EE 478 Capstone Final Report
RFID Interaction Suite

Alyanna Castillo
Patrick Ma
Ryan McDaniels

Contents

List of Tables

List of Figures

1 ABSTRACT

2 INTRODUCTION

These are some example of how to cite and use bullet points A reference to Table 1 and one to the Design Specification, Section 3.

- Overall summary description of the module - 2-3 paragraphs maximum (explanation of use cases goes here)
  - Specification of the public interface to the module
    - Inputs
    - Outputs
    - Side effects
  - Psuedo English description of algorithms, functions, or procedures
  - Timing constraints
  - Error handling

3 DESIGN SPECIFICATION

3.0.1 Design Overview

This system is a RFID-based gaming system. A user is able to play alone against a computer, or can play with other users owning a system using a multiplayer connection feature. Additionally, users can create their own customized cards using a third built-in function of the device.

3.1 Design Requirements

3.1.1 Environmental Requirements

The device must operate in an indoor/outdoor environment with relative humidity consistent with desert and tropical environments. It must be durable enough to withstand various types of users, especially small children. The unit is to be portable and battery operated.

3.1.2 System Input and Output Requirements

The system must be capable of accepting several different types of signals and inputs:

- Standard frequencies used for close-range or Near-Field Communication Radio Frequency Identification (RFID) devices
- Standard frequencies used for commercial wireless communication standards such as wifi networks
- User input from a keypad to navigate menus, enter commands, and provide other alphanumeric information

- When in multiplayer mode, communication and commands from other players must be accepted and responded to.

The system must be capable of providing the following outputs:

- Display information about the current game state through an LCD screen
- Provide status information of individual cards placed on the game board
- Send commands to other systems when in multiplayer mode
- Send wireless signals to program RFID-enabled cards

3.1.3 User Interface

The system must also have the following buttons, switches or interface devices:

- 16-button keypad with alphanumeric character labels
- Reset button that when pressed causes a soft reset of the system
- Power button that turns the system on and off
- An LCD screen viewable from several angles. The screen must be viewable indoors and in overcast conditions

An example User interface is shown in Figure 1.

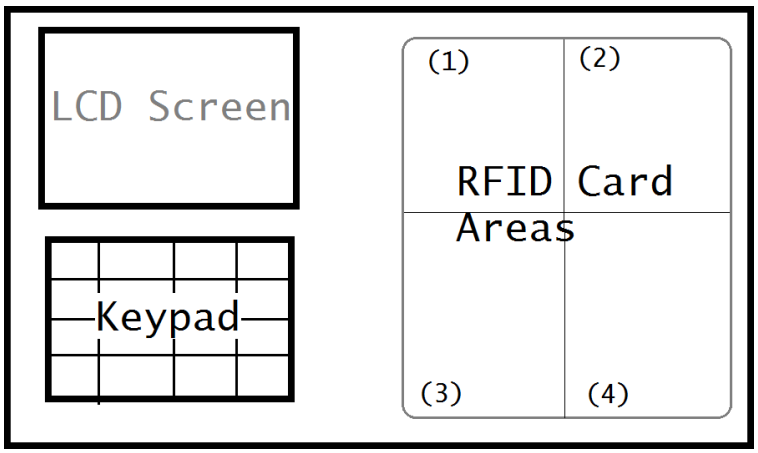


Figure 1: An example front panel layout for the system

3.1.4 Functional Requirement

The system must support the following modes of operation:

Single Player: Allows the user to select from, and play, single player games. Upon entering Single Player mode, the system prompts the user to select a game to play from those stored in memory. Selecting a valid game loads the game from memory and begins game execution.

Multiplayer: Allows the user to select and play multiplayer games. Upon entering Multiplayer mode, the system activates wireless communication and attempts to connect to other users in Multiplayer mode. Once a connection is established, the users select a multiplayer game on both systems and gameplay begins.

Build Card: Allows the user to create custom cards for a game. To create a custom card, the user will input the data via the keypad. The data stored will vary based on game requirements. All cards must contain unique card serial numbers or identifing ID numbers, and a code that specifies which games the card is valid for.

In all three modes, the user will be instructed by the programmed game to place a card in the RFID reading area to interact with the game and control events.

3.1.5 Operating Requirements

The system must operate in a standard commercial or household environment. The system must be portable, wearable on the user's arm, and operate on an internal power supply.

3.1.6 Reliablility and Safety Requirements

The system must be compliance with communication and electromagnetic radiation standards including those from the U.S. Federal Communication Committee, and applicable state and federal child safety laws.

3.2 Identified Use Cases

3.3 Detailed Specifications

3.4 Functional Decomposition

Figure 5 in Appendix D gives a hierarchy of the system software functions. Each function in the functional diagram has a corresponding software task.

The functions are explained below:

Play Game Loads and begins execution of the requested game.

System Scheduler The system scheduler monitors the state of the system and calls system functions as necessary. On system power on or power on reset, the scheduler intitializes system variables and calls the setup() function of each task.

Read/Write Cards Formats and sends commands to the RFID reader. The task also processes the raw data from the RFID reader and stores the card UID and card memory data for later retrieval by other tasks.

Access Database Reads and writes data to the local SRAM. The Access Database task determines where each byte of data is written and writes or requests the data from the memory. The Access Task also maintains a reference table of game data corresponding to the moves, types, etc. corresponding to data encoded on the RFID cards.

User Interaction User interaction involves three functions: Keypad, Display, and Card Reaction.

Keypad The keypad driver determines which key the user has pressed, if any.

Display The display driver interfaces with the LCD over an Serial Peripheral Interface (SPI) to draw game menus and screens. The display driver also formats text strings and numbers from the game for display on the screen.

Card Reactions The Card Reaction task determines when a card had been placed onto the board, tells the system to read and process the card, and sets the card status LEDs.

LED Driver This task changes the output of the LEDs based upon the read status of the cards.

Communication Communication includes three tasks: external serial communication via RS-232, external wireless communication via 2.4GHz wifi, and internal InterPIC communicaton via I^2C

Serial Driver The serial driver sets up the serial RS-232 connection port. The task also processes incoming serial data, saving the result to system memory and alerting the scheduler to run subsequent tasks.

Wireless Driver The wireless driver configures the XBee module, sets up the person-to-person communication link and formats incoming and outgoing data between the game and XBee module.

InterPIC Driver The InterPIC driver configures the I^2C port and sends data over the I^2C connection. The driver wraps and unwraps data sent over the connection in the I^2C protocol. Each once unwrapped, the command is interpretted, data is stored for system use, and the scheduler is alerted to call other tasks.

Build Card The Build Card function prompts the user to input customizing options then encodes the responses for storage on an RFID card. Once data is properly encoded, the task instructs the InterPIC Driver to write the data to the RFID card through the RFID reader.

4 HARDWARE IMPLEMENTATION

4.1 Top Level Design

4.2 Low Level Design

5 SOFTWARE IMPLEMENTATION

5.1 Top Level Design

5.2 Low Level Design

6 PRESENTATION, DISCUSSION, AND ANALYSIS OF THE RESULTS

6.1 Results

6.2 Discussion of Results

6.3 Analysis of Any Errors

The biggest problem with the final version of this project that was present at demo time was the fact that the multiplayer features were not implemented. There was test code that correctly configured the Xbee modules for multiplayer, but they were not completely implemented with the game. The reason for this was because when the I^2C system was implemented, the entire game had to be modified to run within the scheduler, when it was just a single function before. I^2C communication is controlled by the system's interrupt handler, and certain flags are set depending on whether data is being sent or received. Those flags have to be processed, and a game that is running in a function and taking control of the entire system would not allow for those flags to be processed. The time it took to port the game over to running completely within the scheduler made it impossible to get the multiplayer functions completely implemented and working in time.

Another problem at the time of the demo was that card reading was not 100% functional. The system had four distinct sets of data that could be successfully written to a card using the "Build Cards" option from the system's main menu, but the data was not being displayed properly in the singleplayer game. Data coming from the card over the I^2C connection was confirmed to be correct, but the game was not interpreting and displaying the information correctly to the user. This was also a matter of running out of time. For the same reasons as above (converting the game to be run within the flag-based scheduler) the RFID reading still had some kinks to work out at the time of the demo. Those problems were that:

- Monster type was being read incorrectly
- Monster name was being displayed incorrectly
- Complete attack list was not implemented

For demo purposes, only the first attack would be read and it would be copied to all three attack slots. Only the "FAIL" attack ID was programmed to be read, and if the ID did not match the "FAIL" attack's ID, then it was interpreted as a "SCRATCH" attack. This function worked

correctly, but the rest of the attack IDs were not implemented. The monster's level was correctly read as well.

Finally, there was an error when finishing a singleplayer game. When the game was complete, the main menu was not being properly displayed. Once again, the problem was time, and the main menu's controls were still functional. Another menu could be loaded by navigating the invisible menu and choosing an option, letting that screen load, and then pressing the "B" key to return back to the main menu and redraw it.

6.4 Analysis of Implementation Issues and Workarounds

7 TEST PLAN

7.1 Test Specification

7.2 Test Cases

8 SUMMARY AND CONCLUSION

8.1 Final Summary

8.2 Project Conclusions

A BREAKDOWN OF LAB PERSON-HOURS (ESTIMATED)

Person	Design Hrs	Code Hrs	Test/Debug Hrs	Documentation Hrs
Patrick	x	x	x	x
Alyanna	x	x	x	x
Ryan	x	x	x	x

By initializing/signing above, I attest that I did in fact work the estimated number of hours stated. I also attest, under penalty of shame, that the work produced during the lab and contained herein is actually my own (as far as I know to be true). If special considerations or dispensations are due others or myself, I have indicated them below.

B BILL OF MATERIALS

Table 1 lists the bill of materials for the construction of one system.

Bill of Materials

Item	Unit Cost	Quantity	Total Cost
TI HI-Plus RFID Tags	0.91	20	18.20
DLP-RFID2 RFID Reader/Writer	50	3	150
Xbee S1 Wireless Chips	30	2	60
PLA Makerbot Filament	48	1	48
GAL22V10D	3.5	4	14
PICKit 3	45	2	90
CY7C128A SRAM	4	2	8
3.3 Volt Linear Regulator	3.22	2	6.44
Lever Switch, micro SPDT, momentary	2.5	6	15
16-key Numeric Keypad	7.5	2	15
128x169 Color LCD	17	2	34
PIC18F46K22 Microcontrollers	7.7	4	30.8
RGB LED, Common Cathode	4	8	12
(EXTRA)			
(EXTRA)			
(EXTRA)			
Total Cost			

Table 1

C HARDWARE DIAGRAMS

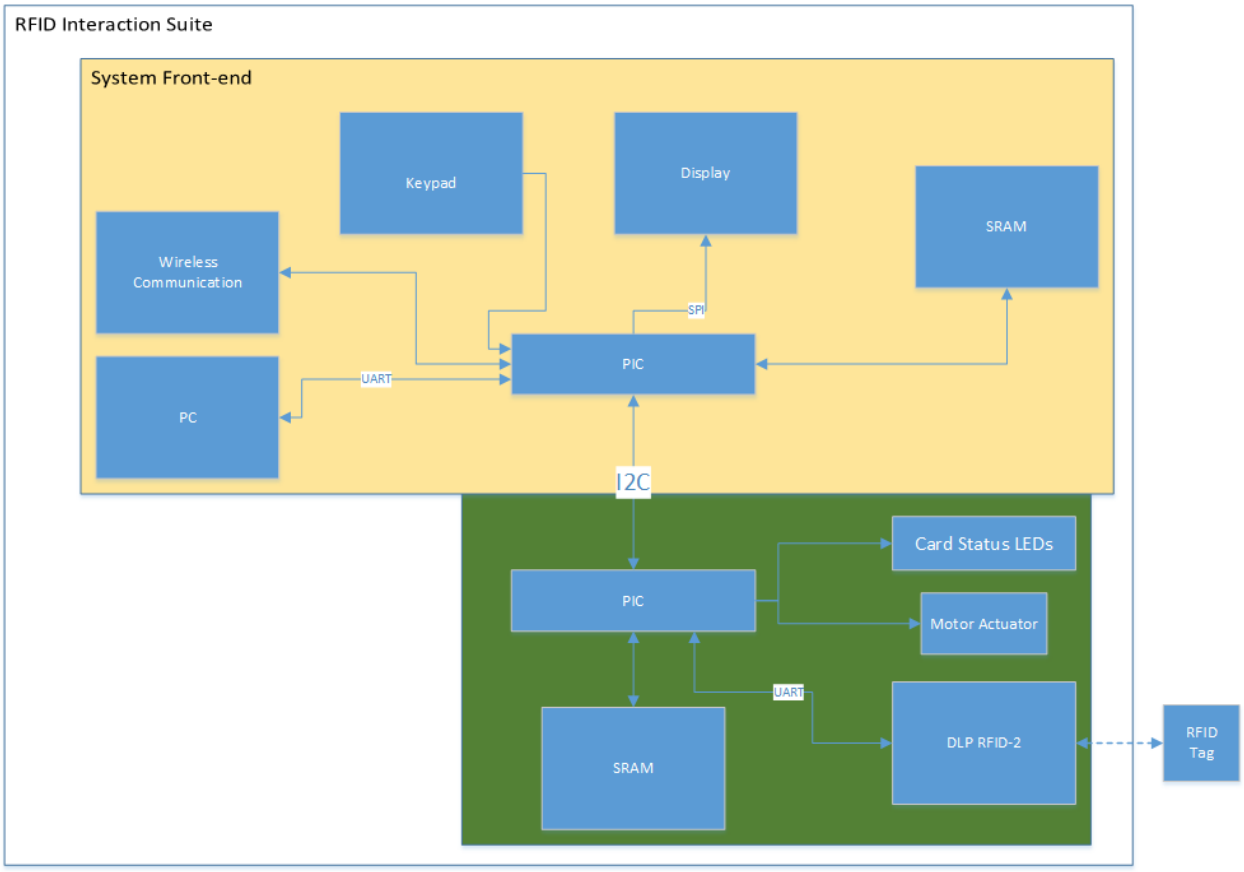


Figure 2: High level block diagram of the system hardware components

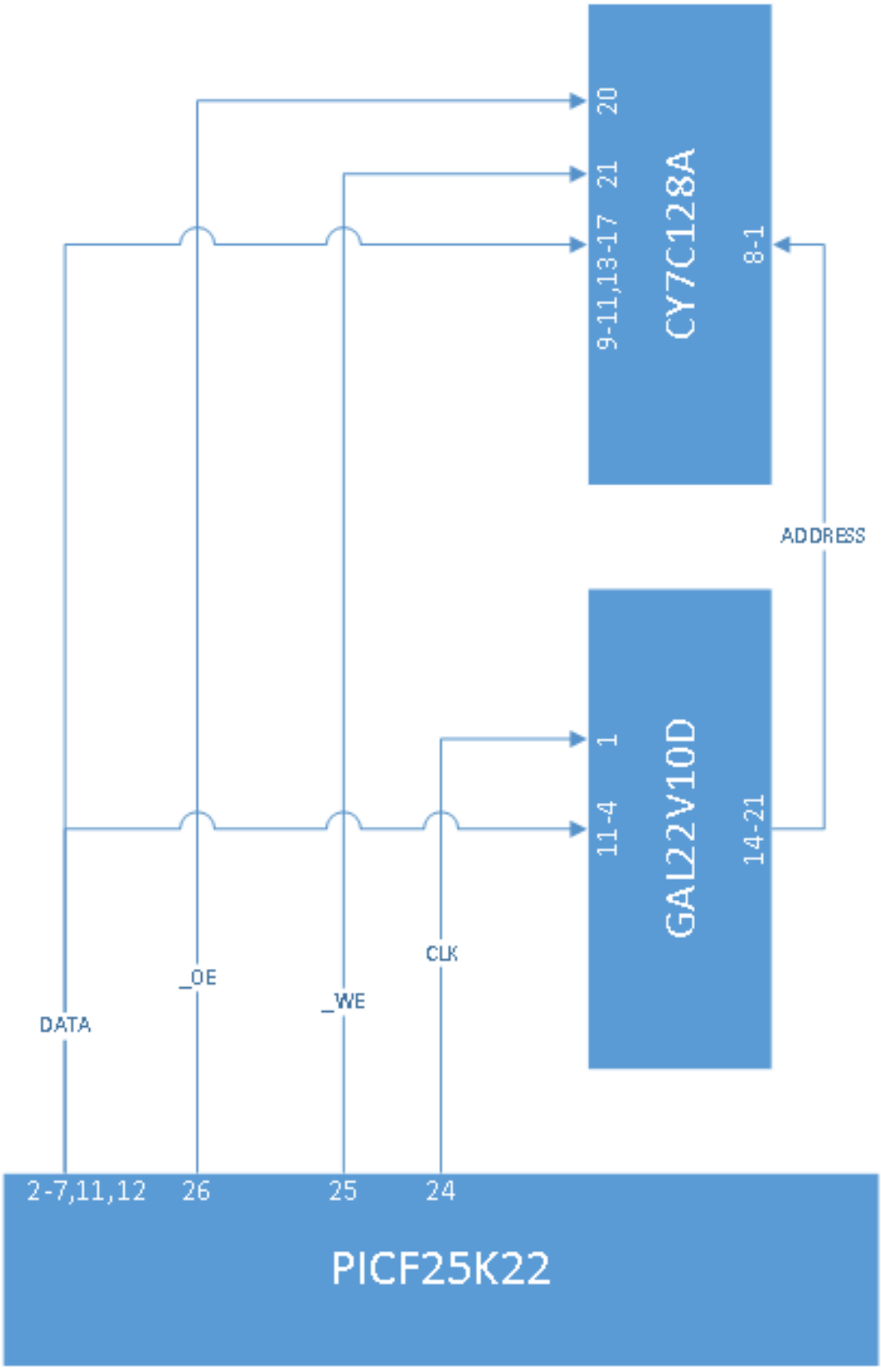


Figure 3: Block diagram of the SRAM hardware system

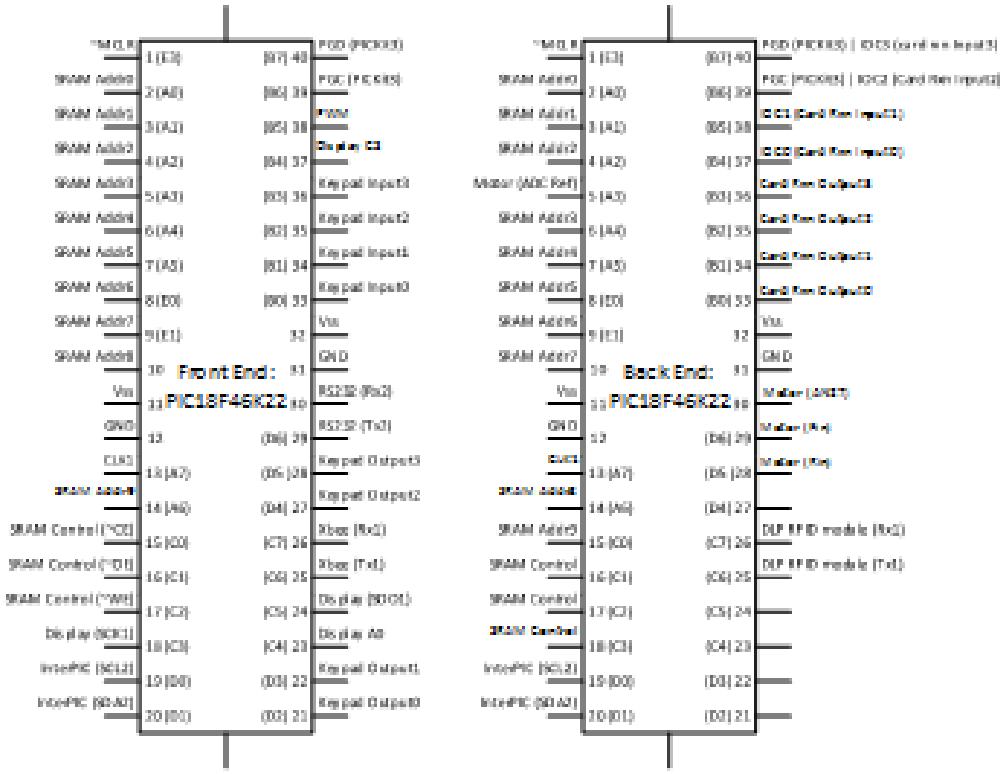


Figure 4: Pinouts to the front- and back-end microcontrollers

D FUNCTIONAL DECOMPOSITION DIAGRAM

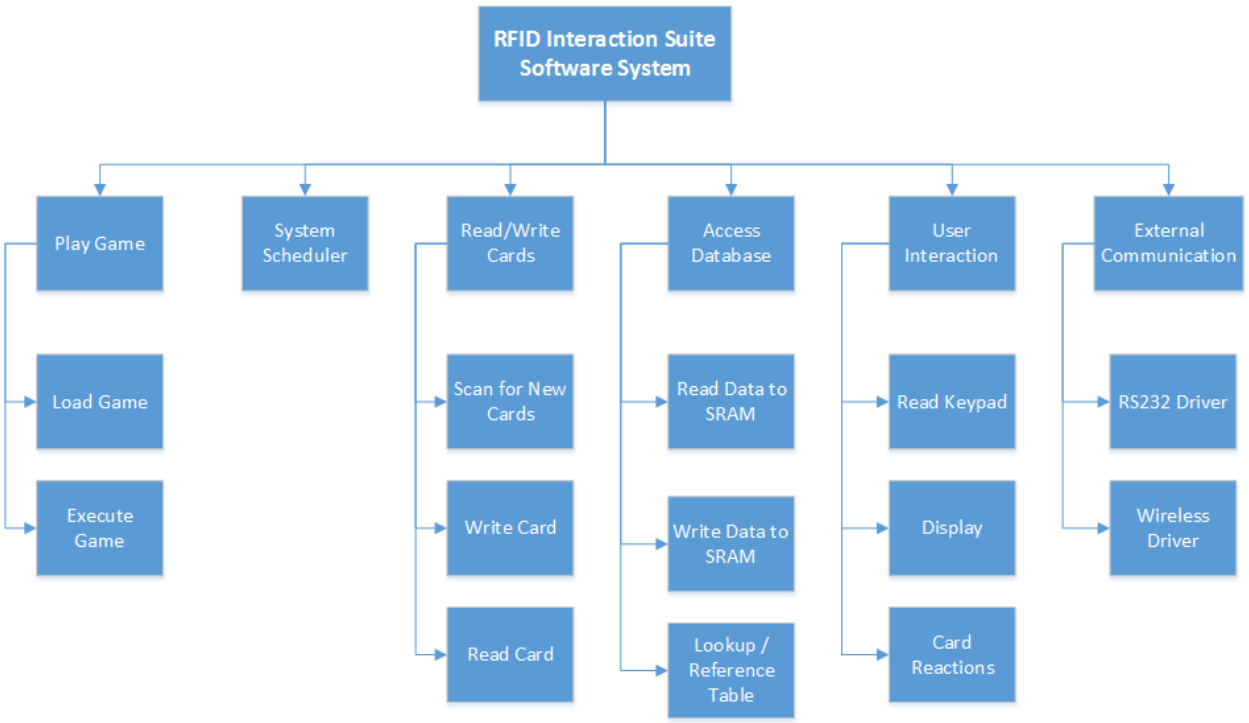
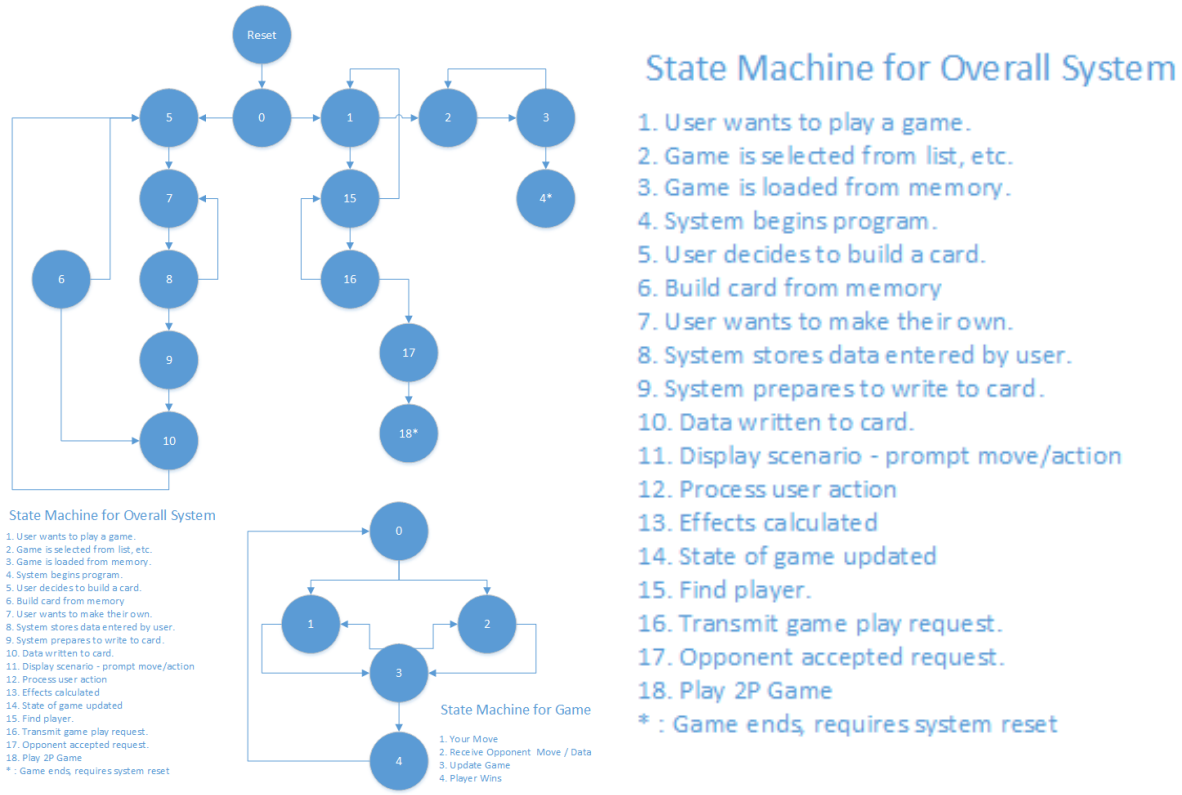


Figure 5: Software functional decomposition showing the major functional divisions and tasks

E STATE DIAGRAMS

E.1 System State Diagram



(a) (b)

Figure 6: State diagram of the primary operating system. Figure 6a shows the states while Figure 6b provides a legend

