Super Scrabble Timer and Scorekeeper

Gaurav Singal
Nathan Hale
Andrew DiMichele
Vishal Govil
Hubert Lin

Embedded Systems Design
Professor Edwards

May 10, 2005

Table of Contents

Project Overview

Our project, the Super Scrabble Timer and Scorekeeper presents an interesting challenge for the scope of Embedded Systems Design. The idea for the project was presented by an actual client interested in marketing this handheld device to the makers of Super Scrabble (Hasbro and Mattel), and eventually making the product available for purchase by consumers.

Accordingly, this handheld device had many requirements – it had to operate on batteries, be inexpensive to mass-produce, be user-friendly and intuitive, and be entirely self-contained. The project's challenges then became many-fold, and involved considerations of power-consumption, cost, user-interface design, and balancing client satisfaction with feasible implementation. Actual implementation challenges would include physical construction and size constraints as well as limitations imposed by batteries and the mobile nature of the device.

The final device is a black plastic box, measuring 23.7cm x 13.5cm. The box was machined to include a 40x4 character LCD display for user input/display, a keypad matrix used to enter letters and numbers, and an array of buttons used to interface the user with the various functions of the timer and score keeper. The device operates on 4 AA batteries and has a knob to adjust the contrast of the display (to compensate for dropping power levels as the batteries get depleted).

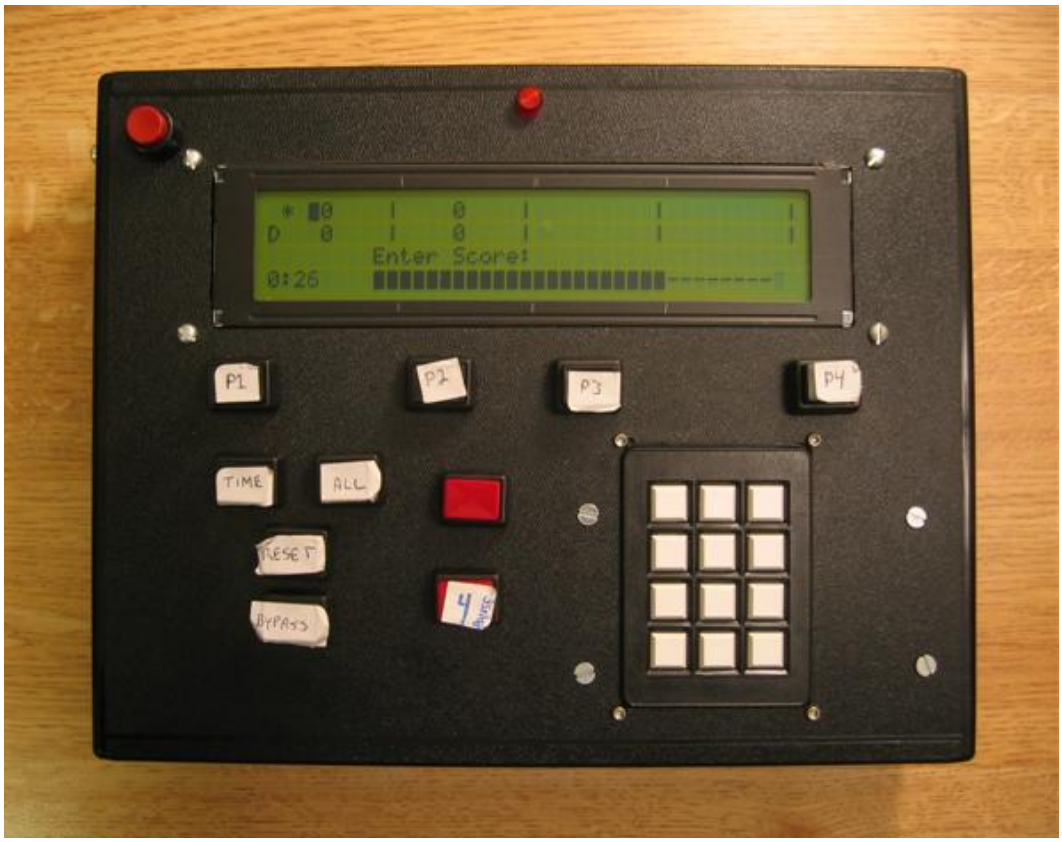


Figure 1: Final Device

Interface Design

Power:

1. Slide the power switch on the right side of the box to the ON position.

Contrast:

1. Turn the knob on the left side of the box to lighten or darken the screen.

Number of Players:

1. After turning on the machine, the screen will prompt you for the number of players.
2. Press a number on the keypad between 2 and 4.

Entering Player Names:

1. Use the keypad to enter a 3-letter name for each player
2. For each button on the keypad that can enter a letter, press it once for the first letter, a second time for the second letter, a third time for the third letter, and if possible, a fourth time for the fourth letter.

3. Press the ENTER button on the keypad move the cursor over one space. Use the Backspace button to correct errors.
4. Press a player button to move the cursor to a player's portion of the screen or press the ENTER button after having entered three letters to a player's name to move to the next player (i.e. from player 1 to player 2).

Entering Time:

1. If you wish for there to be no time limit for any player, press the BYPASS button after entering player names. If during normal play you decide to remove time limits you can press the BYPASS button as well.
2. Press the TIME button to begin entering time limits for each player.
3. Enter a time using the keypad. Use backspace to edit.
4. When you have finished with a time limit press the player button of the player you wish to assign the time to. The time limit should appear under the player's total score.
5. You may reenter time limits for players until you are satisfied.
6. If you wish for all players to have the same time limit, press the ALL button after entering a time.

Playing the game:

1. When you have finished entering time limits, press the START button to begin play. NOTE: after pressing the start button, countdown of player 1's time will immediately begin.
2. A player's time will countdown to zero until a score is entered for that player.
3. When there are 30 seconds left in a player's turn, the LED on the machine will begin to flash. When time runs out, a buzzer will sound, signaling the end of the turn.
4. The machine will pause after every turn until a score is entered for the player whose turn it just was.

Entering a score:

1. During a player's turn, enter the score for that player by using the keypad.
2. Use the Backspace button to edit the score.
3. Press ENTER or the player's Player button to enter the score.
4. The entered score should appear in the player's last score box and should be added to the player's total score.
5. The next player's turn will begin immediately after the score is entered.

Adjusting player score:

1. Total scores for players other then the player whose turn it is can be adjusted during any turn.
2. Enter a number using the keypad. Use the Backspace button to correct mistakes.

3. If you wish to subtract from a player's total score, press the Backspace button before entering any numbers. A negative sign (-) should appear next the "Enter Score: " on the LCD screen.
4. Press the Player button of the player whose score you want to adjust to complete the adjustment.

Stoppage of Play:

1. **PAUSE button**:
   If at any point during a player's turn time should be stopped, press the PAUSE button to stop the countdown. Press the START button to continue the player's turn. You may adjust player scores during the paused phase.
2. **CHALLENGE button**:
   If there is a challenge situation for a player's entered word, press the CHALLENGE button to stop game play until the situation is resolved. Press the START button to continue game play. You may adjust player scores during the paused phase.
3. **RESET button**:
   If a player's turn began before the player was ready, press the RESET button to reset the player's time and restart the player's turn.
4. **BYPASS button**:
   At any point during game play, you may press the BYPASS button to remove time limits from every player.

