

AI.c

This file contains the code for how the AI behaves for controlling the player 1 or player 2 paddle.

The function **AI_Mode** contains flags for determining when to allow the AI to control the player 2 paddle. When activated, we increment a counter that when a certain value is reached, allows a function for the behavior of the AI to be called. This delay is used so that the AI is unable to move the paddle too quickly, this is similar to the delay used in the function updatePaddle which creates delay so the paddle will move at a certain rate and not instantly across.

AI_Behavior contains the logic for how the AI behaves based on multiple variables. Depending on which difficulty setting was set for the AI before the game started, the AI will start tracking the ball when it is at a certain point on the 8x8 grid, and the ball is moving towards the player 2 paddle. When these requirements are met, the ball position and current position of the paddle will be compared with each other and cause the paddle to move accordingly in order to be able to hit the ball. Preventing the AI from tracking the ball until it is at a certain point on the 8x8 grid allows for scenarios where the AI hits the ball less often in easy mode compared to hard mode where it tracks it accurately. If the condition to track the ball is not met, then the AI will move the paddle randomly left or right.

The function **professor_mode** acts similarly to the functions AI_Mode and AI_Behavior, with a few changes. For one, professor mode is used to have an AI start controlling the player 1 paddle, with it activating whenever SW6 and SW3 are pressed simultaneously during a game. However, it tracks the ball much more accurately than the AI for player 2. If standby mode is activated, professor_mode will cause the AI controlling the player 1 paddle to act the same way as a hard mode AI. This allows for both AIs playing against each other to have an equal chance of scoring points.

SSD.c

This file contains the code for displaying appropriate values on the seven segment displays for each player's current score.

The function **decoder** takes in a decimal value and returns a hexadecimal value. This is used to activate the appropriate segments on a seven segment display in order to show a certain number/letter. The seven segment display will show either the player score, or letters and numbers representing the current game setting to be selected.

updateScore takes in two decimal values and calls a decoder() in order to receive hexadecimal values. These hexadecimal values are set in a variable called SSD_Value to create a 16 bit value which is then sent to ShiftReg_SSD one bit at a time in order to have the shift registers activate the seven segment displays.

Ball.c

This file contains the code for the ball moves around the 8x8 grid and tracking its current position.

The **SysTick_Handler** is where most of the logic for moving the ball is located. This function uses case statements to act similarly to a state machine, so each state will be explained individually. It always starts off in STARTING where we set a value for which paddle was hit, which just determines which direction the ball will move in to start, as well as the next state. The MOVEUP state will have the ball traveling straight up to player 2 unless it is going to hit the player 2 paddle, where it will bounce off and go in the other direction randomly away from player 2. Otherwise, if the ball reaches the row where player 2's paddle is, we go to the POINTMADE state. MOVEDOWN works nearly the same way, except the ball will be traveling down. MOVEDIAGLEFT allows the ball to bounce off paddles, go to MOVEDIAGRIGHT if it hits the left wall, move to POINTMADE if a player scores, or simply move diagonally left.

MOVEDIAGRIGHT acts the same way but for moving to diagonally right instead with it moving to MOVEDIAGLEFT if it bounces off the right wall. In order to make the game more interesting, everytime the ball hits the paddle, the ball will increase in speed slightly to a certain point in order to make it challenging for players. POINTMADE is entered whenever a player scores a point. If a player has reached the maximum points, the state ENDGAME is entered, otherwise flags are set to determine which player's score will increment as well as where the ball will be located next. The speed of the ball is also brought back to its starting speed. In ENDGAME, if a normal game has ended, a flag will be set for activating the win screen on whichever player won. If this state is reached during standby mode, then the entire game will simply reset, allowing the two AIs to play against each other indefinitely.

FSMSTATE ballDirection sets the direction the ball will begin traveling based on which player last hit it, and a random number generator which chooses a random value. Finally, **BallCoordinates** acts the same as **PaddleCoordinates** in which it is used to set a value based on the ball's current position which will be used for the AI.

ADCSWTrigger.c

This file contains the code for enabling ADC on a pin in order to receive an analog value which will be converted into a 12 bit number to be used as the seed for srand().

ADC_InSeq3 initiates the ADC conversion and sets the result into a variable as a 12 bit value. Usually this value would be received by a sensor connected to the pin, however, leaving the pin unconnected still allows a random number to be converted each time which is exactly what we want.

srandSeed called ADC_InSeq3 10 times and added the sum of the result each time, with the result then being averaged out afterwards. This process allows a random value to be received nearly every time.

Solenoid.c

This file contains the logic for sending data and activating the solenoids on the 8x8 grid. With all the information received from the shift registers and using that information to set flags and where the paddles will be located, we can call the function **updateGame**, which essentially sets the values that will be sent to the shift registers controlling the solenoids to display the paddles and ball. The variable value is a 64 bit integer that will hold the positions of the paddles and ball. Its least significant 8 bits will be where the player 2 paddle is located, while its most significant bits will be where the player 1 paddle. This is why we shift playerOne_CurrentPosition to the left by 48. This moves it into its most significant 8 bits since playerOne_CurrentPosition has an initial value of 0x3800. playerTwo_CurrentPosition has an initial value of 0x38, so if we just OR, it will already be in the least significant 8 bits. Ball_position is also a 64 bit value, so we can simply OR it with value. Value1 to value4 each receive a different 16 bit portion of the 64 bits in value. While in this for loop, we shift each of the values by i, AND the result by 1, and set it equal to regs[i] which is an array holding the values which will be sent into the shift registers when calling the function write_reg. We also

added code for having the ball immediately appear in front of the player that was scored on, as well as a win screen that displays whenever a player reaches the maximum score.

GameSettings.c

This file contains the code for inputting the settings before the start of a game of pong. The function **GameSettings** contains the logic for choosing the game mode, AI difficulty, and score limit which is all displayed on the seven segment display as each setting is selected. It will start with displaying "2P" which means 2 player mode is being chosen. Pressing SW3 will confirm this setting, while pressing SW7 will cycle to the next option which is "1P", player against AI. With "1P" selected, "d1" will be shown which means the AI is on the easiest setting. "d2" is intermediate while "d3" is the hardest difficulty. Once selected, the final setting is choosing if the score will go up to 3, 5, 7, or 9 points. Once all settings have been chosen, the game can commence. If at any point during this selection of setting the player stops pressing buttons for more than 5 seconds, standby mode will commence.

ScoreSetting is the function called by GameSettings which contains the logic for the max score.

UART_Debug.c

This file contains the code for displaying the 8x8 grid on a serial terminal. This allows for debugging of the game with being able to see current positions of the ball and paddle without having all the hardware setup

HextoBin2 converts the value of one of the rows of the 8x8 grid represented as a string into an 8 bit binary number, and displays it on a serial terminal. This process is repeated 8 times until an entire 8x8 grid is shown in real time, matching what should be displayed on the actual solenoids.

showPosition converts hexadecimal values corresponding to the rows into a string, organizes how the rows will be shown on the terminal, and then calls HextoBin2.

Common.c

This file contains all the variables shared between the different .c files, allowing for easy access to them.

PongGameFinalVer1.c

This file contains all the function calls in order for the game of pong to function on the 8x8 grid of solenoids.

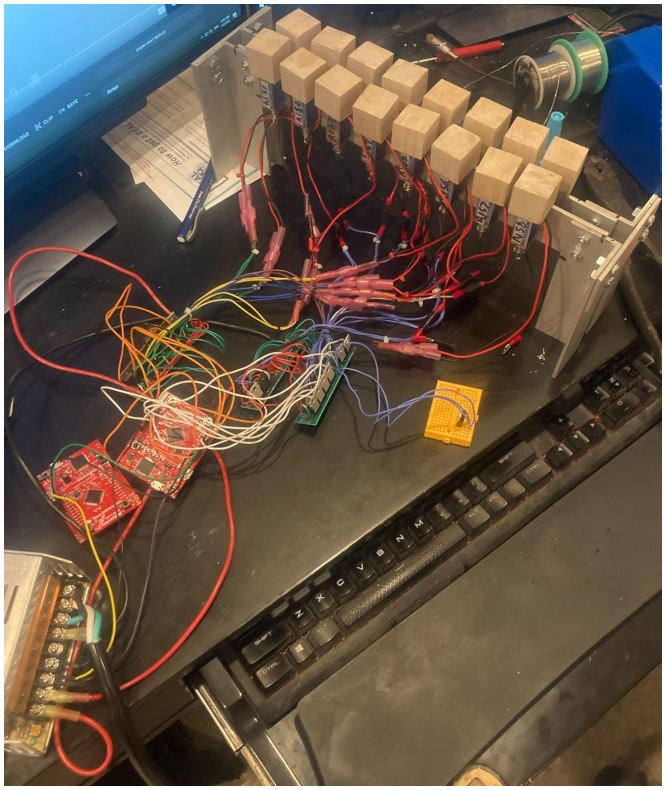
In **main,** everything is initialized such as the GPIO ports, starting positions of the ball and paddles, and scores. During each iteration of the while loop, the data from the controllers will be received in order to determine the next course of action. The game then commences with appropriate functions being based on whether players are going against each other or if standby mode has activated.

Hardware

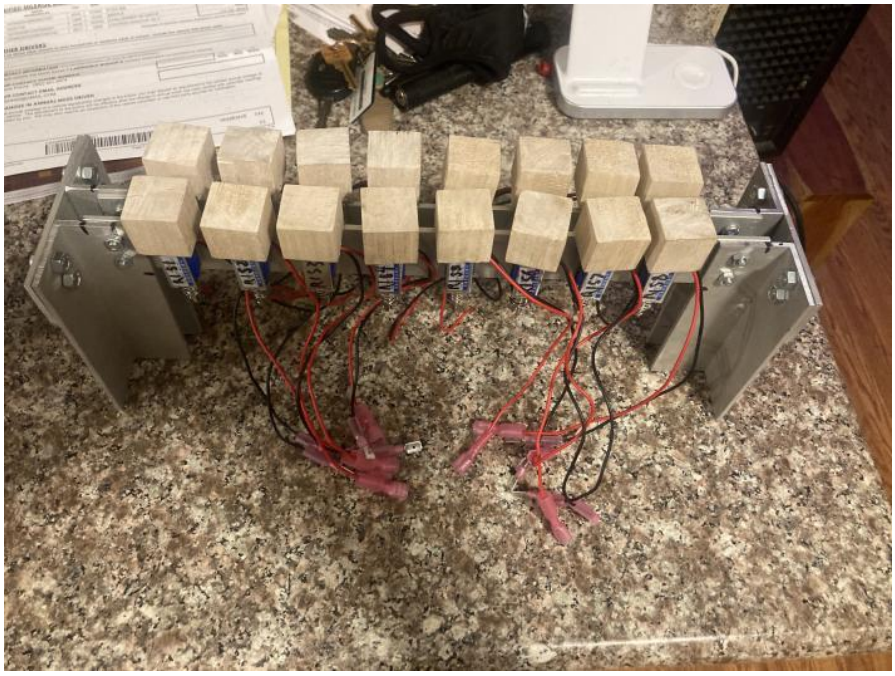
Solenoid Circuitry

The goal of this project is to have an 8x8 grid of solenoids that push a block up to represent a game of pong. The solenoid is an inductor that, when a certain amount of voltage is applied and current flows, creates a magnetic field which will draw a movable core into the coil. When this occurs, the solenoid will execute a push and pull sequence with the coil. Each solenoid will be controlled by a pin from a SIPO shift register. Unlike other electrical components that we can simply connect directly to pins on an MCU using wires such as an LED, a solenoid requires more complexity to work correctly. For one, the solenoid that we chose requires 5V and about 1A, both of which are more than what the MCU pins can supply. Therefore we require a power supply that can supply at least 5v to power the solenoid and 1A for each solenoid. An N-channel MOSFET is also required in order to be able to drive the solenoid with the pin from the MCU. The output pin will be connected to the Gate pin of the MOSFET. When 3.3v from the pin is supplied to the Gate, it will allow current from the 5v battery to flow between the drain and source pins of the MOSFET. This entire configuration will allow us to drive a solenoid. A 10k resistor is placed between Gate and GND in order to ensure current will not flow directly to GND without reaching the Gate. The final component needed in this circuit is a flyback diode. The reason for this is because when power is cut from the solenoid, the electromagnetic field within collapses which creates a large flyback voltage (voltage spike) that can damage the MOSFET or other electrical components. The flyback diode is connected in reverse bias when power is being supplied, which causes it to not impede the operation of the circuit. When power is cut off however, the change of polarity of the inductor causes the diode to be forward biased, causing the current to flow in a closed circuit between the diode and inductor until all the current dissipates.

