Angelica Semenec

CmpE 127 Lab

Lab 5
Programmable Peripherals: Parallel LCD

Abstract:

The purpose of this lab is to incorporate a 16 x 2 LCD into the existing architecture. The LCD will be run in 4-bit mode and so the correct mode on the LCD must be initially set so that the program can write characters to the LCD. The LCD will have the capability of displaying any character typed through the console via a user-interface.

Functionality

A 16 x 2 LCD is integrated into the existing architecture. The purpose of the display is to show whatever input the user writes into the user-interface onto the two available lines for characters.

The LCD is run in 4-bit mode due to the limitation of control signals from the SJTwo board. The lower 4 data bits of the LCD (D_0 - D_3) are tied to ground and are not used in 4-bit mode. The upper 4 data bits are connected to the address/data bus indirectly. The address/data bus bits AD_4 - AD_7 are connected to a ‘245 (octal bus transceiver). When the system is writing to the LCD, the CPU sends signals through the ‘245 into the LCD.

The backlight and display are attached to supply voltage and ground according to their schematics and the contrast (V_0) is attached to the output of a potentiometer so that the contrast can be adjusted accordingly.

The R/W’ signal is connected to Write’. When the program sets the write control signal to high, this will tell the LCD that it will be written to. RS is the control that tells the LCD whether it is receiving an instruction or data and is connected to AD_4. The enable signal is connected to the output of a ‘74. The ‘74 latches in the enable signal that acts like the LCD chip select. The output of the ‘74 is ANDed with the ALE control signal so that the signal can be pulsed without affecting the other components in the board. The CLR’ input is connected to the logical function ALE or Write so that it is cleared when ALE and Write are set to a low voltage. This will essentially deselect the LCD.

In the user-interface, the selection number for writing to LCD is 4 so that when the user enters 4, they will be able to interact with the LCD. Upon selecting the LCD, the program will run through the necessary setup function to change the LCD to 4-bit mode and enable the display to 2 lines. After the setup, the user will be prompted to enter whatever they wish to display to the LCD. Once they enter their data, the LCD will run through the necessary functions to display the characters and then return to the prompt for the user to enter in another word to the LCD. Each time the user enters characters, the LCD is cleared and the cursor is reset to the beginning so the user over-writes anything that was previously written.

Design

1602 16x2 LCD Instruction Set

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears display, resets cursor
Return Home	0	0	0	0	0	0	0	0	1	-	Resets cursor
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Assign cursor direction, enable display shift
Display On/Off	0	0	0	0	0	0	1	D	C	B	Set Display(D) Cursor(C) Blinking(B)
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift
Function Set	0	0	0	0	1	DL	N	F	-	-	DL: 8/4-bit N: 2-line or 1-line F: 5x10 or 5x8 dots
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address
Read busy flag/ address counter	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Read flag and counter
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM

‘245 (Octal Bus Transceiver With 3-State Outputs)

Inputs		**Operation**
OE’	**DIR**	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = High Voltage, L = Low Voltage, X = Immaterial

‘74 (Dual D-Type Positive Edge Triggered Flip-Flop)

Inputs				Outputs	
Pre’	Clr’	Clk	D	Q	Q’
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H^	H^
H	H	^	H	H	L
H	H	^	L	L	H
H	H	L	X	Q0	Q0’

H = High Voltage, L = Low Voltage, X = Immaterial, Q0 = Hold, ^ = Rising Clock Edge

‘4002 (4-Input Nor Gate)

Input				Output
A	B	C	D	Y
L	L	L	L	H
H	X	X	X	L
X	H	X	X	L
X	X	H	X	L

X	X	X	H	L

H = High Voltage, L = Low Voltage, X = Immaterial

‘244 (Octal Buffer and Line Drivers) Voltage Table

Inputs		**Output**
G’	**A**	**Y**
L	L	L
L	H	H
H	X	Z

H = High voltage, L = Low voltage, X = Immaterial, Z = High impedence

IDT6116SA (SRAM) Voltage Table

Mode	**CS’**	**OE’**	**WE’**	**I/O**
Standby	H	X	X	High-Z
Read	L	L	H	$Data_{out}$
Read	L	H	H	High-Z
Write	L	X	L	$Data_{in}$