GUI: LCD

The LCD displays were designed to correspond with the interface design. Specifically, we created four main LCD screens which represent different phases of timing/scorekeeping. These four phases include an initial screen which asks for the number of players in the game, followed by a screen to enter initials of each player, a screen to set the time limit for each player, and finally a timing screen. During actual timing, the time is shown in minutes and seconds, along with a time-left bar which

shortens as time runs out. The four screens can be seen in the following images:

Figure 2: Number of Players Entry

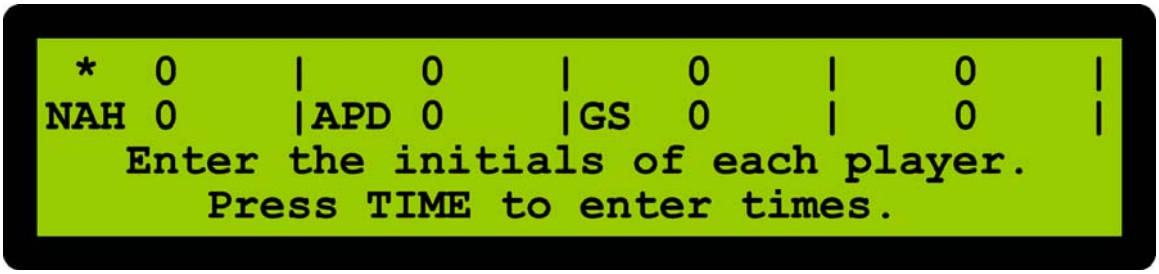


Figure 3: Player Initials Entry

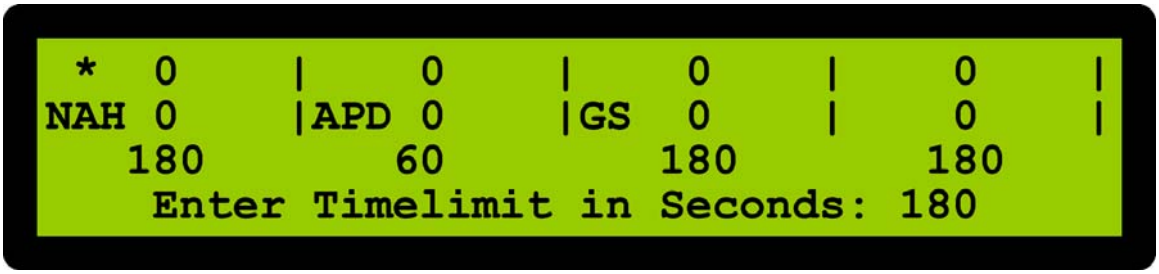


Figure 4: Timelimit Entry

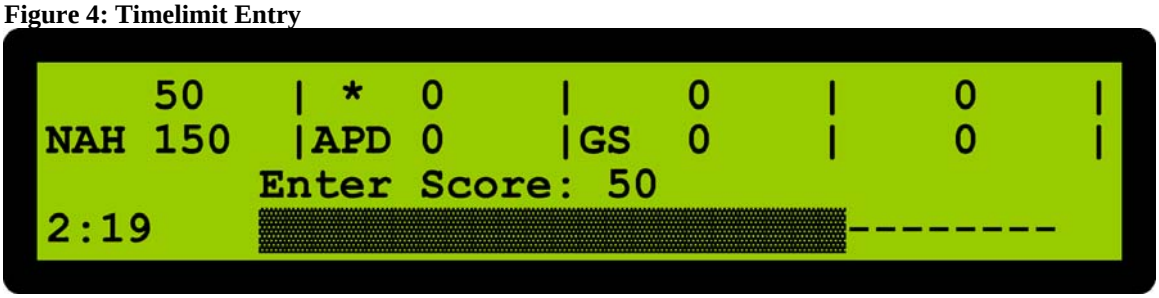


Figure 5: Timing Screen

Software Architecture

The code for this project was split up into 3 parts:

- A keypad driver
- An LCD driver
- The main FSM.

The keypad driver, keypad.c, contains functions to ease the use of the button matrix. Button presses are captured via an interrupt which calls on a function in the driver to register the button press. Not only does this function translate the key code into the correct numeral, but since we use the cell phone style of alphabetic input where repeated presses of the same key produce different characters, it also determines the intended character.

The LCD driver, scrabblelcd.c, contains functions to ease LCD use, particularly output. Actual hardware I/O is handled via a library included with our compiler package so its primary purpose is to initialize the screen and maintain the structure of the GUI. It has functions to put text in the correct places, switch focus between players, and display the time status of the current game. There is another header, called mylcd, which defines several global, hardware-specific functions which are used by the LCD library.

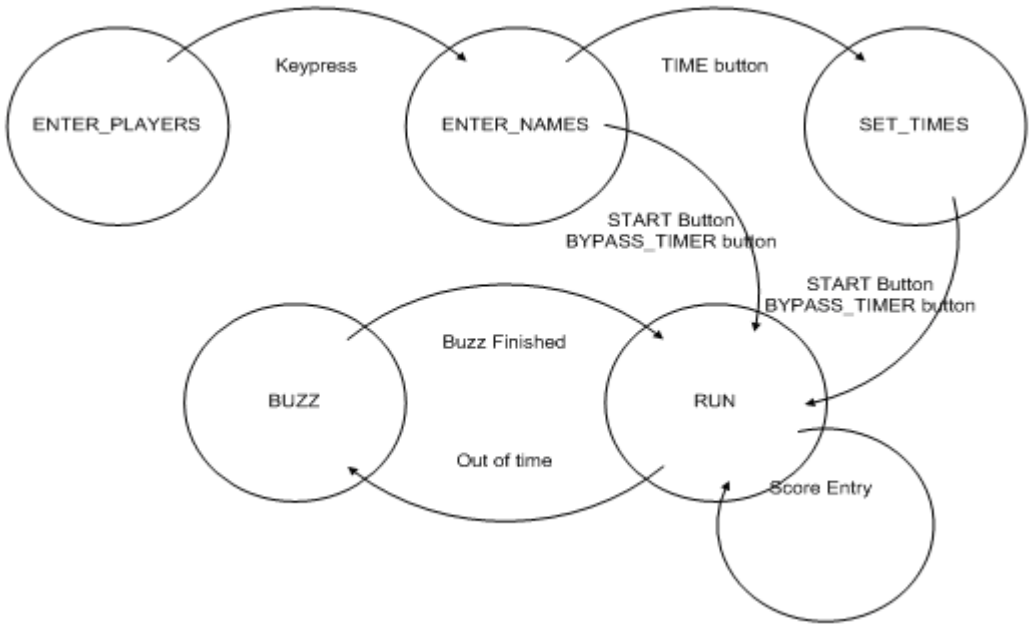
The FSM is the bulk of our code. The file in which it's implemented, capturetest.c (so named because it started a test program but ended up growing into our final implementation), is the main entry point of our software, setting up both button-press capture interrupts and the timer interrupts and initializing all hardware.

The FSM consists of 5 States:

- the ENTER_PLAYERS state
- the ENTER_NAMES state

- the SET_TIMES state
- the RUN state
- the BUZZ state

The following diagram shows how the states interact.



We designed the FSM such that the states are separated based on the type of I/O we perform within each. For example, some states expect numeric input and some expect alphabetic input. States also tend to differ on where they output to the screen and which part of the game they let you modify.

As you can see, the entry point to our FSM is the ENTER_PLAYERS state. This state is simple. It waits for a button press, a single digit between 2 and 4, indicating the number of people that will be playing. It then proceeds to the ENTER_NAMES state.

The ENTER_NAMES state lets the user enter initials for each player. It takes alphabetic input from the keypad to do so. From here, the user can either jump right into the game or set times for each player.

The SET_TIMES state lets the user set a time limit for each player or for all players at the same time. A player doesn't necessarily need a time limit. From here the user must start the game

The RUN state starts by immediately starting the timer for player one. Score entry is allowed while the timer is running and once the current player enters his score play progresses to the next player. Scores may also be entered for other players, which is good for making adjustments if a mistake was made. As time runs low (our client defined this as being when you have 30 seconds or less remaining), a blinking LED is started. When time runs out, control is transferred to the BUZZ state.

The BUZZ state's only purpose is to sound the buzzer for a present amount of time. This state probably isn't necessary but it was included in original code of the system so it persisted. Once the buzz time is up, control is transferred back to the RUN state where it waits for score entry.

Hardware Architecture

We used a PIC18F452 microcontroller to do all of the FSM processing. We decided to use this particular PIC because of its large number of input and output pins (34) and low cost. It also has a 32k onboard program memory which allowed us to keep the pcboard design simple by not requiring any off-board memory. We clocked the PIC with a 1MHz oscillator, down from our original 30MHz plan in order to minimize power consumption. We used a keypad matrix for alphanumeric input, and created our own matrix of buttons to control the device. Both matrices were interfaced to the PIC using MM74C922 16-key encoders which converted the button presses into addresses indicating the row and column of the pressed button, buffering this data, and setting a

data-available flag. Finally, a 40x4 character LCD of type DMC40457NY-LY-B was used for information display, and was interfaced to the PIC using four data lines and three control lines. A fourth control line was added manually to select which of the two onboard controllers should be used at any given time – since the controllers are designed to operate a 40x2 character LCD, our LCD came with two such controllers with separate enable lines. We modified the LCD libraries by adding a “chip-select” control line, which selected which of the two LCD’s should be modified by the sent command. This new control line was integrated into the existing LCD interface using a simple “selector” circuit. The full circuit diagram can be seen below:

