# Microprocessor Systems
# Lab 6: Memory Interfacing

Alvin Chia Nick Choi Samuel Deslandes
Shanmugam Thiagarajan

12/13/16

# 1 Introduction

# 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.

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()

#### 2.1.3 Target handling

### 2.2 Hardware

# 3 Results

# 4 Conclusion

# 5 Appendices

## 5.1 Code

```
//-----------------------------------------------------------------------------
// Final
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// 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();
    speed = -11*sens+65535;
    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;
  // 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”);
  // 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 target center coordinates are valid, add then 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 to left most column of each row with
        (col,row) at its 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
      the target
  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{
  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 0x3580 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;
    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, vref0
      is ref voltage
  ADC0CF = 0xBF;              // AD0SC = 23 for SARclk of 1Mhz, Gain = 1
  AD0EN = 1;                // enable ADC0

  AMX0CF = 0x00;
```

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

## 5.2 Schematics

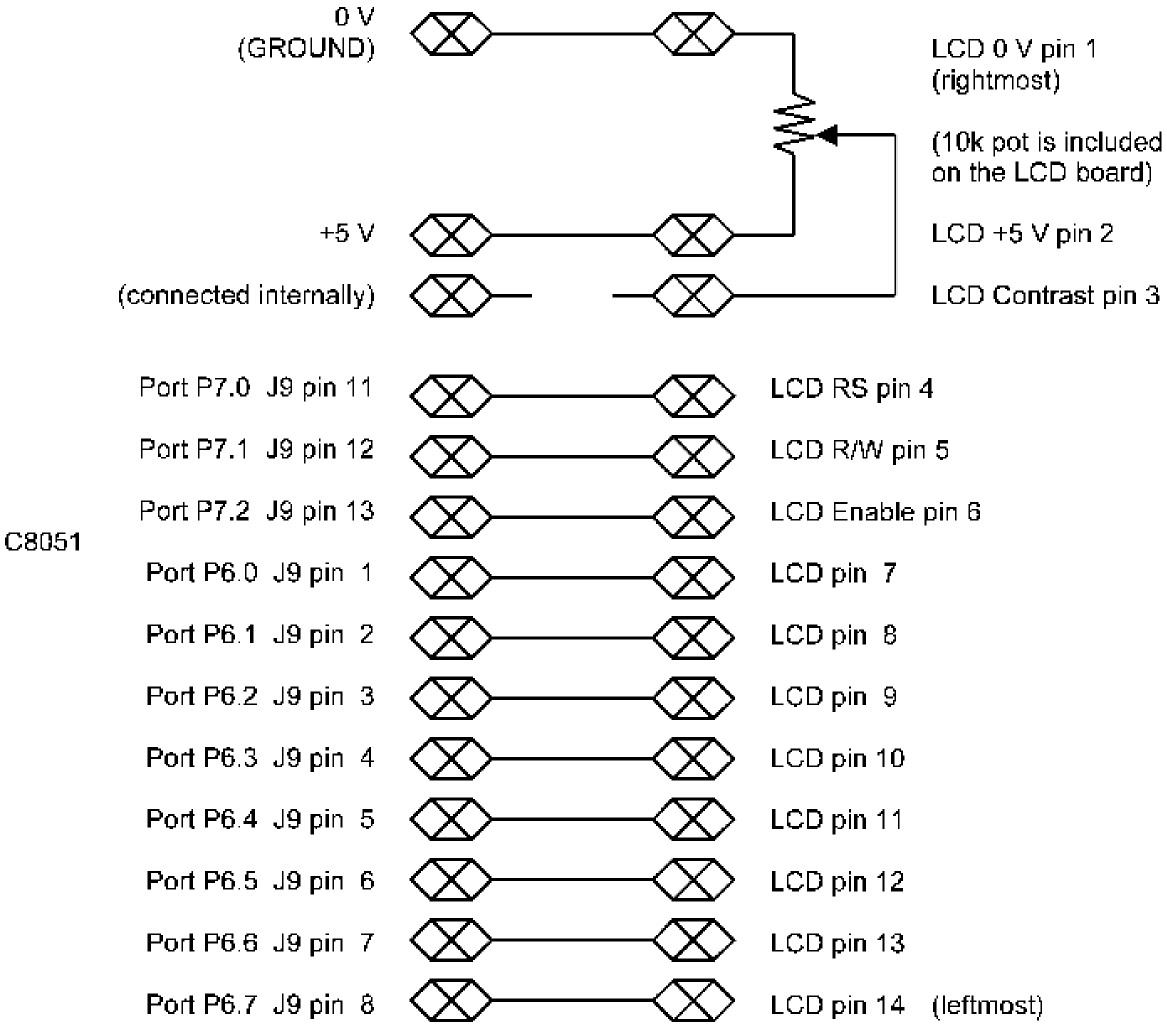

Figure 1: Circuit schematic for LCD[2]

# 6 References

[1] "MPS Lab 6," in RPI ECSE Department, 2016. [Online]. Available: http://www.rpi.edu/dept/ecse/mps/MPS_Lab_Ex6-Magic8Ball.pdf. Accessed: Nov. 27, 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: Nov. 27, 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: Nov. 27, 2016.