Battleship

C8051 Microcontroller

Jeffrey Pistacchio

Christopher Pybus

Microprocessor Systems

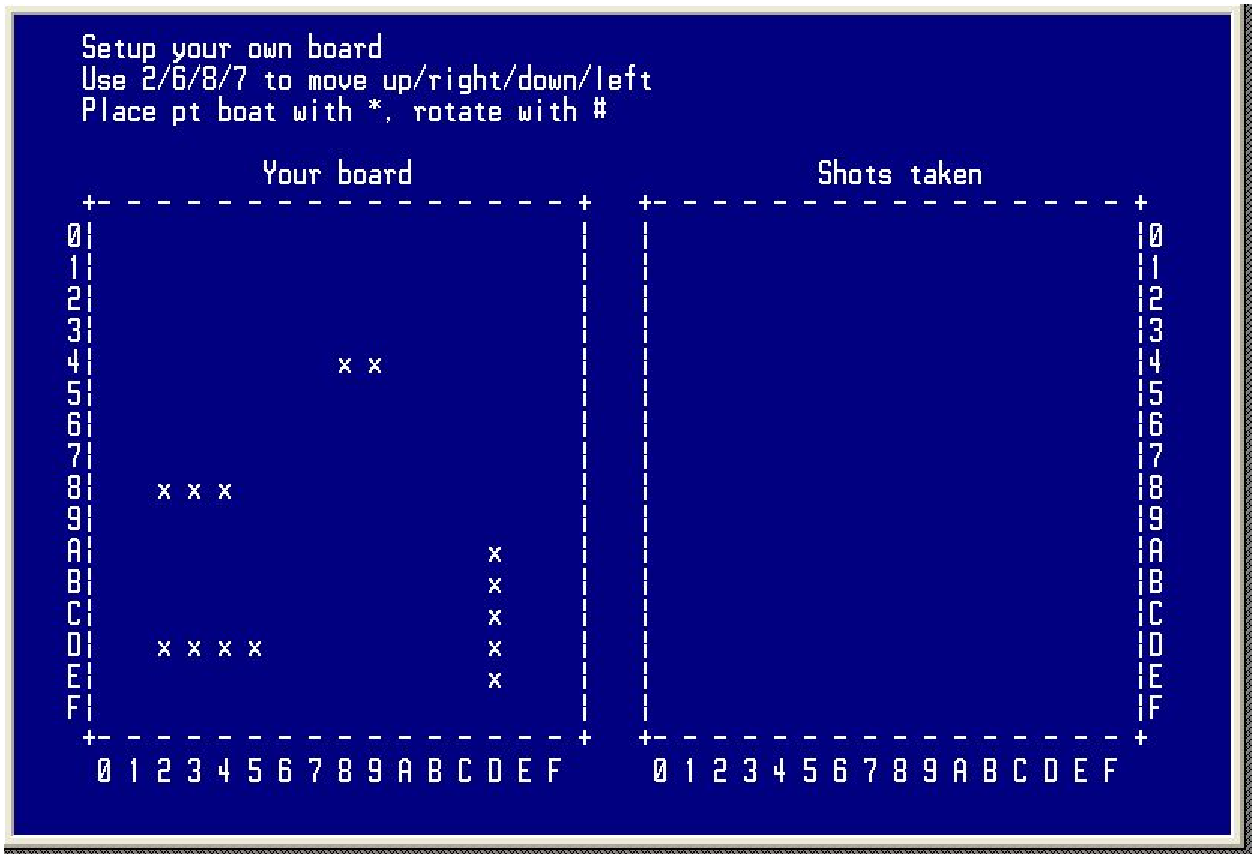
13 December 2016

Introduction

The game “Battleship” is a classic game that many adults today enjoyed in their childhood. The rules of the game are simple; sink the opponent's ships before they sink yours. The goal of this lab was to simulate this game using two C8051 Microprocessors while interfacing UART communication, LCD screen and Keypad input. The keypad serves as a mode of input to take each player’s guess as to where their opponents ship is located. The LCD screen displayed whether the guess was a hit, miss, or sunk ship. The two microcontrollers would communicate with each other to verify this information using UART1. The microcontrollers communicate with their respective terminal displays to show the user board over UART0.

