Microprocessor Systems
Final Project: ANSI Target Game

Alvin Chia Nick Choi Samuel Deslandes
Shanmugam Thiagarajan

12/13/16

1 Introduction

The overall goal of this project was to combine the content of several lab assignments in order to create an unique final product using the 8051. The idea that was pursued was an ANSI target game which utilized a keypad, a LCD display and an analog joystick module, and utilizes topics such as ANSI escape sequences, ADC conversions, timers, and interrupts.

To begin, a menu screen needed to be created in order to give the user options to select before starting their game. Ultimately, the user would be given four options on the menu that represented the difficulty levels for the game. The number of targets and maximum time allowed on the level would be adjusted depending on which option was selected. The selections would be made using the keypad. Also displayed is a brief "How to play" statement. The player uses the joystick to move the cursor around on the screen, and presses the joystick to "shoot" the targets.

Once the difficulty has been chosen, the game starts. Targets will be randomly drawn on the terminal display with a bullseye at the center of the target. Hitting the bullseye is worth 100 points, while any other zone on the target is worth only 10 points. Once all of the targets have been drawn, the timer begins to count down on the LCD display. The score is also displayed on the LCD screen and is updated whenever the user hits a new target.

If the user completes the game before time runs out, they will be brought to a victory screen and will have the option to press the '#' key in order to select a new difficulty. In the case that the user has only 5 seconds left, a buzzer will sound to alert the user of the remaining time. If the player fails to hit all of the targets in time, they will be brought to a game over screen, but will still have the option to press '#' to return to the main menu and keep playing.

2 Methods

2.1 Software

The program for this project was organized into several functions. The 'main()' function handles initializations and repeatedly calls the 'playGame()' function on an infinite loop. 'playGame()' handles the main game play operations such as the main menu, moving the cursor, keeping track of time and score and displaying them on the LCD screen, and checking win and lose game states. Other functions such as 'drawTarget()', 'checkTarget()', and 'eraseTarget()' handle the display and hit detection aspects of targets.

2.1.1 Main() and Configurations

'Main()' begins by disabling the watchdog timer and calling the various initialization functions. In 'PORT_INIT()' the crossbar is configured such that UART0 is enabled in the 'XBR0' SFR, /INT0 and /INT1 are routed to port pins in 'XBR1', and in 'XBR2' the crossbar is enabled with weak pull-up. Next global interrupts and external interrupt 1 is enabled using the bit addressable keywords 'EA' and 'EX1', respectively. /INT1 associated with the button on the joystick is also set to be triggered on falling edge by setting 'IT1' to 1. Lastly the ports are configured. P0.0 and P2.0 are set for push-pull operation for use by

TX and a buzzer respectively. Port3 is used by the keypad, and is configured such that its high nibble in push-pull mode with digital LO output, and its low nibble is in open-drain mode with high impedance.

'SYSCLK_INIT()' initializes the system clock to use just the external oscillator with a frequency of 11.0592Mhz. In 'UART0_INIT()' Timer1 is used to generate a baud rate 115200. To accomplish this Timer1 is configured using the 'TMOD' SFR to be an 8-bit counter with auto-reload. The auto-reload value for the required baud rate is `0xFA`. A reference reload values for each timer, SYSCLK, and desired baud rate can be found on page 295 of the C8051 manual[3]. The UART itself is configured to be in mode 1 using the 'SCON0' SFR. In 'ADC0_INIT()', 'ADC0CF' is set to `0xBF`, resulting in a gain term of $\frac{1}{2}$, and a SARclk of 230400Hz. The can be calculated as follows:

$$ADC0SC = \frac{SYSCLK}{2 \times SARclk} - 1$$

where ADC0SC is bits 3–7 of ADC0CF

The internal reference buffer and bias generator are enabled, producing a voltage of 2.4V, which is used as a reference for conversions. All inputs are also treated as single-ended inputs.

Finally, Timer0 is used to accurately count game time in measurements of a tenth of a second. It's configured as a 16-bit counter using SYSCLK/48 as a base. For an overflow of this timer to occur every tenth of a second, the timer must be set to start counting at `0xA600` on initialization as well as after every overflow. This can be calculated as follows:

$$\frac{1}{48} \times \frac{11059200\text{counts}}{1sec} = \frac{X\text{counts}}{0.1\text{sec}}$$

Solving for X gives $X = 23040$. Since 23040 counts elapse in a tenth of a second, the timer must start counting from $2^{16} - 23040 = 42496$. This corresponds to `0xA600` in hex. Since Timer0 is a 16-bit counter, this must be split between its high and low registers, with TH0 = `0xA6` and TL0 = `0x00`.

2.1.2 playGame()

The 'playGame()' function starts by initializing the cursor position variables 'xPos' and 'yPos' to 1 because indexing for the terminal starts at 1 rather than 0, then resets the terminals main parameters, clears the screen, and calls the 'Menu()' function. The terminal can be controlled by the software via ANSI escape sequences. Table 3 in the Appendicies section below show the sequences used in this program and their functions. The 'Menu()' function displays the four difficulty option available to the player (Easy, Medium, Hard, and GG) to the terminal, and awaits user input from the keypad. This is done using the 'getkeychar()' function which continuously polls a flag variable set by the keypad vector /INT0 ISR and returns the character pressed by the user.

The previously mentioned /INT0 ISR is connected to the hardware on P0.2, and is triggered on a falling edge. Once /INT0 has been triggered, the ISR must decode the signals on port 3 to determine which key has been pressed. The ISR starts by temporarily disabling interrupts on /INT0, then sets the flag variable used in 'getkeychar()'. The rows of the

keypad are decoded first, but before this the lower nibble of P3 is stored in a variable for later use. Once this is done, the output pins on P3 (high nibble) is set to `0x8`, `0x4`, `0x2`, then `0x1`, corresponding to rows 1–4 on the keypad respectively. After each change a small delay is implemented and the input pins on P3 (low nibble) are evaluated to see if they equal `0xF`; if so, then that is the row that had been selected. Columns are decoded using the original value of the low nibble of P3 stored before the decoding process. In this value all of the bits with be 1's, except for one; the selected column corresponds with the location of this bit, which can be determined by evaluating it to `0x07`, `0x0B`, `0x0D`, and `0x0E`. Once the selected column has been identified, a character can be assigned to a global variable "asciichar", which is returned by 'getkeyhar()'. Figure 2 in the appendices section can be used as a reference for which row and column intersection corresponds to which character. Once "asciichar" has been assigned P3 is reset to `0x0F`, and after another pause /INT0 is re-enabled and the program returns from the ISR. Once the user input is returned, a switch/case block is used to select the appropriate level. Which difficulty is selected dictates how much time the player has, as well as how many targets will be drawn to the screen. The table below shows the parameters for each difficulty.