E.2 General Gameplay State Diagram

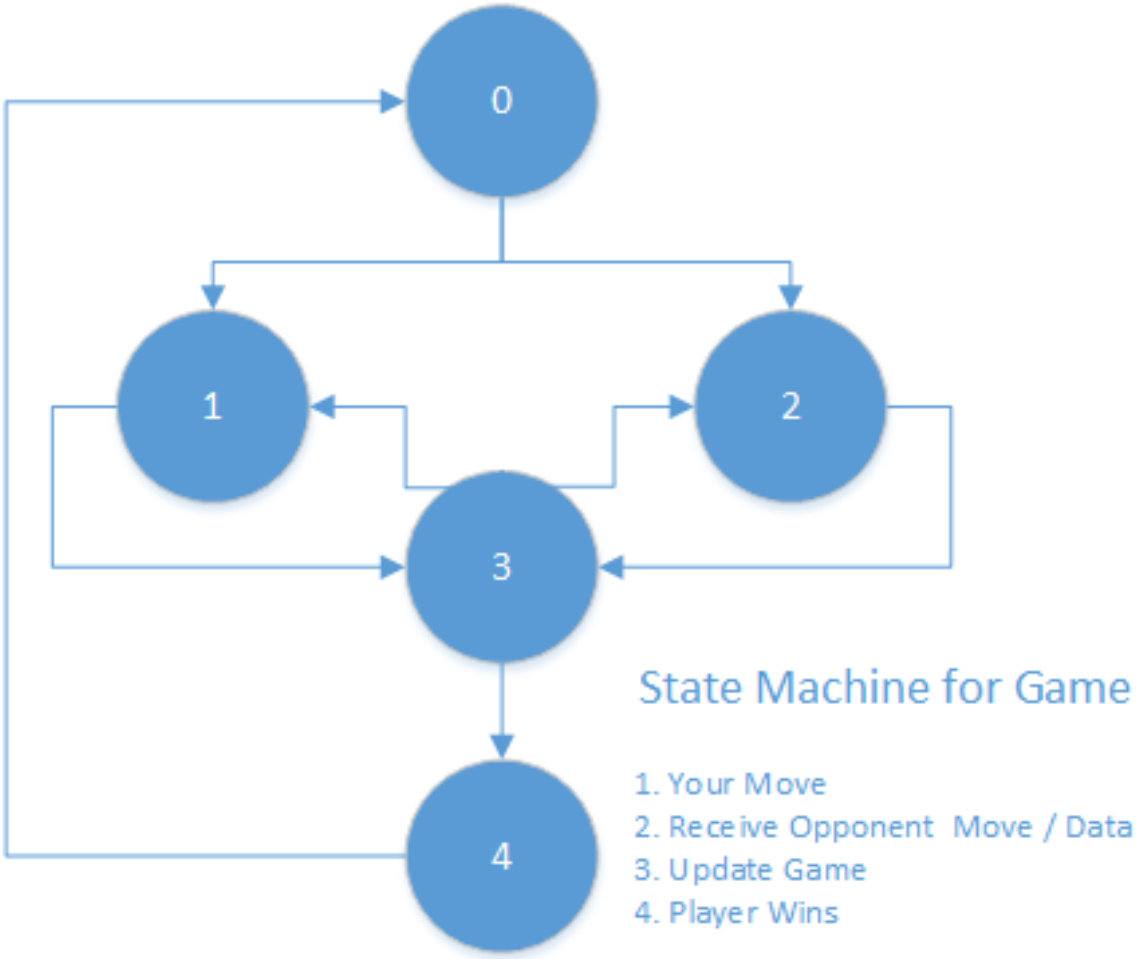


Figure 7: State diagram of basic turn-based game

F CONTROL FLOW DIAGRAMS

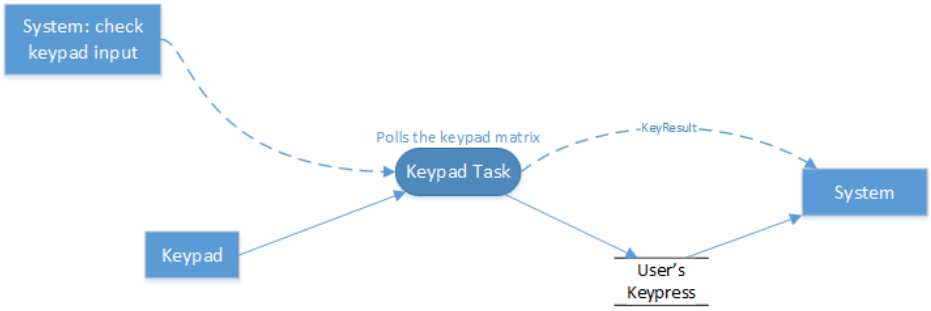


Figure 8: Keypad response

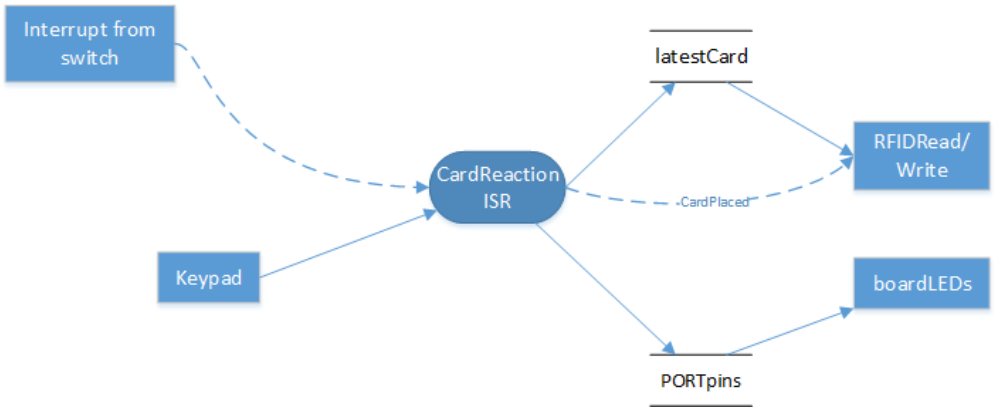


Figure 9: Card Reaction LEDs

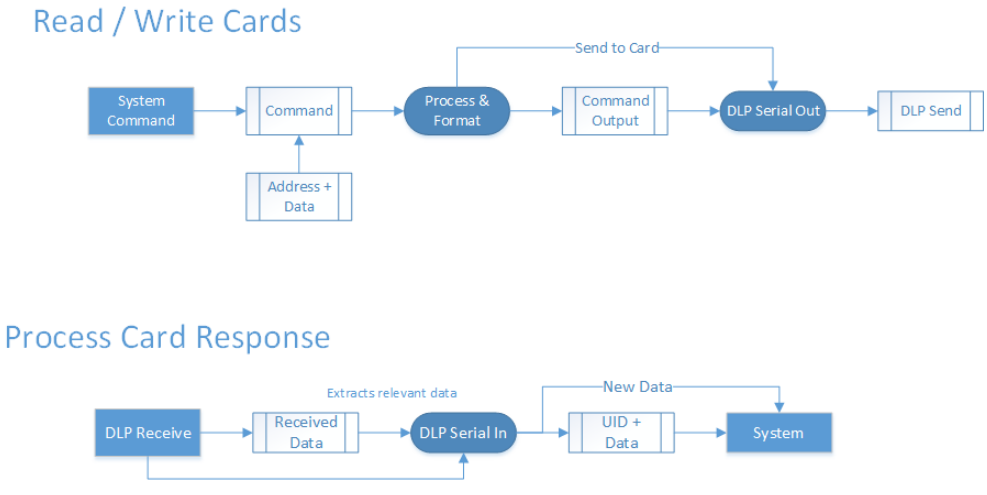
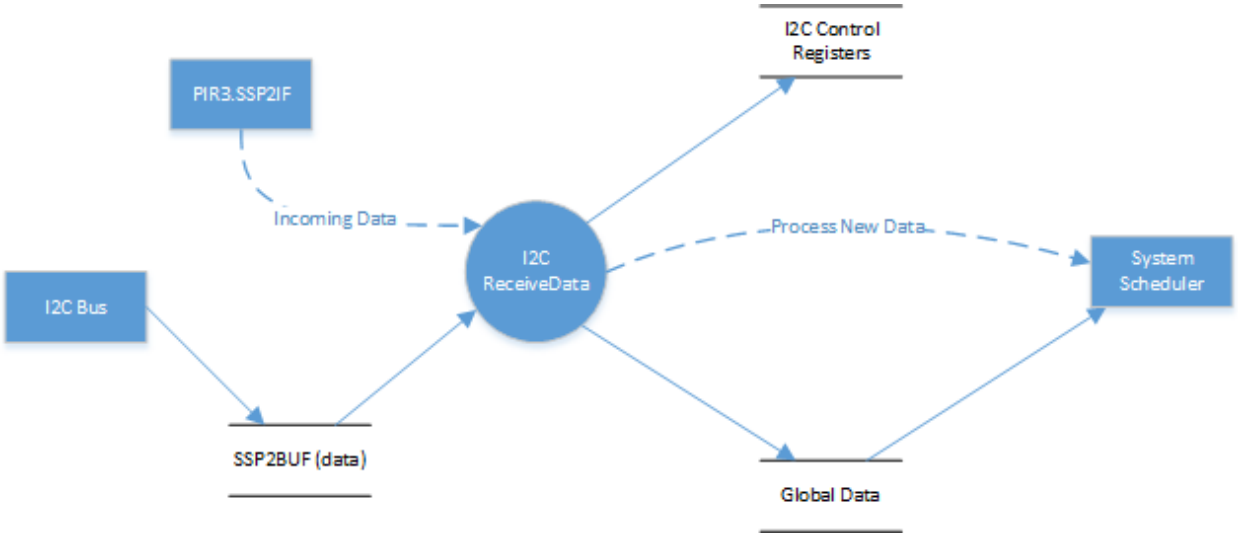
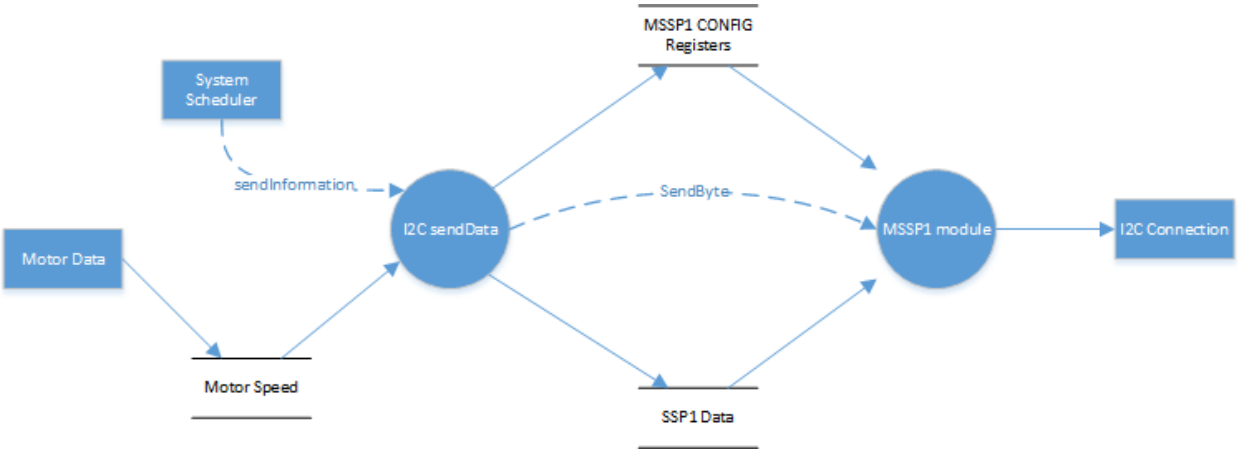


Figure 10: RFID tag reader subsystem



(a)



(b)

Figure 11: I2C communication between microcontrollers. Figure 11a shows the flow for received data and Figure 11b shows the flow for transmitted data.

G PROJECT SCHEDULE

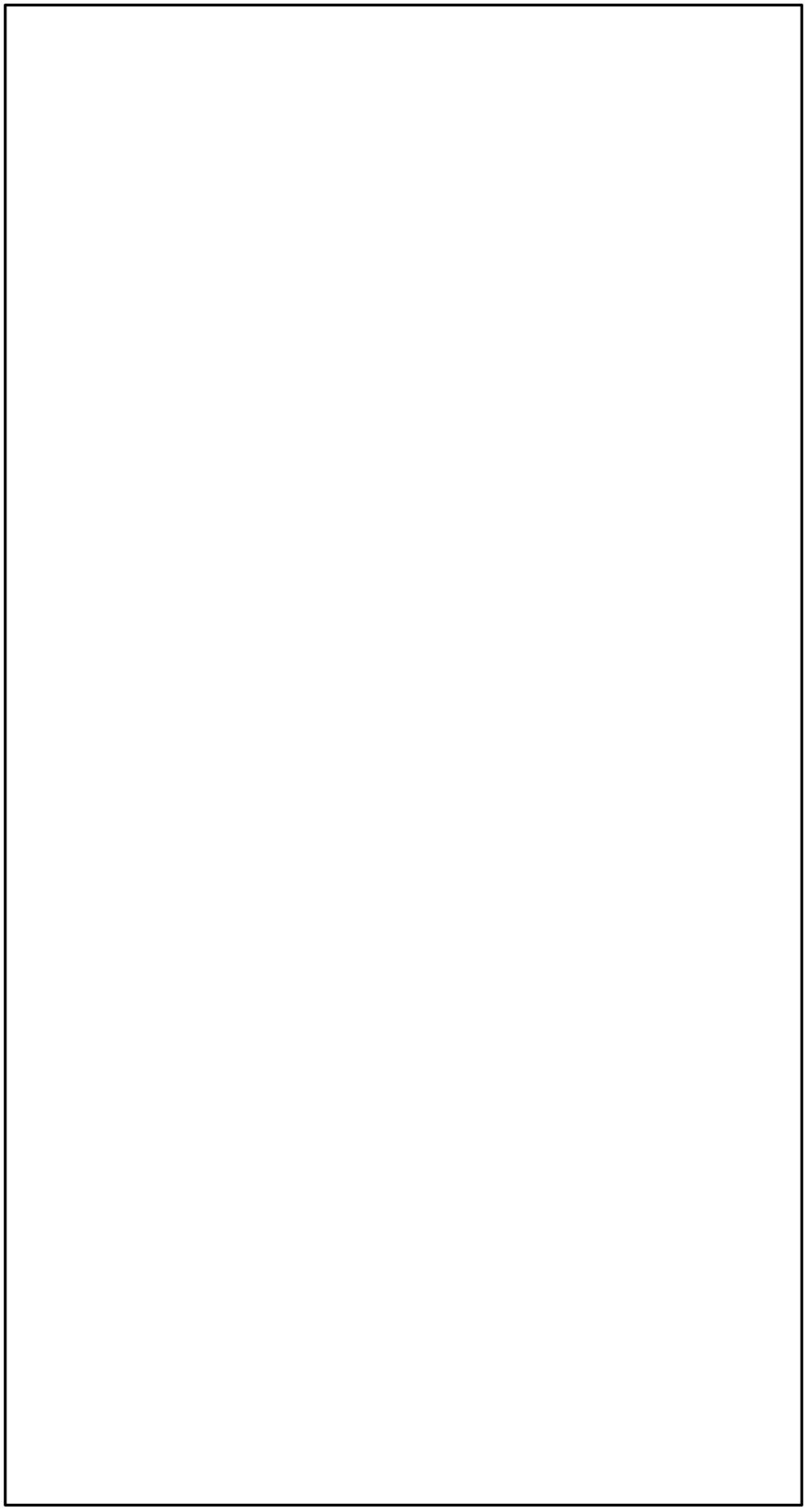


Figure 12: Project Gantt Chart

H SOURCE CODE

Source code for this project is provided below.

H.1 System Scheduler

The system-wide global definitions and variables are also given here for convenience.

../FrontEnd.X/globals.h

```
/*
 * File:   globals.h
 * Author: Patrick
 *
 * Created on May 27, 2014, 4:44 PM
 */

#ifndef GLOBALS_H
#define GLOBALS_H

#ifdef __cplusplus
extern "C" {
#endif

    // microchip libraries
#include <p18f46k22.h>
#include <usart.h>
#include <spi.h>
#include <delays.h>
#include <pwm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "usart.h"
#include "delays.h"
#include <adc.h>
#include <timers.h>
#include "i2c.h"
#include "LED.h"

    // our headers
#include "rs232.h"
#include "keypadDriver.h"
#include "LCD.h"
#include "startup.h"
#include "interrupts.h"
#include "rfidReader.h"
#include "motorDriver.h"
#include "game.h"
#include "i2cComm.h"
```

```
41 #include "xbee.h"
42
43 #define FRONT_NOT_BACK 1 // is this the Front end or backend (should move to
       globals)
44
45 #define CARDSLOT_1 0b00010000
46 #define CARDSLOT_2 0b00100000
47 #define CARDSLOT_3 0b01000000
48 #define CARDSLOT_4 0b10000000
49 #define CARDBLOCKSIZE 4 // in bytes
50
51
52     enum _myBool {
53         FALSE = 0, TRUE = 1
54     };
55     typedef enum _myBool Boolean;
56
57     typedef struct globaldata {
58         short displayPage;
59         Boolean keyFlag;
60         Boolean displayedKey;
61         Boolean goBack;
62         Boolean xbeeFlag;
63         Boolean firstTime;
64         int keyPress;
65         int keyStatus;
66         int cursorPos;
67         int mainMenuSpots[3]; // Find better way to do this
68         int status;
69         int mode;
70   int cardSelect[4];
71   int selectMove[4][3];
72         int game;
73         int newDisplay;
74         int newGame;
75         int newKeyboard;
76         int doneKeyboard;
77
78         /* card block data read in */
79         unsigned char dataBlockNum;
80         unsigned char dataSlotNum;
81         unsigned char dataBlock[16]; // 32 bits of data
82
83         Boolean getInventory;
84
85         Boolean sendI2C; // i2c command prepped for transmission
86         Boolean gotI2C; // i2c command was received
87
88         Boolean updateLEDFlag;
```

```
89          char lastCards;
90          char readCard;  // flag indicating which card slot needs to be read
91          Boolean runGetUpdatedCards;
92      } GlobalState;
93      extern GlobalState globalData;
94
95  #ifdef  __cplusplus
96  }
97  #endif
98
99  #endif  /* GLOBALS_H */
```

../FrontEnd.X/interrupts.h

```
1   /*
2    * File:   interrupts.h
3    * Author: castia
4    *
5    * Created on April 24, 2014, 10:16 PM
6    */
7
8   #ifndef INTERRUPTS_H
9   #define INTERRUPTS_H
10
11  #ifdef  __cplusplus
12  extern "C" {
13  #endif
14
15  void rcISR(void);
16
17
18
19  #ifdef  __cplusplus
20  }
21  #endif
22
23  #endif  /* INTERRUPTS_H */
```

../FrontEnd.X/SchedMain.c

```
1   /*
2    * File:   SchedMain.c
3    * Author: Patrick
4    *
5    * Created on May 27, 2014, 4:35 PM
6    */
7
8   #include <stdio.h>
9   #include <stdlib.h>
10  #include <p18f46k22.h>
```

```
11
12 // included files for each sw function
13 #include "globals.h"
14 #include "rs232.h"
15
16
17 // Function prototypes
18 void systemSetup(GlobalState *data);
19 void setupPWM(void);
20
21 void processUID(char* uid);
22
23
24 // PIC configuration settings
25 /**************Clocking set up ********************/
26 #pragma config WDTEN = OFF     // turn off watch dog timer
27 #pragma config FOSC = ECHPIO6     // Ext. Clk, Hi Pwr
28 #pragma config PRICLKEN = OFF // disable primary clock
29
30 /**************USART set up ********************/
31 #pragma config FCMEN = OFF
32 #pragma config IESO = OFF
33 /************************************************/
34
35 /*
36  *
37  */
38 RFIDDriver readerData;
39 GlobalState globalData;
40 extern I2cDataStruct i2cData;
41
42
43 #pragma code high_vector=0x08
44
45 void interrupt_at_high_vector(void) {
46     _asm GOTO rcISR _endasm
47 }
48 #pragma code
49
50 #pragma interrupt rcISR
51
52 void rcISR(void) {
53     // The input character from UART2 (the RFID reader)
54 #if FRONT_NOT_BACK
55
56 #else
57     unsigned char input;
58     /*
59      * RFID interrupt
```

```
60      */
61
62     if (PIR1bits.RC1IF) {
63         input = RCREG1; // Read fast by directly looking at RCREG
64         //          If we are processing an Inventory command
65         if (readerData.invCom == 1 || readerData.readFlag_1 == 1 || readerData.
    writeFlag_1 == 1) {
66             if (input == 'D' && readerData.nextBlock == 0) {
67                 // Reset the inventory command flag
68                 readerData.invCom = 0;
69
70                 // Begin reading what is inside a block of square brackets
71
72             } else if (input == '[') {
73                 // Go to the beginning of the array, indicate that a block is
    being read
74                 readerData.inputSpot2 = 0;
75                 readerData.nextBlock = 1;
76
77                 // If we are at the end of a block of square brackets
78             } else if (input == ']' && readerData.numUID < MAX_UIDS &&
    readerData.nextBlock == 1) {
79                 // If there is a comma as the first character inside a block,
    then
80                 // discard what is read.  Otherwise, terminate the string and
    increment
81                 // the number of UIDs successfully read.
82                 if (readerData.readFlag_1 == 1) {
83                     readerData.readData[readerData.inputSpot2] = '\0';
84                     readerData.availableData++;
85                 } else if (readerData.readUID[readerData.numUID][0] != ',' &&
    readerData.writeFlag_1 != 1) {
86                     readerData.readUID[readerData.numUID][readerData.inputSpot2
    ] = '\0';
87                     readerData.numUID++;
88                 }
89
90                 // Block of square brackets has be read, set the indicator to
    zero
91                 readerData.nextBlock = 0;
92
93                 // Disable read 1 flag
94                 if (readerData.readFlag_1 == 1) {
95                     readerData.readFlag_1 = 0;
96                 }
97
98                 // Disable write 1 flag
99                 if (readerData.writeFlag_1 == 1) {
100                    readerData.writeFlag_1 = 0;
```

```
101                 }
102
103                 // Put anything inside of a square bracket into the UID array
104             } else if (readerData.nextBlock == 1 && readerData.inputSpot2 <
    UID_SIZE && readerData.numUID < MAX_UIDS) {
105                 if (readerData.readFlag_1 == 1) {
106                     readerData.readData[readerData.inputSpot2] = input;
107                     readerData.inputSpot2++;
108                 } else if (readerData.writeFlag_1 != 1) {
109                     readerData.readUID[readerData.numUID][readerData.inputSpot2
    ] = input;
110                     readerData.inputSpot2++;
111                 }
112
113                 // If we are outside of a block, reset read position and ensure
     that the block
114                 // state indicator is zero.
115             } else {
116                 readerData.inputSpot2 = 0;
117                 readerData.nextBlock = 0;
118                 // Echo back typed character
119                 //Write2USART(input);
120             }
121
122             // In config mode, count the line feeds
123         } else if (readerData.configFlag == 1) {
124             if (input == '\n') {
125                 readerData.lineFeeds++;
126             }
127         } else {
128             // Echo back typed character
129             //Write2USART(input);
130         }
131         PIR1bits.RC1IF = 0;
132     }/*
133          * LED status interrupt
134          * Card slots are:
135          * 0b00010000 = slot 1
136          * 0b00100000 = slot2
137          * 0b01000000 = slot3
138          * 0b10000000 = slot 4
139          */
140     else if (INTCONbits.RBIF == 1) {
141         char presentCards = PORTB;
142
143         globalData.lastCards ^= presentCards; // find which port is triggered
144
145         // set corresponding led status based on current port status
146         // tell rfid moudule to read the card (if needed)
```

```
147         if (globalData.lastCards & CARDSLOT_1) { //first slot
148             if (!(presentCards & CARDSLOT_1)) { // card is placed
149                 ledData.ledStatus[0] = 0;
150                 globalData.readCard |= CARDSLOT_1;
151             } else {
152                 ledData.ledStatus[0] = 3;
153             }
154         } else if (globalData.lastCards & CARDSLOT_2) { //second slot
155             if (!(presentCards & CARDSLOT_2)) { // card is placed
156                 ledData.ledStatus[1] = 0;
157                 globalData.readCard |= CARDSLOT_2;
158             } else {
159                 ledData.ledStatus[1] = 3;
160             }
161         } else if (globalData.lastCards & CARDSLOT_3) { //third slot
162             if (!(presentCards & CARDSLOT_3)) { // card is placed
163                 ledData.ledStatus[2] = 0;
164                 globalData.readCard |= CARDSLOT_3;
165             } else {
166                 ledData.ledStatus[2] = 3;
167             }
168         } else if (globalData.lastCards & CARDSLOT_4) { // fourth slot
169             if (!(presentCards & CARDSLOT_4)) { // card is placed
170                 ledData.ledStatus[3] = 0;
171                 globalData.readCard |= CARDSLOT_4;
172             } else {
173                 ledData.ledStatus[3] = 3;
174             }
175         }
176         globalData.updateLEDFlag = TRUE; // tell led driver to update leds
177         globalData.lastCards = presentCards;
178         PORTB = presentCards;
179         INTCONbits.RBIF = 0; // reset int flag
180     }
181 #endif
182     if (PIR3bits.SSP2IF == 1) { // process i2c interrupt
183         int temp = 0;
184         static unsigned char byteNumber = 0;
185 
186         //        PORTAbits.AN0 = 1;
187 
188         if (SSP2STATbits.P == 1) { // stop condition
189             i2cData.inDataSequence = FALSE;
190             i2cData.inLength = byteNumber;
191             byteNumber = 0;
192             if (!i2cData.transmitting) {
193                 globalData.gotI2C = 1; // alert the scheduler
194             }
```

```
195         } else if ((SSP2STATbits.D_A == 0) && (SSP2STATbits.BF == 1)) { //
    check if address
196             //              SSP2CON2bits.ACKDT = 0;
197             //              SSP2CON1bits.CKP = 1;
198             //              i2cData.dataOut[byteNumber++] = SSP2BUF;
199             temp = SSP2BUF; // get rid of address
200         } else if ((SSP2STATbits.D_A == 1) && (SSP2STATbits.BF == 1)) { //
    check if data
201             i2cData.dataIn[byteNumber++] = SSP2BUF;
202             //              SSP2CON2bits.ACKDT = 0;
203             //              SSP2CON1bits.CKP = 1;
204
205         } else if (SSP2STATbits.S == 1) { // start condition
206             i2cData.inDataSequence = TRUE;
207         }
208         PIR3bits.SSP2IF = 0; // clear the interrupt
209
210         //          PORTAbits.AN0 = 1;
211
212     }
213
214     // Clear interrupts
215
216     //     PIR1bits.TX1IF = 0;
217     PIR1bits.RC1IF = 0;
218     PIR3bits.TX2IF = 0;
219     PIR3bits.RC2IF = 0;
220 }
221
222 void main() {
223     int i = 0;
224     int j = 0;
225     char test[2] = {'2', '\0'};
226
227     systemSetup(&globalData);
228
229 #if FRONT_NOT_BACK
230     TRISAbits.RA0 = 0;
231     ANSELAbits.ANSA0 = 0;
232
233     printMainMenu(&globalData);
234 #else
235     //     TRISDbits.RD4 = 0;
236     //     ANSELDbits.ANSD4 = 0;
237     //     PORTDbits.RD4 = 1;
238 #endif
239
240
241
```

```
242     while (1) {
243 #if FRONT_NOT_BACK
244
245
246         // get the updated cards
247         if (globalData.runGetUpdatedCards == 1) {
248             getUpdatedCards();
249             globalData.runGetUpdatedCards = FALSE;
250         }
251 #else
252         // move the reader -- if we had one -- to the proper location
253
254
255         /*
256          * run LED driver
257          */
258         if (globalData.updateLEDFlag) {
259             globalData.updateLEDFlag = FALSE;
260             updateLEDs();
261         }
262 #endif
263
264         // process pending i2c tasks
265         if (globalData.sendI2C == 1) {
266             sendBytes(i2cData.dataOut, i2cData.outLength);
267             globalData.sendI2C = FALSE;
268         }
269         if (globalData.gotI2C == 1) {
270             processI2C();
271             globalData.gotI2C = 0;
272         }
273
274 #if FRONT_NOT_BACK
275         if (!globalData.keyFlag) {
276             // Check keystroke
277             keypad(&globalData);
278         }
279
280         if (globalData.keyFlag && !globalData.displayedKey) {
281             globalData.keyFlag = FALSE;
282             globalData.displayedKey = TRUE;
283
284             switch (globalData.displayPage) {
285                 case 0: // main menu
286                     processPrintCursor(&globalData, 3, BLUE, WHITE);
287                     if (globalData.newDisplay == 1) {
288                         globalData.newDisplay = 0;
289                         switch (globalData.cursorPos) {
290                             case 0: // go to singleplayer
```

```
291                             globalData.displayPage = 1;
292                             printSelectGame(&globalData);
293                             break;
294                         case 1: // multi
295                             globalData.displayPage = 2;
296                             printSelectGame(&globalData);
297                             break;
298                         case 2: // build
299                             globalData.displayPage = 3;
300                             printBuild(&globalData);
301                             break;
302                         case 0xFF: // back
303                             globalData.displayPage = 0;
304                             break;
305                         default:
306                             globalData.displayPage = 0;
307                             printMainMenu(&globalData);
308                             break;
309                     }
310                 }
311                 break;
312             case 1: // display sp game menu
313                 processPrintCursor(&globalData, 4, GREEN, BLACK);
314                 if (globalData.newDisplay == 1) {
315                     globalData.newDisplay = 0;
316                     switch (globalData.cursorPos) {
317                         case 0: // go to monster game
318                             globalData.displayPage = 4;
319                             singlePlayer(&globalData);
320                             break;
321                         case 1: // go to clue
322                             globalData.displayPage = 5;
323                             clean(BLACK);
324                             prints(0, 35, WHITE, BLACK, "In Progress", 1);
325                             break;
326                         case 2: // go to error
327                             globalData.displayPage = 6;
328                             printBSOD();
329                             break;
330                         case 3: // go to error
331                             clean(BLACK);
332                             prints(0, 35, WHITE, BLACK, "In Progress", 1);
333                             globalData.displayPage = 7;
334                             break;
335                         default:
336                             globalData.displayPage = 0;
337                             printMainMenu(&globalData);
338                             break;
339                     }
```

```
340                         globalData.cursorPos = 0;
341                     }
342                     break;
343                 case 2: // display mp game menu
344                     processPrintCursor(&globalData, 4, GREEN, BLACK);
345                     if (globalData.newDisplay == 1) {
346                         globalData.newDisplay = 0;
347                         switch (globalData.cursorPos) {
348                             case 0: // go to monster game
349                                 globalData.displayPage = 4;
350                                 printSelectGame(&globalData);
351                                 break;
352                             case 1: // go to clue
353                                 globalData.displayPage = 5;
354                                 clean(BLACK);
355                                 prints(0, 35, WHITE, BLACK, "In Progress", 1);
356                                 break;
357                             case 2: // go to error
358                                 globalData.displayPage = 6;
359                                 printBSOD();
360                                 break;
361                             case 3: // go to error
362                                 clean(BLACK);
363                                 prints(0, 35, WHITE, BLACK, "In Progress", 1);
364                                 globalData.displayPage = 7;
365                                 break;
366                             default:
367                                 globalData.displayPage = 0;
368                                 printMainMenu(&globalData);
369                                 break;
370                         }
371                         globalData.cursorPos = 0;
372                     }
373                     break;
374                 case 3: // build cards
375                     processPrintCursor(&globalData, 4, RED, BLACK);
376                     if (globalData.newDisplay == 1) {
377                         globalData.newDisplay = 0;
378                         switch (globalData.cursorPos) {
379                             case 0: // go to monster game
380                                 buildCard(&globalData, 0);
381                                 prints(140, 35, BLACK, RED, "DONE", 1);
382                                 globalData.displayPage = 0;
383                                 printMainMenu(&globalData);
384                                 break;
385                             case 1: // go to clue
386                                 buildCard(&globalData, 1);
387                                 prints(140, 35, BLACK, RED, "DONE", 1);
388                                 globalData.displayPage = 0;
```

```
389                             printMainMenu(&globalData);
390                             break;
391                         case 2: // go to error
392                             buildCard(&globalData, 2);
393                             prints(140, 35, BLACK, RED, "DONE", 1);
394                             globalData.displayPage = 0;
395                             printMainMenu(&globalData);
396                             break;
397                         case 3: // go to error
398                             buildCard(&globalData, 3);
399                             clean(BLACK);
400                             prints(140, 35, BLACK, RED, "DONE", 1);
401                             globalData.displayPage = 0;
402                             printMainMenu(&globalData);
403                             break;
404                         default:
405                             globalData.displayPage = 0;
406                             printMainMenu(&globalData);
407                             break;
408                     }
409                     globalData.cursorPos = 0;
410                 }
411                 break;
412             case 4: // singleplayer monster
413                 singlePlayer(&globalData);
414                 break;
415             case 5: // clue sp
416                 break;
417             case 6: // bsod
418                 break;
419             case 7: // in progress, b goes back
420                 if (globalData.keyPress == 0x0B) {
421                     globalData.displayPage = 0;
422                     printMainMenu(&globalData);
423                 }
424                 break;
425         }
426     }
427
428
429     //        if (!globalData.keyFlag) {
430     //            keypad(&globalData);
431     //        }
432
433     //        mainMenu(&globalData);
434
435     //
436     //        if (globalData.keyFlag && !globalData.displayedKey) { // TODO
    this goes into a display function
```

```
437         //              globalData.keyFlag = FALSE;
438         //              globalData.displayedKey = TRUE;
439         //
440         ////              processDisplay(globalData);
441         //
442         //          }
443
444
445 #else
446         //Doing an inventory command from the Build card menu
447         if (globalData.readCard != 0) {
448
449             //              globalData.getInventory = TRUE;
450             //          }
451             //          if (globalData.getInventory == TRUE) {
452             // get the inventory of cards
453             inventoryRFID();
454
455             // Print out each on to the LCD
456             for (i = 0; i < readerData.availableUIDs; i++) {
457                 if (readerData.readUID[i][0] != ',') {
458                     // Get rid of commas
459                     processUID(readerData.readUID[i]);
460                     //                      printrs(0, 24 + 8 * i, BLACK, RED,
    readerData.readUID[i], 1); // print first UID
461                 }
462             }
463
464             // got inventory, tell frontend
465             i2cData.dataOut[0] = CARD_CHANGE;
466             i2cData.outLength = 1;
467             globalData.sendI2C = TRUE;
468             i2cData.transmissionNum = 0; // begin counting slots
469
470             // Tell UID to be quiet - Works but needs to have at least one uid
    in this state
471             // quietRFID(readerData.readUID[0]);
472
473             //              if (readerData.availableUIDs > 0) {
474             //
475             //                  // block 0 high bits being 0x0000 indicates
    factory card, not custom
476             //                  // block 0 high bits being 0x0001 indicates
    custom card
477             //                  // block 0 low bits indicate the game the card is
     for. 0x0001 is the monster game
478             //                  writeRFID(readerData.readUID[0], 0x00, 0x0000, 0
    x0001); // 7654 3210
```

```
479             //                  writeRFID(readerData.readUID[0], 0x01, 0x0010, 0
    x0001); // hex 7-5 are for level 0x001 is level 1
480             //                  // hex 4 is for type 0 1 2 3
481             //                  // 0x0 fire, 0x1 water, 0x3 earth
482             //                  // last 4 are monster ID
483             //
484             //                  //writes 8 chars in 2 addresses of memory (0x02
    and 0x03 here)
485             //                  char8RFID(readerData.readUID[0], 0x02, "FIREDUDE
    ");
486             //                  writeRFID(readerData.readUID[0], 0x04, 0x0003, 0
    x0201); // Move list by id, has moves 03, 02, and 01
487             //                  readRFID(readerData.readUID[0], 0x00);
488             //                  printrs(0, 32, BLACK, RED, readerData.readData,
    1); // print 1st block
489             //                  readRFID(readerData.readUID[0], 0x01);
490             //                  printrs(0, 40, BLACK, RED, readerData.readData,
    1); // print 2nd block
491             //                  readRFID(readerData.readUID[0], 0x02);
492             //                  printrs(0, 48, BLACK, RED, readerData.readData,
    1); // print 3rd block
493             //                  readRFID(readerData.readUID[0], 0x03);
494             //                  printrs(0, 56, BLACK, RED, readerData.readData,
    1); // print 4th block
495             //                  readRFID(readerData.readUID[0], 0x04);
496             //                  printrs(0, 64, BLACK, RED, readerData.readData,
    1); // print 5th block
497             //              }
498             //
499             //
500             //              readRFID(readerData.readUID[0], 0x01);
501             //              // Print out block on to the LCD
502             //              for (j = 0; j < readerData.availableUIDs; j++) {
503             //                  if (readerData.readUID[j][0] != ',') {
504             //                      // Get rid of commas
505             //                      printrs(0, 24 + 8 * i + 8 * j, BLACK, RED,
    readerData.readUID[j], 1); // print first UID
506             //                  }
507             //              }
508             //
509             //              prints(0, H - 8, BLACK, RED, "Press B to go back.",
    1);
510             //              // Turn off inventory flag
511             //
512             //              globalData.getInventory = FALSE;
513             globalData.readCard = 0;
514         }
515 #endif
516     }
```

```
517     return;
518
519
520 }
521
522 // Reads the UID up until the comma
523
524 void processUID(char* uid) {
525     int i = 0;
526     while (uid[i] != ',') {
527
528         i++;
529     }
530     uid[i] = '\0';
531 }
532
533 void systemSetup(GlobalState *data) {
534
535     rs232Setup2(); // configure USART2
536     rs232Setup1(); // configure USART1
537     i2CSetup();
538     RFIDSetup();
539
540 #if FRONT_NOT_BACK
541     initSPI1();
542     initLCD();
543     keypadSetup(); // configure keypad
544     setupPWM();
545 #else
546     LEDSetup();
547 #endif
548
549     data->displayPage = 0;
550     data->keyFlag = FALSE;
551     data->displayedKey = FALSE;
552     data->keyPress = -1;
553     data->cursorPos = 0;
554     // Select Game Menu
555     data->mode = -1;
556     data->game = -1;
557     // Find better way to do this
558     data->mainMenuSpots[0] = 40;
559     data->mainMenuSpots[1] = 80;
560     data->mainMenuSpots[2] = 120;
561     data->getInventory = FALSE;
562     data->xbeeFlag = FALSE;
563     data->goBack = FALSE;
564     // Game Related Globals
565     data->keyStatus = -1;
```

```
566     memset(data->selectMove, 0, sizeof (int) * 4 * 3);
567     data->selectMove[0][1] = 10;
568     data->cardSelect[0] = 1;
569     data->cardSelect[1] = 0;
570     data->cardSelect[2] = 0;
571     data->cardSelect[3] = 0;
572     data->firstTime = TRUE;
573     data->updateLEDFlag = TRUE;
574     data->lastCards = 0;
575     data->readCard = 0;
576     data->dataBlockNum = 0;
577     data->dataSlotNum = 0;
578     memset(data->dataBlock, 0, sizeof (char) * CARDBLOCKSIZE);
579     data->runGetUpdatedCards = FALSE;
580     data->gotI2C = 0;
581     data->sendI2C = 0;
582     data->displayPage = 0;
583     data->newDisplay = 0;
584     data->newGame = 1;
585     data->newKeyboard = 1;
586     data->doneKeyboard = 0;
587     OpenTimer0(TIMER_INT_OFF & T0_SOURCE_INT & T0_PS_1_32);
588
589     return;
590 }
591
592 void setupPWM(void) {
593     // configure PWM
594
595     TRISBbits.RB5 = 1; // disable PWM output
596     CCPTMRS0 = 0b01000000; // set CCP3 to use Timer 4
597     T4CON = 0b00000111; // set timer prescale 1:16, turn on timer4
598     PR4 = 0xFF; // PR = 77 for 4kHz (cool sound). set to 0xff because will be
599     // set based on the key pressed
600
601
602     CCP3CON = 0b00011100; // set LSB of duty yle, select pwm mode
603     CCPR4L = 0x3E; // set MSB of duty cycle
604     PIR5 = 0b00000000; // clear timer interrupt flag
605     TRISBbits.RB5 = 1; //disable PWM output
606     SetDCPWM4(50); // Square wave
607 }
```

