ECEN-4330
Microprocessor Systems Design

Final Report

Osama Atta

Spring 2022

Department of Electrical
&
Computer Engineering
University of Nebraska-Lincoln
(Omaha Campus)

Table of Contents

1 Summary

This lab revolved around finalizing the PCB design, ordering it, soldering all the parts required, and writing code in embedded C to perform the same functions created in the first three labs of this course: RAM check, move, count, dump, edit, and find. The PCB was designed using EasyEDA as it proved the most straight forward to use and had all the tools required to construct the PCB. The PCB was ordered using JLCPCB as they were the cheapest option especially since the final design of the board was a large size and larger boards come with a greater price tag. After receiving the board relatively quickly, the continuity for all the connections was checked and the parts were soldered on. Then the software was written in embedded C using visual studio code and the embedded IDE extension. After many hours of debugging both for the hardware and software, the final product was demonstrated and signed off. This paper will discuss all the steps mentioned, the design for both the software and hardware, and the issues faced on both fronts and how they were resolved.

2 Objective

The objective of this lab was to finalize the PCB design by moving the circuit designed on the breadboard to a PCB. Also, code needed to be written to implement the functions described in the first three labs to work with the circuit designed. The lab was deemed successful when the RAM functions, Wi-Fi module, and ADC components were performing their respective functions according to the requirements set in lab 5. The RAM functions had to be able to manipulate data in RAM in their different ways, the Wi-Fi module had to be able to receive keypad values from the website, and the ADC components had to respond to the different stimuli presented to them e.g., the photoresistor had to present different values when covered and when a light was shone on it.

3 Introduction

The two big steps in this lab are the design of the software using embedded C and the design of the PCB using EasyEDA. Code in embedded C was relatively simple to grasp as it had many similarities to high level programming languages and made it easier to implement the functions than it was in assembly. The tool that was used to complete the programming portion of this project was visual studio code with the embedded IDE extension. The embedded IDE extension allowed for the use of the SDCC compiler and produced the required .hex file that was required to be flashed on to the AT89LP51 and ROM. EasyEDA was new, however it was very straightforward and proved very easy to use.

4 Hardware Discussion

The microcontroller used in this project is the AT89LP51 with external clock and reset circuitry. 64K of RAM, 64K of ROM, a photoresistor & a temperature sensor each with its own analog to digital converter (ADC), LCD, 7-segment display, and a keypad were all connected to the AT89LP51 controller. To select between all these I/O devices and the external memory a GAL was used as a decoder that was connected to the various chip select pins. The Keypad was connected to port 1 of the AT89LP51 as an input.

For the clock, the crystal oscillator was connected to both XTAL 1 and 2 and to two crystal 5pF capacitors. This circuit was obtained from the AT89LP51 datasheet. The reset circuit consisted of a 10uF capacitor and a 10k ohm resistor along with a reset switch that sent a high signal once pressed. In this case the POL pin was connected to VCC to make the reset pin active high.

Port 0 of the AT89LP51 was connected to all the IO through 10k pull up resistors to act as the data pins (D0 – D7). Since Port 0 is multiplexed it required a demultiplexer to extract A0-A7. The latch enable of the demultiplexer was connected to ALE from the microcontroller and the output enable was grounded. Port 2 of the microcontroller was the remaining address lines, A8-A15, giving a total of 16 address lines.

P3.4 was used as the IOM pin which was connected to the GAL to allow for the interfacing with the various IO devices. A15 and A14 were both connected to the GAL and allowed for the selection of the various IO devices and external memory. The final input connection to the GAL was PSEN. PSEN was used to select the external ROM chips. The GAL has an output for: the temperature sensor, the photoresistor, the 7-segement display, the LCD, RAM 1 & 2, and ROM 1 & 2.

Both the RAM chips had very similar connections except each got their own signal from the decoder to differentiate between them. The address lines and the data lines were connected to both the chips. The write enable and the output enable were connected to the write and read pins respectively of the microcontroller. For the ROM, connections were also very similar for both chips but, the output enable was now connected to PSEN and the write enable was connected to VCC. This was done because we will only ever read from ROM when the PSEN signal is low, and we will never be writing to ROM.

The 7 segment was connected through a latch to the data pins from the microcontroller. The latch had its output enable pin grounded and its latch enable pin connected to the 7-segment chip select from the decoder.

The LCD was connected to the data lines from the microcontroller. The read pin was connected to VCC since we won't be taking data in from the LCD. This also seemed to help with some issues related to the current being supplied to the LCD when the circuit was built on the breadboard. The write pin of the LCD was connected to the write pin on the microcontroller. The LCD C/D pin was connected to P3.5 of the microcontroller. The behavior of P3.5 will be set in the code so that the LCD will get the appropriate C/D signals. The CS pin of the LCD is connected to the CS_LCD signal from the GAL.

Both the ADCs were connected in a similar fashion. The data lines from the microcontroller were connected to them. The read pin for each was connected to the read pin from the microcontroller. The chip selects pins from the GAL were connected to their respective device. The mode pin was connected to ground due to instruction from the datasheet.

Finally, the ESP 8266 was connected serially to the AT89LP51 via its RxD and TxD pins. Also, a voltage divider was used to supply the ESP with a consistent 3.3V. The enable and reset pins of the ESP were also connected to the 3.3V source.

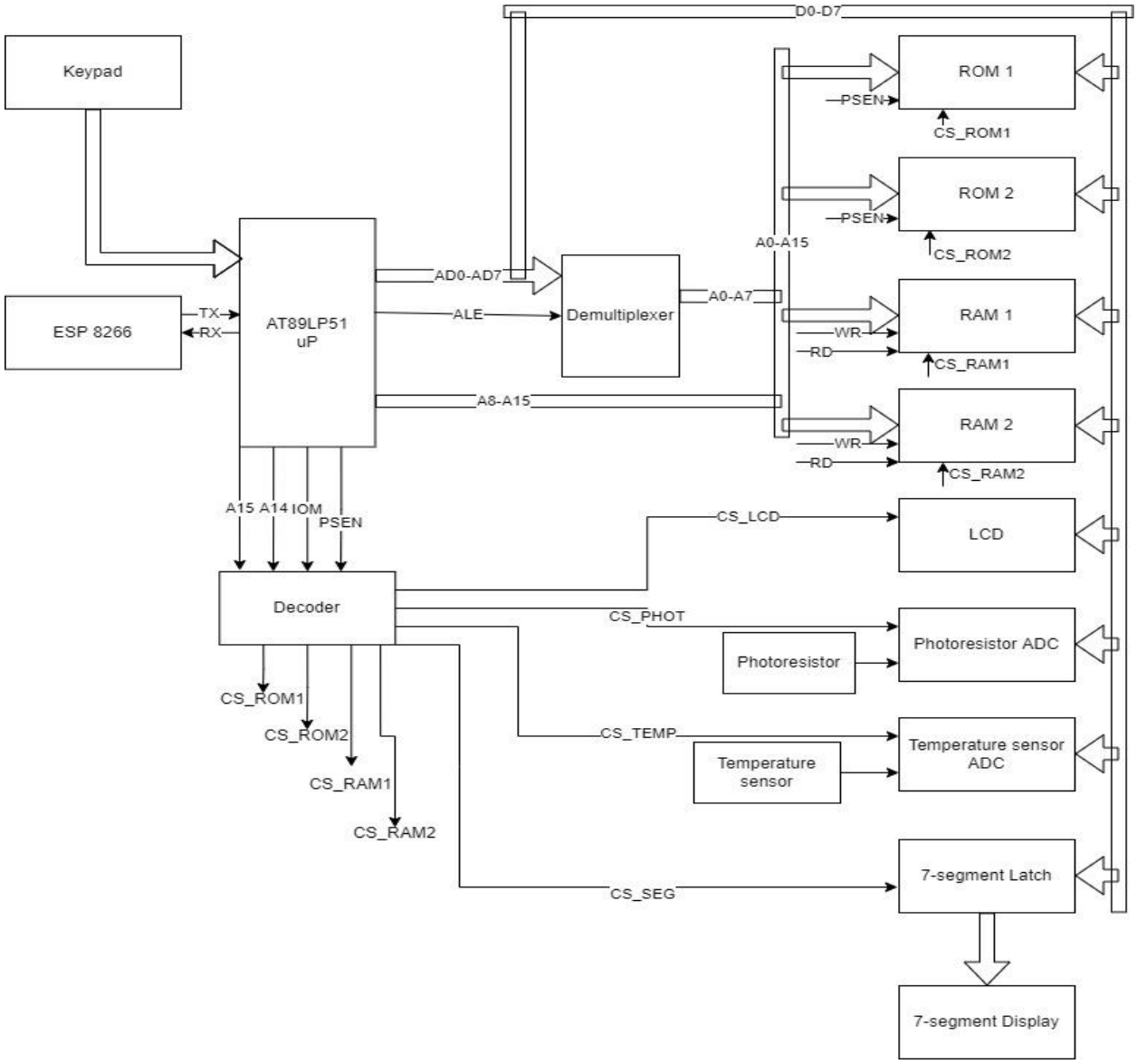
To power the whole system, a DC power jack connected through a fuse and a switch was added to the circuit.

After this design was tried and tested on a breadboard, it was put into schematic form in EasyEDA and the PCB was generated.

For the PCB, simple design procedures were utilized. The top surface was used for horizontal traces, and the bottom was used for vertical traces. Via's were utilized to make this form of routing possible. The width of the traces was set to 0.381mm. The VCC lines were set to be 0.508mm in width. Also, headers in various locations, such as the microcontroller and decoder, were added in case debugging was required.

After the design was finalized and approved by Mr. Boeding, the PCB was ordered via JLCPCB as it offered the cheapest prices and a reasonable delivery time. The final board came out to be 312.6mm * 216mm. Once it had arrived, the lines were checked for continuity and the various components were soldered on using a soldering iron.

4.1 Block Diagram



4.2 Wiring/Trace/Bus

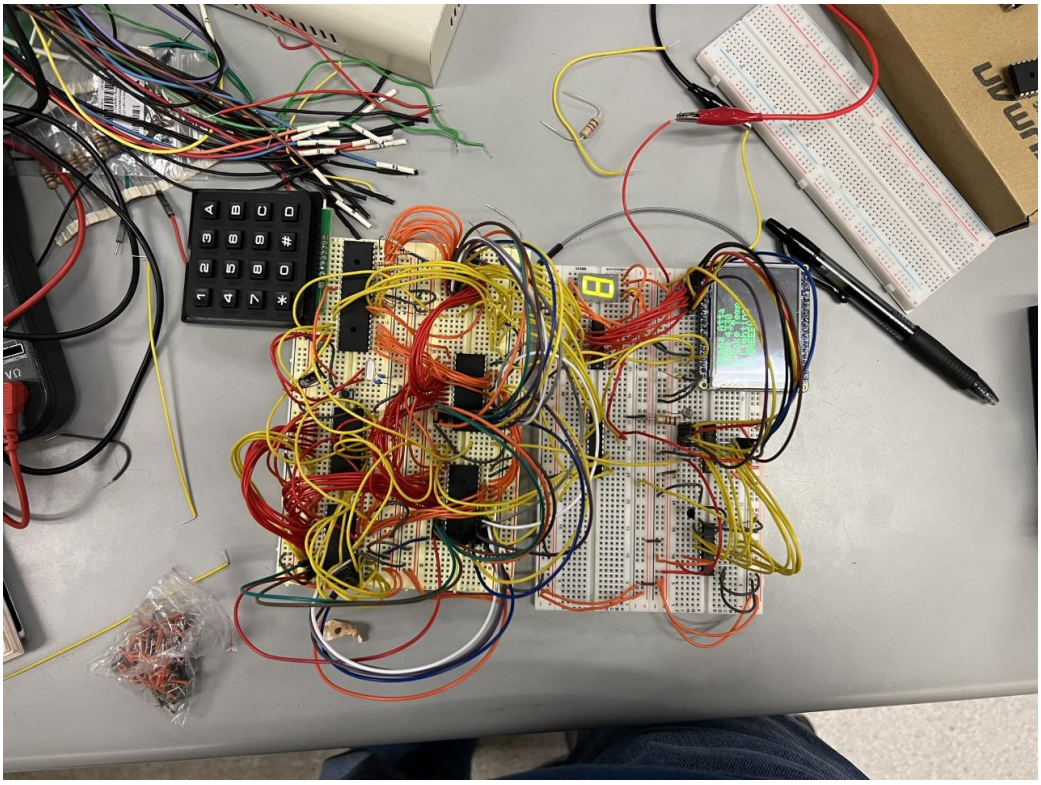


Figure 1: Breadboard wiring

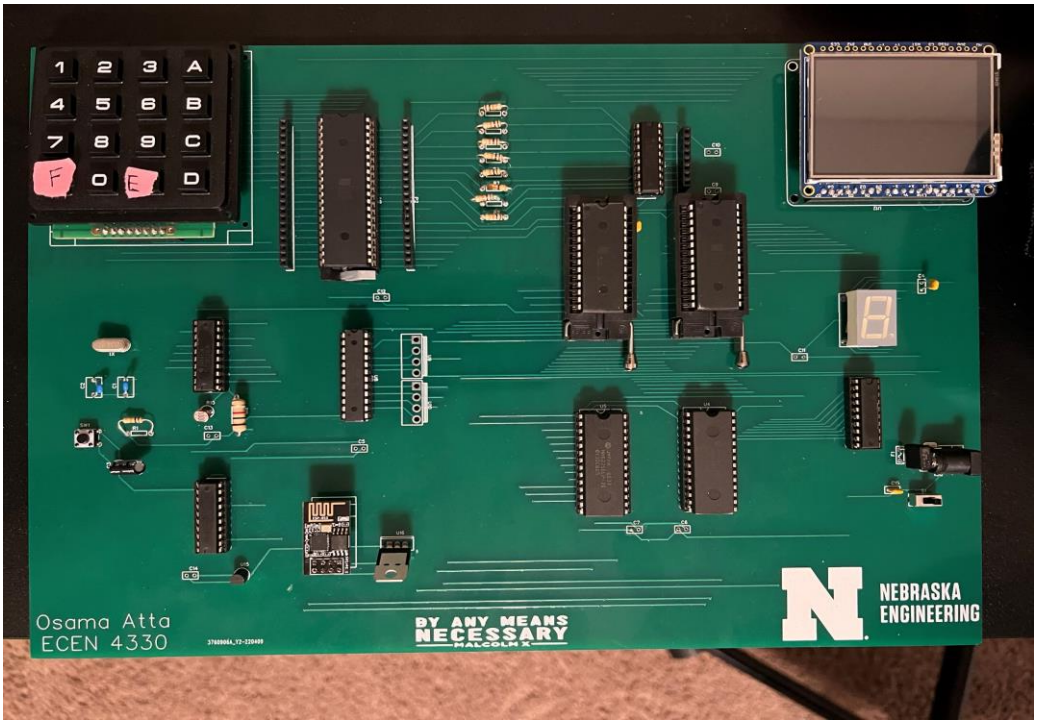


Figure 2: Final PCB

5 Software Discussion

5.1 Function Blocks

The code for this project was written in embedded C and is split into a main that initializes the LCD and the UART to start the ESP. The main then calls the main menu function which is a loop that waits for a user to select one of the ram functions or to see the values from the ADC's. If one of the ram functions is selected, it will call the setup for that respective function which will take in all the data required from the user, such as the block size and starting address. After all the data is taken from the user it is passed on to the execute for that respective function which then performs the respective task. The Wi-Fi module is setup so that it takes in an input anytime one is expected from a keypad.