Two Rows of Solenoids Connected to Circuitry

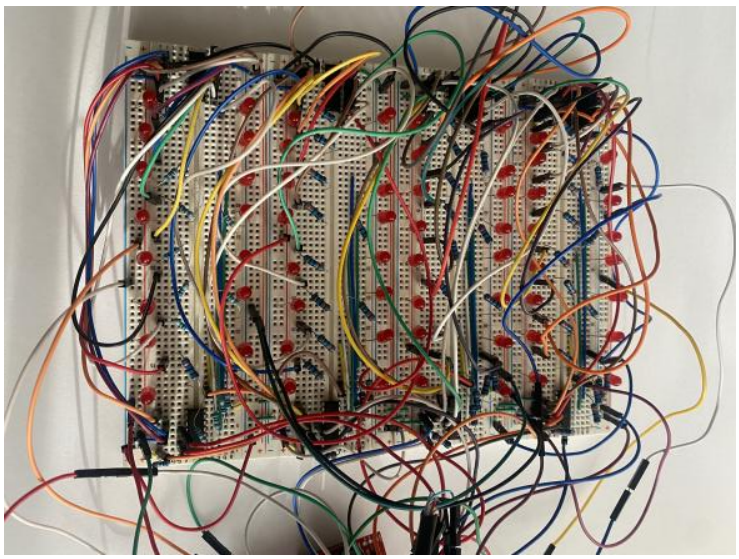


Set Up for 2 Rows of Solenoids



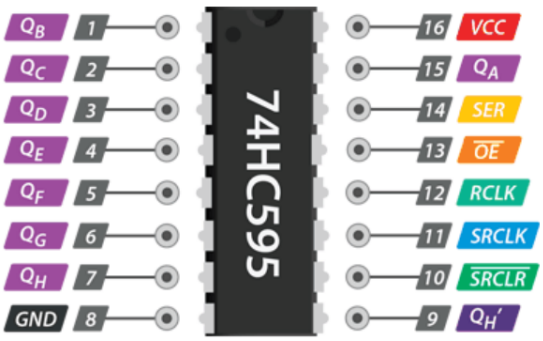
LED Circuit

With the circuitry for all the solenoids not being completed right away, we decided to build an led circuit that would represent the 64 solenoids. This would allow us to test the code on here and have it working so that it can be ready when using it to control the solenoids.



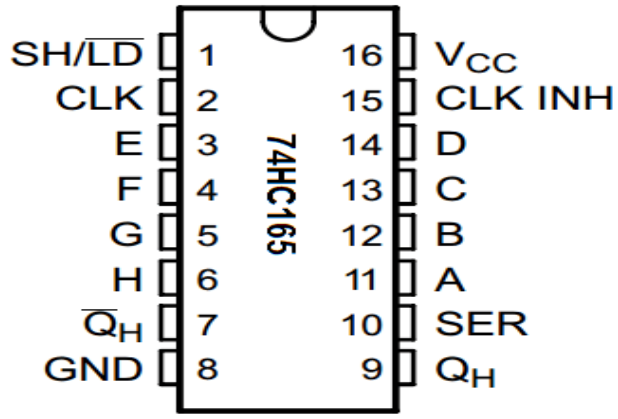
74HC595 Shift Register

With having to control so many solenoids individually, as well as 4 seven segment displays, this would require an extensive amount of I/O pins from our MCU which does not have enough available. In order to solve this, we decided to use several SIPO shift registers.
These shift registers have 16 different pins. QA to QH are output pins that will connect to components we want to activate such as leds or solenoids. SER is the data pin which receives values from the MCU that will be sent out as outputs. OE is an output enable that allows current to flow to outputs when set low. SRCLR clears the data inside the register when set high. RCLK is the latch pin which stores the data inside the storage register which will then show up in the outputs. SRCLK is the clock pin which will shift bits into the register. QH' outputs bit 7 of the shift register, which allows daisy chaining with multiple shift registers. Based on the code written, data is shifted into the register at every rising edge of the clock cycle and SER receives a new bit. After 8 clock cycles, there will be 8 bits stored in the register. Once the latch is set high, the high bits will cause connected outputs to activate while low bits will cause outputs to turn off.



74HC165 Shift Register

In order to create custom controllers for each player, the controllers would need 8 buttons each. This would use 16 pins from the MCU which similarly to the problem we had with the solenoids, is too many. To solve this, we used a PISO shift register which would allow it to receive data from 8 buttons connected to it. Pins A-H would be where data is received from the buttons connected to them. CLK is simply the clock which is manually set in the code. QH is where the current value in bit H is stored. SH/LD is a latch signal, and CLK INH prevents the clock from working when high active. This shift register works by first setting the latch to low then high to immediately read in all the values from the buttons. Then, at every rising edge of the clock, the bit in pin H is stored in SER as an input which is then stored in a variable in the code. Then, H receives the value stored in G since all bits shift to the left by 1. After 8 clock cycles, 8 different bits have been sent by SER which are stored in a variable for future usage.

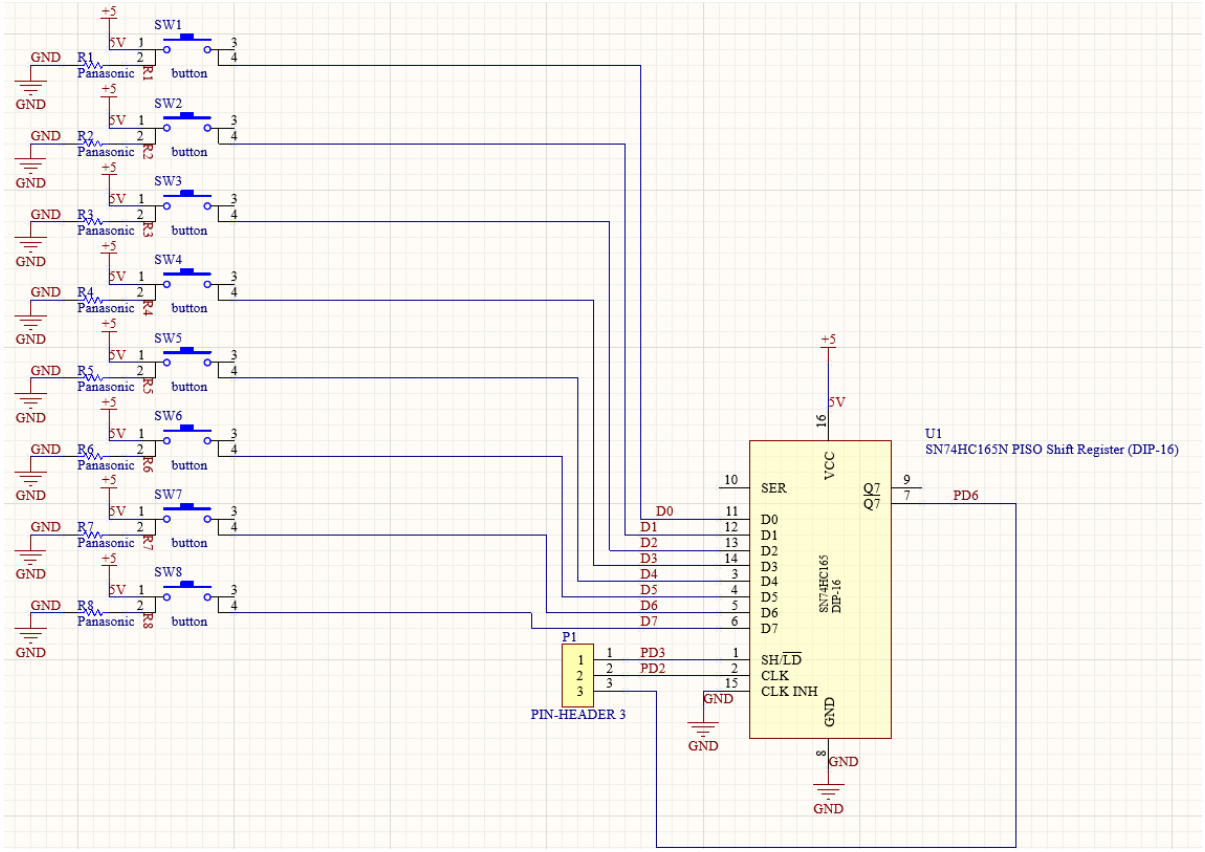


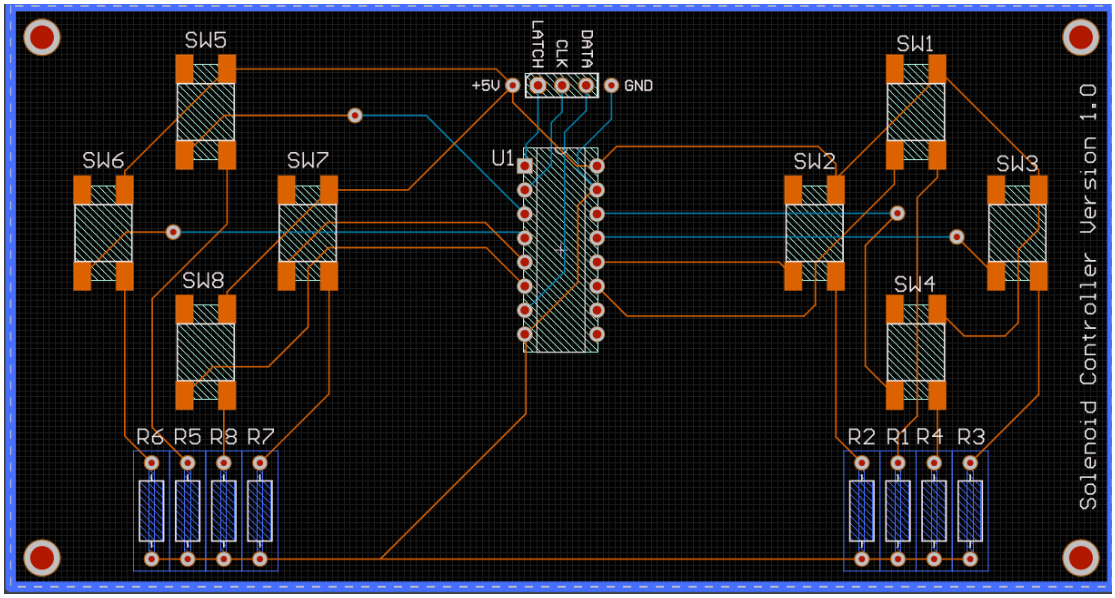
PCBs

Since this was going to be a large project with many different connections and parts, we decided to design several PCBs to make this process easier, and have stable connections. All PCBs were made in Altium Circuitmaker.

Controller PCB Ver 1

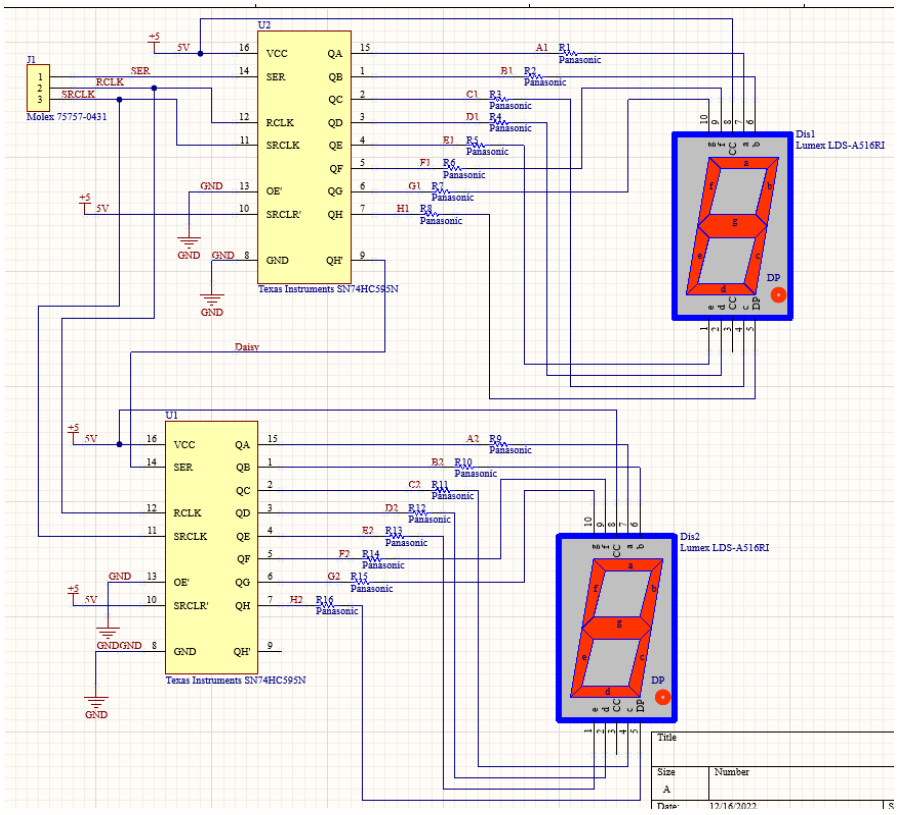
After making sure the 74HC165 shift registers worked properly with the buttons, we designed a controller PCB replicating it. The buttons were organized to be as symmetrical as possible with one another. An ethernet cable was used as the wire to connect from the controller to the microcontroller.

