Final Project Report

Zachary Bradley

A0270061

Joshua Christiansen

A02304980

1. Introduction.

The final project discussed in this paper is Tic-Tac-Toe. The Tic-Tac-Toe project consists of four main parts: the STM32L476 microcontroller, the Liquid Crystal Display (LCD), the five push buttons, and the UART display. To start this project, a new µVision project was created and then previous labs were used to cover sections of code previously implemented. Once the project had been created, the next step was to create a wiring diagram. This diagram shows all the GPIO port connections, and how to properly wire the LCD. The next step was to begin writing code and resolving problems. The first course of action was to be able to create and display a Tic-Tac-Toe board onto the screen using UART. By reusing previous labs' code, this became an easy thing to accomplish. After UART was established, the next step was to incorporate a user-friendly display to play Tic-Tac-Toe. The decision was then made to use a board that contained a cursor and a

temporary variable. The way this works is the cursor is represented by a lowercase ‘x’ and ‘o’. The cursor moves around the board before being placed. If there are any ‘X’ or ‘O’ on the board, the cursor can move over these characters, and it will show the lowercase ‘x’ and ‘o’ until the cursor is then moved off the placed character. The push buttons allow the cursor to move and eventually place an ‘X’ or ‘O’ character to the board permanently. The next step was to complete the game logic of Tic-Tac-Toe. This consists of checking for a winner by three in a row. The logic incorporated checking both diagonals of the board, three rows, and three columns. If all characters are placed in these rows, columns, or diagonals, the player whose character it corresponds with wins the game. Once the game logic was incorporated, the next step was to print the winner to the LCD. The code for this was found from previous labs that had incorporated the LCD before. Once the proper messages were printed, the next step was to end the game on the screen. To accomplish this, ASCII art was used to print “Tic-Tac-Toe” to begin the game, and “Game Over” to end the game. The user was then prohibited from playing the game any further. The functionality of Tic-Tac-Toe was implemented using the C programming language.

2. Scope

The scope of this document will cover the project introduction, design overview which includes requirements, dependencies, theory of operation, and design

alternatives, design details, testing, conclusion, and the appendix. The appendix is where images will be located, and they will be referenced throughout the document. This document will cover specifics on how the project was conceived, designed, and implemented. Some things this document will not cover include screenshots of the entirety of the implemented code and an in-depth analysis of the components used. For the criteria called for in this document, covering the topics mentioned above is sufficient.

3. Design Overview

3.1. Requirements.

The requirements of this project are described here: “A suitable project must have some sort of input -- such as buttons or switches or sensors -- and some sort of observable output. Multiple inputs and/or outputs are ideal. Project difficulty should be commensurate with the 4+ weeks that you have to work on it. It is not sufficient to simply modify previous labs.”

The Tic-Tac-Toe project meets these requirements by using five push buttons as inputs, UART and LCD as output, and the difficulty of the project was enough to last four weeks. Although previous labs were used in the design, they were not simply modified, there was incorporation of entirely new concepts and designs to complete the Tic-Tac-Toe project.

Another piece of the project requirements include: “The goal of this proposal is to make a convincing argument that the project is **feasible but challenging**. Describe the basic functionality you want to achieve and the parts and materials you will need to do so. High level block diagrams, flowcharts, etc. are useful here.”

To meet these project requirements, the documentation below describes the initial idea of the project:

Our project is going to be Tic-Tac-Toe. Our description is below:

1. Our input will be from five buttons. There will be an up, down, left, right, and center button. The buttons up, down, left, and right will control where the player is going to place their ‘X’ or ‘O’ but will not be placed until the center button is pressed.
2. Our observable output will be printed to the computer screen via UART. After discussing our idea with our TA, we think this is totally feasible.
3. We have multiple inputs as we will have 5 buttons to move direction on the screen. We will also have multiple outputs as we plan to print “Player (X or O) Wins!” on the LCD.
4. We believe this project to be difficult enough to warrant 4+ weeks of work as we are currently unfamiliar with UART. We also believe we have all the resources in our microcontroller kit to be able to do this project.

5. The basic functionality of this project is to be able to play Tic-Tac-Toe with a friend using a microcontroller, a computer screen, an LCD, and 5 buttons. We also will need a USB cable to connect the microcontroller USB protocol to the computer, which we already have.

We were able to accomplish all the tasks described in the documentation above for this project.