A game is intended to proceed as follows. Each player has a terminal which displays their own board and a board to show where they have fired. Each player will use their keypad to place each ship in its respective place on their own board, using the keys 2, 4, 6, and 8 as arrow keys, * to rotate the ship and # to place the ship. Once each board is complete and both players are ready, each player is prompted for a coordinate to shoot at the other player’s ship. The C8051 uses UART1 to communicate with the other C8051 to see if the guess was a hit or miss. An ‘X’ or an ‘O’ is placed on the target board as a hit or miss, respectively. This process repeats until one player no longer has any ships, leaving the other player as the victor and commander of the seas. An example of the board can be seen in Figure 1.

Figure 1: Terminal Example



```
 Setup your own board
 Use 2/6/8/7 to move up/right/down/left
 Place pt boat with *, rotate with #

             Your board                            Shots taken
 +- - - - - - - - - - - - - - - - +   +- - - - - - - - - - - - - - - - +
0|                                |   |                                |0
1|                                |   |                                |1
2|                                |   |                                |2
3|                                |   |                                |3
4|                x x             |   |                                |4
5|                                |   |                                |5
6|                                |   |                                |6
7|                                |   |                                |7
8|    x x x                       |   |                                |8
9|                                |   |                                |9
A|                          x     |   |                                |A
B|                          x     |   |                                |B
C|                          x     |   |                                |C
D|    x x x x               x     |   |                                |D
E|                          x     |   |                                |E
F|                                |   |                                |F
 +- - - - - - - - - - - - - - - - +   +- - - - - - - - - - - - - - - - +
  0 1 2 3 4 5 6 7 8 9 A B C D E F      0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Methods and Procedures

Part 1

The first part of the project was obtaining input from the keypad. Much of this section used recycled code from lab six. The keypad and LCD played a large role in this project. All user input was obtained through the keypad. The vertical wires (columns) are all help high to 5 volts with a 10k pull up resistor. The horizontal wires (rows) are all held low. The vertical wires are also all connected to an AND gate, and the output of that gate is connected to the INT0 interrupt on the 8051 microprocessor. When a button is pressed, it shorts the vertical wire to ground, which changes the output of the AND gate and sends a signal to INT0, telling the 8051 that a button has been pressed, but nothing else about which button has been pressed.