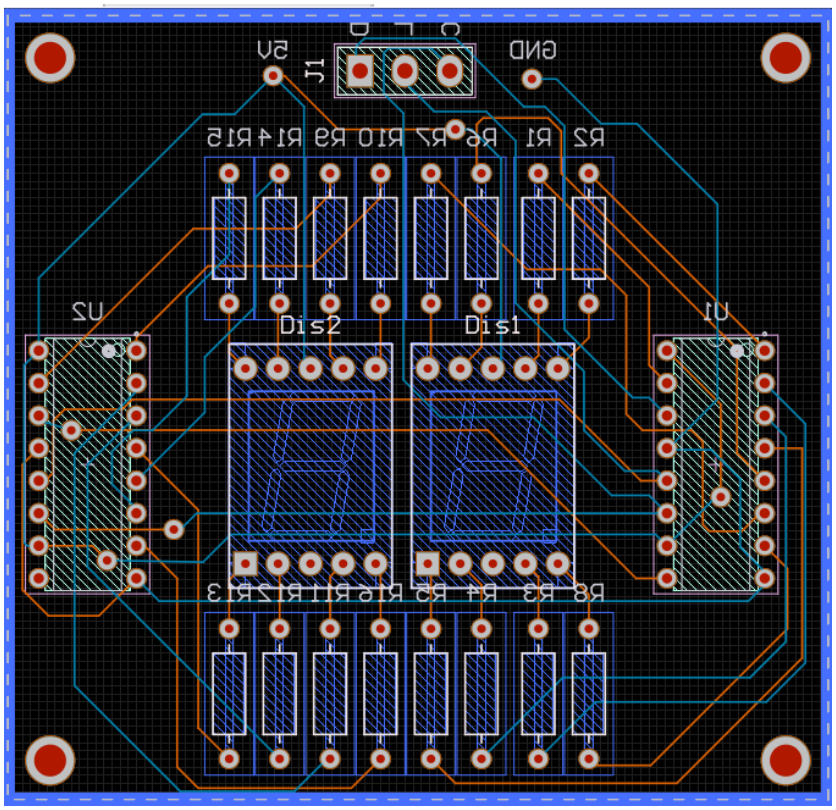




Scoreboard PCB

This PCB for the scoreboard uses two 74HC595 shift registers to control two seven segment displays. Most of the components besides the displays, were placed on the bottom layer to keep the top layer as clear as possible. This was done because originally, we planned to mount the PCBs on the wooden table.

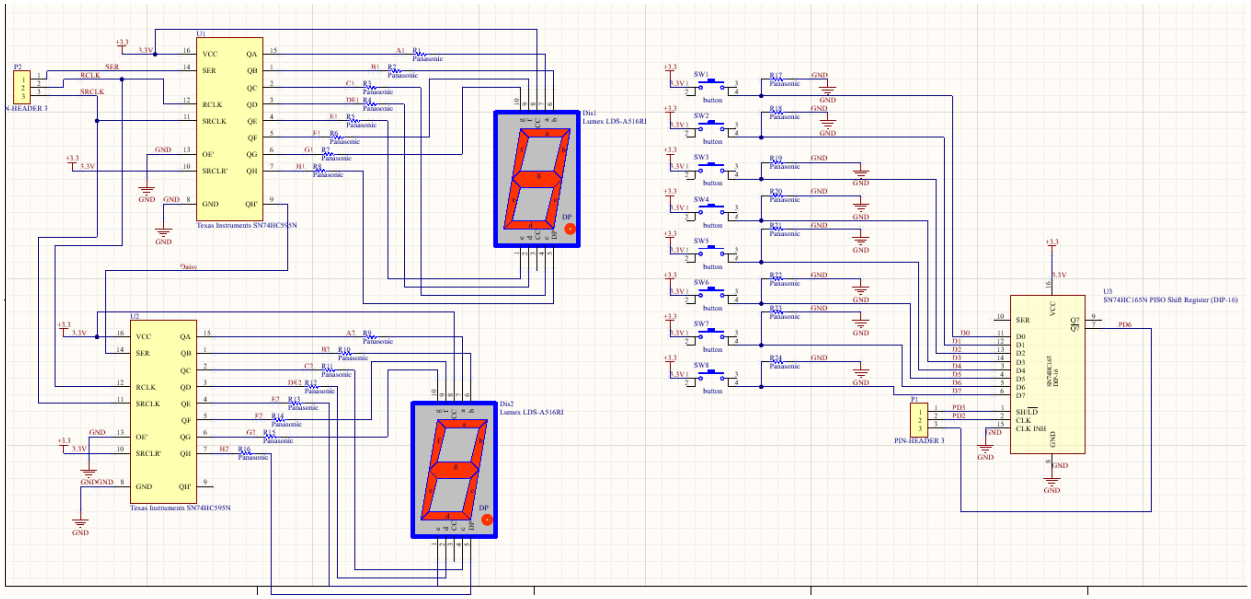


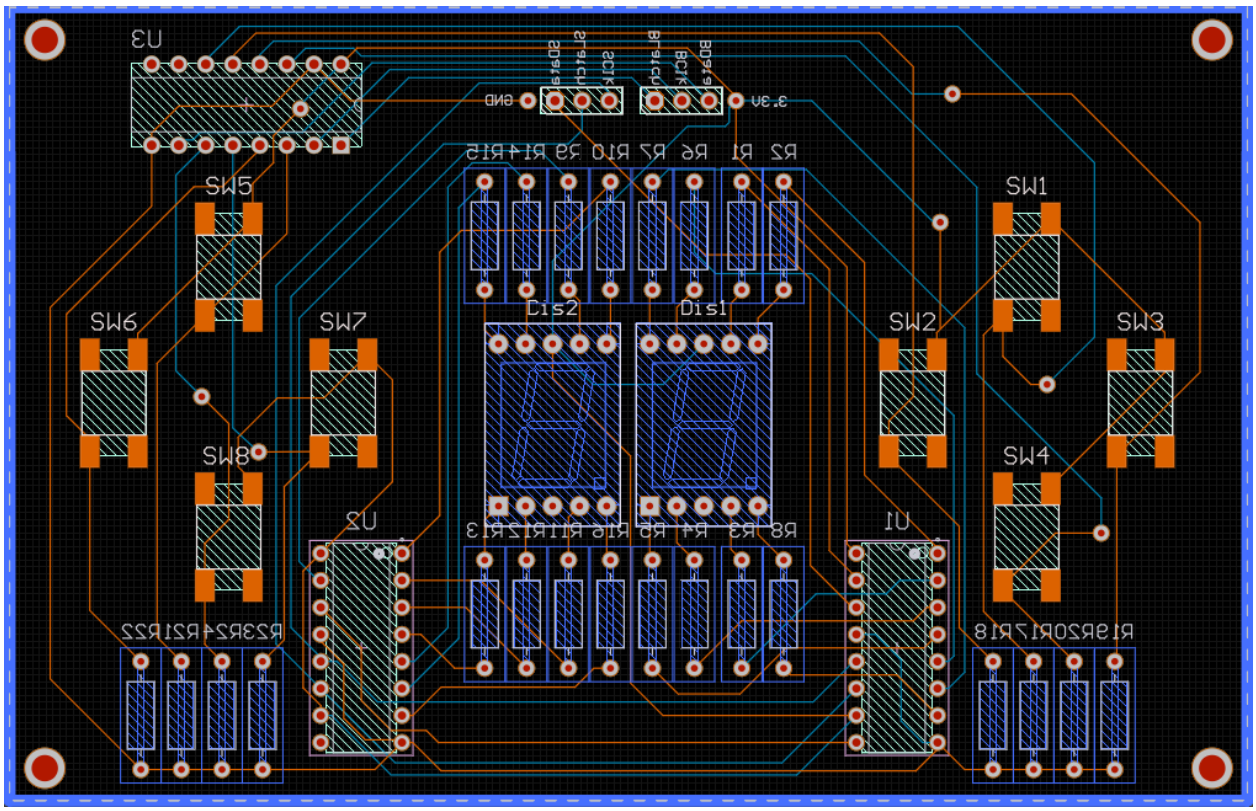


Controller PCB Final Version

With understanding how to use the shift registers and having them work with the buttons and seven segment display, we could now design the PCB for the controller. We decided to include the seven segment display within the middle of the PCB so that players could easily see the score there at all times.

First, we created a schematic with all the connections properly labeled so that when continued to the PCB layout, all the required connections could be easily seen and done. All that was left was organizing everything and routing all the connections properly in the confined space.