3.2. Dependencies.

The design depends on the microcontroller to provide power to the push buttons and the LCD. The microcontroller has ports that provide 5 volts of power as well as a ground source that was connected to the positive and negative terminals of the two breadboards used in this project. The microcontroller depends on the USB connected to the PC to provide power to itself. The wiring realization is shown in *Figure 3.*

3.3. Theory of operation.

Figure 1 shows the schematic of the port connections in a high-level overview of the total system. As seen in the image, the microcontroller communicates to the PC using Universal Asynchronous Receiver and Transmitter (UART). UART is sent through the USB port of the PC to communicate directly to the PC interface. The way UART works is through asynchronous communication. This means both the

sender and receiver agree on a baud rate for data delivery which negates the clock signal. A baud rate is the number of bits physically transferred per second. The baud rate for our project was 9600. FT232R converts the UART port to a standard USB interface which the PC can connect via a cable to the microcontroller. The sender and receiver use the same transmission speed. The data is sent using a series of bits. There is a start bit, data bits, parity bit, and stop bits. The start bit controls the start of transmission. The data bits represent the actual data being transferred. The parity bit is used for error checking and is optional. The stop bits control the end of transmission.

The next section discussed is how the microcontroller communicates with the LCD using GPIOB ports 0 through 7. 5 Volts of power are delivered to the LCD through the microcontroller as well as ground. The way the LCD works is through four basic components: segments and duty cycle, controller tasks, connections, and initialization. The segments work through specific sections being turned on or off using full duty cycle or half duty cycle. Duty ratios are configured using a drive bias. A drive bias is calculated using *Equation 1:* $\frac{1}{Number\ of\ Voltage\ Levels - 1}$. The drive bias controls the number of voltage levels used to illuminate each section of the seven-segment display. The controller tasks include converting ASCII characters into the segment patterns to be viewed on the LCD. This type of processing power makes printing characters to the LCD much easier instead of

having to hard code which segments should illuminate and when. Next is the connections. The LCD has pins such as RS (Register Select), RW (Read/Write), and E (Enable). LED- and LED+ illuminate the LCD while V_o and VDD control the backlight contrast. The full wiring diagram is shown in *Figure 1*. Initialization is configured within the confines of the code. Commands such as LCD_Displaystring, LCD_Init, and LCD_Clear are all examples of initializing the LCD to display the contents.

3.4. Design Alternatives

Another approach for the Tic-Tac-Toe project was to use two boards. One board would be printed to the screen and be updated by a second board after an ‘X’ or ‘O’ is placed. This process would continue, and the same winner logic would be used to determine a winner. This design was not implemented because we concluded it would be simpler to have a temp variable to store the placed ‘X’ and ‘O’ on the board while the cursor moves over those placed squares to show where the user is currently.

Another design alternative was to use a UART receiver to enable an external keyboard to be used. This design alternative was deemed overkill because for our game there are only five needed buttons. A keyboard contains well over five buttons and would need another UART interface to communicate. This design was

then deemed unnecessary, and we decided to go with the five push buttons instead.

4. Design Details

4.1 Microcontroller Port Connections

In the schematic below shown as *Figure 1*, we show that we are going to connect our five buttons to GPIOC[4:0]. These buttons will be connected via an external breadboard and wired directly back to the microcontroller. We will also use a smaller breadboard to connect the LCD to the microcontroller, with the following port connections: GPIOB[1:0] will be connected to the E and R_s ports respectively. A resistor is placed between V_o and a 5V port connection. VDD and $LED+$ are connected straight to 5V. GPIOB[7:4] will be connected to the ports DB7, DB6, DB5, and DB4 respectively. DB0, DB1, DB2, and DB3 are all tied to ground so that the LCD will run in 4-bit mode. We also have a connection between the microcontroller USB protocol and the computer. The UART protocol will be enabled using GPIOA[2:0] set to alternate function mode. *Figure 3* shows the realization of this wiring, and the USB protocol is also shown below as *Figure 2*.

The following code snippets show the realization of these port connections in software. First, the clocks for both GPIOC and GPIOB must be enabled. Then, the MODER register for GPIOB[7:4] and GPIOB[1:0] must be set to output mode to display to the LCD. The MODER register for GPIOC[4:0] must be set to input

mode to receive input from the buttons. The PUPDR register for GPIOC[4:0] must also be set to pull-up because the buttons are wired to ground.

```
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN; //GPIOB clock enable
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN; //GPIOC clock enable


GPIOB->MODER &= 0xFFFF00F0;
GPIOB->MODER |= 0x00005505;  //Sets GPIOB[7:4] and GPIOB[1:0] MODER to 01/output mode for LCD

GPIOC->MODER &= 0xFFFFFC00;
GPIOC->MODER |= 0x00000000;  //Sets GPIOC[4:0] MODER to 00/input mode for buttons

GPIOC->PUPDR &= 0xFFFFFC00;
GPIOC->PUPDR |= 0x00000155;  //Sets GPIOC[4:0] PUPDR to 01/pull-up mode buttons
```