When the 8051 receives the signal that a button has been pressed, it quickly sends pulses of power to each of the row wires and then analyzes the input from the column wires in order to determine exactly which button was pressed. The functions used for this portion of the project are described in detail below:

`char getKeypadPersistant(void)` - Once this function is called, a while loop in the function is started and does not stop until the user presses a key. This is useful for when the program needs to wait for input.

`char getKeypadPersistantNav(void)` - This is very similar to the previous function in that it will not stop until the user presses a key, however, for this function it will only return acceptable navigation button presses. In this case, the navigation buttons are 2, 4, 6, 7, #, and *. If any other character is pressed, the function will ignore it and continue waiting for one of the 6 accepted buttons.

`void SW2_ISR (void) __interrupt 0` - This function is an interrupt function that is called every time the INT0 port goes low, which happens every time a button is pressed. Inside this function is where each row on the keypad is pulsed to find out which row and column combination the button press originated from.

Part 2

Initially setting up the board involved prompting the user to place each ship in its specific spot. This was accomplished by operating the keypad like arrow keys with numbers 2,4,6,and 8 operating as the up,left,right, and down arrows. The # key rotated the ship and the * key was used to confirm the placement of the ship. This involved a series of if statements to ensure the proper input was given and reprinting the board each time a ship was moved. In addition many precautions were taken so that ships could not leave the board boundaries, overlap eachother, or rotate incorrectly. The functions used for the intial setup are described below.

`void displayBoardOutline()` - This function prints out the outline of the board, the initial instructions, and the numbering system on the outside of the boards.

`void printBoard()` - This function prints out the current setup of the user’s board. It will display all ship placements and any strikes the opponent has taken.

`void printOpBoard()` - This function prints out the current status of the opponent's board. It will display all the strikes that the user has taken, and whether they have been hits or misses. This is the board to the right side of the screen.

`char canMove(unsigned int x_next, unsigned int y_next, unsigned int orientation_next, unsigned int size)` - This function is one of the most complex. It determines if the user (who is currently placing ships on their own board) can move the ship in the requested direction. For example, if the user pressed the '8' key, requesting to move the ship down one row, the canMove function needs to check if that new position is still on the board and/or currently taken by another ship. This function also handles rotations of ships.