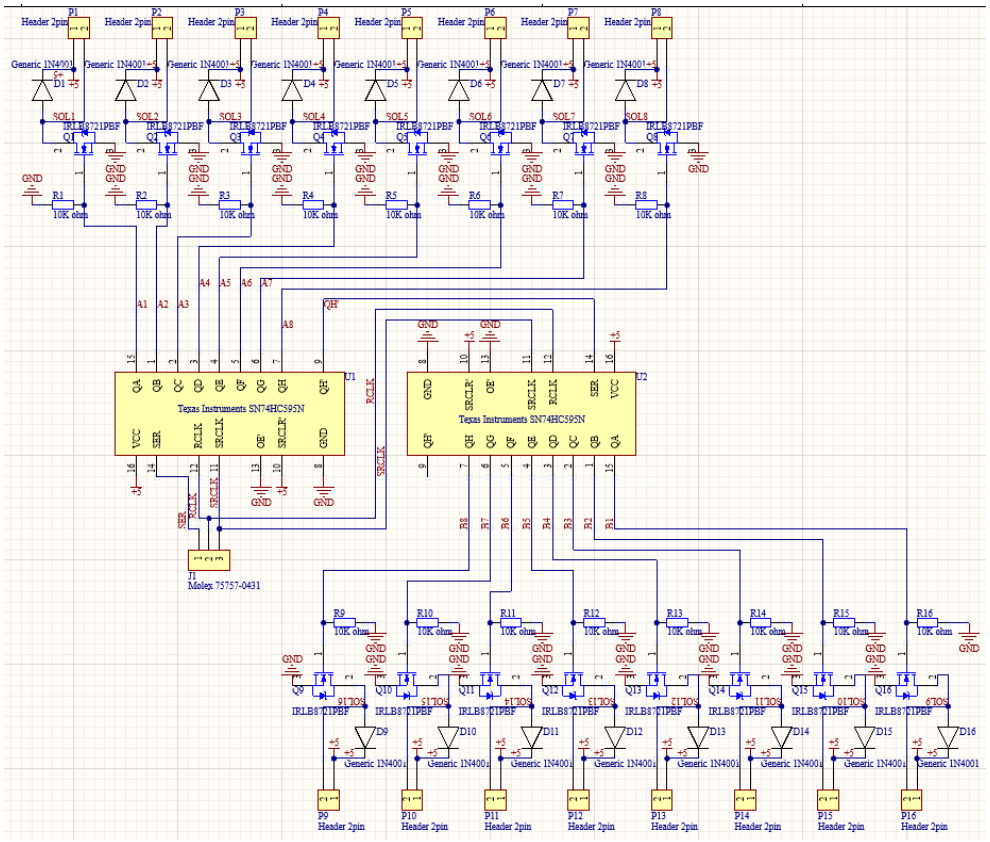


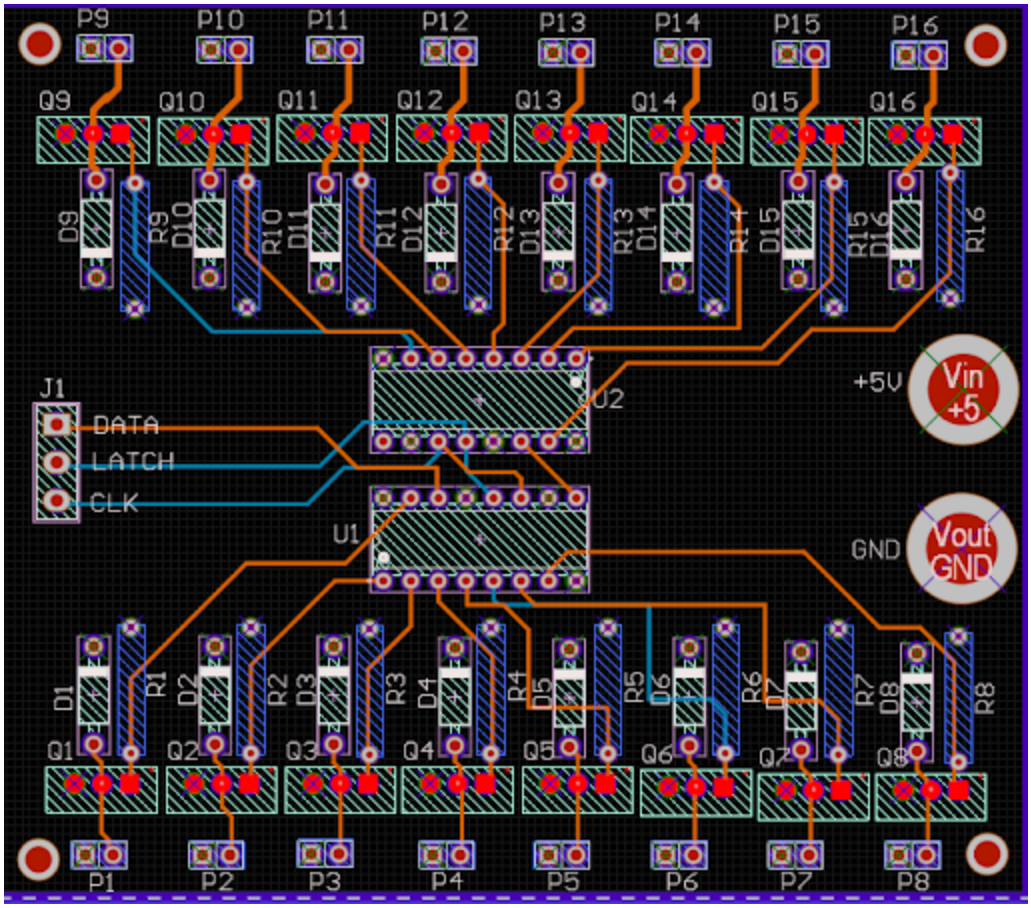

U3
SW5
SW1
SW6
SW7
SW2
SW3
SW8
SW4
U2
U1
Dis2
Dis1

Solenoid Driver PCB

After understanding how the solenoids work and what they require to keep the components safe under operation we designed a solenoid driver board. This board would power up to sixteen solenoids at once using two 595 shift registers.

We first made a schematic that allowed us to connect everything and give them proper names and group names. We then started the PCB design and opted for a four layer board instead of a two layer board, so we could house the PWR and GND planes in the center two layers and keep the data lines on the top and bottom layers.

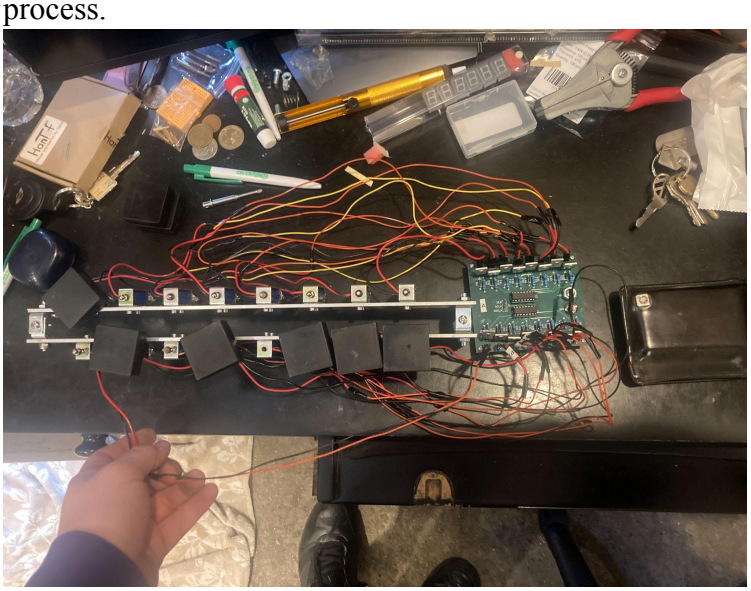


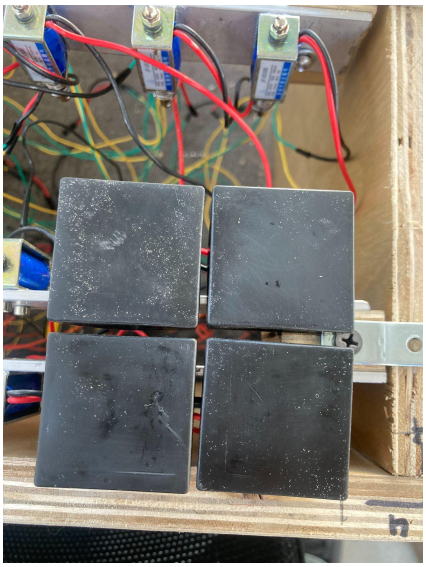


Build Process

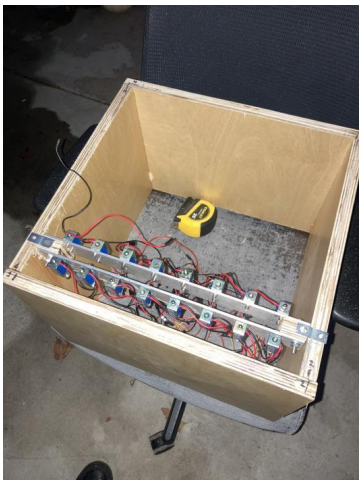
The first stage of the build process was to create a block that would sit on top of the solenoid and house and LED for illuminating the plastic block. This block had to be light enough so the solenoid could lift it and as big as possible but for people to see it move. This is where 3D printing and designing came into play and after eight block versions we came up with this design. Below shows a version history of the blocks from earliest(left) to latest(right).

The second build stage was to create the solenoid rack that would hold sixteen solenoids each so it could be paired with a solenoid driver board. This took some time and planning to properly create, because the space between the solenoids had to be accurate enough that when the blocks are installed on top they wouldn't hit one another and would still have a decent enough gap between them so a spacer could be added to straighten the blocks if they tried to go sideways when they were actuated. The two pictures below accurately show our result from the process.





The third build stage was to build a box to house everything we have built up to this point. The box needed to be able to house everything we have built and used to power and run the game so far, so this called for a box that would be 1.3 feet wide, 1.3 feet long, and 1 foot high. Creating the box was quite simple, but installing everything into the box was another challenge we had to face in this task. The first image below shows the box walls completed with us test fitting the first solenoid row. The second image is all eight rows tested and put equal distance from each other to see if the block distance is adequate. The last image shows the bottom of the box has been installed and a fan hole has been cut to cool the whole project.



The final stage of this project was assembling the whole project together and testing for any bugs or issues with anything when fully assembled.

How to Play A Game

When starting the game, it starts off in the main menu where player 1 will select the settings for the game. These settings will be displayed on the seven segment displays. These settings are listed here,

2P - 2 Player Mode, 1P - 1 Player against AI

d1, d2, d3, - Easy, medium, and hard difficulty

P3, P5, P7, P9 - Max score limit

If no button is pressed for a certain amount of time while selecting settings, standby mode will start

Standby mode has two AIs control the paddles and play a game of pong against each other indefinitely until player 1 presses a button again

Button Functionalities

SW1 - pauses/unpauses game

- SW2 - exits professor mode
- SW3 - selects game setting/launches ball
- SW6 - moves paddle to the left
- SW7 - cycles through game settings/moves paddle to the right
- SW8 - restarts entire game
- SW3 and SW6 pressed together - enter professor mode

Customer Needs/Specification

- 1. **NO INPUT DELAY**
    - Seemingly immediate feedback
    - Response time of 20 ms when button is pressed
- 2. **EASY SETUP AND ACCESSIBILITY**
    - Once fully assembled, simply plug into wall outlet
    - Anybody can play a game and have fun
- 3. **ABLE TO PLAY TWO PLAYERS OR AGAINST AI**

    - Game settings included to select the mode
    - AI comes with multiple difficulty settings
- 4. **PLAYER ONBOARD POSITION EASILY VISIBLE**
    - Player’s controllers will be on opposing sides
    - Blocks move up indicating paddle location
    - LEDs light up indicating paddle location
- 5. **SCOREBOARD**
    - Seven Segment Displays located on both player controllers to easily see current score
    - Also displays game setting information
- 6. **PORTABILITY**
    - Table should be able to be moved from room to room
    - Two people should only be necessary to move the device
- 7. **SAFETY**
    - All electronic components will be secured down safely and away from users' reach.
    - Users should not be able to see any of the electronic components except for the power cable and controller cable
- 8. **REAL WORLD SIMULATION**
    - Blocks will move and interact like a ball and paddle in a game of pong.
    - Ball increases in speed over time to make the game challenging
- 9. **GAME SETTINGS**
    - Players can select different settings to change how the game behaves
    - Players can change the game mode, AI difficulty, and score limit
- 10. **SMOOTH PADDLE MOVEMENT**
    - Button can be held to move paddle at a constant rate, or pressed repeatedly to move faster
- 11. **STANDBY MODE**
    - Players will be able to witness a game of pong without actually playing themselves

    - After 5 seconds, two AIs begin playing a game of pong indefinitely until a player presses any button to exit this mode
- 12. **CHEAT CODES**
    - Game acts as a real game where there are hidden button combinations that can give a player an advantage
    - Pressing SW3 and SW6 on player 1 controller will enable professor mode, where an AI takes over the paddle and can never miss the ball

Similar Products

Air Hockey Table

This product is similar in the sense that the goal is to hit a projectile and get past the opponent's paddle to score a point. This is more of a manual game that doesn't include any code or hardware however. It's also similar because it's a mechanical pong game project just like ours.

Mechanical Pong

This project follows the same goal as ours, with a few differences. For one, the ball is represented by a block that can move around the screen using magnets within the tables. The paddles are attached to a rail with a belt for moving left and right. They also use Arduino boards which are different from our TM4C microcontroller.
https://hackaday.com/2016/05/28/pong-in-real-life-mechanical-pong/

Societal and Environmental Impact or Importance

This project was meant to be more of an interest project instead of benefitting society. However, we were able to make optimizations in order to have lower costs and less space used than previously intended. This is due to the shift registers we decided to use that helped us in limiting the amount of pins needed on our microcontroller. Without them, we were going to have 5 microcontrollers in this project which would have been more costly.

Subcomponent Descriptions

TM4C - The microcontroller used in this project is the Tiva TM4C123GH6PM Microcontroller. The microcontroller contains a 32-bit ARM Cortex-M4F Processor Core that runs at 16MHZ.