Difficulty	Time (s)	# of Targets
Easy	60	5
Medium	45	7
Hard	30	9
GG	15	11

Table 1: Game level parameters

Once these are set, the 'drawTarget()' function is called, and the program returns to the 'playGame()' routine. The details of 'drawTarget()' will be discussed in the next section.

Back in the 'playGame()' function, the program then resets the cursor to its home position (1,1), starts Timer0, and enters an infinite loop. In this loop is where the game is played. First 'readADC()' is called, which performs analog-to-digital conversions on the x and y inputs on the joystick, as well as from a potentiometer used to control the joystick sensitivity. ADC0 on the 8051 accepts up to 8 inputs, using a multiplexer to select a channel for conversion. The joystick y-axis is on channel 0 (AIN0.0), the x-axis on channel 1 (AIN0.1), and the potentiometer wiper on channel 3 (AIN0.2); the aforementioned multiplexer can be controlled via the 'AMX0SL' SFR. An A-D conversion is initiated by setting 'AD0BUSY'. The conversion process ends when the conversion interrupt flag goes digital LO, at which point the converted value can be read from the 'ADC0' register. It is important to implement a small delay after changing channels on the multiplexer to allow the system to stabilize.

Once these values have been read, the x and y-axis values are compared against threshold values, and the sensitivity value from the potentiometer is used to calculate a wait time for the while loop. If the x/y values exceed the thresholds, the cursor moves in the appropriate direction on the terminal, and the cursor position variables are updated. Checks are made to ensure that the cursor position does not exceed the playable range on the terminal[1]. The

[1]Playable region is defined by terminal row and column settings

cursor position is changed using ANSI escape sequences. The sensitivity value is mapped to wait times ranging from 655350–27035. This wait time is the time between each "frame" of the game. As such, the longer the wait time, the slower the cursor speed and the more control the player has over the cursor; conversely, the smaller the wait time the faster the cursor moves, allowing the player to reach more targets in a shorter span of time. The mapping from the converted sensitivity value to a wait time is as follows:

$$\text{Wait\_time} = -11 \times \text{sensitivity} + 65535$$

This wait time has no affect on the time keeping mechanism of the game, as this is done via interrupts. Every tenth of a second the Timer0 ISR is triggered and a counter variable representing the number of elapsed tenths of seconds is incremented. In the 'playGame()' infinite loop this counter is evaluated and if it equals 10, it is reset and the time variable set based on difficulty is decremented.

In order to give user feedback, the amount of time remaining as well as the player's score is displayed on the first and second lines on the LCD respectively. All LCD functions can be found in the "LCD.c" file as well as its header file, which should be included. Before anything is printed to the LCD, the screen is cleared using 'lcd_clear()'. Next, using 'sprintf()', the first line is prepared to be sent to the LCD. Calling 'lcd_goto(0)' moves the cursor to the top left corner and 'lcd_puts()' will print the prepared "string" to the LCD. The same is done for the second line, using 'lcd_goto(`0x40`)' to move the cursor into the correct position. When there is 5 seconds remaining, output pin P2.0 is set to digital HI, which activates a buzzer; at 4 seconds remaining P2.0 is set back to digital LO and the buzzer turns off.

The other remaining tasks of the 'playGame()' are checking end game conditions. If the player has hit all of the targets, 'WinScreen()' is called and a victory screen is printed to the terminal. If the player fails to eliminate all of the targets before time runs out, the 'LoseScreen()' function is called an a game over screen is displayed. In both cases, the program breaks out of the infinite loop within 'playGame()', and using 'getkeychar()', the program will wait until the player presses the '#' character on the keypad, at which point the function returns to 'main()', where 'playGame()' is called yet again.

2.1.3 Target handling

The functions 'drawTarget()', 'checkTarget()', and 'eraseTarget()' deal with the displaying of the targets onto the terminal as well as hit detection. As mentioned before, 'drawTarget()' is called after a difficulty selection is made within the 'Menu()' function. This function draws the required number of target onto the terminal at random positions. The function starts by clearing the terminal, then for each target to be drawn, a random row and column is chosen using the 'rand()' function. These coordinates represent the center of the target and are added to arrays which store the coordinates of the targets. To draw the targets, the program moves the cursor to the position one above and to the left of the center, and prints three blocks[2] in white. In a similar manner 3 blocks are printed on the two rows below the first, creating a 3x3 box of block characters. The cursor is the moved to the center of the box and a red "bullseye" block is printed.

[2]U+2588, 219 on the extended ASCII table, cannot be displayed in LaTeX

When generating random locations for the center of targets, certain precautions are made to ensure that the entire target can be displayed, and that no two targets overlap. For the first case, random numbers are generated within the range from 2—$(\#COLS - 2)$ for the columns, and 2—$(\#ROWS - 2)$ for the rows, where #ROWS and #COLS represent the number of rows and columns the terminal displays. This is done using modular arithmetic using $row = rand()\%(\#ROWS - 2) + 2$ and $col = rand()\%(\#COLS - 2) + 2$. To ensure that no overlap between targets, each time a random position is generated, the program check this position against the positions of all the previously drawn targets and makes sure that the 3x3 boxes to be drawn are not in range of each other.

The button the on joystick is connected to /INT1 on the 8051; as such when the button is pressed the associated 'stickPress()' ISR function is called. This ISR calls the 'checkTarget()' hit detection function, passing the current position of the cursor as its arguments. This function iterates through the arrays of target center positions and compares then to the passed in position. If passed in position is exactly the location of a target center, then the player is awarded 100 points to their score, and if it is one adjacent position away from the center, the player has hit the peripheral region of the target and the player is awarded 10 points. Once a hit has been detected the function stops iterating through the arrays of target positions, saves the cursors position, calls the 'eraseTarget()' function passing in the center position of the target that was hit, restores the previously saved cursor position, decrements the variable storing the number of targets on the screen, and changes the values of the center position of the hit target both to 200. This is done to ensure that the player cannot hit the same target again, as 200 is outside the playable range.

The 'eraseTarget()' function performs identically to the 3x3 box drawing portion of the 'drawTarget()' function. It moves the cursor to the positions one above and to the left of the passed in position, one to the left, and one below and to the left, printing three space characters at each position essentially overwriting the block characters with whitespace.

2.2 Hardware

2.2.1 Joystick, Potentiometer, and Buzzer

The analog joystick module used has 5 pins: 5V, V_x, V_y, SCL, and GND. The V_x and V_y pins output an analog voltage corresponding to how far in either direction the joystick is pushed. At its center position both pins output 2.5V; as the joystick is move up or to the right the output voltage increases to 5V, and decreases to 0V when moved in the opposite directions. These two pins were connected to AIN0.1 and AIN0.0 on the 8051, respectively. The SCL pin is connected to the button on the joystick; when pressed this pin outputs 0V. This is connected to /INT1 on the 8051 on P0.3. This corresponds to pin 19 on the 60-pin bus.