5.1.1 main:

The main initializes the LCD using the provided LCD initialization function (TFT_LCD_INIT()). Then the LCD is set to rotation 3, the screen is filled black, the text is set to green, and the cursor is put in position (0,0). After the screen is done initializing the UART is initialized to setup the ESP 8266 Wi-Fi Module. Finally, the main menu function is called.

5.1.2 Main menu:

The main menu function presents the user with the choices they are able to make to proceed. The options are A for Ram Check, B for Move, C for Count, D for Dump, E for Edit, F for Find, 1 for Temp & Phot, and 2 for Free Type. If the user does not press one of the options an error is given, and they are returned to the menu. After one of the ram functions (options A-F) is selected, the respective setup function is selected where the inputs from the user are taken except for ram check which will do all the required steps in one function. If '1' is selected on the keypad, the values read from the photoresistor and the temperature sensors are displayed in hex format.

5.1.3 Ram Check:

Ram check will print 'A' on the 7-segment display and then prompt the user for a byte hex value that will be used to check ram with. The value that the user inputs will be put in every location in RAM and read back and checked if it matches with what was sent out. After the value is sent to every location in RAM the value being sent out is complemented and sent out again and checked. If the check is successful for every location with both the original and complemented values, the LCD will read "Test Passed". If the test fails while writing the original value to RAM the LCD will display that it failed during "stage one" along with the location in RAM where it failed and the value that was received from RAM instead of the expected value. The same is done with the complemented value except now the LCD will say it failed during "stage two." After completing the test, the ram function will return to the main menu.

5.1.4 Move:

When the move option is selected from the main menu the setupMove function is called. The setupMove will firstly display a 'B' on the 7-segment display. It will then prompt the user to input what size block type they would like to use for the operation (1 for Byte, 2 for word, or 3 for double word). Next, the user is prompted to enter the number of blocks (4 hex digits) they would like to work with. Then, the user is asked to input the source address (4 hex digits) and the destination address (4 hex digits) for the move operation. After all these pieces of data are gathered, the data is sent to the moveExecute function. The move execute will print a "Please wait" message on the LCD to let the user know that the operation might take some time. After the number of blocks is calculated, a while loop loops through the addresses and moves the data from the source to the destination while using the number of blocks as a counter. The source address and destination address are incremented with every loop. After the move function is complete, "Move Complete" is printed to the LCD. The user is then returned to the main menu.

5.1.5 Count:

When the user selects the count function in the main menu, the setupCount function is called. This function will print ‘C’ on the 7-segment display. It will then prompt the user to input what size block type they would like to use for the operation (1 for Byte, 2 for word, or 3 for double word). Next, the user is prompted to enter the number of blocks they would like to work with (4 hex digits). It will also ask the user to input the starting address (4 hex digits) and the value to look for (2 hex digits). After all this data is collected, the executeCount function is called. Firstly, the proper size is calculated based on the data type. The user is then prompted with what value is being looked for. While that is happening the total instances of the value being looked for within the specified range are counted and stored. After that is done, the user is presented with the first instance found of the value being looked for and the total number of instances found within the range specified. The user is then given an option to select between exiting the function and returning to the main menu or to look at the next instance found. To exit the user must press ‘E’ on the keypad and to move to the next instance the user needs to select ‘0’. When the user is on the second page or greater the option to also go back a page becomes available (press ‘1’). If no instances are found, then the user is prompted with a message that reads ‘Value not found.”

5.1.6 Dump:

When the user selects the dump function in the main menu, the setupDump function is called. This function will print ‘D’ on the 7-segment display. It will then prompt the user to input what size block type they would like to use for the operation (1 for Byte, 2 for word, or 3 for double word). Next, the user is prompted to enter the number of blocks they would like to work with (4 hex digits). Next, the starting address is requested from the user (4 hex digits). After that data is collected the number of values per page is determined based on the data type selected. For a byte, 10 values are on each page, for a word, 20 values are on a page, and for a double word, 40 values are on each page. After all this data is calculated, the execute dump function is called. The execute dump will display the page number and the number of blocks being dumped in total at the top of the screen. After that, the values from RAM are displayed on the screen along with their respective addresses. After the appropriate number of values is printed onto the LCD, the user is asked if they would like to exit (press ‘E’), go to the next page (press ‘0’), or go to the previous page (press ‘1’). After the user exits from the function, they are returned to the main menu.

5.1.7 Edit:

When the user selects the edit function in the main menu, the setupEdit function is called. This function will print E on the 7-segment display. It will then prompt the user to enter the address (4 hex digits) that the user would like to edit in RAM. Once that data is taken in, the executeEdit function is called. The executeEdit function will then display address selected, the current value in that address. It will then prompt the user for the new value (2 hex digits). After the value is input, the value is written to RAM and read back and stored in a variable. The value that is read back is then printed to the LCD. This is done as an error checking measure to make sure what the user has written is actually in RAM or not. After the edit is complete, the user is then asked if they would like to move on to the next address in memory (press ‘0’) or to exit the function (press ‘1’). Once the user exits, they are returned to the main menu.

5.1.8 Find:

When the user selects the find function in the main menu, the setupFind function is called. This function will print ‘F’ on the 7-segment display. It will then prompt the user to input what size block type they would like to use for the operation (1 for Byte, 2 for word, or 3 for double word). Next, the user is prompted to enter the number of blocks they would like to work with (4 hex digits). It will also ask the user to input the starting address (4 hex digits) and the value to look for (2 hex digits). After all this data is collected, the executeFind function is called. Firstly, the proper size is calculated based on the data type. The user is then prompted with what value is being

looked for. After that is done, the user is presented with the first instance found of the value being looked for. The user is then given an option to select between exiting the function and returning to the main menu or to look at the next instance found. To exit the user must press ‘E’ on the keypad and to move to the next instance the user needs to select ‘0’. When the user is on the second page or greater the option to also go back a page becomes available and will need to press ‘1’ to do that. If no instances are found, then the user is prompted with a message that reads ‘Value not found.”

5.2 Sequential Diagram for Program

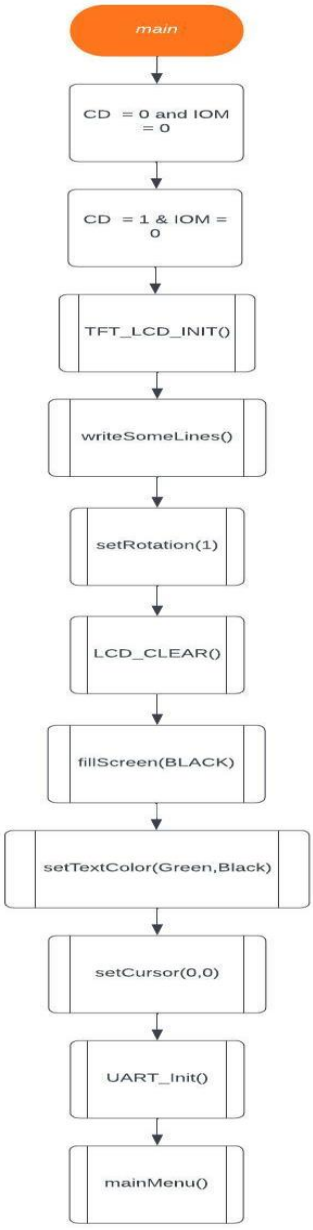


Figure 3: Flowchart for main()

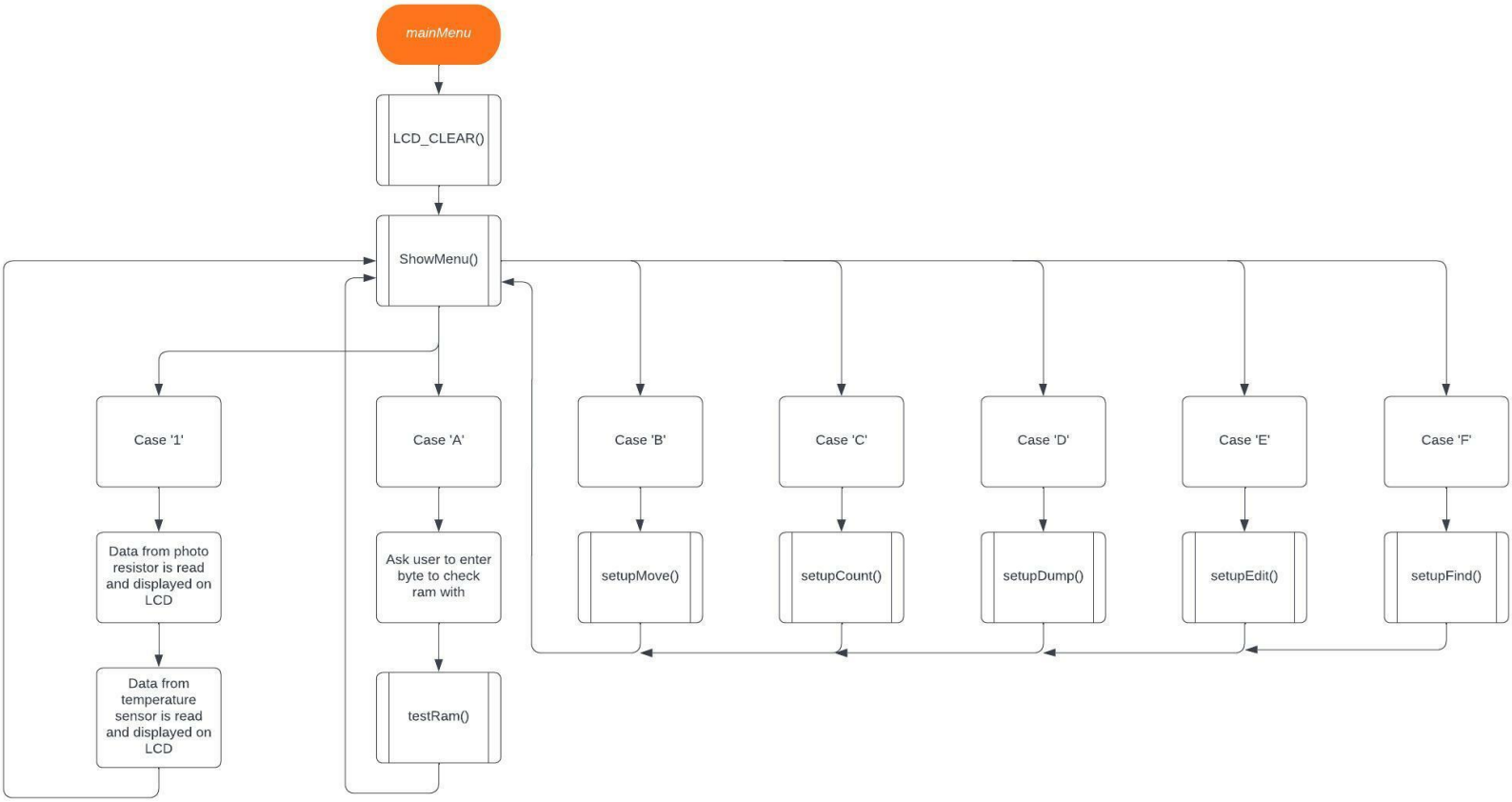


Figure 4: Flowchart for mainMenu()

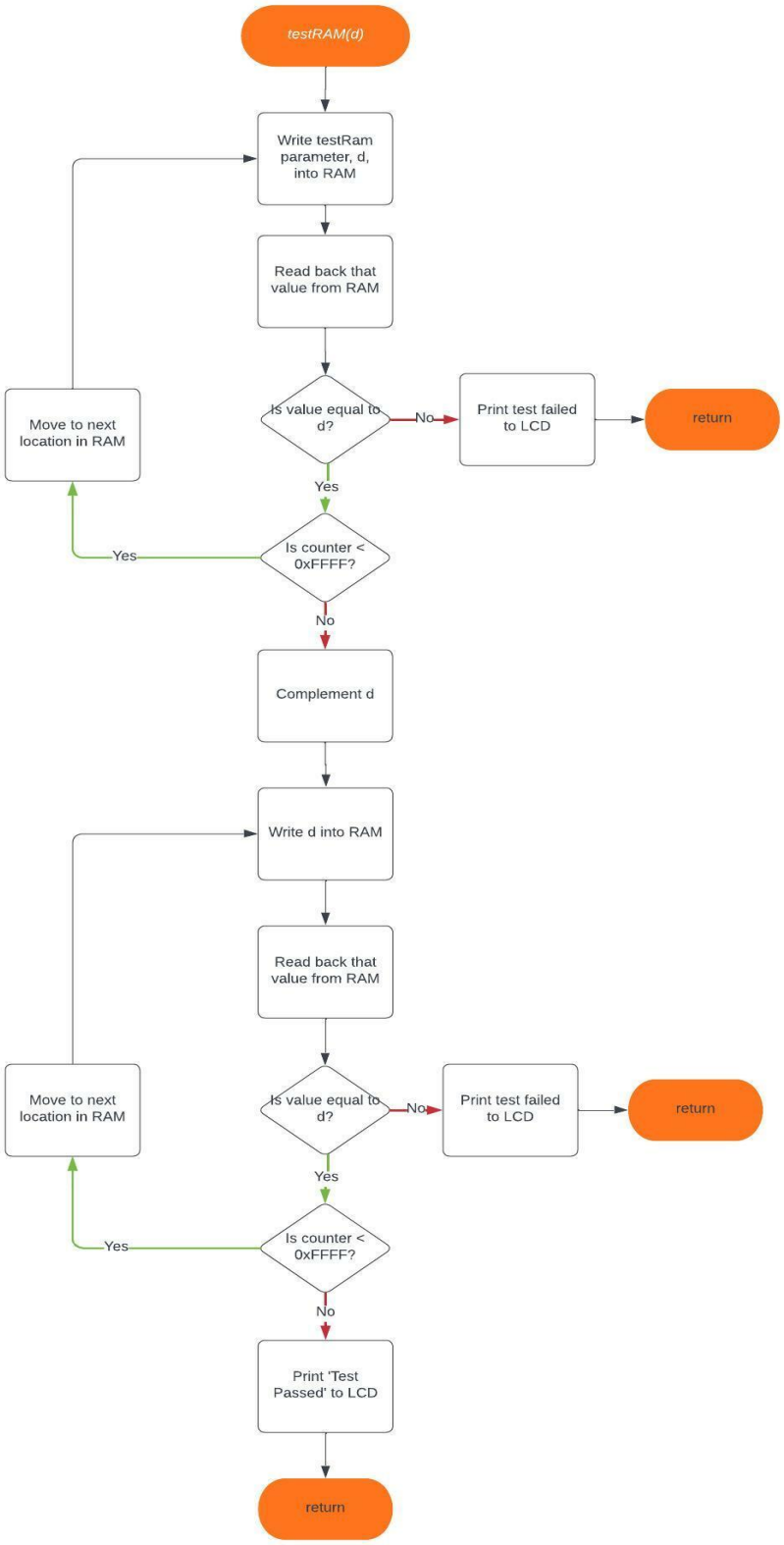


Figure 5: Flowchart for testRam()

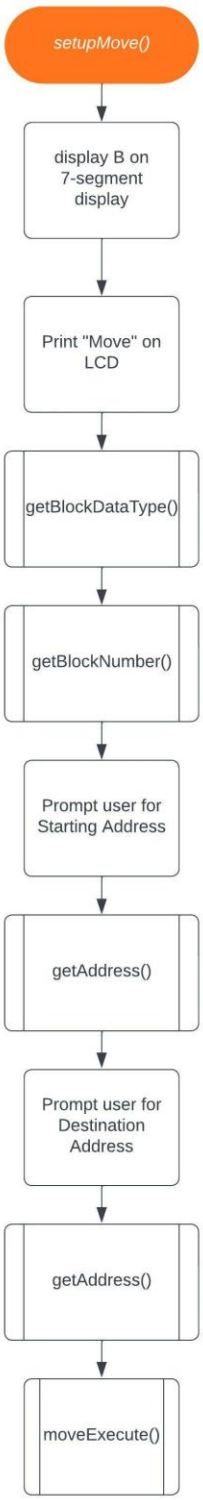


Figure 6: Flowchart for setupMove()

