CMPE 127, MICROPROCESSOR DESIGN 1

SECTION 02/03, FALL 2020

LAB MANUAL FOR REMOTE LABORATORY INSTRUCTION

Prepared By:

Dr. Ilkan Çokgör, Instructor

Steven Lu

**Lab 1: Introduction**

During this semester you will be designing and building your own microcomputer complete with a microprocessor unit, an address bus and an address decoding circuit, a control bus, a data bus, random access memory, Input/Output interface, the interrupt structure, a keypad as the input device, and a Liquid Crystal Display as the output device. The microprocessor will be emulated by a specialized microcontroller board, the “SJTwo” (Figure 1). You will be developing software in C language to control the microprocessor functions and will load this software onto SJTwo to emulate a microprocessor.

|  |  |
| --- | --- |
|  | Figure 1: The *SJTwo* microprocessor emulation board. |

The rest of the computer will be built on a prototype board (Figure 2). We will use only the solder pad grid array portion of the board for remote laboratory instruction, as shown inside the red box in Figure 2. Both the SJTwo and the prototype board are provided by the Software & Computer Engineering Society at SJSU.

|  |  |
| --- | --- |
|  |  |

Figure 2: Prototype board for the microcomputer build.

**Required Parts and Tools:**

A list of all required parts, tools and consumables to build your microcomputer will be provided in a separate spreadsheet file (CMPE 127 Parts List) by your instructor. It is essential that you order all parts as soon as you have access to this list. Each part and tool have been selected from a high-volume stock. Lack of parts or tools will not be accepted as an excuse for not completing the lab assignments on time.

Hand drawing of the circuits is not allowed. You will need to use an engineering grade circuit schematic tool. LogicWorks is a “good” schematics tool. It must be purchased if you do not have it.

Go to <https://designworkssolutions.com/logicworks/>

Add to cart. During checkout use the coupon code “lwwsjsu40” for a 40% discount.

You can use any other schematics tools as long as it generates engineering grade schematics.

**Sources for Parts and Electronic Components:**

The Parts List include links to where they can be ordered. However, it is strongly recommended that you check Excess Solutions first (see below for the address). Excess Solutions is near campus, and has good inventory and reasonable prices.

* Excess Solutions (Near campus; 1555 S 7th St #4D, San Jose, CA 95112)
* Anchor-electronics (2040 Walsh Avenue, Santa Clara, CA 95050)
* Mouser Electronics (www.mouser.com)
* Digikey (www.digikey.com)
* Arrow Electronics (www.arrow.com)
* Jameco (www.jameco.com)
* Amazon.com

**Laboratory Grading Policy:**

Each laboratory session has report submission requirements. Laboratory reports are due at the end of each session. Reports are due to be submitted electronically through Canvas. Laboratory reports will be graded over full grade scale if submitted on time. Laboratory reports submitted within 24 hours of submission deadline will be graded over 50% scale. No laboratory report submissions will be accepted past 24 hours after submission deadline.

Each laboratory assignment builds on the previous assignments. You will need to complete the laboratory assignments in order to be able to proceed even if you have not submitted your report on time.

**Laboratory Schedule:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Section** | **Lab #** | **Date** | **Day** | **Time** | **Lab Topic** | **Points Towards Grade** |
| 02 | 1 | January 27, 2020 | Mon | 18:00-20:50 | Introduction |  |
| 03 | 1 | January 29, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 2 | February 3, 2020 | Mon | 18:00-20:50 | SJTwo Board Interface Set up | 2 |
| 03 | 2 | February 5, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 3 | February 10, 2020 | Mon | 18:00-20:50 | Prototype board build | 3 |
| 03 | 3 | February 12, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 4 | February 17, 2020 | Mon | 18:00-20:50 | Control signals and data/address bus demultiplexing | 2 |
| 03 | 4 | February 19, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 5 | February 24, 2020 | Mon | 18:00-20:50 | Control signals and data/address bus demultiplexing | 3 |
| 03 | 5 | February 26, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 6 | March 2, 2020 | Mon | 18:00-20:50 | Address decoding circuit, memory map development and SRAM interface | 2 |
| 03 | 6 | March 4, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 7 | March 9, 2020 | Mon | 18:00-20:50 | Address decoding circuit, memory map development and SRAM interface | 3 |
| 03 | 7 | March 11, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 8 | March 16, 2020 | Mon | 18:00-20:50 | Programmable I/O Interface | 2 |
| 03 | 8 | March 18, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 9 | March 23, 2020 | Mon | 18:00-20:50 | Programmable I/O Interface | 3 |
| 03 | 9 | March 25, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
|  |  | March 30, 2020 | Mon | 18:00-20:50 | No Laboratory |  |
|  |  | April 1, 2020 | Wed | 18:00-20:50 | No Laboratory |  |
|  |  |  |  |  |  |  |
| 02 | 10 | April 6, 2020 | Mon | 18:00-20:50 | Keypad interface and Polled I/O | 2 |
| 03 | 10 | April 8, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 11 | April 13, 2020 | Mon | 18:00-20:50 | Keypad interface and Polled I/O | 3 |
| 03 | 11 | April 15, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 12 | April 20, 2020 | Mon | 18:00-20:50 | Interrupt I/O | 2 |
| 03 | 12 | April 22, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 13 | April 27, 2020 | Mon | 18:00-20:50 | Interrupt I/O | 3 |
| 03 | 13 | April 29, 2020 | Wed | 18:00-20:50 |
|  |  |  |  |  |  |  |
| 02 | 14 | May 4, 2020 | Mon | 18:00-20:50 | Extra Credit: LCD Interface | 3 (Extra Credit) |
| 03 | 14 | May 6, 2020 | Wed | 18:00-20:50 |

