Angelica Semenec

CmpE 127 Lab

Lab 3
Peripheral Interfacing: Polling Keypad

Abstract:

The purpose of this lab is to interface a 4 x 4 keypad with the existing architecture. When the IO device is active and ready to take input, the two SRAM will be deactivated and the keypad will transmit signals via the address/data bus to the CPU. The software will determine from the returned signals of the keypad if a button has been pushed and which button the data corresponds to and print the character to the console.

Functionality

This lab interfaces a 4 x 4 keypad with the existing architecture. The keypad allows the user to enter in data via the keypad rather than through a console interface. The keypad allows the user to enter in numbers 0 through 9, A through D, ‘ # ‘ (octothorpe), and ‘ * ‘ (asterisk).

While the keypad is active, it will be controlling half of the signals on the address/data bus, AD_4 through AD_7, and those will be input into the SJTwo board. The SJTwo board will be controlling the other half of the address/data bus, AD_0 through AD_3. These signals will be input to the keypad in a continuous loop until a key press has been detected, starting with address LLLH, then LLHL, then LHLL, then HLLL and will loop back around to LLLH. This allows the circuit to check each of the rows of the keypad.

The 4-bit address that is send into the keypad is regulated by a ‘244 (octal buffer and line driver). When the user wants to write using the keypad, the control signal for write will be set to high, the control signal for M/IO’ will be set to low to denote that the IO device (keypad) is selected, and ALE will also be set to low. When M/IO’ is low, it deactivates the chip selects of both of the SRAMs so that only the IO device is controlling the bus.

The 4-bit address that is transmitted from the keypad is latched into a ‘373 (D-latch). If none of the keys are pressed, the pull-down resistors set the address to LLLL and the software determines that no key has been pressed. When a button is pushed, it completes the circuit with the output of the ‘244. When a high signal selects the proper row, the column of the keypad that registered high is recorded in the ‘373. As soon as any of the address bits input to the ‘373 are registered to be high, the software will check its table for the corresponding character that matches the selected row and column returned by the signals. It will then print that character onto the console and then return to its state of iterating through the addresses until it detects another key press.