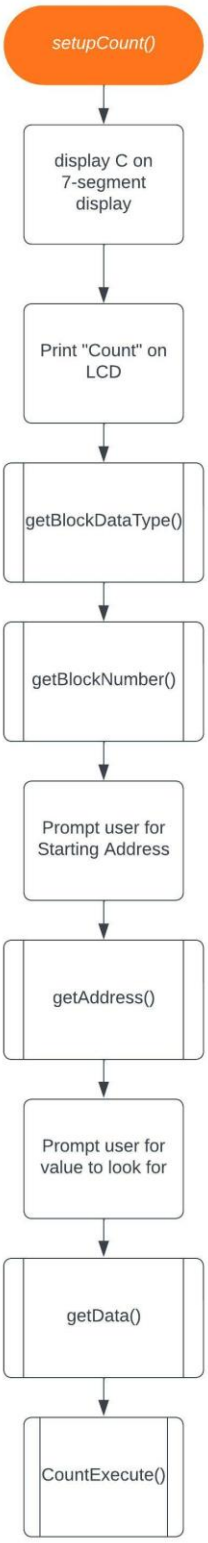


Figure 7: Flowchart for setupCount()

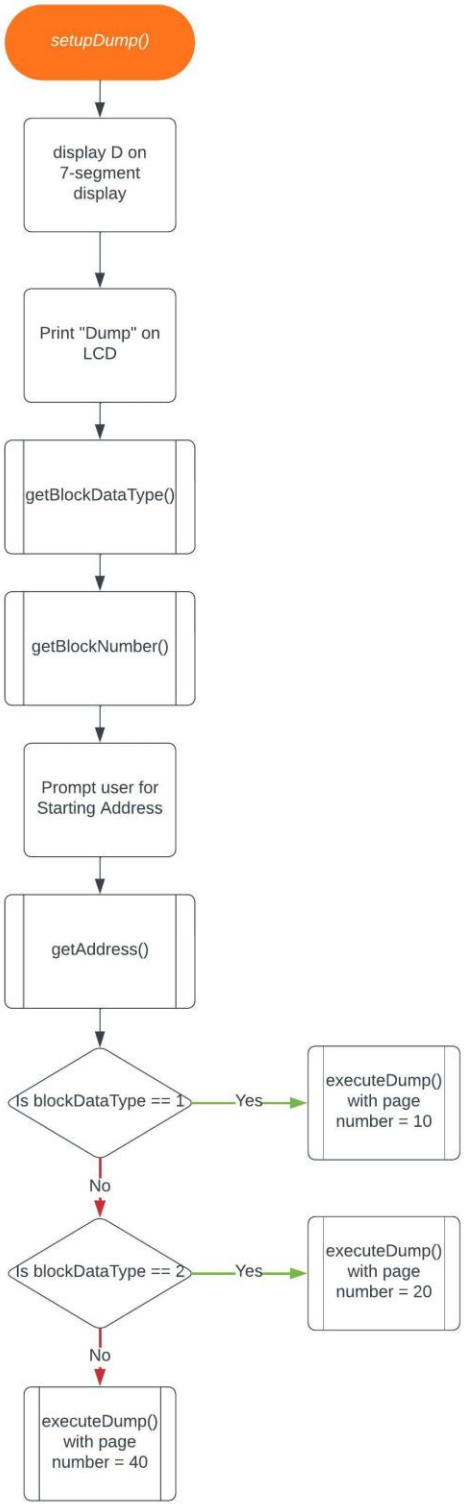


Figure 8: Flowchart for setupDump()

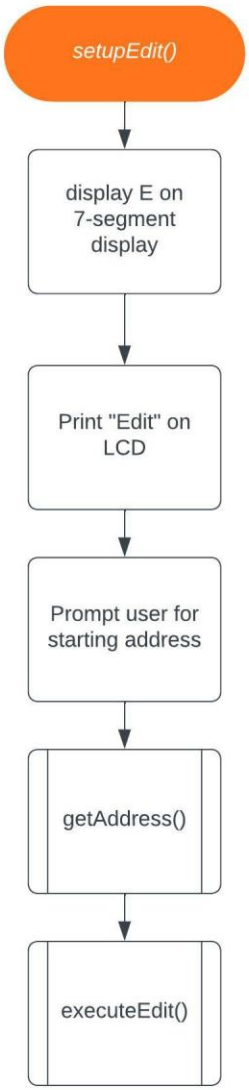


Figure 9: Flowchart for setupEdit()

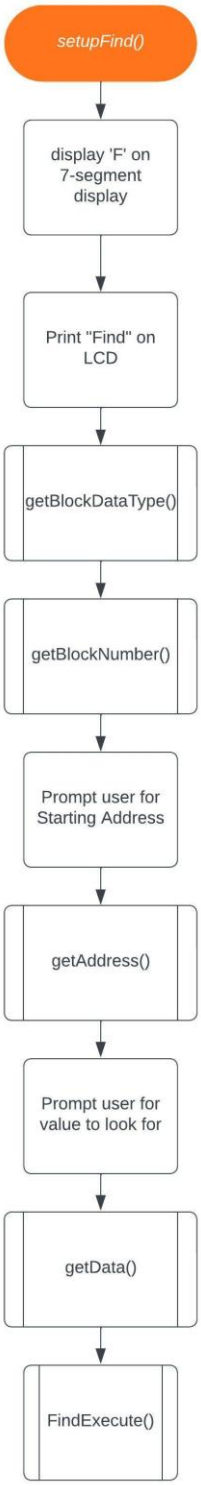
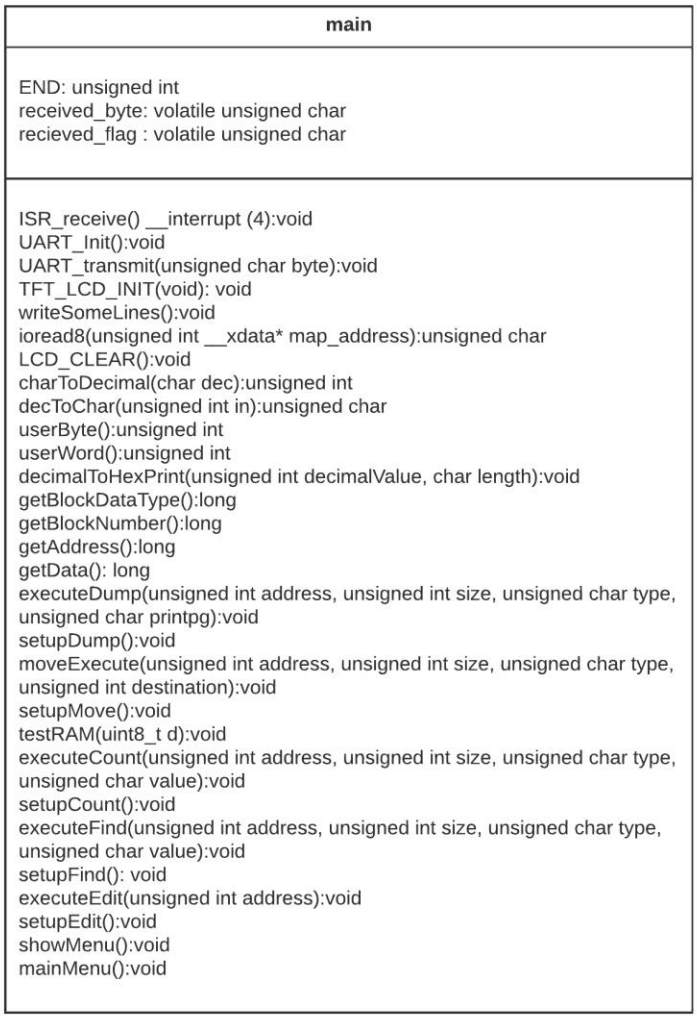


Figure 10: Flowchart for setupFind()

5.3 UML diagram



6 Problem Discussion

There were many issues that were faced throughout this lab both on the hardware and software side of things. The first issue that arose revolved around a ZIF socket that I had ordered for the AT89LP51 to make it easy to remove from the circuit when I wanted to reprogram it. The issue was that the ZIF socket would not fit into the footprint I have chosen of the microcontroller. To remedy this, I used headers that did fit the footprint and did allow me to connect the ZIF socket to them. However, the pins on the socket were not long enough to make a connection with the leads inside the header so there was no connection established between the microcontroller and the rest of the circuit. The issue was first identified when the LCD was connected, and nothing would happen. After various hypothesizes to why nothing was happening, I decided to check the continuity of the microcontroller with the rest of the PCB and found that there was an inconsistent connection. To remedy this issue, I had to remove the headers by desoldering them. The method used to desolder them was by using a desoldering pump.

The next issue that arose during this project was the fact that MCU IDE would not compile using SDCC on any of the computers that I have. So, after a few hours of trying to fix the issue with MCU I

eventually decided to find an alternative. The alternative I found, in my opinion, works better than MCU and is streamlined and easy to use. This alternative was visual studio code with the embedded IDE extension. This extension allowed for quick compilation and would directly create a hex file that is ready to be loaded onto the microcontroller. The only issue with this IDE is there is no support for simulations.

The third issue that occurred was the move function. I wrote the move function last as it was giving me some issues. All the logic seemed correct, but it was not performing the actions I was expecting it to perform on the microcontroller. The move function would occur, but it would not be reflected once the dump function was ran meaning the move never actually occurred. I traced this down to the fact that the C compiler will optimize things like this that it deems redundant to increase efficiency. This was resolved by setting some variables to volatile. After setting the variables to volatile, the performance of the function took a massive hit, but the move operation would actually occur and could be seen in the dump function.

I think that the issue with the ZIF socket is easily avoidable in the future, but it has also taught me to check the continuity as much as possible especially after connecting new components. The issue with the IDE was a great one to have as it gave me the chance to explore other IDEs and helped me find one that better suited my needs. The final issue regarding the move function helped me understand what volatile does in C and I believe will be especially useful in future embedded software design.

7 Conclusion

This project saw the design and creation of a PCB that contained an AT89LP51 with external clock and reset circuitry interfacing with 64K of RAM, 64K of ROM, a photoresistor & a temperature sensor each with its own analog to digital converter (ADC), LCD, 7-segment display, a keypad, and a ESP 8266 Wi-Fi module. All the memory chips and IO devices were selected using a GAL decoder that utilized some signals from the microcontroller to manage them. After the PCB was designed on EasyEDA and ordered on JLCPCB, software that implemented several RAM functions that were introduced earlier on during the semester were implemented using embedded C. The project was deemed successful when all the function behaved as expected, the Wi-Fi module was able to receive data, and the ADCs were changing values based on the stimuli presented to them.

This project was a great introduction to what the scale could look like for projects with a microcontroller. It demonstrated the potential of a relatively small microcontroller and what it could achieve. I personally think this is a great project in preparation for capstone as it introduces many new tools and concepts. Another important factor of this project is it allows us to practice organizational skills by making sure that every step to get the final product working is done and accounted for.