H.2 Build Cards

../FrontEnd.X/buildCard.c

```
1 void printBuildMenu(GlobalState* globalData) {
```

```
2   int i = 0;
3
4   prints(7, 10, BLACK, RED, "Available Cards:", 1);
5   processPrintCursor(globalData, size, RED, BLACK);
6 }
7
8 void buildCardMenu(GlobalState* globalData) {
9   char* buildOptions[3] = {"Add Name", "Add Move", "Remove Move"};
10  int build;
11  printMenu(buildOptions, PINK, WHITE, BLACK, BLACK, 3};
12  build = processPrintCursor(globalData, 3, BLACK, PINK);
13  switch(build) {
14    // Add Name
15    case 0:
16      addName();
17      break;
18    // Add Move
19    case 1:
20      addMove();
21      break;
22    // Remove Move
23    case 2:
24      removeMove();
25      break;
26    // Go Back
27    case 0xFF:
28      globalData->goBack = TRUE;
29      break;
30  }
31 }
32
33 void printAddNameMenu(GlobalState* globalData) {
34  clean(RED);
35  prints(35, 7, RED, BLACK, "Add the following name?", 1);
36
37  // Draw Box with name inside
38    drawBoxFill(0, 20, 20, V - 1, WHITE);
39    drawBoxBorder(0, 20, 20, V - 1, 2, BLACK, 8);
40  printrs(35, 25, BLACK, WHITE, name, 1);
41  prints(35, 40, RED, BLACK, "1. Yes", 1);
42  prints(35, 60, RED, BLACK, "2. No", 1);
43  prints(0, H - 8, RED, BLACK, "Press B to return.", 1);
44 }
45
46 void addName(GlobalState* globalData) {
47  char* name[9] = "";
48  printKeyboard(globalData, name, "NAME?", 9);
49
50 }
```

```
51
52
53 /*          //Doing an inventory command from the Build card menu
54         if (globalData.getInventory == TRUE) {
55             // get the inventory of cards
56             inventoryRFID();
57
58             // Print out each on to the LCD
59             for (i = 0; i < readerData.availableUIDs; i++) {
60                 if (readerData.readUID[i][0] != ',') {
61                     // Get rid of commas
62                     processUID(readerData.readUID[i]);
63                     printrs(0, 24 + 8 * i, BLACK, RED, readerData.readUID[i],
    1); // print first UID
64                 }
65             }
66             // Tell UID to be quiet - Works but needs to have at least one uid
    in this state
67             // quietRFID(readerData.readUID[0]);
68
69             if (readerData.availableUIDs > 0) {
70
71                 // block 0 high bits being 0x0000 indicates factory card, not
    custom
72                 // block 0 high bits being 0x0001 indicates custom card
73                 // block 0 low bits indicate the game the card is for. 0x0001
    is the monster game
74                 writeRFID(readerData.readUID[0], 0x00, 0x0000, 0x0001); // 7654
     3210
75                 writeRFID(readerData.readUID[0], 0x01, 0x0010, 0x0001); // hex
    7-5 are for level 0x001 is level 1
76                 // hex 4 is for type 0 1 2 3
77                 // 0x0 fire, 0x1 water, 0x3 earth
78                 // last 4 are monster ID
79
80                 //writes 8 chars in 2 addresses of memory (0x02 and 0x03 here)
81                 char8RFID(readerData.readUID[0], 0x02, "FIREDUDE");
82                 writeRFID(readerData.readUID[0], 0x04, 0x0003, 0x0201); // Move
     list by id, has moves 03, 02, and 01
83                 readRFID(readerData.readUID[0], 0x00);
84                 printrs(0, 32, BLACK, RED, readerData.readData, 1); // print 1
    st block
85                 readRFID(readerData.readUID[0], 0x01);
86                 printrs(0, 40, BLACK, RED, readerData.readData, 1); // print 2
    nd block
87                 readRFID(readerData.readUID[0], 0x02);
88                 printrs(0, 48, BLACK, RED, readerData.readData, 1); // print 3
    rd block
89                 readRFID(readerData.readUID[0], 0x03);
```

```
90                  printrs(0, 56, BLACK, RED, readerData.readData, 1); // print 4
    th block
91                  readRFID(readerData.readUID[0], 0x04);
92                  printrs(0, 64, BLACK, RED, readerData.readData, 1); // print 5
    th block
93              }
94
95
96              readRFID(readerData.readUID[0], 0x01);
97              // Print out block on to the LCD
98              for (j = 0; j < readerData.availableUIDs; j++) {
99                  if (readerData.readUID[j][0] != ',') {
100                     // Get rid of commas
101                     printrs(0, 24 + 8 * i + 8 * j, BLACK, RED, readerData.
    readUID[j], 1); // print first UID
102                 }
103             }
104
105             prints(0, H - 8, BLACK, RED, "Press B to go back.", 1);
106             // Turn off inventory flag
107
108             globalData.getInventory = FALSE;
109         } */
```

H.3 Example Game

Both a single player and multiplayer version of the same game are provided.

../FrontEnd.X/game.h

```
1  #include "globals.h"
2
3  typedef enum _myTypes {
4      FIRE = 0, WATER = 1, EARTH = 2
5  }Type;
6
7  typedef struct _move {
8      char moveName[9];
9      Type moveType;
10     int baseDamage;
11     int uses;
12 } Move;
13
14 typedef struct _monster {
15     char monsterName[9];
16     int monsterID;
17     int level;
18     Type monsterType;
19     Move movelist[3];
```

```
20 } Monster;
21
22 typedef struct _gameData {
23     int myScore;
24     int oppScore;
25     short turn;
26     int gameOver;
27     int monSel;
28     int moveSel;
29     Monster* myMonster;
30     Monster* oppMonster;
31     Move* myMove;
32     Move* oppMove;
33     char name[5];
34 } gameData;
35
36 void setupGame();
37
38 void processKeyboard(GlobalState* globalData, char* name, int size);
39 void printKeyboard(GlobalState* globalData, char* inputType, int size);
40
41 // Game Prototypes
42 void singlePlayer(GlobalState* globalData);
43 void multiPlayer(GlobalState* globalData);
44 void buildCards(GlobalState* globalData);
45 void getCards(void);
46 void buildCard(GlobalState* globalData, int sel);
47
48 // Helper Prototypes
49 int findPlayer(GlobalState* globalData);
50 int gameStatus(void);
51 int attack(Move* attack, Monster* monster, int targetScore);
52 void selectCard(GlobalState* globalData);
53 void pickMove(GlobalState* globalData);
54
55 // Xbee Prototypes
56 void sendMove(void);
57 void receiveMove(void);
58 void sendScore(void);
59 int receiveScore(void);
60
61 // Game Display Prototypes
62 void printGame(GlobalState* globalData);
63 void printSelect(GlobalState* globalData);
64 void printAttackMenu(GlobalState* globalData, Monster* card);
65 void printResults(void);
66 Move getMoveFromList(char id);
67 /*
68 typedef struct _gameData {
```

```
69      int myScore;
70      int oppScore;
71      short turn;
72      int gameOver;
73      int myMove;
74      int oppMove;
75      char* moveName;
76  } gameData;
77
78  */
```