Next, both the HSI clock and the GPIOA clock must be enabled for the LCD and UART respectively. The MODER register for GPIOA[2:1] must be set to alternate function mode, and the AFR register for GPIOA must also be enabled for UART to work correctly.

```
RCC->CR |= RCC_CR_HSION;
while((RCC->CR & RCC_CR_HSIRDY) == 0);  //HSI clock enable
RCC->CFGR |= RCC_CFGR_SW_HSI; //switch to HSI clock
RCC->AHB2ENR |= RCC_AHB2ENR_GPIOAEN; //GPIOA clock enable
//bits 2 and 3 need to be alternate function mode GPIOA (10)
GPIOA->MODER &= 0xFFFFFF0F; //Set wanted bits to 0
GPIOA->MODER |= 0x000000A0; //Bits 2 and 3 are now 10 which is alternate function mode
GPIOA->AFR[0] |= 0x77 << (4*2);
```

4.2 Button Input Logic

The button input logic is handled by a function called readButtons. This function operates by looking at the respective Input Data Register connected to each button, and moving a cursor through an initialized 3x3 character array named Board. The current position of the cursor is monitored by two variables named cursorx and

cursory. If the UP button is pushed, a 1 is added to cursory. If the RIGHT button is pushed, a 1 is added to cursorx. Similar logic is implemented for moving the cursor down or to the left. If the SELECT button is pushed, an X or an O is placed at the current position of the cursor, depending on what player is placing. After placing, the player is switched. Then, the checkWinner function is called after every placement to see if that placement resulted in a win. The following code snippets show the readButtons function. *Figure 6*, *Figure 7* and *Figure 8* show player X and player O winning the game, and a draw game respectively. Each figure shows the board with the placed characters.

```
void readButtons() {

    // Check if the LEFT button (GPIOC 0) is pressed
    if (!(GPIOC->IDR & 4)) {
      if (cursory != 0) {
          //board[cursorx][cursory] = ' ';
          board[cursorx][cursory] = temp;
          cursory -= 1; // Move left
          Debounce();
      }
    }

    // Check if the RIGHT button (GPIOC 1) is pressed
    if (!(GPIOC->IDR & 8)) {
      if (cursory != 2) {
          board[cursorx][cursory] = temp;
          cursory += 1; // Move right
          Debounce();
      }
    }

    // Check if the UP button (GPIOC 2) is pressed
    if (!(GPIOC->IDR & 1)) {
      if (cursorx != 0) {
          board[cursorx][cursory] = temp;
          cursorx -= 1; // Move up
          Debounce();
      }
    }

    // Check if the DOWN button (GPIOC 3) is pressed
    if (!(GPIOC->IDR & 2)) {
      if (cursorx != 2) {
          board[cursorx][cursory] = temp;
          cursorx += 1; // Move down
          Debounce();
      }
    }
```

```
// Check if the SELECT button (GPIOC 4) is pressed
if (!(GPIOC->IDR & 16)) {
  // Check if the selected cell is empty
        if (player == 1) {
          if (temp == ' '){
            board[cursorx][cursory] = 'X'; // Player 1 marks 'X'
            winner = checkWinner();
            if (winner == 'X'){
              LCD_DisplayString(0, "Player X wins");
              writeGameOver();
            }
            else if (winner == 'D'){
              LCD_DisplayString(0, "It's a Draw");
              writeGameOver();
            }
            player = 2;           // Switch to Player 2
          }
        }
        else {
          if (temp == ' '){
            board[cursorx][cursory] = 'O'; // Player 2 marks 'O'
            winner = checkWinner();
            if (winner == 'O'){
              LCD_DisplayString(0, "Player O wins");
              writeGameOver();
            }
            else if (winner == 'D'){
              LCD_DisplayString(0, "It's a Draw");
              writeGameOver();
            }
            player = 1;           // Switch to Player 1
          }
        }
    Debounce();
    Debounce();
}
```

4.3 Writing the Board

Writing the board to UART is handled by a function called WriteBoard. This works by first initializing each row of the board that is later overwritten by the characters placed by the readButtons function. A cursor is placed in the form of little “x” or a little “o” to show the player where they currently are in the board. In order to not overwrite previous placements of a big “X” or a big “O”, a temp variable is used. If a cursor placement would overwrite a previous placement, the previous character is saved into temp. Then later once the cursor has moved off of the square, temp is placed back onto the board at the given position. Every time a cursor or character is placed, an updated version of the board is written to UART.