A 10kΩ potentiometer was also used as an analog voltage input source to the 8051. The positive terminal was connected to 5V, the negative to ground, and the wiper to AIN0.2 on the 8051, located on pin 49 on the 60-pin bus.

A buzzer was also wired with the positive terminal connected to P2.0 of the 8051, and the negative terminal to ground. P2.0 corresponds to pin 29 on the 60-pin bus.

Schematics for these components can be found in Figure 3 in the appendices section

below.

2.2.2 ADC

This project called for use of the ADC on the 8051. Rather than using the pins on the 60pin bus for AIN0.0 and AIN0.1, the Analog I/O terminal block (J20) was used. A schematic for this can be viewed in Figure 4 the appendicies section below. The internal 2.4V reference generator was used for VREF0. The output of the x and y-axes of the joystick were connected to AIN0.1 and AIN0.0 respectively. The sensitivity potentiometer output was connected to AIN0.2. AGND was connected to a common ground on the board.

2.2.3 LCD and Keypad

In this project a LCD screen was wired to the 8051. A schematic for this can be viewed in Figure 1 in the appendices section below. The LCD module used for this project has 14 pins. Pins 1 and 2 are connections to ground and power, respectively. Pin 3 controls the screen contrast, which can be controlled using a potentiometer which comes as a part of the module. Pins 4, 5, and 6 are the register select, read/write select, and enable control signals, which were connected to P7.0–P7.2 on the 8051. Since port 7 was left in open-drain mode, a 10kohm pull-up resistor was placed between P7.2 and power. Pins 7 to 14 are data pins, which were connected to all of port 6 on the 8051. Port 6 should be configured as open-drain with high-impedance on every pin. This does not have to be done explicitly in the code because this can be accomplished using the associated SFRs' default values.

A keypad was also interfaced to the 8051 for this project. A schematic can be viewed in Figure 2 in the appendices section below. The keypad requires 8 wires, 4 for the columns and 4 for the rows, each of which are connected to port 3 on the 8051. The columns are connected to the lower nibble of port 3 (P3.0–P3.3) which is configured as open-drain for input. These wires are held high using a 3.3V and 10kΩ pull-up resistors, and are also used as inputs to a four input and gate. The output of this gate is used as the /INT0 interrupt source and is connected to P0.2, corresponding to pin 20 on the 60-pin bus. The rows are connected to the high nibble of port 3 (P3.4–P3.7), which is configured as push-pull for output. These wires are held low by the software.

3 Results

All of the intended features were successfully implemented. Starting from the main menu screen, the keypad accurately allowed for the user to select a difficulty and for each difficulty the correct number of targets were drawn and the correct amount of time was allotted. No overlap in targets or targets cropped by the edge of the terminal was observed. During game play when the joystick was moved the cursor moved accordingly, and a noticeable difference in cursor speed was observed when the sensitivity potentiometer was turned. Time was accurately being kept track of and displayed on the LCD screen along with the score which was promptly updated with the correct value when the player hit the target. As planned for, 100 points were awarded for hitting the bullseye, and 10 for hitting the outer tier. If the player pressed the joystick while not on a target, there was no change in score, even if

there was previously a target drawn in that location. When there was 5 seconds remaining the buzzer turned on, and a second later turned off. Lastly, if the player ran out of the the game over screen appeared, and if the player hit all of the targets the victory screen was displayed. Pressing the '#' key on the keypad on either of these screens successfully allowed the user to return to the main menu.

4 Conclusion

The results of our project matched the initial goals and expectations for the game. There are however additional features and improvements that can be made to the game. If given more time, a leaderboard feature could be added which would keep track of the highest scores obtained. Currently, the project uses most of the internal RAM available on the 8051 so additional RAM chips or the segment of xram addressed by "xdata" keyword would have to be utilized to properly store all of the game variables.