H = High voltage, L = Low voltage, X = Immaterial

‘373 Voltage Table

D_n	**LE**	**OE’**	**O_n**
H	H	L	H
L	H	L	L
X	L	L	Q_0
X	X	H	Z*

H = High voltage, L = Low voltage, X = immaterial, Z = High impedence*

‘641 Voltage Table

Control Inputs		**Operation**
G’	**DIR**	**‘641**
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = High voltage, L = Low voltage, X = immaterial

‘08 (2-input AND gate) Voltage Table

Inputs		**Output**
A	**B**	**Y**
L	X	L
X	L	L
H	H	H

H = High voltage, L = Low voltage

‘04 (Hex Inverter) Voltage Table

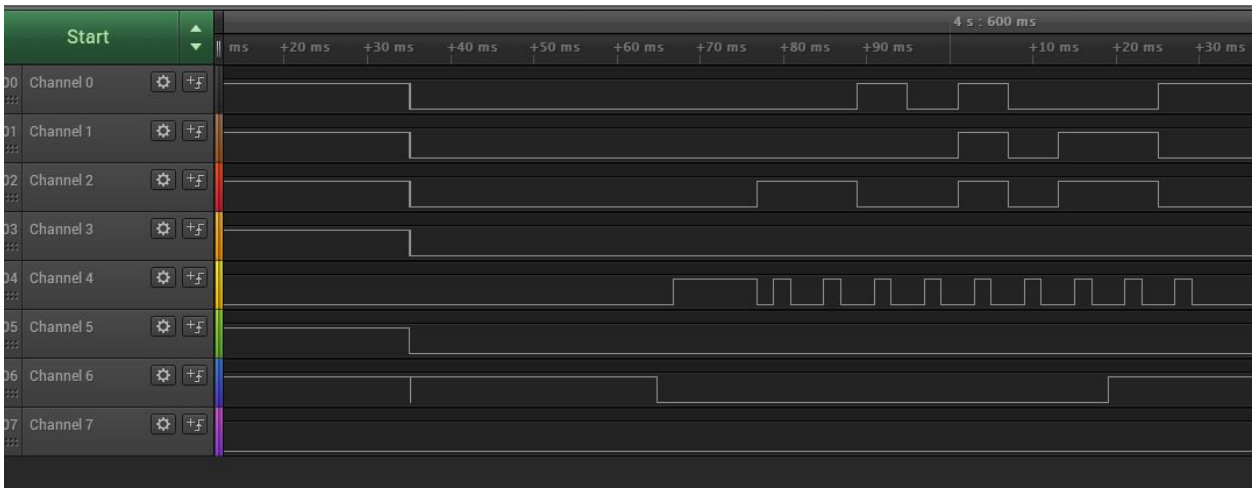
Input	**Output**
A	**Y**
L	H
H	L

H = High voltage, L = Low voltage

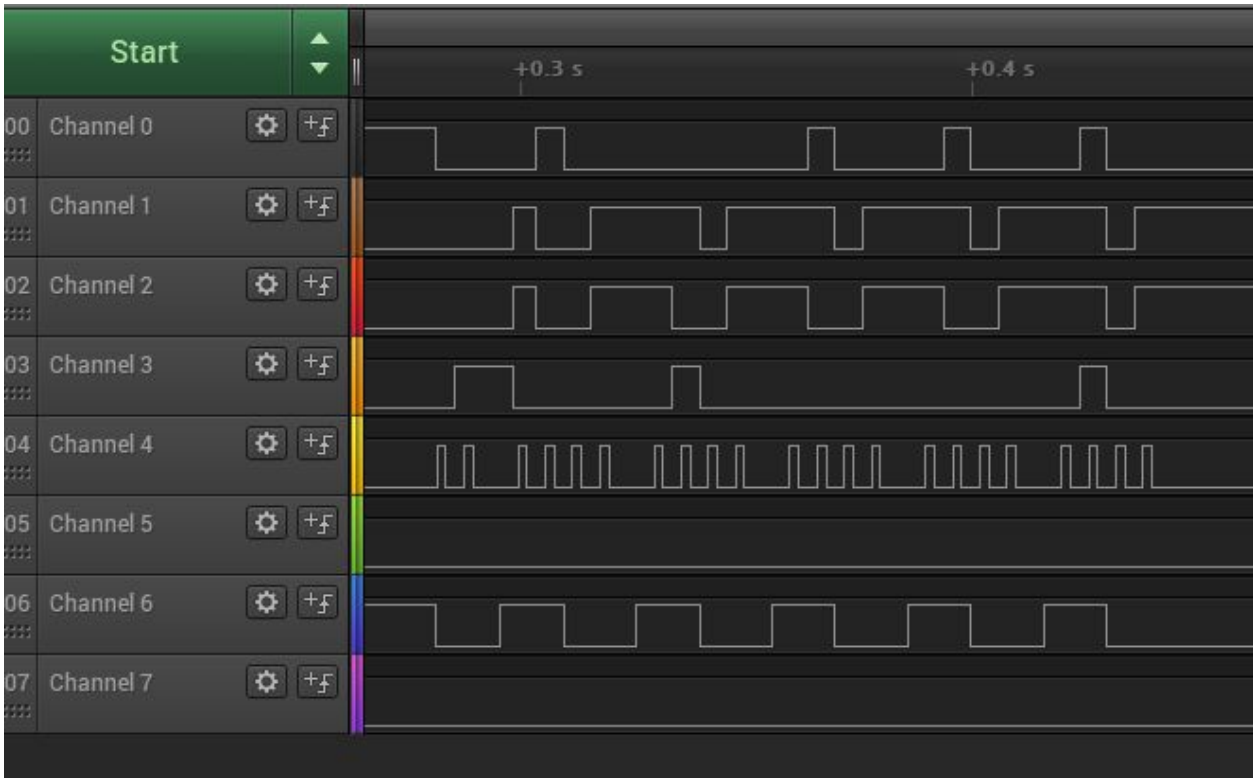
‘11 (3-Input AND gate) Voltage Table

Inputs			Output
A	**B**	**C**	**Y**
H	H	H	H
L	X	X	L
X	L	X	L
X	X	L	L

H = High voltage, L = Low voltage, X = Immaterial

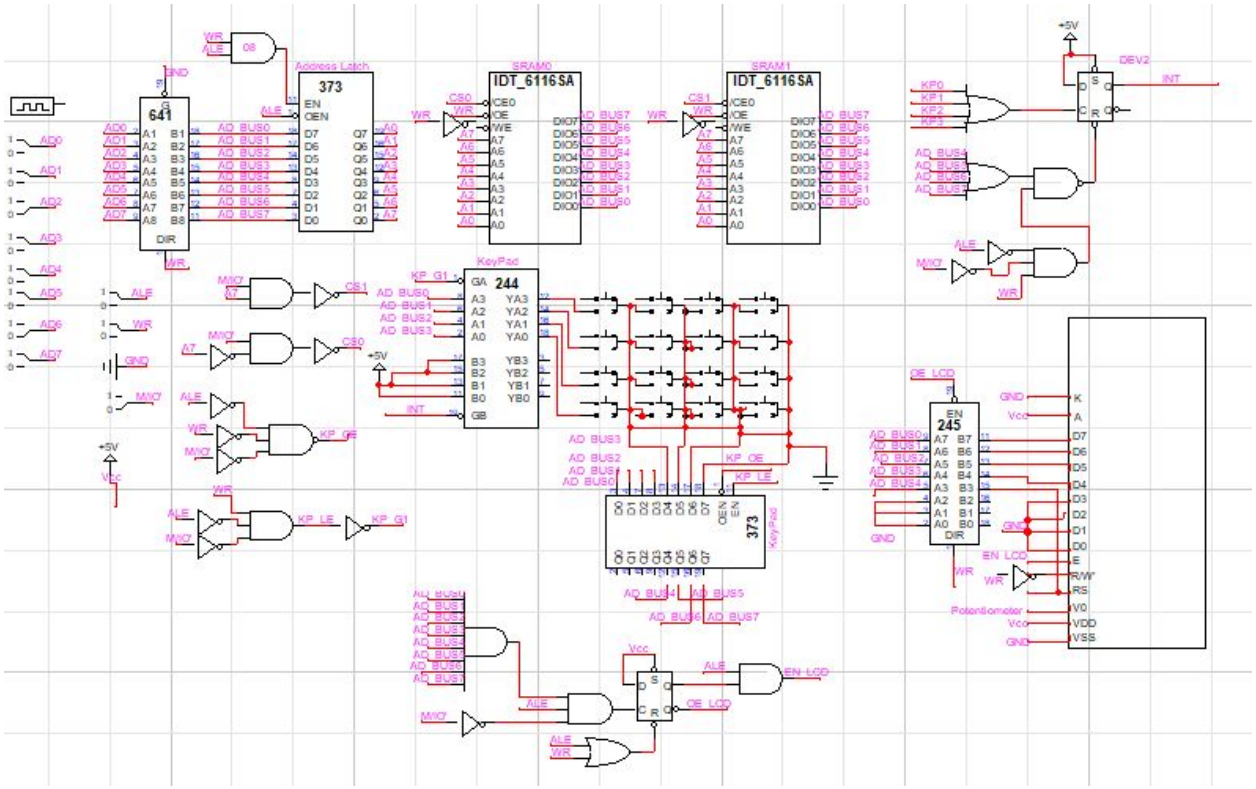