`void addToBoard(unsigned int x, unsigned int y, unsigned int orientation, unsigned int size)` - After a user pressed '*' to confirm a ship's placement on their board, this function adds the information to the matrix used to store the user's ship placement information.

`void placeShip(unsigned int ship)` - This function is the main control function for the initial setup of the game. It handles all interaction with the user and calls all the previous functions directly or indirectly. It asks for user input by calling keypad function above, and then passes the input to the canMove function, and acts on the value returned from that. If able to move, it erases the old ship and places a new one in the requested position.

Part 3

The final part of the lab included dictating how the two microcontrollers would communicate with each other. This part involved creating a specific communication protocol for the project's needs. Since the program would be sending information between microcontrollers relatively frequently this protocol needed to be implemented to prevent messages from being confused or misinterpreted. The team decided on a simple protocol: each message would be 4 characters long, with different characters signifying different actions/events. This can be seen in the table below.

Table 1: UART communication messages

`R-R-R-!`	Sent to the other board after user has finished placing ships.
`X-:-Y-!`	Coordinates of a strike from the sender. The X and Y are replaced with 0-F characters signifying 0-15.
`K-I-A-!`	Sent to the other board when a strike from the opponent has hit a ship
`K-I-X-!`	Sent to the other board when a strike has killed an entire ship. “X” would be replaced with the ship identifier (2-ptboat, 3-cruiser, 4-battleship, 5-carrier)
`M-I-S-!`	Sent to the other board to tell it that the strike coordinates received evaluated to a miss.

The functions for this portion of the project are described in more detail below.

`void sendMessage(char *str)` - This function is how the two microprocessors communicate during the game. A char array (string) is passed into the function and then the function loops through each character, sending them one at a time to the other UART, with a pause in between each character.

`int checkIfAlreadyShotThere(char row, char col)` - After a user types in a coordinate to strike at, the coordinate is checked by this function to find out if the user has already taken a shot at that location. It returns a true or false.

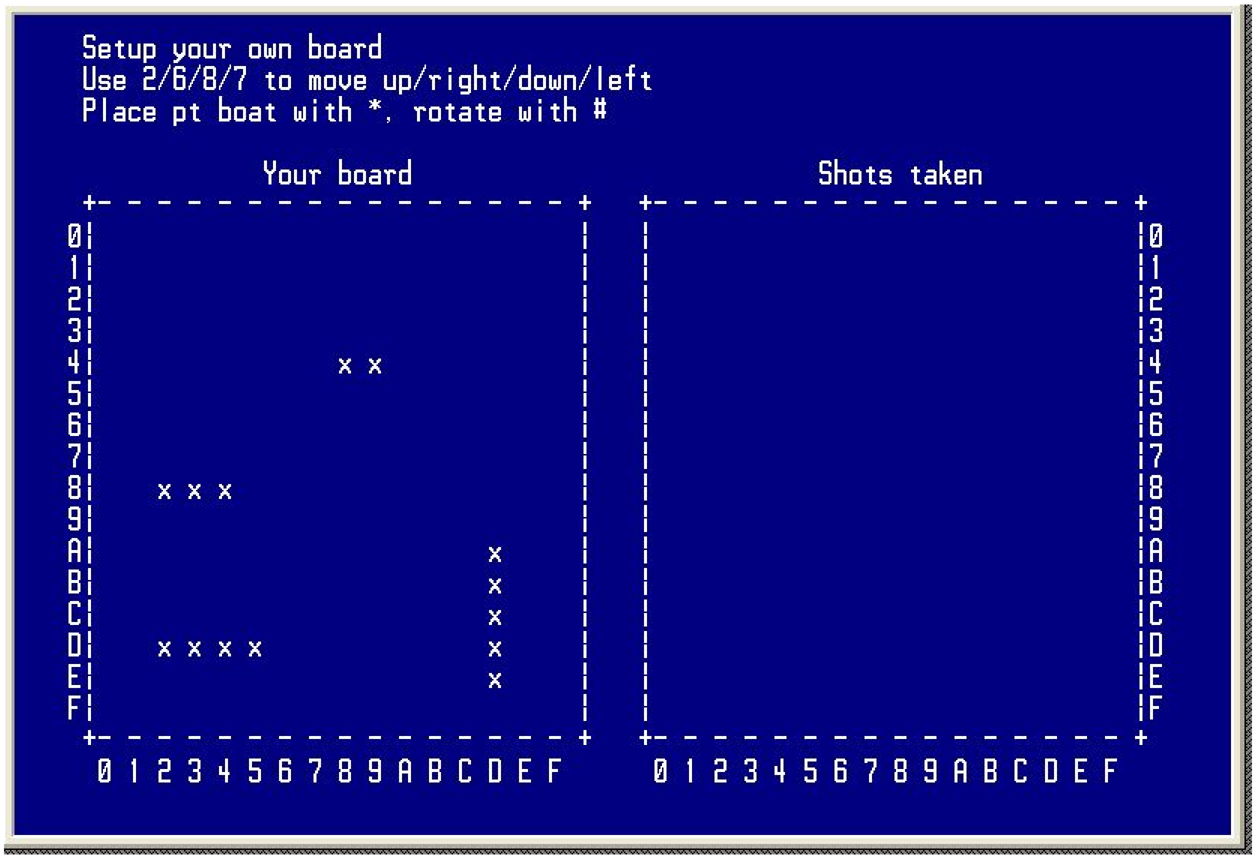
`int checkIfHit(char row, char col)` - After a strike location comes in from the other microprocessor, it is run through this function to check if it has hit or missed a ship. If it hits a ship, it reduces the stamina of that ship by one.

`void addHitOrMiss(char row, char col, int hitMiss)` - This function adds the strike (whether it be a hit or a miss) to the strike board on the right, and then calls the printOpBoard function so that output can be seen on the ANSI terminal.

Results and Analysis

All parts of the project operated as intended. The following Figures illustrate the project's demonstration.

Figure 2: Setup of the board

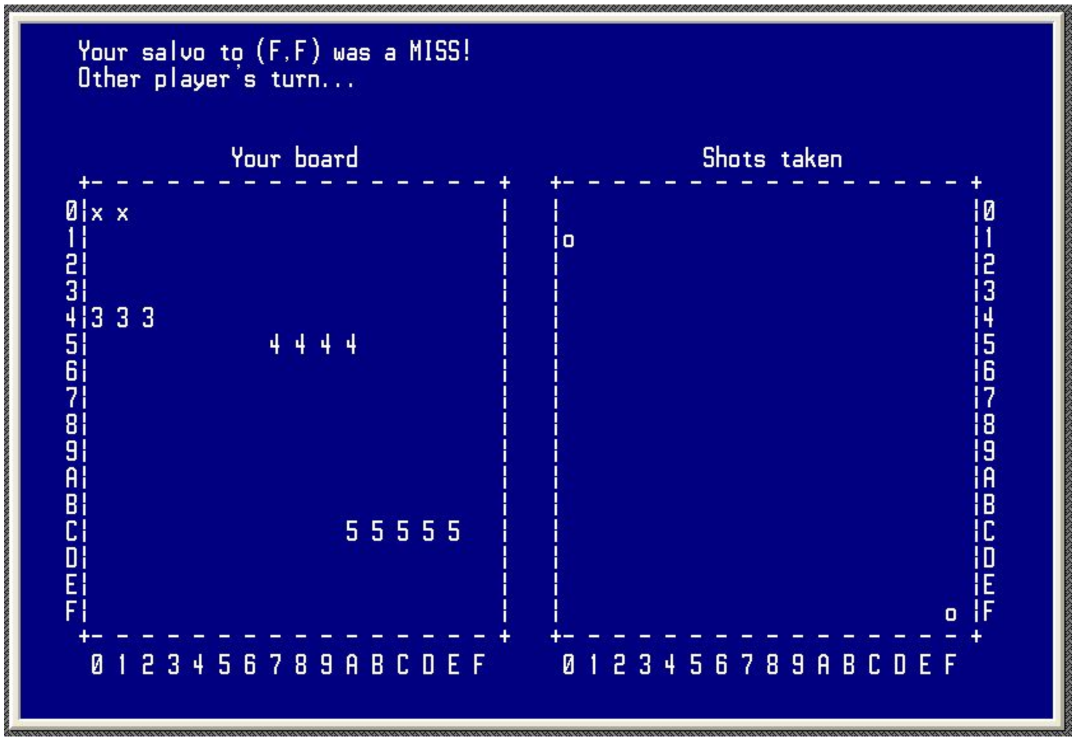