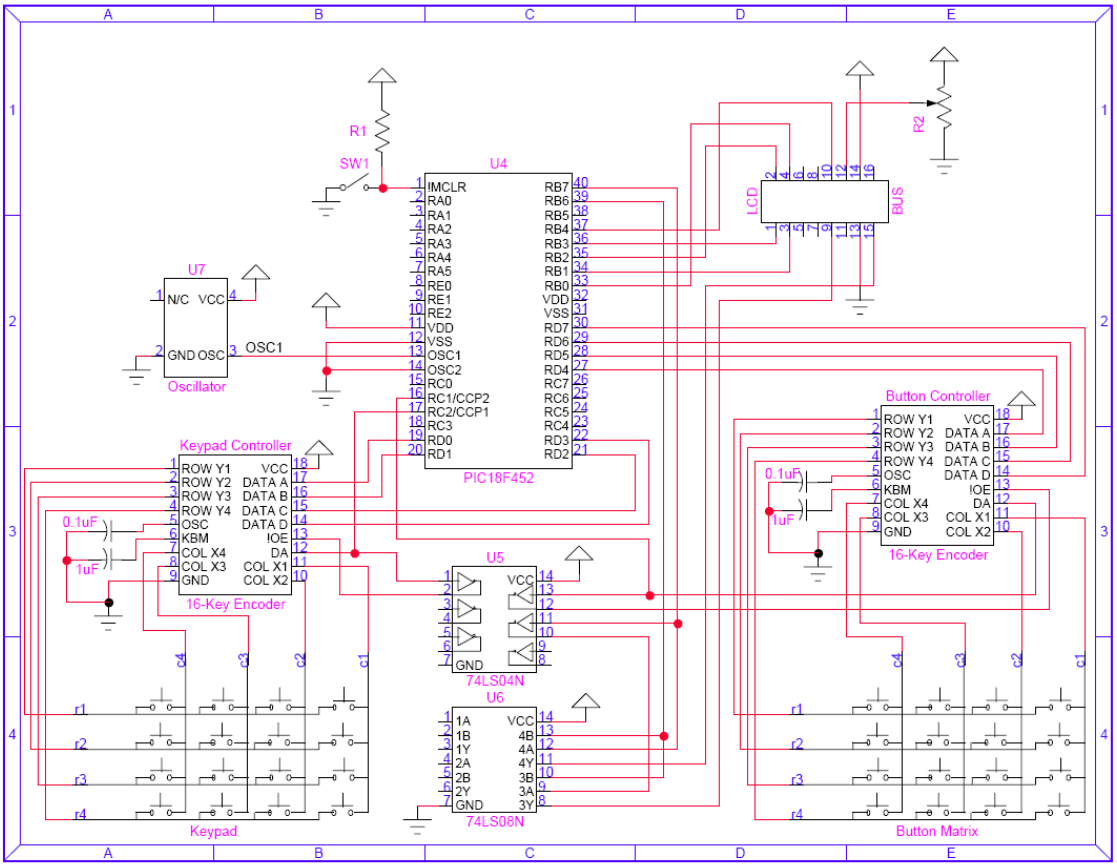


Figure 6: Circuit Diagram

Our hardware was divided onto two pc boards. One pc board was used exclusively for the button matrices. This had the buttons soldered to the board, along with appropriate wiring to handle the signals. This board and the buttons soldered to it were connected to the front plate of the box. We used a ribbon cable to connect this board to a second board, screwed to the back side of the box. This was the main board with the PIC, oscillator, decoders, and other logic components. This board has another ribbon cable which connects this board to the LCD, which is attached to the front plate of the box. The battery holder is also attached to the backside of the box, and soldered to the main board. The battery holder uses 4 AA batteries, providing 6V to the entire circuit.