Waveform from 4-bit operation setup. Channel 0 through 3 is connected to LCD D_7 through D_4 respectively. Channel 4 is connected to the enable signal. Channel 5 is connected to R/W’ and channel 6 is connected to RS.



Output waveform displays the text “Hello” to the LCD. Each channel is the same as those listed in the above waveform description.

Schematics



Lab 5 Circuit Schematics

Parts List

1. 4 x 4 Keypad
2. 1602 LCD
3. IDT6116SA: CMOS Static RAM 16K (2K x 8-bit)
4. CD74LS4002: Dual 4-Input NOR Gate
5. SN74LS04: Hex Inverter
6. SN74LS08: Quadruple 2-Input Positive-AND Gate
7. SN74LS11: Triple 3-Input Positive-AND Gate
8. SN74LS74: Dual D-Type Positive Edge-Tripped Flip-Flops
9. SN74LS244: Octal Buffers and Line Drivers with 3-State Outputs
10. SN74LS245: Octal Bus Transceiver with 3-State Outputs
11. SN74HC373: Octal D-Type Transparent Latch
12. SN74LS641: Octal Bus Transceiver
13. 20-pin wire-wrapping DIP socket
14. 14-pin wire-wrapping DIP socket
15. 30 gauge wire

16. 40-pin ribbon cable
17. 1 Kohm resistors
18. SJTwo board
19. PCB

Algorithms

‘244 GA’:

GA’ = (WR * (M/IO’)’ * ALE’)’

‘373 LE:

(Latch Enable for the keypad)

LE = WR * (M/IO’)’ * ALE’

‘373 OE’:

(Output Enable for the keypad)

OE’ = (WR’ * OE’ * ALE’)’

‘74 CLK:

(Clock signal for Interrupt flag)

CLK = KP0 + KP1 + KP2 + KP3

KP is the output address of the keypad before the ‘373

‘74 CLR’:

(Clear signal for interrupt flag)

CLR’ = [(‘373)(Q0 + Q1 + Q2 + Q3)*(ALE’ * (M/IO’)’ * WR’)]’

INT:

(Interrupt signal)

INT = (‘74) Q

LCD EN:

EN = (‘74)Q * ALE’

LCD RS:

RS = AD_4

LCD R/W’

R/W’ = W’

‘245 DIR:

DIR = W

‘245 OE’:

OE’ = (‘74)Q’ * ALE’

Demo

The .hex file is generated through the WSL using the command “make build”. Once the .hex file is generated, it can be uploaded to the board via the hyperload tool. Hercules is then used to interact with the board via the interface.