This works by copying the board to the initialized rows, and then writing them to UART using the USART_Write function created in a previous lab. Formatting characters for the board are also written to UART. The following code snippets show the WriteBoard function. *Figure 4* shows the Tic-Tac-Toe board displayed on UART.

```
void WriteBoard() {
  char Row1[12] =  {' ', 'X', ' ', '|', ' ', 'X', ' ', '|',' ', 'X', ' ', '\n'};
  char Row2[12] =  {' ', 'X', ' ', '|', ' ', 'X', ' ', '|',' ', 'X', ' ', '\n'};
  char Row3[12] =  {' ', 'X', ' ', '|', ' ', 'X', ' ', '|',' ', 'X', ' ', '\n'};
  char Row4[12] =  {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',' ', ' ', ' ', '\n'};
  char Row5[12] =  {'-', '-', '-', '+', '-', '-', '-', '+','-', '-', '-', '\n'};


  if (player == 1) {
    if(board[cursorx][cursory] != 'x'){
      temp = board[cursorx][cursory];
    }
    board[cursorx][cursory] = 'x'; //This just shows player 1's x blinking on the screen to show what will be placed
  }//Check X before placing next o, same with O, check O before placing next x
  //The cursor will affect the game logic if we do it any other way
  //winner = checkWinner();
  //LCD_DisplayString(0, winner);
  if (player == 2) {
    if(board[cursorx][cursory] != 'o'){
      temp = board[cursorx][cursory];
    }
    board[cursorx][cursory] = 'o'; //This just shows player 2's o blinking on the screen to show what will be placed
  }
  //winner = checkWinner();
  //LCD_DisplayString(0, winner);

  Row1[1] = board[0][0];
  Row1[5] = board[0][1];
  Row1[9] = board[0][2];
  USART_Write(Row1, 12);
  USART_Write(Row5, 12);
  Row2[1] = board[1][0];
  Row2[5] = board[1][1];
  Row2[9] = board[1][2];
  USART_Write(Row2, 12);
```

```
    USART_Write(Row5, 12);
    Row3[1] = board[2][0];
    Row3[5] = board[2][1];
    Row3[9] = board[2][2];
    USART_Write(Row3, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
    USART_Write(Row4, 12);
  }
```

4.4 Game Logic

The game logic is handled by a function called checkWinner. This function works by first iterating through all of the rows of the board, then the columns, and then the diagonals. If three of the same characters are detected, the character at the index of the given row, column, or diagonal is returned, either an X or an O. If no winner is detected, the function checks for a draw. This works by iterating through the whole board with a nested for loop, and if a blank space is detected, a space is returned, meaning the game is still in progress. Otherwise, a D is returned signifying that the game is a draw. The following code snippet shows the checkWinner function.

```
char checkWinner() {
    int i;

    // Check rows
    for (i = 0; i < 3; i++) {
        if (board[i][0] == board[i][1] && board[i][1] == board[i][2] && board[i][0] != ' ') {
            return board[i][0]; // Row win
        }
    }

    // Check columns
    for (i = 0; i < 3; i++) {
        if (board[0][i] == board[1][i] && board[1][i] == board[2][i] && board[0][i] != ' ') {
            return board[0][i]; // Column win
        }
    }

    // Check diagonals
    if (board[0][0] == board[1][1] && board[1][1] == board[2][2] && board[0][0] != ' ') {
        return board[0][0]; // Main diagonal
    }
    if (board[0][2] == board[1][1] && board[1][1] == board[2][0] && board[0][2] != ' ') {
        return board[0][2]; // Secondary diagonal
    }

    // Check for draw
    for (i = 0; i < 3; i++) {
        int j;
        for (j = 0; j < 3; j++) {
            if (board[i][j] == ' ') {
                return ' '; // Game still in progress
            }
        }
    }

    return 'D'; // Draw
}
```