**Preparing for the Lab:**

1. Each lab has a prelab assignment. Student needs to make sure to complete the pre-lab requirements before coming to the lab. There is absolutely no spare time during the lab to complete any pre-lab assignments. Some of the prelab assignments are very involved. It is highly recommended that the student starts preparing the prelab assignments a week in advance.
2. You will be required to make connections by wire wrapping. There are many instructional videos available on YouTube about wire-wrapping that you may find useful (see below). Using color coding (e.g. Vcc: red, GND: black, data: yellow, address: green, control: purple, etc.) will make troubleshooting your wiring easier if necessary. **Always check the continuity of your connections with a multimeter after wire wrapping.** You will need a wire wrapping tool and 30AWG solid wire. Please refer to the parts list for examples of what to buy.

Here are some useful links for learning how to wire wrap:

**Videos:**

Introduction to Wire Wrapping - <https://www.youtube.com/watch?v=IXvEDM-m9CE>

Using a Wire Wrap Tool - <https://www.youtube.com/watch?v=yQc22mxoUNI>

**Articles:**

Wire Wrapping - <https://www.jameco.com/Jameco/workshop/techtip/wirewrap.html>

Insulation Stripping - <https://learn.sparkfun.com/tutorials/working-with-wire/how-to-use-a-wire-wrap-tool>

1. ORDER THE PARTS NOW!

**Laboratory Assignments and Reports:**

All Laboratory assignments and reports are due on the day of the lab. All reports will be submitted via Canvas electronically. Circuit schematics, timing diagrams, software code and any other required assignments can be submitted in pdf or a picture format form. It is your responsibility to submit assignments that are clearly readable.

**Trouble Shooting:**

There are programs provided on Canvas to help troubleshoot hardware problems. Each of these software snippets are intended to be used for their specific lab.

In each program, there are instructions listed that you can follow in order to verify that your chip is properly connected and powered. A multimeter will be required for verification, as you will be looking at voltages in order to determine connectivity.

Every single chip added to the board has a program to run – check that your wire connections are correct before asking for help on the software!

**Lab 2: SJTwo Board Interface Set up**

The purpose of this lab is to install a working environment of the SJTwo’s Framework.

We will be using a Linux distribution called Ubuntu to run our development environment for the SJTwo board. Using the Ubuntu terminal greatly improves speed of development once set up, and allows us to gain basic experience with a Linux terminal.

Go to <https://sjsu-dev2.readthedocs.io/en/latest/index.html> , click on the “Getting Started with the SJTwo” link located at the bottom right of the page, and follow all instructions carefully and completely. Refer to the following as you go through each set up step and to complete your Lab 2 assignments:

**Installing WSL for Windows 10 Users**

1. If you are using a Windows machine, you need to install WSL for Windows 10 Users. Before proceeding with this step, *you will need to update to the latest Windows version.*

Follow the “Install WSL on Windows 10” link and make sure to follow all instructions in “Windows Subsystem for Linux Installation Guide for Windows 10” page. *HINT: You can locate Windows PowerShell under the Start menu. Right click on Windows PowerShell to run as Administrator.*

The Linux distribution that we will be using is *Ubuntu*. Follow all instructions, including installing, initializing and updating and upgrading Ubuntu in the “Windows Subsystem for Linux Installation Guide for Windows 10”. Mac users will need to use a virtualbox and download an ubuntu VM. *Make sure to note your Unix username and password. You will need it in the following steps.*

**Installing Serial Device Driver for Windows 10 and Mac Users**

1. Now go back to <https://sjsu-dev2.readthedocs.io/en/latest/getting_started/getting_started/> and follow instructions under “Install the Serial Device Driver for Windows 10 and Mac.”