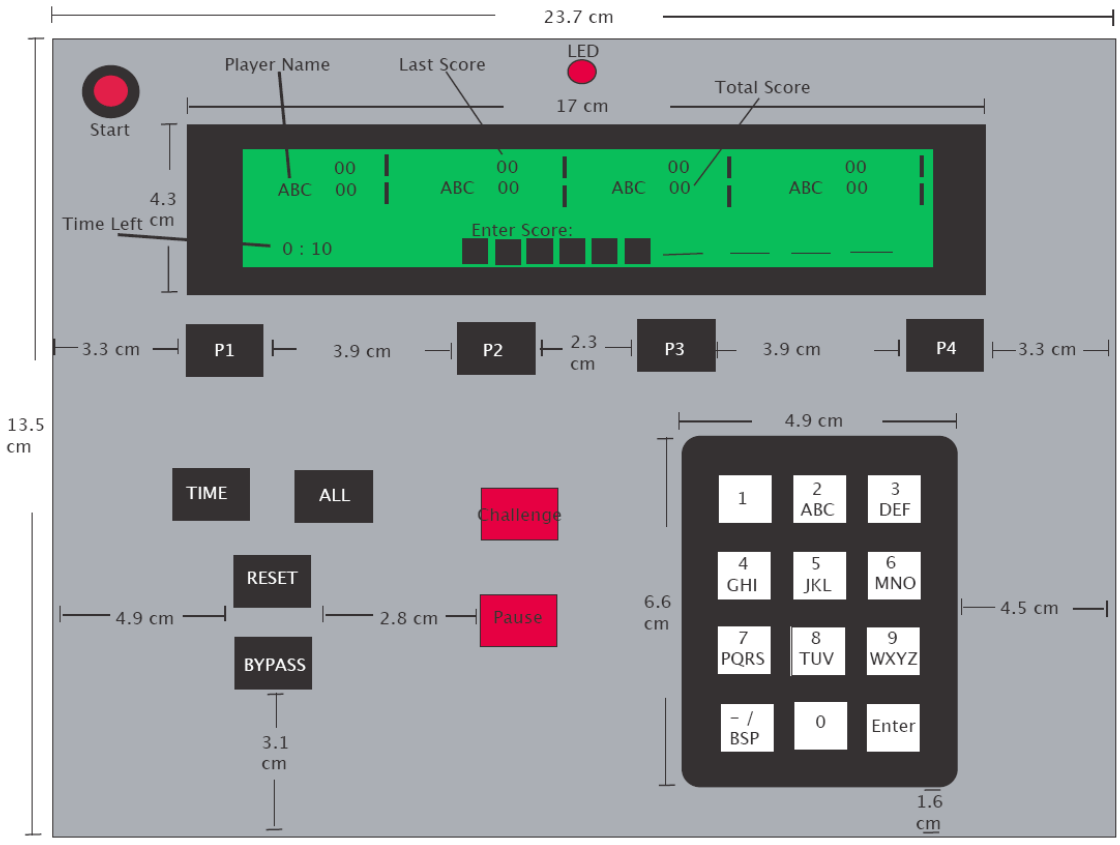


Figure 7: Drawing of Machined Device

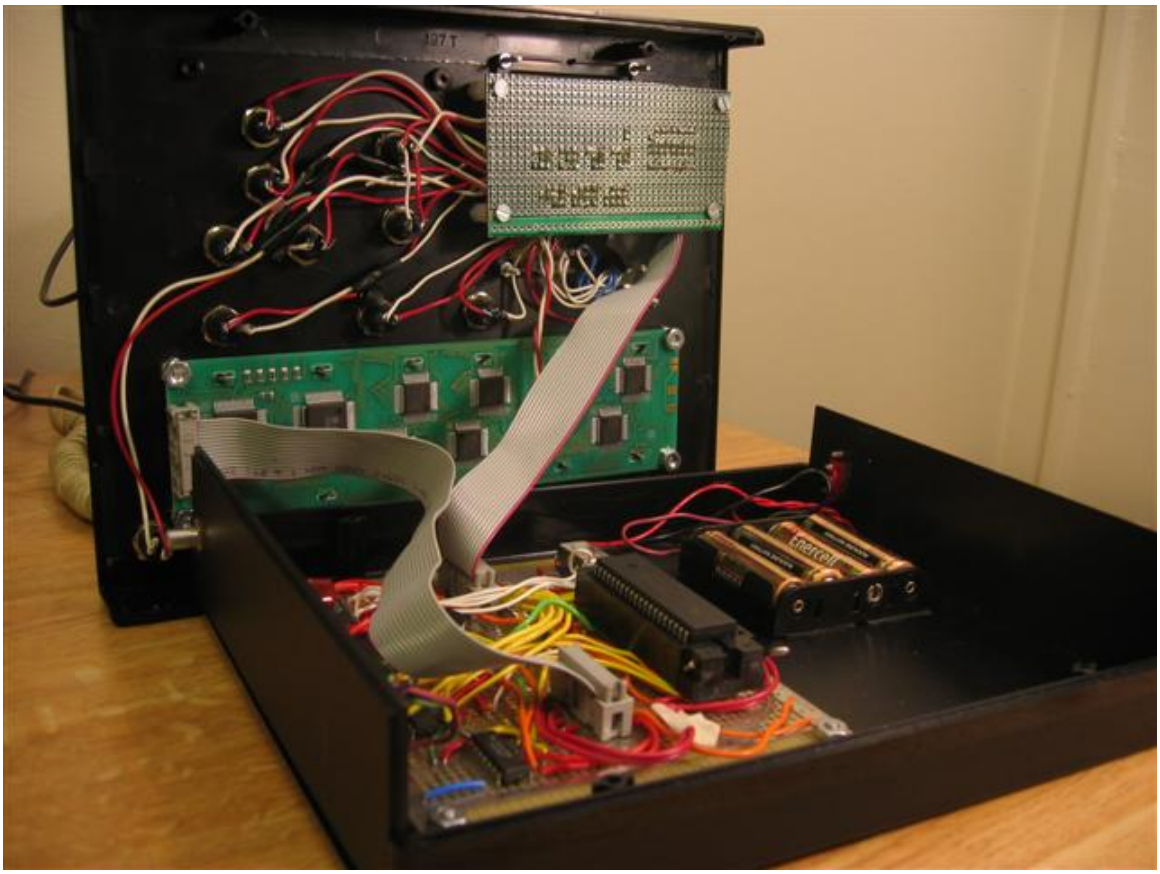


Figure 8: Internals of Machined Box Showing LCD, Dual PCBoards, Bus Ribbon Cables, and Battery Holder

Division of Labor

PIC Programming:
Gaurav worked primarily on the software architecture needed to add a controller to the LCD screen. Andrew worked on the Finite State Machine used to control the PIC.

Hardware Architecture Construction:
Nathan, Hubert and Vishal worked primarily on wiring and soldering the initial PC board, which contained the PIC sockets, decoders, AND chip, Inverters, and sockets for the ribbon cable. Nathan and Vishal then worked on wiring and soldering the second PC board, which contained the input from all the buttons on the machine as well as the LED. This work was done in the 12th floor student lab of the Mudd Building.

Box Machining:
Hubert and Gaurav worked on machining the box in the Mechanical Engineering lab on the 2nd floor of the Mudd building. The head machinist, Walter Khan, helped them with measurements and hole cutting.

Lessons Learned/Advice For Future

1) Start Early – Things take a lot longer than you expect, especially debugging
2) Have detailed schematics before constructing hardware
3) Keep the project feasible given the timeframe
4) Don’t have loose wires, and use stranded wires when possible
5) Be very careful when applying power to a circuit, accidental shorts can occur frequently and easily.
6) Have ready extra hardware for you circuit. You will break or fry your chips during your first attempt at wiring.

Appendix: Code Listing

capturetest.h

```
#ifndef __CAPTURETEST_H
#define __CAPTURETEST_H

//Struct to hold player info data
typedef struct s_player_info_t {
      char name[4];
      unsigned int timelimit; //0 for no limit
      int last_score;
      int score;
      int bingos;
} player_info_t;

//Function prototypes

void init(void);              //Initialize program
void timer_isr (void);        //Timer interrupt
void capture1_isr (void);     //Key press interrupt
void capture2_isr (void);     //Button press interrupt

void initRunMode(int playernum);    //Initialize run mode
void dopause(void);                       //Pause
void unpause(void);                       //unpause
void togglepause(void);                 //toggle pause
void do_outoftime(void);                //Out of time code
void un_outoftime(void);                //Reverse out of time
void nolimit(void);                       //Set no time limit


#define BUZZ_LENGTH 4 //Amount of time to buzz for in seconds

#define FLASH_RATE_ON  1 //Flash on time in seconds
#define FLASH_RATE_OFF 2 //Flash off time in seconds
#define FLASH_START 30   //Start timer when there are this many seconds
left

#define CCP1 (ccpflags & 0x01)      //Capture 1 bit
#define CCP2 (ccpflags & 0x02)      //Capture 2 bit

#define SET_CCP1 ccpflags=(ccpflags | 0x01)     //Set the bit
#define SET_CCP2 ccpflags=(ccpflags | 0x02)     //

#define CLR_CCP1 ccpflags=(ccpflags & 0x02)     //Clear the bit
#define CLR_CCP2 ccpflags=(ccpflags & 0x01) //

//states for FSM:
#define ENTER_NAMES_STATE               0
#define SET_TIMES_STATE                 1
#define RUN_STATE                       2
#define BUZZ_STATE                            3
#define ENTER_PLAYERS_STATE         4
```

```
//buttons
#define TIME_BYPASS_BUTTON    2
#define TIME_BUTTON                 11
#define ALL_BUTTON                  3
#define PAUSE_BUTTON          4
#define START_BUTTON          0
#define RESET_TIMER_BUTTON  1

#define PLAYER1_BUTTON              7
#define PLAYER2_BUTTON              5
#define PLAYER3_BUTTON              6
#define PLAYER4_BUTTON              10

//KEYPAD BUTTONS
#define ENTER_KEY             11    //These are on the keypad
#define PLUS_KEY              11
#define NEXT_KEY              0
#define SPACE_KEY             1
#define BACKSPACE_KEY         10
#define MINUS_KEY             10

#endif
```

