Angelica Semenec

CmpE 127 Lab

Lab 4
Peripheral Interfacing: Interrupt Driven Keypad

Abstract:

The purpose of this lab is to generate an interrupt signal using the 4 x 4 keypad. The interrupt signal is generated by pressing a key on the keypad. When a key is pressed, the program will enter an interrupt service routine (ISR) and print the character corresponding to the key that was pressed to the console before returning to its previous location in the program.

Functionality

The 4 x 4 keypad is used to generate an interrupt signal. When the board registers an interrupt signal, it enters into an interrupt service routine (ISR). The ISR checks the state of the keypad by iterating through addresses sent to the lower 4 bits of the address/data bus starting with 0001, to 0010, to 0100, to 1000 and then repeats until it detects a button press. Once the button is detected, it will go to a table of values that contain the corresponding characters and print the correct character to the console.

One half of the input to the ‘244 is connected to the power supply of +5 V. The control signal, G’, is connected to the interrupt signal. When the interrupt signal is 0, this half of the ‘244 is active. If a button is pressed, the ‘74 will clock a high voltage into its contents and the interrupt signal will change for low to high. When the interrupt signal changes to high, the ‘244 that is connected to the power supply switches off and the SJTwo board registers an interrupt on the rising edge of the signal.

Within the software, the program switches into the ISR function. If necessary, it saves the status of the address/data bus and then enters the keypad function. Once the correct character has been printed to the console, the ISR function resets the control signals of the function it was previously in if necessary as well as the status of the address/data bus and then resumes its previous function.

When the keypad finds the correct character within the ISR function, this also clear the interrupt flag stored within the ‘74. Once the interrupt flag is cleared, the signal drops to low and the half of the ‘244 connected to the power supply is turned back on again and is ready to detect more key presses.