5 Appendices

5.1 Code

```
//-----------------------------------------------------------------------------
// ANSI Target game
//-----------------------------------------------------------------------------
// By Alvin Chia, Nick Choi, Samuel Deslandes, and Shanmugam Thiagarajan
// This program plays a target game on an ANSI terminal using the 8051 microprocessor.
// A joystick is used to move the cursor and shoot the targets, a keypad is used to
// receive user input, and an LCD screen displays game statistics.
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f120.h>
#include <stdio.h>
#include <stdlib.h>
#include "putget.h"
#include "LCD.h"
#include "LCD.c"
//-----------------------------------------------------------------------------
// Defines
//-----------------------------------------------------------------------------
#define EXTCLK      11059200            // External oscillator frequency in Hz
#define SYSCLK      11059200            // Output of PLL derived from (EXTCLK * 9/4)
#define BAUDRATE    115200              // UART baud rate in bps
#define COLS_       80
#define ROWS_       50
#define CHAR_       219                 // Ascii code for block
//-----------------------------------------------------------------------------
// Global Constants
//-----------------------------------------------------------------------------
int ADCx;                               // ADC variables
int ADCy;
int sens;
char time;                              // Game variables
int score = 0;
char nboxes;                            // Number of targets on screen
char str[16];                           // Char buffer for use with sprintf() for printing to LCD
char rows[11];                          // Arrays for row and col coordinates for center of targets
char cols[11];
```

```
char timer0_flag = 0;                   // Flag for
signed char xPos, yPos;                 // Coordinates of cursor
char asciichar;                         // Key wakeup vars
//char portvalue;
//char keyvalue;
char keyflag = 0;
unsigned int i;                         // General variable for use in for loops
//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
void main(void);
void SYSCLK_INIT(void);                 // Inits
void PORT_INIT(void);
void UART0_INIT(void);
void TIMER0_INIT(void);
void ADC0_INIT(void);
void read_ADC(void);
void KeypadVector(void) __interrupt 0;  // External interrupt for keypad
void stickPress(void) __interrupt 2;    // External interrupt pushing analog stick
void TIMER0_ISR(void) __interrupt 1;    // Timer0 interrupt. Measures tenths of seconds
char getkeychar(void);                  // Works like getchar() but for keypad input
void Menu(void);                        // Displays game main menu and awaits input
void drawTarget(void);                  // Draws targets onto the terminal
void eraseTarget(char row, char col);   // Removes target at (col,row) from the screen
void checkTarget(char x, char y);       // Hit detection for target at (col,row)
void WinScreen(void);                   // Display game victory screen
void LoseScreen(void);                  // Display game over screen
void playGame(void);                    // Game main routine
//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main(void)
{
    WDTCN = 0xDE;                       // Disable the watchdog timer
    WDTCN = 0xAD;

    PORT_INIT();                        // Initialize the Crossbar and GPIO
    SYSCLK_INIT();                      // Initialize the oscillator
    UART0_INIT();                       // Initialize UART0
    TIMER0_INIT();                      // Initialize TIMER0
    ADC0_INIT();
    lcd_init();                         // Initialize LCD

    SFRPAGE = UART0_PAGE;               // Direct output to UART0
    EX0 = 1;

    while(1){
        playGame();
    }

}

//Main gameplay rountine
void playGame(){
    unsigned int speed;
    xPos = 1;                           // Initial cursor position is home: (1,1)
    yPos = 1;
    score = 0;

    printf("\033[1;37;44m");            // Set terminal background to blue and foreground to yellow.\
                                           Colors are bright.
    printf("\033[2J");                  // Clear screen and return cursor to home position
    Menu();                             // Display game main menu
    printf("\033[H");                   // Return cursor to home position

    TR0 = 1;                            // Start timer0
    while(1){
        // Cursor movement control
```

```
        read_ADC();                         // Get ADC vals
        speed = -11*sens+65535;             // Map sensitivity
        if(ADCx > 3200){                    // Move left
            printf("\033[1D");
            xPos-=1;
            if(xPos < 1) xPos = 1;
        }
        if(ADCx < 500 ){                    // Move right
            printf("\033[1C");
            xPos+=1;
            if(xPos > COLS_) xPos = COLS_;
        }
        if(ADCy < 150){                     // Move up
            printf("\033[1A");
            yPos-=1;
            if(yPos < 1) yPos = 1;
        }
        if(ADCy > 3200){                    // Move down
            printf("\033[1B");
            yPos+=1;
            if(yPos > ROWS_) yPos = ROWS_;
        }

        // Time control
        if(timer0_flag == 10){
            timer0_flag = 0;
            time-=1;
        }

        // Write to lcd
        sprintf(str,"%u",time);             // Display time
        lcd_clear();
        lcd_goto(0);
        lcd_puts(str);

        lcd_goto(0x40);                     // Display score
        sprintf(str,"Score: %u",score);
        lcd_puts(str);

        // Check win
        if(nboxes==0){
            WinScreen();
            break;
        }

        // Check lose
        if(time == 0){
            LoseScreen();
            break;
        }

        // At 5 seconds remaining turn the buzzer on for one second
        if(time == 5){
            P2 = 0x01;
        }
        if(time == 4){
            P2 = 0x00;
        }

        // Variable delay serves as game "frame rate"
        for(i=0;i<speed;i++);
    }

    // Stop timer and reset value
    TR0 = 0;
    TH0 = 0xA6;
    TL0 = 0x00;
    printf("\033[28;26HPress '#' to return to menu.");
```

```
    // Wait for user to press '#' on the keypad
    while(1){
        if(getkeychar()=='#') break;
    }
}

// Displays main menu and awaits user input
void Menu(void)
{
    char choice;
    // Display game mode options
    printf("Select difficulty level\n\r");
    printf("A.) Easy\n\r");
    printf("B.) Medium\n\r");
    printf("C.) Hard\n\r");
    printf("D.) GG\n\r");
    printf("\n\n\rUse the joystick to move the cursor over the targets.\n\r");
    printf("Press the joystick in to 'shoot' the target.\n\r");
    printf("More points are awarded for hitting the bullseye.");
    // Wait for and parse user input
    choice = getkeychar();
    switch(choice){
        case 'A':
            time = 60;
            nboxes = 5;
            drawTarget();
            break;
        case 'B':