capturetest.c

```
#include <p18cxxx.h>
#include <p18f452.h>
#include <xlcd.h>
#include <stdlib.h>
#include <delays.h>
#include <timers.h>
#include <capture.h>
#include "..\mylcd\mylcd.h"
#include "..\scrabblelcd\scrabblelcd.h"
#include "..\keypad\keypad.h"
#include "capturetest.h"

#define TRUE 1
#define FALSE 0

//Number of seconds passed between timer interrupts
#define SECONDS_PER_TIMECOUNT       .25 //This is for the 1MHz clock.
                                        //For 30MHz, set
this to (4.0/30)
                                        //and change the
timer interrupt prescaler

//data capture flags
volatile char ccpflags = 0;

//Variables used in interrupts must be declared volatile
volatile unsigned int timecount = 0;      //timer interrupt count
volatile unsigned int buzzcount = 0;      //interrupt count for buzzer
timing
volatile unsigned int button_press_val;   //holds vaue of button press
```

```
volatile unsigned int pause = FALSE;      //flag to determine pause
status
volatile unsigned int flashticks = 0;     //total flash time in number
of interrupts
volatile unsigned int flashon = FALSE;    //flag set if LED should be
flashing

unsigned int flashval = 0;          //value of the flash bit (LED on or
off)
unsigned int currentflash = 0;      //total time of current LED
unsigned int haslimit = TRUE; //flag to determine time limit status for
current player
unsigned int timelimit = 60;  //time limit of current player in seconds
unsigned int negscore = FALSE;      //flag to set pos/neg status of
score
unsigned int timeleft;              //amount of time left for current
player
unsigned int outoftime = FALSE;     //flag set if current player if out
of time
unsigned int buzzed = FALSE;  //flag set if the buzzer has already gone
off

int charnum=0;                          //keeps track of position in
entering data
int player=0;                           //keeps track of the current
player
char score_entry_string[10];  //stores entered score as string
int numplayers = 4;                     //stores the number of
players playing
player_info_t player_info[4]; //structure to hold player info

/*
* For PIC18cxxx devices the high interrupt vector is found at
* 00000008h. The following code will branch to the
* high_interrupt_service_routine function to handle
* interrupts that occur at the high vector.
*/
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
      //Poll each interrupt bit and branch to the correct routine
      //Have to use asm goto so that the function returns correctly
      if (INTCONbits.TMR0IF == 1) {
            _asm GOTO timer_isr _endasm
      }
      if (PIR1bits.CCP1IF == 1) {
            _asm GOTO capture1_isr _endasm
      }
      if (PIR2bits.CCP2IF == 1) {
            _asm GOTO capture2_isr _endasm
      }
}
#pragma code

#pragma interrupt timer_isr save=PROD //Define timer_isr as an isr
void timer_isr (void)
{
```

```
      INTCONbits.TMR0IF = 0;//Turn the interrupt flag off
      if (pause == FALSE) {
            timecount++;      //Increment the time if not paused
      }
    if (flashon == TRUE) {
          flashticks++; //If the flasher is on, increment the flash
time
          currentflash++;     //
      }
      buzzcount++; //Increment the buzzer time
}

#pragma interrupt capture1_isr //define capture1_isr as an isr
void capture1_isr (void)
{
      //This interrupt is called when a keypad button is pressed
      PIR1bits.CCP1IF = 0;    //Turn the interrupt flag off
      registerPadPress();           //Store the current key press
      SET_CCP1;                     //Set the flag indicating a key
press
}

#pragma interrupt capture2_isr //define capture2_isr as an isr
void capture2_isr (void)
{
      //This interrupt is called when a large button is pressed
      PIR2bits.CCP2IF = 0;    //Turn the interrupt flag off
      SET_CCP2;                     //Set the flag indicating a button
press
      button_press_val = (PORTD & (0xf0)) >> 4; //Calculate the index
of the button
}

void main (void)
{
      char strtime[20]; //Character buffers to hold text data
      char buff[10];          //

      int charentered = FALSE;      //Has a character been entered (and
not displayed)?
      char lastchar;                      //Last character entered
      unsigned int redraw = FALSE;//Should a redraw be performed
      int key;                            //Hold a key press value
      char time_entry_string[10];   //stores entered time as string
      int i;      //Commonly used in for loops

      int state = ENTER_PLAYERS_STATE;

      //initialize
      init();
      scrabbleLCDinit();
      setPlayerTurn(0);
      charnum = 0;

      TRISA = 0x00;     //Set up port A as an output port
      PORTA = 0x00;     //Erase port A
```

```
      OpenTimer0 (TIMER_INT_ON & T0_SOURCE_INT & T0_16BIT & T0_PS_1_1);
      //set TO_PS_1_16 for

                                    //for 30MHz clock
      OpenCapture1 (CAPTURE_INT_ON & C1_EVERY_RISE_EDGE);   //Set up
the capture ports
      OpenCapture2 (CAPTURE_INT_ON & C2_EVERY_RISE_EDGE);   //
      RCONbits.IPEN = 1;            //Turn on interrupt priority levels
      INTCONbits.GIEH = 1;    //Turn on high priority interrupts

      ////////////////////////////////////////////////////////////
      //The main FSM loop.
      // We originally implemented this as a big switch statment but
      // some compiler optimization was causing strange errors (like
      // the code from the wrong state would randomly be run) so we
      // implemented as an if-elseif instead.
      while (1)
      {
            //{{{ ENTER_PLAYERS_STATE
            //Get the number of players from the user
        if (state == ENTER_PLAYERS_STATE) {
                  //Check for a captured keypress
                  if (CCP1) {
                        setPos(3, 35);
                        key = getPadNum();
                        //Make sure it's a valid number of players
                        if (key < 5 && key > 1) {
                    numplayers = key;
                              //Draw the screen for the next state
                    setPos(2, 0);
                    putrsXLCD("   Enter the initials of each player.
");
                        putrsXLCD("      Press TIME to enter times. ");
                    dispScores(player_info, numplayers);
                    placeCursor(0,0);
                    state = ENTER_NAMES_STATE;  //Switch states
                        }
                        CLR_CCP1; //Clear the capture flag
                  }
        }
            //}}}
            //{{{ ENTER_NAMES_STATE
            // Get the initials of each player
            else if (state == ENTER_NAMES_STATE) {
                  //keypress captured
                  if (CCP1) {
                        //If enter is pressed store the current
character and advance
                        if (getPadNum()==ENTER_KEY) {
                              if (charentered == TRUE) {
                                    player_info[player].name[charnum] =
lastchar;
                                    charentered = FALSE;
                              }
                              charnum++;
```

```
                              //Advance to the next player if past 3
characters
                              if (charnum==3) {
                                    player++;
                                    if (player>numplayers-1) player=0;
                                    charnum=0;
                                    setPlayerTurn(player);
                              }
                              placeCursor(player, charnum);
                        }

                        //If backspace is pressed, erase the current
character and go back.
                        else if (getPadNum()==BACKSPACE_KEY) {
                              placeChar(player, charnum, ' ');
                              charnum--;
                              if (charnum<0) {
                                    player--;
                                    charnum=2;
                                    setPlayerTurn(player);
                              }
                              if (player<0) player = numplayers-1;
                              placeCursor(player, charnum);
                        }

                        //If not enter or backspace then this is a
character
                        //Write the char to the screen and buffer it
                        else {
                              charentered = TRUE;
                              lastchar = nextChar();
                              placeChar(player, charnum, lastchar);
                        }
                        CLR_CCP1; //Clear the keypress flag
                  }

                  //Button press captured
                  if (CCP2) {
                        CLR_CCP2; //Clear the button press flag
                        switch (button_press_val) {
                              case TIME_BUTTON: //Done entering names,
advance to the time state
                                    charnum = 0;
                                    clearLine(2);
                                    scrabbleLCDinitSetTimeMode();
                                    state = SET_TIMES_STATE;
                                    break;

                              case TIME_BYPASS_BUTTON:      //Done
entering names, advance straight to
                              case START_BUTTON:
      //gameplay. No timelimits will be entered
                                    initRunMode(0);
                                    scrabbleLCDinitRunMode();
                                    state = RUN_STATE;
                                    break;
```

```
                              case PLAYER1_BUTTON:          //Move the
cursor to player 1
                                    if (charentered == TRUE) {

      player_info[player].name[charnum] = lastchar;
                                          charentered = FALSE;
                                    }
                                    charnum = 0;
                                    player = 0;
                                    placeCursor(player, charnum);
                                    break;

                              case PLAYER2_BUTTON:          //Move the
cursor to player 2
                                    if (charentered == TRUE) {

      player_info[player].name[charnum] = lastchar;
                                          charentered = FALSE;
                                    }
                                    player = 1;
                                    charnum = 0;
                                    placeCursor(player, charnum);
                                    break;

                              case PLAYER3_BUTTON:          //Move the
cursor to player 3
                                    if (charentered == TRUE) {

      player_info[player].name[charnum] = lastchar;
                                          charentered = FALSE;
                                    }
                                    charnum = 0;
                                    player = 2;
                                    placeCursor(player, charnum);
                                    break;

                              case PLAYER4_BUTTON:          //Move the
cursor to player 4
                                    if (charentered == TRUE) {

      player_info[player].name[charnum] = lastchar;
                                          charentered = FALSE;
                                    }
                                    charnum = 0;
                                    player = 3;
                                    placeCursor(player, charnum);
                                    break;
                        }
                  }
            }
            //}}}
            //{{{ SET_TIMES_STATE
            // Set the time limits for each player
            else if (state == SET_TIMES_STATE) {
                  if (CCP1) {
                        setPos(3, TIME_ALL_MODE_TIME_POS + charnum);
                        key = getPadNum();
```

```
                        if (key==ENTER_KEY){
                              //No effect
                              //Must hit a player button to assign a
time
                        }
                        //Erase the current character and go back
                        else if (key==BACKSPACE_KEY) {
                              putchar(' ');
                              time_entry_string[charnum] = ' ';
                              charnum--;

                              if (charnum<0) { charnum = 0; }
                              setPos(3, TIME_ALL_MODE_TIME_POS +
charnum);
                              putchar(' ');
                        }
                        //If not enter or backspace, must be a number
                        //Display the number and buffer it
                        else {
                              if (charnum < 5) {
                                    putchar(key + '0');
                                    time_entry_string[charnum] = (key +
'0');
                                    charnum++;
                              }
                        }
                        CLR_CCP1; //Clear the keypress flag
                  }

                  //Button press captured
                  if (CCP2) {
                        CLR_CCP2; //Clear button press flag
                        switch (button_press_val) {
                              time_entry_string[charnum] = '\0'; //Null
terminator for string

                              //Assign the time limit to all players
                              case ALL_BUTTON:
                                    for (i = 0; i < numplayers; i++){
                                          player_info[i].timelimit =
atoi(time_entry_string);
                                    }
                                    redraw = TRUE;
                                    break;

                              //Assign the time limit to player 1
                              case PLAYER1_BUTTON:
                                    player_info[0].timelimit =
atoi(time_entry_string);
                                    redraw = TRUE;
                                    break;

                              //Assign the time limit to player 2
                              case PLAYER2_BUTTON:
                                    player_info[1].timelimit =
atoi(time_entry_string);
```

```
                                    redraw = TRUE;
                                    break;

                              //Assign the time limit to player 3
                              case PLAYER3_BUTTON:
                        if (numplayers < 3) break;
                                    player_info[2].timelimit =
atoi(time_entry_string);
                                    redraw = TRUE;
                                    break;

                              //Assign the time limit to player 4
                              case PLAYER4_BUTTON:
                        if (numplayers < 4) break;
                                    player_info[3].timelimit =
atoi(time_entry_string);
                                    redraw = TRUE;
                                    break;

                              //Changed your mind, don't want to enter
times. Bypass timing altogether
                              //and advance right into gameplay
                              case TIME_BYPASS_BUTTON:
                                    for (i=0; i < numplayers; i++) {
