Angelica Semenec

CmpE 127 Lab

Lab 2
Peripheral Interfacing: SRAM Memory

Abstract:

The purpose of this lab is to interface two SRAMs with the existing architecture created in Lab 1. The SRAM utilizes the addresses created in the previous lab and inputs them as the address signal into the SRAM chips. Using the M/IO’ signal generated from the SJTwo board and the 8th address bit, each of the SRAM chips can be selected, one at a time, and data can be written to or read from a specific address.

Functionality

This circuit functions as a storage for 256 bytes of memory using two SRAM chips (IDT6116SA). Each SRAM can utilize up to 11 address lines, however the CPU only produces 8 and so only 8 of the address inputs of each chip is used. The unused inputs, A_8 through A_{10}, are tied to the power supply to avoid any floating inputs.

Each SRAM can have data written to a specific memory address and have data read from a specific memory address. When memory is being written to or read from, the address is first latched into the ‘373. The ‘373 latch enable is then closed to avoid data from the address/data bus changing the latched address. When the data is read or written, the output of the ‘373 is enabled and the address is tied to the address inputs of the SRAM. The 8th address, A_7, is used to operate the chip select. To select the lower half of memory, A_7 must be low and the M/IO* must be high to indicate the circuit is interacting with the memory device. To select the upper half of memory, A_7 must be high and M/IO* must be high as well.

The SJTwo board does not output a “read” signal. Instead, the write signal is used to determine when the SRAM is being read from or written to. When write is active, the function will be to write data to a memory address. When write is not active, the function will be to read data from a memory address.

The software that is flashed onto the SJTwo board is designed so that the user can easily interface with the memory devices. A menu prompts the user to input a function: 1. Read, 2. Write, or 3. Quit. If the user wants to read from memory, they can input a memory address and the contents will be printed to console as a decimal value. If the user wants to write to a memory location, they will be prompted for the address and then the data to be written. If the user wishes to quit, they will exit the program.

Design

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

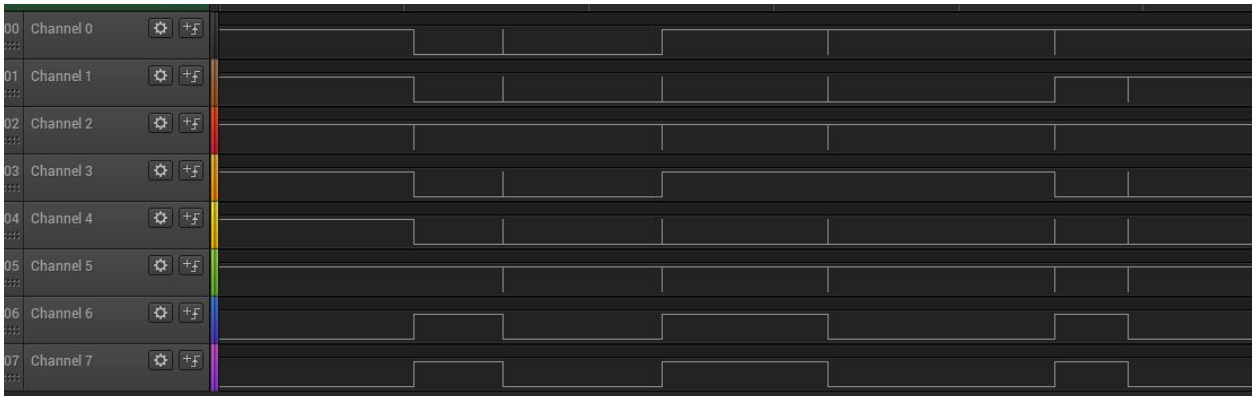
Control Inputs		**Operation**
G’	**DIR**	**‘641**
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