            time = 45;
            nboxes = 7;
            drawTarget();
            break;
        case 'C':
            time = 30;
            nboxes = 9;
            drawTarget();
            break;
        case 'D':
            time = 15;
            nboxes = 11;
            drawTarget();
            break;
        default:
            lcd_clear();
            lcd_puts((char*)&"Invalid input");
    }
}

// Waits for and returns user input on the keypad
char getkeychar(){
    while(!keyflag);
    keyflag = 0;
    return asciichar;
}

// Displays game victory screen
void WinScreen(void)
{
    printf("\033[H");
    printf("\033[30m");
    printf("\033[47m");
    printf("\033[2J");
    printf("\033[24;35H");
    for(i=0; i<10; i++)
    {
        printf("-");
    }
    printf("\033[25;36H");
```

```
    printf("You WIN!");
    printf("\033[25;35H");
    printf("|");
    printf("\033[25;44H");
    printf("|");
    printf("\033[26;35H");
    for(i=0; i<10; i++)
    {
        printf("-");
    }
}

// Displays game over screen
void LoseScreen(void)
{
    printf("\033[H");
    printf("\033[30m");
    printf("\033[47m");
    printf("\033[2J");
    printf("\033[24;34H");
    for(i=0; i<12; i++)
    {
        printf("-");
    }
    printf("\033[25;35H");
    printf("You LOSE!!");
    printf("\033[25;34H");
    printf("|");
    printf("\033[25;45H");
    printf("|");
    printf("\033[26;34H");
    for(i=0; i<12; i++)
    {
        printf("-");
    }
}

// Draws n randomly placed target to the terminal
void drawTarget(){
    char row,col,j;
    char redo;                              // Flag for use in targe placement
    printf("\033[2J");                      // Clear screen and return cursor to home position
    for(j=0;j<nboxes;j++)
    {
        redo = 0;
        row = rand()%(ROWS-2) +2;           // Randomly generate coordinate for center of target
        col = rand()%(COLS-2) +2;

        for(i=0;i<j;i++){                   // Iterate through array of currently places targets and
            if((row >= rows[i]-2)&&\        // ensure that no targets overlap
               (row <= rows[i]+2)&&\
               (col >= cols[i]-2)&&\
               (col <= cols[i]+2)){
                redo+=1;
                break;
            }
        }

        if(redo){                           // If a collision was detected, do not increment j and
            j-=1;                           // randomly generate a new center coordinate
            continue;
        }

        rows[j] = row;                      // If new coordinates are valid, add them to the arrays
        cols[j] = col;

        // Draw targets
        printf("\033[37m");                 // change color to white
```

```
        printf("\033[%d;%dH",row-1,col-1);  // Move cursor one above and to the left of center
        printf("%c%c%c",CHAR_,CHAR_,CHAR_); // and print 3 block
        printf("\033[%d;%dH",row+1,col-1);
        printf("%c%c%c",CHAR_,CHAR_,CHAR_);
        printf("\033[%d;%dH",row,col-1);
        printf("%c%c%c",CHAR_,CHAR_,CHAR_);
        // Red bullseye
        printf("\033[%d;%dH",row,col);      // At the center location
        printf("\033[31m");                 // change color to red
        printf("%c",CHAR_);                 // and draw block


    }
}

// Erase target centered at (col,row)
void eraseTarget(char row,char col){
    printf("\033[%d;%dH",row-1,col-1);      // Move cursor to left most column of each row of box
    printf("   ");                          // and overwrite with spaces
    printf("\033[%d;%dH",row,col-1);
    printf("   ");
    printf("\033[%d;%dH",row+1,col-1);
    printf("   ");
}

// Hit detection for targets
void checkTarget(char x, char y){
    char k;
    char hit = 0;                           // Hit detected flag

    // Scan all target coordinates in arrays
    for(k=0;k<11;k++)
    {
        // row above
        if(y==rows[k]-1){
            if((x>=cols[k]-1)&&(x<=cols[k]+1)){
                score+=10;
                hit+=1;
                break;
            }
        }
        // row below
        if(y==rows[k]+1){
            if((x>=cols[k]-1)&&(x<=cols[k]+1)){
                score+=10;
                hit+=1;
                break;
            }
        }
        // center row
        if(y==rows[k]){
            if(x==cols[k]){
                // Bullseye
                score+=100;
                hit+=1;
                break;
            }
            if((x==cols[k]-1)||(x==cols[k]+1)){
                score+=10;
                hit+=1;
                break;
            }
        }
    }

    // If hit detected
    if(hit){
        printf("\033[s");                   // Store cursor position
```

```
        eraseTarget(rows[k],cols[k]);           // Erase selected target
        printf("\033[u");                       // Restore cursor location
        rows[k] = 200;                          // Move selected target coordinates to
        cols[k] = 200;                          // unreachable location so they can't be hit again
        nboxes-=1;                              // Reduce number of remaining targets
    }
}

// Function to handle ADC
void read_ADC(){
    // Get joystick y value
    AMX0SL = 0x00;
    for(i=0;i<300;i++);
    AD0INT = 0;                                 // Clear conversion interrupt flag
    AD0BUSY = 1;                                // Start conversion
    while(!AD0INT);                             // Wait for conversion to end
    ADCy = ADC0;

    // Get joystick x value
    AMX0SL = 0x01;
    for(i=0;i<300;i++);
    AD0INT = 0;                                 // Clear conversion interrupt flag
    AD0BUSY = 1;                                // Start conversion
    while(!AD0INT);                             // Wait for conversion to end
    ADCx = ADC0;

    // Get sensistivity control potentiometer voltage
    AMX0SL = 0x02;
    for(i=0;i<300;i++);
    AD0INT = 0;                                 // Clear conversion interrupt flag
    AD0BUSY = 1;                                // Start conversion
    while(!AD0INT);                             // Wait for conversion to end
    sens = ADC0;
}

// External Interrupt 0  for keypad
void KeypadVector(void) __interrupt 0{
    char portvalue,keyvalue;
    EX0 = 0;                        // Disable /INT0
    keyflag = 1;
    keyvalue = P3 & 0x0F;
    // Try first row
    P3=0x8F;                        // check if row one (top) was active
    for(i = 0; i<400; i++);         // wait for the output and input pins to stabilize
    portvalue = P3 & 0x0F;          // read the value of the lower 4 bits
    if (portvalue == 0x0F)          // if this row was selected then the value will be 0x0F
    {
        if (keyvalue == 0x07){      // look at the value of the low 4 bits
            asciichar = '1';        // return the value of the matching key
        }
        else if (keyvalue == 0x0B){
            asciichar = '2';
        }
        else if (keyvalue == 0x0D){
            asciichar = '3';
        }
        else{
            asciichar = 'A';