32-bit ARM Cortex-M4F Processor Core
256 KB single-cycle Flash Memory
32 KB single-cycle SRAM
6 General-Purpose I/O (GPIO) ports: 4 8-bit ports (labeled A,B,C,D), 1 6-bit port (E), 1 5-bit port(F)
8 UART ports
Max GPIO high-level input voltage: 5.5V
Min GPIO high-level output voltage: 2.4V
V_{DD} supply voltage: 3.15V to 3.63V
1 internal and 1 external oscillator
Execution speed/ Clock Speed with Crystal: 16MHZ
Execution speed/ Clock Speed with PLL: 80 MHz

Mosfet (IRLB8721PBF) - The IRLB8721PBF is an N-channel Enhancement type mosfet that will act as a switch component for the solenoid. Whenever the mosfet receives a certain amount of voltage, it will transfer into the ON state and open the gate of the mosfet. Through action, a constant voltage and current flow will be produced between the source and drain terminals. As such, the solenoid will be able to receive enough to turn on and push the blocks for the 3d game table. If not enough voltage is applied to the mosfet, then the mosfet will stay in the OFF state and not allow voltage or current to flow through. The specs for the MOSFET are as follows:

Max Drain Source Voltage: 30v
Max Gate Source Voltage: +-20v
Drain Source Breakdown Voltage: 30v
Gate-Source Threshold Voltage: 1.35v to 2.35v
Maximum Drain Current: 62A to 44A (Decreases when temperature increases)
Drain Source On-state Resistance: 6.5 to 8.7 ohms

Seven Segment Display(5161BS) - The seven segment display is an electronic display that contains 8 LEDs. These 8 LEDs are separated into their own specific segment and receive their own input from the program. With specific inputs, numbers between 0-9 can be displayed to the user to signify the score of the 3-D game table.

Forward Voltage: 1.8v
Reverse Current: 50μA
Operating Temperature: -20 C to 75 C

Solenoid (JF-0530B) - The solenoid is an inductor that, when a certain amount of voltage is applied and current flows, creates a magnetic field which will draw a movable core into the coil. When this occurs, the solenoid will execute a push and pull sequence with the coil. This push and pull sequence will be used to move the blocks of the 3-D game table to move up and down.

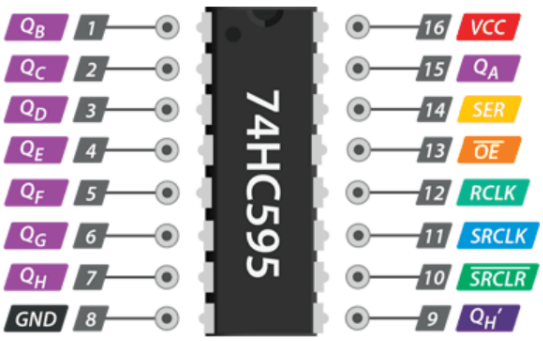
Internal Resistance: 4 ohm
Voltage draw: DC 6v
Amperage Draw: 1A-1.25A (depending on temp)

Buttons (JF-0020) - Whenever an input is signaled, the input is unstable and will oscillate between ‘0’ and ‘1’ because of the mechanical nature of the switch. In order to resolve this issue, we can create a debounce delay through software, so that the input can become stable and the microcontroller will be able to receive the correct data. These buttons will be used in both player controllers in order to allow for multiple functionalities such as moving paddles, pausing the game, restarting the game, and selecting game settings.

Max Operating Voltage: DC 12v 50mA
Operating temperature range: -25C-85C
Contact resistance: 50m ohm
Insulation resistance: 100m ohm

Diode (1N4001) - A diode is a device that acts as a one way switch for current. It allows current to flow easily from one direction, but severely restricts current flowing from the opposite direction. In our case we have a flyback diode. The reason for this is because when power is cut to the solenoid, the electromagnetic field collapses which creates a large flyback voltage (voltage spike) that can damage the MOSFET or other electrical components. The flyback diode is connected in reverse bias when power is being supplied, which causes it to not impede the operation of the circuit. When power is cut off however, the change of polarity of the inductor causes the diode to be forward biased, causing the current to flow in a closed circuit between the diode and inductor until all the current dissipates.
Forward Voltage: 1.1V
Average Forward Current: 1A
Max Operating Temperature: 175 C
Peak Reverse voltage: 50V

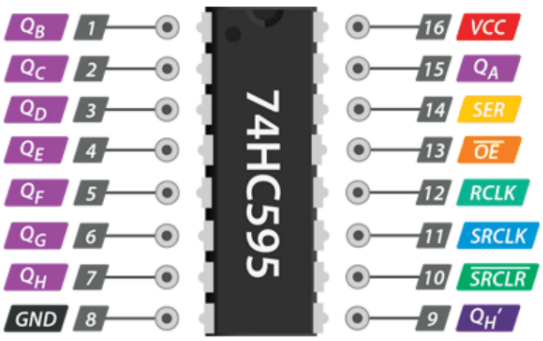
8 bit Serial-In-Parallel-Out Shift Register (74HC595) - With having to control so many solenoids individually, as well as 4 seven segment displays, this would require an extensive amount of I/O pins from our MCU which does not have enough available. In order to solve this, we decided to use several SIPO shift registers.
These shift registers have 16 different pins. QA to QH are output pins that will connect to components we want to activate such as leds or solenoids. SER is the data pin which receives values from the MCU that will be sent out as outputs. OE is an output enable that allows current to flow to outputs when set low. SRCLR clears the data inside the register when set high. RCLK is the latch pin which stores the data inside the storage register which will then show up in the outputs. SRCLK is the clock pin which will shift bits into the register. QH' outputs bit 7 of the shift register, which allows daisy chaining with multiple shift registers. Based on the code written, data is shifted into the register at every rising edge of the clock cycle and SER receives a new bit. After 8 clock cycles, there will be 8 bits stored in the register. Once the latch is set high, the high bits will cause connected outputs to activate while low bits will cause outputs to turn off.



Min Supply Voltage: 2v
Max Supply Voltage: 6v
DC Input Voltage , Output Voltage: 0 - VCC
DC Input Current, Per Pin: +- 20 mA
DC Output Current, Per Pin: +- 35 mA
Operating Temperature: -55 to 125 C

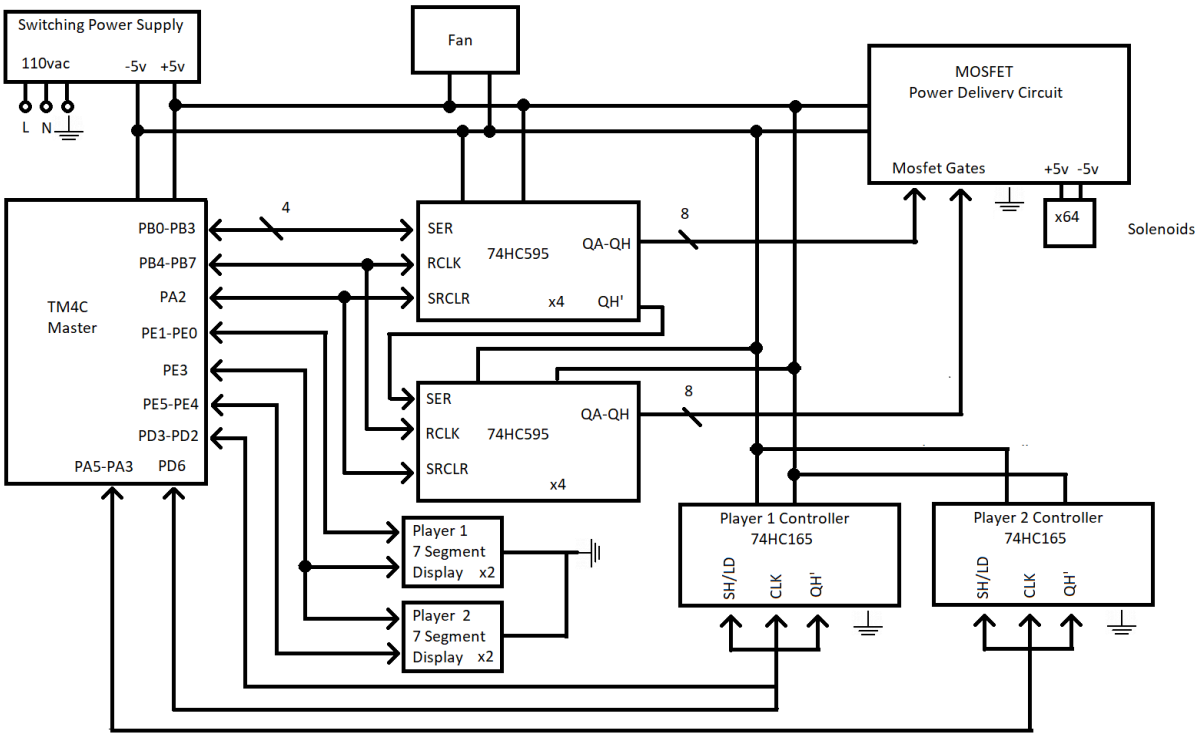
8 bit Parallel-In-Serial-Out Shift Register (74HC165) - With having to control so many solenoids individually, as well as 4 seven segment displays, this would require an extensive amount of I/O pins from our MCU which does not have enough available. In order to solve this, we decided to use several SIPO shift registers.

These shift registers have 16 different pins. QA to QH are output pins that will connect to components we want to activate such as leds or solenoids. SER is the data pin which receives values from the MCU that will be sent out as outputs. OE is an output enable that allows current to flow to outputs when set low. SRCLR clears the data inside the register when set high. RCLK is the latch pin which stores the data inside the storage register which will then show up in the outputs. SRCLK is the clock pin which will shift bits into the register. QH' outputs bit 7 of the shift register, which allows daisy chaining with multiple shift registers. Based on the code written, data is shifted into the register at every rising edge of the clock cycle and SER receives a new bit. After 8 clock cycles, there will be 8 bits stored in the register. Once the latch is set high, the high bits will cause connected outputs to activate while low bits will cause outputs to turn off.



Operating Voltage: 2v - 6v
±4-mA Output Drive at 5 V
Low Input Current of 1 µA Maximum
Parallel-to-Serial Data Conversion

Overall System Functional Block Diagram or Diagrams



Table

Power Delivery Circuit	Signal Name	Input/Output	Description	Signal Type	Voltage	Current	Operration
	Mosfet Gates	Input	Receives signal from slave MCU on when to close MOSFET gates	Digital	3.3v	NA	Receives HIGH signal to close mosfet and LOW signal to open
	Power + Power -	Output	Sends power from the MOSFETs to the solenoids when the Gate	Analog	5v	1A	Will turn on the solenoids when the MOSFET gates close sending 5v 1A to the connected solenoid.

			is closed				

TM4C	Signal Name	Input/Output	Description	Signal Type	Voltage	Current	Operation
	PA2	Output	Clock pin shared between all solenoid register registers	Digital	3.3v	NA	Holds the clock cycle for all solenoid shift registers so that all solenoids can update at the same time
	PA3	Output	Clock pin for Player 2 Shift Reg	Digital	3.3v	NA	Holds the clock pulses for Player 2 controller to receive inputs from Player 2
	PA4	Output	Latch pin for Player 2 Shift Reg	Digital	3.3v	NA	Sends HIGH signal to latch to update the shift register on the Player 2 controller and LOW signal when not updating
	PA5	Input	Data pin for Player 2 Shift Reg	Digital	3.3v	NA	Holds the data bits that will be transmitted to the shift register on Player 2 controller
	PB0-PB3	Output	Latch pin for Shift Reg for solenoids	Digital	3.3v	NA	Sends HIGH signal to latch to update the shift register on the solenoid circuitry and LOW signal when not updating
	PB4-PB7	Output	Data pin for Shift Reg for solenoids	Digital	3.3v	NA	Holds the data bits that will be transmitted to the shift register on the solenoid circuitry
	PD2	Output	Clock pin for Player 1 Shift Reg	Digital	3.3v	NA	Holds the clock pulses for Player 2 controller to receive inputs from Player 2
	PD3	Output	Latch pin for Player 1 Shift Reg	Digital	3.3v	NA	Sends HIGH signal to latch to update the shift register on the Player 2 controller and LOW signal when not updating
	PD6	Input	Data pin for	Digital	3.3v	NA	Holds the data bits that