                                          player_info[i].timelimit = 0;
                                    }
                              //Don't break, still need to start
gameplay
                              case START_BUTTON:
                                    initRunMode(0);
                                    scrabbleLCDinitRunMode();
                                    state = RUN_STATE;
                        } //end switch(button_press_val)

                        //display new time limits on line 2
                        if (redraw) {
                              charnum = 0;
                              setPos(3, TIME_ALL_MODE_TIME_POS);
                              putrsXLCD("     ");
                              redraw = FALSE;
                              clearLine(2);
                              for (i = 0; i < numplayers; i++) {
                        if (player_info[i].timelimit == 0)
                            continue; //No time for some players
                                    setPos(2, i*10 + 3);
                                    putrsXLCD("     ");
                                    itoa(player_info[i].timelimit,
buff);
                                    setPos(2, i*10 + 3);
                                    putsXLCD(buff);
                              }
                              setPos(3, TIME_ALL_MODE_TIME_POS +
charnum);
                        }
                  }
            }
//}}}
```

```
            //{{{ RUN_STATE
            // Perform actual gameplay
            else if (state == RUN_STATE) {
                  redraw = TRUE;    //Need to redraw the time

                  //Check if the flashing led is on.
            if (flashon) {
                        //Alternate between on for FLASH_RATE_ON and
off for FLASH_RATE_OFF
                if (flashval == 1 && currentflash *
SECONDS_PER_TIMECOUNT > FLASH_RATE_ON) {
                    currentflash = 0;
                    flashval = 0;
                    PORTA = 0x00;
                }
                else if (flashval == 0 && currentflash *
SECONDS_PER_TIMECOUNT > FLASH_RATE_OFF) {
                      currentflash = 0;
                      flashval = 1;
                      PORTA = 0b00000001;
                  }
            }
            else {
                        //Just in case
                  PORTA=0x00;
            }

                  //If this player has a time limit and the timer is
not paused then
                  //check all time constraints and update the display
                  if ( haslimit && !pause && ((int)(timelimit -
timecount*SECONDS_PER_TIMECOUNT)) < timeleft ) {
                        timeleft = timelimit -
timecount*SECONDS_PER_TIMECOUNT;

                        //Turn the flasher on when there are
FLASH_START seconds left
                if (timeleft <= FLASH_START && !flashon) {
                    flashon = TRUE;
                    currentflash = 0;
                    flashval = 1;
                    PORTA = 1;
                }

                        //Out of time, perform necessary operations
                        if (timeleft <= 0) {
                              do_outoftime();
                              redraw = FALSE;
                              //Redraw once more now
                              dispTimeSecs(player, timeleft);

      dispTimeBarRatio((float)timeleft/(float)timelimit);
                              timeleft = 0;
                              //If we haven't buzzed already, start the
buzzer
                              if (!buzzed) {
                                    state = BUZZ_STATE;
```

```
                                    buzzed = TRUE;
                                    buzzcount = 0;
                              }
                              //We need to wait for score entry before
we advance to the next player
                        }

                        //Redraw the time status
                        if (redraw == TRUE) {
                              redraw = FALSE;
                              dispTimeSecs(player, timeleft);

      dispTimeBarRatio((float)timeleft/(float)timelimit);
                        }
                  }

                  //Key press capture
                  if (CCP1) {
                        setPos(2, ENTER_SCORE_VALUE_POS + 1 + charnum);
                        key = getPadNum();

                        //If enter is pressed, apply the score to the
current player
                        if (key==ENTER_KEY && charnum > 0){
                              score_entry_string[charnum] = '\0';
                              player_info[player].last_score =
atoi(score_entry_string);
                    if (negscore) { //Adjust for negative score entry
                        player_info[player].last_score = -
player_info[player].last_score;
                        }
                                player_info[player].score +=
player_info[player].last_score;
                    negscore = FALSE;
                              dispScores(player_info, numplayers);
//Update the display
                              initRunMode((player+1)%numplayers);
                              scrabbleLCDinitRunMode();
                        }

                        //If backspace is pressed erase the last digit.
If there are no more digits
                        //then toggle the negative operator
                        else if (key==BACKSPACE_KEY) {
                              charnum--;
                              score_entry_string[charnum] = ' ';
                              setPos(2, ENTER_SCORE_VALUE_POS + 1 +
charnum);
                              putchar(' ');
                              if (charnum<0) {  //Toggle negative
                        setPos(2, ENTER_SCORE_VALUE_POS + 1 + charnum);
                        if (negscore) {
                            negscore = FALSE;
                            putchar(' ');
                        }
                        else
                        {
```

```
                        negscore = TRUE;
                        putchar('-');
                    }
                    putchar(' ');
                    charnum = 0;
                }
                        setPos(2, ENTER_SCORE_VALUE_POS + 1 +
charnum);
                    }

                    //Otherwise, if the key isn't enter it's a
digit
                    //Append the digit to the current number
                    else if (key != ENTER_KEY) {
                        if (charnum < 5) {
                            putchar(key + '0');
                            score_entry_string[charnum] = (key
+ '0');
                            charnum++;
                        }
                    }

                    CLR_CCP1; //Clear the keypress flag
                }

                //Button press event
                if (CCP2) {
                    CLR_CCP2;//Clear the button press flag
                    switch (button_press_val) {

                        //The start button unpauses if paused
                        case START_BUTTON:
                            unpause();
                            break;

                        //The pause button pauses if not paused
                        case PAUSE_BUTTON:
                            dopause();
                            break;

                        //Apply the currently entered score to
player 1.
                        //If it is player 1's turn advance to the
next player
                        case PLAYER1_BUTTON:
                    if (charnum == 0) break;
                            score_entry_string[charnum] = '\0';
                            player_info[0].last_score =
atoi(score_entry_string);
                    if (negscore) {
                        player_info[0].last_score = -
player_info[0].last_score;
                    }
                            player_info[0].score +=
player_info[0].last_score;
                            redraw = TRUE;
                    negscore = FALSE;
```

```
                        charnum = 0;      //Start a new string
                        if (player==0) {  //Advance to next player
                                  initRunMode((player+1)%numplayers);
                                    scrabbleLCDinitRunMode();
                        }
                                    break;

                              //Apply the currently entered score to
player 2.
                              //If it is player 2's turn advance to the
next player
                              case PLAYER2_BUTTON:
                        if (charnum == 0) break;
                                    score_entry_string[charnum] = '\0';
                                    player_info[1].last_score =
atoi(score_entry_string);
                        if (negscore) {
                            player_info[1].last_score = -
player_info[1].last_score;
                        }
                                    player_info[1].score +=
player_info[1].last_score;
                                    redraw = TRUE;
                        negscore = FALSE;
                        charnum = 0;
                        if (player==1) {
                                  initRunMode((player+1)%numplayers);
                                    scrabbleLCDinitRunMode();
                        }
                                    break;

                              //Apply the currently entered score to
player 3.
                              //If it is player 3's turn advance to the
next player
                              case PLAYER3_BUTTON:
                        if (charnum == 0) break;
                        if (numplayers < 3) break; //Check that there
are indeed at least 3 players
                                    score_entry_string[charnum] = '\0';
                                    player_info[2].last_score =
atoi(score_entry_string);
                        if (negscore) {
                            player_info[2].last_score = -
player_info[2].last_score;
                        }
                                    player_info[2].score +=
player_info[2].last_score;
                                    redraw = TRUE;
                        negscore = FALSE;
                        charnum = 0;
                        if (player==2) {
                                  initRunMode((player+1)%numplayers);
                                    scrabbleLCDinitRunMode();
                        }
                                    break;
```

```
                              //Apply the currently entered score to
player 14
                              //If it is player 4's turn advance to the
next player
                              case PLAYER4_BUTTON:
                        if (charnum == 0) break;
                        if (numplayers < 4) break;    //Check that
there are 4 players
                                    score_entry_string[charnum] = '\0';
                                    player_info[3].last_score =
atoi(score_entry_string);
                        if (negscore) {
                            player_info[3].last_score = -
player_info[2].last_score;
                        }
                                    player_info[3].score +=
player_info[3].last_score;
                                    redraw = TRUE;
                        negscore = FALSE;
                        charnum = 0;
                        if (player==3) {
                                  initRunMode((player+1)%numplayers);
                                    scrabbleLCDinitRunMode();
                        }
                                    break;

                              //Don't want to use the timer anymore
                              //Eliminate the timer for the current
player as well as all other players
                    case TIME_BYPASS_BUTTON:
                        for (i=0; i < numplayers; i++) {
                                          player_info[i].timelimit = 0;
                                    }
                        nolimit();
                        haslimit = FALSE;
                        break;

                              //Restart the timer for the current
player
                    case RESET_TIMER_BUTTON:
                        initRunMode(player);
                        break;
                        }

                        //Redraw the score
                        if (redraw) {
                              redraw = FALSE;
                              dispScores(player_info, numplayers);
                              clearLine(2);
                              if (pause) {
                                    setPos(2,0);
                                    putrsXLCD("PAUSED");
                              }
                              setPos(2, ENTER_SCORE_VALUE_POS - 13);
                              if (negscore)
                                    putrsXLCD("Enter Score:-");
                              else
```

```
                                    putrsXLCD("Enter Score:");

                              setPos(2, ENTER_SCORE_VALUE_POS + 1);
                              charnum = 0;
                        }
                  }
            }
            //}}}
            //{{{ BUZZ_STATE
            // Sound the buzzer for some time then return to the run
state
            else if (state == BUZZ_STATE) {
                  //Turn buzzer on
                  PORTA |= 0b00000100;
                  timeleft = BUZZ_LENGTH -
buzzcount*SECONDS_PER_TIMECOUNT;
                  if (timeleft <= 0) {
                        //buzz time up, turn buzzer off
                        PORTA &= 0b11111011;
                        timeleft = 0;
                        state = RUN_STATE;
                  }
            }
            //}}}
      } // end while
}

//Initialize the player info structs
void init(void) {
      int i;
      for (i=0; i<numplayers; i++) {
            player_info[i].name[0] = '\0';
            player_info[i].timelimit = 0;
            player_info[i].last_score = 0;
            player_info[i].score = 0;
            player_info[i].bingos=0;
      }
}

//Initialze the gameplay state for the current player
void initRunMode(int playernum) {
      charnum = 0;
      player = playernum;
    flashticks = 0;
    flashon = FALSE;
      setPlayerTurn(player);
      timecount=0;
      timelimit = player_info[playernum].timelimit;
      timeleft = player_info[playernum].timelimit;
    if (timelimit == 0)
        haslimit = FALSE;
    else
        haslimit = TRUE;
      buzzed = FALSE;
      un_outoftime();
}
```

```
//Turn the pause flag on
void dopause(void) {
      pause = TRUE;
      setPos(2, 0);
      putrsXLCD("PAUSED");
}

//Turn the pause flag off
void unpause(void) {
      pause = FALSE;
      //setPos(2, TIME_START_COL + 7);
      setPos(2, 0);
      putrsXLCD("      ");
}

//Toggle the pause flag
void togglepause(void) {
      pause = 1-pause;
      setPos(2, 0);
      if(pause==TRUE) {
            putrsXLCD("PAUSED");
      } else {
            putrsXLCD("      ");
      }
}

//If out of time, write it to the lcd
void do_outoftime(void) {
      outoftime = TRUE;
      setPos(3, 20);
      putrsXLCD("OUT OF TIME");
}

//Clear the out of time text
void un_outoftime(void) {
      outoftime = FALSE;
      setPos(3, 20);
      putrsXLCD("           ");
}

//Set up the screen for a player with no time limit
void nolimit(void) {
    setPos(3, 19);
    putrsXLCD("NO TIME LIMIT");
}
```