        }

        P3 = 0x0F;                  // put output lines back to 0
        while (P3 != 0x0F);         // wait while the key is still pressed
        for(i = 0; i<20000; i++);   // wait for output and input pins to stabilize after key is released
        EX0 = 1;
        return;
    }
    // Try second row
    P3=0x4F;                        // check if row one (top) was active
```

```
for(i = 0; i<400; i++);          // wait for the output and input pins to stabilize
portvalue = P3 & 0x0F;           // read the value of the lower 4 bits
if (portvalue == 0x0F)           // if this row was selected then the value will be 0x0F
{
    if (keyvalue == 0x07){       // look at the value of the low 4 bits
        asciichar = '4';         // return the value of the matching key
    }
    else if (keyvalue == 0x0B){
        asciichar = '5';
    }
    else if (keyvalue == 0x0D){
        asciichar = '6';
    }
    else{
        asciichar = 'B';
    }

    P3 = 0x0F;                   // put output lines back to 0
    while (P3 != 0x0F);          // wait while the key is still pressed
    for(i = 0; i<20000; i++);    // wait for output and input pins to stabilize after key is released
    EX0 = 1;
    return;
}
// Try third row
P3=0x2F;                         // check if row one (top) was active
for(i = 0; i<400; i++);          // wait for the output and input pins to stabilize
portvalue = P3 & 0x0F;           // read the value of the lower 4 bits
if (portvalue == 0x0F)           // if this row was selected then the value will be 0x0F
{
    if (keyvalue == 0x07){       // look at the value of the low 4 bits
        asciichar = '7';         // return the value of the matching key
    }
    else if (keyvalue == 0x0B){
        asciichar = '8';
    }
    else if (keyvalue == 0x0D){
        asciichar = '9';
    }
    else{
        asciichar = 'C';
    }

    P3 = 0x0F;                   // put output lines back to 0
    while (P3 != 0x0F);          // wait while the key is still pressed
    for(i = 0; i<20000; i++);    // wait for output and input pins to stabilize after key is released
    EX0 = 1;
    return;
}
// Try last row
P3=0x1F;                         // check if row one (top) was active
for(i = 0; i<400; i++);          // wait for the output and input pins to stabilize
portvalue = P3 & 0x0F;           // read the value of the lower 4 bits
if (portvalue == 0x0F)           // if this row was selected then the value will be 0x0F
{
    if (keyvalue == 0x07){       // look at the value of the low 4 bits
        asciichar = '*';         // return the value of the matching key
    }
    else if (keyvalue == 0x0B){
        asciichar = '0';
    }
    else if (keyvalue == 0x0D){
        asciichar = '#';
    }
    else{
        asciichar = 'D';
    }

    P3 = 0x0F;                   // put output lines back to 0
```

```
        while (P3 != 0x0F);          // wait while the key is still pressed
        for(i = 0; i<20000; i++);    // wait for output and input pins to stabilize after key is released
        EX0 = 1;
        return;
    }
}

// External interrupt 1 ISR for joystick button
void stickPress(void) __interrupt 2{
    for(i=0;i<300;i++);
    checkTarget(xPos,yPos);
}

// Timer0 interrupt ISR for counting tenths of seconds
void TIMER0_ISR(void) __interrupt 1{
    // reset timer to 0xA600 and increment flag
    TH0 = 0xA6;
    TL0 = 0x00;
    timer0_flag += 1;
}

//-----------------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------------
//
// Initialize the system clock to use a 11.0592MHz crystal as its clock source
//
void SYSCLK_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE = CONFIG_PAGE;
    OSCXCN  = 0x77;                     // Start ext osc with 11.0592MHz crystal
    for(i=0; i < 256; i++);             // Wait for the oscillator to start up
    while(!(OSCXCN & 0x80));
    CLKSEL  = 0x01;
    OSCICN  = 0x00;

    SFRPAGE = SFRPAGE_SAVE;             // Restore SFR page
}

//-----------------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------------
//
// Configure the Crossbar and GPIO ports
//
void PORT_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;             // Save Current SFR page

    SFRPAGE  = CONFIG_PAGE;
    XBR0     = 0x04;                    // Enable UART0
    XBR1     = 0x14;                    // Route /INT0 and /INT1 to port pins
    XBR2     = 0x40;                    // Enable Crossbar and weak pull-up

    EA       = 1;                       // Enable global interrupts
    EX1      = 1;                       // Enable external interrupt 1

    P0MDOUT |= 0x01;                    // Set TX0 on P0.0 pin to push-pull
    P2MDOUT  = 0x01;                    // Set buzzer port (P2.0) for push-pull
    P2       = 0x00;                    // Set P2.0 for digital LO
    P3MDOUT  = 0xF0;                    // Set P3 high nibble as output, low nibble as input
    P3       = 0x0F;                    // P3 high nibble set to 0v
```

```
    SFRPAGE  = TIMER01_PAGE;
    IT1      = 1;                          // /INT1 triggered on falling edge

    SFRPAGE  = SFRPAGE_SAVE;               // Restore SFR page
}

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <baudrate> and 8-N-1
//
void UART0_INIT(void)
{
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;                // Save Current SFR page

    SFRPAGE = TIMER01_PAGE;
    TMOD   &= ~0xF0;
    TMOD   |=  0x20;                       // Timer1, Mode 2, 8-bit reload
    TH1     = 0xFA;                        // Set Timer1 reload baudrate value T1 Hi Byte
    CKCON  |= 0x10;                        // Timer1 uses SYSCLK as time base
    TL1     = TH1;
    TR1     = 1;                           // Start Timer1

    SFRPAGE = UART0_PAGE;
    SCON0   = 0x50;                        // Mode 1, 8-bit UART, enable RX
    SSTA0   = 0x10;                        // SMOD0 = 1
    TI0     = 1;                           // Indicate TX0 ready

    SFRPAGE = SFRPAGE_SAVE;                // Restore SFR page
}

// Timer init
void TIMER0_INIT(void){
    char SFRPAGE_SAVE;
    SFRPAGE_SAVE = SFRPAGE;

    SFRPAGE = TIMER01_PAGE;

    TMOD &= 0xF0;                  // Timer0, Mode 1: 16-bit counter/timer.
    TMOD |= 0x01;
    TH0 = 0xA6;                    // Set high byte such that timer0 starts at 0xA600
    CKCON &= ~0x09;
    CKCON |= 0x02;                 // Timer0 uses SYSCLK/48 as base
    TL0 = 0x00;                    // Set high byte such that timer0 starts at 0xA600


    SFRPAGE = CONFIG_PAGE;
    ET0 = 1;                       // Enable timer0 interrupt

    SFRPAGE = SFRPAGE_SAVE;
}

// ADC init
void ADC0_INIT(void){
    char SFRPAGE_SAVE;

    SFRPAGE_SAVE = SFRPAGE;
    SFRPAGE = ADC0_PAGE;

    REF0CN |= 0x03;                        // turn on internal ref buffer and bias generator
    ADC0CF = 0xBF;                         // AD0SC = 23 for SARclk of 230400hz, Gain = 1
    AD0EN = 1;                             // enable ADC0

    AMX0CF = 0x00;
```

```
    SFRPAGE = SFRPAGE_SAVE;                 // Restore SFR page
}
```