8 Appendix

8.1 Schematic Design

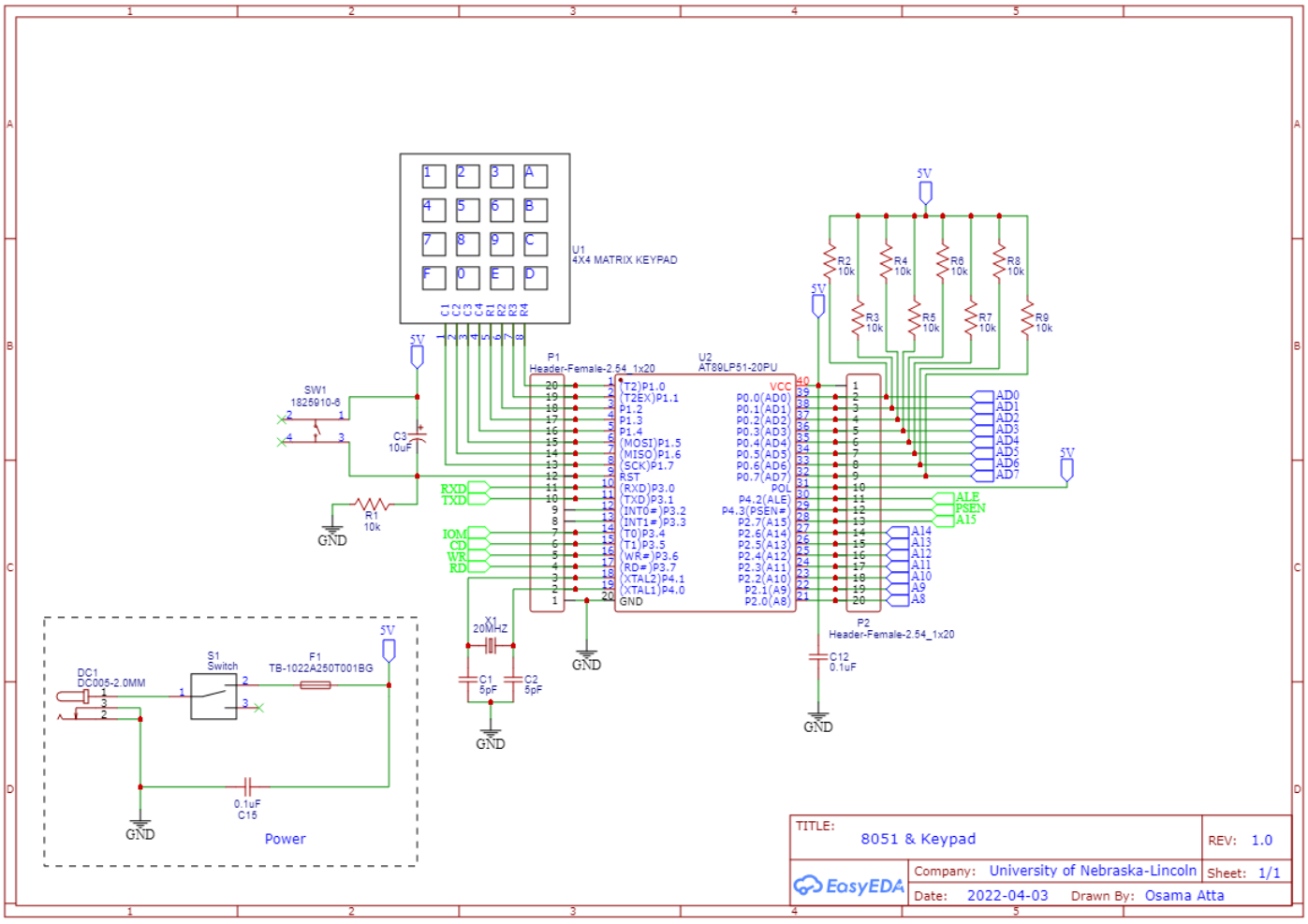


Figure 11: AT89LP51, reset circuitry, oscillator, power supply, and keypad

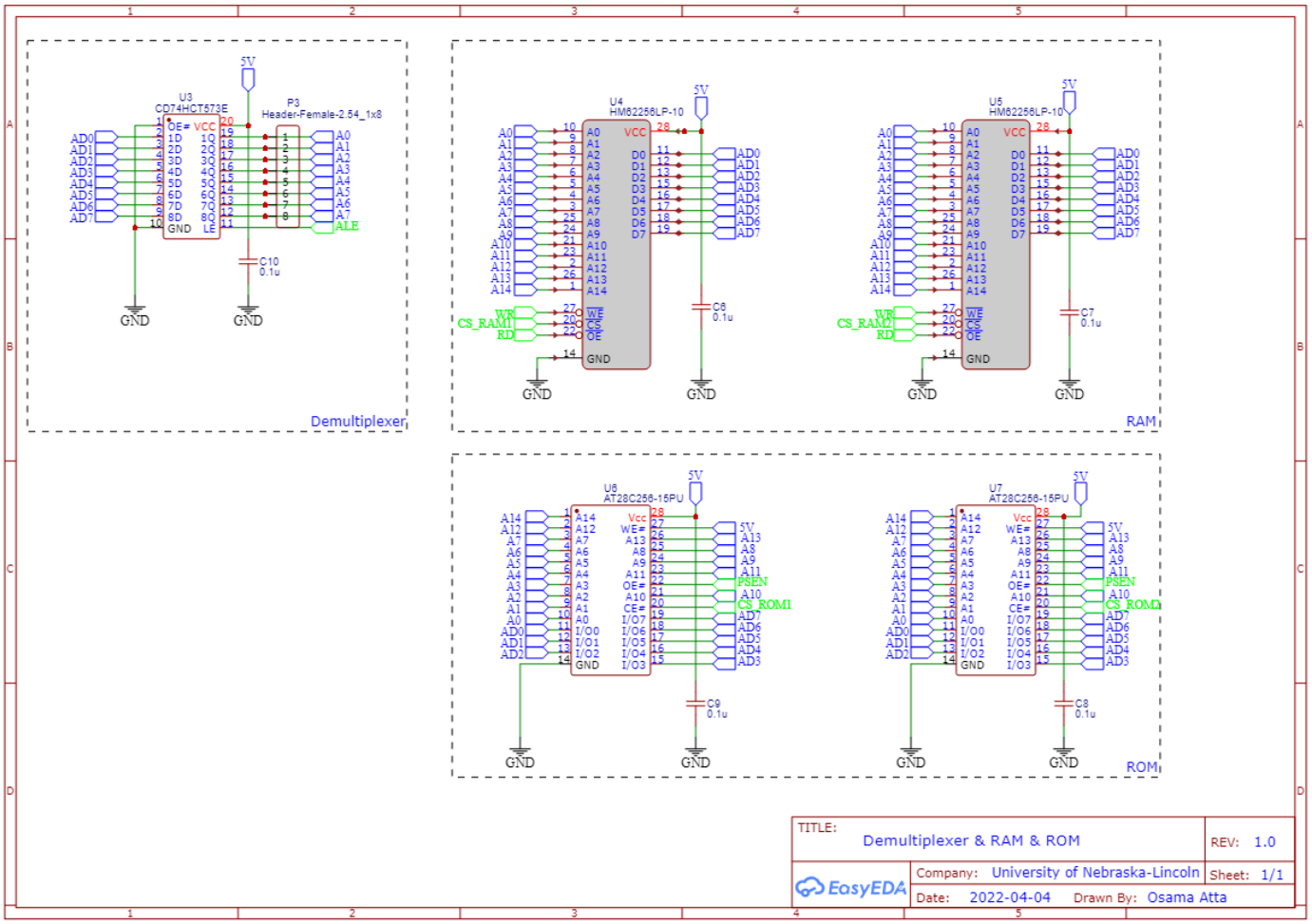


Figure 12: Demultiplexer, RAM, and ROM

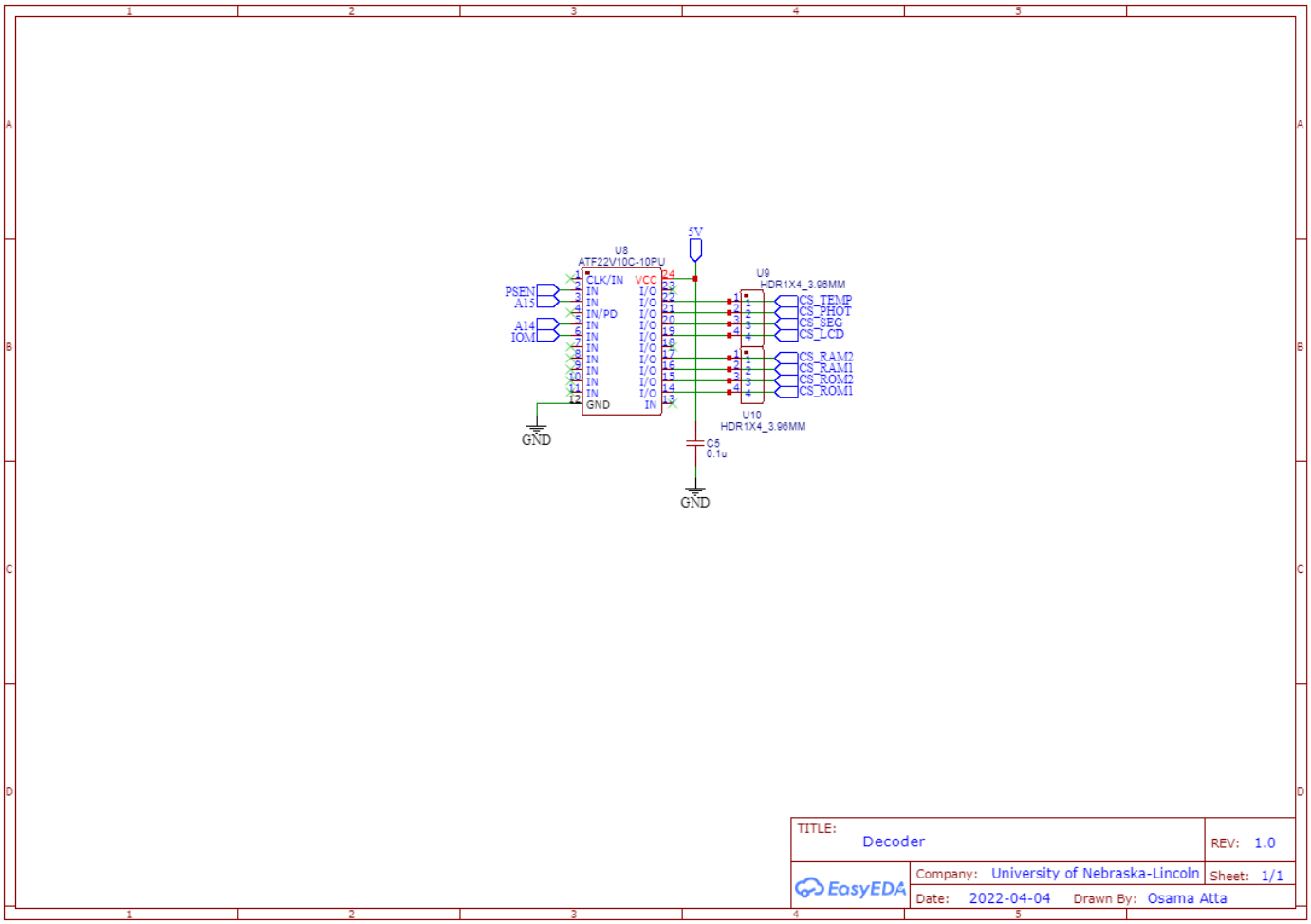


Figure 13: Decoder

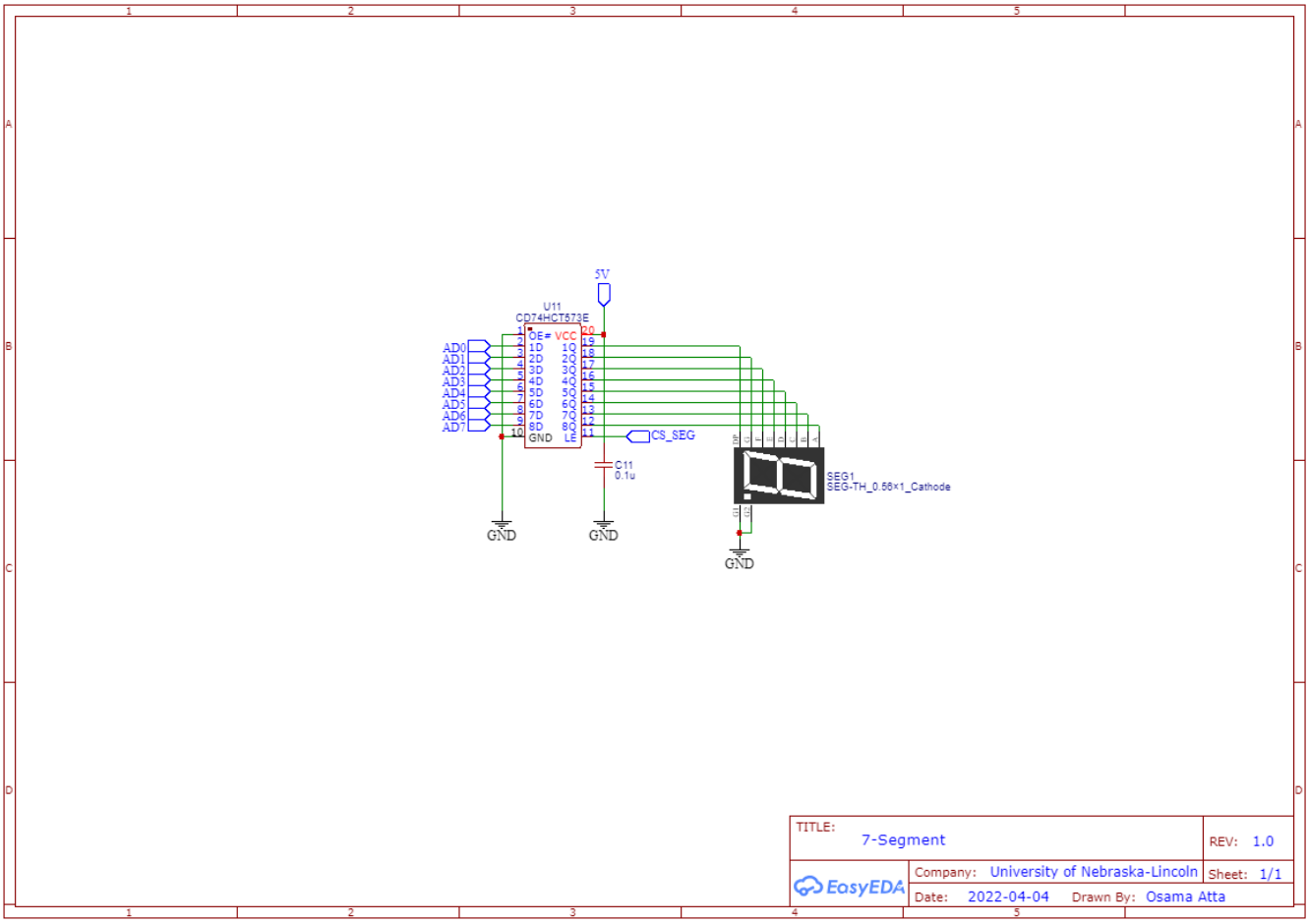


Figure 14: 7-segment display

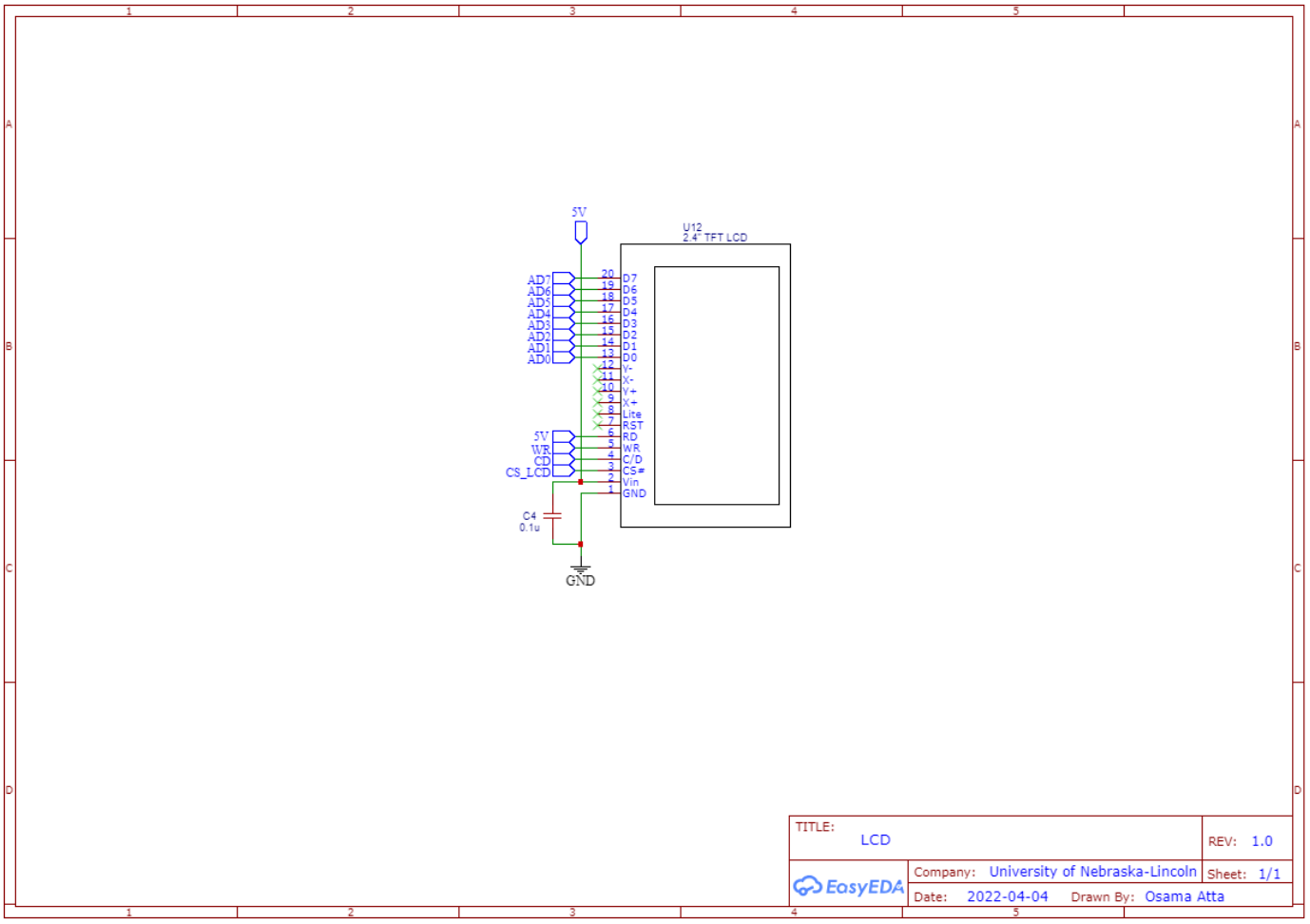


Figure 15: LCD

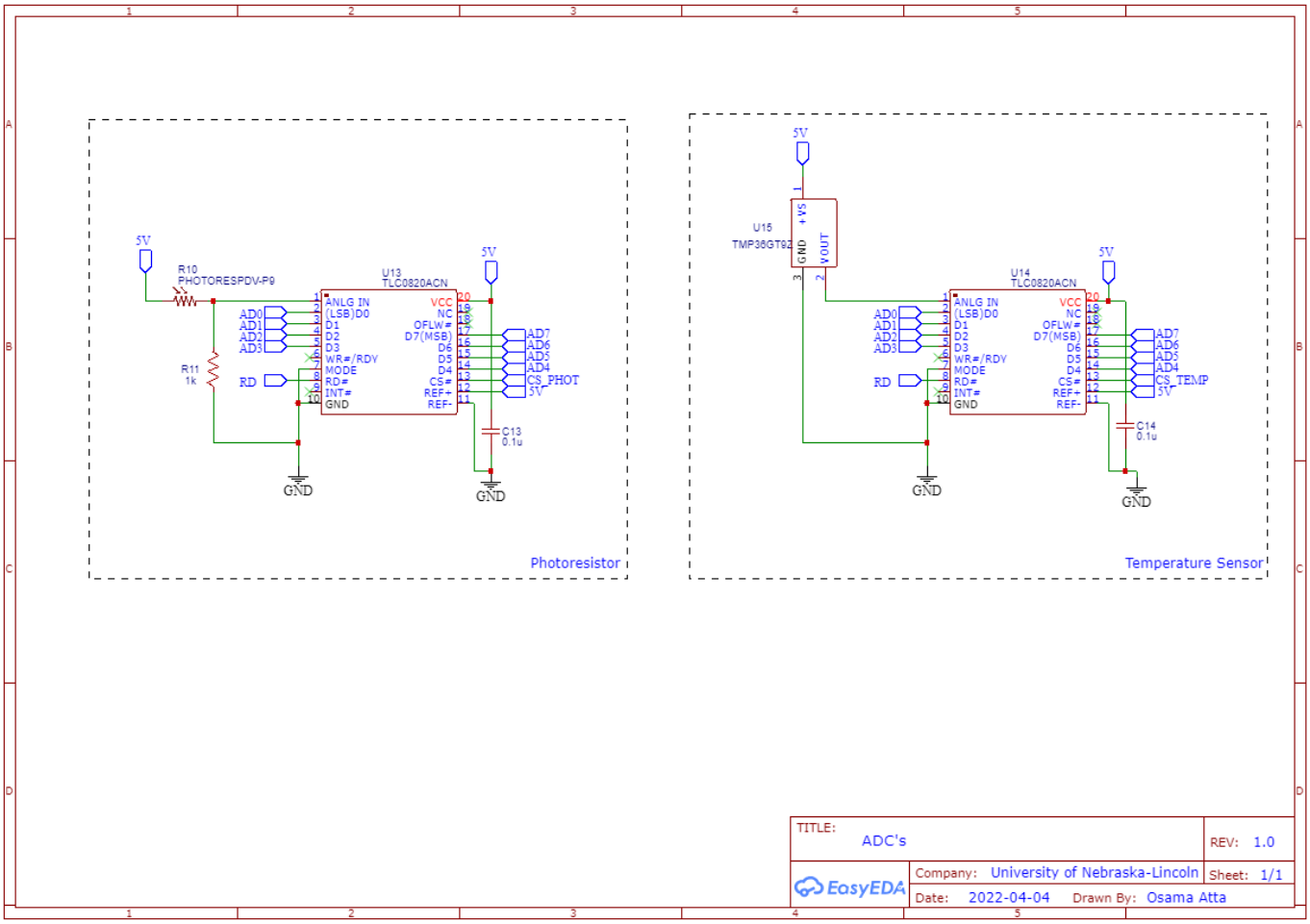


Figure 16: ADCs

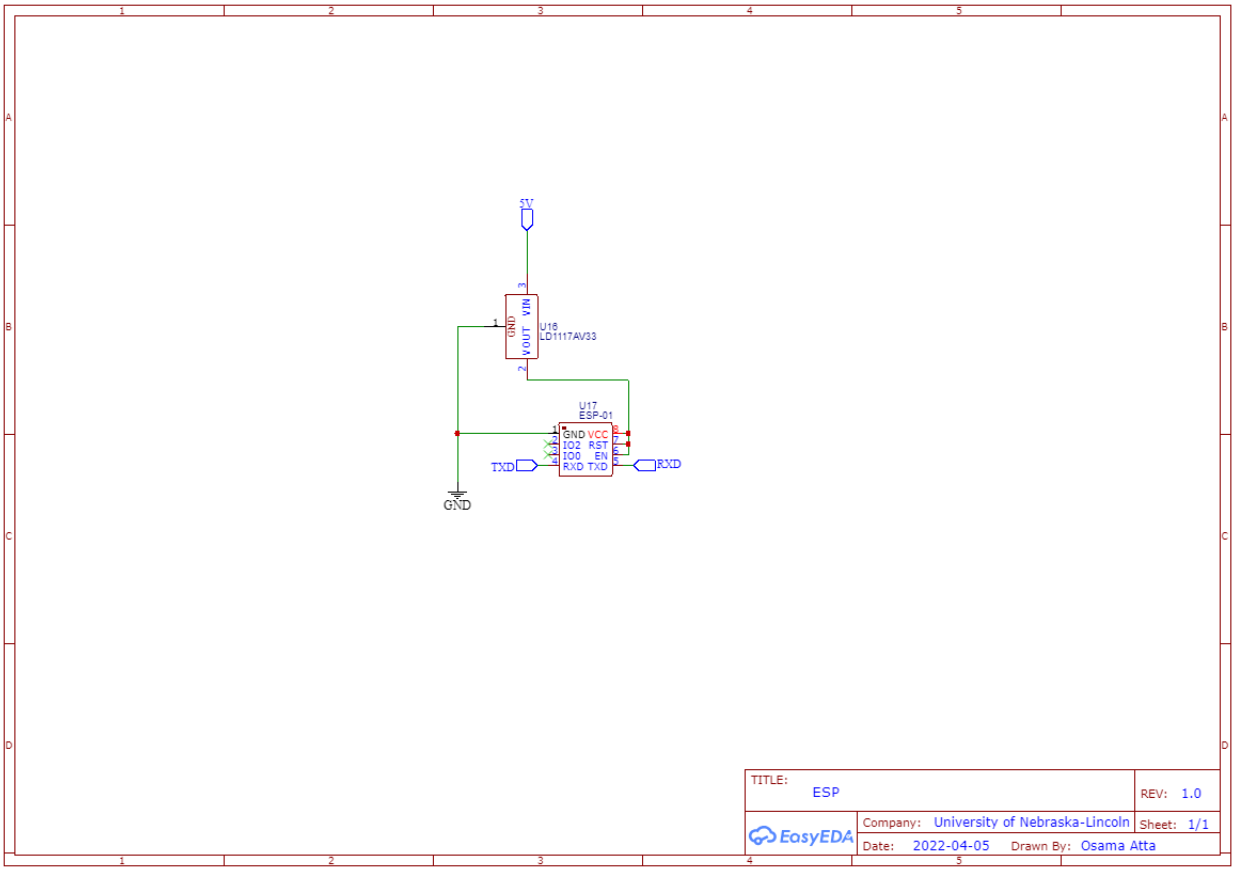
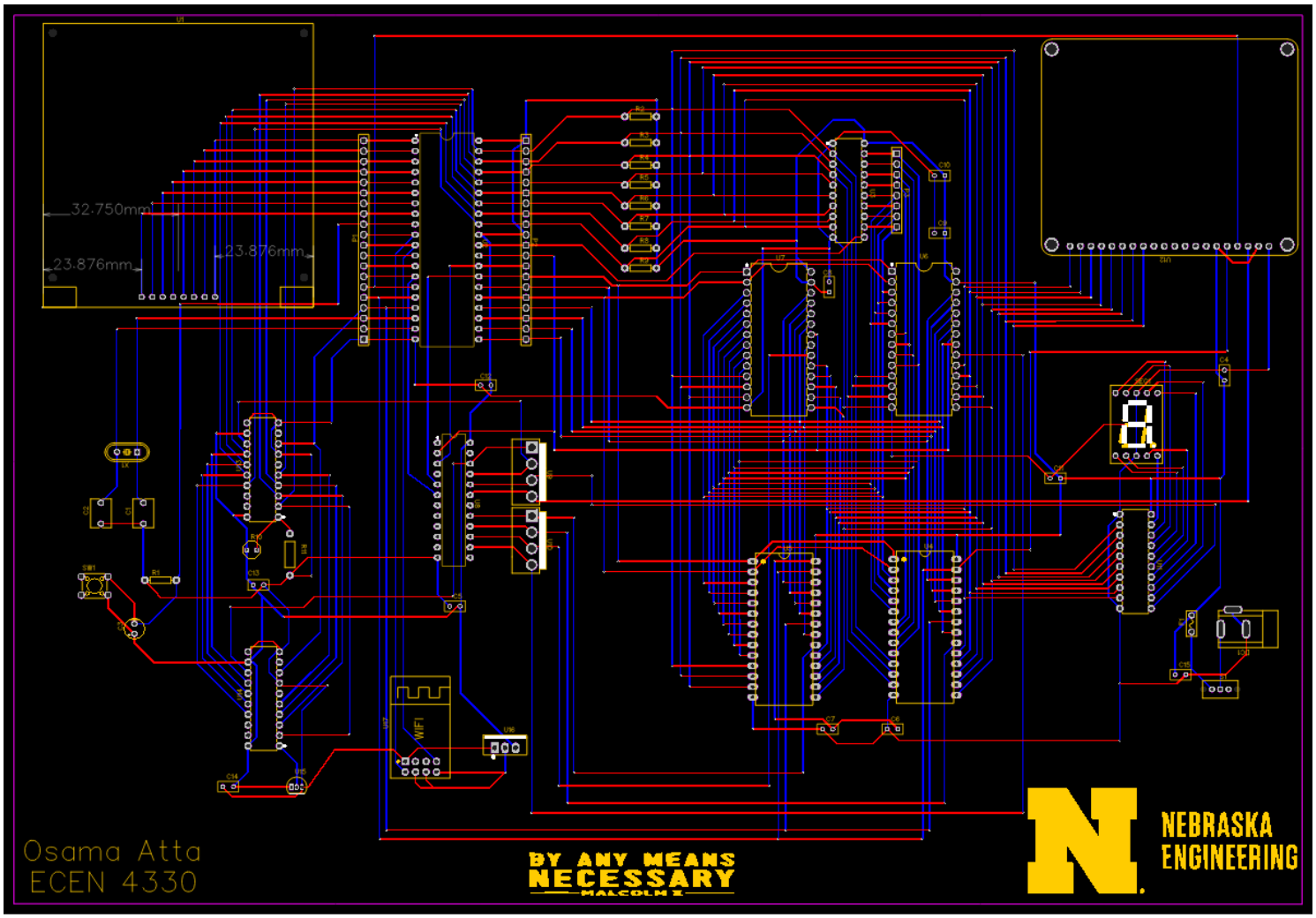


Figure 17: ESP 8266 Wi-Fi module

8.2 PCB Design



8.3 Bill of Materials

Part Name	Part Specification
Socket - 0.6"	40-pin processor
Socket - 0.6"	28-pin RAM
Socket - 0.3"	24-pin GAL
Socket - 0.3"	20-pin 573 & ADC
Socket - 0.3"	18-pin RTC
ZIF Socket	28-pin ROM
Microprocessor	AT89LP51
8-bit Latch/Demultiplexer	74HCT573
7-Segment LED (Common A)	**UC5621-11-R**
Crystal (Oscillator)	20 MHz
Crystal Capacitor	5pF
Slide Switch	SPDT
SPST Push-Button Switch	Reset Signal
Fuse	2A
8-pin Female header	Keypad mount
40-pin Male header	LCD & Keypad
16-pin Female header	LCD mount
24-pin Female header	Sensors
Capacitor	Electrolytic 10uF
Capacitor	Ceramic 0.1 uF
Resistor	1K ohm
Resistor	10K ohm
RAM 32kx8	62256-lp70
EEPROM 32kx8	AT28C256-15PU
GAL	AT22V10Q
Red LED	Power indicator
ADC	TLC0820ACN
DC power jack	2.1mm
Wall power adapter	5V DC
4x4 Keypad	
LCD	2.4" TFT LCD
ESP8266	WRL-17146
USB to UART	LC234X
Photoresistor	161
Temperature Sensor	TMP36