4.5 ASCII Art

The ASCII art included printing the words “Tic-Tac-Toe” before each game started, and the words “Game Over” after each game ended. It is handled by the functions writeTicTacToe and writeGameOver. These functions work in a very similar manner to the writeBoard function. “Rows” of characters are initialized with the desired formatting and placement of characters to spell the words, and then they are written to UART one at a time. Delay functions are also called in between writing to give sufficient time for the user to read the messages before the game starts. The following code snippet shows the writeGameOver function. The writeTicTacToe function looks nearly identical. *Figure 5* shows the Game Over screen displayed on UART.

```
void writeGameOver() {
  char Game[33] =  {'8', '8', '8', '8', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Game1[33] = {'8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'o', 'o', 'o', ' ', ' ', ' ', ' ', '8', 'o', 'o', 'o', ' ', ' ', 'o', ' ', 'o', ' ', ' ', ' ', ' ', '8', '8', '8', '8', '8', '8', '8', ' ', '\n'};
  char Game2[33] = {'8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '8', '8', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Game3[33] = {'8', ' ', '8', '8', '8', '8', ' ', '8', 'o', 'o', 'o', '8', '8', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', '8', 'o', 'o', 'o', 'o', 'o', ' ', ' ', ' ', '\n'};
  char Game4[33] = {'8', ' ', ' ', ' ', ' ', '8', ' ', ' ', '8', '8', ' ', ' ', ' ', '8', '8', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Game5[33] = {'8', '8', '8', '8', '8', '8', ' ', '8', '8', ' ', ' ', '8', '8', '8', ' ', '8', ' ', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', '8', '8', '8', '8', '8', '8', '8', '8', '\n'};
  char newline[1] = {'\n'};

  char Over[33] =  {'8', '8', '8', '8', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Over1[33] = {'8', ' ', ' ', ' ', ' ', '8', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', '8', '8', '8', '8', '8', '8', '8', ' ', ' ', ' ', ' ', '8', '8', '8', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Over2[33] = {'8', ' ', ' ', ' ', ' ', '8', ' ', ' ', '8', 'o', ' ', ' ', 'o', '8', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Over3[33] = {'8', ' ', ' ', ' ', ' ', ' ', '8', ' ', ' ', '8', ' ', ' ', '8', ' ', ' ', ' ', '8', 'o', 'o', 'o', 'o', 'o', ' ', ' ', ' ', ' ', ' ', ' ', '8', '8', '8', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Over4[33] = {'8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', 'o', ' ', 'o', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};
  char Over5[33] = {'8', '8', '8', '8', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', '8', '8', '8', '8', '8', '8', '8', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', '8', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '\n'};

  USART_Write(Game, 33);
  USART_Write(Game1, 33);
  USART_Write(Game2, 33);
  USART_Write(Game3, 33);
  USART_Write(Game4, 33);
  USART_Write(Game5, 33);
  USART_Write(newline, 1);

  delay_ms(8000);

  USART_Write(Over, 33);
  USART_Write(Over1, 33);
  USART_Write(Over2, 33);
  USART_Write(Over3, 33);
  USART_Write(Over4, 33);
  USART_Write(Over5, 33);
  USART_Write(newline, 1);
```

5. Testing

5.1 The board shall print to the screen via UART to view observable output.

5.1a Code was re-used from previous labs to ensure that the microcontroller could print to the screen. A Tic-Tac-Toe board was then created and sent through UART to the screen.

5.1b The observations to verify were to view the Tic-Tac-Toe board on the screen.

5.1c The Tic-Tac-Toe board appeared on the screen marking the completion of this test.

5.2 Five push buttons shall be used as input to move the cursor on the screen.

5.2a The test procedure included implementing button logic, and ensuring that one button press was recorded at a time. This required debounce logic which was tested through sending values into a debounce function until it was determined to be the right speed for one button press to be recorded.

5.2b The observations to verify were to view one button press moving the cursor on the screen only one square at a time on the Tic-Tac-Toe board.

5.2c The push buttons only moved the cursor one square at a time for each button press marking the completion of this test.

5.3 An LCD will be incorporated as observable output showing the player that won the game.

5.3a The test procedure began with writing “Tic-Tac-Toe” to initialize the LCD and ensure that it is printing. The next step was configuring the logic for the game winner and printing the winner to the screen when that win was detected.

5.3b The observations to verify were to view the LCD printing “Player X wins!”, “Player O wins!”, or “It’s a draw” after the winner had been determined.

5.3c The LCD printed the correct winner after a win was detected marking the completion of this test.

6. Conclusion

The test results enabled the project to run smoothly. Tic-Tac-Toe was able to be played how we had envisioned, and the game was the same game everyone knows how to play. The performance worked great, and the result is how we expected. There was observable output to both the LCD and the screen using UART. The game logic worked by analyzing three in a row in columns, rows, and both diagonals. The winner was printed to the LCD while the ASCII art “Game Over” was printed to the screen. No button presses could overwrite the previous placed ‘X’ or ‘O’.

The only piece of this project that could be optimized would be to make the LCD easier to read. The LCD had a backlight that was very bright, and the contrast was such that the words were not easily read while looking straight down at the LCD. This design could have been improved by adding a larger resistor between V_o and ground. For the sake of this project, the functionality was about as good as it could be for a game of Tic-Tac-Toe.

In conclusion, the game performed as expected, met all requirements, and achieved all test cases.