```
     Setup your own board
     Use 2/6/8/7 to move up/right/down/left
     Place pt boat with *, rotate with #

                  Your board                           Shots taken
    +- - - - - - - - - - - - - - - - -+   +- - - - - - - - - - - - - - - - -+
   0|                                 |   |                                 |0
   1|                                 |   |                                 |1
   2|                                 |   |                                 |2
   3|                                 |   |                                 |3
   4|                 x x             |   |                                 |4
   5|                                 |   |                                 |5
   6|                                 |   |                                 |6
   7|                                 |   |                                 |7
   8|     x x x                       |   |                                 |8
   9|                                 |   |                                 |9
   A|                           x     |   |                                 |A
   B|                           x     |   |                                 |B
   C|                           x     |   |                                 |C
   D|     x x x x               x     |   |                                 |D
   E|                           x     |   |                                 |E
   F|                                 |   |                                 |F
    +- - - - - - - - - - - - - - - - -+   +- - - - - - - - - - - - - - - - -+
      0 1 2 3 4 5 6 7 8 9 A B C D E F       0 1 2 3 4 5 6 7 8 9 A B C D E F
```

In figure 2 above you can see that the user was prompted to place the PT Boat (the two x's towards the top). If the user were to press #, the boat would be rotated vertically, and if the user were to press *, the Pt boat would be placed in that spot.

In figure 3 below, it can be seen that the user sent a salvo to (F,F) and it resulted in a missed shot. The terminal then lets the player know that the other player is current planning their attack, and to wait for them to finish. It can also be seen on the right side of the screen that the current user has previously taken a shot (and missed) at (0,1).

Figure 3: Missed shot



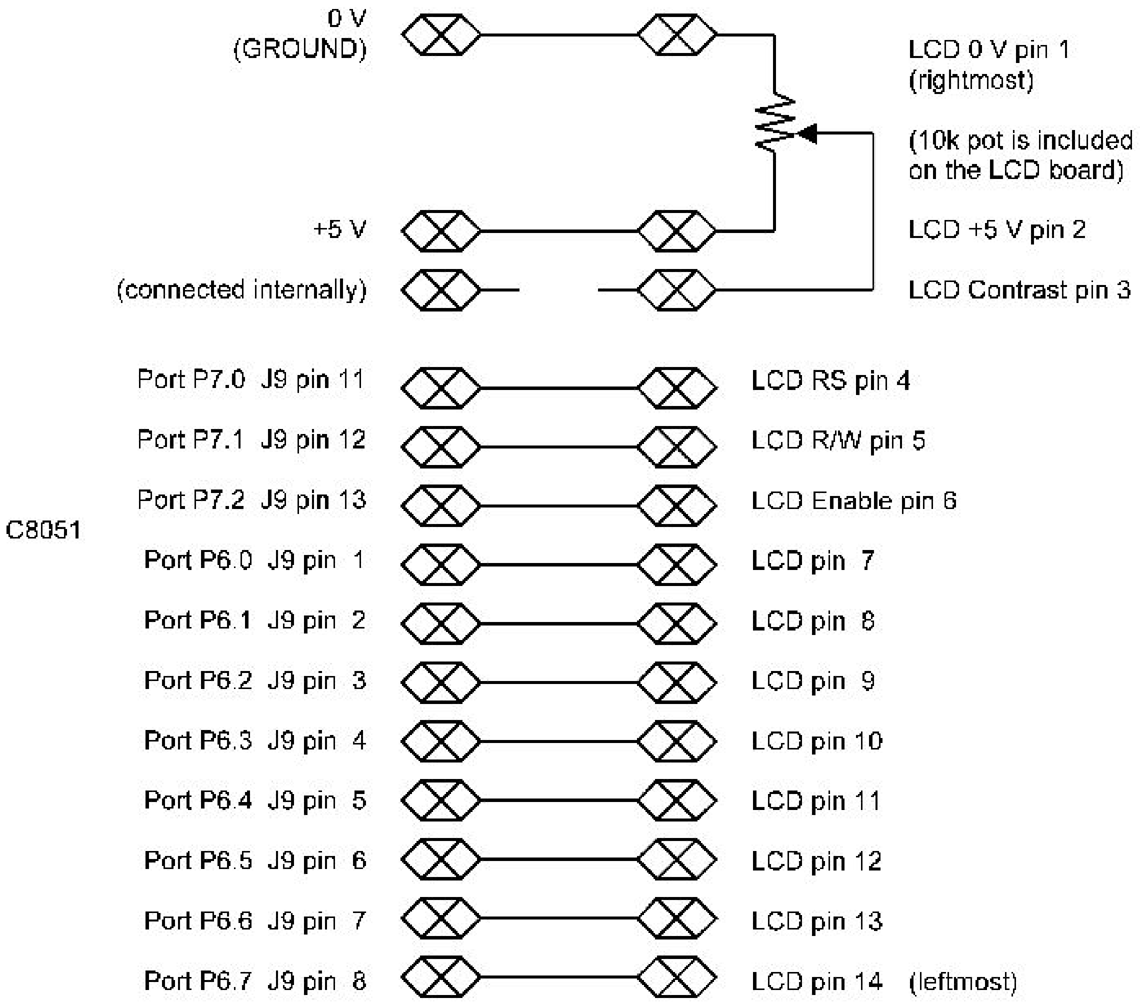
```
  Your salvo to (F,F) was a MISS!
  Other player's turn...