../FrontEnd.X/game.c

```
1   /*
2
3
4    */
5   #include <stdio.h>
6   #include <stdlib.h>
7   #include "globals.h"
8   #include "game.h"
9   #include "LCD.h"
10
11  #define HEALTH 100
12
13
14  gameData game;
15  Monster myMonsterList[3];
16  Monster customMoster;
17
18  void setupGame() {
19      game.myScore = HEALTH;
20      game.oppScore = HEALTH;
21      game.turn = 0;
22      game.gameOver = 0;
23      game.monSel = 0;
24      game.moveSel = 0;
25      game.myMove = NULL;
26      game.oppMove = NULL;
27      game.myMonster = NULL;
28      game.oppMonster = NULL;
29      srand(ReadTimer0());
30  }
31
32  void singlePlayer(GlobalState* globalData) {
33      static int tempScore = 0;
34      static int transition = 0;
35      static int movedone = 0;
36      // First time playing?
```

```
37      // Read first time from SRAM
38      if (globalData->firstTime == TRUE) {
39          if (globalData->newKeyboard) {
40              printKeyboard(globalData, "NAME?", 5);
41          } else {
42              processKeyboard(globalData, game.name, 5);
43          }
44      }
45      if (globalData->doneKeyboard) {
46          globalData->firstTime = FALSE;
47
48
49          if (globalData->newGame) {
50              globalData->newGame = 0;
51              // Setup new game
52              setupGame();
53              tempScore = 0;
54              transition = 0;
55              game.turn = rand() % 2;
56              printGame(globalData);
57              getCards(); // Assume cards already read by interrupts on switches
    (?)
58          }
59
60
61          // Wait till someone presses D to continue
62          if (globalData->keyPress == 0x0D) {
63              transition = 1;
64          }
65
66          // Begin game with computer
67          if (transition) {
68              if (!game.gameOver) {
69                  // Situation depends on game.turn
70                  if (game.turn) {
71                      if (game.moveSel == 0) {
72                          pickMove(globalData);
73                      }
74                      if (game.moveSel == 1) {
75
76                          tempScore = game.oppScore;
77                          game.oppScore = attack(game.myMove, game.myMonster,
    game.oppScore);
78                          if (game.oppScore == tempScore) {
79                              prints(0, 10, RED, BLACK, "                    ",
    1);
80                              prints(0, 18, RED, BLACK, "                    ",
    1);
```

```
81                              prints(0, 26, RED, BLACK, "                    ",
    1);
82                              prints(0, 10, RED, BLACK, "Missed!", 1);
83                          } else {
84                              prints(0, 10, RED, BLACK, "                    ",
    1);
85                              prints(0, 18, RED, BLACK, "                    ",
    1);
86                              prints(0, 26, RED, BLACK, "                    ",
    1);
87                              prints(0, 10, RED, BLACK, "Your", 1);
88                              printrs(30, 10, RED, BLACK, game.myMonster->
    monsterName, 1);
89                              prints(84, 10, RED, BLACK, "used", 1);
90                              printrs(0, 18, RED, BLACK, game.myMove->moveName,
    1);
91                              prints(0, 26, RED, BLACK, "-     ", 1);
92                              integerprint(6, 26, RED, BLACK, tempScore - game.
    oppScore, 1);
93                          }
94                          game.moveSel = 0;
95                          movedone = 1;
96                      }
97                  } else {
98                      // Computer randomly picks a monster and attack
99                      game.oppMonster = &myMonsterList[rand() % 3 ];
100                     game.oppMove = &game.oppMonster->movelist[rand() % 3];
101                     tempScore = game.myScore;
102                     game.myScore = attack(game.oppMove, game.oppMonster, game.
    myScore);
103                     if (game.myScore == tempScore) {
104                         prints(0, 10, RED, BLACK, "                    ", 1);
105                         prints(0, 18, RED, BLACK, "                    ", 1);
106                         prints(0, 26, RED, BLACK, "                    ", 1);
107                         prints(0, 10, RED, BLACK, "Enmy Missed!", 1);
108                     } else {
109                         prints(0, 10, RED, BLACK, "                    ", 1);
110                         prints(0, 18, RED, BLACK, "                    ", 1);
111                         prints(0, 26, RED, BLACK, "                    ", 1);
112                         prints(0, 10, RED, BLACK, "Enmy", 1);
113                         printrs(30, 10, RED, BLACK, game.oppMonster->
    monsterName, 1);
114                         prints(84, 10, RED, BLACK, "used", 1);
115                         printrs(0, 18, RED, BLACK, game.oppMove->moveName, 1);
116                         prints(0, 26, RED, BLACK, "-     ", 1);
117                         integerprint(6, 26, RED, BLACK, tempScore - game.
    myScore, 1);
118                     }
119                     movedone = 1;
```

```
120                 }
121
122                 if (movedone == 1) {
123                     prints(0, 40, YELLOW, BLACK, "Your Score: ", 1);
124                     prints(0, 55, YELLOW, BLACK, "      ", 1);
125                     integerprint(0, 55, YELLOW, BLACK, game.myScore, 1);
126                     prints(0, 70, WHITE, BLACK, "Opponent Score: ", 1);
127                     prints(0, 85, WHITE, BLACK, "      ", 1);
128                     integerprint(0, 85, WHITE, BLACK, game.oppScore, 1);
129                     // Check game status
130                     game.gameOver = gameStatus();
131                     game.turn = !game.turn;
132                     transition = 0;
133                     movedone = 0;
134                 }
135                 //                 if (game.gameOver == 1) {
136                 //                     if (globalData->newGame == 0) {
137                 //                         printResults();
138                 //                     }
139                 //                 }
140             } else {
141                 if (globalData->newGame == 0) {
142                     printResults();
143                 }
144             }
145         }
146         // Display results once there is a lost
147     }
148 }
149
150 void multiPlayer(GlobalState* globalData) {
151     int connect = 0;
152     int hostOrFind = 0; // 0 host game, 1 find game
153     char* mySelections[2] = {"Host Game", "Find Game"};
154     setupXbee();
155     printMenu(mySelections, BLACK, GRAY, WHITE, YELLOW, 2);
156     prints(25, 7, YELLOW, GRAY, "Multiplayer", 1);
157     processPrintCursor(globalData, 2, BLACK, YELLOW);
158     switch (hostOrFind) {
159         case 0:
160             prints(3, 48, YELLOW, GRAY, "Waiting for players...", 1);
161             hostGame();
162             break;
163         case 1:
164             prints(3, 88, YELLOW, GRAY, "Looking for games...", 1);
165             findGame();
166             break;
167         case 0xFF:
168             return;
```

```
169             break;
170     }
171
172
173
174     setupGame();
175
176     //     // Find other players
177     //     while (!connect) {
178     //         connect = findPlayer();
179     //     }
180     //
181     //     // NOTE: Need to build something to determine who goes first
182     //     // Compare Xbee ID's perhaps?
183     //
184     //     // Begin game
185     //     while (!game.gameOver && connect) {
186     //         if (game.turn) {
187     //             // Pick Move
188     //             game.myMove = pickMove(globalData);
189     //             // Send Move
190     //             sendMove();
191     //             // Receive new score after opponent takes damage
192     //             game.oppScore = receiveScore();
193     //             printGame(globalData);
194     //             prints(0, 10, RED, BLACK, "                    ", 1);
195     //             prints(0, 10, RED, BLACK, "Opponent took damage:", 1);
196     //             prints(0, 21, RED, BLACK, "-", 1);
197     //             integerprint(6, 21, RED, BLACK, game.myMove, 1);
198     //         } else {
199     //             // Receive player move and take damage
200     //             game.myScore = attack(receiveMove(), game.myScore);
201     //             prints(0, 10, RED, BLACK, "                    ", 1);
202     //             prints(0, 10, RED, BLACK, "Taken damage:", 1);
203     //             prints(0, 21, RED, BLACK, "-", 1);
204     //             integerprint(6, 21, RED, BLACK, game.oppMove, 1);
205     //             // Send new score
206     //             sendScore();
207     //         }
208     //         prints(0, 45, YELLOW, BLACK, "        ", 1);
209     //         integerprint(0, 45, YELLOW, BLACK, game.myScore, 1);
210     //         prints(0, 75, WHITE, BLACK, "        ", 1);
211     //         integerprint(0, 75, WHITE, BLACK, game.oppScore, 1);
212     //         // Check game status
213     //         game.gameOver = gameStatus();
214     //         game.turn =!game.turn;
215     //     }
216     //     if (!connect) {
217     //         printBSOD();
```

```
218     //     } else {
219     //         printResults();
220     //     }
221 }
222
223
224 // Program data on the card
225 // Monster ID
226 // Attack ID
227
228 void buildCard(GlobalState* globalData, int sel) {
229     char myCommand[32];
230
231     switch (sel) {
232         case 0:
233             // firedude
234             // Enter data to be stored in card
235             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
236             myCommand[1] = 0x00; // slot
237             myCommand[2] = 0x00; // block
238             strcpypgm2ram(&myCommand[3], "FIRE");
239             sendBytes(myCommand, 7);
240             Delay1KTCYx(250);
241
242             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
243             myCommand[1] = 0x00; // slot
244             myCommand[2] = 0x01; // block
245             strcpypgm2ram(&myCommand[3], "DUDE");
246             sendBytes(myCommand, 7);
247             Delay1KTCYx(250);
248
249             Delay1KTCYx(250);
250             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
251             myCommand[1] = 0x00; // slot
252             myCommand[2] = 0x02; // block
253             myCommand[3] = 0x00; // type
254             myCommand[4] = 0x01; // level
255             myCommand[5] = 0x10; // mov1mov2 scratch,ember
256             myCommand[6] = 0x02; // mov3 blank, hotflame
257             sendBytes(myCommand, 7);
258
259             break;
260         case 1:
261             // earthguy
262             // Enter data to be stored in card
263             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
264             myCommand[1] = 0x00; // slot
265             myCommand[2] = 0x00; // block
266             strcpypgm2ram(&myCommand[3], "EART");
```

```
            sendBytes(myCommand, 7);
            Delay1KTCYx(250);

            myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
            myCommand[1] = 0x00; // slot
            myCommand[2] = 0x01; // block
            strcpypgm2ram(&myCommand[3], "HGUY");
            sendBytes(myCommand, 7);
            Delay1KTCYx(250);

            Delay1KTCYx(250);
            myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
            myCommand[1] = 0x00; // slot
            myCommand[2] = 0x02; // block
            myCommand[3] = 0x02; // type
            myCommand[4] = 0x01; // level
            myCommand[5] = 0x43; // mov1mov2 pound,pebble
            myCommand[6] = 0x05; // mov3 blank,earthquake
            sendBytes(myCommand, 7);
            break;
        case 2:
            // waterman
            // Enter data to be stored in card
            myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
            myCommand[1] = 0x00; // slot
            myCommand[2] = 0x00; // block
            strcpypgm2ram(&myCommand[3], "WATE");
            sendBytes(myCommand, 7);
            Delay1KTCYx(250);

            myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
            myCommand[1] = 0x00; // slot
            myCommand[2] = 0x01; // block
            strcpypgm2ram(&myCommand[3], "RMAN");
            sendBytes(myCommand, 7);
            Delay1KTCYx(250);

            Delay1KTCYx(250);
            myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
            myCommand[1] = 0x00; // slot
            myCommand[2] = 0x02; // block
            myCommand[3] = 0x01; // type
            myCommand[4] = 0x01; // level
            myCommand[5] = 0x16; // mov1mov2 scratch,squirt
            myCommand[6] = 0x07; // mov3 blank, soak
            sendBytes(myCommand, 7);
            break;
        case 3:
            // peckol
```

```
316             // Enter data to be stored in card
317             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
318             myCommand[1] = 0x00; // slot
319             myCommand[2] = 0x00; // block
320             strcpypgm2ram(&myCommand[3], "DRPE");
321             sendBytes(myCommand, 7);
322             Delay1KTCYx(250);
323
324             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
325             myCommand[1] = 0x00; // slot
326             myCommand[2] = 0x01; // block
327             strcpypgm2ram(&myCommand[3], "CKOL");
328             sendBytes(myCommand, 7);
329             Delay1KTCYx(250);
330
331             Delay1KTCYx(250);
332             myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card block
333             myCommand[1] = 0x00; // slot
334             myCommand[2] = 0x02; // block
335             myCommand[3] = 0x02; // type
336             myCommand[4] = 0x09; // level
337             myCommand[5] = 0xFF; // mov1mov2 FAIL,FAIL
338             myCommand[6] = 0x0F; // mov3 blank, FAIL
339             sendBytes(myCommand, 7);
340             Delay1KTCYx(250);
341
342             myCommand[0] = REQUEST_CARD_DATA; // Write 32-bit data to card
    block
343             myCommand[1] = 0x00; // slot
344             myCommand[2] = 0x00; // block
345             sendBytes(myCommand, 3);
346             Delay1KTCYx(250);
347             Delay1KTCYx(250);
348             break;
349     }
350
351
352
353
354     globalData->newDisplay = 1;
355     globalData->displayPage = 0;
356     // peckol card
357     // firedude
358     // earthguy
359     // waterman
360     // combogrl
361
362     // Write data to card
363
```

```
364 }
365
366 /* The following functions are helper commands for the main games.
367  * Set Up
368  * Moves
369  * Display
370  */
371
372 // Sets up connection with other players
373
374 int findPlayer(GlobalState* globalData) {
375     //     printMultiplayerSetup(globalData);
376     return 0;
377 }
378
379 // Checks if a winner has been found
380
381 int gameStatus() {
382     return (0 == game.oppScore || 0 == game.myScore);
383 }
384
385 // Regame.turns new score of player after taking damage
386
387 int attack(Move* attack, Monster* monster, int targetScore) {
388     int hitPCT = (3 * monster->level) + 75;
389     int totalDmg = attack->baseDamage + 2 * (monster->level);
390
391     if (hitPCT > (rand() % 101)) {
392
393         if (attack->moveType == monster->monsterType) {
394             totalDmg = (totalDmg * 120) / 100;
395         } else if (attack->moveType == (Type) (((3 + monster->monsterType) - 1)
    % 3)) {
396             totalDmg = (totalDmg * 80) / 100;
397         }
398     } else {
399         totalDmg = 0;
400     }
401     if (totalDmg < targetScore) {
402         return (targetScore - totalDmg);
403     } else {
404         return 0;
405     }
406 }
407
408 // User selects move based off of available cards
409 // Regame.turns damage of chosen move
410
411 void selectCard(GlobalState* globalData) {
```

```
412     static int displayed = 0;
413
414     // Beep off
415     TRISBbits.RB5 = 1;
416     if (displayed == 0) {
417         displayed = 1;
418         printSelect(globalData);
419         prints(0, 5, WHITE, RED, "Choose a card by its slot number:", 1);
420     } else {
421
422         if ((0 == globalData->keyPress || globalData->keyPress > 5)) {
423             globalData->keyStatus = 1;
424             prints(0, 5, WHITE, RED, "
                ", 1);
425             prints(0, 5, WHITE, RED, "Invalid input. Please enter a key between
    1 to 4:", 1);
426             // No card available
427         } else if (NULL == &myMonsterList[globalData->keyPress - 1]) {
428             globalData->keyStatus = 2;
429             prints(0, 5, WHITE, RED, "                                    ", 1);
430             prints(0, 5, WHITE, RED, "No card found. Please try again:", 1);
431             // Selected card is available
432         } else {
433             if (globalData->keyPress == 0x04) {
434                 game.myMonster = &customMoster;
435             } else {
436                 game.myMonster = &myMonsterList[globalData->keyPress - 1];
437             }
438             game.monSel = 1;
439             displayed = 0;
440         }
441     }
442 }
443
444 void getCards() {
445     strcpypgm2ram(myMonsterList[0].monsterName, "FIREDUDE");
446     myMonsterList[0].monsterID = 0x01;
447     myMonsterList[0].monsterType = FIRE;
448     myMonsterList[0].level = 2;
449     strcpypgm2ram(myMonsterList[0].movelist[0].moveName, "EMBER");
450     myMonsterList[0].movelist[0].baseDamage = 10;
451     myMonsterList[0].movelist[0].moveType = FIRE;
452     myMonsterList[0].movelist[0].uses = 10;
453     strcpypgm2ram(myMonsterList[0].movelist[1].moveName, "SCRATCH");
454     myMonsterList[0].movelist[1].baseDamage = 6;
455     myMonsterList[0].movelist[1].moveType = EARTH;
456     myMonsterList[0].movelist[1].uses = 15;
457     strcpypgm2ram(myMonsterList[0].movelist[2].moveName, "HOTFLAME");
458     myMonsterList[0].movelist[2].baseDamage = 20;
```

```
459     myMonsterList[0].movelist[2].moveType = FIRE;
460     myMonsterList[0].movelist[2].uses = 3;
461
462     strcpypgm2ram(myMonsterList[1].monsterName, "EARTHGUY");
463     myMonsterList[1].monsterID = 0x02;
464     myMonsterList[1].monsterType = EARTH;
465     myMonsterList[1].level = 1;
466     strcpypgm2ram(myMonsterList[1].movelist[0].moveName, "PEBBLE");
467     myMonsterList[1].movelist[0].baseDamage = 12;
468     myMonsterList[1].movelist[0].moveType = EARTH;
469     myMonsterList[1].movelist[0].uses = 5;
470     strcpypgm2ram(myMonsterList[1].movelist[1].moveName, "POUND");
471     myMonsterList[1].movelist[1].baseDamage = 3;
472     myMonsterList[1].movelist[1].moveType = EARTH;
473     myMonsterList[1].movelist[1].uses = 20;
474     strcpypgm2ram(myMonsterList[1].movelist[2].moveName, "ROCKSLDE");
475     myMonsterList[1].movelist[2].baseDamage = 30;
476     myMonsterList[1].movelist[2].moveType = EARTH;
477     myMonsterList[1].movelist[2].uses = 2;
478
479
480     strcpypgm2ram(myMonsterList[2].monsterName, "WATERMAN");
481     myMonsterList[2].monsterID = 0x03;
482     myMonsterList[2].monsterType = WATER;
483     myMonsterList[2].level = 1;
484     strcpypgm2ram(myMonsterList[2].movelist[0].moveName, "SQUIRT");
485     myMonsterList[2].movelist[0].baseDamage = 12;
486     myMonsterList[2].movelist[0].moveType = WATER;
487     myMonsterList[2].movelist[0].uses = 7;
488     strcpypgm2ram(myMonsterList[2].movelist[1].moveName, "SCRATCH");
489     myMonsterList[2].movelist[1].baseDamage = 6;
490     myMonsterList[2].movelist[1].moveType = EARTH;
491     myMonsterList[2].movelist[1].uses = 15;
492     strcpypgm2ram(myMonsterList[2].movelist[2].moveName, "SOAK");
493     myMonsterList[2].movelist[2].baseDamage = 15;
494     myMonsterList[2].movelist[2].moveType = WATER;
495     myMonsterList[2].movelist[2].uses = 5;
496
497 //     strcpypgm2ram(myMonsterList[3].monsterName, "COMBOGRL");
498 //     myMonsterList[3].monsterID = 0x04;
499 //     myMonsterList[3].monsterType = EARTH;
500 //     myMonsterList[3].level = 3;
501 //     strcpypgm2ram(myMonsterList[3].movelist[0].moveName, "SOAK");
502 //     myMonsterList[3].movelist[0].baseDamage = 15;
503 //     myMonsterList[3].movelist[0].moveType = WATER;
504 //     myMonsterList[3].movelist[0].uses = 5;
505 //     strcpypgm2ram(myMonsterList[3].movelist[1].moveName, "PEBBLE");
506 //     myMonsterList[3].movelist[1].baseDamage = 12;
507 //     myMonsterList[3].movelist[1].moveType = EARTH;
```

```
508 //     myMonsterList[3].movelist[1].uses = 5;
509 //     strcpypgm2ram(myMonsterList[3].movelist[2].moveName, "EMBER");
510 //     myMonsterList[3].movelist[2].baseDamage = 10;
511 //     myMonsterList[3].movelist[2].moveType = FIRE;
512 //     myMonsterList[3].movelist[2].uses = 10;
513
514
515
516 }
517
518 // User selects move based off of available cards
519 // Regame.turns damage of chosen move
520
521 void pickMove(GlobalState* globalData) {
522     static int displaymove = 0;
523     if (game.monSel == 0) {
524         selectCard(globalData);
525     } else {
526         if (displaymove == 0) {
527             printAttackMenu(globalData, game.myMonster);
528             prints(8, 5, WHITE, BLUE, "Please select an attack: ", 1);
529             displaymove = 1;
530         } else {
531             if (((0x0A > globalData->keyPress) || (globalData->keyPress > 0x0C)
    )) {
532                 globalData->keyStatus = 1;
533                 prints(8, 5, WHITE, BLUE, "
                                ", 1);
534                 prints(8, 5, WHITE, BLUE, "Invalid attack input. Please select
    from the options below: ", 1);
535             } else {
536                 displaymove = 0;
537                 globalData->keyStatus = 0;
538                 switch (globalData->keyPress) {
539                     case 0x0A:
540                         game.myMove = &game.myMonster->movelist[0];
541                         break;
542                     case 0x0B:
543                         game.myMove = &game.myMonster->movelist[1];
544                         break;
545                     case 0xC:
546                         game.myMove = &game.myMonster->movelist[2];
547                         break;
548                 }
549                 game.monSel = 0;
550                 game.moveSel = 1;
551                 game.myMove->uses--;
552                 printGame(globalData);
553             }
```

```
554         }
555     }
556 }
557
558 // Thing to consider: User pressing a key multiple times???
559 // Regame.turn the amount of damage the move makes
560
561 /* Xbee functions for multiplayer
562  *
563  */
564
565 // Xbee: Send move to opponent
566
567 void sendMove() {
568     game.myMove;
569 }
570
571 // Xbee: Receive move from opponent
572
573 void receiveMove() {
574     game.oppMove = 0;
575 }
576
577 // Xbee: Send my score to opponent after taking damage
578
579 void sendScore() {
580     game.myScore;
581 }
582
583 // Xbee: Receive new score from opponent after attacking
584
585 int receiveScore() {
586     return 0;
587 }
588
589 /* Display Functions for the Game
590  * These functions are built to display the game screen.
591  */
592
593
594
595 void printGame(GlobalState* globalData) {
596     // LCD menu
597     // Beep off
598     TRISBbits.RB5 = 1;
599     clean(BLACK);
600     prints(0, 0, YELLOW, BLACK, "NAME: ", 1);
601     printrs(36, 0, YELLOW, BLACK, game.name, 1);
602     if (game.turn) {
```

```
603         prints(0, 18, YELLOW, BLACK, "Your move!", 1);
604     } else {
605         prints(0, 18, YELLOW, BLACK, "Opponent's turn.", 1);
606     }
607 
608     prints(0, 40, YELLOW, BLACK, "Your Score: ", 1);
609     integerprint(0, 55, YELLOW, BLACK, game.myScore, 1);
610     prints(0, 70, WHITE, BLACK, "Opponent Score: ", 1);
611     integerprint(0, 85, WHITE, BLACK, game.oppScore, 1);
612 
613     prints(0, H - 8, WHITE, BLACK, "Press D to continue.", 1);
614 }
615 
616 void printMultiplayerSetup(GlobalState* globalData) {
617     // LCD menu
618     clean(GREEN);
619     drawBoxFill(0, 0, 20, V - 1, GREEN);
620     drawBox(0, 0, 20, V - 1, 2, WHITE);
621     prints(35, 7, WHITE, BLACK, "Main Menu", 1);
622     prints(35, globalData->mainMenuSpots[0], WHITE, GREEN, "Host Game", 1);
623     prints(35, globalData->mainMenuSpots[1], WHITE, GREEN, "Join Game", 1);
624     prints(35, globalData->mainMenuSpots[2], WHITE, GREEN, "Nevermind.", 1);
625     prints(0, H - 8, WHITE, BLUE, "2-UP,8-DOWN,D-ENTER", 1);
626     prints(25, globalData->mainMenuSpots[globalData->cursorPos], WHITE, GREEN,
    ">", 1);
627 }
628 
629 void printKeyboard(GlobalState* globalData, char* inputType, int size) {
630     int i = 0;
631     globalData->newKeyboard = 0;
632     clean(BLUE);
633     prints(0, H - 32, WHITE, BLUE, "B-DELETE", 1);
634     prints(0, H - 24, WHITE, BLUE, "2-UP,8-DOWN", 1);
635     prints(0, H - 16, WHITE, BLUE, "4-LEFT,6-RIGHT", 1);
636     prints(0, H - 8, WHITE, BLUE, "D-SEL,#-DONE", 1);
637     prints(0, 0, WHITE, BLUE, inputType, 1);
638     for (i = 0; i < size - 1; i++) {
639         prints(12 * i, 8, WHITE, BLUE, " _", 1);
640     }
641     drawBox(0, 16, 40, V - 1, 2, WHITE);
642     drawBox(0, 19, 34, V - 1, 1, WHITE);
643     prints(3, 23, WHITE, BLUE, " A B C D E F G H I J", 1);
644     prints(3, 33, WHITE, BLUE, " K L M N O P Q R S T", 1);
645     prints(3, 43, WHITE, BLUE, " U V W X Y Z [ \\ ] ^", 1);
646     prints(3, 23, WHITE, BLUE, ">", 1);
647 
648     //     keypad(globalData);
649     //
650     //     while (globalData->keyPress != 0x0F) {
```

```
651     //         keypad(globalData);
652     //         if (globalData->keyFlag && !globalData->displayedKey) {
653     //             globalData->keyFlag = FALSE;
654     //             globalData->displayedKey = TRUE;
655 }
656
657 void processKeyboard(GlobalState* globalData, char* name, int size) {
658     static short pos[] = {0, 0};
659     static short letters = 0;
660
661     if (globalData->newKeyboard == 1) {
662         globalData->newKeyboard = 0;
663         pos[0] = 0;
664         pos[1] = 0;
665         letters = 0;
666     } else {
667
668         switch (globalData->keyPress) {
669             case 0x08:
670                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, " ", 1);
671                 if (pos[1] < 2) {
672                     pos[1]++;
673                 }
674                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, ">", 1);
675                 break;
676             case 0x02:
677                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, " ", 1);
678                 if (pos[1] > 0) {
679                     pos[1]--;
680                 }
681                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, ">", 1);
682                 break;
683             case 0x04:
684                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, " ", 1);
685                 if (pos[0] > 0) {
686                     pos[0]--;
687                 }
688                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, ">", 1);
689                 break;
690             case 0x06:
691                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, " ", 1);
692                 if (pos[0] < 9) {
693                     pos[0]++;
694                 }
695                 prints(3 + 12 * pos[0], 10 * pos[1] + 23, WHITE, BLUE, ">", 1);
696                 break;
697             case 0x0D:
698                 if (letters < size - 1) {
699                     name[letters] = (char) (65 + pos[0] + 10 * pos[1]);
```

```
700                     name[letters + 1] = '\0';
701                     ASCII(6 + 12 * letters, 8, WHITE, BLUE, name[letters], 1);
702                     letters = letters + 1;
703                     if (letters == size - 1) {
704                         prints(42 + 12 * 5, 0, WHITE, BLUE, "end", 1);
705                     }
706                 }
707                 break;
708             case 0x0B:
709                 if (letters > 0) {
710                     if (letters == size - 1) {
711                         prints(42 + 12 * 5, 0, WHITE, BLUE, "   ", 1);
712                     }
713                     letters--;
714                     ASCII(6 + 12 * letters, 8, WHITE, BLUE, '_', 1);
715                     name[letters] = '\0';
716                 }
717                 break;
718             case 0x0F:
719                 globalData->doneKeyboard = 1;
720                 globalData->newKeyboard = 1;
721                 break;
722         }
723 
724     }
725 }
726 
727 
728 
729 
730 // Select a card to play
731 
732 void printSelect(GlobalState* globalData) {
733     char myCommand[16];
734     int i = 0;
735     // Beep off
736     TRISBbits.RB5 = 1;
737 
738     // LCD menu
739     clean(RED);
740     // Display commands to select a slot - the LED's should indicate if a card
    is read
741     //    drawBoxFill(15, 29, 26, 85, BLACK)
742     for (i = 0; i < 4; i++) {
743         drawBoxFill(15, 29 + 35 * i, 26, 85, BLACK);
744         if ( i != 3) {
745             printrs(20, 38 + 35 * i, WHITE, BLACK, myMonsterList[i].monsterName
    , 1);
746             prints(20, 38 + 8 + 35 * i, WHITE, BLACK, "Lvl:", 1);
```

```
747             integerprint(38, 38 + 8 + 35 * i, WHITE, BLACK, myMonsterList[i].
    level, 1);
748             switch (myMonsterList[i].monsterType) {
749                 case FIRE:
750                     drawBoxFill(61, 38 + 8 + 35 * i - 1, 8, 30, RED);
751                     prints(62, 38 + 8 + 35 * i, WHITE, RED, "FIRE", 1);
752                     break;
753                 case WATER:
754                     drawBoxFill(61, 38 + 8 + 35 * i - 1, 8, 35, CYAN);
755                     prints(62, 38 + 8 + 35 * i, WHITE, CYAN, "WATER", 1);
756                     break;
757                 case EARTH:
758                     prints(62, 38 + 8 + 35 * i, WHITE, BLACK, "EARTH", 1);
759                     break;
760             }
761         } else {
762             myCommand[0] = REQUEST_CARD_DATA; // Write 32-bit data to card
    block
763             myCommand[1] = 0x00; // slot
764             myCommand[2] = 0x00; // block
765             sendBytes(myCommand, 3);
766             customMoster.monsterName[0] = (char)(((globalData->dataBlock[0] - '
    0') << 4) | ((globalData->dataBlock[1] -'0')));
767             customMoster.monsterName[1] = (char)(((globalData->dataBlock[2] - '
    0') << 4) | ((globalData->dataBlock[3] -'0')));
768             customMoster.monsterName[2] = (char)(((globalData->dataBlock[4] - '
    0') << 4) | ((globalData->dataBlock[5] -'0')));
769             customMoster.monsterName[3] = (char)(((globalData->dataBlock[6] - '
    0') << 4) | ((globalData->dataBlock[7] -'0')));
770             Delay1KTCYx(250);
771             myCommand[0] = REQUEST_CARD_DATA; // Write 32-bit data to card
    block
772             myCommand[1] = 0x00; // slot
773             myCommand[2] = 0x01; // block
774             sendBytes(myCommand, 3);
775             strcpy(customMoster.monsterName, globalData->dataBlock);
776             customMoster.monsterName[4] = (char) (((globalData->dataBlock[0] -
    '0') << 4) | ((globalData->dataBlock[1] - '0')));
777             customMoster.monsterName[5] = (char) (((globalData->dataBlock[2] -
    '0') << 4) | ((globalData->dataBlock[3] - '0')));
778             customMoster.monsterName[6] = (char) (((globalData->dataBlock[4] -
    '0') << 4) | ((globalData->dataBlock[5] - '0')));
779             customMoster.monsterName[7] = (char) (((globalData->dataBlock[6] -
    '0') << 4) | ((globalData->dataBlock[7] - '0')));
780             customMoster.monsterName[8] = '\0';
781             Delay1KTCYx(250);
782             myCommand[0] = REQUEST_CARD_DATA; // Write 32-bit data to card
    block
783             myCommand[1] = 0x00; // slot
```

```
784             myCommand[2] = 0x02; // block
785             sendBytes(myCommand, 3);
786             Delay1KTCYx(250);
787             customMoster.monsterType = (Type) (globalData->dataBlock[0] - '0');
788             customMoster.level = (globalData->dataBlock[3] - '0');
789             customMoster.movelist[0] = getMoveFromList(globalData->dataBlock
    [4]);
790             customMoster.movelist[1] = getMoveFromList(globalData->dataBlock
    [4]);
791             customMoster.movelist[2] = getMoveFromList(globalData->dataBlock
    [4]);
792
793
794             printrs(20, 38 + 35 * i, WHITE, BLACK, customMoster.monsterName, 1)
    ;
795             prints(20, 38 + 8 + 35 * i, WHITE, BLACK, "Lvl:", 1);
796             integerprint(38, 38 + 8 + 35 * i, WHITE, BLACK, customMoster.level,
     1);
797             switch (customMoster.monsterType) {
798                 case FIRE:
799                     drawBoxFill(61, 38 + 8 + 35 * i - 1, 8, 30, RED);
800                     prints(62, 38 + 8 + 35 * i, WHITE, RED, "FIRE", 1);
801                     break;
802                 case WATER:
803                     drawBoxFill(61, 38 + 8 + 35 * i - 1, 8, 35, CYAN);
804                     prints(62, 38 + 8 + 35 * i, WHITE, CYAN, "WATER", 1);
805                     break;
806                 case EARTH:
807                     prints(62, 38 + 8 + 35 * i, WHITE, BLACK, "EARTH", 1);
808                     break;
809             }
810
811
812 //              0 0       0 1     1 0       02
813 //
814 //              30 30    30 31    31 30    30 32
815 //
816 //              Delay1KTCYx(250);
817 //              myCommand[0] = WRITE_CARD_BLOCK; // Write 32-bit data to card
    block
818 //              myCommand[1] = 0x00; // slot
819 //              myCommand[2] = 0x02; // block
820 //
821 //              myCommand[3] = 0x00; // type
822 //              myCommand[4] = 0x01; // level
823 //              myCommand[5] = 0x10; // mov1mov2 scratch,ember
824 //              myCommand[6] = 0x02; // mov3 blank, hotflame
825 //              sendBytes(myCommand, 7);
826
```

```
827
828         }
829     }
830     prints(20, 30, WHITE, BLACK, "Slot 1", 1);
831     prints(20, 65, WHITE, BLACK, "Slot 2", 1);
832     prints(20, 100, WHITE, BLACK, "Slot 3", 1);
833     prints(20, 135, WHITE, BLACK, "Slot 4", 1);
834 }
835
836 Move getMoveFromList(char id) {
837     Move newMove;
838     switch(id) {
839         case 'F':
840             strcpypgm2ram(newMove.moveName, "FAIL");
841             newMove.baseDamage = 100;
842             newMove.moveType = EARTH;
843             newMove.uses = 10;
844             break;
845         default:
846             strcpypgm2ram(newMove.moveName, "SCRATCH");
847             newMove.baseDamage = 6;
848             newMove.moveType = EARTH;
849             newMove.uses = 15;
850             break;
851     }
852     return newMove;
853
854 }
855
856 // Select an attack.
857
858 void printAttackMenu(GlobalState* globalData, Monster* card) {
859     int i = 0;
860     // Beep off
861     TRISBbits.RB5 = 1;
862
863     // LCD menu
864     clean(BLUE);
865
866     switch (card->monsterType) {
867         case FIRE:
868             drawBoxFill(0, 31, 8, 48, RED);
869             printrs(0, 32, WHITE, RED, card->monsterName, 1);
870             break;
871         case WATER:
872             drawBoxFill(0, 31, 8, 48, CYAN);
873             printrs(0, 32, WHITE, CYAN, card->monsterName, 1);
874             break;
875         case EARTH:
```

```
876             drawBoxFill(0, 31, 8, 48, BLACK);
877             printrs(0, 32, WHITE, BLACK, card->monsterName, 1);
878             break;
879     }
880     prints(54, 32, WHITE, BLUE, "Lvl:", 1);
881     integerprint(84, 32, WHITE, BLUE, card->level, 1);
882
883     prints(0, 40, YELLOW, BLUE, "A. Attack with:", 1);
884     prints(0, 80, YELLOW, BLUE, "B. Attack with: ", 1);
885     prints(0, 120, YELLOW, BLUE, "C. Attack with: ", 1);
886
887     for (i = 0; i < 3; i++) {
888         switch (card->movelist[i].moveType) {
889             case FIRE:
890                 drawBoxFill(0, 40 + 40 * i + 8 - 1, 8, 83, RED);
891                 printrs(0, 40 + 40 * i + 8, WHITE, RED, card->movelist[i].
    moveName, 1);
892                 prints(54, 40 + 40 * i + 8, WHITE, RED, "FIRE", 1);
893                 break;
894             case WATER:
895                 drawBoxFill(0, 40 + 40 * i + 8 - 1, 8, 83, CYAN);
896                 printrs(0, 40 + 40 * i + 8, WHITE, CYAN, card->movelist[i].
    moveName, 1);
897                 prints(54, 40 + 40 * i + 8, WHITE, CYAN, "WATER", 1);
898                 break;
899             case EARTH:
900                 drawBoxFill(0, 40 + 40 * i + 8 - 1, 8, 83, BLACK);
901                 printrs(0, 40 + 40 * i + 8, WHITE, BLACK, card->movelist[i].
    moveName, 1);
902                 prints(54, 40 + 40 * i + 8, WHITE, BLACK, "EARTH", 1);
903                 break;
904         }
905         prints(0, 40 + 40 * i + 16, WHITE, BLUE, "Min damage:", 1);
906         integerprint(60, 40 + 40 * i + 16, YELLOW, BLUE, card->movelist[i].
    baseDamage, 1);
907         prints(0, 40 + 40 * i + 24, WHITE, BLUE, "Uses left:", 1);
908         integerprint(72, 40 + 40 * i + 24, YELLOW, BLUE, card->movelist[i].uses
    , 1);
909     }
910 }
911
912 // Displays Game Results
913
914 void printResults() {
915     // Beep off
916     TRISBbits.RB5 = 1;
917     prints(0, 15, YELLOW, BLACK, "GAME OVER", 1);
918
919     if (game.myScore > game.oppScore) {
```

```
920             clean(BLACK);
921             prints(0, 15, YELLOW, BLACK, "You won!", 1);
922             prints(0, 30, YELLOW, BLACK, "Your Score: ", 1);
923             integerprint(0, 45, YELLOW, BLACK, game.myScore, 1);
924             prints(0, 60, WHITE, BLACK, "Opponent Score: ", 1);
925             integerprint(0, 75, WHITE, BLACK, game.oppScore, 1);
926         } else {
927             clean(RED);
928             prints(0, 15, BLACK, RED, "You lost", 1);
929             prints(0, 30, BLACK, RED, "Your Score: ", 1);
930             integerprint(0, 45, WHITE, RED, game.myScore, 1);
931             prints(0, 60, YELLOW, RED, "Opponent Score: ", 1);
932             integerprint(0, 75, YELLOW, RED, game.oppScore, 1);
933         }
934         if (globalData.keyPress == 0x0B) {
935             globalData.displayPage = 0;
936             globalData.newGame = 1;
937         }
938 }
```

H.4 I^{2}C InterPIC Communication

../FrontEnd.X/i2cComm.h

```
1  /*
2   * File:   i2cComm.h
3   * Author: Patrick
4   *
5   * Created on June 5, 2014, 2:22 AM
6   */
7
8  #ifndef I2CCOMM_H
9  #define I2CCOMM_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14 #include "globals.h"
15
16 #define MAX_IN_LENGTH 20
17 #define MAX_OUT_LENGTH 20
18
19     /*
20      * Configures the MSSP2 module for master/slave-idle at 100kHz
21      */
22     void i2CSetup();
23
24     /*
```

```
25      * Sends the first numBytes bytes of the char array
26      */
27     int sendBytes(char *data, int numBytes);
28
29     /*
30      * Processes received data and commands
31      */
32     void processI2C();
33
34     typedef struct i2cDataStruct {
35         char inDataSequence;
36         char destAddr;
37         char myAddr;
38         char dataOut[MAX_OUT_LENGTH]; // data from the i2c
39         unsigned char outLength;
40         char dataIn[MAX_IN_LENGTH];
41         unsigned char inLength;
42         int transmissionNum; // the curent transmission length
43         char transmitting; // are we transmitting
44     } I2cDataStruct;
45
46     /************Inter-PIC Commands*****************************/
47 #define INVALID_COMMAND      0xFF //received command is not recognized
48 #define RECEIVE_ERROR        0xFE //received command cannot be fulfilled
49 #define END_OF_TRANSMISSION  0xFD //
50     /*-------------------Front End SendS-------------------------------*/
51 #define REQUEST_CARD_UPDATE  0x02 // Ask for cards present
52 #define REQUEST_CARD_DATA    0x04 //  Requests data from a block
53 #define WRITE_CARD_BLOCK     0xFC // Write 32-bit data to card block
54 #define WRITE_AFI            0xFB // Write data to Attribute Family Identifier
55 #define WRITE_DSFID          0xFA // Write data to Data Structure Format ID
56     /*-------------------Back End Sends-------------------------------*/
57 #define CARD_CHANGE          0x01 // Indicates cards in play changed
58 #define CARD_UID             0x02 // Sending slot# + UID
59 #define CARD_DATA_BLOCK     0x04 // Sends requested block data
60     /*****************************************************************/
61
62 #ifdef  __cplusplus
63 }
64 #endif
65
66 #endif  /* I2CCOMM_H */
```