**Installation**

1. Follow the steps under “Installation” in

<https://sjsu-dev2.readthedocs.io/en/latest/getting_started/getting_started/>

Windows users should put the SJSU-Dev2 directory in /mnt/c/ directory.

Go to <https://github.com/SJSU-Dev2/SJSU-Dev2>  and click on the green “Code” button to Download ZIP.

The downloaded file name will be “SJSU-Dev2-master”. Change it to “SJSU-Dev2”.

NOTE: You will need to run the set up utility from the Ubuntu terminal. Open your Ubuntu terminal, go to SJSU-Dev2 folder and execute ./setup. You will need your Ubuntu password to run the setup utility.

**Building and Loading the "Hello World" Application**

1. Follow the steps under “Building and Loading the "Hello World" Application” to build and load the "Hello World" application to your SJTwo board.

Before starting this stage, you will need to connect your SJTwo board to your computer’s USB port.

NOTE: After setting up Telemetry, be sure to set the Baud Rate to 38400 from the pull down menu on the Telemetry page.

After Step 3, view the ‘Hello World’ output messages from the board using Telemetry (or Hercules).

Take a screen shot of your Telemetry or Hercules terminal and submit it via Canvas as your Lab 2 assignment.

**Creating Your Own Project**

1. Go to projects/ directory. Copy ‘starter’ folder and change its name to “CMPE127Lab”. Make sure to copy the starter folder with all its contents.
2. Using your favorite code text editor create a Bus.hpp file and copy the following. Bus.hpp will develop into your microprocessor bus control program.

|  |
| --- |
| #include <cstdint>  #include "L1\_Peripheral/lpc40xx/gpio.hpp"  class Bus  {    public:    void Initialize()    {      // Setting each pin as Open Drain      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\_enable.SetAsOutput();      ale.SetAsOutput();      m\_io.SetAsOutput();  write.SetAsOutput();  read.SetAsOutput();      interrupt.SetAsInput();    }    void MemWrite(uint8\_t address, uint8\_t data)    {    // Fill out rest of Memory Write Bus cycle here...    }  void IOWrite(uint8\_t address, uint8\_t data)    {  // Fill out I/O Write Bus cycle here...    }  uint8\_t MemRead(uint8\_t address)    {      // Fill out Memory Read Bus cycle here...    }  uint8\_t IORead(uint8\_t address)    {      // Fill out I/O Read Bus cycle here...    }    private:    sjsu::lpc40xx::Gpio ad[8] = {      sjsu::lpc40xx::Gpio(2, 2),      sjsu::lpc40xx::Gpio(2, 5),      sjsu::lpc40xx::Gpio(2, 7),      sjsu::lpc40xx::Gpio(2, 9),      sjsu::lpc40xx::Gpio(0, 15),      sjsu::lpc40xx::Gpio(0, 18),      sjsu::lpc40xx::Gpio(0, 1),      sjsu::lpc40xx::Gpio(0, 10),    };    sjsu::lpc40xx::Gpio write\_enable = sjsu::lpc40xx::Gpio(0, 17);    sjsu::lpc40xx::Gpio ale          = sjsu::lpc40xx::Gpio(0, 22);    sjsu::lpc40xx::Gpio m\_io         = sjsu::lpc40xx::Gpio(0, 0);    sjsu::lpc40xx::Gpio interrupt    = sjsu::lpc40xx::Gpio(0, 11);  sjsu::lpc40xx::Gpio write     = sjsu::lpc40xx::Gpio(0, 16);  sjsu::lpc40xx::Gpio read     = sjsu::lpc40xx::Gpio(2, 8);  }; |

1. Using your favorite code text editor create a main.cpp file and copy the following. This will develop into your microprocessor programming code.

|  |
| --- |
| #include <cstdint>  #include "Bus.hpp"  int main()  {  uint8\_t address;  uint8\_t data;  Bus bus;  bus.Initialize();  // Fill out your code here...  return 0;  } |