8.4 Demo Images

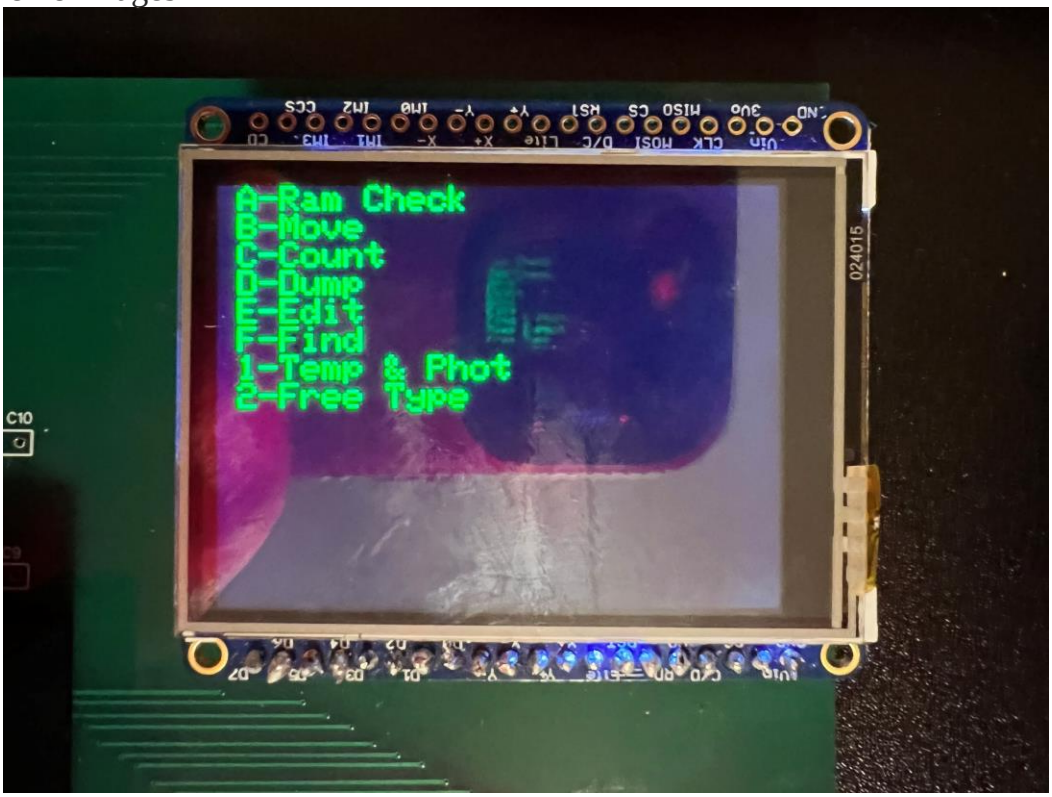


Figure 18: Main menu

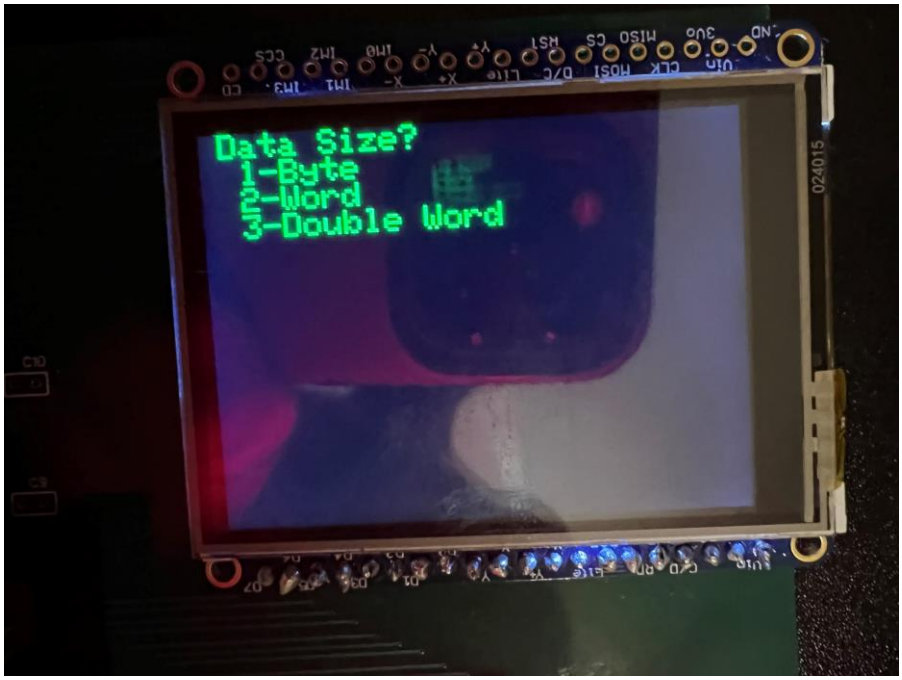


Figure 19: Data size prompt

Figure 20: Dump function starting at 8000h

Figure 21: Find function searching for 00h starting at 8000h

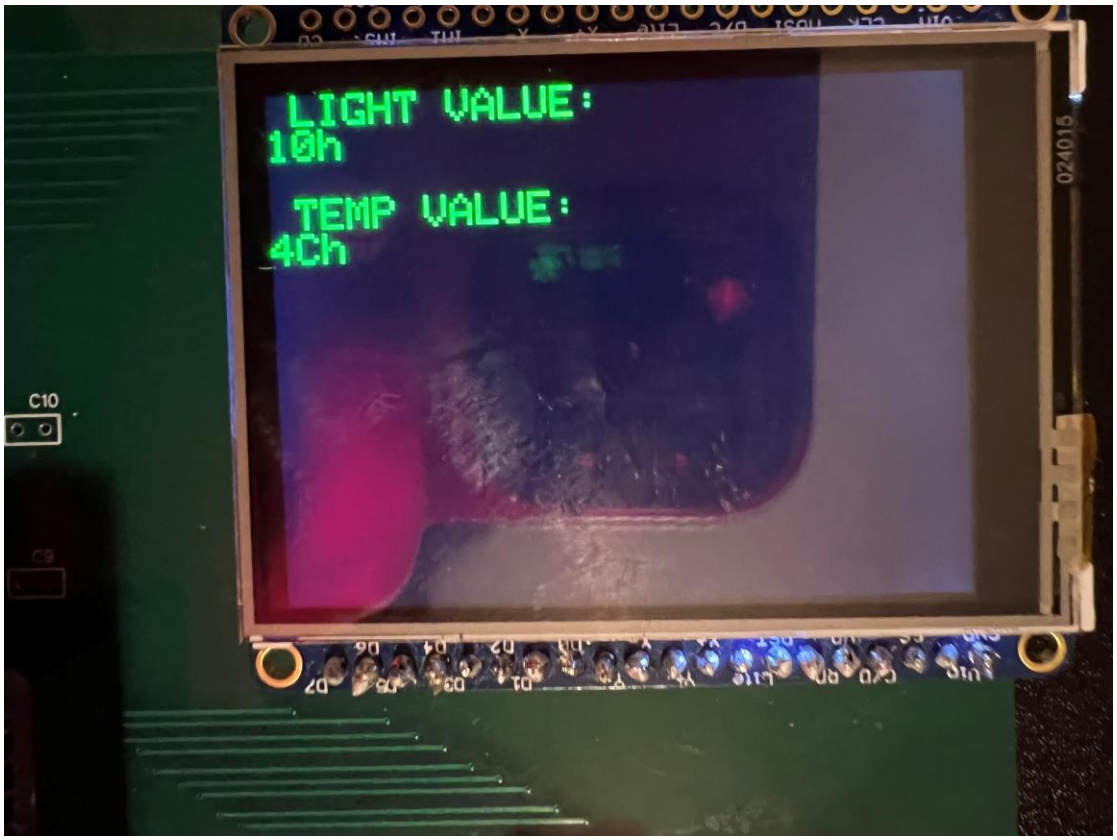


Figure 22: Reading from ADCs

8.5 Implementation Code for Microcontroller

```
/*
       edited by: Osama Atta
       author: Matthew Boeding
       version: v3.0
  Adapted from: Subharthi Banerjee, Ph.D.

       README

/// The sole reason to provide this code is to make your TFTLCD (ILI9341)
/// up and running

/// Note: Most of the code is in one place. This is not ideal and may be changed
// in the future

/// Use C or inline assembly program as you please.

/// ** the code uses P0 for 8-bit interface
/// ** IOM --> P3^4
/// ** CD --> P3^5
////  I recommend leaving these definitions for UART implementation later.
////
/// ** RD --> P3^7
/// ** WR --> P3^6

/// Refer to the header file to change decoding addresses for your specific design.
```

```
/// Please do not post any of the code from this course to GITHUB.

*/


/*//  *************** IMPORTANT ********************/

/* It may need redfinition of pins like*/
/*#include <8051.h> */


#include "ecen4330lcdh.h"
#include "font.h"

/* keypad configuration*/
uint8_t keypad[4][4] =     {{'D','E','0','F'},
                           {'C','9','8','7'},
                           {'B','6','5','4'},
                           {'A','3','2','1'} };
uint8_t colloc, rowloc;
/* store it in a variable the lcd address*/
__xdata uint8_t* lcd_address = (uint8_t __xdata*) __LCD_ADDRESS__;
__xdata uint8_t* seg7_address = (uint8_t __xdata*) __SEG_7_ADDRESS__;

unsigned int END = __END_RAM__; //65534


volatile unsigned char received_byte=0;
volatile unsigned char recieved_flag = 0;

 #define write8inline(d) {              \
       IOM = 1;                                              \
       *lcd_address = d;                                     \
       IOM = 0;                                              \
}

#define write8 write8inline
/* data write*/
#define write8DataInline(d) {     \
       CD = 1;                                  \
       write8(d);                               \
}
/* command or register write*/
#define write8RegInline(d) {      \
       CD = 0;                                  \
       write8(d);                               \
}

/* inline definitions*/
#define write8Reg write8RegInline
#define write8Data write8DataInline

uint16_t cursor_x, cursor_y;  /*// cursor_y and cursor_x globals*/
uint8_t textsize, rotation; /*// textsize and rotation*/
uint16_t
    textcolor,      /*///< 16-bit background color for print()*/
    textbgcolor;    /*///< 16-bit text color for print()*/
uint16_t
    _width,         /*///< Display width as modified by current rotation*/
    _height;        /*///< Display height as modified by current rotation*/
```

```
void ISR_receive() __interrupt (4) {
       if (RI == 1){
              //received_byte = SBUF;
              received_byte = SBUF - 0x40;
              RI = 0;
              recieved_flag= 1;

       }
}

void iowrite8(uint8_t __xdata* map_address, uint8_t d) {
       IOM = 1;
       *map_address = d;
       *map_address = d;
       *map_address = d;
       *map_address = d;
       IOM = 0;
}

void delay (int16_t d) /*// x 1ms*/
{
       int i,j;
       for (i=0;i<d;i++) /*// this is For(); loop delay used to define delay value in
Embedded C*/
       {
       for (j=0;j<1000;j++);
       }
}

void writeRegister8(uint8_t a, uint8_t d) {
       /*IOM = 0;*/
       CD = __CMD__;
       write8(a);
       CD = __DATA__;
       write8(d);
       /*IOM = 1;*/
}

void writeRegister16(uint16_t a, uint16_t d){
       uint8_t hi, lo;
       hi = (a) >> 8;
       lo = (a);
       /*/IOM = 0;
   //  CD = 0;*/
       write8Reg(hi);
       write8Reg(lo);
       hi = (d) >> 8;
       lo = (d);
       CD = 1 ;
       write8Data(hi);
       write8Data(lo);
/*/    IOM =1;*/
}

void UART_Init(){
    SCON = 0x50;  // Asynchronous mode, 8-bit data and 1-stop bit
    TMOD = 0x20;  // Timer1 in Mode2. in 8 bit auto reload
    TH1 =  0xFB;  // Load timer value for 9600 baudrate
    TR1 = 1;      // Turn ON the timer for Baud rate generation
    ES  = 1;      // Enable Serial Interrupt
```

```
    EA  = 1;      // Enable Global Interrupt bit
}


void UART_transmit(unsigned char byte){
    SBUF = byte;
    while(TI == 1);
    TI = 0;
}


void setCursor(uint16_t x, uint16_t y){
      cursor_x = x;
      cursor_y = y;
}
/* set text color*/
void setTextColor(uint16_t x, uint16_t y){
      textcolor =  x;
      textbgcolor = y;
}

/* set text size*/
void setTextSize(uint8_t s){
      if (s > 8) return;
      textsize = (s>0) ? s : 1 ;
}

void setRotation(uint8_t flag){
      switch(flag) {
            case 0:
                  flag = (ILI9341_MADCTL_MX | ILI9341_MADCTL_BGR);
                  _width = TFTWIDTH;
                  _height = TFTHEIGHT;
                  break;
            case 1:
                  flag = (ILI9341_MADCTL_MV | ILI9341_MADCTL_BGR);
                  _width = TFTHEIGHT;
                  _height = TFTWIDTH;
                  break;
            case 2:
                  flag = (ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR);
                  _width = TFTWIDTH;
                  _height = TFTHEIGHT;
                  break;
        case 3:
                  flag = (ILI9341_MADCTL_MX | ILI9341_MADCTL_MY | ILI9341_MADCTL_MV
| ILI9341_MADCTL_BGR);
                  _width = TFTHEIGHT;
                  _height = TFTWIDTH;
                  break;
            default:
                  flag = (ILI9341_MADCTL_MX | ILI9341_MADCTL_BGR);
                  _width = TFTWIDTH;
                  _height = TFTHEIGHT;
                  break;
      }
      writeRegister8(ILI9341_MEMCONTROL, flag);
}

/* set address definition*/
void setAddress(uint16_t x1,uint16_t y1,uint16_t x2,uint16_t y2){
```

```
        /*IOM = 0;*/
        write8Reg(0x2A);
        write8Data(x1 >> 8);
        write8Data(x1);
        write8Data(x2 >> 8);
        write8Data(x2);

        write8Reg(0x2B);
        write8Data(y1 >> 8);
        write8Data(y1);
        write8Data(y2 >> 8);
        write8Data(y2);
        /*write8Reg(0x2C);*/
    /*/IOM =1;*/


}

void TFT_LCD_INIT(void){
        /*char ID[5];*/
        /*//int id;*/
        _width = TFTWIDTH;
        _height = TFTHEIGHT;

        /*all low*/
        IOM = 1;
        /*RDN = 1;*/
        CD = 1;

        write8Reg(0x00);
        write8Data(0x00);write8Data(0x00);write8Data(0x00);
        /*IOM = 1;*/
        delay(200);

        /*IOM = 0;*/

        writeRegister8(ILI9341_SOFTRESET, 0);
    delay(50);
    writeRegister8(ILI9341_DISPLAYOFF, 0);
    delay(10);




    writeRegister8(ILI9341_POWERCONTROL1, 0x23);
    writeRegister8(ILI9341_POWERCONTROL2, 0x11);
    write8Reg(ILI9341_VCOMCONTROL1);
              write8Data(0x3d);
              write8Data(0x30);
    writeRegister8(ILI9341_VCOMCONTROL2, 0xaa);
    writeRegister8(ILI9341_MEMCONTROL, ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR);
    write8Reg(ILI9341_PIXELFORMAT);
        write8Data(0x55);write8Data(0x00);
    writeRegister16(ILI9341_FRAMECONTROL, 0x001B);

    writeRegister8(ILI9341_ENTRYMODE, 0x07);
    /* writeRegister32(ILI9341_DISPLAYFUNC, 0x0A822700);*/
```

```
    writeRegister8(ILI9341_SLEEPOUT, 0);
    delay(150);
    writeRegister8(ILI9341_DISPLAYON, 0);
    delay(500);
             setAddress(0,0,_width-1,_height-1);
     /* ************* Start Initial Sequence ILI9341 controller **********/

    IOM = 0;
}
void drawPixel(uint16_t x3,uint16_t y3,uint16_t color1)
{

       /* not using to speed up*/
       /*/if ((x3 < 0) ||(x3 >= TFTWIDTH) || (y3 < 0) || (y3 >= TFTHEIGHT))
       //{
       //     return;
       //}*/
       setAddress(x3,y3,x3+1,y3+1);

       /*/IOM = 0;*/

    CD=0; write8(0x2C);

       CD = 1;
       write8(color1>>8);write8(color1);
       /*IOM = 1;*/
}



void fillRect(uint16_t x,uint16_t y,uint16_t w,uint16_t h,uint16_t color){
       if ((x >= TFTWIDTH) || (y >= TFTHEIGHT))
       {
              return;
       }

       if ((x+w-1) >= TFTWIDTH)
       {
              w = TFTWIDTH-x;
       }

       if ((y+h-1) >= TFTHEIGHT)
       {
              h = TFTHEIGHT-y;
       }

       setAddress(x, y, x+w-1, y+h-1);
    /*IOM = 0;*/


       write8Reg(0x2C);
       /*IOM = 1; IOM = 0;*/
       CD = 1;
       for(y=h; y>0; y--)
       {
              for(x=w; x>0; x--)
              {

                     write8(color>>8); write8(color);
```

```
            }
        }
        /*IOM = 1;*/
}

void fillScreen(uint16_t Color){
        /*uint8_t VH,VL;*/
        long len = (long)TFTWIDTH * (long)TFTHEIGHT;

         int blocks;

   uint8_t  i, hi = Color >> 8,
              lo = Color;

    blocks = (uint16_t)(len / 64); /* 64 pixels/block*/
       setAddress(0,0,TFTWIDTH-1,TFTHEIGHT-1);

       /*IOM = 0;*/


       write8Reg(0x2C);
       /*IOM = 1; IOM = 0;*/
              CD = 1;
              write8(hi); write8(lo);

              len--;
              while(blocks--) {
      i = 16; /* 64 pixels/block / 4 pixels/pass*/
      do {

                           write8(hi); write8(lo);write8(hi); write8(lo);
                           write8(hi); write8(lo);write8(hi); write8(lo);
      } while(--i);
    }
    for(i = (char)len & 63; i--; ) {

      write8(hi); write8(lo);

    }

       /*IOM = 1;*/

}

void drawChar(int16_t x, int16_t y, uint8_t c,uint16_t color, uint16_t bg, uint8_t
size){
        uint8_t i = 0;
        uint8_t j = 0;
        if ((x >=TFTWIDTH) || /* Clip right*/
            (y >=TFTHEIGHT)           || /* Clip bottom*/
            ((x + 6 * size - 1) < 0) || /* Clip left*/
            ((y + 8 * size - 1) < 0))   /* Clip top*/
        {
              return;
        }

        for (i=0; i<6; i++ )
        {
              uint8_t line;

              if (i == 5)
```

```
		{
			line = 0x0;
		}
		else
		{
			line = pgm_read_byte(font+(c*5)+i);
		}

		for (j = 0; j<8; j++)
		{
			if (line & 0x1)
			{
				if (size == 1) /* default size*/
				{
					drawPixel(x+i, y+j, color);
				}
				else {  /* big size*/
					fillRect(x+(i*size), y+(j*size), size, size, color);
				}
			} else if (bg != color)
			{
				if (size == 1) /*default size*/
				{
					drawPixel(x+i, y+j, bg);
				}
				else
				{  /* big size*/
					fillRect(x+i*size, y+j*size, size, size, bg);
				}
			}

			line >>= 1;
		}
	}

}

void write(uint8_t c)/*write a character at setted coordinates after setting location
and colour*/
{
	if (c == '\n')
	{
		cursor_y += textsize*8;
		cursor_x  = 0;
	}
	else if (c == '\r')
	{
		/* skip em*/
	}
	else
	{
		drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize);
		cursor_x += textsize*6;
	}
}

void LCD_string_write(int8_t *str)
{
	int16_t i;
	for(i=0;str[i]!=0;i++)	/* Send each char of string till the NULL */
	{
```

```
		write(str[i]);		/* Call transmit data function */
	}
}



void freeType() {
	uint8_t count = 0;
	uint8_t d;
	while(1){


		if (count == 8) {
			d = '\n';
			count = 0;
			write(d);
		}
		else{
			d = keyDetect();
			write(d);
		}

		count++;
	}
}

uint8_t keyDetect(){
	__KEYPAD_PORT__=0xF0;					/*set port direction as input-output*/
	do
	{
		__KEYPAD_PORT__ = 0xF0;
		colloc = __KEYPAD_PORT__;
		colloc&= 0xF0;		/* mask port for column read only */
	}while(colloc != 0xF0);			/* read status of column */

	do
	{
		do
		{
			if (recieved_flag == 1){
				recieved_flag = 0;
				return received_byte; // this will return value from ESP
when keydetect is being called
			}

			delay(20);	/* 20ms key debounce time */
			colloc = (__KEYPAD_PORT__ & 0xF0);		/* read status of column
*/
		}while(colloc == 0xF0);	/* check for any key press */

		delay(1);
		colloc = (__KEYPAD_PORT__ & 0xF0);
	}while(colloc == 0xF0);
	while(1)
	{
		/* now check for rows */
		__KEYPAD_PORT__= 0xFE;
			/* check for pressed key in 1st row */
		colloc = (__KEYPAD_PORT__ & 0xF0);
		if(colloc != 0xF0)
		{
```

```
                        rowloc = 0;
                        break;
                }

                __KEYPAD_PORT__ = 0xFD;
        /* check for pressed key in 2nd row */
                colloc = (__KEYPAD_PORT__ & 0xF0);
                if(colloc != 0xF0)
                {
                        rowloc = 1;
                        break;
                }

        __KEYPAD_PORT__ = 0xFB;                 /* check for pressed key in 3rd row */
        colloc = (__KEYPAD_PORT__ & 0xF0);
        if(colloc != 0xF0)
        {
                rowloc = 2;
                break;
        }

        __KEYPAD_PORT__ = 0xF7;                 /* check for pressed key in 4th row */
        colloc = (__KEYPAD_PORT__ & 0xF0);
        if(colloc != 0xF0)
        {
                rowloc = 3;
                break;
        }
}

        if(colloc == 0xE0)
        {
                return(keypad[rowloc][0]);
        }
        else if(colloc == 0xD0)
        {
                return(keypad[rowloc][1]);
        }
        else if(colloc == 0xB0)
        {
                return(keypad[rowloc][2]);
        }
        else
        {
                return(keypad[rowloc][3]);
        }
}


void writeSomeLines(){
        setRotation(1);
  fillScreen(BLACK);

        setTextSize(5);
        setTextColor(CYAN, BLACK);
        LCD_string_write("Welcome\n");
        setTextSize(2);
        LCD_string_write("Osama Atta\n");
        LCD_string_write("ECEN-4330\n"); //welcome screen
        delay(200);
}
```

```
/*=============================================================================================
============================================== */
unsigned char ioread8(unsigned int __xdata* map_address) {
	unsigned char d = 0;
	IOM = 1;
	delay(50);
	d = *map_address;
	d = *map_address;
	d = *map_address; //three times for possible delay in IOM
	IOM = 0;
	return d;
}
/*=============================================================================================
============================================== */
void LCD_CLEAR() {
	fillScreen(BLACK);
	setCursor(0,0);
}
/*=============================================================================================
============================================== */
unsigned int charToDecimal(char dec){
	if (dec == '0'){
		return 0;
	}
	else if (dec == '1'){
		return 1;
	}
	else if (dec == '2'){
		return 2;
	}
	else if (dec == '3'){
		return 3;
	}
	else if (dec == '4'){
		return 4;
	}
	else if (dec == '5'){
		return 5;
	}
	else if (dec == '6'){
		return 6;
	}
	else if (dec == '7'){
		return 7;
	}
	else if (dec == '8'){
		return 8;
	}
	else if (dec == '9'){
		return 9;
	}
	else if (dec == 'A'){
		return 10;
	}
	else if (dec == 'B'){