../FrontEnd.X/i2cComm.c

```
1 /*
2  * Created by: Patrick
3  * June 2, 2014
4  *
```

```
5  * I2C system for the front and back ends. The system is configured as a
6  * Master/slave-idle. The i2c uses MSSP2 for communication
7  */
8
9  #include "globals.h"
10
11 /*******Configuration Settings*****************************/
12 #define MASTER  0b00001000 //I2C  Master mode
13 #define SLAVE   0b00000110 // I2C slave mode, 7-bit address
14 #define SSPEN   0b00100000  /* Enable serial port and configures SCK, SDO, SDI
       */
15 #define SSPDIS  0b11011111   // disable serial port
16 #define SLEW_OFF 0b10000000 /* Slew rate disabled for 100kHz mode */
17 #define BAUD    0x31;  // master address value for 100kHz baud rate
18 ///************Inter-PIC Commands*****************************/
19 //#define INVALID_COMMAND     0xFF //received command is not recognized
20 //#define RECEIVE_ERROR       0xFE //received command cannot be fulfilled
21 //#define END_OF_TRANSMISSION 0xFD //
22 ///*--------------------Front End Sends-------------------------------*/
23 //#define REQUEST_CARD_UPDATE 0x02 // Ask for cards present
24 //#define REQUEST_CARD_DATA   0x04 //  Requests data from a block
25 //#define WRITE_CARD_BLOCK    0xFC // Write 32-bit data to card block
26 //#define WRITE_AFI           0xFB // Write data to Attribute Family Identifier
27 //#define WRITE_DSFID         0xFA // Write data to Data Structure Format ID
28 ///*-----------------------Back End Sends-------------------------------*/
29 //#define CARD_CHANGE         0x01 // Indicates cards in play changed
30 //#define CARD_UID            0x02 // Sending slot# + UID
31 //#define CARD_DATA_BLOCK    0x04 // Sends requested block data
32 ///*************************************************************/
33
34 I2cDataStruct i2cData;
35
36 /*******Private prototypes*********************/
37 void switchToSlave(void);
38 void switchToMaster(void);
39 void sendStop(void);
40 void sendStart(void);
41 /**********************************************/
42
43 /*
44  * Initialization function for the i2c module
45  */
46 void i2CSetup() {
47     // setup the struct
48 #if FRONT_NOT_BACK
49     i2cData.destAddr = 0xFE; // 8-bit address, 7-bit address is 0x7F
50     i2cData.myAddr = 0x00;
51 #else
52     i2cData.destAddr = 0x00; // 0b0000000
```

```
53      i2cData.myAddr = 0xFE;
54  #endif
55
56      i2cData.inDataSequence = FALSE;
57      memset(i2cData.dataIn, '\0', sizeof(char) * MAX_IN_LENGTH);
58      memset(i2cData.dataOut, '\0', sizeof(char) * MAX_OUT_LENGTH);
59      i2cData.inLength = 0;
60      i2cData.outLength = 0;
61      i2cData.transmissionNum = 0;
62      i2cData.transmitting = 0;
63
64      // setup D0, D1 as inputs
65      TRISDbits.TRISD0 = 1;
66      TRISDbits.TRISD1 = 1;
67
68      // setup associated ANSEL bits as digital
69      ANSELDbits.ANSD0 = 0;
70      ANSELDbits.ANSD1 = 0;
71
72      // enable the interrupt priority bits
73      INTCONbits.GIEH = 1; // global int enable
74      INTCONbits.PEIE = 1; // peripheral int enable
75      IPR3bits.SSP2IP = 1; // MSSP interrupt high priority
76      PIE3bits.SSP2IE = 1; // MSSP interrupt enable
77
78      // configure MSSP2 for i2c communication
79      SSP2CON1 &= SSPDIS; // disable module
80      SSP2STAT |= SLEW_OFF;
81
82      SSP2CON1 = SLAVE;
83      SSP2ADD = i2cData.myAddr;
84
85      SSP2CON2 = 0b00000000; // disable general call interrupt
86      SSP2CON3 = 0b01100000; // enable stop int, enable start int, addr/data hold
87
88      SSP2CON1 |= SSPEN; // enable module
89
90  }
91
92  /*
93   * Process received commands
94   */
95  void processI2C() {
96      unsigned int slotNum;
97      unsigned int i;
98      unsigned char blockNum;
99      unsigned char data[4];
100 #if FRONT_NOT_BACK // receives backends commands
101     // switch on command
```

```
102     // parse data into parts
103     switch (i2cData.dataIn[0]) {
104         case CARD_CHANGE:
105             globalData.runGetUpdatedCards = TRUE;
106             readerData.availableUIDs = 0;
107             break;
108         case CARD_UID:
109             slotNum = i2cData.dataIn[1]; // slot number
110             strncpy(&readerData.readUID[slotNum], &i2cData.dataIn[2], 16); //
    move UID
111             readerData.availableUIDs++;
112             if (readerData.availableUIDs < 4) { // get the next card
113                 globalData.runGetUpdatedCards = TRUE;
114             }
115             break;
116         case CARD_DATA_BLOCK:
117             globalData.dataSlotNum = i2cData.dataIn[1]; // get the clot number
118             globalData.dataBlockNum = i2cData.dataIn[2]; // get the block
    number
119             strncpy(&globalData.dataBlock[0], &i2cData.dataIn[3], 8); // copy
    the blcok data
120             break;
121         case INVALID_COMMAND:
122
123             break;
124         case RECEIVE_ERROR:
125
126             break;
127         case END_OF_TRANSMISSION:
128
129             break;
130         default:
131             i2cData.dataOut[0] = INVALID_COMMAND;
132             i2cData.outLength = 1;
133             break;
134     }
135 #else
136     switch (i2cData.dataIn[0]) {
137         case REQUEST_CARD_UPDATE:
138             //send all four cards TODO:
139             // need to make sure all of these are sent somehow
140             slotNum = i2cData.transmissionNum++; // get our count
141             i2cData.dataOut[0] = CARD_UID;
142             i2cData.dataOut[1] = slotNum;
143             strncpy(&i2cData.dataOut[2], &readerData.readUID[slotNum], 16);
144             i2cData.outLength = 10;
145             globalData.sendI2C = TRUE;
146             break;
147         case REQUEST_CARD_DATA:
```

```
148             slotNum = i2cData.dataIn[1]; // get the requested slot
149             blockNum = i2cData.dataIn[2]; // get the requested block
150             i2cData.dataOut[0] = CARD_DATA_BLOCK; // cmd
151             i2cData.dataOut[1] = slotNum; // slot
152             i2cData.dataOut[2] = blockNum; // block
153             readRFID(readerData.readUID[slotNum], blockNum);
154             strncpy((char*)&i2cData.dataOut[3],(char*) &readerData.readData[2],
    8);
155 //              for (i = 3; i < 7; i++) {// read from sram
156 //                  i2cData.dataOut[i] = readData(256 * slotNum + 4 * blockNum +
    (i - 3));
157 //              }// format response
158             i2cData.outLength = 11;
159             globalData.sendI2C = TRUE; // send the data
160             break;
161         case WRITE_CARD_BLOCK:
162             slotNum = i2cData.dataIn[1]; // get slot
163             blockNum = i2cData.dataIn[2]; //get block
164             strncpy(&data[0], &i2cData.dataIn[3], 4); // get data bytes
165             // move to card
166             writeRFID(readerData.readUID[slotNum], blockNum, ((int)data[0] << 8
    | (int)data[1]), ((int)data[2] << 8 | (int)data[3])); // write
167             break;
168         case WRITE_AFI:
169
170             break;
171         case WRITE_DSFID:
172
173             break;
174         case INVALID_COMMAND:
175
176             break;
177         case RECEIVE_ERROR:
178
179             break;
180         case END_OF_TRANSMISSION:
181
182             break;
183         default:
184             i2cData.dataOut[0] = INVALID_COMMAND;
185             i2cData.outLength = 1;
186             break;
187     }
188 #endif
189
190 }
191
192 /* send bytes as the master.  checks the status of the bus before entering
193  * master mode. Returns a negative number if error occurs during writing,
```

```
194  * otherwise exits with 0. Exit state: slave mode.
195  */
196
197 int sendBytes(char *data, int numBytes) {
198     int i = 0;
199     signed char status = 0;
200
201     // enter masater mode if no data is being sent (stop bit last seen)
202     if (!i2cData.inDataSequence) {
203         switchToMaster();
204
205     } else {
206         return -1;
207     }
208
209     sendStart();
210
211     status = WriteI2C2(i2cData.destAddr & 0b11111110); // send address with
        write
212     if (status < 0) {// if collision, revert to slave, reset data sequence flag
        to transfer ok
213         sendStop();
214         switchToSlave();
215         return status;
216     } else { // if no collision,
217         for (i = 0; i < numBytes; i++) { //write out Numbytes of data
218             status = WriteI2C2(data[i]);
219             if (status < 0) {// if nack or wcol, break
220                 sendStop();
221                 switchToSlave();
222                 return status;
223             }
224         }
225     }
226     sendStop();
227     switchToSlave(); // switch back to slave mode
228 }
229
230 /*
231  * Sets module to master mode. Note this clears various flags in the module
232  */
233 void switchToMaster() {
234     // set address reg to baud rate?
235     // switch out of RCEN mode?
236     SSP2CON1 &= SSPDIS; // diable module
237     SSP2CON1 = MASTER; // change mode
238     SSP2ADD = BAUD; // Set baud rate
239     SSP2CON1 |= SSPEN; // enable module
240     i2cData.transmitting = 1;
```

```
241 }
242
243 /*
244  * Sets module to slave mode. Note this clears various flags in the module
245  */
246 void switchToSlave() {
247     SSP2CON1 &= SSPDIS; // diable module
248     SSP2CON1 = SLAVE; // change mode
249     SSP2ADD = i2cData.myAddr; // update address buffer
250     SSP2CON1 |= SSPEN; // enable module
251     i2cData.transmitting = 0;
252 }
253
254 /*
255  * Sends a stop bit. Module must be in Master mode
256  */
257 void sendStop() {
258     SSP2CON2bits.PEN = 1; // send stop
259     while (SSP2CON2bits.PEN == 1);
260     i2cData.inDataSequence = FALSE;
261 }
262
263 /*
264  * Sends a start bit. Module must be in Master mode
265  */
266 void sendStart() {
267     SSP2CON2bits.SEN = 1; // send start bit
268     i2cData.inDataSequence = TRUE;
269     while (SSP2CON2bits.SEN == 1); // or use IdleI2C2()
270 }
271
272 //Front end only: requests the updated cards from the backend
273
274 void getUpdatedCards() {
275 #if FRONT_NOT_BACK
276     i2cData.dataOut[0] = REQUEST_CARD_UPDATE; // get update
277     i2cData.outLength = 1;
278     globalData.sendI2C = TRUE; // send the command
279 #endif
280 }
```

H.5 Keypad Driver

../FrontEnd.X/keypadDriver.h

```
1 /*
2  * File:   keypadDriver.h
3  * Author: ma
```

```
 *
 * Created on May 23, 2014, 9:10 PM
 */

#ifndef KEYPADDRIVER_H
#define KEYPADDRIVER_H

#ifdef  __cplusplus
extern "C" {
#endif

#include "globals.h"
    // prototypes
    /*
     * keypadSetup() initializes the GPIO pins used by the keypad module.
     * Pins B0–B3 are used as digital inputs with internal weak pull up to Vcc
     * Pins C0–C3 are used as digital outputs
     */
    void keypadSetup(void);

    /*
     * scans the keypad and returns a numeric value of the key
     * which is pressed, or -1 otherwise. The codding is:
     *  numbers = corresponding number
     * A = 10
     * B = 11
     * C = 12
     * D = 13
     * * = 14
     * # = 15
     */
    void keypad(struct globaldata *gData);


#ifdef  __cplusplus
}
#endif

#endif  /* KEYPADDRIVER_H */
```

../FrontEnd.X/keypadDriver.c

```
/*
 * Created: Patrick Ma
 * Date: May 21, 2014
 *
 * keypadDriver.c
 *
 * The software driver for the keypad. The current option uses 8 pins and polls
```

```
8  * the matrix looking for key presses. Each row is polled (pins 4-7) and the
9  * keys are read from pins 0-3
10 *
11 * The keypad has a layout like this:
12 *
13 *    1|  2|  3|  A| pin
14 *  ------------------ 4
15 *    4|  5|  6|  B|
16 *  ------------------ 5
17 *    7|  8|  9|  C|
18 *  ------------------ 6
19 *    *|  0|  #|  D|
20 *  ------------------ 7
21 *     |   |   |   |
22 * pin 0   1   2   3
23 *
24 */
25
26 #define __KEYPAD_DEBUG 0
27
28 #include <stdio.h>
29 #include <stdlib.h>
30 #include <usart.h>
31 #include <delays.h>
32
33 #include "keypadDriver.h"
34
35 #ifdef __KEYPAD_DEBUG
36 #include "rs232.h"  // use serial for debugging
37 #include <p18f46k22.h>
38
39 ///***************Clocking set up *********************/
40 //#pragma config WDTEN = OFF    // turn off watch dog timer
41 //#pragma config FOSC = ECHP    // Ext. Clk, Hi Pwr
42 //#pragma config PRICLKEN = OFF // disable primary clock
43 ///*****************************************************/
44 //#pragma config PBADEN = OFF   // turn off the ADCs for whatever pins I'm
      using
45
46 /*****************************************************/
47 #endif
48
49 int resetPins(int key);
50 int checkForInput(void);
51
52 //void main() {
53 //    char keyNum = 0;
54 //    keypadSetup();
55 //    rs232Setup1(); // setup the serial port
```

```
56 //
57 //     while (1) { // just loop for test
58 //         keyNum = (char) checkForInput() + '`';
59 //         Write1USART(keyNum);
60 //
61 //         Delay1KTCYx(1);
62 //     }
63 //
64 //}
65
66 #define CODE_SIZE 6
67 #define NUM_KEYS 16
68 #define DEBUG_BSOD 6
69
70 // Sounds range from 4600Hz to 1600 Hz
71 int keySounds[NUM_KEYS] = {81, 141, 74, 194, 67, 119, 91, 97, 173, 110, 86,
       129, 70, 155, 103, 77};
72
73 // EEDEGF# (1318.51), (1318.51) (2349.32) (1318.51) (3135.96) (2959.96)
74 int secretSounds[CODE_SIZE] = {118, 118, 132, 118, 99, 118};
75 int secretCode[CODE_SIZE] = {6,6,6,4,2,0};
76 short codeCounter = 0;
77
78 // 6 Stars will make a BSOD happen
79 short debugCount = 0;
80 /*
81  * Polls the keypad for a key press. The key is stored in the global data
       struct given.
82  */
83 void keypad(GlobalState *gData) {
84     gData->keyPress = checkForInput();
85     if (gData->keyPress >= 0 && !gData->displayedKey) {
86         gData->keyFlag = TRUE;
87         // Weak debounce
88         Delay10TCYx(20);
89
90         if (gData->keyPress == 14) {
91             debugCount++;
92         } else {
93             debugCount = 0;
94         }
95
96         if (debugCount == DEBUG_BSOD) {
97             gData->keyPress = 0xFF;
98         }
99
100        if (gData->keyPress == secretCode[codeCounter]) {
101            PR4 = secretSounds[codeCounter];
102            codeCounter = (codeCounter + 1) % CODE_SIZE;
```

```
103         } else {
104             // Set tone
105             PR4 = keySounds[gData->keyPress];
106             codeCounter = 0;
107         }
108 
109         // Beep on
110         TRISBbits.RB5 = 0;
111     } else if (gData->keyPress < 0) {
112         gData->displayedKey = FALSE;
113         // Beep off
114         TRISBbits.RB5 = 1;
115     }
116 }
117 
118 // checks the keypad for key press. returns the first key press sensed.
119 // Returns the number of the key pressed (* = 14, # = 15)
120 
121 int checkForInput() {
122     char scan;
123 
124     PORTDbits.RD2 = 0; // check row1
125     Delay10TCYx(10);
126     scan = (PORTBbits.RB3 << 3 | PORTBbits.RB2 << 2 | PORTBbits.RB1 << 1 |
        PORTBbits.RB0);
127     switch (scan) {
128         case 0b00001110: // is on Bbits.RB3
129             return resetPins(1);
130         case 0b00001101: // on pin 5
131             return resetPins(2);
132         case 0b00001011: // on pin 6
133             return resetPins(3);
134         case 0b00000111: // on pin 7
135             return resetPins(10);
136     }
137 
138     PORTDbits.RD2 = 1; // check row2
139     PORTDbits.RD3 = 0;
140     Delay10TCYx(10);
141     scan = (PORTBbits.RB3 << 3 | PORTBbits.RB2 << 2 | PORTBbits.RB1 << 1 |
        PORTBbits.RB0);
142     switch (scan) {
143         case 0b00001110: // is on Bbits.RB3
144             return resetPins(4);
145         case 0b00001101: // on pin 5
146             return resetPins(5);
147         case 0b00001011: // on pin 6
148             return resetPins(6);
149         case 0b00000111: // on pin 7
```

```
150             return resetPins(11);
151     }
152
153
154     PORTDbits.RD3 = 1; // check row3
155     PORTDbits.RD4 = 0;
156     Delay10TCYx(10);
157     scan = (PORTBbits.RB3 << 3 | PORTBbits.RB2 << 2 | PORTBbits.RB1 << 1 |
        PORTBbits.RB0);
158     switch (scan) {
159         case 0b00001110: // is on Bbits.RB3
160             return resetPins(7);
161         case 0b00001101: // on pin 5
162             return resetPins(8);
163         case 0b00001011: // on pin 6
164             return resetPins(9);
165         case 0b00000111: // on pin 7
166             return resetPins(12);
167     }
168
169     PORTDbits.RD4 = 1; // check row4
170     PORTDbits.RD5 = 0;
171     Delay10TCYx(10);
172     scan = (PORTBbits.RB3 << 3 | PORTBbits.RB2 << 2 | PORTBbits.RB1 << 1 |
        PORTBbits.RB0);
173     switch (scan) {
174         case 0b00001110: // is on Bbits.RB3
175             return resetPins(14);
176         case 0b00001101: // on pin 5
177             return resetPins(0);
178         case 0b00001011: // on pin 6
179             return resetPins(15);
180         case 0b00000111: // on pin 7
181             return resetPins(13);
182     }
183
184     return resetPins(-1); // assume no key was pressed.
185 }
186
187
188 // resets the driving pins
189
190 int resetPins(int key) {
191     PORTDbits.RD5 = 1; // set outputs HIGH
192     PORTDbits.RD4 = 1;
193     PORTDbits.RD3 = 1;
194     PORTDbits.RD2 = 1;
195     return key;
196 }
```

```
197
198 /* sets all pins to input or output and disables analog in;
199  * sets initial port outputs to HIGH
200  */
201 void keypadSetup() {
202     // initialize pins 4-7 HIGH
203     PORTDbits.RD2 = 1;
204     PORTDbits.RD3 = 1;
205     PORTDbits.RD4 = 1;
206     PORTDbits.RD5 = 1;
207
208     // pins 4-7 are toggled, pins 0-3 are monitored
209     TRISDbits.RD5 = 0;
210     TRISDbits.RD4 = 0;
211     TRISDbits.RD3 = 0;
212     TRISDbits.RD2 = 0;
213
214     TRISBbits.RB0 = 1;
215     TRISBbits.RB1 = 1;
216     TRISBbits.RB2 = 1;
217     TRISBbits.RB3 = 1;
218
219     ANSELBbits.ANSB3 = 0; // disable analog input
220     ANSELBbits.ANSB2 = 0;
221     ANSELBbits.ANSB1 = 0;
222     ANSELBbits.ANSB0 = 0;
223     ANSELDbits.ANSD2 = 0;
224     ANSELDbits.ANSD3 = 0;
225     ANSELDbits.ANSD4 = 0;
226     ANSELDbits.ANSD5 = 0;
227
228
229     // enable weak pull ups on ports b0-b3
230     WPUB = WPUB & 0b11111111;
231     // enable pull ups on portB globally
232     INTCON2 = INTCON2 & 0b01111111;
233
234     return;
235 }
```

H.6 LCD Driver

../FrontEnd.X/LCD.h

```
1 /*
2  * File:   LCD.h
3  * Author: castia
4  *
```

```
 * Created on May 25, 2014, 7:53 PM
 */

#ifndef LCD_H
#define LCD_H

#ifdef __cplusplus
extern "C" {
#endif

#include "globals.h"

#define V 128
#define H 160

#define RGB565(r,g,b) ((((r>>3)<<11) | (g>>3) | ((b>>2)<<5)))

#define ST7735_NOP      0x00
#define ST7735_SWRESET  0x01
#define ST7735_RDDID    0x04
#define ST7735_RDDST    0x09

#define ST7735_SLPIN    0x10
#define ST7735_SLPOUT   0x11
#define ST7735_PTLON    0x12
#define ST7735_NORON    0x13

#define ST7735_INVOFF   0x20
#define ST7735_INVON    0x21
#define ST7735_DISPOFF  0x28
#define ST7735_DISPON   0x29
#define ST7735_CASET    0x2A
#define ST7735_RASET    0x2B
#define ST7735_RAMWR    0x2C
#define ST7735_RAMRD    0x2E

#define ST7735_PTLAR    0x30
#define ST7735_COLMOD   0x3A
#define ST7735_MADCTL   0x36

#define ST7735_FRMCTR1  0xB1
#define ST7735_FRMCTR2  0xB2
#define ST7735_FRMCTR3  0xB3
    #define ST7735_INVCTR   0xB4
#define ST7735_DISSET5  0xB6

#define ST7735_PWCTR1   0xC0
#define ST7735_PWCTR2   0xC1
#define ST7735_PWCTR3   0xC2
```

```
#define ST7735_PWCTR4   0xC3
#define ST7735_PWCTR5   0xC4
#define ST7735_VMCTR1   0xC5

#define ST7735_RDID1    0xDA
#define ST7735_RDID2    0xDB
#define ST7735_RDID3    0xDC
#define ST7735_RDID4    0xDD

#define ST7735_PWCTR6   0xFC

#define ST7735_GMCTRP1 0xE0
#define ST7735_GMCTRN1 0xE1

// Color definitions
#define BLACK RGB565(0,0,0)
#define RED RGB565(255,0,0)
#define GREEN RGB565(0,255,0)
#define BLUE RGB565(0,0,255)
#define YELLOW RGB565(255,255,0)
#define CYAN RGB565(0,255,255)
#define PINK RGB565(255,0,255)
#define GRAY RGB565(192,192,192)
#define WHITE RGB565(255,255,255)


#define CS 0b00010000 //Low = select, High = deselect.
#define RE 0b00000010 //High = normal, Low = reset.
#define A0 0b00010000 //Low = Command, High = Data.

void delay(int x);
void sendcomand(char input);
void senddata(char input);
void SetPix(char x, char y, int color);
void clean(int color);
void initLCD(void);
void ASCII(char x, char y, int color, int background, char letter, char size);


int customColor(int r, int g, int b);
void drawBoxFill(char x, char y,char height, char width, int color);
void drawBox(char x, char y, char height, char width, int border, int color);

typedef struct globaldata GlobalState;

// Print Functions
void prints(char x, char y, int color, int background, const char message[],
    char size);
```

```
101 void printrs(char x, char y, int color, int background, char* message, char
        size);
102 void integerprint(char x, char y, int color, int background,int integer, char
        size);
103 void printMenu(char** select, int background, int box, int boxBorder, int text,
         int size);
104 void processPrintCursor(GlobalState* globalData, int size, int background, int
        text);
105
106
107 // System Specific functions
108 void mainMenu(GlobalState* globalData);
109 void selectGameMenu(GlobalState* globalData);
110
111 // System Specific Print Functions
112 void printBSOD(void);
113 void printMainMenu(GlobalState* globalData);
114 void printSelectGame(GlobalState* globalData);
115 void printBuild(GlobalState *globalData);
116
117 // Test Purposes
118 void printBuildCard1(GlobalState *globalData);
119
120 // depreciated
121 //void box(char x, char y, char high, char breth, int color);
122 //void processDisplay(GlobalState* globalData);
123 //void nextPage(GlobalState* globalData, int cursorPos);
124
125 #ifdef __cplusplus
126 }
127 #endif
128
129 #endif /* LCD_H */
```