A user-interface is generated and the user is prompted to input what operation they would like to perform. To operate the LCD, the operation corresponds to the number 4. When they enter the number 4, the control signals for the LCD to be set up to be 4-bits is generated. Once the LCD is setup and ready for input, the user is prompted to write to the LCD. The user can type anything they wish via the keyboard with the exception of the space key. Due to the restriction of the function scanf() within c++, the function stops at the white space character and enters in the word.

After the user enters in whatever information they wish to be displayed, the code runs through a program that sends each character to the LCD. Each character is generated and displayed one at a time to the display until it has finished. If the word is longer than 16 characters, it will continue onto the second line. The maximum amount of characters that can be displayed at once is 32. The LCD also supports special characters.

Source Code

main.cpp

```
//Libraries
#include <project_config.hpp>

#include <cstdint>
#include <iterator>

#include <math.h>

#include "L2_HAL/displays/led/onboard_led.hpp"
#include "utility/log.hpp"
#include "utility/time.hpp"

//Bus class
```

```
class Bus
{
   public:
   enum class ControlType
   {
       kMemory = 0,
       kIO
   };




//Bus Constructor
   Bus()
   {
       Write.SetAsOutput();
       ALE.SetAsOutput();
       M_IO.SetAsOutput();

       Status = 0;

       ad[0].GetPin().SetAsOpenDrain();
       ad[1].GetPin().SetAsOpenDrain();
       ad[2].GetPin().SetAsOpenDrain();
       ad[3].GetPin().SetAsOpenDrain();
       ad[4].GetPin().SetAsOpenDrain();
       ad[5].GetPin().SetAsOpenDrain();
       ad[6].GetPin().SetAsOpenDrain();
       ad[7].GetPin().SetAsOpenDrain();
   }
//Write from console to memory
   void WritetoMem(int address, int data)
   {
       sendAddress(address);
//Status bit is updated for ISR to keep track of where in the program it stopped
       Status = 0;
       writeData(data);
       Status = 0;

       return;
   }
//Read from console to memory
   int ReadfromMem(int address)
   {
       sendAddress(address);
       Status = 0;
       int data = readData();
       Status = 0;

       return data;
   }
```

```
//Function to use the LCD
void useLCD()
   {
       Status = 0;
       //Sets up signals to latch the enable signal
       M_IO.SetLow();
       Write.SetHigh();
       ALE.SetHigh();
       for (int i = 0; i < 8; i ++)
       {
           ad[i].SetAsOutput();
           ad[i].SetHigh();
       }
       //Sets D7 - D4 to low
       Set_Low();
       //Function to initialize LCD for 4-bit operation
       LCD_setup_4bit();

       //Function that prompts the user to enter characters
       LCD_prompt();

       //Function clears the enable flag
       ALE.SetLow();
       Write.SetHigh();
       return;
   }

   //Function prompts the user to enter characters
   void LCD_prompt()
   {
       char text[32];
       //char c;
       //int i = 0;
       while (true)
       {
           printf("\nWrite to LCD: ");
           scanf("%s", text);
          //Function clears display when user types in new word
           clear_display();
           for (int i = 0; i < strlen(text); i ++)
           {
               if(i == 16)
               {
                   //Function moves cursor to second row after 16 characters
                   move_cursor();
               }
               //Writes character to display
               write_char(text[i]);
           }
       }
       return;
   }
```

```
//Moves cursor to bottom line
void move_cursor()
{
    ad[4].SetLow();
    Set_Low();
    ad[0].SetHigh();
    ad[1].SetHigh();
    pulse_enable();
    Set_Low();
    pulse_enable();
    ad[4].SetHigh();
}

//Clears display and resets cursor
void clear_display()
{
    Set_Low();
    ad[4].SetLow();
    pulse_enable();
    ad[3].SetHigh();
    pulse_enable();
    return;
}

//Writes character to LCD
void write_char(char symbol)
{