		return 11;
	}
	else if (dec == 'C'){
		return 12;
	}
	else if (dec == 'D'){
```

```
		return 13;
	}
	else if (dec == 'E'){
		return 14;
	}
	else{
		return 15;
	}
}
/*============================================================================================
============================================== */
unsigned char decToChar(unsigned int in) {
	if (in == 0){
		return '0';
	}
	else if (in == 1){
		return '1';
	}
	else if (in == 2){
		return '2';
	}
	else if (in == 3){
		return '3';
	}
	else if (in == 4){
		return '4';
	}
	else if (in == 5){
		return '5';
	}
	else if (in == 6){
		return '6';
	}
	else if (in == 7){
		return '7';
	}
	else if (in == 8){
		return '8';
	}
	else if (in == 9){
		return '9';
	}
	else if (in == 10){
		return 'A';
	}
	else if (in == 11){
		return 'B';
	}
	else if (in == 12){
		return 'C';
	}
	else if (in == 13){
		return 'D';
	}
	else if (in == 14){
		return 'E';
	}
	else if(in == 15){
		return 'F';
	}
    else{
```

```
        return 1000;
    }
}
/*=============================================================================================
============================================== */
unsigned int userByte() { //check difference between xdata and internal in execution
      char num0, num1;

      num0 = keyDetect(); //take in value
      write(num0); //print the value entered
      num1 = keyDetect();
      write(num1);
      LCD_string_write("\n");

      return ((charToDecimal(num0)*16) + (charToDecimal(num1)*1)); //hex
}
/*=============================================================================================
============================================== */
unsigned int userWord() { //check difference between xdata and internal in execution
       char value0, value1, value2, value3;

      value0 = keyDetect(); //take in value
      write(value0); //print value entered
      value1 = keyDetect();
      write(value1);
      value2 = keyDetect();
      write(value2);
      value3 = keyDetect();
      write(value3);
      LCD_string_write("\n");

      return (charToDecimal(value0)*4096) + (charToDecimal(value1)*256) +
(charToDecimal(value2)*16) + (charToDecimal(value3)*1); //hex
}
/*=============================================================================================
============================================== */
void decimalToHexPrint(unsigned int decimalValue, char length){ //no array decimal to
hex print function

    unsigned int divValue0, divValue1, divValue2, divValue3;

    unsigned int temp = decimalValue;

    divValue3 = temp / 4096 ;
    temp = temp % 4096;

    divValue2 = temp / 256 ;
    temp = temp % 256;

    divValue1 = temp / 16;
    temp = temp % 16 ;

    divValue0 = temp / 1 ;
    temp = temp % 1 ;

    if(length == 4){ //word
        write(decToChar(divValue3));
             write(decToChar(divValue2));
             write(decToChar(divValue1));
             write(decToChar(divValue0));
             write('h');
```

```
    }
    else if(length == 2){ //byte
        write(decToChar(divValue1));
              write(decToChar(divValue0));
        write('h');
    }

}
/*============================================================================================
============================================== */
long getBlockDataType() {
       __xdata unsigned char takenIn;

       do {
              LCD_CLEAR();
              __code unsigned char* ask = " Data Size?\n  1-Byte\n  2-Word\n  3-Double
Word\n";
              LCD_string_write(ask);

              takenIn = keyDetect();
              write(takenIn); //print input
              delay(100);
       } while (takenIn != '1' && takenIn != '2' && takenIn != '3');      //stay in the
loop until a valid selection is made

       return takenIn;
}
/*============================================================================================
============================================== */
long getBlockNumber(){
    __xdata unsigned int input;
    do{
        LCD_CLEAR();
        LCD_string_write(" Input block number\n");
        input = userWord();
    }while(input <= 0); //greater than zero error checking

    LCD_string_write("input value:\n");
    decimalToHexPrint(input,4); //print the value that was input to validate
    delay(200);
    return input;
}
/*============================================================================================
============================================== */
long getAddress(){// created for simplicity

    __xdata unsigned int input;

    input = userWord();//simply use userWord function but also print input.

    LCD_string_write("Value:\n");
    decimalToHexPrint(input,4);
    delay(200);

    return input;
}
/*============================================================================================
============================================== */
long getData(){
       __xdata unsigned int input;
```

```
    input = userByte();//simply use userByte function but also print input.

    LCD_string_write("Value:\n");
    decimalToHexPrint(input,2);
    delay(200);

    return input;
}
/*============================================================================================
============================================== */
void executeDump(volatile __xdata unsigned int address, volatile __xdata unsigned int
size, volatile __xdata unsigned char type, volatile __xdata unsigned char printpg){

       volatile __xdata unsigned int *addressPointer;
       volatile __xdata unsigned int counter = 0;
       volatile __xdata unsigned int pageNumber = 1;
       volatile __xdata unsigned int pageCounter = 0;
       volatile __xdata unsigned int ramValue;
       volatile __xdata char pageSelect;
       __code  unsigned char* pageText = "<--1  E-Exit  0-->\n"; //paging interface
       volatile __xdata unsigned int pageSelected;
       volatile __xdata unsigned int executeComplete;
       volatile __xdata unsigned int loop;
       volatile __xdata unsigned int typecount;

       do{ //loop for total pages
              pageSelected = 0;
              executeComplete = 0;
              pageCounter = 0;
              typecount = 0;

              LCD_CLEAR();
              LCD_string_write("Dump Page:");
              decimalToHexPrint(pageNumber, 4); //page number

              if(type == 1){
                     LCD_string_write("\nDumping ");
                     decimalToHexPrint(size, 4);
                     LCD_string_write(" Bytes\n"); //how much stuff to be printed
              }
              else if (type == 2){
                     LCD_string_write("\nDumping ");
                     decimalToHexPrint(size, 4);
                     LCD_string_write(" Words\n");
              }
              else if (type == 4){
                     LCD_string_write("\n");
                     decimalToHexPrint(size, 4);
                     LCD_string_write(" Double Words\n");
              }

              while((pageCounter < printpg) && (counter < size)) { //loop for values in
a page

                     if(address + type >  END){
                            break; //avoid overflow
                     }


                     decimalToHexPrint(address, 4);
```

```
                    LCD_string_write(":");

                    for(loop = 0; loop < type; loop++) //loop for data per line
                    {
                           addressPointer = (unsigned int __xdata*) address;
                           ramValue = *addressPointer;

                           decimalToHexPrint(ramValue, 2);
                           pageCounter+=1;

                           if(address ==  END){
                           break; //avoid overflow
                           }
                           else{
                                  address +=1;
                           }
                    }
                    LCD_string_write("\n");
                    delay(100);

                    counter +=1;
                    typecount += 1;
                    if(address ==  END || counter == size ){

                           if (address == END)
                           {
                                  addressPointer = (unsigned int __xdata*) (0xFFFF);
//for printing 0xFFFF
                                  ramValue = *addressPointer;
                                  decimalToHexPrint((unsigned int) addressPointer, 4);
                                  LCD_string_write(":");
                                  decimalToHexPrint(ramValue, 2);
                                  LCD_string_write("\n");

                           }

                           break;
                    }
             }

             LCD_string_write(pageText); //print paging interface

             do {

                    pageSelect = keyDetect();

                    switch (pageSelect) {
                           case '0':  //going forward
                           if(counter < size && address + type <=  END){

                                  pageNumber += 1;
                                  pageSelected = 1;
                           }
                           break;
                           case '1'://going back
                           if(pageNumber > 1){

                                  address = address - (printpg + pageCounter);
                                  counter = counter - (10 + typecount);
                                  pageNumber -= 1;
                                  pageSelected = 1;
```

```
                        }
                        break;
                        case 'E': //exit
                               delay(100);
                               pageSelected = 1;
                               executeComplete = 1;
                               break;

                  }
            } while (pageSelected == 0);
      }while (executeComplete == 0);


}
/*============================================================================================
============================================== */
void setupDump(){
      iowrite8(seg7_address,0x7A);     //value is D
      delay(200);

      LCD_CLEAR();

    LCD_string_write(" DUMP");
    delay(100);

    __xdata unsigned char blockDataType = charToDecimal(getBlockDataType()); //will
have decimal
    __xdata unsigned int blockNumber = getBlockNumber();
      LCD_CLEAR();
      LCD_string_write(" Input starting address\n");
    __xdata unsigned int startingAddress = getAddress();

     if(blockDataType == 1){ //simply for select the amount of values per page
               executeDump(startingAddress, blockNumber, 1, 10);
        }
        else if (blockDataType == 2){
               executeDump(startingAddress, blockNumber, 2, 20);
        }
        else  if (blockDataType == 3){
               executeDump(startingAddress, blockNumber, 4, 40);
        }

}
/*============================================================================================
============================================== */
void moveExecute(volatile __xdata unsigned int address, volatile __xdata unsigned int
size, volatile __xdata unsigned char type, volatile __xdata unsigned int destination){
      LCD_CLEAR();
      LCD_string_write(" Please wait...\n");


      volatile __xdata unsigned int *currentAddress = 0;
      volatile __xdata unsigned char ramValue = 0;


      switch (type){ //calculate proper size based on datatype
               case 2:
                        size = size * 2;
               break;
               case 3:
                        size = size * 4;
```

```
                break;
        }

        while(address <=  END && destination <=  END && size != 0) //loop for size
        {

                //IOM = 0;
                currentAddress = (unsigned int __xdata*) (address); //Points to source
address
                ramValue = *currentAddress;  //Reading the RAM address

                currentAddress = (unsigned int __xdata*) (destination); //Points to
destination address
                *currentAddress = ramValue;        //Writing the RAM Address

                delay(100);

                address += 1;
                destination += 1;
                size -= 1;

        }

        LCD_CLEAR();
        LCD_string_write("Move Complete");
        delay(200);

}
/*=============================================================================
========================================== */
void setupMove(){

        iowrite8(seg7_address,0x3E);      //value is B
        delay(200);

        LCD_CLEAR();

    LCD_string_write(" MOVE");
    delay(100);
       LCD_CLEAR();

       volatile __xdata unsigned char blockDataType =
charToDecimal(getBlockDataType()); //will have decimal value
    volatile __xdata unsigned int blockNumber = getBlockNumber();

       LCD_CLEAR();
       LCD_string_write(" Input starting address\n");
    volatile __xdata unsigned int startingAddress = getAddress(); //source address
       LCD_CLEAR();

       LCD_string_write(" Input destiation address\n");
       volatile __xdata unsigned int destinationAddress = getAddress(); //destination
address
       LCD_CLEAR();

       moveExecute(startingAddress, blockNumber, blockDataType, destinationAddress);

}
/*=============================================================================
========================================== */
void testRAM(uint8_t d){
```

```
        __xdata unsigned int i;
        __xdata unsigned int weNeedToFindThis;
        __xdata unsigned int *ram_address;

      iowrite8(seg7_address,0xEE);      //value is A
      delay(100);

      for (i = __START_RAM__; i < __END_RAM__; i++) {
            ram_address = (unsigned int __xdata*)(i);
            *ram_address = d;
            weNeedToFindThis = *ram_address;
            if (weNeedToFindThis != d ){
                  LCD_CLEAR();
                  LCD_string_write("Test Failed\n");
                  LCD_string_write("Stage one. Address:\n");
           decimalToHexPrint(i,4);
                  LCD_string_write("value recieved:\n");
                  decimalToHexPrint(weNeedToFindThis,2);
                  delay(200);
                  return;
            }

      }

      d = ~(d); //complement

      for (i = __START_RAM__; i < __END_RAM__; i++) {
            ram_address = (unsigned int __xdata*)(i);
            *ram_address = d;

            weNeedToFindThis = *ram_address;

            if (weNeedToFindThis != d ){
                  LCD_CLEAR();
                  LCD_string_write("Test Failed\n");
                  LCD_string_write("Stage Two. Address:\n");
           decimalToHexPrint(i,4);
                  LCD_string_write("value recieved:\n");
                  decimalToHexPrint(weNeedToFindThis,2);
                  delay(200);
                  return;
            }


      }