../FrontEnd.X/LCD.c

```
1
2 #include "globals.h"
3 #include "LCD.h"
4
5 const rom char font[255][5] = {
6     {0x00, 0x00, 0x00, 0x00, 0x00},
7     {0x3E, 0x5B, 0x4F, 0x5B, 0x3E},
8     {0x3E, 0x6B, 0x4F, 0x6B, 0x3E},
9     {0x1C, 0x3E, 0x7C, 0x3E, 0x1C},
10    {0x18, 0x3C, 0x7E, 0x3C, 0x18},
11    {0x1C, 0x57, 0x7D, 0x57, 0x1C},
12    {0x1C, 0x5E, 0x7F, 0x5E, 0x1C},
13    {0x00, 0x18, 0x3C, 0x18, 0x00},
```

```
    {0xFF, 0xE7, 0xC3, 0xE7, 0xFF},
    {0x00, 0x18, 0x24, 0x18, 0x00},
    {0xFF, 0xE7, 0xDB, 0xE7, 0xFF},
    {0x30, 0x48, 0x3A, 0x06, 0x0E},
    {0x26, 0x29, 0x79, 0x29, 0x26},
    {0x40, 0x7F, 0x05, 0x05, 0x07},
    {0x40, 0x7F, 0x05, 0x25, 0x3F},
    {0x5A, 0x3C, 0xE7, 0x3C, 0x5A},
    {0x7F, 0x3E, 0x1C, 0x1C, 0x08},
    {0x08, 0x1C, 0x1C, 0x3E, 0x7F},
    {0x14, 0x22, 0x7F, 0x22, 0x14},
    {0x5F, 0x5F, 0x00, 0x5F, 0x5F},
    {0x06, 0x09, 0x7F, 0x01, 0x7F},
    {0x00, 0x66, 0x89, 0x95, 0x6A},
    {0x60, 0x60, 0x60, 0x60, 0x60},
    {0x94, 0xA2, 0xFF, 0xA2, 0x94},
    {0x08, 0x04, 0x7E, 0x04, 0x08},
    {0x10, 0x20, 0x7E, 0x20, 0x10},
    {0x08, 0x08, 0x2A, 0x1C, 0x08},
    {0x08, 0x1C, 0x2A, 0x08, 0x08},
    {0x1E, 0x10, 0x10, 0x10, 0x10},
    {0x0C, 0x1E, 0x0C, 0x1E, 0x0C},
    {0x30, 0x38, 0x3E, 0x38, 0x30},
    {0x06, 0x0E, 0x3E, 0x0E, 0x06},
    {0x00, 0x00, 0x00, 0x00, 0x00},
    {0x00, 0x00, 0x5F, 0x00, 0x00},
    {0x00, 0x07, 0x00, 0x07, 0x00},
    {0x14, 0x7F, 0x14, 0x7F, 0x14},
    {0x24, 0x2A, 0x7F, 0x2A, 0x12},
    {0x23, 0x13, 0x08, 0x64, 0x62},
    {0x36, 0x49, 0x56, 0x20, 0x50},
    {0x00, 0x08, 0x07, 0x03, 0x00},
    {0x00, 0x1C, 0x22, 0x41, 0x00},
    {0x00, 0x41, 0x22, 0x1C, 0x00},
    {0x2A, 0x1C, 0x7F, 0x1C, 0x2A},
    {0x08, 0x08, 0x3E, 0x08, 0x08},
    {0x00, 0x80, 0x70, 0x30, 0x00},
    {0x08, 0x08, 0x08, 0x08, 0x08},
    {0x00, 0x00, 0x60, 0x60, 0x00},
    {0x20, 0x10, 0x08, 0x04, 0x02},
    {0x3E, 0x51, 0x49, 0x45, 0x3E},
    {0x00, 0x42, 0x7F, 0x40, 0x00},
    {0x72, 0x49, 0x49, 0x49, 0x46},
    {0x21, 0x41, 0x49, 0x4D, 0x33},
    {0x18, 0x14, 0x12, 0x7F, 0x10},
    {0x27, 0x45, 0x45, 0x45, 0x39},
    {0x3C, 0x4A, 0x49, 0x49, 0x31},
    {0x41, 0x21, 0x11, 0x09, 0x07},
    {0x36, 0x49, 0x49, 0x49, 0x36},
```

```
    {0x46, 0x49, 0x49, 0x29, 0x1E},
    {0x00, 0x00, 0x14, 0x00, 0x00},
    {0x00, 0x40, 0x34, 0x00, 0x00},
    {0x00, 0x08, 0x14, 0x22, 0x41},
    {0x14, 0x14, 0x14, 0x14, 0x14},
    {0x00, 0x41, 0x22, 0x14, 0x08},
    {0x02, 0x01, 0x59, 0x09, 0x06},
    {0x3E, 0x41, 0x5D, 0x59, 0x4E},
    {0x7C, 0x12, 0x11, 0x12, 0x7C},
    {0x7F, 0x49, 0x49, 0x49, 0x36},
    {0x3E, 0x41, 0x41, 0x41, 0x22},
    {0x7F, 0x41, 0x41, 0x41, 0x3E},
    {0x7F, 0x49, 0x49, 0x49, 0x41},
    {0x7F, 0x09, 0x09, 0x09, 0x01},
    {0x3E, 0x41, 0x41, 0x51, 0x73},
    {0x7F, 0x08, 0x08, 0x08, 0x7F},
    {0x00, 0x41, 0x7F, 0x41, 0x00},
    {0x20, 0x40, 0x41, 0x3F, 0x01},
    {0x7F, 0x08, 0x14, 0x22, 0x41},
    {0x7F, 0x40, 0x40, 0x40, 0x40},
    {0x7F, 0x02, 0x1C, 0x02, 0x7F},
    {0x7F, 0x04, 0x08, 0x10, 0x7F},
    {0x3E, 0x41, 0x41, 0x41, 0x3E},
    {0x7F, 0x09, 0x09, 0x09, 0x06},
    {0x3E, 0x41, 0x51, 0x21, 0x5E},
    {0x7F, 0x09, 0x19, 0x29, 0x46},
    {0x26, 0x49, 0x49, 0x49, 0x32},
    {0x03, 0x01, 0x7F, 0x01, 0x03},
    {0x3F, 0x40, 0x40, 0x40, 0x3F},
    {0x1F, 0x20, 0x40, 0x20, 0x1F},
    {0x3F, 0x40, 0x38, 0x40, 0x3F},
    {0x63, 0x14, 0x08, 0x14, 0x63},
    {0x03, 0x04, 0x78, 0x04, 0x03},
    {0x61, 0x59, 0x49, 0x4D, 0x43},
    {0x00, 0x7F, 0x41, 0x41, 0x41},
    {0x02, 0x04, 0x08, 0x10, 0x20},
    {0x00, 0x41, 0x41, 0x41, 0x7F},
    {0x04, 0x02, 0x01, 0x02, 0x04},
    {0x40, 0x40, 0x40, 0x40, 0x40},
    {0x00, 0x03, 0x07, 0x08, 0x00},
    {0x20, 0x54, 0x54, 0x78, 0x40},
    {0x7F, 0x28, 0x44, 0x44, 0x38},
    {0x38, 0x44, 0x44, 0x44, 0x28},
    {0x38, 0x44, 0x44, 0x28, 0x7F},
    {0x38, 0x54, 0x54, 0x54, 0x18},
    {0x00, 0x08, 0x7E, 0x09, 0x02},
    {0x18, 0xA4, 0xA4, 0x9C, 0x78},
    {0x7F, 0x08, 0x04, 0x04, 0x78},
    {0x00, 0x44, 0x7D, 0x40, 0x00},
```

```
    {0x20, 0x40, 0x40, 0x3D, 0x00},
    {0x7F, 0x10, 0x28, 0x44, 0x00},
    {0x00, 0x41, 0x7F, 0x40, 0x00},
    {0x7C, 0x04, 0x78, 0x04, 0x78},
    {0x7C, 0x08, 0x04, 0x04, 0x78},
    {0x38, 0x44, 0x44, 0x44, 0x38},
    {0xFC, 0x18, 0x24, 0x24, 0x18},
    {0x18, 0x24, 0x24, 0x18, 0xFC},
    {0x7C, 0x08, 0x04, 0x04, 0x08},
    {0x48, 0x54, 0x54, 0x54, 0x24},
    {0x04, 0x04, 0x3F, 0x44, 0x24},
    {0x3C, 0x40, 0x40, 0x20, 0x7C},
    {0x1C, 0x20, 0x40, 0x20, 0x1C},
    {0x3C, 0x40, 0x30, 0x40, 0x3C},
    {0x44, 0x28, 0x10, 0x28, 0x44},
    {0x4C, 0x90, 0x90, 0x90, 0x7C},
    {0x44, 0x64, 0x54, 0x4C, 0x44},
    {0x00, 0x08, 0x36, 0x41, 0x00},
    {0x00, 0x00, 0x77, 0x00, 0x00},
    {0x00, 0x41, 0x36, 0x08, 0x00},
    {0x02, 0x01, 0x02, 0x04, 0x02},
    {0x3C, 0x26, 0x23, 0x26, 0x3C},
    {0x1E, 0xA1, 0xA1, 0x61, 0x12},
    {0x3A, 0x40, 0x40, 0x20, 0x7A},
    {0x38, 0x54, 0x54, 0x55, 0x59},
    {0x21, 0x55, 0x55, 0x79, 0x41},
    {0x21, 0x54, 0x54, 0x78, 0x41},
    {0x21, 0x55, 0x54, 0x78, 0x40},
    {0x20, 0x54, 0x55, 0x79, 0x40},
    {0x0C, 0x1E, 0x52, 0x72, 0x12},
    {0x39, 0x55, 0x55, 0x55, 0x59},
    {0x39, 0x54, 0x54, 0x54, 0x59},
    {0x39, 0x55, 0x54, 0x54, 0x58},
    {0x00, 0x00, 0x45, 0x7C, 0x41},
    {0x00, 0x02, 0x45, 0x7D, 0x42},
    {0x00, 0x01, 0x45, 0x7C, 0x40},
    {0xF0, 0x29, 0x24, 0x29, 0xF0},
    {0xF0, 0x28, 0x25, 0x28, 0xF0},
    {0x7C, 0x54, 0x55, 0x45, 0x00},
    {0x20, 0x54, 0x54, 0x7C, 0x54},
    {0x7C, 0x0A, 0x09, 0x7F, 0x49},
    {0x32, 0x49, 0x49, 0x49, 0x32},
    {0x32, 0x48, 0x48, 0x48, 0x32},
    {0x32, 0x4A, 0x48, 0x48, 0x30},
    {0x3A, 0x41, 0x41, 0x21, 0x7A},
    {0x3A, 0x42, 0x40, 0x20, 0x78},
    {0x00, 0x9D, 0xA0, 0xA0, 0x7D},
    {0x39, 0x44, 0x44, 0x44, 0x39},
    {0x3D, 0x40, 0x40, 0x40, 0x3D},
```

```
161     {0x3C, 0x24, 0xFF, 0x24, 0x24},
162     {0x48, 0x7E, 0x49, 0x43, 0x66},
163     {0x2B, 0x2F, 0xFC, 0x2F, 0x2B},
164     {0xFF, 0x09, 0x29, 0xF6, 0x20},
165     {0xC0, 0x88, 0x7E, 0x09, 0x03},
166     {0x20, 0x54, 0x54, 0x79, 0x41},
167     {0x00, 0x00, 0x44, 0x7D, 0x41},
168     {0x30, 0x48, 0x48, 0x4A, 0x32},
169     {0x38, 0x40, 0x40, 0x22, 0x7A},
170     {0x00, 0x7A, 0x0A, 0x0A, 0x72},
171     {0x7D, 0x0D, 0x19, 0x31, 0x7D},
172     {0x26, 0x29, 0x29, 0x2F, 0x28},
173     {0x26, 0x29, 0x29, 0x29, 0x26},
174     {0x30, 0x48, 0x4D, 0x40, 0x20},
175     {0x38, 0x08, 0x08, 0x08, 0x08},
176     {0x08, 0x08, 0x08, 0x08, 0x38},
177     {0x2F, 0x10, 0xC8, 0xAC, 0xBA},
178     {0x2F, 0x10, 0x28, 0x34, 0xFA},
179     {0x00, 0x00, 0x7B, 0x00, 0x00},
180     {0x08, 0x14, 0x2A, 0x14, 0x22},
181     {0x22, 0x14, 0x2A, 0x14, 0x08},
182     {0xAA, 0x00, 0x55, 0x00, 0xAA},
183     {0xAA, 0x55, 0xAA, 0x55, 0xAA},
184     {0x00, 0x00, 0x00, 0xFF, 0x00},
185     {0x10, 0x10, 0x10, 0xFF, 0x00},
186     {0x14, 0x14, 0x14, 0xFF, 0x00},
187     {0x10, 0x10, 0xFF, 0x00, 0xFF},
188     {0x10, 0x10, 0xF0, 0x10, 0xF0},
189     {0x14, 0x14, 0x14, 0xFC, 0x00},
190     {0x14, 0x14, 0xF7, 0x00, 0xFF},
191     {0x00, 0x00, 0xFF, 0x00, 0xFF},
192     {0x14, 0x14, 0xF4, 0x04, 0xFC},
193     {0x14, 0x14, 0x17, 0x10, 0x1F},
194     {0x10, 0x10, 0x1F, 0x10, 0x1F},
195     {0x14, 0x14, 0x14, 0x1F, 0x00},
196     {0x10, 0x10, 0x10, 0xF0, 0x00},
197     {0x00, 0x00, 0x00, 0x1F, 0x10},
198     {0x10, 0x10, 0x10, 0x1F, 0x10},
199     {0x10, 0x10, 0x10, 0xF0, 0x10},
200     {0x00, 0x00, 0x00, 0xFF, 0x10},
201     {0x10, 0x10, 0x10, 0x10, 0x10},
202     {0x10, 0x10, 0x10, 0xFF, 0x10},
203     {0x00, 0x00, 0x00, 0xFF, 0x14},
204     {0x00, 0x00, 0xFF, 0x00, 0xFF},
205     {0x00, 0x00, 0x1F, 0x10, 0x17},
206     {0x00, 0x00, 0xFC, 0x04, 0xF4},
207     {0x14, 0x14, 0x17, 0x10, 0x17},
208     {0x14, 0x14, 0xF4, 0x04, 0xF4},
209     {0x00, 0x00, 0xFF, 0x00, 0xF7},
```

```
210     {0x14, 0x14, 0x14, 0x14, 0x14},
211     {0x14, 0x14, 0xF7, 0x00, 0xF7},
212     {0x14, 0x14, 0x14, 0x17, 0x14},
213     {0x10, 0x10, 0x1F, 0x10, 0x1F},
214     {0x14, 0x14, 0x14, 0xF4, 0x14},
215     {0x10, 0x10, 0xF0, 0x10, 0xF0},
216     {0x00, 0x00, 0x1F, 0x10, 0x1F},
217     {0x00, 0x00, 0x00, 0x1F, 0x14},
218     {0x00, 0x00, 0x00, 0xFC, 0x14},
219     {0x00, 0x00, 0xF0, 0x10, 0xF0},
220     {0x10, 0x10, 0xFF, 0x10, 0xFF},
221     {0x14, 0x14, 0x14, 0xFF, 0x14},
222     {0x10, 0x10, 0x10, 0x1F, 0x00},
223     {0x00, 0x00, 0x00, 0xF0, 0x10},
224     {0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
225     {0xF0, 0xF0, 0xF0, 0xF0, 0xF0},
226     {0xFF, 0xFF, 0xFF, 0x00, 0x00},
227     {0x00, 0x00, 0x00, 0xFF, 0xFF},
228     {0x0F, 0x0F, 0x0F, 0x0F, 0x0F},
229     {0x38, 0x44, 0x44, 0x38, 0x44},
230     {0x7C, 0x2A, 0x2A, 0x3E, 0x14},
231     {0x7E, 0x02, 0x02, 0x06, 0x06},
232     {0x02, 0x7E, 0x02, 0x7E, 0x02},
233     {0x63, 0x55, 0x49, 0x41, 0x63},
234     {0x38, 0x44, 0x44, 0x3C, 0x04},
235     {0x40, 0x7E, 0x20, 0x1E, 0x20},
236     {0x06, 0x02, 0x7E, 0x02, 0x02},
237     {0x99, 0xA5, 0xE7, 0xA5, 0x99},
238     {0x1C, 0x2A, 0x49, 0x2A, 0x1C},
239     {0x4C, 0x72, 0x01, 0x72, 0x4C},
240     {0x30, 0x4A, 0x4D, 0x4D, 0x30},
241     {0x30, 0x48, 0x78, 0x48, 0x30},
242     {0xBC, 0x62, 0x5A, 0x46, 0x3D},
243     {0x3E, 0x49, 0x49, 0x49, 0x00},
244     {0x7E, 0x01, 0x01, 0x01, 0x7E},
245     {0x2A, 0x2A, 0x2A, 0x2A, 0x2A},
246     {0x44, 0x44, 0x5F, 0x44, 0x44},
247     {0x40, 0x51, 0x4A, 0x44, 0x40},
248     {0x40, 0x44, 0x4A, 0x51, 0x40},
249     {0x00, 0x00, 0xFF, 0x01, 0x03},
250     {0xE0, 0x80, 0xFF, 0x00, 0x00},
251     {0x08, 0x08, 0x6B, 0x6B, 0x08},
252     {0x36, 0x12, 0x36, 0x24, 0x36},
253     {0x06, 0x0F, 0x09, 0x0F, 0x06},
254     {0x00, 0x00, 0x18, 0x18, 0x00},
255     {0x00, 0x00, 0x10, 0x10, 0x00},
256     {0x30, 0x40, 0xFF, 0x01, 0x01},
257     {0x00, 0x1F, 0x01, 0x01, 0x1E},
258     {0x00, 0x19, 0x1D, 0x17, 0x12},
```

```
259     {0x00, 0x3C, 0x3C, 0x3C, 0x3C},
260     {0x00, 0x00, 0x00, 0x00, 0x00}
261 };
262 
263 void delay(int x) {
264     int j = 0;
265     for (j = 0; j < x; j++);
266 }
267 
268 // Sends "command" to the LCD
269 
270 void sendcomand(char input) {
271     int j = 0;
272     PORTC &= ~A0;
273     PORTB &= ~CS;
274     SSP1BUF = input;
275     for (j = 0; j < 1; j++);
276     PORTB |= CS;
277 }
278 
279 // Sends "data" to the LCD
280 
281 void senddata(char input) {
282     int j = 0;
283     PORTC |= A0;
284     PORTB &= ~CS;
285     SSP1BUF = input;
286     for (j = 0; j < 1; j++);
287     PORTB |= CS;
288 }
289 
290 // Sets individual pixels at coords x and y  to the given color
291 
292 void SetPix(char x, char y, int color) {
293     char Hig = 0;
294     char Low = color & 0x00ff;
295     color >>= 8;
296     Hig = color;
297 
298     sendcomand(ST7735_CASET); // Column addr set
299     senddata(0x00);
300     senddata(x); // XSTART
301     senddata(0x00);
302     senddata(x + 1); // XEND
303 
304     sendcomand(ST7735_RASET); // Row addr set
305     senddata(0x00);
306     senddata(y); // YSTART
307     senddata(0x00);
```

```
308     senddata(y + 1); // YEND
309
310     sendcomand(ST7735_RAMWR);
311
312     senddata(Low);
313     senddata(Hig);
314 }
315
316
317
318 // Returns a custom color in RGB values
319
320 int customColor(int r, int g, int b) {
321     return RGB565(r, g, b);
322 }
323
324 // Draw a border in a box shape at upper left x, y coords
325 // Box will be height tall, width wide, have a border of the given size and
    color
326 // of the given color
327
328 void drawBox(char x, char y, char height, char width, int border, int color) {
329     border -= 1;
330     drawBoxFill(x, y, border, width, color);
331     drawBoxFill(x + width - border, y, height, border, color);
332     drawBoxFill(x, y + height - border, border, width, color);
333     drawBoxFill(x, y, height, border, color);
334 }
335
336 // Draws a box and fills it in with the given color
337
338 void drawBoxFill(char x, char y, char height, char width, int color) {
339     char Hig = 0;
340     char Low = color & 0x00ff;
341     int i = 0;
342     int j = 0;
343     Hig = color >> 8;
344
345     sendcomand(ST7735_CASET); // Column addr set
346     senddata(0x00);
347     senddata(x); // XSTART
348     senddata(0x00);
349     senddata(x + width); // XEND
350
351     sendcomand(ST7735_RASET); // Row addr set
352     senddata(0x00);
353     senddata(y); // YSTART
354     senddata(0x00);
355     senddata(y + height); // YEND
```

```
356
357     sendcomand(ST7735_RAMWR);
358
359     for (j = 0; j <= width; j++) {
360         for (i = 0; i <= height; i++) {
361             senddata(Low);
362             senddata(Hig);
363         }
364     }
365 }
366
367 // Cleans the entire screen as a certain color
368
369 void clean(int color) {
370
371     char Hig = 0;
372     int x = 0;
373     int y = 0;
374     char Low = color & 0x00ff;
375     color >>= 8;
376     Hig = color;
377
378     sendcomand(ST7735_CASET); // Column addr set
379     senddata(0x00);
380     senddata(0); // XSTART
381     senddata(0x00);
382     senddata(V); // XEND
383
384     sendcomand(ST7735_RASET); // Row addr set
385     senddata(0x00);
386     senddata(0); // YSTART
387     senddata(0x00);
388     senddata(H); // YEND
389
390     sendcomand(ST7735_RAMWR);
391
392     for (x = 0; x < V; x++) {
393         for (y = 0; y < H; y++) {
394             senddata(Low);
395             senddata(Hig);
396         }
397     }
398 }
399
400 // Initializes the LCD screen by sending a bajillion commands
401
402 void initLCD() {
403     PORTB &= ~CS;
404     delay(0);
```

```
405     //     PORTC |= RE;
406     //     delay(100);
407     //     PORTC &= ~RE;
408     //     delay(100);
409     //     PORTC |= RE;
410     delay(10000);
411     sendcomand(ST7735_SWRESET); //  1: Software reset, 0 args, w/delay 150ms 0
        x01
412     delay(1000);
413     sendcomand(ST7735_SLPOUT); //  2: Out of sleep mode, 0 args, w/delay 500ms
        0x11
414     delay(10000);
415     sendcomand(ST7735_FRMCTR1); //  3: Frame rate ctrl - normal mode) 3 args: 0
        xb1
416     senddata(0x01);
417     senddata(0x2C);
418     senddata(0x2D); //      Rate = fosc/(1x2+40) * (LINE+2C+2D)
419     sendcomand(ST7735_FRMCTR2); //  4: Frame rate control - idle mode) 3 args:
        0xb2
420     senddata(0x01);
421     senddata(0x2C);
422     senddata(0x2D); //      Rate = fosc/(1x2+40) * (LINE+2C+2D)
423     sendcomand(ST7735_FRMCTR3); //  5: Frame rate ctrl - partial mode) 6 args:
        0xb3
424     senddata(0x01);
425     senddata(0x2C);
426     senddata(0x2D); //      Dot inversion mode
427     senddata(0x01);
428     senddata(0x2C);
429     senddata(0x2D); //      Line inversion mode
430     sendcomand(ST7735_INVCTR); //  6: Display inversion ctrl) 1 arg) no delay:
        0xb4
431     senddata(0x07); //      No inversion
432     sendcomand(ST7735_PWCTR1); //  7: Power control) 3 args) no delay: 0xc0
433     senddata(0xA2);
434     senddata(0x02); //      -4.6V
435     senddata(0x84); //      AUTO mode
436     sendcomand(ST7735_PWCTR2); //  8: Power control) 1 arg) no delay: 0xc1
437     senddata(0xC5); //      VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
438     sendcomand(ST7735_PWCTR3); //  9: Power control) 2 args) no delay: 0xc2
439     senddata(0x0A); //      Opamp current small
440     senddata(0x00); //      Boost frequency
441     sendcomand(ST7735_PWCTR4); // 10: Power control) 2 args) no delay:
442     senddata(0x8A); //      BCLK/2) Opamp current small & Medium low
443     senddata(0x2A);
444     sendcomand(ST7735_PWCTR5); // 11: Power control) 2 args) no delay:
445     senddata(0x8A);
446     senddata(0xEE);
447     sendcomand(ST7735_VMCTR1); // 12: Power control) 1 arg) no delay:
```

```
448     senddata(0x0E);
449     sendcomand(ST7735_INVOFF); // 13: Don't invert display) no args) no delay 0
        x20
450     sendcomand(ST7735_MADCTL); // 14: Memory access control (directions)) 1 arg
        :
451     senddata(0xC8); //      row addr/col addr); bottom to top refresh
452     sendcomand(ST7735_COLMOD); // 15: set color mode); 1 arg); no delay:
453     senddata(0x05); //      16-bit color
454     sendcomand(ST7735_GMCTRP1); // Gamma correction
455     senddata(0x0f);
456     senddata(0x1a);
457     senddata(0x0f);
458     senddata(0x18);
459     senddata(0x2f);
460     senddata(0x28);
461     senddata(0x20);
462     senddata(0x22);
463     senddata(0x1f);
464     senddata(0x1b);
465     senddata(0x23);
466     senddata(0x37);
467     senddata(0x00);
468     senddata(0x07);
469     senddata(0x02);
470     senddata(0x10);
471     sendcomand(ST7735_GMCTRN1);
472     senddata(0x0f);
473     senddata(0x1b);
474     senddata(0x0f);
475     senddata(0x17);
476     senddata(0x33);
477     senddata(0x2c);
478     senddata(0x29);
479     senddata(0x2e);
480     senddata(0x30);
481     senddata(0x30);
482     senddata(0x39);
483     senddata(0x3f);
484     senddata(0x00);
485     senddata(0x07);
486     senddata(0x03);
487     senddata(0x10);
488
489     sendcomand(ST7735_NORON); //  3: Normal display on, no args, w/delay 10ms 0
        x13
490     delay(100);
491     sendcomand(ST7735_DISPON); //  4: Main screen turn on, no args w/delay 100
        ms 0x29
492     delay(1000);
```

```
493 }
494
495 // Looks up the pixels to set when writing ascii character
496
497 void ASCII(char x, char y, int color, int background, char letter, char size) {
498     char data;
499     char q = 0;
500     char z = 0;
501     char d = 0;
502     char b = 0;
503
504     for (q = 0; q < 5; q++) {
505         data = font[letter][q];
506         for (z = 0; z < 8 * size; z++) {
507             if ((data & 1) != 0) {
508                 for (d = 0; d < size; d++) {
509                     for (b = 0; b < size; b++) {
510                         SetPix(x + (q * size) + d, y + (z * size) + b, color);
511                     }
512                 }
513             } else {
514                 for (d = 0; d < size; d++) {
515                     for (b = 0; b < size; b++) {
516                         SetPix(x + (q * size) + d, y + (z * size) + b,
    background);
517                     }
518                 }
519             }
520             data >>= 1;
521         }
522     }
523 }
524
525 // Prints a string that is sent in " " marks
526 // It will start at coords x and y, with background colors and text colors
527 // Font size can be specified.
528
529 // Each line with size 1 should be around 8 pixels apart, font will wrap around
530 // the screen if it's too long
531
532 void prints(char x, char y, int color, int background, const char messageOld[],
      char size) {
533     const far rom char* message = (const far rom char*) messageOld;
534     while (*message) {
535         ASCII(x, y, color, background, *message++, size);
536         x += 6 * size;
537         if (x > 120) {
538             x = 0;
539             y += 8 * size;
```

```
540         }
541     }
542 }
543
544 // Prints a string saved in ram already
545 // It will start at coords x and y, with background colors and text colors
546 // Font size can be specified.
547
548 // Each line with size 1 should be around 8 pixels apart, font will wrap around
549 // the screen if it's too long
550
551 void printrs(char x, char y, int color, int background, char* message, char
    size) {
552     while (*message) {
553         ASCII(x, y, color, background, *message++, size);
554         x += 6 * size;
555         if (x > 120) {
556             x = 0;
557             y += 8 * size;
558         }
559     }
560 }
561
562 // Prints an INT value to the screen
563
564 void integerprint(char x, char y, int color, int background, int integer, char
    size) {
565     unsigned char tenthousands = 0;
566     unsigned char thousands = 0;
567     unsigned char hundreds = 0;
568     unsigned char tens = 0;
569     unsigned char ones = 0;
570     if (integer >= 10000) {
571         tenthousands = integer / 10000;
572         ASCII(x, y, color, background, tenthousands + 48, size);
573     }
574     if (integer >= 1000) {
575         thousands = ((integer - tenthousands * 10000)) / 1000;
576         x += 6;
577         ASCII(x, y, color, background, thousands + 48, size);
578     }
579     if (integer >= 100) {
580         hundreds = (((integer - tenthousands * 10000) - thousands * 1000)) /
    100;
581         x += 6;
582         ASCII(x, y, color, background, hundreds + 48, size);
583     }
584     if (integer >= 10) {
585         tens = (integer % 100) / 10;
```

```
586         x += 6;
587         ASCII(x, y, color, background, tens + 48, size);
588     }
589     ones = integer % 10;
590     x += 6;
591     ASCII(x, y, color, background, ones + 48, size);
592 }
593
594 // Prints menu for operating system functions
595 void printMenu(char** select, int background, int box, int boxBorder, int text,
        int size) {
596     int i;
597     // Beep off
598     TRISBbits.RB5 = 1;
599     // Change background
600     clean(background);
601
602     // Draw Box
603     drawBoxFill(0, 0, 20, V - 1, box);
604     drawBox(0, 0, 20, V - 1, 2, boxBorder);
605
606     // Write Select Options
607     for (i = 0; i < size; i++) {
608         // Do I have to deference this pointer?
609         prints(35, 20 * i + 40, text, background, select[i], 1);
610     }
611 }
612
613 // Prints cursor and processes the selection based off of the index i
614 // Top of List [ 0 1 2 3] Bottom of List
615
616 void processPrintCursor(GlobalState* globalData, int size, int background, int
        text) {
617
618     switch (globalData->keyPress) {
619             // Move up = Press 2
620         case 0x02:
621             // Clear original cursor
622             prints(25, 20 * globalData->cursorPos + 40, text, background, " ",
        1);
623
624             // Find new position of cursor
625             globalData->cursorPos = ((size + globalData->cursorPos) - 1) % size
        ;
626
627             // Print cursor in new position
628             prints(25, 20 * globalData->cursorPos + 40, text, background, ">",
        1);
629             break;
```

```
630             // Move down = Press 8
631         case 0x08:
632             // Clear original cursor
633             prints(25, 20 * globalData->cursorPos + 40, text, background, " ",
    1);
634
635             // Find new position of cursor
636             globalData->cursorPos = (globalData->cursorPos + 1) % size;
637
638             // Print cursor in new position
639             prints(25, 20 * globalData->cursorPos + 40, text, background, ">",
    1);
640             break;
641             // Hit back button "B"
642         case 0x0B:
643             globalData->cursorPos = 0xFF;
644             globalData->newDisplay = 1;
645             break;
646         case 0x0D:
647             prints(25, 20 * globalData->cursorPos + 40, text, background, ">>>"
    , 1);
648             globalData->newDisplay = 1;
649             break;
650         default:
651             break;
652     }
653 }
654
655
656 /* The following functions are print functions for the operating system of the
657  * device.
658  */
659
660 // Prints the main menu
661 void printMainMenu(GlobalState* globalData) {
662     // LCD menu
663     // Main Menu Array
664     const rom char *mainMenu[3] = {"Singleplayer", "Multiplayer", "Build Card"
    };
665     printMenu(mainMenu, BLUE, CYAN, WHITE, WHITE, 3);
666     prints(35, 7, WHITE, CYAN, "Main Menu", 1);
667     prints(25, 40, WHITE, BLUE, ">", 1);
668
669     /*
670     clean(BLUE);
671     drawBoxFill(0, 0, 20, V - 1, CYAN);
672     drawBox(0, 0, 20, V - 1, 2, WHITE);
673
674     prints(35, globalData->mainMenuSpots[0], WHITE, BLUE, "Single Player", 1);
```

```
675     prints(35, globalData->mainMenuSpots[1], WHITE, BLUE, "Multiplayer", 1);
676     prints(35, globalData->mainMenuSpots[2], WHITE, BLUE, "Build Cards", 1);
677      */
678     prints(0, H - 8, WHITE, BLUE, "2-UP,8-DOWN,D-ENTER", 1);
679 }
680
681 // Print menu for selecting available games.
682 void printSelectGame(GlobalState *globalData) {
683     char* selectGame[4] = {"Duel Game", "Clue", "Empty", "Empty"};
684
685     printMenu(selectGame, GREEN, YELLOW, BLACK, BLACK, 4);
686     prints(25, 7, BLACK, YELLOW, "Available Games:", 1);
687     prints(25, 40, BLACK, GREEN, ">", 1);
688     prints(0, H - 8, BLACK, YELLOW, "2-UP,8-DOWN,D-ENTER", 1);
689 }
690
691 void printBuild(GlobalState *globalData) {
692     char* selectGame[4] = {"FIREDUDE", "EARTHGUY", "WATERMAN", "DRPECKOL"};
693
694     printMenu(selectGame, RED, BLACK, BLUE, WHITE, 4);
695     prints(25, 7, BLACK, BLUE, "Make a card:", 1);
696     prints(25, 40, BLACK, GREEN, ">", 1);
697     prints(0, H - 8, BLACK, YELLOW, "2-UP,8-DOWN,D-ENTER", 1);
698 }
699
700 // Print blue screen of death
701 void printBSOD() {
702     // Beep off
703     TRISBbits.RB5 = 1;
704     clean(BLUE);
705
706     drawBoxFill(30, 39, 8, 60, GRAY);
707     prints(35, 40, BLUE, GRAY, " Windows ", 1);
708
709     prints(0, 50, WHITE, BLUE, "An error has occurred,", 1);
710     prints(0, 58, WHITE, BLUE, "To continue:", 1);
711     prints(0, 74, WHITE, BLUE, "Remove the battery or", 1);
712     prints(0, 82, WHITE, BLUE, "power supply.", 1);
713     prints(0, 98, WHITE, BLUE, "Error: 0E : BFF9B3D4", 1);
714
715     prints(10, 114, WHITE, BLUE, "Dumping memory...", 1);
716     while (1);
717 }
718
719 // Draws the build card menu, begins an inventory read
720
721 void printBuildCard1(GlobalState *globalData) {
722     // first page of build card
723     clean(RED);
```

```
724     prints(0, 0, BLACK, RED, "It looks like you want to build a card.", 1);
725     prints(0, 16, BLACK, RED, "Available cards:", 1);
726 
727     // Tell system we need to do an inventory command
728     globalData->getInventory = TRUE;
729 }
730 
731 /*
732  * Select Menus for the operating system
733  * Menu goes back to previous menu after being done with a certain process
734  */
735 
736 void mainMenu(GlobalState* globalData) {
737 //     // If returning from a previous menu, re-print main menu
738 //     if(globalData->goBack) {
739 //         printMainMenu(globalData);
740 //         globalData->mode = processPrintCursor(globalData, 3, BLUE, WHITE);
741 //         globalData->goBack = FALSE;
742 //     }
743 //     // Keep checking cursor until they stop hitting back button
744 //     while(globalData->mode == 0xFF) {
745 //         globalData->mode = processPrintCursor(globalData, 3, BLUE, WHITE);
746 //     }
747     if(globalData->goBack) {
748         printMainMenu(globalData);
749         globalData->goBack = FALSE;
750     }
751     processPrintCursor(globalData, 3, BLUE, WHITE);
752     // Switch menus based off of selection
753     switch (globalData->mode) {
754         // Single Player
755         case 0:
756             selectGameMenu(globalData);
757             break;
758         // Multiplayer
759         case 1:
760             selectGameMenu(globalData);
761             break;
762         // Build Card
763         case 2:
764             printBuildCard1(globalData);
765             break;
766     }
767 }
768 
769 // Select game to play based off of the mode (single/multiplayer)
770 void selectGameMenu(GlobalState* globalData) {
771     // Print select game menu
772     printSelectGame(globalData);
```

```
773     processPrintCursor(globalData, 4, GREEN, BLACK);
774     // Run chosen game based off of the multiplayer/single-player mode
775     switch (globalData->game) {
776         // Game Slot 1
777         case 0:
778             if(0 == globalData->mode) {
779                 singlePlayer(globalData);
780             } else {
781                 multiPlayer(globalData);
782             }
783             break;
784         // Game Slot 2
785         case 1:
786             clean(BLACK);
787             prints(0, 35, WHITE, BLACK, "In Progress",1);
788             break;
789         // Game Slot 3
790         case 2:
791             printBSOD();
792             break;
793         // Game Slot 4
794         case 3:
795             clean(BLACK);
796             prints(0, 35, WHITE, BLACK, "In Progress",1);
797             break;
798         // Hit Back Button
799         case 0xFF:
800             globalData->goBack = TRUE;
801             break;
802     }
803
804     globalData->keyPress = -1;
805     // Continue scanning the keypad until the user hits "D" for enter
806     while (globalData->keyPress != 0x0B) {
807         keypad(globalData);
808         if (globalData->keyFlag && !globalData->displayedKey) {
809             globalData->keyFlag = FALSE;
810             globalData->displayedKey = TRUE;
811         }
812     }
813     // Return to main menu after finished.
814     globalData->goBack = TRUE;
815 }
816
817 // Depreciated
818
819 /*
820 // Visuals and navigation for operating system menus
821 void processDisplay(GlobalState* globalData) {
```

```
822     // Different controls for each page being displayed
823     switch (globalData->displayPage) {
824         // Main menu
825         case 0:
826             switch (globalData->keyPress) {
827                 // Press 2 to move up
828                 case 0x02:
829                     // Moves the cursor up 1 space.  Loops around
830                     prints(25, globalData->mainMenuSpots[globalData->cursorPos
    ], WHITE, BLUE, " ", 1);
831                     globalData->cursorPos = ((3 + globalData->cursorPos) - 1) %
     3;
832                     prints(25, globalData->mainMenuSpots[globalData->cursorPos
    ], WHITE, BLUE, ">", 1);
833                     break;
834                 // Press 8 to move down
835                 case 0x08:
836                     // Moves the cursor down 1 space.  Loops around
837                     prints(25, globalData->mainMenuSpots[globalData->cursorPos
    ], WHITE, BLUE, " ", 1);
838                     globalData->cursorPos = (globalData->cursorPos + 1) % 3;
839                     prints(25, globalData->mainMenuSpots[globalData->cursorPos
    ], WHITE, BLUE, ">", 1);
840                     break;
841                 // D is the enter key. Figure out the next page
842                 case 0x0D:
843                     prints(25, globalData->mainMenuSpots[globalData->cursorPos
    ], WHITE, BLUE, ">>>", 1);
844 
845                     // Cursor position determines next page. Add 1 to remove
    main menu "case 0" from the
846                     // list of options when navigating out of the main menu
847                     nextPage(globalData, globalData->cursorPos + 1);
848                     break;
849                 case 0xFF: // Debug BSOD
850                     nextPage(globalData, 255);
851                 default:
852                     break;
853             }
854             break;
855         // Singleplayer
856         case 1:
857             switch (globalData->keyPress) {
858                 // B to go back
859                 case 0x0B:
860                     nextPage(globalData, 0);
861                     break;
862                 default:
863                     break;
```

```
            }
            break;
        // Multiplayer
        case 2:
            switch (globalData->keyPress) {
                // B to go back
                case 0x0B:
                    nextPage(globalData, 0);
                    break;
                default:
                    break;
            }
            break;
        // build cards
        case 3:
            switch (globalData->keyPress) {
                // B to go back
                case 0x0B:
                    nextPage(globalData, 0);
                    break;
                default:
                    break;
            }
            break;
        default:
            printBSOD();
    }
}

void nextPage(GlobalState* globalData, int cursorPos) {
    // Beep off
    TRISBbits.RB5 = 1;

    switch (cursorPos) {
            // Send a 0 to go to main menu
        case 0:
            globalData->displayPage = 0;
            printMainMenu(globalData);
            break;
            // Getting a position of 1 does singleplayer
        case 1:
            globalData->displayPage = 1;
            // Print singleplayer menu
            globalData->displayPage = 1;
            singlePlayer(globalData);
            break;
            // Getting a position of 2 does multiplayer
        case 2:
            globalData->displayPage = 2;
```

```
913             // Print multiplayer menu
914             clean(GREEN);
915             globalData->xbeeFlag = TRUE;
916             prints(0, 0, BLACK, WHITE, "Nothing here. Press B to go back.", 1);
917             break;
918             // Getting a position of 3 does build cards
919         case 3:
920             globalData->displayPage = 3;
921             // Print build cards menu
922             printBuildCard1(globalData);
923             break;
924             // Error
925         default:
926             // BSOD
927             clean(BLUE);
928             globalData->keyFlag = TRUE;
929             globalData->displayedKey = FALSE;
930             globalData->displayPage = 255;
931     }
932 }
933
934 */
```