Appendices

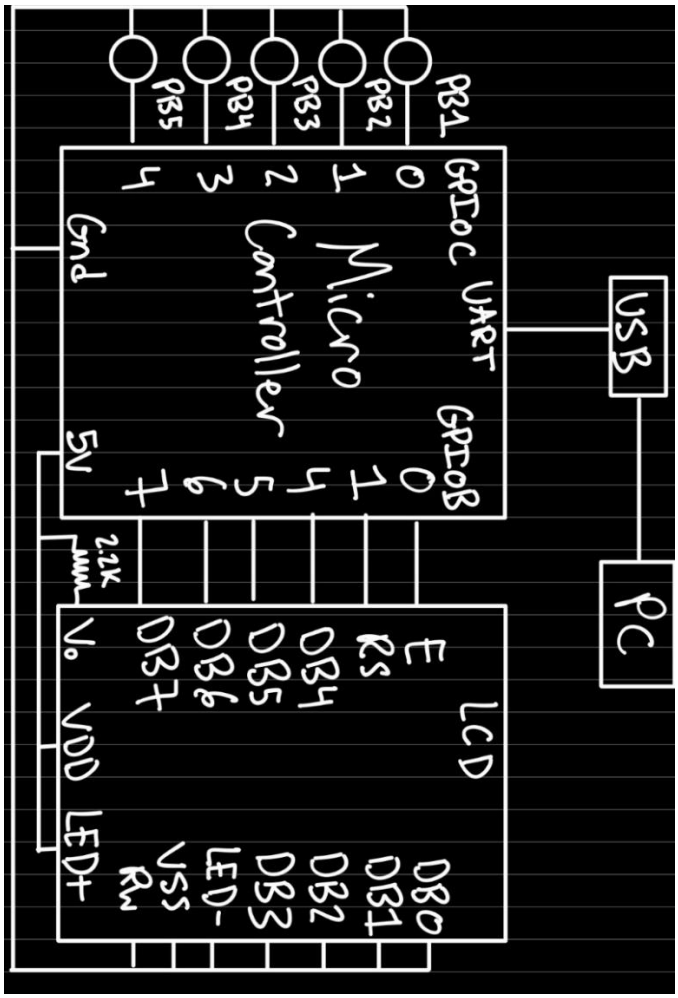

Figure 1 Schematic of Port Connections

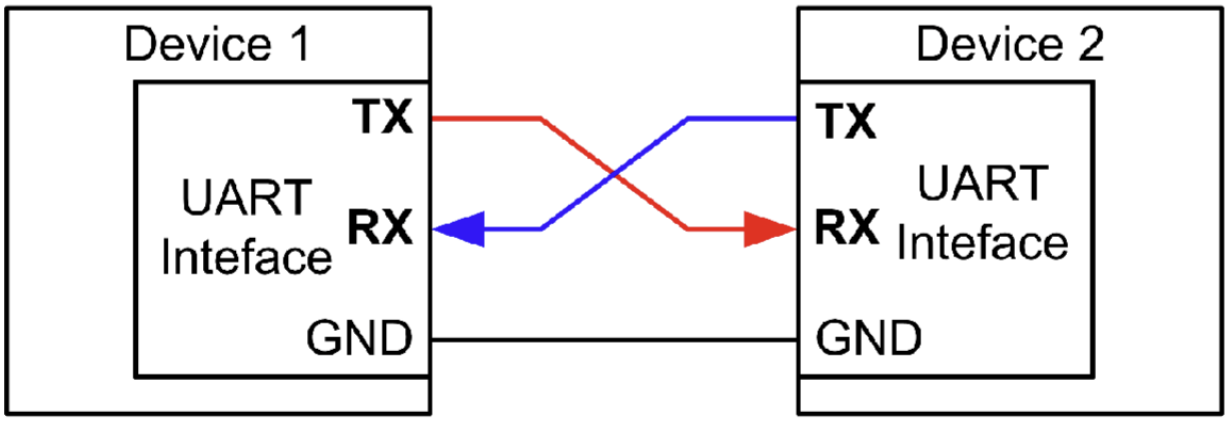

Figure 2 USB Protocol.