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

‘08 Voltage Table

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

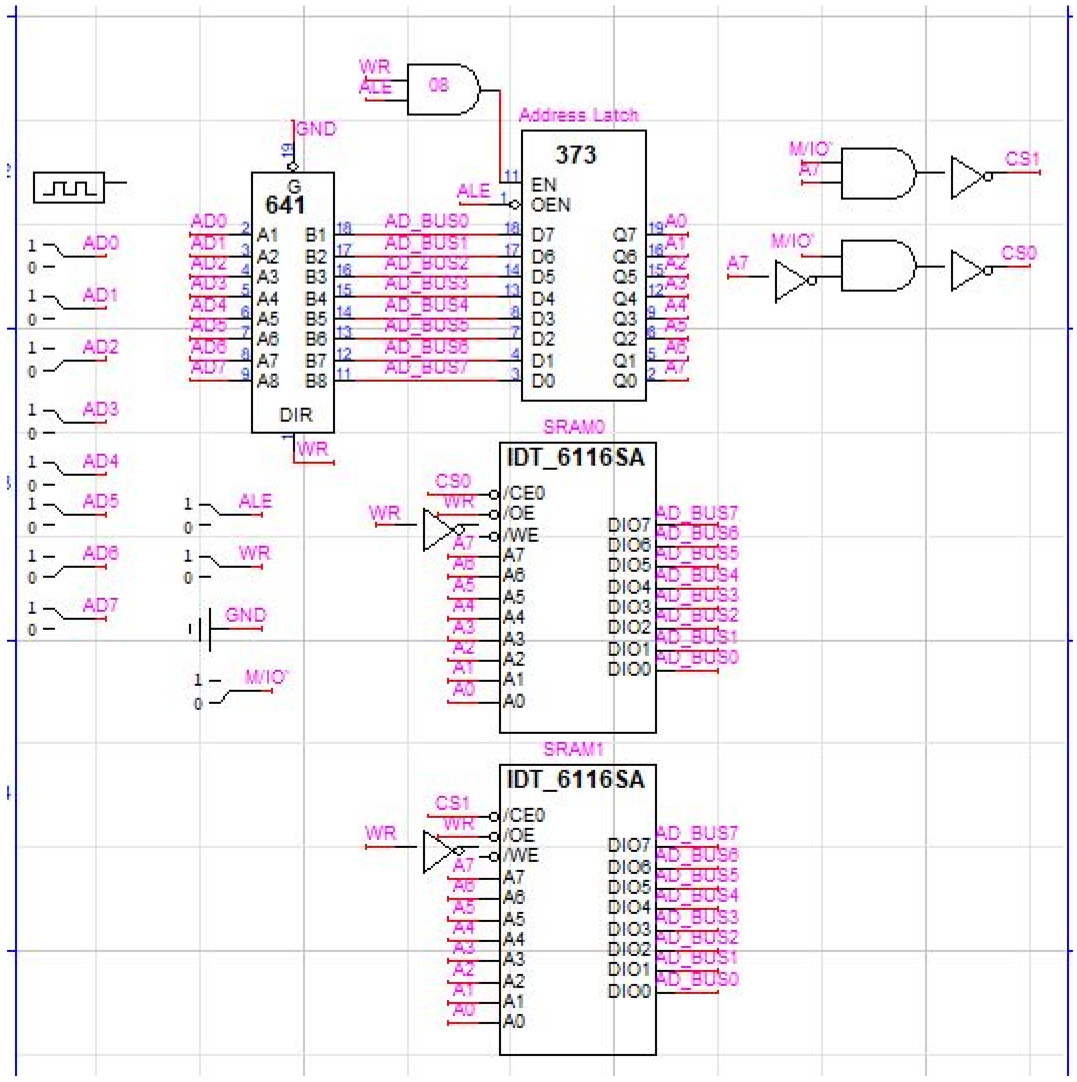
H = High voltage, L = Low voltage



Logic Analyzer Waveform: Channel 0 - Channel 3 correspond to the data output of bits 0 through 3 respectively of the lower half of memory. Channel 4 corresponds to the chip select of the lower half of memory and Channel 5 corresponds to the chip select of the upper half of memory. Channel 6 corresponds to the output enable of the lower half of memory and Channel 7 corresponds to the output enable of the upper half of memory.