H.7 Card Reaction Control

../FrontEnd.X/LED.h

```
1  /*
2   * Status LED control header file
3   * June 8, 2014
4   */
5
6  #ifndef LED_H
7  #define LED_H
8
9  #ifdef  __cplusplus
10 extern "C" {
11 #endif
12
13 #define NUM_SLOTS 4 // Number of card slots on board
14
15     typedef struct {
16         char ledStatus[NUM_SLOTS]; // each char represents a slot
17     }LEDDriverStruct;
18
19     extern LEDDriverStruct ledData;
20
21     void LEDSetup(void);
```

```
22      void updateLEDs(void);
23
24  #ifdef  __cplusplus
25  }
26  #endif
27
28  #endif  /* LED_H */
```

../FrontEnd.X/LED.c

```
1   #include "globals.h"
2
3   /*
4    * Prototypes
5    */
6   void LEDSelect(char card, char status);
7   void LEDColor(char status);
8
9
10  /*
11   * Port RB1 = Red
12   * Port RB0 = Green
13   */
14
15  LEDDriverStruct ledData;
16
17  /*
18   * Setup the LED Driver
19   */
20  void LEDSetup(void) {
21      TRISB = 0x30; //0xF0; // set pins 0:3 as outputs, 4:7 as inputs
22      ANSELB = 0x00; // disable analog input
23
24      WPUB |= 0x30; //0xF0; // enable internal pullup on card inputs
25      INTCON2bits.RBPU = 0;
26
27      // initialize ledData
28      memset(ledData.ledStatus, 0x03, sizeof(char) * NUM_SLOTS);
29      updateLEDs();
30
31      PORTB = 0;
32      LATB = PORTB ; // clear existing mismatch conditions
33
34      IOCB = 0x30; //0xF0; // enable IOC interrupts on pins B4:B7
35      INTCONbits.RBIE = 1; // enable PortB interrupts
36      INTCON2bits.RBIP = 1; // set priority level to high
37      INTCONbits.GIE = 1; // enable general purpose interrupts
38
39
```

```
40 }
41
42
43 /*
44  * Primary led driverr
45  */
46 void updateLEDs() {
47     char status = 0;
48     char card = 0;
49     for (card = 0; card < NUM_SLOTS; card++) {
50         status = ledData.ledStatus[card];
51         LEDSelect(card, status);
52     }
53 }
54
55
56 /*
57  * updates the LEDs with the appropriate status
58  */
59 void LEDColor(char status) {
60     switch (status) {
61         case 0:
62             // Yellow: Card registered, but not read.
63             PORTBbits.RB0 = 1;
64             PORTBbits.RB1 = 1;
65             break;
66         case 1:
67             // Green: Card successfully read.
68             PORTBbits.RB0 = 1; //green
69             PORTBbits.RB1 = 0; //red
70
71             break;
72         case 2:
73             // Red: Error, card not read.
74             PORTBbits.RB0 = 0;
75             PORTBbits.RB1 = 1;
76             break;
77         default:
78             // : Loading
79             PORTBbits.RB0 = 0;
80             PORTBbits.RB1 = 0;
81             break;
82     }
83 }
84
85 /* Alternate set up to reduce number of pins:
86  * 2 pins to select the LED
87  * 2 pins to select the LED color
88  */
```

```
89
90
91  void LEDSelect(char card, char status) {
92      switch (card) {
93          case 0:
94              // Select Card Reader 1
95              PORTBbits.RB3 = 0;
96              PORTBbits.RB2 = 0;
97              LEDColor(status);
98              break;
99          case 1:
100             // Select Card Reader 2
101             PORTBbits.RB3 = 0;
102             PORTBbits.RB2 = 1;
103             LEDColor(status);
104             break;
105         case 2:
106             // Select Card Reader 3
107             PORTBbits.RB3 = 1;
108             PORTBbits.RB2 = 0;
109             LEDColor(status);
110             break;
111         case 3:
112             // Select Card Reader 4
113             PORTBbits.RB3 = 1;
114             PORTBbits.RB2 = 1;
115             LEDColor(status);
116             break;
117         default:
118             break;
119     }
120 }
```

H.8 Motor Driver

Note: This feature was not implemented in the final product

../FrontEnd.X/motorDriver.h

```
1  /*
2   * File:   motorDriver.h
3   * Author: Patrick Ma
4   *
5   * Created on June 1, 2014, 12:28 AM
6   */
7
8  #ifndef MOTORDRIVER_H
9  #define MOTORDRIVER_H
10
```

```
#ifdef  __cplusplus
extern "C" {
#endif

    void motorSetup(void);
    void move(int location);


#ifdef  __cplusplus
}
#endif

#endif  /* MOTORDRIVER_H */
```

../FrontEnd.X/motorDriver.c

```
/*
 *  software for controlling the motor actuator
 */

#include "globals.h"

#define ADC_PRECISION 5 // precision of the ADC, to allow motor settling
void adjustMotor(int direction);

void motorSetup() {
    // setup inputs & outputs
    PORTAbits.VREFP = 1; // port A3 (vref plus)
    PORTDbits.AN27 = 1; // analog input for motor feedback
    PORTDbits.RD6 = 0; // Fin
    PORTDbits.RD5 = 0; // Rin

    // setup port A3, D7 for analog input, others as no analog
    ANSELAbits.ANSA3 = 1;
    ANSELDbits.ANSD7 = 1;
    ANSELDbits.ANSD6 = 0;
    ANSELDbits.ANSD5 = 0;

    // setup ADC reader
    OpenADC(ADC_FOSC_2 & ADC_RIGHT_JUST & ADC_12_TAD,
            ADC_CH27 & ADC_INT_OFF,
            ADC_REF_VDD_VREFPLUS & ADC_REF_VDD_VSS);

    Delay10TCYx(5);
}

/*
 * move the motor to the given location
 */
```

```
34 void move(int location) {
35     int posn = 0;
36     ADCON0bits.GO = 1; //ConvertADC();
37     while (0 == ADCON0bits.GO); // spin while busy
38     posn = ((((unsigned int)ADRESH)<<8)|(ADRESL) - location) / ADC_PRECISION;
39     while (0 != posn) { // move forward or back depending on the location
40         if (posn < 0) {
41             adjustMotor(1);
42         } else {
43             adjustMotor(0);
44         }
45         Delay100TCYx(1);
46         ADCON0bits.GO = 1; //ConvertADC(); // determine new location and if any
       change is needed
47         while (0 == ADCON0bits.GO); // spin while busy
48         posn = ((((unsigned int)ADRESH)<<8)|(ADRESL) - location) /
      ADC_PRECISION;
49     }
50     PORTDbits.RD6 = 1; // lock the reader
51     PORTDbits.RD7 = 1;
52 }
53
54 // moves the motor forward (1) or backwards(0)
55
56 void adjustMotor(int direction) {
57     PORTDbits.RD6 = direction;
58     PORTDbits.RD7 = !direction;
59 }
```

H.9 RFID Reader Driver

../FrontEnd.X/rfidReader.h

```
1  /*
2   * File:   rfidReader.h
3   * Author: Patrick
4   *
5   * Created on May 29, 2014, 12:45 PM
6   */
7
8  #ifndef RFIDREADER_H
9  #define RFIDREADER_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15     /********ReaderData sizes**********************/
```

```
16 #define UID_SIZE 24
17 #define MAX_UIDS 5  // number of UIDs stored
18 #define RFID_BLOCK 10
19
20 void resetRFID(void);
21 void RFIDSetup(void);
22 void pingRFID(void);
23 void inventoryRFID(void);
24 void quietRFID(char* uid);
25 void sendToRFID2(char* myInput);
26 void writeRFID(char* uid, char block, int highData, int lowData);
27 void readRFID(char* uid, char block);
28 void char8RFID(char* uid, char block, char* myString);
29 //void processRFIDCmd(void);
30
31 typedef struct {
32     // Read UIDs, length can be optimized
33     // Currently can read only 3 UIDs before we get errors based on the size
34     // of the array
35     char readUID[MAX_UIDS][UID_SIZE];
36     char readData[16];
37
38     short lineFeeds;
39     short configFlag;
40
41     // Current spot in the array processing for input from RFID
42     int inputSpot2;
43
44     // The UID that we are reading. (first, second, etc)
45     int numUID;
46
47     // Weather or not we are in a block of square brackets
48     short nextBlock;
49
50     // If the user input was an inventory command
51     short invCom;
52
53     // If the user input was a single block read command
54     short readFlag_1;
55
56     // Are the UIDs available to read? Number of available UIDS
57     int availableUIDs;
58
59     // Read mode set?
60     short readMode;
61
62     int availableData;
63
64     short writeFlag_1;
```

```
65 } RFIDDriver;
66 extern RFIDDriver readerData;
67
68 #ifdef __cplusplus
69 }
70 #endif
71
72 #endif /* RFIDREADER_H */
```

../FrontEnd.X/rfidReader.c

```
1  /*
2   * File:   ioBuffer.c
3   * Author: Patrick, Ryan
4   *
5   * Created on May 15, 2014
6   * a simple program that buffers user serial input then sends it out
7   * sequentially all at once.
8   */
9
10 #include "globals.h"
11
12 /********Command constants**********************/
13 #define READER_INPUT_LENGTH 64
14 #define QUIET_LEN 37
15 #define END_COM "0000" // End of a command
16 #define STAY_QUIET "0112000304182202" // Beginning part of stay quiet, add 0000
17 #define AGC "0109000304F0000000"
18 #define REG_WRITE "010C000304100021010200000"
19 #define PING "0108000304FF0000"
20 #define INVENTORY "010B000304140601000000"
21 #define READ_SINGLE "0113000304182220"
22 #define WRITE_SINGLE "0117000304186221"
23 #define READ_SING_LEN 39
24 #define WR_SING_LEN 47
25
26 void sendToRFID(char* myString);
27 void setupRead(void);
28
29 // Set up on USART 2
30
31 // Send a ping commmand to the rfid
32
33 void pingRFID() {
34     sendToRFID(PING);
35 }
36
37 // Send an inventory command to the RFID
38
```

```
39 void inventoryRFID() {
40     // Set the inventory command flag for the interrupt
41     readerData.invCom = 1;
42     // If we have not set read mode yet, go to config mode
43     if (readerData.readMode == 0) {
44         // Config mode
45         readerData.configFlag = 1;
46 
47         // Disable current flag
48         readerData.invCom = 0;
49 
50         // Send read commands
51         setupRead();
52 
53         // Turn off config
54         readerData.configFlag = 0;
55 
56         // Turn back on inventory mode, indicate read set
57         readerData.invCom = 1;
58         readerData.readMode = 1;
59     }
60     // Send inventory command
61     sendToRFID(INVENTORY);
62 
63     // Wait until interrupt finishes
64     while (readerData.invCom == 1);
65     readerData.availableUIDs = readerData.numUID;
66 
67     // Reset the UID counters
68     readerData.numUID = 0;
69     readerData.lineFeeds = 0;
70 }
71 
72 // Send a quiet command to the given uid
73 
74 void quietRFID(char* uid) {
75     // Holds the command
76     char quietCommand[QUIET_LEN]; // {STAY_QUIET, uid, END_COM};
77     // Beginning part of command
78     strcatpgm2ram(quietCommand, STAY_QUIET);
79 
80     // Concatenate the uid
81     strcat(quietCommand, uid);
82 
83     // Add 0000 for the ending bits
84     strcatpgm2ram(quietCommand, END_COM);
85 
86     // Do sendToRFID2 because it is already saved to ram
87     sendToRFID2(quietCommand);
```

```
88      return;
89  }
90
91  #define NAME_LEN 8
92  void char8RFID(char* uid, char block, char* myString1) {
93      int i = 0;
94      int myNumHigh = 0;
95      int myNumLow = 0;
96      char myString[9] = { 0 };
97      strcpypgm2ram(myString, myString1);
98      myNumHigh |= (int)myString[0] << 8;
99      myNumHigh |= (int)myString[1];
100     myNumLow |= (int)myString[2] << 8;
101     myNumLow |= (int)myString[3];
102
103     writeRFID(uid, block, myNumHigh, myNumLow); // name: first 4 chars left to
        right FI RE
104
105     myNumHigh = 0;
106     myNumLow = 0;
107
108     myNumHigh |= (int)myString[4] << 8;
109     myNumHigh |= (int)myString[5];
110     myNumLow |= (int)myString[6] << 8;
111     myNumLow |= (int)myString[7];
112
113     writeRFID(uid, block+1, myNumHigh, myNumLow); // name: last 4 chars left to
        right  DU DE
114
115
116 }
117
118 void writeRFID(char* uid, char block, int highData, int lowData) {
119     // Holds the command
120     char writeCommand[WR_SING_LEN]; // {STAY_QUIET, uid, END_COM};
121     char dataHex[5];
122     if (readerData.availableUIDs > 0) {
123         memset(writeCommand, '\0', WR_SING_LEN * sizeof (char));
124         readerData.writeFlag_1 = 1;
125         // If we have not set read mode yet, go to config mode
126         if (readerData.readMode == 0) {
127             // Config mode
128             readerData.configFlag = 1;
129             readerData.writeFlag_1 = 0;
130
131             // Send read commands
132             setupRead();
133
134             // Turn off config
```

```
135             readerData.configFlag = 0;
136             readerData.writeFlag_1 = 1;
137         }
138 
139         // Beginning part of command
140         strcatpgm2ram(writeCommand, WRITE_SINGLE);
141 
142         // Concatenate the uid and block
143         strcat(writeCommand, uid);
144 
145         sprintf(dataHex, "%02x", (int) block);
146         strcat(writeCommand, dataHex);
147 
148         sprintf(dataHex, "%04x", highData);
149         strcat(writeCommand, dataHex);
150 
151         sprintf(dataHex, "%04x", lowData);
152         strcat(writeCommand, dataHex);
153 
154         // Add 0000 for the ending bits
155         strcatpgm2ram(writeCommand, END_COM);
156 
157         // Do sendToRFID2 because it is already saved to ram
158         sendToRFID2(writeCommand);
159 
160         // Wait until interrupt finishes
161         while (readerData.writeFlag_1 == 1);
162     }
163     return;
164 }
165 
166 void readRFID(char* uid, char block) {
167     // Holds the command
168     char readCommand[READ_SING_LEN]; // {STAY_QUIET, uid, END_COM};
169     char blockHex[3];
170     if (readerData.availableUIDs > 0) {
171         memset(readCommand, '\0', READ_SING_LEN * sizeof (char));
172         readerData.readFlag_1 = 1;
173         // If we have not set read mode yet, go to config mode
174         if (readerData.readMode == 0) {
175             // Config mode
176             readerData.configFlag = 1;
177 
178             // Disable current flag
179             readerData.readFlag_1 = 0;
180 
181             // Send read commands
182             setupRead();
183 
```

```
184             // Turn off config
185             readerData.configFlag = 0;
186 
187             readerData.readFlag_1 = 1;
188         }
189 
190         // Beginning part of command
191         strcatpgm2ram(readCommand, READ_SINGLE);
192 
193         // Concatenate the uid and block
194         strcat(readCommand, uid);
195 
196         sprintf(blockHex,"%02x",(int)block);
197         strcat(readCommand, blockHex);
198 
199         // Add 0000 for the ending bits
200         strcatpgm2ram(readCommand, END_COM);
201 
202         // Do sendToRFID2 because it is already saved to ram
203         sendToRFID2(readCommand);
204 
205         // Wait until interrupt finishes
206         while (readerData.readFlag_1 == 1);
207     }
208     return;
209 }
210 
211 /* // Depreciated
212 void processRFIDCmd() {
213     int i;
214     //     // Controls the RESET for the RFID reader
215     //     TRISBbits.RB5 = 0;
216     //     ANSELBbits.ANSB5 = 0;
217 
218     // Set up UART to computer and RFID
219     //     rs232Setup1(); // sets pc RX=C7, tx=C6
220     //     rs232Setup2(); // sets dlp rx=b7, tx=b6
221 
222     // Start the RFID with a reset
223     //resetRFID();
224 
225     // Get RFID attention
226     //sendToRFID("0");
227 
228     //     while (1) {
229     //         // Read user input from computer
230     //         readBytesUntil(readerData.userInput2, '\r', READER_INPUT_LENGTH);
231     //         putc1USART('\r');
232     //         putc1USART('\n');
```

```
233
234     // Ping command
235     sendToRFID("\n");
236     if (strcmppgm2ram(readerData.userInput2, "ping") == 0) {
237         sendToRFID(PING);
238         // Inventory Command
239     } else if (strcmppgm2ram(readerData.userInput2, "inventory") == 0) {
240         // Set the inventory command flag for the interrupt
241         readerData.invCom = 1;
242
243         // Set the RFID reader to Read mode and send the Inventory command
244         if (readerData.readMode == 0) {
245             readerData.configFlag = 1;
246             readerData.invCom = 0;
247             setupRead();
248             readerData.configFlag = 0;
249             readerData.invCom = 1;
250             readerData.readMode = 1;
251         }
252         sendToRFID(INVENTORY);
253
254         // Wait until interrupt finishes
255         while (readerData.invCom == 1);
256         // Print all the UIDs
257         for (i = 0; i < readerData.numUID; i++) {
258             puts2USART(readerData.readUID[i]);
259             putc2USART('\r');
260             while (Busy2USART());
261             putc2USART('\n');
262         }
263         readerData.availableUIDs = readerData.numUID;
264
265         // Reset the number of UIDs read
266         readerData.numUID = 0;
267         readerData.lineFeeds = 0;
268
269         // Send the "Stay Quiet" command.
270         // WARNING: THIS IS HARDCODED TO ONLY WORK WITH THE PROTOCARD
271     } else if (strcmppgm2ram(readerData.userInput2, "quiet") == 0) {
272         sendToRFID(STAY_QUIET);
273
274         // Any errors will reset the RFID reader
275     } else {
276         //resetRFID();
277     }
278     //    }
279     return;
280 }
281  */
```

```
282
283 // Sends the string to the DLP RFID 2
284
285 void sendToRFID(char* myString) {
286     // Copy string into an input array
287     char myInput[READER_INPUT_LENGTH];
288     // Says whether the input has finished sending or not
289     short inputFinished = 0;
290     int i = 0;
291
292     // Copy string to ram so it can be read correctly
293     strcpypgm2ram(myInput, myString);
294
295     // Send character by character
296     while (!inputFinished) {
297         if (myInput[i] != '\0') {
298             while (Busy1USART());
299             Write1USART(myInput[i]);
300             i++;
301         } else {
302             inputFinished = 1;
303         }
304     }
305
306     // Short delay, try removing
307     Delay10TCYx(100);
308     return;
309 }
310
311 // Use this for when you need to send a string that isn't sent
312 // in double quotes (not this: "mystring!")
313 //
314 // see sendToRFID for more comments
315
316 void sendToRFID2(char* myInput) {
317     short inputFinished = 0;
318     int i = 0;
319     while (!inputFinished) {
320         if (myInput[i] != '\0') {
321             while (Busy1USART());
322             Write1USART(myInput[i]);
323             i++;
324         } else {
325             inputFinished = 1;
326         }
327     }
328     Delay10TCYx(100);
329     return;
330 }
```

```
331
332 // Set up read commands
333
334 void setupRead() {
335     // Set to register write mode
336     sendToRFID(REG_WRITE);
337
338     // Needs to read two line breaks from dlp before
339     // continuing.
340     while (readerData.lineFeeds < 2);
341
342     // Reset line feeds for next command
343     readerData.lineFeeds = 0;
344
345     // Set AGC mode
346     sendToRFID(AGC);
347
348     // We need to read only one line feed when setting AGC
349     while (readerData.lineFeeds < 1);
350
351     // Reset line feeds
352     readerData.lineFeeds = 0;
353
354     // Send new line to clear out any junk
355     // Try removing
356     sendToRFID("\n");
357     return;
358 }
359
360 // Depreciated - we don't use them anymore
361
362 void resetRFID() {
363
364     PORTBbits.RB5 = 1;
365     PORTBbits.RB5 = 0;
366     PORTBbits.RB5 = 1;
367
368 }
369
370 // initialize the RFID reader and variables
371
372 void RFIDSetup() {
373     // initialize local vars
374     readerData.inputSpot2 = 0;
375     readerData.numUID = 0;
376     readerData.nextBlock = 0;
377     readerData.invCom = 0;
378     readerData.readMode = 0;
379     readerData.lineFeeds = 0;
```

```
380     readerData.configFlag = 0;
381     readerData.availableUIDs = FALSE;
382     memset(readerData.readUID, '\0', MAX_UIDS * UID_SIZE * sizeof (char));
383     memset(readerData.readData, '\0', 16*sizeof(char));
384     readerData.readFlag_1 = 0;
385     readerData.writeFlag_1 = 0;
386     readerData.availableData = 0;
387
388
389 #if !FRONT_NOT_BACK
390     // Get RFID attention if not already
391     sendToRFID("\n");
392 #endif
393 }
```