Design

‘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

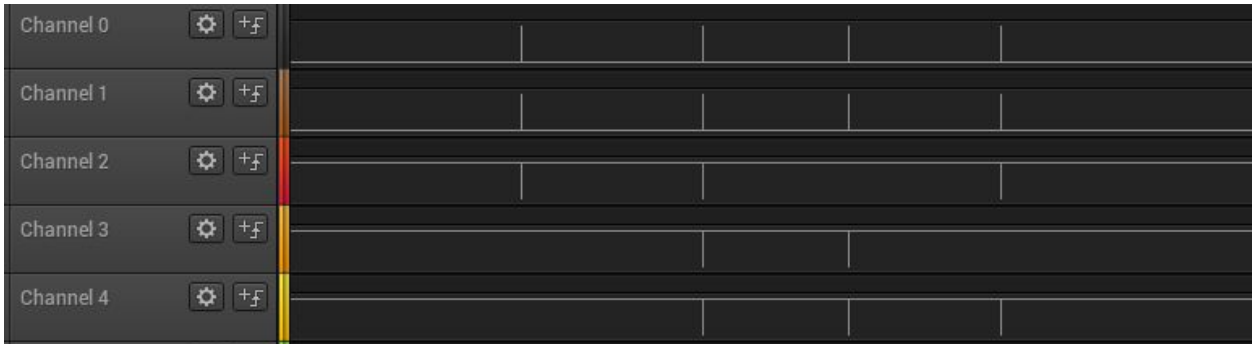
Input	**Output**
A	**Y**
L	H
H	L

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

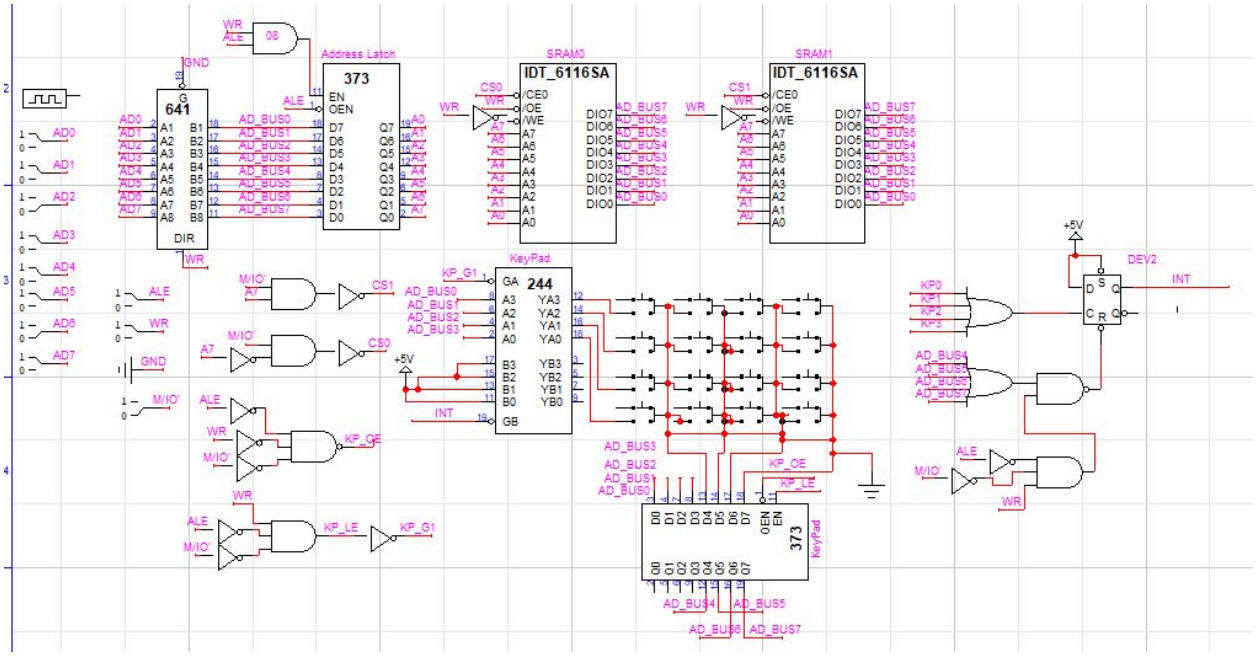
H = High voltage, L = Low voltage, X = Immaterial



Channel 0 and 1 are connected to the interrupt signal. When a button is pushed, the interrupt signal is generated. Channel 2 through 4 are connected to the address bits that are input to the ‘4002 that generates the clock signal for the ‘74 that stores the interrupt flag.. On the first button push, the character 2 is pushed, then 9, then 4, then A.

Console output for the pushed buttons corresponding to the above waveform.

Schematics



Lab 4 Circuit Schematics

Parts List

1. 4 x 4 Keypad
2. IDT6116SA: CMOS Static RAM 16K (2K x 8-bit)
3. CD74LS4002: Dual 4-Input NOR Gate
4. SN74LS04: Hex Inverter
5. SN74LS08: Quadruple 2-Input Positive-AND Gate
6. SN74LS11: Triple 3-Input Positive-AND Gate
7. SN74LS74: Dual D-Type Positive Edge-Tripped Flip-Flops
8. SN74LS244: Octal Buffers and Line Drivers with 3-State Outputs
9. SN74HC373: Octal D-Type Transparent Latch
10. SN74LS641: Octal Bus Transceiver
11. 20-pin wire-wrapping DIP socket
12. 14-pin wire-wrapping DIP socket
13. 30 gauge wire
14. 40-pin ribbon cable
15. 1 Kohm resistors
16. SJTwo board
17. 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

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.

When a user enters 1, they can read the data from a certain address. When a user enters 2, they can write data to a specific address. When the user presses 3, they can type using the keypad. When a user presses 0, they can quit the program.

When the keypad is pressed, regardless of where in the program the state is currently, the program will stop and enter the interrupt service routine. It will print the character on the keypad that was pressed to console and then resume whatever function it was previously doing.

To demo the circuit, a button is pushed and the character is printed onto the screen. When the user is writing or reading to or from memory, the user can press a keypad and print a character to the screen before resuming the read or write.

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;
   }
//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 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
      {
        printf("\n The operation entered is invalid.\n");
      }

    }
  }
  return 0;
}
```