5.2 Schematics

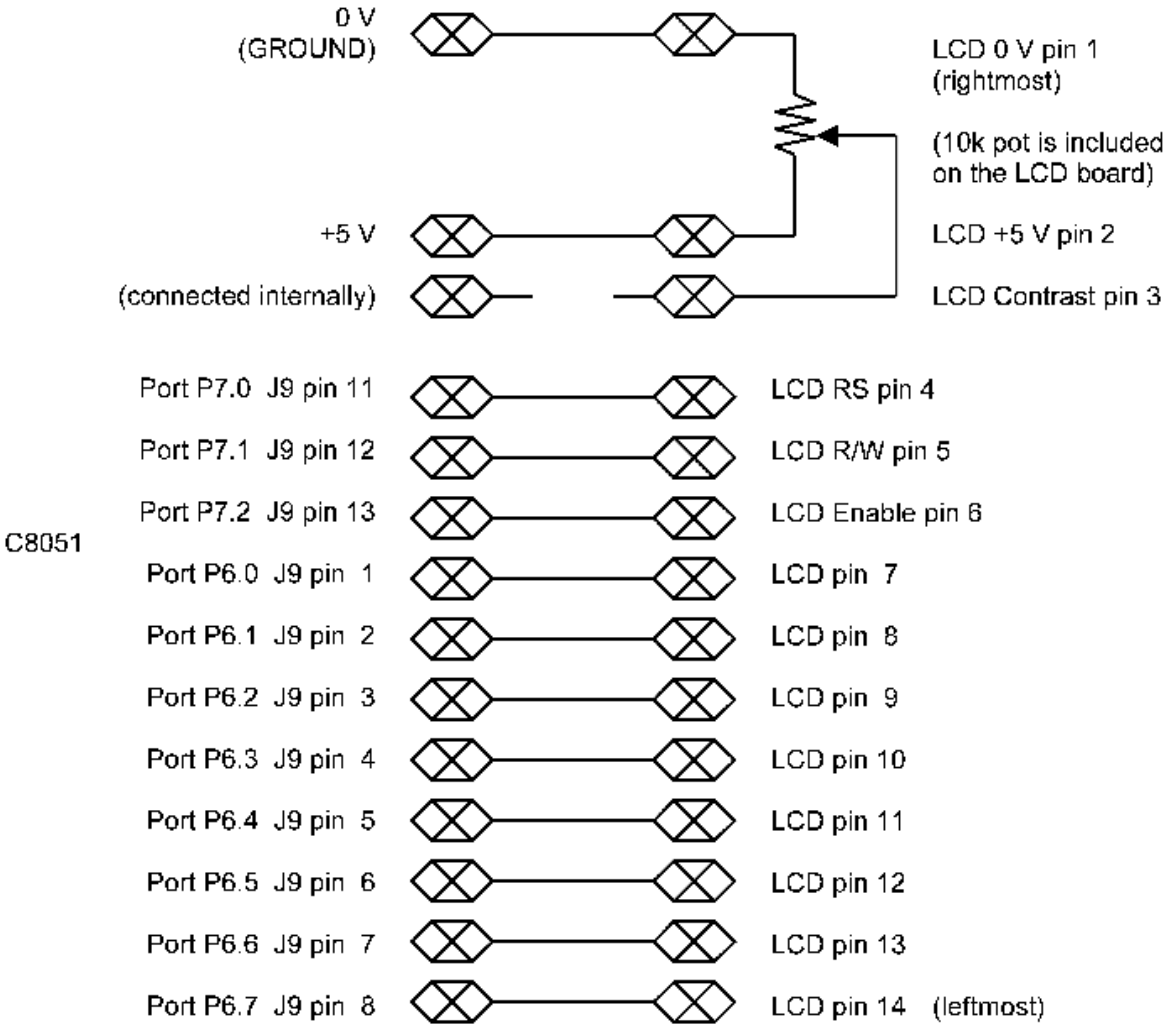

Figure 1: Circuit schematic for LCD[2]

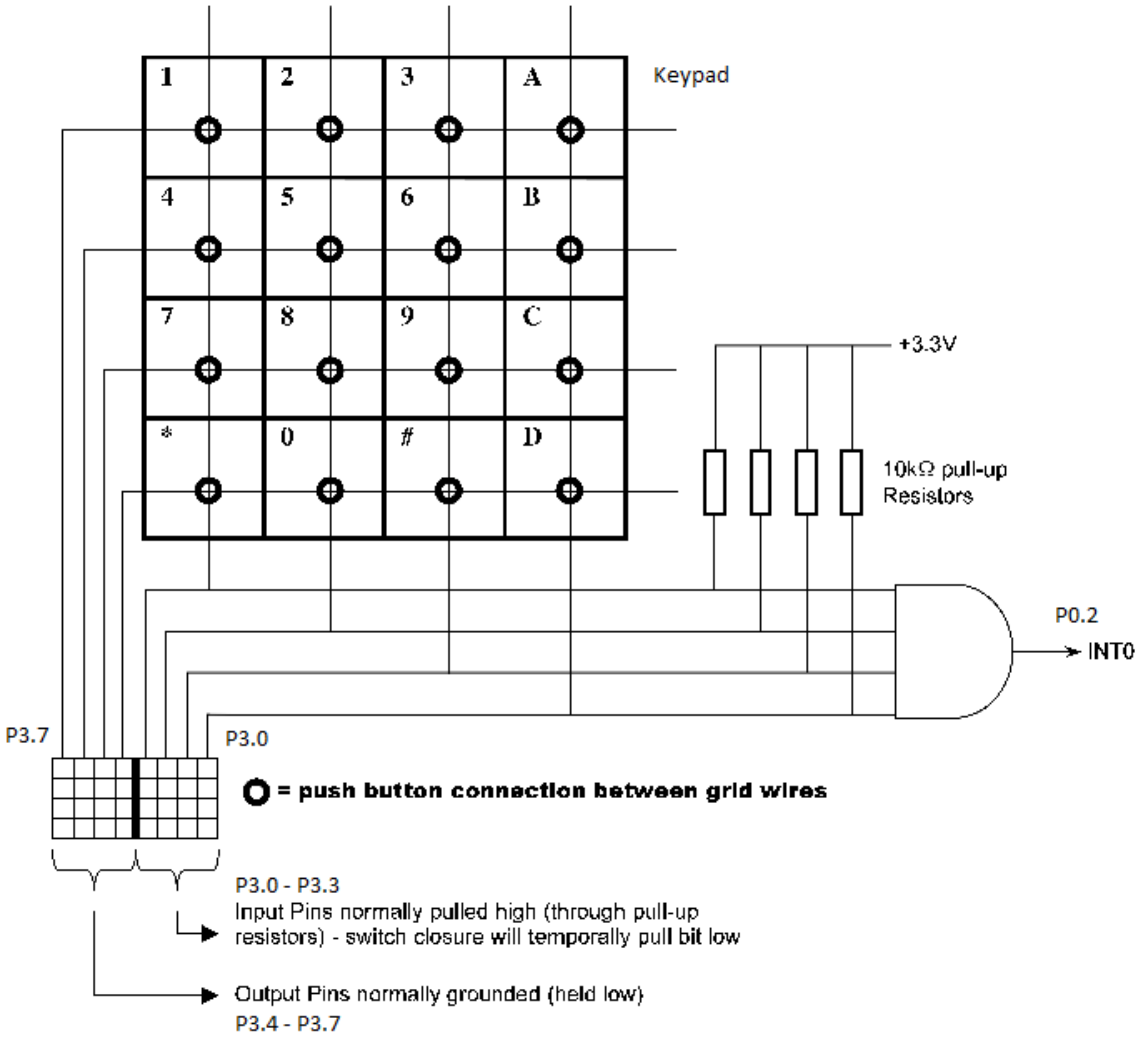

Figure 2: Circuit schematic for keypad[1]