1. Some of the utilities that you need to know to develop your Bus.hpp and main.cpp:

SetAsOutput()- sets a GPIO pin as an output pin

SetAsInput()- sets a GPIO pin as an input pin

SetLow() – sets an output GPIO pin to LOW (logic 0; signal level 0V)

SetHigh()– sets an output GPIO pin to HIGH (logic 1; signal level 5V)

Use examples:

read.SetAsOutput();

interrupt.SetAsInput();

ad[0].SetHigh();

write.SetLow();

12) These utilities are aids in the development and troubleshooting of your code:

LOG\_INFO(“*your message here*”); /\* Publishes a message to the serial terminal \*/

/\* Use with #include "utility/log.hpp" \*/

sjsu::Delay(*X*ms); /\* Introduces a delay \*/

/\* Use with #include "utility/time.hpp" \*/

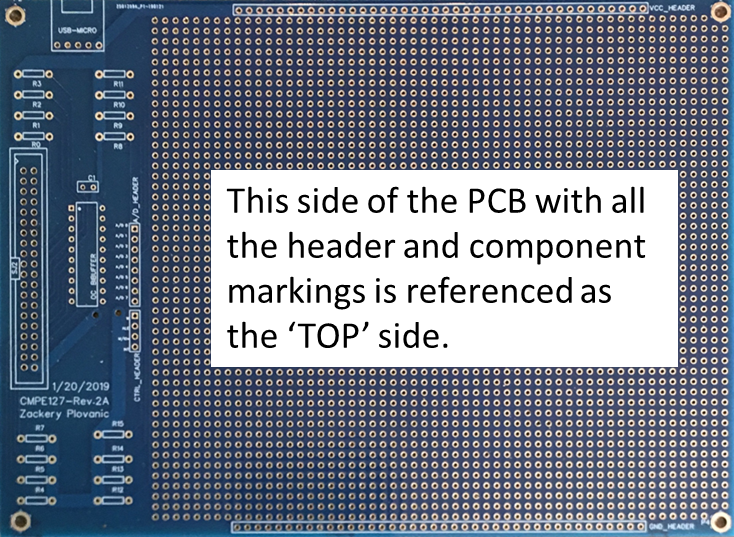
13) After using Telemetry or Hercules, you will need to disconnect or close the port for your SJTwo board to be able to load a program by ‘make flash’ again.

**Lab 3: Prototype board build**

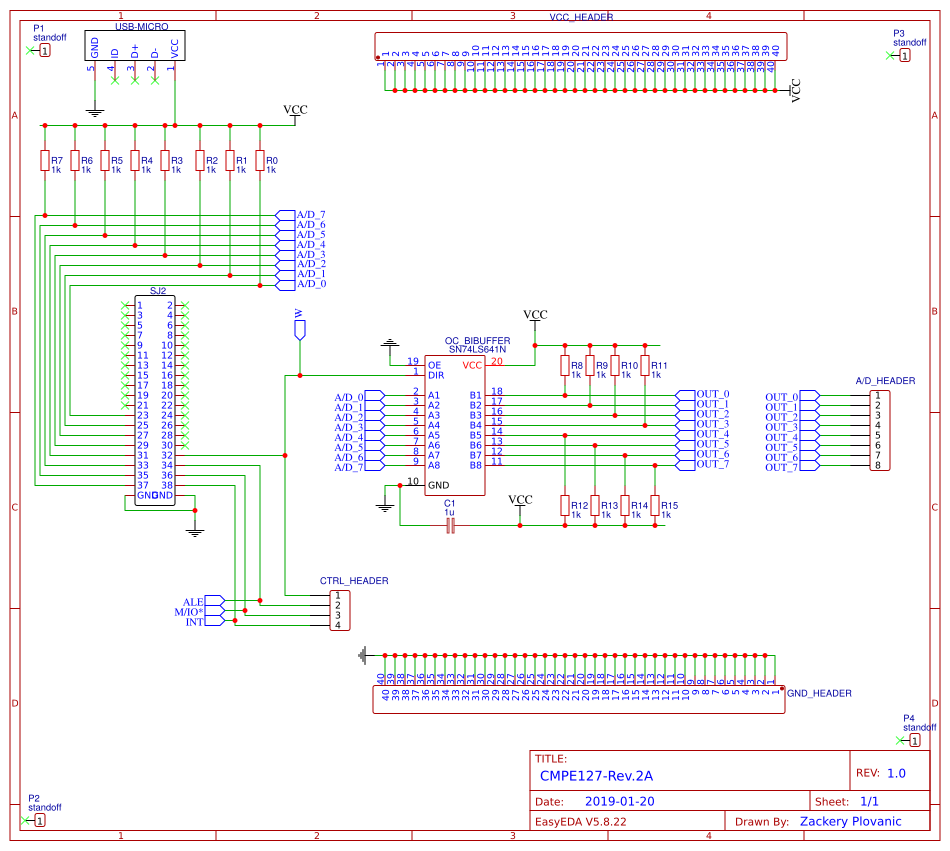
The purpose of this lab is to bring up your prototype board to be used in subsequent labs.