      LCD_CLEAR();
      LCD_string_write("Test Passed");
      delay(200);
      return;
}
/*=============================================================================================
============================================== */
void executeCount(__xdata unsigned int address, __xdata unsigned int size, __xdata
unsigned char type, __xdata unsigned char value){

      volatile __xdata unsigned int *currentAddress = 0;
      volatile __xdata unsigned char ramValue;
      volatile __xdata unsigned char search = 0;
      volatile __xdata unsigned int pageSelected;
```

```
	volatile __xdata unsigned int executeComplete;
	volatile __xdata unsigned int count = 0;
	volatile __xdata unsigned char dir = 1;
	volatile __xdata char ctrl;
	volatile __xdata unsigned int sizeCount = size;
	volatile __xdata unsigned int addressHold = address;

	switch (type){
		case 2:
			size = size * 2;
		break;
		case 3:
			size = size * 4;
		break;
	}


	LCD_string_write(" Searching for:");
	decimalToHexPrint(value, 2);

	delay(200);

	do{

		currentAddress = (unsigned int __xdata*) addressHold; //Points to current
address
		ramValue = *currentAddress;	//Reading the RAM address

		if(ramValue == value){
			count += 1;
		}

		addressHold += 1;
		sizeCount -= 1;

	}while((addressHold <=  END) && (sizeCount > 0));


	do{
		pageSelected = 0;
		executeComplete = 0;
		do{


			currentAddress = (unsigned int __xdata*) address; //Points to
current address
			ramValue = *currentAddress;	//Reading the RAM address


			if(ramValue == value){
				LCD_CLEAR();
				LCD_string_write("Searching ");
				decimalToHexPrint(ramValue, 2);
				write('h');

				LCD_string_write("\nValue Found at \nAddress: ");
				decimalToHexPrint(address, 4);

				delay(100);
```

```
                LCD_string_write("\nTotal Found: ");
                decimalToHexPrint(count, 4);

                delay(100);

                break;
        }

        if(dir == 1){
                address += 1;
                size -= 1;
        }
        else if (dir == 0){
                address -= 1;
                size += 1;
        }

}while((address <=  END) && (size > 0) );

if(count == 0){
        LCD_string_write("\nValue NOT Found!\n");
        break;
}

LCD_string_write("\n<--1  E-Exit  0-->\n");

do {
        ctrl = keyDetect();

        switch (ctrl) {
                case '0':  //NExt Page
                if(size > 0 && address <  END){
                        LCD_string_write("0 - Next Page\n");
                        delay(50);

                        dir = 1;            //Direction is onwards;
                        address += 1;       //Next address
                        size -= 1;          //Update the Size counter
                        pageSelected = 1;
                }
                break;
                case '1':    //Previous Page
                if(count > 1){
                        if(size == 0 || address ==  END){
                                count += 1;
                        }
                        LCD_string_write("1 - Previous Page");
                        delay(50);

                        dir = 0;            //Direction is backwards
                        address -= 1;       //Previous address
                        size += 1;          //Update the Size Counter
                        pageSelected = 1;
                }
                break;
                case 'E':
                pageSelected = 1;
                executeComplete = 1;
                break;
                //default:
```

```
                    }

             } while (pageSelected == 0);
       }while (executeComplete == 0);


       LCD_string_write("\nCount Complete");

       delay(150);
}
/*============================================================================================
============================================== */
void setupCount(){
       iowrite8(seg7_address,0x9C);      //value is C
       delay(200);

       LCD_CLEAR();

    LCD_string_write(" Count");
    delay(100);
       LCD_CLEAR();

       __xdata unsigned char blockDataType = charToDecimal(getBlockDataType()); //will
have decimal value
    __xdata unsigned int blockNumber = getBlockNumber();

       LCD_CLEAR();
       LCD_string_write(" Input starting address\n");
    __xdata unsigned int startingAddress = getAddress(); //starting address
       LCD_CLEAR();

       LCD_string_write(" Input value to count\n");
       __xdata unsigned int lookupValue = getData();// value to count
       LCD_CLEAR();

       executeCount(startingAddress, blockNumber, blockDataType, lookupValue);
}
/*============================================================================================
============================================== */
void executeFind(volatile __xdata unsigned int address, volatile __xdata unsigned int
size, volatile __xdata unsigned char type, volatile __xdata unsigned char value){

       volatile __xdata unsigned int *currentAddress = 0;
       volatile __xdata unsigned char ramValue = 0;
       volatile __xdata unsigned char search = 0;
       volatile __xdata unsigned int pageSelected;
       volatile __xdata unsigned int executeComplete;
       volatile __xdata unsigned int count = 0;
       volatile __xdata unsigned char dir = 1;        //1 =Forward, 0 = backwards
       volatile __xdata char ctrl;


       switch (type){
              case 2:
                     size = size * 2;
              break;
              case 3:
                     size = size * 4;
              break;
       }
```

```
	LCD_string_write(" Searching for:");
	decimalToHexPrint(value, 2);

	delay(100);


	do{
		pageSelected = 0;
		executeComplete = 0;
		do{

			currentAddress = (unsigned int __xdata*) address; //Points to
current address
			ramValue = *currentAddress;	//Reading the RAM address
			ramValue = *currentAddress;

			if(ramValue == value){
				LCD_CLEAR();
				LCD_string_write("Searching ");
				decimalToHexPrint(ramValue, 2);
				write('h');

				LCD_string_write("\nValue Found at \nAddress: ");
				decimalToHexPrint(address, 4);

				delay(50);



				if(dir == 1){
					count += 1;
				}
				else if (dir == 0){
					count -= 1;
				}




				break;
			}
			if(dir == 1){
				address += 1;
				size -= 1;
			}
			else if (dir == 0){
				address -= 1;
				size += 1;
			}

		}while((address <=  END) && (size > 0) );

		if(count == 0){
			LCD_string_write("\nValue NOT Found!\n");
			break;
		}

		LCD_string_write("\n<--1  E-Exit  0-->\n");

		do {
```

```
                        ctrl = keyDetect();

                        switch (ctrl) {
                               case '0':  //Next Page
                               if(size > 0 && address <  END){
                                      LCD_string_write("0 - Next Page\n");
                                      delay(50);

                                      dir = 1;            //Direction is onwards;
                                      address += 1;       //Next address
                                      size -= 1;          //Update the Size counter
                                      pageSelected = 1;
                               }
                               break;
                               case '1':    //Previous Page
                               if(count > 1){
                                      if(size == 0 || address ==  END){
                                             count += 1;
                                      }
                                      LCD_string_write("1 - Previous Page");
                                      delay(50);

                                      dir = 0;            //going backwards
                                      address -= 1;       //Previous address
                                      size += 1;          //Update the Size Counter
                                      pageSelected = 1;
                               }
                               break;
                               case 'E':
                               pageSelected = 1;
                               executeComplete = 1;
                               break;
                        }

                } while (pageSelected == 0);
        }while (executeComplete == 0);


        LCD_string_write("\nCount Complete");

        delay(150);
}
/*=========================================================================================
========================================== */
void setupFind(){

        iowrite8(seg7_address,0x8E);      //value is F
        delay(200);

        LCD_CLEAR();

    LCD_string_write(" FIND");
    delay(100);
       LCD_CLEAR();

        __xdata unsigned char blockDataType = charToDecimal(getBlockDataType()); //will
have decimal value
    __xdata unsigned int blockNumber = getBlockNumber();

       LCD_CLEAR();
       LCD_string_write(" Input starting address\n");
```

```
    __xdata unsigned int startingAddress = getAddress(); //get starting address
      LCD_CLEAR();

      LCD_string_write(" Input value to count\n");
      __xdata unsigned int lookupValue = getData(); //get value to look for
      LCD_CLEAR();

      executeFind(startingAddress, blockNumber, blockDataType, lookupValue);
}
/*=============================================================================
============================================== */
void executeEdit(__xdata unsigned int address){

      volatile __xdata unsigned int *currentAddress = 0;
      volatile __xdata unsigned char ramValue = 0;
      volatile __xdata char ctrl;
      volatile __xdata unsigned int pageSelected;
      volatile __xdata unsigned int executeComplete;

      do{
            pageSelected = 0;
            executeComplete = 0;

            LCD_CLEAR();

            LCD_string_write("\nAddress: ");
            decimalToHexPrint(address, 4); //print address we are on

            LCD_string_write("\nCurrent Value: ");
            currentAddress = (unsigned int __xdata*) address; //Points to current
address
            ramValue = *currentAddress;                  //Reading the RAM address
            decimalToHexPrint(ramValue, 2);

            LCD_string_write("\nEnter New Value\n");
            ramValue = userByte();
            *currentAddress = ramValue;            //Editing the RAM Address

            LCD_string_write("\nAddress: ");
            decimalToHexPrint(address, 4);

            LCD_string_write("\nNew Value: ");
            currentAddress = (unsigned int __xdata*) address; //Points to current
address
            ramValue = *currentAddress;                  //Reading the RAM address
            decimalToHexPrint(ramValue, 2);



            LCD_string_write("\n1-Exit  0-->\n");
            do {

                  ctrl = keyDetect();

                  switch (ctrl) {
                        case '0':  //Next Page
                        if(address < END){   //avoid Overflow
                              address += 1;       //next Address
                              LCD_string_write("0 - Continue");
                              delay(50);
                              pageSelected = 1;
```

```
                                executeComplete = 0;
                            }
                            break;
                            case '1': //exit
                            LCD_string_write("1 - Exiting");
                            delay(50);
                            LCD_string_write("\nEdit Complete");
                            delay(100);

                            pageSelected = 1;
                            executeComplete = 1;
                            break;

                    }

            } while (pageSelected == 0);

      }while(executeComplete == 0);
}
/*============================================================================================
============================================== */
void setupEdit(){

      iowrite8(seg7_address,0x9E);     //value is E
      delay(200);

      LCD_CLEAR();
      LCD_string_write("EDIT");
    delay(100);
      LCD_CLEAR();


      LCD_CLEAR();
      LCD_string_write("Input address\n");
    __xdata unsigned int address = getAddress(); //get address to start editing at
      LCD_CLEAR();

      executeEdit(address);
}
/*============================================================================================
============================================== */
void showMenu() { //menu text
      __code unsigned char* menu = " A-Ram Check\n B-Move\n C-Count\n D-Dump\n E-
Edit\n F-Find\n 1-Temp & Phot\n 2-Free Type";
      LCD_string_write(menu);
}
/*============================================================================================
============================================== */
void mainMenu() {

      LCD_CLEAR();

      while(1) {
            LCD_CLEAR();
            showMenu();
            __xdata unsigned char menuSelection = keyDetect();
            LCD_string_write("\n");
            delay(50);
            write(menuSelection);
            delay(100);
```

```
		//Find what the key press was
		switch (menuSelection) {
			case 'A':
				LCD_CLEAR();
				__xdata uint8_t valueToTestRAM;

				LCD_string_write(" Enter Byte:\n");
				valueToTestRAM = userByte();

				LCD_CLEAR();
				LCD_string_write(" Please Wait...");
				testRAM(valueToTestRAM);

				break;

			case 'B' :
				setupMove();
				break;

			case 'C' :
				setupCount();
				break;

			case 'D' :
				setupDump();
				break;

			case 'E':
				setupEdit();
				break;

			case 'F':
				setupFind();
				break;

			case '1' :
				LCD_CLEAR();
				__xdata unsigned char valueFromPhot =
ioread8(__PHOT_ADDRESS__);
				valueFromPhot = ioread8(__PHOT_ADDRESS__);
				LCD_string_write(" LIGHT VALUE:\n");
				decimalToHexPrint(valueFromPhot,2);

				__xdata unsigned char valueFromTemp = ioread8((uint8_t
__xdata*)(__TEMP_ADDRESS__));
				valueFromTemp = ioread8((uint8_t
__xdata*)(__TEMP_ADDRESS__));
				LCD_string_write("\n\n TEMP VALUE:\n");
				decimalToHexPrint(valueFromTemp,2);
				delay(300);
				break;

			case '2':
			 LCD_string_write(" Free Type: \n");
				while(1) {
					freeType();
				}
				break;

			default:
				LCD_CLEAR();
```

```
                        LCD_string_write(" No choice made");
                        delay(100);
                }
        }
}
/*=============================================================================================
============================================== */
void main(void) {
        CD = 0;
        IOM = 0;



        IOM = 0;
        CD = 1;

        TFT_LCD_INIT();

        writeSomeLines();

        setRotation(1);
        LCD_CLEAR();
        fillScreen(BLACK);
        setTextColor(GREEN, BLACK);
        setCursor(0,0);

        UART_Init();

        mainMenu();
}
```