    int ascii = int(symbol);
    printf("\nsym: %s, # %d ", &symbol, ascii);
    int binary[8];
    ad[4].SetHigh();
    for (int i = 0; i < 8; i ++)
    {
        if((ascii & int(pow(2,i))) > 0)
        {
            binary[i] = 1;
        }
        else
        {
            binary[i] = 0;
        }
    }
    for (int i = 7; i >= 0; i --)
    {
        printf("%d", binary[i]);
    }
    //Set upper 4 bits
    int count = 3;
    for(int i = 4; i < 8; i ++)
    {
        if(binary[i] == 0)
        {
            ad[count].SetLow();
```

```
            }
            else
            {
                ad[count].SetHigh();
            }
            count --;
        }
        pulse_enable();
        //Set lower 4 bits
        count = 3;
        for(int i = 0; i < 4; i ++)
        {
            if(binary[i] == 0)
            {
                ad[count].SetLow();
            }
            else
            {
                ad[count].SetHigh();
            }
            count --;
        }
        pulse_enable();
        inc_cursor();
    }

    //Increments cursor for next letter
    void inc_cursor()
    {
        Set_Low();
        ad[4].SetLow();
        pulse_enable();
        ad[1].SetHigh();
        ad[2].SetHigh();
        pulse_enable();
    }

    //LCD setup function to initialize for 4-bit operation
    void LCD_setup_4bit()
    {
        power_on();
        configure_4bit();
    }

    //Power on settings
    void power_on()
    {

        Delay(15);
        ad[4].SetLow();
        Delay(1);
        ALE.SetLow();
        Delay(5);
```

```
        ALE.SetHigh();
    }


        //Sets up 4-bit operation
        void configure_4bit()
    {
        ad[2].SetHigh();
        pulse_enable();
        pulse_enable();
        Set_Low();
        ad[0].SetHigh();
        //ad[2].SetHigh();
        pulse_enable();
        Set_Low();
        pulse_enable();
        ad[0].SetHigh();
        ad[1].SetHigh();
        ad[2].SetHigh();
        pulse_enable();
        Set_Low();
        pulse_enable();
        ad[1].SetHigh();
        ad[2].SetHigh();
        pulse_enable();
        ad[4].SetHigh();
        ad[1].SetHigh();
        pulse_enable();
        Set_Low();
        ad[0].SetHigh();
        pulse_enable();
    }

    //Resets D7 - D4 to low
    void Set_Low()
    {
        ad[0].SetLow();
        ad[1].SetLow();
        ad[2].SetLow();
        ad[3].SetLow();
    }

    //Pulses the enable signal
    void pulse_enable()
    {
        Delay(1);
        ALE.SetLow();
        Delay(1);
        ALE.SetHigh();
        Delay(1);
    }

//Called by ISR function
```

```
//Allows for use of Bus object
   void ISR_call(void)
   {
       //Array keeps track of bit state on address bus
       int bitArr[8];
       switch (Status)
       {
           //Case 0: Current bits do not need to be saved
           case 0:
               Break;



           //Case 1: Status is in SendAddress
           //Bit status saved in case function has begun calculating sent address
           case 1:
           //Case 2: Status is in WriteData
           //Bit status should be saved if function has begun calculating data bits
           //Case 1 and 2 require same functionality for bit save
           case 2:
               for (int i = 0; i < 8; i ++)
               {
                   if (ad[i].Read() == true)
                   {
                       bitArr[i] = 1;
                   }
                   else
                   {
                       bitArr[i] = 0;
                   }
               }
               break;
           //Case 3: Status is in ReadData
           //Current bit status does not need to be saved
           case 3:
               break;
       }

       //Write letter from keypad to console
       WritefromIO();

       switch (Status)
       {
           //Status does not need to be updated
           case 0:
               return;
           //Status is in SendAddress
           //Control signals reset
           case 1:
               ALE.SetHigh();
               M_IO.SetLow();