keypad.h

```
//Prototypes
void initKeypads(void);                         //Initialize
unsigned int registerPadPress(void);      //Store a press
unsigned int getPadNum(void);             //Return number pressed
char nextChar(void);                            //Return char entered
```

keypad.c

```
#include <p18cxxx.h>
#include <p18f452.h>
#include "keypad.h"

unsigned int _key_last = 99;  //Last key pressed
char _key_prev = 0;                        //Previous key pressed

//Map button codes to the right key number
unsigned int keymap[] = {1, 2, 3, -1, 4, 5, 6, -1, 7, 8, 9, -1, 10, 0,
11};

//Set up port D to take keypad input
void initKeypads(void) {
      TRISD = 0xFF;
}

//Get the last key pressed
unsigned int getPadNum(void) {
      return _key_last;
}

//Store the current key being pressed
unsigned int registerPadPress(void) {
      unsigned int data = PORTD & 0x0F;   //Grab data
      char lastpress = _key_last;
      _key_last = 99;   //Default value

/*    switch (data) {
            case 0: _key_last = 1; break;
            case 1: _key_last = 2; break;
            case 2: _key_last = 3; break;
            case 4: _key_last = 4; break;
            case 5: _key_last = 5; break;
            case 6: _key_last = 6; break;
            case 8: _key_last = 7; break;
            case 9: _key_last = 8; break;
            case 10: _key_last = 9; break;
            case 12: _key_last = 10; break;
            case 13: _key_last = 0; break;
            case 14: _key_last = 11; break;
      }           */
      _key_last = keymap[data];

      if (_key_last != lastpress)
            _key_prev = 0;    //Used to determine desired character
      return _key_last;
}

//Get the character desired from the keypad
//The keypad works like a phone number pad
//By hitting the same key over and over, you scroll through
//The characters assigned to that key
char nextChar(void) {
      switch (_key_last) {
            case 2:
```

```
                  _key_prev =
(_key_prev==0||_key_prev=='C')?'A':_key_prev+1;
                  break;
            case 3:
                  _key_prev =
(_key_prev==0||_key_prev=='F')?'D':_key_prev+1;
                  break;
            case 4:
                  _key_prev =
(_key_prev==0||_key_prev=='I')?'G':_key_prev+1;
                  break;
            case 5:
                  _key_prev =
(_key_prev==0||_key_prev=='L')?'J':_key_prev+1;
                  break;
            case 6:
                  _key_prev =
(_key_prev==0||_key_prev=='O')?'M':_key_prev+1;
                  break;
            case 7:
                  _key_prev =
(_key_prev==0||_key_prev=='S')?'P':_key_prev+1;
                  break;
            case 8:
                  _key_prev =
(_key_prev==0||_key_prev=='V')?'T':_key_prev+1;
                  break;
            case 9:
                  _key_prev =
(_key_prev==0||_key_prev=='Z')?'W':_key_prev+1;
                  break;
            default:
                  _key_prev = ' ';
      }
      return _key_prev;
}
```

mylcd.h

```
#ifndef __mylcd_h
#define __mylcd_h

//oscillator speed, in KHz. should ideally be defined by including file
#define OSC_SPD_LCD 1000.0

#define DELAY_PER_MS (OSC_SPD_LCD/4000.0)

void initLCD( void );                                   //Initialize
void DelayFor18TCY( void );                             //18 instr delay
void DelayPORXLCD( void );                              //15ms delay
void DelayXLCD( void );                                 //8ms delay
```

```
void setLine(int i);                                    //move cursor
here
void setPos(int row, int col);                          //move cursor
here
void clearLine( int lineNum );                          //clear this line
void delay ( unsigned int millisecs );          //delay for this long
void delayMicro( unsigned int microsecs );      //delay for this long
void writeFloatToLCD ( float f );               //write a float value
void putchar( char c ); //same as putcXLCD but adds a 160us delay
beforehand

#endif
```