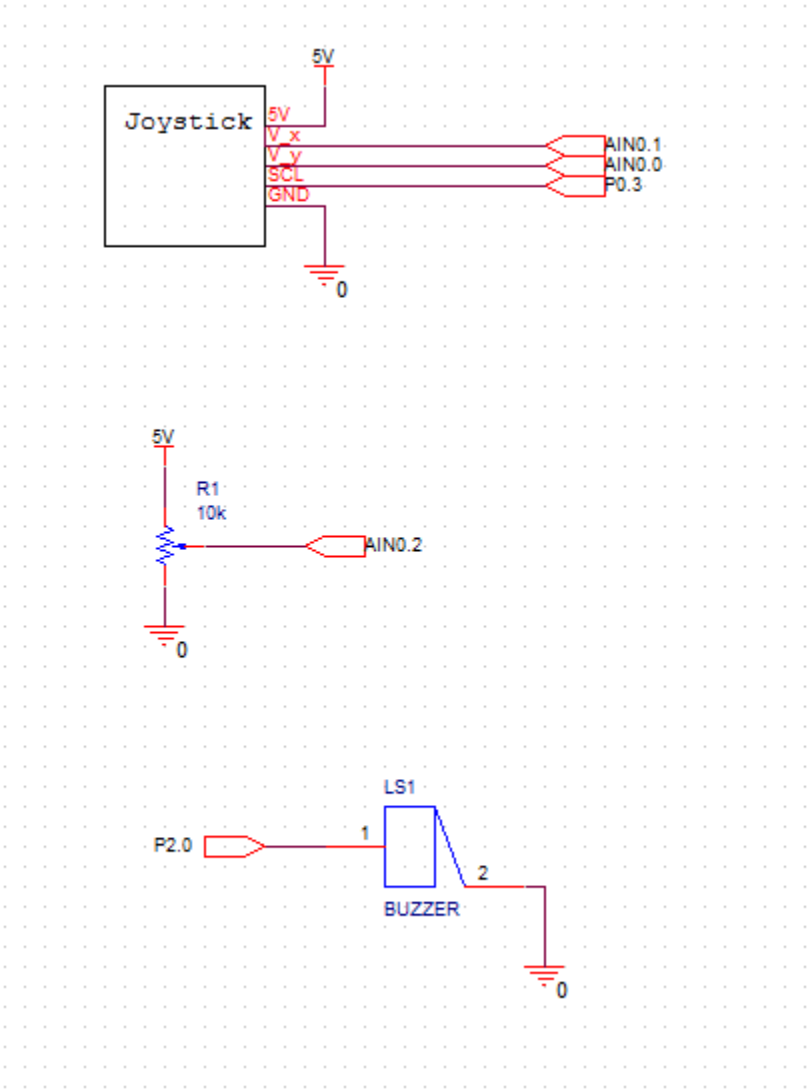

Figure 3: Schematic for joystick, potentiometer, and buzzer connection

5.3 Analog I/O Terminal block

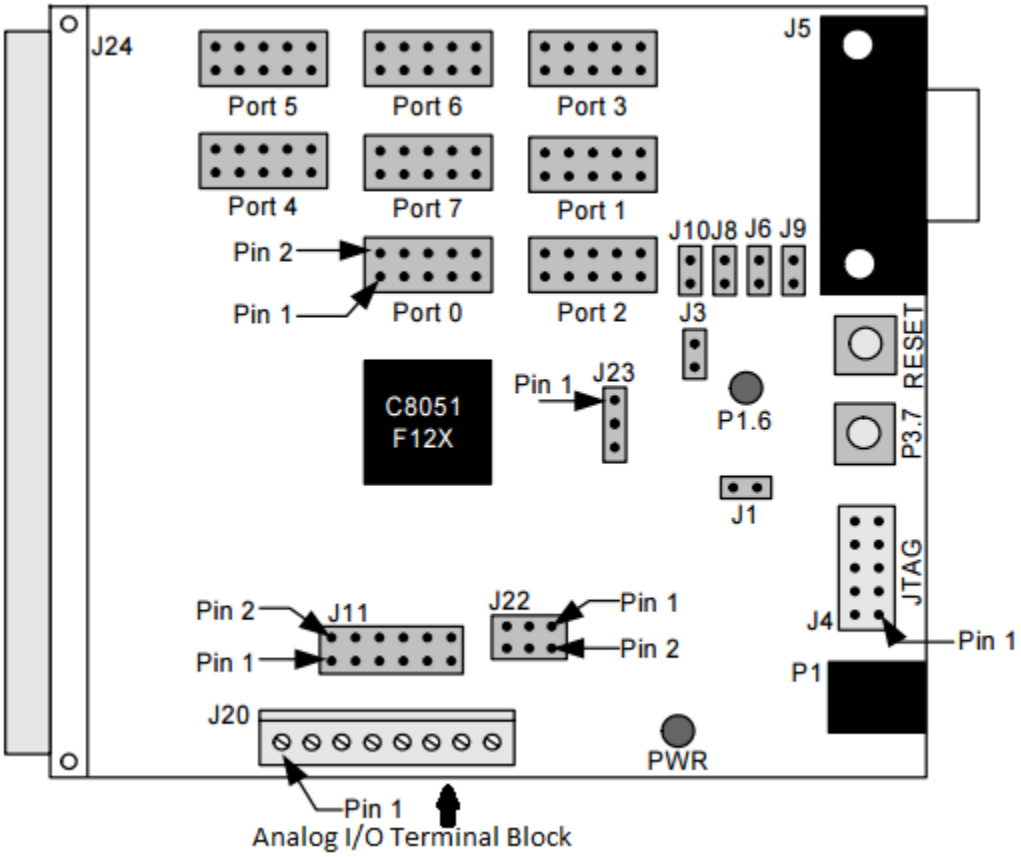

Figure 4: C8051F120 Targer Board layout[1]

Pin#	Description
1	CP0+
2	CP0-
3	DAC0
4	DAC1
5	AIN0.0
6	AIN0.1
7	VREF0
8	AGND (Analog Ground)

Table 2: J20 Terminal Block reference table

5.3.1 ANSI Escape Sequence Table

ANSI escape sequence	Function
\033[2J	Clear screen and return cursor to home
\033[1;33;43m	Set foreground color to yellow and background color to white, bright
\033[1;37m	Set foreground color to white, bright
\033[1;31m	Set foreground color to red, bright
\033[30m	Set foreground color to black, dim
\033[47m	Set background color to white, dim
\033[{row};{col}H	Move cursor position to ({row},{col})
\033[1A	Move cursor up one row
\033[1B	Move cursor down one row
\033[1C	Move cursor right one column
\033[1D	Move cursor left one column
\033[H	Return cursor to home position
\033[s	Save cursor position
\033[u	Restore cursor position

Table 3: Quick reference table of ANSI escape sequences

6 References

[1]"C8051F12x Development Kit User's Guide ," in RPI ECSE Department, Rev. 0.6,May 2005. [Online]. Available: `https://www.ecse.rpi.edu/courses/CStudio/Silabs/C8051F12x-DK.pdf`. Accessed: Dec. 12, 2016.

[2]"Interfacing a Hitachi HD44780 to a Silicon Laboratories C8051F120," in RPI ECSE Department, 2016. [Online]. Available: `http://www.rpi.edu/dept/ecse/mps/LCD_Screen-8051.pdf`. Accessed: Dec. 12, 2016.

[3]"C8051 Manual," in RPI ECSE Department, 1.4 ed., 2005. [Online]. Available: `https://www.ecse.rpi.edu/courses/CStudio/Silabs/C8051F12x-13x.pdf`. Accessed: Dec. 12, 2016.

[4]"MPS Lab 1," in RPI ECSE Department, 2016. [Online]. Available: `http://www.rpi.edu/dept/ecse/mps/MPS_Lab_Ex1-IDE_ANSI.pdf`. Accessed: Dec. 12, 2016.

[5]"Ascii Table - ASCII character codes and html, octal, hex and decimal chart conversion", Asciitable.com, 2016. [Online]. Available: `http://www.asciitable.com/`. Accessed: Dec. 12, 2016.