H.10 RS-232 Serial Connection

../FrontEnd.X/rs232.h

```
1  /*
2   * File:   rs232.h
3   * Author: castia
4   *
5   * Created on April 24, 2014, 10:10 PM
6   */
7  void rs232Setup1(void);
8  void rs232Setup2(void);
9  void readBytesUntil1USART(char* myStorage, char stopChar, int size);
10 void readBytesUntil2USART(char* myStorage, char stopChar, int size);
```

../FrontEnd.X/rs232.c

```
1  //#include "rs232.h"
2  //#include <p18f25k22.h>
3  //#include <usart.h>
4  #include "globals.h"
5  // setup for USART1
6
7  void rs232Setup1() {
8      //  Set RX as input, TX as output
9      TRISCbits.TRISC7 = 1;
10     TRISCbits.TRISC6 = 0;
11     // Enable digital for all c pins
12     ANSELCbits.ANSC6 = 0;
13     ANSELCbits.ANSC7 = 0;
14
15     //  Configure UART, 115200 baud with 20MHz clock.
16     //Open1USART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
       USART_EIGHT_BIT & USART_BRGH_HIGH, 10);
```

```
17      //  Configure UART, 9600 baud with 20MHz clock.
18  #if FRONT_NOT_BACK
19      Open1USART(USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
        USART_EIGHT_BIT & USART_BRGH_HIGH,129);
20  #else
21      Open1USART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
        USART_EIGHT_BIT & USART_BRGH_HIGH,10);
22  #endif
23      // Enable Priority
24      RCONbits.IPEN = 1;
25      // High priority receive interrupt
26      IPR1bits.RCIP = 1;
27      // Enable all high priority interrupts
28      INTCONbits.GIEH = 1;
29  }
30
31  // setup for USART2
32
33  void rs232Setup2() {
34      //  Set RX as input, TX as output
35      TRISDbits.TRISD7 = 1;
36      TRISDbits.TRISD6 = 0;
37      ANSELDbits.ANSD6 = 0;
38      ANSELDbits.ANSD7 = 0;
39
40      //  Configure UART, 115200 baud with 20MHz clock.
41      Open2USART(USART_TX_INT_OFF & USART_RX_INT_ON & USART_ASYNCH_MODE &
        USART_EIGHT_BIT & USART_BRGH_HIGH, 10);
42
43      // Enable Priority
44      RCONbits.IPEN = 1;
45      // High priority receive interrupt
46      IPR3bits.RC2IP = 1;
47      // Enable all high priority interrupts
48      INTCONbits.GIEH = 1;
49  }
50
51  // Precondition: USART 1 is open & configured
52
53  void readBytesUntil1USART(char* myStorage, char stopChar, int size) {
54      int i = 0;
55      char message;
56
57      while (!DataRdy1USART());
58      message = getc1USART();
59      while (message != stopChar && i < (size - 1)) {
60          myStorage[i] = message;
61          i++;
62          while (!DataRdy1USART());
```

```
63          message = getc1USART();
64      }
65
66      myStorage[i] = '\0';
67      i = 0;
68  }
69  void readBytesUntil2USART(char* myStorage, char stopChar, int size) {
70      int i = 0;
71      char message;
72
73      while (!DataRdy2USART());
74      message = getc2USART();
75      while (message != stopChar && i < (size - 1)) {
76          myStorage[i] = message;
77          i++;
78          while (!DataRdy2USART());
79          message = getc2USART();
80      }
81
82      myStorage[i] = '\0';
83      i = 0;
84  }
```

H.11 SRAM Primary Memory

Note: There are two different files for each microcontroller due to hardware configurations

../FrontEnd.X/SRAM.h

```
1  /*
2   * File:   SRAM.h
3   * Author: castia
4   *
5   * Created on April 24, 2014, 1:59 AM
6   */
7
8  #ifndef SRAM_H
9  #define SRAM_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15 void SRAMsetUp(void);
16 void setUpIn(void);
17 void setUpOut(void);
18 int readData(int adx);
19 void writeData(int adx, int data);
20
```

```
21
22 #ifdef  __cplusplus
23 }
24 #endif
25
26 #endif  /* SRAM_H */
```

../FrontEnd.X/SRAMfront.c

```
1  //#include "GLOBALS_H"
2  #include <p18f46k22.h>
3  #include "SRAM.h"
4  #include <delays.h>
5
6  /*
7   * This is used only for the Frontend
8   */
9
10 #if FRONT_NOT_BACK
11 unsigned short OE;
12 unsigned short WE;
13 unsigned short store;
14
15 void SRAMsetUp() {
16     // Write Enable
17     TRISCbits.TRISC2 = 0;
18
19     // Output Enable
20     TRISCbits.TRISC1 = 0;
21     // Store
22     TRISCbits.TRISC0 = 0;
23
24     // Default Values
25     PORTCbits.RC2 = 1;
26     PORTCbits.RC1 = 1;
27     PORTCbits.RC0 = 1;
28
29     // Set digitalOut
30     ANSELAbits.ANSA0 = 0;
31     ANSELAbits.ANSA1 = 0;
32     ANSELAbits.ANSA2 = 0;
33     ANSELAbits.ANSA3 = 0;
34     ANSELAbits.ANSA5 = 0;
35     ANSELEbits.ANSE0 = 0;
36     ANSELEbits.ANSE1 = 0;
37     ANSELEbits.ANSE2 = 0;
38
39     ANSELCbits.ANSC2 = 0;
40     //ANSELCbits.ANSC1 = 0; NOT NEEDED FOR PORT C
```

```
41      //ANSELCbits.ANSC0 = 0;
42
43      setUpIn();
44  }
45
46  void setUpOut () {
47      // Set data outputs
48      TRISAbits.TRISA0 = 0;
49      TRISAbits.TRISA1 = 0;
50      TRISAbits.TRISA2 = 0;
51      TRISAbits.TRISA3 = 0;
52      TRISAbits.TRISA4 = 0;
53      TRISAbits.TRISA5 = 0;
54      TRISAbits.TRISA6 = 0;
55      TRISE &= 0b11111000;
56  }
57
58  void setUpIn () {
59      // Set data inputs
60      TRISA = 0xFF;
61      TRISE = 0xFF;
62  }
63
64  // Reading data
65  int readData (int adx) {
66      int myRead = 0;
67      SRAMsetUp();
68      setUpOut();
69
70      // Setting up Address
71      PORTAbits.RA0 = (adx >> 0) & 0x01;
72      PORTAbits.RA1 = (adx >> 1) & 0x01;
73      PORTAbits.RA2 = (adx >> 2) & 0x01;
74      PORTAbits.RA3 = (adx >> 3) & 0x01;
75      PORTAbits.RA4 = (adx >> 4) & 0x01;
76      PORTAbits.RA5 = (adx >> 5) & 0x01;
77      LATEbits.LATE0 = (adx >> 6) & 0x01; // problem?
78      LATEbits.LATE1 = (adx >> 7) & 0x01; // problem?
79      LATEbits.LATE2 = (adx >> 8) & 0x01; // problem?
80      PORTAbits.RA6 = (adx >> 9) & 0x01;
81
82      // Store in MAR
83      PORTCbits.RC0 = 0;
84      PORTCbits.RC0 = 1;
85
86      // I/O are inputs
87      setUpIn();
88
89      // Output Enable
```

```
90      PORTCbits.RC1 = 0;
91      Delay10TCYx(5);
92      // Get the first 6 bits of Port A and the first 2 bits of port E
93      myRead = (PORTA & 0x3F) | ((PORTE << 6) & 0xC0);
94  //    Delay10TCYx(5);
95      // Output Enable
96      PORTCbits.RC1 = 1;
97
98      return myRead;
99
100 }
101
102
103
104 // Writing data
105 void writeData(int adx, int data) {
106     SRAMsetUp();
107     setUpOut();
108
109     // Setting up Address
110     PORTAbits.RA0 = (adx >> 0) & 0x01;
111     PORTAbits.RA1 = (adx >> 1) & 0x01;
112     PORTAbits.RA2 = (adx >> 2) & 0x01;
113     PORTAbits.RA3 = (adx >> 3) & 0x01;
114     PORTAbits.RA4 = (adx >> 4) & 0x01;
115     PORTAbits.RA5 = (adx >> 5) & 0x01;
116     LATEbits.LATE0 = (adx >> 6) & 0x01; // problem?
117     LATEbits.LATE1 = (adx >> 7) & 0x01; // problem?
118     LATEbits.LATE2 = (adx >> 8) & 0x01; // problem?
119     PORTAbits.RA6 = (adx >> 9) & 0x01;
120
121     // Store in MAR
122     PORTCbits.RC0 = 0;
123     PORTCbits.RC0 = 1;
124
125     // Send Data
126     PORTAbits.RA0 = (data >> 0) & 0x01;
127     PORTAbits.RA1 = (data >> 1) & 0x01;
128     PORTAbits.RA2 = (data >> 2) & 0x01;
129     PORTAbits.RA3 = (data >> 3) & 0x01;
130     PORTAbits.RA4 = (data >> 4) & 0x01;
131     PORTAbits.RA5 = (data >> 5) & 0x01;
132     LATEbits.LATE0 = (data >> 6) & 0x01; // problem?
133     LATEbits.LATE1 = (data >> 7) & 0x01; // problem?
134
135     // Write Enable
136     PORTCbits.RC2 = 0;
137     //Delay10TCYx(10);
138     PORTCbits.RC2 = 1;
```

```
}
#endif
```

../FrontEnd.X/SRAMback.c

```
//#include "GLOBALS_H"
#include "globals.h"
#include "SRAM.h"
#include <delays.h>

/*
 * SRAM for the backend
 */

#if !FRONT_NOT_BACK

unsigned short OE;
unsigned short WE;
unsigned short store;

void SRAMsetUp() {
    // Write Enable
    TRISCbits.TRISC3 = 0;

    // Output Enable
    TRISCbits.TRISC2 = 0;

    // Store
    TRISCbits.TRISC1 = 0;

    // Default Values
    PORTCbits.RC3 = 1;
    PORTCbits.RC2 = 1;
    PORTCbits.RC1 = 1;

    // Set digitalOut
    ANSELAbits.ANSA0 = 0;
    ANSELAbits.ANSA1 = 0;
    ANSELAbits.ANSA2 = 0;
    ANSELAbits.ANSA5 = 0;
    ANSELEbits.ANSE0 = 0;
    ANSELEbits.ANSE1 = 0;
    ANSELEbits.ANSE2 = 0;

    ANSELCbits.ANSC3 = 0;
    ANSELCbits.ANSC2 = 0;
    //ANSELCbits.ANSC1 = 0; NOT NEEDED FOR PORT C

    setUpIn();
```

```
45 }
46
47 void setUpOut() {
48     // Set data outputs
49     TRISAbits.TRISA0 = 0;
50     TRISAbits.TRISA1 = 0;
51     TRISAbits.TRISA2 = 0;
52     TRISAbits.TRISA4 = 0;
53     TRISAbits.TRISA5 = 0;
54     TRISAbits.TRISA6 = 0;
55     TRISE &= 0b11111000;
56     TRISCbits.TRISC0 = 0;
57 }
58
59 void setUpIn() {
60     // Set data inputs
61     TRISAbits.TRISA0 = 1;
62     TRISAbits.TRISA1 = 1;
63     TRISAbits.TRISA2 = 1;
64     TRISAbits.TRISA4 = 1;
65     TRISAbits.TRISA5 = 1;
66     TRISAbits.TRISA6 = 1;
67     TRISE = 0xFF;
68     TRISCbits.TRISC0 = 1;
69 }
70
71 // Reading data
72
73 int readData(int adx) {
74     int myRead = 0;
75     SRAMsetUp();
76     setUpOut();
77
78     // Setting up Address
79     PORTAbits.RA0 = (adx >> 0) & 0x01;
80     PORTAbits.RA1 = (adx >> 1) & 0x01;
81     PORTAbits.RA2 = (adx >> 2) & 0x01;
82     PORTAbits.RA4 = (adx >> 3) & 0x01;
83     PORTAbits.RA5 = (adx >> 4) & 0x01;
84     LATEbits.LATE0 = (adx >> 5) & 0x01; // problem?
85     LATEbits.LATE1 = (adx >> 6) & 0x01; // problem?
86     LATEbits.LATE2 = (adx >> 7) & 0x01; // problem?
87     PORTAbits.RA6 = (adx >> 8) & 0x01;
88     PORTCbits.RC0 = (adx >> 9) & 0x01;
89
90     // Store in MAR
91     PORTCbits.RC1 = 0;
92     PORTCbits.RC1 = 1;
93
```

```
94      // I/O are inputs
95      setUpIn();
96
97      // Output Enable
98      PORTCbits.RC2 = 0;
99      Delay10TCYx(5);
100     // Get the first 3 bits of Port A, bits 4-5 of port A, and the first 3 bits
        of port E
101     myRead = (PORTA & 0x07) | ((PORTA & 0x18) >> 1) | ((PORTE & 0x07) << 5);
102             //      Delay10TCYx(5);
103             // Output Enable
104             PORTCbits.RC2 = 1;
105
106     return myRead;
107
108 }
109
110
111
112 // Writing data
113
114 void writeData(int adx, int data) {
115     SRAMsetUp();
116     setUpOut();
117
118     // Setting up Address
119     PORTAbits.RA0 = (adx >> 0) & 0x01;
120     PORTAbits.RA1 = (adx >> 1) & 0x01;
121     PORTAbits.RA2 = (adx >> 2) & 0x01;
122     PORTAbits.RA4 = (adx >> 3) & 0x01;
123     PORTAbits.RA5 = (adx >> 4) & 0x01;
124     LATEbits.LATE0 = (adx >> 5) & 0x01; // problem?
125     LATEbits.LATE1 = (adx >> 6) & 0x01; // problem?
126     LATEbits.LATE2 = (adx >> 7) & 0x01; // problem?
127     PORTAbits.RA6 = (adx >> 8) & 0x01;
128     PORTCbits.RC0 = (adx >> 9) & 0x01;
129
130     // Store in MAR
131     PORTCbits.RC1 = 0;
132     PORTCbits.RC1 = 1;
133
134     // Send Data
135     PORTAbits.RA0 = (data >> 0) & 0x01;
136     PORTAbits.RA1 = (data >> 1) & 0x01;
137     PORTAbits.RA2 = (data >> 2) & 0x01;
138     PORTAbits.RA4 = (data >> 3) & 0x01;
139     PORTAbits.RA5 = (data >> 4) & 0x01;
140     LATEbits.LATE0 = (data >> 5) & 0x01; // problem?
141     LATEbits.LATE1 = (data >> 6) & 0x01; // problem?
```

```
142     LATEbits.LATE2 = (data >> 7) & 0x01; // problem?
143
144     // Write Enable
145     PORTCbits.RC3 = 0;
146     //Delay10TCYx(10);
147     PORTCbits.RC3 = 1;
148 }
149
150 #endif
```

H.12 SPI Initialization

Note: SPI used for communication with the LCD display

../FrontEnd.X/startup.h

```
1  /*
2   * File:   startup.h
3   * Author: castia
4   *
5   * Created on May 25, 2014, 8:00 PM
6   */
7
8  #ifndef STARTUP_H
9  #define STARTUP_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15 #include "globals.h"
16
17     void initClock(void);
18     void initSPI1(void);
19
20 #ifdef __cplusplus
21 }
22 #endif
23
24 #endif /* STARTUP_H */
```

../FrontEnd.X/startup.c

```
1  #include "startup.h"
2
3  // sets the external clock pin as input
4
5  void initClock() {
6      TRISAbits.RA7 = 1;
7  }
```

```
8
9  // sets up SPI port for Display communication
10
11 void initSPI1() {
12     // Could use the C18 function, but this gives more control
13     SSP1STAT = 0b01000000;
14     SSP1CON1 = 0b00000000;
15     SSP1CON1bits.SSPEN = 1;
16     SSP1CON2 = 0b00000000;
17
18     TRISCbits.RC4 = 0; // A0
19     //     TRISCbits.RC1 = 0; // RST
20     TRISBbits.RB4 = 0; // CS
21     TRISCbits.RC3 = 0; // SCK1
22     TRISCbits.RC5 = 0; // SDO1
23     ANSELBbits.ANSB4 = 0;
24     ANSELCbits.ANSC3 = 0;
25     ANSELCbits.ANSC4 = 0;
26     ANSELCbits.ANSC5 = 0;
27 }
```

H.13 Wireless Connectivity via Xbee

../FrontEnd.X/xbee.h

```
1  /*
2   * File:   xbee.h
3   * Author: castia
4   *
5   * Created on June 8, 2014, 2:20 PM
6   */
7
8  #ifndef XBEE_H
9  #define XBEE_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15 #include "globals.h"
16
17 #define FIND_ID "1234"
18 #define PLAYER_ID "5555"
19 #define HOST_ID "6666"
20 #define SEARCH_NET "4444"
21 #define PLAY_NET "8765"
22
23 void hostGame(void);
```

```
24 void findGame(void);
25 void generateNetwork(char* newNetwork);
26 void generateID(char* otherID);
27 void setupHost(void);
28 void setupClient(void);
29
30 void resetXbee(void);
31 void setupXbee(void);
32 void setXbeeNetwork(char* myNetwork, char* myID, char* myDL);
33
34
35 #ifdef __cplusplus
36 }
37 #endif
38
39 #endif /* XBEE_H */
```

../FrontEnd.X/xbee.c

```
1  #include "globals.h"
2
3  char myATMY[5];
4
5  void hostGame() {
6      char playerID[16] = {0};
7      char newNetwork[5] = {0};
8          // Beep off
9      TRISBbits.RB5 = 1;
10     srand(ReadTimer0());
11     setupHost();
12     while(Busy1USART());
13     while (!DataRdy1USART());
14     while (DataRdy1USART()) {
15         getc1USART();
16     }
17     putc1USART('2');
18     while(Busy1USART());
19 //     // Broadcast current ID for players
20 //     while (!DataRdy1USART()) {
21 //         puts1USART(myATMY);
22 //         while (Busy1USART());
23 //         putc1USART('\r');
24 //         while (Busy1USART());
25 //         Delay10TCYx(50);
26 //     }
27 //
28 //     // Read player id
29 //     readBytesUntil1USART(playerID, '\r', 16);
30 //
```

```
31 //     // Talk directly to player
32 //     setXbeeNetwork(SEARCH_NET, myATMY, (const rom char*)playerID);
33 //
34 //     // Create a private network
35 //     generateNetwork(newNetwork);
36 //
37 //     // Contact player with new network
38 //     putrs1USART(newNetwork);
39 //     while (Busy1USART());
40 //     putc1USART('\r');
41 //     while (Busy1USART());
42 //
43 //     // Change networks and broadcast until response
44 //     setXbeeNetwork(newNetwork, myATMY, (const rom char*)playerID);
45 //     while (!DataRdy1USART()) {
46 //         putc1USART('1');
47 //         while (Busy1USART());
48 //     }
49 //
50 //     // get response
51 //     getc1USART();
52 //
53 //     // send confirmation m
54 //     putc1USART('0');
55 //     while (Busy1USART());
56 }
57
58 void findGame() {
59     char hostID[16] = {0};
60     char newNetwork[16] = {0};
61         // Beep off
62     TRISBbits.RB5 = 1;
63     srand(ReadTimer0());
64     setupClient();
65
66     while(Busy1USART());
67     putc1USART('1');
68     while(Busy1USART());
69
70     while (!DataRdy1USART());
71     while (DataRdy1USART()) {
72         getc1USART();
73     }
74 //
75 //     while (!DataRdy1USART());
76 //     // want between two \r's, so do this twice
77 //     readBytesUntil1USART(hostID, '\r', 16);
78 //     readBytesUntil1USART(hostID, '\r', 16);
79 //     // Configure to send to host
```

```
80 //     setXbeeNetwork(SEARCH_NET, myATMY, (const rom char*)hostID);
81 //
82 //     // change own id to avoid getting any more input
83 //     generateID(hostID);
84 //
85 //     //broadcast new id
86 //     puts1USART(myATMY);
87 //     while (Busy1USART());
88 //     putc1USART('\r');
89 //     while (Busy1USART());
90 //
91 //     // wait for network
92 //     while (!DataRdy1USART());
93 //
94 //     // read new network
95 //     readBytesUntil1USART(newNetwork, '\r', 16);
96 //
97 //     // Change the network
98 //     setXbeeNetwork(newNetwork, myATMY, (const rom char*)hostID);
99 //
100 //     // Wait until contacted by host on new network
101 //     while (!DataRdy1USART());
102 //     getc1USART();
103 //
104 //     // respond
105 //     putc1USART('0');
106 //     while (Busy1USART());
107 //
108 //     // wait for confirmation
109 //     while (!DataRdy1USART());
110 //     getc1USART();
111 }
112
113 void generateNetwork(char* newNetwork) {
114     int network = 0;
115     while (network == 0 || (strcmppgm(newNetwork, SEARCH_NET) != 0)) {
116         network = rand() % 0xFFFE;
117         sprintf(newNetwork, "%04x", network);
118     }
119 }
120
121 void setupHost() {
122     // set network to SEARCH_NET
123     // set id to random
124     // set ATDL to FIND_ID
125
126 //     generateID("FFFE");
127 //     setXbeeNetwork(SEARCH_NET, myATMY, FIND_ID);
128     strcpypgm2ram(myATMY, HOST_ID);
```

```
129     setXbeeNetwork(SEARCH_NET, myATMY, FIND_ID);
130
131 }
132
133 void setupClient() {
134 //     strcpypgm2ram(myATMY, FIND_ID);
135 //     setXbeeNetwork(SEARCH_NET, myATMY, "0000");
136     strcpypgm2ram(myATMY, FIND_ID);
137     setXbeeNetwork(SEARCH_NET, myATMY, HOST_ID);
138     // set id to FIND_ID
139     // set network to SEARCH_NET
140 }
141
142 void generateID(char* otherID2) {
143     int ID = 0;
144
145     ID = rand();
146     sprintf(myATMY, "%04x", ID);
147 }
148
149 //void configureATDL(char* hostID) {
150 //     // Tx set low
151 //     TXSTA1bits.TXEN1 = 0;
152 //     PORTCbits.RC6 = 0;
153 //
154 //     resetXbee();
155 //     // Reenable Tx
156 //     TXSTA1bits.TXEN1 = 1;
157 //     putrs1USART("ATDL");
158 //     while (Busy1USART());
159 //     putrs1USART(hostID);
160 //     while (Busy1USART());
161 //
162 //     // ATWR,AC,CN - Write changes to nonVolatile memory
163 //     // ATAC - Apply changes
164 //     // ATCN - Exit config mode
165 //     //Carriage return
166 //     putrs1USART("WR,AC,CN\r");
167 //     while (Busy1USART());
168 //     while (!DataRdy1USART());
169 //
170 //}
171
172 void resetXbee(void) {
173     //Reset Pin- configure these to be outputs
174     //     PORTBbits.RB7 = 0;
175     PORTAbits.RA0 = 0;
176     // Delay 10 Instruction cycles, pulse must be at least 200ns;
177     Delay10TCYx(5);
```

```
178     //     PORTBbits.RB7 = 1;
179     PORTAbits.RA0 = 1;
180     // Two pulses/responses come in on startup
181     while (!DataRdy1USART());
182     while (!DataRdy1USART());
183 }
184
185 void setupXbee(void) {
186
187     //     TRISBbits.RB7 = 0;
188     //     PORTBbits.RB7 = 1;
189     TRISAbits.RA0 = 0;
190     ANSELAbits.ANSA0 = 0;
191     PORTAbits.RA0 = 1;
192 }
193
194 void setXbeeNetwork(char* myNetwork, char* myID, char* myDL) {
195     char myCom[12] = { 0 };
196     // Tx set low
197     TXSTA1bits.TXEN1 = 0;
198     PORTCbits.RC6 = 0;
199
200     resetXbee();
201     // Reenable Tx
202     TXSTA1bits.TXEN1 = 1;
203     while (DataRdy1USART()) {
204         getc1USART();
205     }
206     strcpypgm2ram(myCom, "ATID");
207     strcat(myCom, myID);
208     strcatpgm2ram(myCom, "\r");
209     puts1USART(myCom);
210     while (Busy1USART());
211     while (DataRdy1USART()) {
212         getc1USART();
213     }
214
215     strcpypgm2ram(myCom, "ATMY");
216     strcatpgm2ram(myCom, myNetwork);
217     strcatpgm2ram(myCom, "\r");
218     puts1USART(myCom);
219     while (Busy1USART());
220     while (DataRdy1USART()) {
221         getc1USART();
222     }
223
224     strcpypgm2ram(myCom, "ATDH");
225     strcatpgm2ram(myCom, "0000");
226     strcatpgm2ram(myCom, "\r");
```

```
227     puts1USART(myCom);
228     while (Busy1USART());
229     while (DataRdy1USART()) {
230         getc1USART();
231     }
232
233     strcpypgm2ram(myCom, "ATDL");
234     strcatpgm2ram(myCom, myDL);
235     strcatpgm2ram(myCom, "\r");
236     puts1USART(myCom);
237     while (Busy1USART());
238     while (DataRdy1USART()) {
239         getc1USART();
240     }
241
242     putrs1USART("ATWR,CN\r");
243     Delay1KTCYx(170);
244     putrs1USART("A");
245     while (DataRdy1USART()) {
246         getc1USART();
247     }
248 //
249 //      putrs1USART("ATCN\r");
250 //      while (Busy1USART());
251 //      while (!DataRdy1USART());
252 //      while (DataRdy1USART()) {
253 //          getc1USART();
254 //      }
255
256
257 //      //putrs1USART("ATID");
258 //      while (Busy1USART());
259 //      putrs1USART(myNetwork);
260 //      while (Busy1USART());
261 //      putrs1USART(",MY");
262 //      while (Busy1USART());
263 //      puts1USART(myID);
264 //      while (Busy1USART());
265 //      putrs1USART(",DL");
266 //      while (Busy1USART());
267 //      putrs1USART(myDL);
268 //      while (Busy1USART());
269 //      putrs1USART(",DH");
270 //      while (Busy1USART());
271 //      putrs1USART("0000");
272 //      while (Busy1USART());
273
274     // Config Commands
275     //Carriage return
```

```
276     // Config Commands
277     //Carriage return
278     // etc
279     // Exit
280
281     // ATWR,AC,CN - Write changes to nonVolatile memory
282     // ATAC - Apply changes
283     // ATCN - Exit config mode
284     //Carriage return
285 //    putrs1USART(",WR,AC,CN\r");
286 //    while (Busy1USART());
287 //    while (!DataRdy1USART());
288 //    while (DataRdy1USART()) {
289 //        getc1USART();
290 //    }
291     //
292     //    putrs1USART("ATID\r");
293     //    while(Busy1USART());
294     //    while (!DataRdy1USART());
295 }
```