Design

‘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

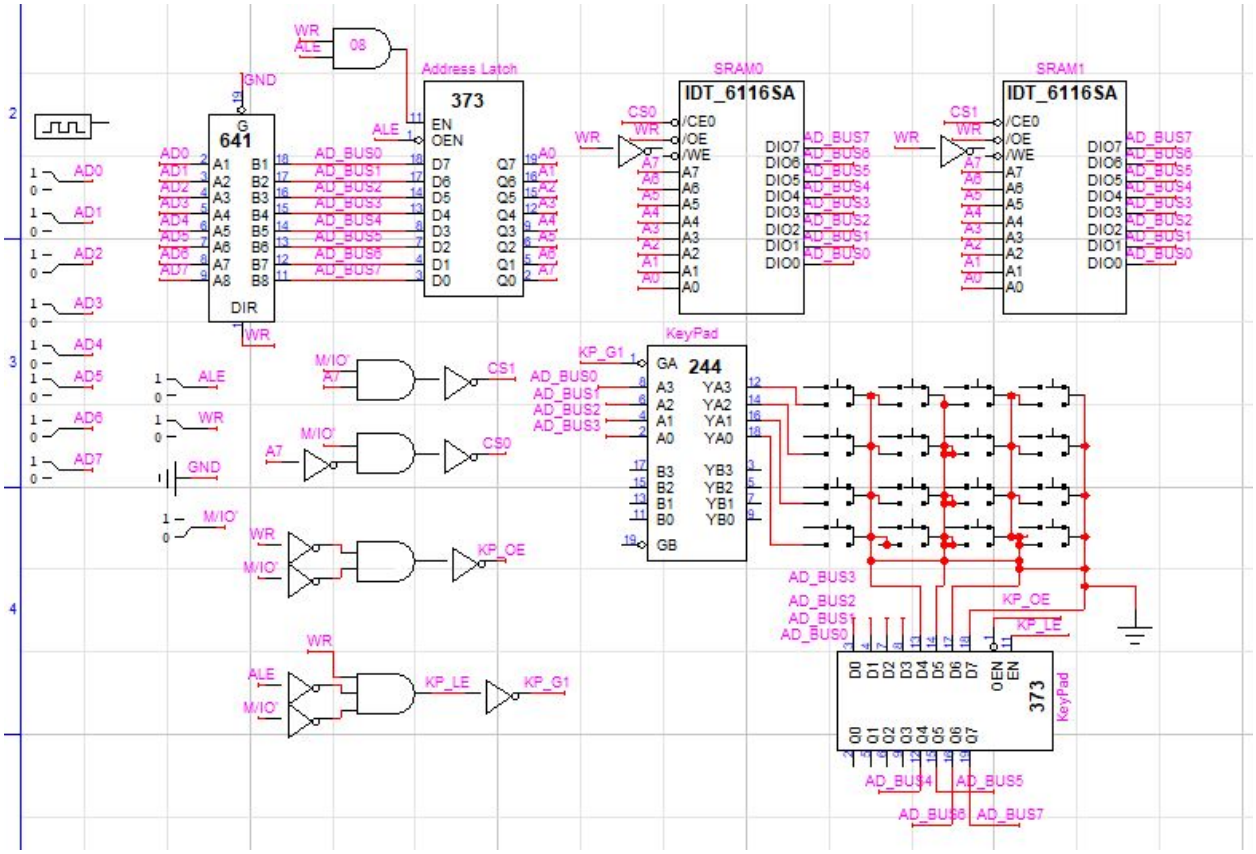
H = High voltage, L = Low voltage

‘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

Schematics



Lab 3 Circuit Schematics

Parts List

1. 4 x 4 Keypad
2. IDT6116SA: CMOS Static RAM 16K (2K x 8-bit)
3. SN74LS04: Hex Inverter
4. SN74LS08: Quadruple 2-Input Positive-AND Gate
5. SN74LS11: Triple 3-Input Positive-AND Gate
6. SN74LS244: Octal Buffers and Line Drivers with 3-State Outputs
7. SN74HC373: Octal D-Type Transparent Latch
8. SN74LS641: Octal Bus Transceiver
9. 20-pin wire-wrapping DIP socket
10. 14-pin wire-wrapping DIP socket
11. 30 gauge wire
12. 40-pin ribbon cable
13. 1 Kohm resistors
14. SJTwo board
15. 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’)’

Demo

The .cpp file must first be converted into a .hex file using the command “make build”. Once the .hex file is generated, it can be flashed onto the SJTwo board through the hyperload tool. When it successfully flashed, the program is run using the Hercules application.

The reset button on the board must be pressed to start the program. Once it reboots, an interface will appear in the console asking the user to input the operation they’d like. To quit the program, the user can enter ‘0’. To read from a memory address, the user inputs ‘1’. To write to a memory address, the user enters ‘2’. And to write from the keypad to console, the user enters ‘3’.

After the users enters ‘3’ to write to console from keypad, a new line will appear and the program will wait until the user has pressed a button. The user can press on any of the 16 buttons on the 4 x 4 keypad. Once the button is pressed, the corresponding character will be displayed on the console. If two keys are pressed at the same time, the console will show both buttons output in an alternating fashion. When the user presses the ‘ # ‘ character, the program will quit writing from the keypad and go back to displaying the user interface requesting input of an operation.

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

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

       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 data from console to memory
   void WritetoMem(int address, int data)
   {
       sendAddress(address);
       writeData(data);

       return;
   }

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

       return data;
   }

//Write data from IO keypad to console
   void WritefromIO(void)
   {
       //Sets up initial signals
       setupIO();

       int address = 0;
       int state = 0;
       //Address 72 corresponds to ‘ # ‘
       //If user enters ‘ # ‘, the program will return control to main
       while (address != 72)
       {
           //Sets up the lower half of the address bits
           setIOaddress(state);
           //Checks for a key press and generates the address
           address = checkKeyPress(state);
           if (address != 0)
           {
```

```
//Prints the character corresponding to the calculated address
                printChar(address);
            }
            //Updates the state of the lower bits
            state = nextState(state);
        }
        //Returns control to main
        return;
    }

//Sets up control signals and input/output
    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 the lower four bits of the 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 each of the upper 4 address bits
    int checkKeyPress(int state)
    {
        Write.SetLow();
        for (int i = 4; i < 8; i ++)
        {
            //If pressed, value will be high and true
            if (ad[i].Read() == true)
            {
                //Returns the address of the button pressed
                return (int(pow(2,state) + int(pow(2, i) ) ) );
            }
        }
        //If no buttons pressed, returns 0
        return 0;
    }

//Prints the character corresponding to the 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("\n%c", key);
    }
```

```
//Calculates the next state to set up the lower 4 bits of the IO address
   int nextState(int state)
   {
       state ++;
       if (state == 4)
           state = 0;
       return state;
   }

//Sends address from console to the address latch
   void sendAddress(int address)
   {
       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)
   {
       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 console to memory
   int readData()
   {
       ALE.SetLow();
       Write.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;
   }

//Sets up the pin positions with the corresponding signals
   private:
   Gpio Write = Gpio(0, 17);
   Gpio ALE = Gpio(0, 22);
   Gpio M_IO = Gpio(0, 0);
   Gpio Int = Gpio(0, 11); //Not used till Lab 4

   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)
   };
};

//Main function
int main(void)
{
 Bus obj;
 int op = -1;
//Creates user interface
   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 Operation: ");
    scanf("%d", &op);
    if (op != -1 && op != 0)
    {
      //Reads from memory with user input address through console
      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);
      }
      //Writes to memory with user input address and data through console
      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);
      }
      //Passes control to IO keypad
      else if (op == 3)
      {
          obj.WritefromIO();
      }
      //Prints if operation is invalid
      else
      {
        printf("\n The operation entered is invalid.\n");
      }

    }
  }
  return 0;
}
```