               Write.SetHigh();
               for (int i = 0; i < 8; i ++)
```

```
                {
                    ad[i].SetAsOutput();
                }
                for (int i = 0; i < 8; i++)
                {
                    if (bitArr[i] == 0)
                    {
                        ad[i].SetLow();
                    }
                    else
                    {
                        ad[i].SetHigh();
                    }

                }
                return;
            //Status is in WriteData
            case 2:
                ALE.SetLow();
                M_IO.SetHigh();
                Write.SetHigh();
                for (int i = 0; i < 8; i ++)
                {
                    ad[i].SetAsOutput();
                }
                for (int i = 0; i < 8; i++)
                {
                    if (bitArr[i] == 0)
                    {
                        ad[i].SetLow();
                    }
                    else
                    {
                        ad[i].SetHigh();
                    }

                }
                return;
            //Status is in ReadData
            case 3:
                Write.SetLow();
                ALE.SetLow();
                M_IO.SetHigh();
                for (int i = 0; i < 8; i ++)
                {
                    ad[i].SetAsInput();
                }
                return;
        }
    }
//Main function to write from IO to console
    void WritefromIO(void)
    {
```

```
        //Initializes required IO signals
        setupIO();

        //Address and state saved
        int address = 0;
        int state = 0;









        while (address == 0)
        {
            //Sets up lower 4 bits of IO address
            setIOaddress(state);
            //Checks for a key press
            address = checkKeyPress(state);
            //If key was pressed, prints corresponding character
            if (address != 0)
            {
                printChar(address);
                return;
            }
            Updates the state
            state = nextState(state);
        }
        return;
    }

//Initial IO signals set up
    void setupIO(void)
    {
        ALE.SetLow();
        M_IO.SetLow();
        Write.SetHigh();

        ad[0].SetAsOutput();
        ad[1].SetAsOutput();
        ad[2].SetAsOutput();
        ad[3].SetAsOutput();

        ad[4].SetAsInput();
        ad[5].SetAsInput();
        ad[6].SetAsInput();
        ad[7].SetAsInput();
    }

//Sets up lower 4 bits of IO address
    void setIOaddress(int state)
    {
        Write.SetHigh();
```

```
        switch (state)
        {
            case 0:
                ad[0].SetHigh();
                ad[1].SetLow();
                ad[2].SetLow();
                ad[3].SetLow();
                break;
            case 1:
                ad[0].SetLow();
                ad[1].SetHigh();
                ad[2].SetLow();
                ad[3].SetLow();
                break;
            case 2:
                ad[0].SetLow();
                ad[1].SetLow();
                ad[2].SetHigh();
                ad[3].SetLow();
                break;
            case 3:
                ad[0].SetLow();
                ad[1].SetLow();
                ad[2].SetLow();
                ad[3].SetHigh();
                break;
        }
    }

//Checks if button was pressed
    int checkKeyPress(int state)
    {
        Write.SetLow();
        for (int i = 4; i < 8; i ++)
        {
            if (ad[i].Read() == true)
            {
                return (int(pow(2,state) + int(pow(2, i) ) ) );
            }
        }
        return 0;
    }

//Prints character corresponding to address
    void printChar(int address)
    {
        char key;
        switch (address)
        {
            case 17: key = '1'; break;
            case 18: key = '4'; break;
            case 20: key = '7'; break;
```

```
            case 24: key = '*'; break;
            case 33: key = '2'; break;
            case 34: key = '5'; break;