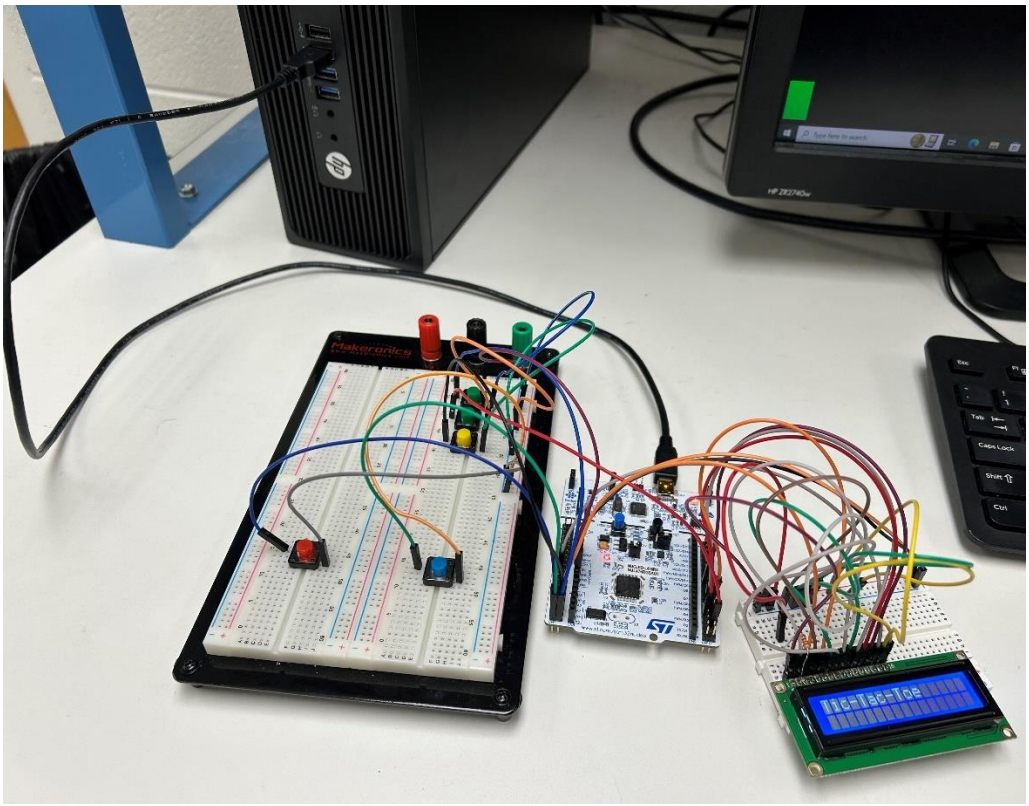

Figure 3 Wiring Overview

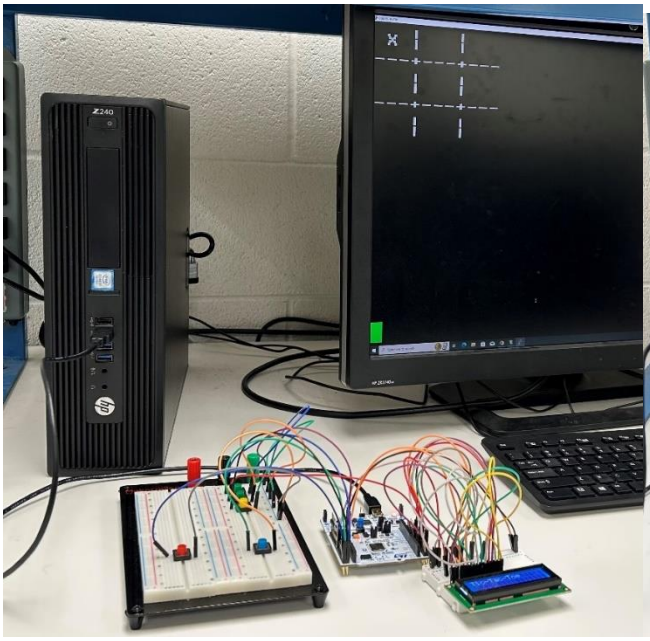

Figure 4 Tic-Tac-Toe Board Displayed

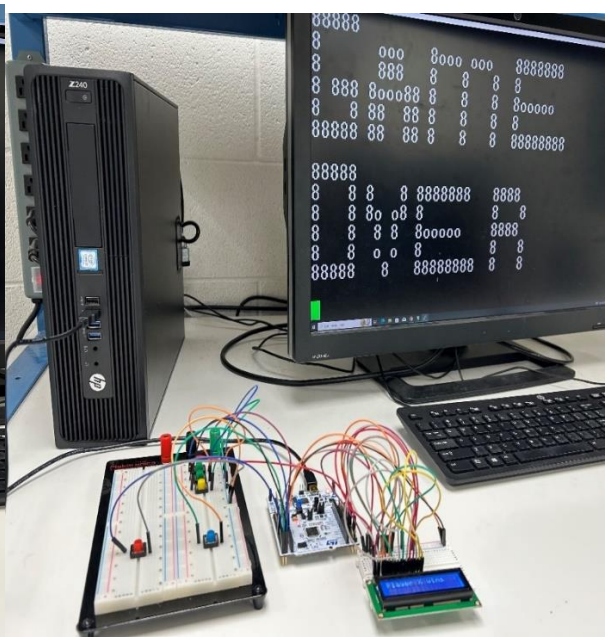

Figure 5 Game Over Screen