mylcd.c

```
#include <delays.h>
#include "mylcd.h"
#include <xlcd.h>
#include <stdlib.h>
#include <string.h>

void initLCD( void ) {
      // configure external LCD
      DelayPORXLCD();
      OpenXLCD( FOUR_BIT & LINES_5X7, 0 );
      delayMicro(160);
      OpenXLCD( FOUR_BIT & LINES_5X7, 1 );
      delayMicro(160);
      WriteCmdXLCD(CURSOR_OFF);
}

//18 instruction delay
void DelayFor18TCY( void )
{
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
      Nop();
}

//15-20ms delay
void DelayPORXLCD( void )
{
      Delay1KTCYx(20 * DELAY_PER_MS); //Delay of 15ms
      return;
}

//8ms delay
```

```
void DelayXLCD( void )
{
      Delay1KTCYx(8 * DELAY_PER_MS); //Delay of 8ms
      return;
}

//Put a character on the lcd
void putchar( char c ) {
      delayMicro(160);
      putcXLCD(c);
}

//accepts 0 to 3, sets cursor to that line
void setLine(int i) {

      int LINE[2];

      LINE[0] =0x00;
      LINE[1] =0x40;

      CS_PIN = (i % 4)/2;     //select chip 2 if line = 2 or 3
      WriteCmdXLCD(0x80 | LINE[i%2]);
      return;
}

//Set absolute position of cursor
void setPos(int row, int col) {

      int LINE[2];
      int ddr_addr;

      LINE[0] =0x00;
      LINE[1] =0x40;

      CS_PIN = (row % 4)/2;   //select chip 2 if row = 2 or 3
      ddr_addr = LINE[row%2] + col;
      delayMicro(160);
      WriteCmdXLCD(0x80 | ddr_addr);
      return;
}

//Clear a line on the lcd
void clearLine( int lineNum ) {
      setLine(lineNum);
      putrsXLCD("                                        ");
      setLine(lineNum);
      return;
}


void delayMicro( unsigned int microsecs ) {
      unsigned int delayCount = (int)((double)microsecs * DELAY_PER_MS
/ ((double)10));
      //Delay functions take char as input
      while (delayCount > 255) {
            Delay10TCYx(256);
            delayCount -= 256;
```

```
	}
	Delay10TCYx(delayCount);
	return;
}

void delay (unsigned int millisecs) {
	unsigned int delayCount = (int)((double)millisecs * DELAY_PER_MS
/ ((double)10));
	//Delay functions take char as input
	while (delayCount > 255) {
		Delay10KTCYx(256);
		delayCount -= 256;
	}
	Delay10KTCYx(delayCount);
	return;
}


//expensive, hacked version -- may want to eventually optimize this
void writeFloatToLCD ( float f ) {
	int fint = (int)f;
	int fint2;
	char str[10];
	char str2[4];
	char decimalpoint[2] = ".";
	char zeroes2[3] = "00";
	char zeroes1[2] = "0";

	//output >1 part
	itoa(fint, str);
	//while(BusyXLCD());
	//putsXLCD(str);

	//output decimal point
	while(BusyXLCD());
	//strcat(str, ".");
	strcat(str, decimalpoint);
	//putrsXLCD(".");

	//output 3 decimal places
	fint2 = (int)(f * 1000);
	fint2 -= fint*1000;
	if (f < 10) {
		strcat(str,zeroes2);
	} else if (f < 100) {
		strcat(str,zeroes1);
	}

	itoa(fint2, str2);
	strcat(str, str2);

	while(BusyXLCD());
	putsXLCD(str);
}
```

```
/*
void waitAndWrite(int i, const rom char *buffer) {
            while( BusyXLCD() );    //wait until LCD is not busy
            setLine(i);
            putrsXLCD(buffer);
}
*/
```

scrabblelcd.h

```
#define PLAYER_NAME_LENGTH 8
#define NUM_DIGITS_SCORE 5          //total score
#define NUM_DIGITS_BINGOS 5         //Actual prev score

#define TIME_START_COL 0                  //Where to start displaying
time
#define ENTER_SCORE_VALUE_POS 21    //Where to enter score
#define TIME_BAR_LENGTH 30
#define TIME_ALL_MODE_TIME_POS 32

#include "../mylcd/mylcd.h"
#include "../capturetest/capturetest.h"
#include <xlcd.h>
#include <stdlib.h>
#include <string.h>

//diff lcd init for each mode
void scrabbleLCDinit(void);
void scrabbleLCDinitSetTimeMode(void);
void scrabbleLCDinitReadyMode(void);
void scrabbleLCDinitRunMode(void);

void placeCursor(int playernum, int charnum);
void placeChar(int playernum, int charnum, char c);

//should be called after position set, because
// it is screen dependent.
void cursorOn(void);
void cursorOff(void);

//Display the time or time bar
void dispTimeSecs(int playernum, int secs);
void dispTime(int playernum, int mins, int secs);
void dispTimeBar(int mins, int secs, int totalmins, int totalsecs);
void dispTimeBarRatio(float ratio);

//Display all player scores
void dispScores(player_info_t *players, int numplayers);

//Display scores for certain player
void setScore(int playernum, int score);  //Total score
void setBingos(int playernum, int bingos);      //Actually last score

//Set ths player's turn
void setPlayerTurn(int playernum);
```

scrabblelcd.c

```
#include "./scrabblelcd.h"

//Initialize the lcd display
void scrabbleLCDinit() {
      initLCD();
      setPos(0,0);
      putrsXLCD("         |");
      setPos(0,10);
      putrsXLCD("         |");
      setPos(0,20);
      putrsXLCD("         |");
      setPos(0,30);
      putrsXLCD("         |");

      setPos(1,1);
      putrsXLCD("        |");
      setPos(1,11);
      putrsXLCD("        |");
      setPos(1,21);
      putrsXLCD("        |");
      setPos(1,31);
      putrsXLCD("        |");

      setPos(2, 0);
      putrsXLCD("Enter the number of players (2-4): ");
    setPos(2, 35);
}

//Initialize the set time state on the lcd
void scrabbleLCDinitSetTimeMode(void) {
      clearLine(2);
      clearLine(3);
      putrsXLCD("   Enter Timelimit in Seconds:");
}

//Initialize the ready state on the lcd
void scrabbleLCDinitReadyMode(void) {
      clearLine(2);
      putrsXLCD("Ready.");
}

//Initialize run mode on the lcd
void scrabbleLCDinitRunMode(void) {
      clearLine(2);
      clearLine(3);
      setPos(2, ENTER_SCORE_VALUE_POS - 13);
      putrsXLCD("Enter Score:");
}

//For name entry, place the cursor in the appropriate spot
//playernum goes from 0 to 3
//charnum goes from 0 to PLAYER_NAME_LENGTH-1
void placeCursor(int playernum, int charnum) {
```

```
	setPos(1, playernum*10 + charnum);
	cursorOn();
}

//For name entry, place the character in the appropriate spot
void placeChar(int playernum, int charnum, char c) {
	setPos(1, playernum*10 + charnum);
	cursorOn();
	putchar(c);
	setPos(1, playernum*10 + charnum);
}

//Turn the cursor on
void cursorOn(void){
	delayMicro(160);
	WriteCmdXLCD(CURSOR_ON);
}

//Turn the cursor off (doesn't work)
void cursorOff(void){
	delayMicro(160);
	WriteCmdXLCD(CURSOR_OFF);
}

//Given the time in seconds, display it
void dispTimeSecs(int playernum, int secs) {
	dispTime(playernum, secs/60, secs%60);
}

//Display the time left
void dispTime(int playernum, int mins, int secs) {
	char buff[10];
	setPos(3,0);
	putrsXLCD("      "); //6 spaces cleared
	setPos(3,0);
	itoa(mins, buff);
	putsXLCD(buff);
	itoa(secs,buff);
	putchar(':');
	if (secs < 10) putchar('0');
	putsXLCD(buff);
}

//Display the time bar given the time left and total time
void dispTimeBar(int mins, int secs, int totalmins, int totalsecs) {
	float ratio = ((60*mins + secs) / (60 * totalmins + totalsecs));
	dispTimeBarRatio(ratio);
}

//Display the time bar with only a certain percentage filled in
void dispTimeBarRatio(float ratio) {
	int i, length;
	setPos(3,((40-TIME_START_COL-6-
TIME_BAR_LENGTH)/2)+TIME_START_COL+6);
	cursorOff();
	length = (int)(ratio * TIME_BAR_LENGTH);
	for(i=0; i<length; i++) {
```

```
            putchar(0xff); //block
      }
      for(i=length; i<TIME_BAR_LENGTH; i++) {
            putchar('-'); //dash
      }
}

//Display the scores for all players
void dispScores(player_info_t *players, int numplayers) {
      int i;
      for (i = 0; i < numplayers; i++) {
            setScore(i, players[i].score);
            setBingos(i, players[i].last_score);
      }
}

//Display the score for this player
void setScore(int playernum, int score) {
      int i;
      char buff[10];
      setPos(1, playernum*10 + 4);
      cursorOff();
      for (i=0; i<NUM_DIGITS_SCORE; i++) putchar(' ');
      setPos(1, playernum*10 + 4);
      itoa(score, buff);
      putsXLCD(buff);
}

//We no longer display bingos
//this instead displays the last score
void setBingos(int playernum, int bingos) {
      int i;
      char buff[10];
      setPos(0, playernum*10 + 4);
      cursorOff();
      for (i=0; i<NUM_DIGITS_BINGOS; i++) putchar(' ');
      setPos(0, playernum*10 + 4);
      itoa(bingos, buff);
      putsXLCD(buff);
}

//Update the asterisk indicating the player's turn
void setPlayerTurn(int playernum) {
      int i;
      for (i=0; i<4; i++) {
            setPos(0, 1+i*10);
            putchar(' ');
      }
      setPos(0, 1+playernum*10);
      putchar('*');
      setPos(0, 3+playernum*10);
}
```