            case 36: key = '8'; break;
            case 40: key = '0'; break;
            case 65: key = '3'; break;
            case 66: key = '6'; break;
            case 68: key = '9'; break;
            case 72: key = '#'; break;
            case 129: key = 'A'; break;
            case 130: key = 'B'; break;
            case 132: key = 'C'; break;
            case 136: key = 'D';
        }
        printf("%c\n", key);
    }

//Updates state of lower 4 bits of IO address
    int nextState(int state)
    {
        state ++;
        if (state == 4)
            state = 0;
        return state;
    }

//Sends address from console to memory
    void sendAddress(int address)
    {
        Status = 1;

        ALE.SetHigh();
        M_IO.SetLow();
        Write.SetHigh();

        for (int i = 0; i < 8; i++)
        {
            ad[i].SetAsOutput();
            if((address & int(pow(2,i))) > 0)
            {
                ad[i].SetHigh();
            }
            else
            {
                ad[i].SetLow();
            }
        }
        return;
    }

//Writes data from console to memory
    void writeData(int data)
    {
```

```
        Status = 2;

        ALE.SetLow();
        M_IO.SetHigh();
        Write.SetHigh();

        for (int i = 0; i < 8; i++)
        {
            ad[i].SetAsOutput();
            if((data & int(pow(2,i))) > 0)
            {
                ad[i].SetHigh();
            }
            else
            {
                ad[i].SetLow();
            }
        }
        return;
    }

//Reads data from memory to console
    int readData()
    {
        Status = 3;

        Write.SetLow();
        ALE.SetLow();
        M_IO.SetHigh();

        int data = 0;

        for (int i = 0; i < 8; i ++)
        {
            ad[i].SetAsInput();
            if(ad[i].Read() == true)
            {
                data = data + int(pow(2,i));
            }
        }

        return data;
    }

    private:
    Gpio Write = Gpio(0, 17);
    Gpio ALE = Gpio(0, 22);
    Gpio M_IO = Gpio(0, 0);

//Status allows ISR to know where it was paused in the program
    int Status;

    Gpio ad[8] = {
```

```
     Gpio(2, 2),
     Gpio(2, 5),
     Gpio(2, 7),
     Gpio(2, 9),
     Gpio(0, 15),
     Gpio(0, 18),
     Gpio(0, 1),
     Gpio(0, 10)
   };
};



//Bus object made global to allow ISR access to it
Bus obj;

//Calls Bus class member function
void ISR()
{
   obj.ISR_call();
}

//Main function
int main(void)
{
//Sets up interrupt function
   Gpio Int = Gpio(0, 11);
   Int.GetPin().SetMode(Pin::Mode::kPullUp);
   Int.AttachInterrupt(&ISR, GpioInterface::Edge::kEdgeRising);
   Int.EnableInterrupts();
//Users interface to interact with the hardware
    int op = -1;
   while(op != 0)
   {
     printf("\n Please enter the number of the function you'd like to perform:");
         printf("\n 0. Quit");
         printf("\n 1. Read data from memory");
         printf("\n 2. Write data to memory");
         printf("\n 3. Write from keypad");
         printf("\n 4. Use LCD");
         printf("\n Operation: ");
     scanf("%d", &op);
     if (op != -1 && op != 0)
     {
       if(op == 1)
       {
         int address = 0;
         printf("\n Address: ");
         scanf("%d", &address);
         int data = obj.ReadfromMem(address);
         printf("\n The data stored at address %d is %d", address, data);
       }
       else if(op == 2)
```

```
      {
        int address = 0;
        int data = 0;
        printf("Address: ");
        scanf("%d", &address);
        printf("Data: ");
        scanf("%d", &data);
        obj.WritetoMem(address, data);
        printf("\n The number %d has been written to address %d", data, address);
      }
      else if (op == 3)
      {
          obj.WritefromIO();
      }
      else if (op == 4)
      {
          Int.DisableInterrupts();
          obj.userLCD();
          Int.EnableInterrupts();
      }
      else
      {
        printf("\n The operation entered is invalid.\n");
      }

    }
  }
  return 0;
}
```