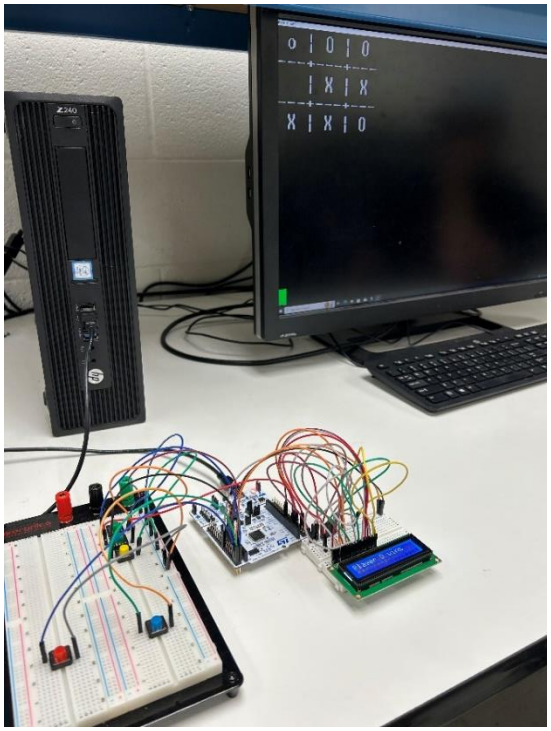

Figure 6 Player O Winning Game

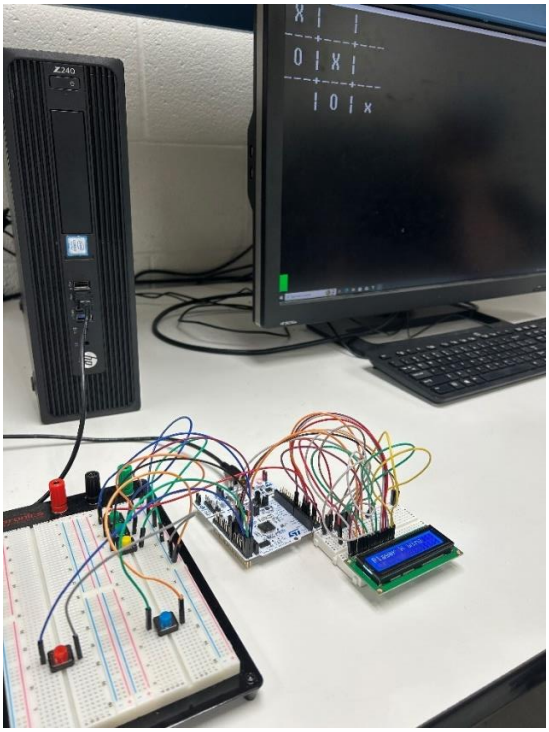

Figure 7 Player X Winning Game

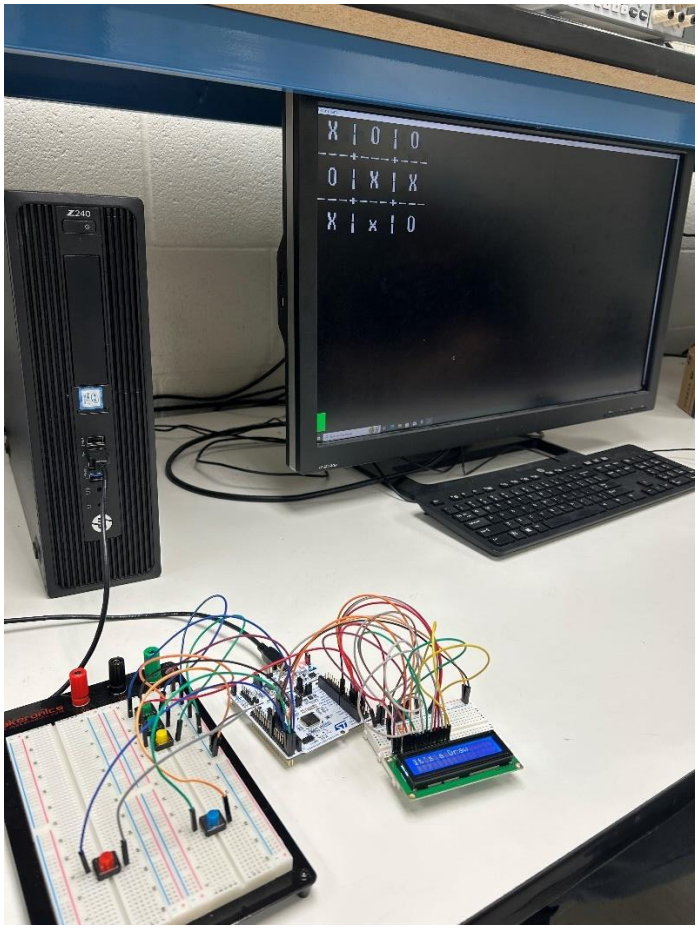

Figure 8 Draw Game