**IMPORTANT:**

The face of the prototype board with the header and component markings will be referenced as the ‘*top*’ of the PCB and the opposite face will be referenced as the *‘bottom*’ of the PCB. Soldering headers and components on the wrong side may cause difficulty in wiring your board. Refer to the picture below.



Schematic diagram for the prototype board is given below:



Pre-lab:

1. Solder the 2 x 20 pin dual male header prototype board. IMPORTANT: The shorter legs should go through the via holes and be soldered from the bottom of the board. The longer header pins should be facing up as shown in Figure 3 below.

|  |  |
| --- | --- |
|  | Figure 3: 2x20 pin dual male header as soldered on the prototype board. |

1. Solder the 4-pin CTRL-HEADER and 8-pin A/D\_HEADER. IMPORTANT: The shorter pins should go through the via holes from the bottom of the board and be soldered on the top of the board. The longer header pins should be facing DOWN as shown in Figure 4a below. This configuration will allow you to use the longer pins for wire-wrapping at the bottom of the board (Figure 4b).

|  |  |
| --- | --- |
|  | Figure 4a: CTRL and A/D Headers should be soldered so that the longer pins are at the bottom of the board. |
|  | Figure 4b: Picture of the bottom of the board after soldering the A/D and CTRL header pins. Longer pins will be used for wire-wrapping. |

1. Solder 1 x 40 pin male headers for VCC\_HEADER and GND-HEADER. IMPORTANT: The shorter pins should go through the via holes from the bottom of the board and be soldered on the top of the board. The longer header pins should be facing DOWN as shown in Figure 5 below.

|  |  |
| --- | --- |
|  | Figure 5: 2 each of 1 x 40 pin male headers soldered to VCC\_HEADER and GND\_HEADER. Longer pins at the bottom of the board will be used for wire-wrapping. |

1. Solder the sixteen 1K Ohm resistors where shown on the board (Figure 6a). Remember to trim the excess resistor legs (Figure 6b).

|  |  |
| --- | --- |
| Figure 6a | Figure 6b |

1. Solder the 20-pin DIP socket onto the OC\_Buffer label on the prototype board (Figure 7). IMPORTANT: Pay attention to the direction of the socket. At this time, also solder the 10uF and 0.1uF decoupling capacitors. Note that the 10uF capacitor is a polarized electrolytic capacitor and its negative terminal should be connected to the ground.

|  |  |
| --- | --- |
|  | Figure 7 |

1. Solder the Micro-USB break-out board to the top left of the prototype board as follows: First solder the male header to the micro-USB break-out board using the long legs of the header through the micro-USB board (Figure 8a). Then, solder the header to the prototype board (Figure 8b). IMPORTANT: The Micro-USB connector is very fragile. Avoid cantilevering the connector when a USB cable is connected.

|  |  |
| --- | --- |
| Figure 8a | Figure 8b |

1. You will need to create the Read and Write control signal headers yourself. Insert the long legs of a three-pin male header from top of the board. Solder the legs to the pads. You will need to solder two 30AWG wires from pins #6 and #7 of the internal row of the ribbon connector. Refer to Figure 9 for correct connections.

|  |  |
| --- | --- |
| Figure 9a: Read/Write header wiring. | Figure 9b: Read/Write header top view. |

1. Refer to the schematic of the protoboard and perform a continuity test for each connection.
   1. Check the continuity between pins #6 and #7 of the internal row of the ribbon connector and the Read and Write headers,
   2. Check the continuity between A/D pins on the ribbon cable connector and the input pins of the SN74LS641 (check this on the DIP socket pins without inserting in the SN74LS641),
   3. Check the continuity between the output pins of the SN74LS641 (you can check this on the DIP socket pins without inserting in the SN74LS641) and the A/D header pins,
   4. Check the continuity between the GND and VCC pins of the micro-USB break out board and the GND and VCC Header pins,
   5. Check the continuity of the W, ALE, M\_IO, and INT pins of the ribbon connector and the CTRL Header pins,
   6. Check and verify values of all pull-up resistors.
2. Plug the SN74LS641 in the socket paying attention to correct orientation.