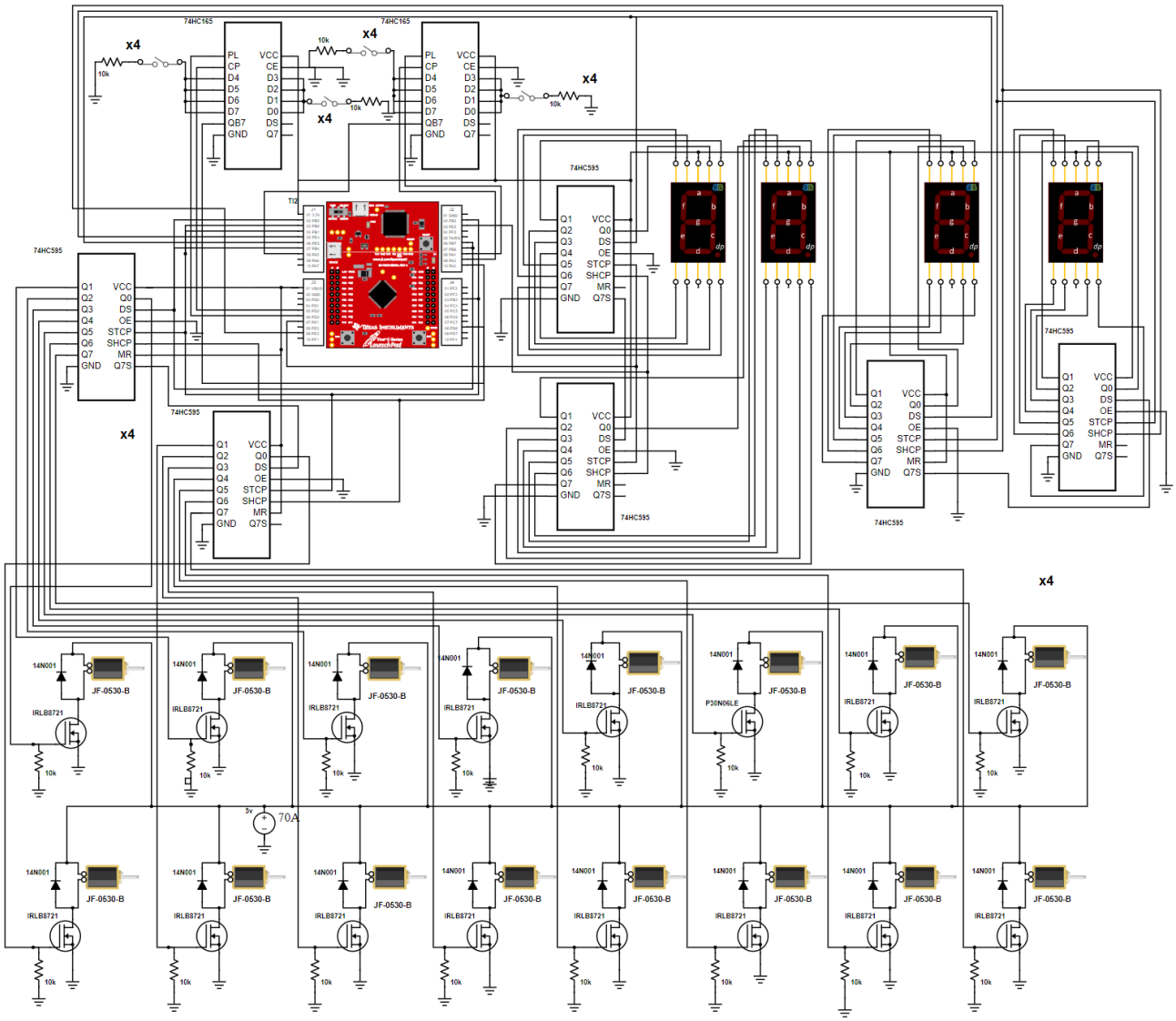
			Player 1 Shift Reg				will be transmitted to the shift register on Player 2 controller
	PE5	Output	Clock pin shared for all SSDs	Digital	3.3v	NA	Holds the clock pulses for all the SSDs
	PD7	Output	Data pin for Player 1 SSD	Digital	3.3v	NA	Holds the data bits that will be transmitted to the shift register on Player 1 SSD
	PE1	Output	Latch pin for Player 1 SSD	Digital	3.3v	NA	Sends HIGH signal to latch to update the shift register on the Player 1 SSD and LOW signal when not updating
	PE3	Output	Data pin for Player 2 SSD	Digital	3.3v	NA	Holds the data bits that will be transmitted to the shift register on Player 2 SSD
	PE4	Output	Latch pin for Player 2 SSD	Digital	3.3v	NA	Sends HIGH signal to latch to update the shift register on the Player 2 SSD and LOW signal when not updating

Player 1 Controller	Signal Name	Input/Output	Description	Signal Type	Voltage	Current	Operration
	SW1	Input	Sends a signal to MCU	Digital	3.3v	NA	Outputs a HIGH signal when the player wants to pause the game
	SW2	Input	Sends a signal to MCU	Digital	3.3v	NA	Outputs a HIGH signal when Player 1 wants to exit professor mode
	SW3	Input	Sends a signal to MCU	Digital	3.3v	NA	Outputs a HIGH signal when Player 1 wants to select the gaming settings or launch the ball
	SW6	Input	Sends a signal to MCU when player wants to move	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move left

	SW7	Input	Sends a signal to MCU	Digital	3.3v	NA	Outputs a HIGH signal when Player 1 wants to cycle through gaming setting or move to the right
	SW8	Input	Sends a reset signal to MCU	Digital	3.3v	NA	Outputs a HIGH signal when Player 1 wants to restart the entire game
	L Button	Output	Sends a signal to Master MCU	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move Left
	R Button	Output	Sends a signal to Master MCU when player wants to move	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move Right
	1 Player Button	Output	Sends a signal to Master MCU	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move play by himself
	2 Player Button	Output	Sends a signal to Master MCU	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move play with 2 people

Player 2 Controller	Signal Name	Input/Output	Description	Signal Type	Voltage	Current	Operration
	SW3	Input	Sends a signal to MCU	Digital	3.3v	NA	Outputs a HIGH signal when the players want to launch the ball
	SW6	Input	Sends a signal to MCU when player wants to move	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move left
	SW7	Input	Sends a signal to MCU when player wants to move	Digital	3.3v	NA	Outputs a HIGH signal when the players want to move right

Complete Schematic



Bill of Materials

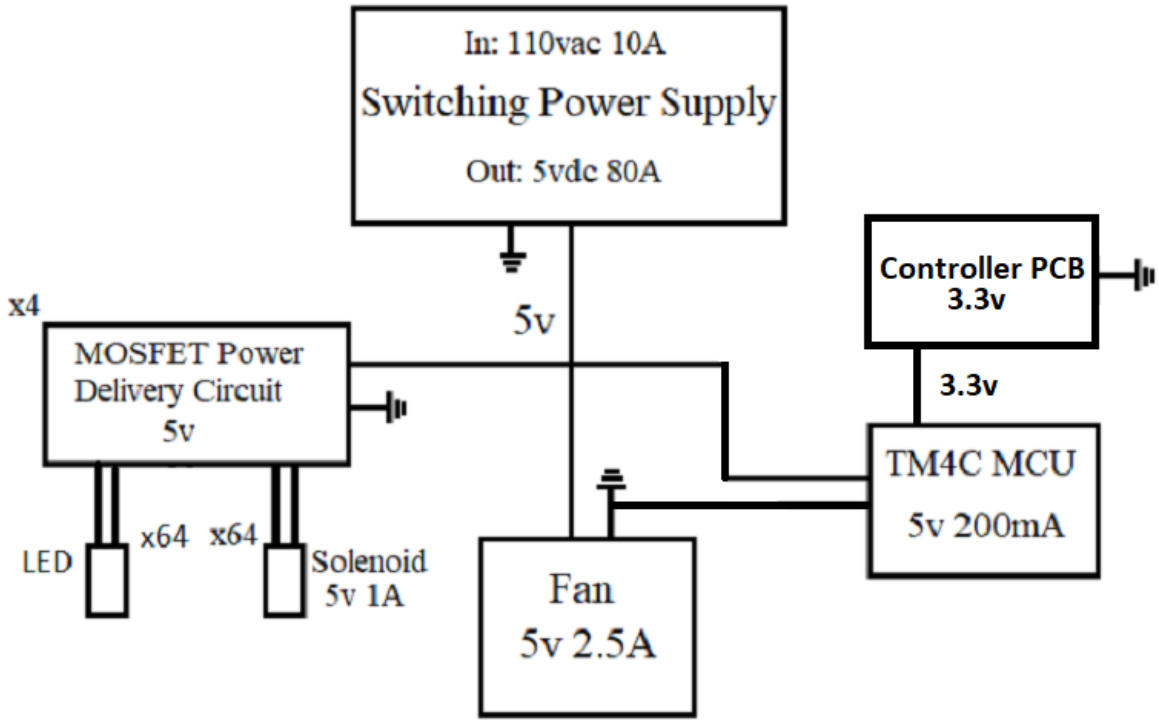
Part	Part #	Description	Package Used	Distributor	Cost	Quantity
Micro controller	tm4c123 gh6pm	5v 200mA micro controller		Digikey	$20	5
Solenoid	JF-0530 B	6v 1A Electro Magnet		Aliexpress	$1.15	64
Power supply	YC5V80	5V 80A		Amazon	$22	1

	ATUS	Power supply				
Mosfet	IRLB8721PBF	N Channel Mosfet with low gate voltage		Amazon	$11	7 (10 pack)
Diode	1N4001	1A 50V diode		Amazon	$6	1 (100 pack)
10K Resistor	[E5P011] 10K ohm	10K ohm/ 0.5w/ +-1%		Amazon	$6	1 (100 pack)
24 Gauge wire	26121539	36ft of solid core wire		Amazon	$14	1 (36ft)
White LED	x00143nkf5	Ultra White LED		Amazon	$10	1(100 pack)
Solenoid Circuit PCB	NA	PCB for Solenoid Circuitry		PCBWay	$50	1(5 pack)
Final Controller PCB	NA	PCB for Final Controller containing Seven Segment Displays		PCBWay	$50	1(5 pack)
8 bit Serial-In-Parallel-Out Shift Register	74HC595	Shift Register for Controlling Solenoids and Seven Segment Display		Amazon	$6.99	1(25 pack)
8 bit Parallel-In-Serial-Out Shift Register	74HC165	Shift Register for Controlling Player Controller		Digikey	$8	1(8 pack)
Wood	NA	Wood game table used to set solenoids in place and protect project		Home Depot	$30	1

		circuitry				
Resin	NA	Plastic used to create blocks		Amazon	$20	1

Project Total Cost - $383.59

Power Management



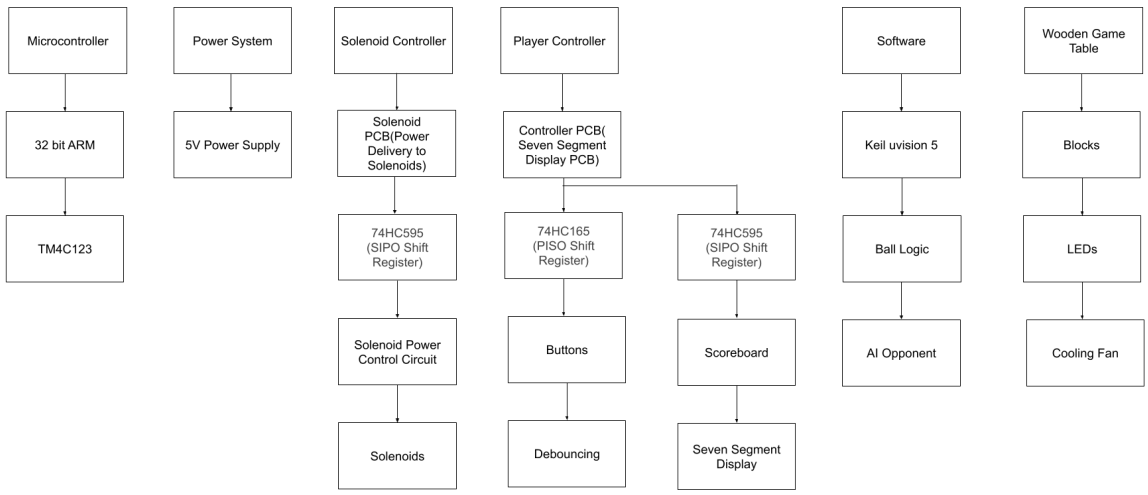
Power Budget:

- **Fan:** 2.5A (Constant Draw)
- **MOSFET Circuit:** 64A(Total Solenoid draw) + 1.28A(Total LED draw)
- **TM4C MCU:** 200mA (Total Draw)
- **Controller PCB**: 10mA(Total Draw)