8.6 Implementation Code for GAL/PAL (Decoder)

```
Name     Decoder ;
PartNo   p22v10 ;
Date     3/19/2022 ;
Revision 01 ;
Designer Engineer ;
Company  University of Nebraska-Lincoln ;
Assembly None ;
Location  ;
Device   p22v10 ;

/* *************** INPUT PINS *********************/
PIN 2 = PSEN                          ; /*                                 */
PIN 3 = A15                           ; /*                                 */
PIN 5 = A14                           ; /*                                 */
PIN 6 = IOM                           ; /*                                 */
PIN 10 = WR                           ; /*                                 */

/* *************** OUTPUT PINS *********************/
PIN 14 = CS_ROM1                        ; /*                                    */
PIN 15 = CS_ROM2                        ; /*                                    */
PIN 16 = CS_RAM1                        ; /*                                    */
PIN 17 = CS_RAM2                        ; /*                                    */
PIN 20 = CS_SEG                         ; /*                                    */
PIN 19 = CS_LCD                         ; /*                                    */
PIN 21 = CS_PHOT                        ;
PIN 22 = CS_TEMP                        ;
```

```
/*********** Equations ***********/
CS_RAM1  = !(PSEN & !A15 & !IOM)              ;/*                    */
CS_RAM2  = !(PSEN & A15 & !IOM)               ;/*                    */
CS_ROM1  = !(!PSEN & !A15)                    ;/*                    */
CS_ROM2  = !(!PSEN & A15)                     ;/*                    */
CS_SEG   = PSEN & A15 & IOM & !A14            ;/*                    */
CS_LCD   = !(PSEN & !A15 & IOM & A14)         ;/*                    */
CS_PHOT  = !(PSEN & !A15 & IOM & !A14)        ;
CS_TEMP  = !(PSEN & A15 & IOM & A14)          ;
```