**Laboratory Background:**

You are given 8 Address/Data Pins. These are 8 of the LPC4078FBD80 microcontroller GPIO pins which will be used to simulate the address/data bus:

* ad[0]: GPIO(2,2);
* ad[1]: GPIO(2,5);
* ad[2]: GPIO(2,7);
* ad[3]: GPIO(2,9);
* ad[4]: GPIO(0,15);
* ad[5]: GPIO(0,18);
* ad[6]: GPIO(0,1);
* ad[7]: GPIO(0,10)

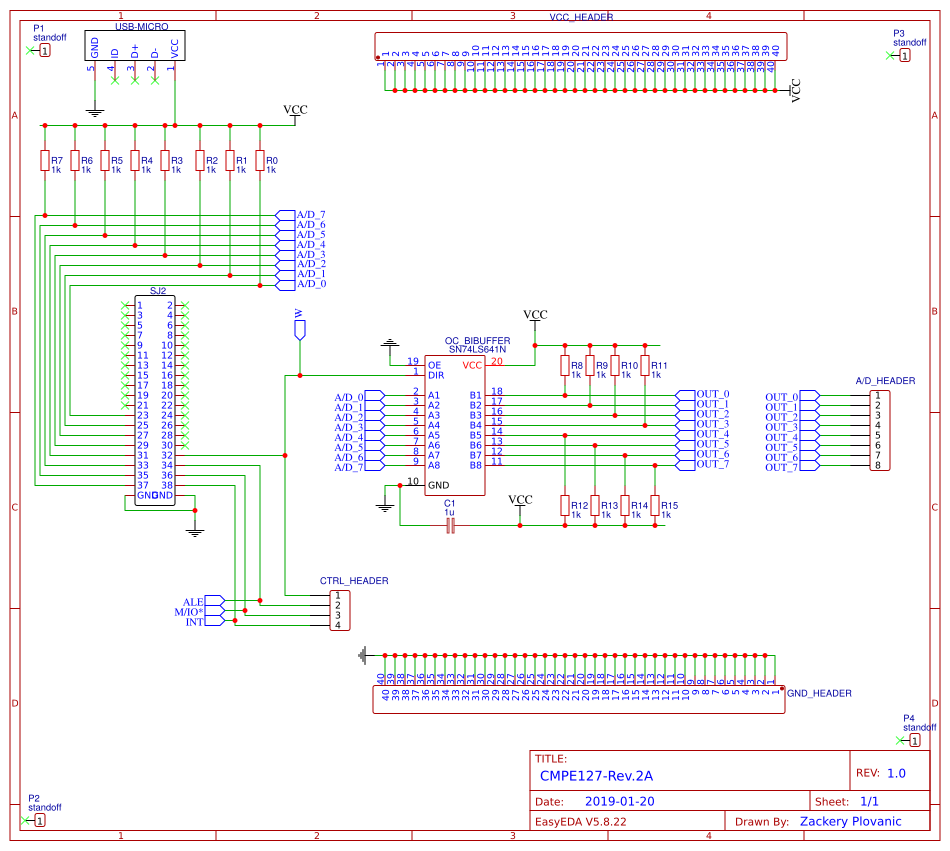
You are given 6 Control Pins. These are 6 of the LPC4078FBD80 microcontroller GPIO pins which will be used to simulate to simulate the control bus:

* W: GPIO(0,17);
* ALE: GPIO(0,22);
* M/IO\*: GPIO(0,0);
* INT: GPIO(0,11);
* WR: GPIO(0,16);
* RD: GPIO(2,8)

An Octal Transceiver (SN74LS641N) is connected to the 8 A/D Pins.

The W control pin is connected to the SN74LS641N transceiver DIR pin. This control signal is used to select the direction of the bus (See Figure below).

The OE pin is connected to the ground - means no bus isolation (See Figure below).



Laboratory Assignment:

1) Develop your main.cpp and Bus.hpp programs so that by calling the MemWrite function from main.cpp you are outputting the following logic signals on the output pins of the SN74LS641:

ad[0]: 0

ad[1]: 1

ad[2]: 0

ad[3]: 1

ad[4]: 0

ad[5]: 1

ad[6]: 0

ad[7]: 1

Refer to the SN74LS641 datasheet and the prototype board schematics to utilize the right control signal for the correct operation of the octal bus transceiver.