Total Power Budget: 81.28A
Used(Min - Max): 3.5A - 68.68A

Project Management – WBS, Gantt Chart, and the 490B 5 Demo Contract

WBS



Gantt Chart

Task Name	Status	Start Date	End Date	Assigned To	Duration	Description
Planning/Reports		**01/31/22**	**12/14/22**		**228d**	
Project Brainstorming	Complete	01/31/22	01/31/22	All Members	1d	3 main ideas to discuss
Project Proposal 1	Complete	01/31/22	01/31/22	All Members	1d	Presenting main idea of 3d game table
Presentation Planning and Practicing (Midterm)	Complete	03/02/22	03/14/22	All Members	9d	Explaining 3d game table in greater detail
CECS490A Project Midterm Report/Proposal	Complete	03/14/22	03/14/22	All Members	1d	Report and explaining the 3d game table with all necessary details
Work Breakdown	Complete	03/02/22	03/07/22	All Members	4d	Diagram that describes all concepts of the project and how they connect with
Block Diagram	Complete	03/07/22	03/07/22	All Members	1d	Diagram that shows the complete structure of the project
Bill of Materials (BOM)	Complete	03/07/22	03/14/22	All Members	6d	Defining tools and costs of materials
CECS 490A Final Project Report	Complete	04/11/22	05/03/22	All Members	17d	Final Report with all deliverables for Part1
Schematic Design	Complete	04/18/22	04/22/22	All Members	5d	Detailed schematic of the project
Schematic Design with PCB	Complete	12/05/22	12/12/22	JR Joshua Rodriguez	6d	Detailed schematic of the project with PCB
Work Breakdown 2.0	Complete	04/24/22	04/24/22	All Members	1d	Updating Work Breakdown
Poster Board	Complete	04/24/22	05/02/22	All Members	7d	Building Poster Board to display project
Poster Board 2.0	Complete	12/12/22	12/13/22	CV Carlos Verduzco	2d	Building new Poster Board for Expo
CECS 490B Midterm Report	Complete	08/17/22	10/14/22	All Members	43d	Midterm Report with all deliverables
CECS490B Final Report	In Progress	10/14/22	12/14/22	All Members	44d	Final Report with all deliverables
CECS490A Expo Presentation	Complete	05/03/22	05/03/22	All Members	1d	Presenting 1st part of project to other companies and corporations
CECS490B Expo Presentation	Not Started	12/14/22	12/14/22	All Members	1d	Presenting complete project to other companies and corporations
490B 5 Demo Contract	Complete	04/25/22	04/27/22	All Members	3d	Describing Demos that will be made next semester
Gannt Chart	Complete	04/24/22	05/02/22	JR Joshua Rodriguez	7d	Gannt Chart displays all work over 2 semesters
Overall System Functional Block Diagram	Complete	05/01/22	05/02/22	All Members	2d	Block Diagram showcasing all connections
Overall System Functional Block Diagram 2.0	Complete	10/02/22	10/05/22	NB Nicholas Bishop	4d	Block Diagram showcasing all connections with PCB
Power Management System	Complete	04/30/22	05/02/22	NB Nicholas Bishop	2d	Diagram of all Power used in project

Task Name	Status	Start Date	End Date	Assigned To	Duration	Description
Software Development		**02/17/22**	**12/30/22**		**227d**	
Software Development UART	Complete	04/11/22	04/13/22	JR Joshua Rodriguez	3d	Programming master and slave MCU
UART Debugging	Complete	04/13/22	04/18/22	All Members	4d	Fixing problems with UART communication
Single Solenoid Software	Complete	02/17/22	02/17/22	CV Carlos Verduzco	1d	Programming one solenoid to move
Ltspice Simulation	Complete	04/24/22	04/27/22	CV Carlos Verduzco	4d	Simulation of all Electrical Circuitry
Shift Register Implementation	Complete	08/15/22	08/18/22	CV Carlos Verduzco	4d	Programing of Shift Register to light LEDs
Paddle Controller Software	Complete	09/15/22	09/22/22	CV Carlos Verduzco	6d	Programming of interaction between ball and paddle pixel
Ball Logic	Complete	09/01/22	09/09/22	CV Carlos Verduzco	7d	Programming of ball speed and motion
Greater Controller Functionally	Complete	10/08/22	10/14/22	CV Carlos Verduzco	6d	Programming of controller with 8 buttons
Seven Segment Display Logic	Complete	10/23/22	10/27/22	CV Carlos Verduzco	5d	Programing of Scoreboard with SSD
UART Board Simulation	Complete	10/12/22	10/14/22	CV Carlos Verduzco	3d	Programing of Virtual Simulation of game with UART
Random Generator for Ball Motion	Complete	10/15/22	10/17/22	JR Joshua Rodriguez	2d	Programming of random motion of ball
AI Implmentation	Complete	12/23/22	12/30/22	CV Carlos Verduzco	6d	Programming and Implementation of AI in game
Game Settings	Complete	11/12/22	11/15/22	CV Carlos Verduzco	3d	Programming of Main Menu
Standby Mode	Complete	11/08/22	11/11/22	CV Carlos Verduzco	4d	Programming of Standby Mode
Hardware Development		**02/17/22**	**12/14/22**		**215d**	
Single Solenoid Circuit	Complete	02/17/22	02/17/22	NB Nicholas Bishop	1d	Making one solenoid to lift up a block
Led Solenoid Circuit	Complete	03/29/22	03/31/22	All Members	3d	Making a row of LEDs light up with UART
Row of Solenoids Circuit	Complete	03/29/22	04/20/22	NB Nicholas Bishop	17d	Making several rows of solenoids move up and down
Moving Blocks Circuit	Complete	04/29/22	05/02/22	NB Nicholas Bishop	2d	Making row of solenoids lift up and down blocks
Solenoid Driver Board PCB	Complete	07/09/22	07/26/22	NB Nicholas Bishop	13d	Making PCB for Solenoid Electrical Circuitry
Led Test Circuit	Complete	08/18/22	09/05/22	CV Carlos Verduzco	13d	Making LED Circuit to Test and Debug Game Code
IR Sensor Circuit	Complete	09/22/22	09/26/22	JR Joshua Rodriguez	3d	Making Circuit to develop random number generation for ball
Block Implementation and Generation	Complete	09/26/22	10/24/22	NB Nicholas Bishop	21d	Making Blocks that will be lifted by solenoids
Controller PCB	Complete	10/07/22	10/17/22	CV Carlos Verduzco	7d	Making PCB for player controllers
Seven Segment Display PCB	Complete	10/17/22	11/02/22	JR Joshua Rodriguez	13d	Making PCB for Scoreboard
Controller With SSD PCB	Complete	11/23/22	12/01/22	JR Joshua Rodriguez	7d	Combining Player controllers and Scoreboard into 1 PCB
Hardware Failure	Complete	11/21/22	11/21/22	All Members	1d	Entire project died due to failed shift registers
Hardware Reconstruction	Complete	11/21/22	12/14/22	All Members	18d	Rebuilding everything about the project
PCB Demo	Complete	08/22/22	09/07/22	All Members	13d	Having 2 rows of solenoids working with PCBs
All Solenoids Demo	Complete	09/07/22	09/21/22	All Members	11d	Having all 4 rows of solenoids working with PCBs
All Solenoids with Controllers Demo	Complete	09/21/22	10/19/22	All Members	21d	Having controllers implemented to control paddles
All Solenoids with AI Demo	Complete	10/19/22	11/02/22	All Members	11d	Whole game working with scoreboard
Final Project Demo	In Progress	11/02/22	11/16/22	All Members	11d	Entire project functioning with AI included
Final Demo	Not Started	12/14/22	12/14/22	All Members	1d	Entire project is complete and presented

5 Demos Contract (Updated)

Demo 1: Two Rows of Solenoids with a PCB

- **Objective:** To test and verify that two rows of solenoids are able to be activated correctly while connected to a custom made PCB.
- **Verification:** This test will showcase how a shift register will update and transmit specific binary values into a set of 8 solenoids . These values can be inputted manually and will be in the format of 8 bit binary numbers. Certain solenoids will be activated based on these values.
- **Measurable Outcome:** A successful demo would be if any set of 8 bit binary numbers can be set for the row data, and the correct solenoids activate based on these inputs.

Demo 2: All Rows with PCBs

- **Objective:** To test and verify that eight rows of solenoids are able to be controlled correctly while connected to custom made PCBs.
- **Verification:** This test will showcase how 8 shift registers will work in conjunction to update and transmit values into all sets of solenoids on the board. The solenoids will activate based on these values and send confirmation messages back.
- **Measurable Outcome:** A successful demo would be if the shift registers are able to quickly and accurately send data to all the solenoids. All solenoids should properly represent the data being sent by the shift registers.

Demo 3: Functioning Controllers Added In with a PCB

- **Objective:** To test and verify being able to have two players control solenoids in the rows representing their paddles using controllers connected to a custom made PCB.
- **Verification:** This test will show the paddle that is represented by blocks, move left or right according to input from the controllers. A message on the serial terminal will display saying which direction the paddle has moved and which player's paddle is the one that moved.
- **Measurable Outcome:** A successful demo would be the paddles moving left or right correctly based on the button pressed on the controllers. Paddle must also be unable to move left or right when it reaches the edges of the row it is in (reaches boundaries of the screen). Paddles must be able to move at the same time visually if both players press a button.

Demo 4: Game and Scoreboard

- **Objective:** To test and verify being able to play a game of pong between two players with a seven segment display updating the score.

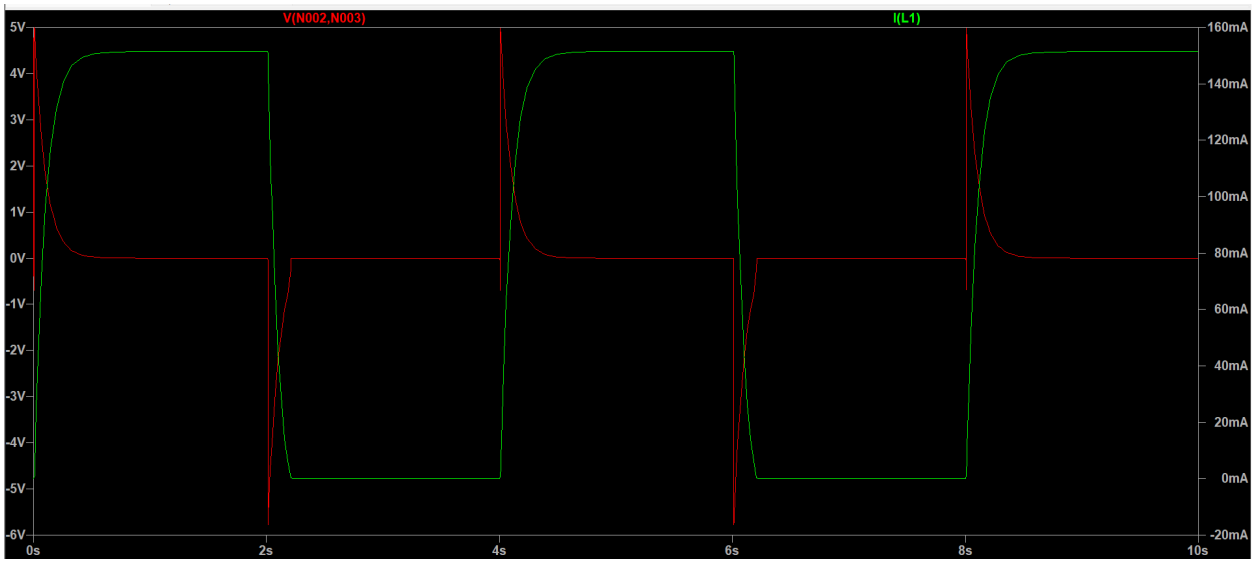
- **Verification:** This test will show a block representing the ball moving across the 8x8 grid and change its position when it interacts with the paddle. If the paddle hits the ball, then the ball will move straight across the screen or move across the screen diagonally in two unique directions. If the ball reaches the row where a player's paddle is located without the paddle hitting it, the opposing player scores a point with the seven segment display updating to show this. The ball will then return to the middle of the screen to start the next round.
- **Measurable Outcome:** A successful demo would be being able to play a full game of pong until the score limit of 9 is reached.