              Your board                        Shots taken
 +- - - - - - - - - - - - - - - - +  +- - - - - - - - - - - - - - - - +
0|x x                             |  |                                |0
1|                                |  |o                               |1
2|                                |  |                                |2
3|                                |  |                                |3
4|3 3 3                           |  |                                |4
5|              4 4 4 4           |  |                                |5
6|                                |  |                                |6
7|                                |  |                                |7
8|                                |  |                                |8
9|                                |  |                                |9
A|                                |  |                                |A
B|                                |  |                                |B
C|                    5 5 5 5 5   |  |                                |C
D|                                |  |                                |D
E|                                |  |                                |E
F|                                |  |                              o |F
 +- - - - - - - - - - - - - - - - +  +- - - - - - - - - - - - - - - - +
  0 1 2 3 4 5 6 7 8 9 A B C D E F      0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Conclusion

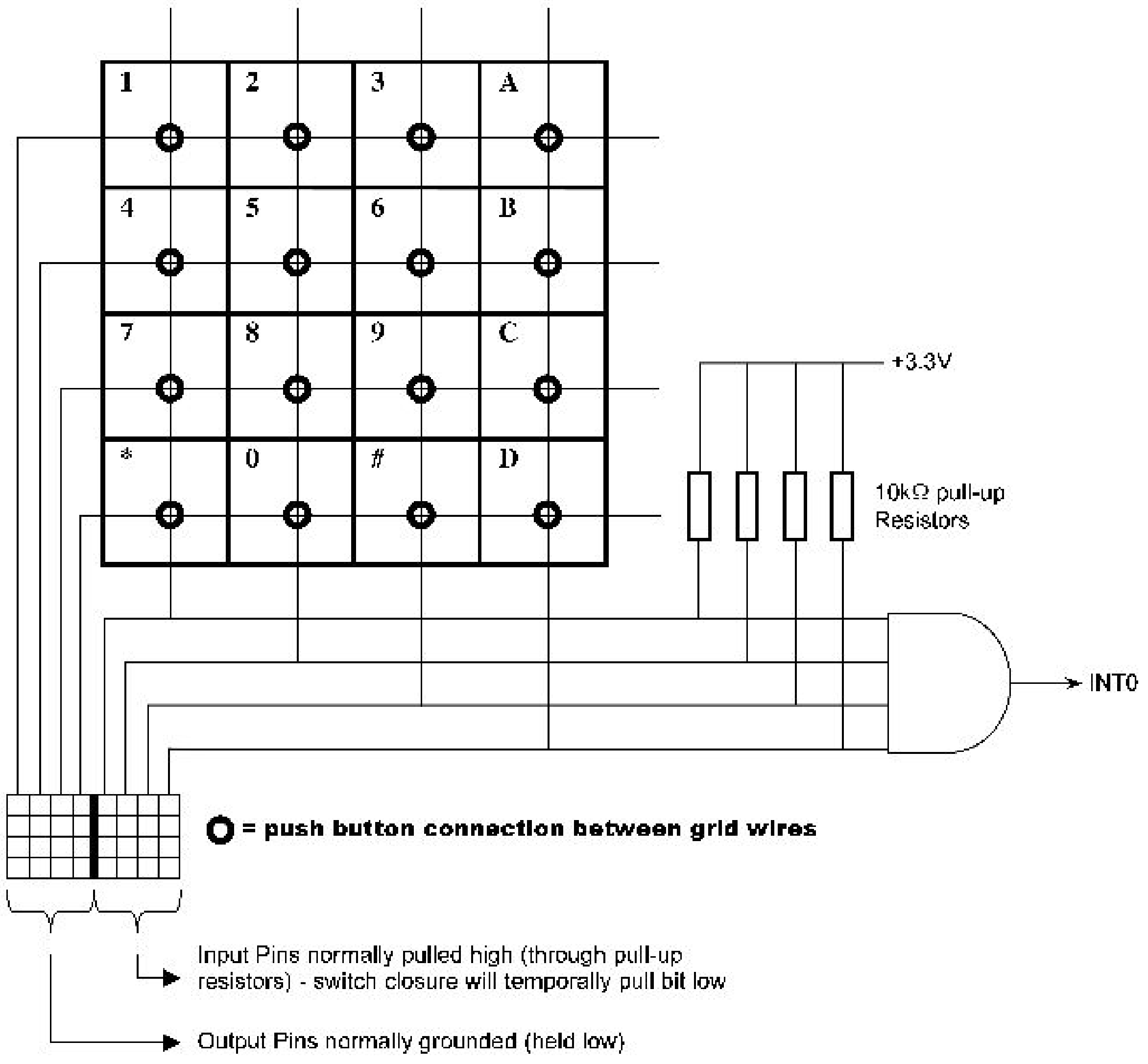
The team successfully accomplished all parts of the project by making several smaller programs that accomplished all the subgoals. Dividing the project into separate goals made the exercises significantly more approachable due to implementing the divide and conquer mindset. Each goal was simple enough that each group member could complete a goal by the time they came to class, and spend lab time debugging code and building hardware.

Given more time the team would have liked to explore different ways of accomplishing these tasks, For example, rather than having the microcontrollers communicate over UART, they could have communicated over an internet server or possibly through an RF receiver/transmitter. Furthermore, a different display method could have been used instead of the ProComm terminal. This could have opened doors to making the game more visually pleasing instead of its current text-based display. Also they team could have explored different modes of playing sounds to interact with the game. Using a buzzer or a speaker to play the noise of a sinking ship or a rocket firing could add to the authenticity of the game.

Appendix A - Schematic for LCD Screen



Appendix B - Schematic for Keypad


1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
+3.3V
10kΩ pull-up Resistors
INT0
= push button connection between grid wires
Input Pins normally pulled high (through pull-up resistors) - switch closure will temporally pull bit low
Output Pins normally grounded (held low)