2) Check the SN74LS641N **B** output pins with a multimeter or a logic analyzer to verify that you are able to output the desired signals on the address/data bus.

Laboratory Report:

1. Demonstrate to the instructor the signals at the A/D Header pins either by a logic analyzer, or by a multimeter.
2. Submit your main.cpp and Bus.hpp code in Canvas.

**Lab 4: Control signals and data/address bus demultiplexing**

The goal of this lab is to familiarize yourselves with the many control signals of a microprocessor and implement the logic circuit to demultiplex address and data buses.

Prelab:

Study the control signals on your PCB.

Refer to Figure 10a and b, and Figure 11:

|  |  |
| --- | --- |
| Figure 10a: SJTwo connector pin designations | Figure 10b: Protoboard connector pin designations that correspond to the SJTwo pins. |

Address/Data:

**ad[0]-ad[7]**: The address/data bus lines are the multiplexed address and data bus of the processor and contain the memory address or the I/O port number whenever ALE signal is active (logic 1) or data whenever ALE is inactive (logic 0). These pins are accessible from the A/D\_HEADER on the prototype board. The A/D\_HEADER pins are connected to the output of the SN74LS641. When these pins contain memory address or the I/O port number, or when data is being output, the SN74LS641 must be in the output mode (A data to B bus). When these pins contain data that is being read, the SN74LS641 must be in input mode (B data to A bus). The SN74LS641 output or input modes are selected by the **write-enable** pin (W).

Control Signal Definitions:

**read**: Whenever the read signal is logic 0, the data bus is receptive to data from the memory or I/O devices connected to the system. You will need to connect the connector pin to a male-header pin on your prototype board to access this signal. This pin is shown as RD in Figure 10b.

**interrupt**: Interrupt request is used to request a hardware interrupt. This pin is accessible from the CTRL\_HEADER, INT pin.

**m\_io**: The M/IO pin selects memory or I/O. This pin indicates that the microprocessor address bus contains either a memory address or an I/O port address. When m\_io is HIGH, the memory space is addressed, and when m\_io is LOW, the I/O space is addressed. This signal is accessible from the CTRL\_HEADER, M/IO\* pin.

**write**: The write control signal indicates that the microprocessor is outputting data to a memory or I/O device. During the time that the write is a logic 0, the data bus contains valid data for memory or I/O. This pin is shown as WR in Figure 10b. You will need to connect the connector pin to a male-header pin on your PCB to access this signal. Do not confuse this signal with **write\_enable** (W) which is used to control the direction of the SN74LS641 transceiver.

**ale**: Address latch enable shows that the microprocessor address/data bus contains address information. This address can be a memory address or an I/O port number. This signal is accessible from the CTRL\_HEADER, ALE pin.

**write\_enable**: Write enable signal controls the direction of the SN74LS641 octal transceiver. When address/data lines contain memory address or the I/O port number, or when data is being output, the **write\_enable** should be logic 1. When address/data lines contain data that is being read, the **write\_enable** should be logic 0. This signal is accessible from the CTRL\_HEADER, W pin.

|  |  |
| --- | --- |
|  | Figure 11 |

Laboratory Assignment:

1) Design the logic circuit to demultiplex the microprocessor address and data buses. Use an SN74LS373 octal latch to demultiplex the address/data bus connections ad[0]-ad[7]. Refer to the SN74LS373 datasheet for pin diagrams and logic tables.

Laboratory Report:

1) Submit the schematic of you circuit diagram through Canvas. Make sure to label address and data lines.

**Lab 5: Control signals and data/address bus demultiplexing**

This laboratory session continues from Lab 4.

Prelab:

1. Build the logic circuit you designed in Lab 4 on your prototype board. Glue the wire-wrapping socket (anchoring it at the four end pins should be enough to hold it in place securely) for the SN74LS373 before starting to wire wrap.

Laboratory Assignment:

1. Develop the memory write (MemWrite) function in your Bus.hpp bus control program so that you can output the 8-bit address on the A/D bus, latch the address, and then output the 8-bit data on the A/D bus.
2. Develop your main.cpp to demonstrate that your address/data demultiplexing circuit works as follows: select an 8-bit address and latch the address on SN74LS373. Then output your 8-bit data on the bus. You should read the address at the SN74LS373 output pins, and read your data at the SN74LS373 input pins. You can check your address and data either by a logic analyzer, or by with a multimeter.