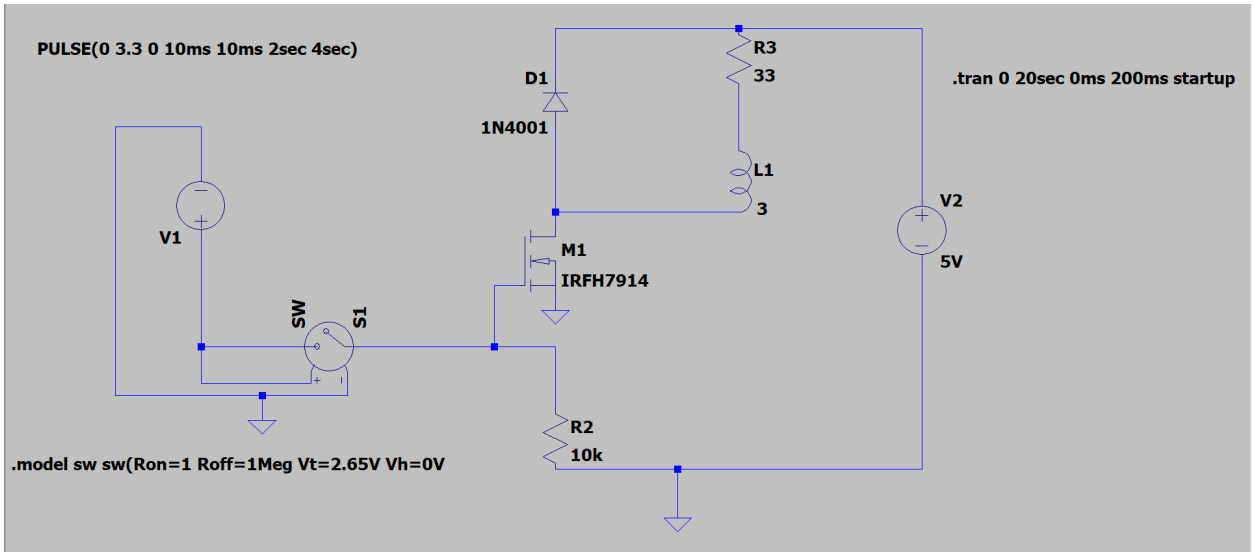
Demo 5: AI Added Into Game and Table Completed

- **Objective:** To test and verify being able to play a game of pong against an AI with the table the game will be encased in completed.
- **Verification:** This test will function the same as demo 4, however only one real player will be controlling a paddle. The other paddle will be controlled by an AI that will move it based on the current position of the ball. The AI will sometimes miss hitting the ball in order to give the player a chance at winning.
- **Measurable Outcome:** A successful demo would be if the AI is able to hit the ball back for the duration of the game. It must also be able to be beaten by the real player.
-

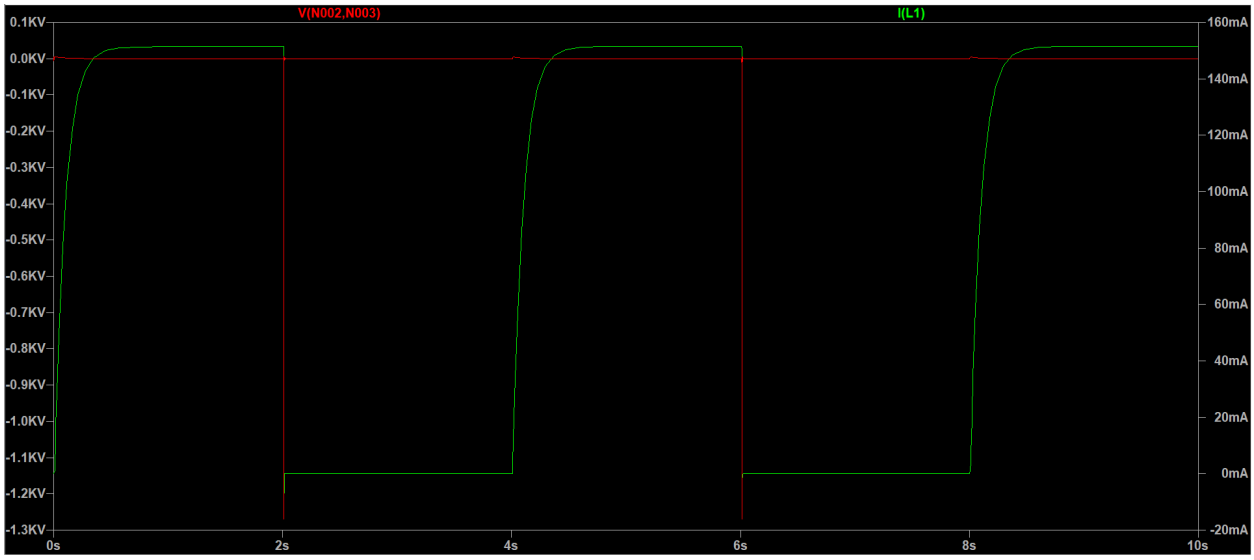
Circuit Simulations and Verification

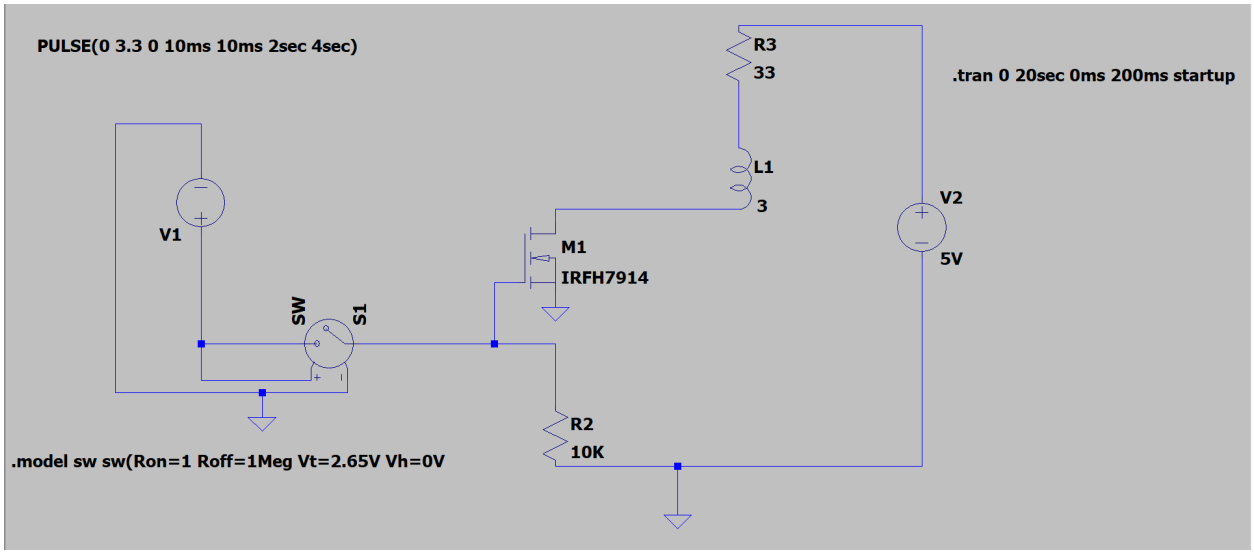
LTspice Simulations for Solenoid Circuit





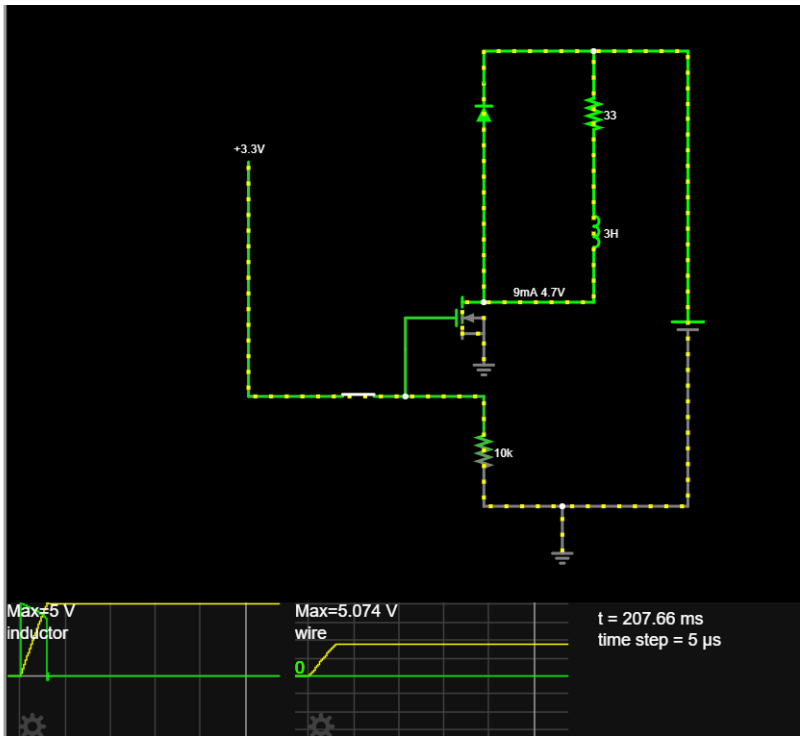
Diode in place: In this hardware simulation we have a diode in place in between the solenoid so that when the circuit is shut off the excess power that bleeds off from the collapsing magnetic field of the solenoid will slowly bleed off.





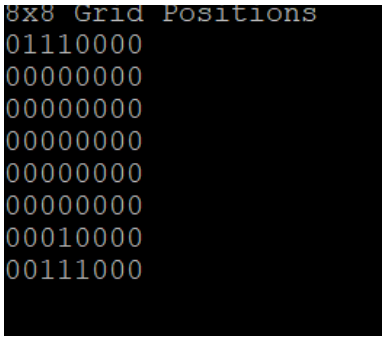
No Diode: In this simulation we remove the diode that's parallel with the solenoid. When power is cut off from the solenoid, the magnetic field will start to collapse on the solenoid and will generate a very high flyback voltage of -1.3KV that will destroy components inline with it.

Falstad Simulation



Serial Terminal Verification

Throughout the process of writing the code, there were many instances in which the game did not behave as intended. In order to debug this, we decided to make use of the UART capabilities of the MCU to see information on a serial terminal. We could display any data to verify it was correct such as information from the buttons, data sent to the shift registers, or the current state of the game. Below we can see the bit values corresponding to activated solenoids. Even with no hardware being plugged in, the grid would update on the terminal to see a game of pong being played by the AIs.



Complete Source Code

Complete source code is available on our Github page:
https://github.com/NickBishop97/Solenoid-Pong-Game

Expo Poster

3D Pong Game

Team Magnetic Flux

By: Nicholas Bishop, Joshua Rodriguez, Carlos Verduzco

Overview

This project is a three dimensional game table that uses blocks to simulate a game of pong. These blocks are controlled by devices called solenoids, that can move them up and down based on the state of the game.

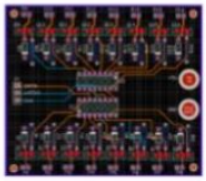
Shift Registers

A limited amount of I/O pins on the microcontroller would cause problems on how to control all the solenoids, buttons, and seven segment displays. Shift Registers allow us to save pins by sending or receiving data using only a few pins. This is done by data being sent out or received one bit at a time every clock cycle. The speed of this process seems instantaneous when witnessing, allowing the entire game to function properly.

Test Circuit

8x8 grid of LEDs organized in the same manner as the 3D Pong Game. This allowed us to test the software and game logic here before testing it on the solenoids.

Solenoid Driver Circuit PCB

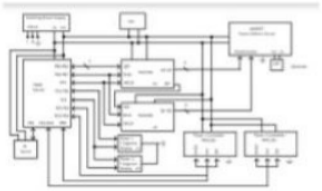


4 Layer PCB that includes all the circuitry for powering 16 solenoids. Includes SIPO shift registers, N-channel MOSFETS, flyback diodes, resistors, and connectors for the microcontroller and power supply

Game Settings

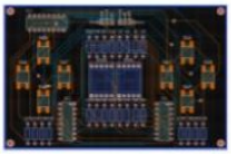
2P - 2 Player Mode, 1P - 1 Player against AI
d1, d2, d3 - Easy, Medium, and Hard AI difficulty
P3, 5, 7, 9 - Max Score Limit

Hardware Block Diagram



Controller PCB

2 Layer PCB representing the player controllers and scoreboard. Allows for multiple functionalities to be performed, as well as seeing the current score at all time. Includes PISO and SIPO shift registers, buttons, seven segment displays, resistors, and connectors for the microcontroller and power supply



Controller Functionalities

SW1 - pauses/unpauses game, SW2 - exits special mode,
SW3 - selects game setting/launches ball, SW6 - moves paddle to the left
SW7 - cycles through game settings/moves paddle to the right,
SW8 - restarts entire game

Project Progress