In the waveform shown above, the circuit is first reset. 68 is then written into address 5 and then read from it. 45 is then written into address 10 and read. In the final write and read cycle, 7 is written to address 45 and then read back out.

Schematics



Lab 2 Circuit Schematics

Part List

1. IDT6116SA: CMOS Static RAM 16K (2K x 8-bit)
2. SN74LS641: Octal Bus Transceiver
3. SN74HC373: Octal D-Type Transparent Latch
4. SN74LS08: Quadruple 2-Input Positive-AND Gates
5. 20-pin wire-wrapping headers
6. 14-pin wire-wrapping headers
7. 30 gauge wire
8. 40-pin ribbon cable
9. 1 Kohm resistors
10. SJTwo board
11. PCB

Algorithms

OE’ (Output Enable) ‘373:

OE’ = ALE

LE (Latch Enable) ‘373:

LE = ALE * WR

CS0’ (Chip Select 0) SRAM:

Selects the lower half of memory

CS0’ = (A_7’ * M/IO’)’

CS1’ (Chip Select 1) SRAM:

Selects the upper half of memory

CS1’ = (A_7 * M/IO’)’

OE’ SRAM:

OE’ = WR

WE’ SRAM:

WE’ = WR’

Demo

The .hex file must first be created by running the command “make build” in the WSL terminal for Lab 2. Once it is created, the hyperload can be opened and the .hex file can then be loaded onto the SJTwo board with the working program. After the program is loaded, Hercules is opened and the user can now interface with the program working on the board after the middle reset button is pushed on the SJTwo.

The console will output text to the user prompting them to enter the number of an operating, 1 for reading, 2 for writing, and 3 to quit. When writing to the board, the user will be prompted to enter in the address that the data will be loaded into and then the data that they wish to write. Once it has successfully completed, it will output a message to the user and display the prompt for selecting the functional operation again. When reading, the user is prompted only for the address and then the data is fetched from the address in the appropriate SRAM and displayed on the screen.

To demo the lab, a few different values were written into different addresses. The addresses were then read back to ensure the data was stored correctly and in the proper location.

A logic analyzer was then connected to the data output of the upper SRAM as well as the chip select input of both SRAM chips and the output enable. The simulation was recorded for 60 seconds, which recorded data being written into the upper SRAM memory and then read back out. The resulting waveform was then displayed using the logic analyzer software to show how the values changed over time with each operation.

Source Code

main()

```
//Libraries used
#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 Definition of functions
class Bus
{
   public:
   enum class ControlType
   {
       kMemory = 0,
       kIO
   };

//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();
   }

//Function writes data to memory
   void WritetoMem(int address, int data)
   {
       sendAddress(address);
       writeData(data);

       return;
   }

//Function reads data from a memory address
   int ReadfromMem(int address)
   {
       sendAddress(address);
       int data = readData();

       return data;
   }

//Function sends address to the address latch
   void sendAddress(int address)
   {
       ALE.SetHigh();
       Write.SetHigh();
       M_IO.SetLow();

       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;
   }

//Function writes data to memory
   void writeData(int data)
   {
       ALE.SetLow();
       Write.SetHigh();
       M_IO.SetHigh();

     //Changes input decimal value to binary
       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;
   }

//Function reads data
   int readData()
   {
       ALE.SetLow();
       Write.SetLow();
       M_IO.SetHigh();

       int data = 0;

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

       return data;
   }

//Pins utilized
   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 = 0;

//Console interaction loop that allows a user to choose what operation they’d like
// to perform
   while(op != 3)
```

```
  {
    printf("\n Please enter the number of the function you'd like to perform:");
        printf("\n 1. Read data from memory");
        printf("\n 2. Write data to memory");
        printf("\n 3. Quit");
        printf("\n Operation: ");
    scanf("%d", &op);
    if (op != -1 && op != 3)
    {
      //Calls the Read function
      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);
      }
      //Calls the Write function
      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);
      }
      //Prints if invalid operation value is entered
      else
      {
        printf("\n The operation entered is invalid.\n");
      }

    }
  }
  return 0;
}
```