Laboratory Report:

1. Demonstrate to the instructor the signals at the input and output of the SN74LS373 pins either by a logic analyzer, or by a multimeter.
2. Submit your main.cpp and Bus.hpp code in Canvas.

**Lab 6: Address decoding circuit, memory map development and SRAM interface**

The purpose of this lab is to design the memory address decoding circuit so that two static random-access memory (SRAM) chips can be interfaced with the microprocessor emulator. This exercise will show you how multiple SRAM chips can be interfaced to a processor, hence expanding the addressable memory space. You will be developing a ‘memory map’, which is a key component of microprocessor-based system design.

Prelab:

Study the datasheets for SN74LS138 and 6116.

Laboratory Assignment:

* 1. Design a logic circuit using the SN74LS138 3-to-8 line decoder for decoding the addresses for two 6116 SRAMs. Use A6 and A7 address lines for SRAM chip selection. You will need to use the m\_io control signal in your design so that your memory addresses are isolated from your I/O addresses. The logic circuit schematic needs to show clearly all data, address, and control bus connections. Refer to the datasheets of the SN74LS138 and 6116 for pin configurations and logic tables. Each memory location should have a unique address.
  2. Create the truth table for memory chip-select inputs. This table will have the A6 and A7 address lines and all relevant control lines as inputs, and memory chip-select signals as the outputs.
  3. Create your memory address map. On your memory address map, show the starting and ending addresses for each SRAM chip. Refer one of the chips as SRAM1 and the other as SRAM2. The address map should be represented in hexadecimal numbers.

Laboratory Report:

1. Submit your logic circuit schematic.
2. Submit the truth table for memory chip-select input pins. This table will have the A6 and A7 address lines and all relevant control lines as inputs, and memory chip-select signals as the outputs.
3. Submit your memory address map.
4. Create a WRITE and READ ‘sequence of events’ diagram similar to a timing diagram that shows the logic levels and timing of the control signals that you use to write and read to and from the SRAMs. Submit this chart.

**Lab 7: Address decoding circuit, memory map development and SRAM interface**

This laboratory session continues from Lab 6.

Prelab:

1. Build the circuitry you designed in Lab 6 on your prototype board. Use wire-wrapping type sockets for your SRAMS. Glue the sockets to your PCB before you start wire-wrapping.

Laboratory Assignment:

1. Develop your Bus.hpp bus control program to add memory read (MemRead) functionality.
2. Write a main.cpp program to demonstrate that any address from your memory map can be written to and read from. Pay particular attention to the timing diagrams in the data sheets. The control signals should be activated in a certain order for your circuit to work. Modify your Bus.hpp program if new control signals need to be added. The memory address to write to or to read from will be entered from your computer keyboard. The data that is read from memory can be printed on the SJTwo display or on your computer screen.

Laboratory Report:

1. Demonstrate to the instructor the working circuit by writing to and reading from both SRAM chips.
2. Submit your main.cpp and Bus.hpp code in Canvas.

**CMPE-127 SP20**

**Laboratory Assignment 7 Supplement**

**Dr. Ilkan Çokgör, Dr. Haluk Özemek**

1. You will be further developing the bus.MemWrite()and bus.MemRead() functions to write to, and read from, any location on your memory map.
2. How to implement *read* on the SJTwo GPIO pins (i.e. how to read ad[*x*]):

You will need to use the function call: ad[*x*].read(). This function returns 1 if ad[*x*] is HIGH, and returns 0 is ad[*x*] is LOW.

You will need to read each pin into a variable in bus.MemRead() to return to main(). Here is a sample code:

uint8\_t data = 0;

for (int i = 0; i < 8; i++)

{

data = data | (ad[i].Read() << i);

}

Don't forget to set each ad[*x*] pin to input **IF** you are reading: ad[*x*].SetAsInput

1. Write a processor initialization function to call when you first initialize main() (just like how you call bus.Initialize()). This will allow you to set the processor control pins to a known state before you call bus.MemWrite() or bus.MemRead(). When you are returning from these functions, don’t forget to set the control pins to their initialized state. Here are the recommended initial states for the control pins that we are currently using:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| W (write\_enable) | ALE | WE\* | RD\* | M/IO\* |
| H | L | H | H | H |

1. Laboratory Demonstration: (Needs to be performed in the order given below)

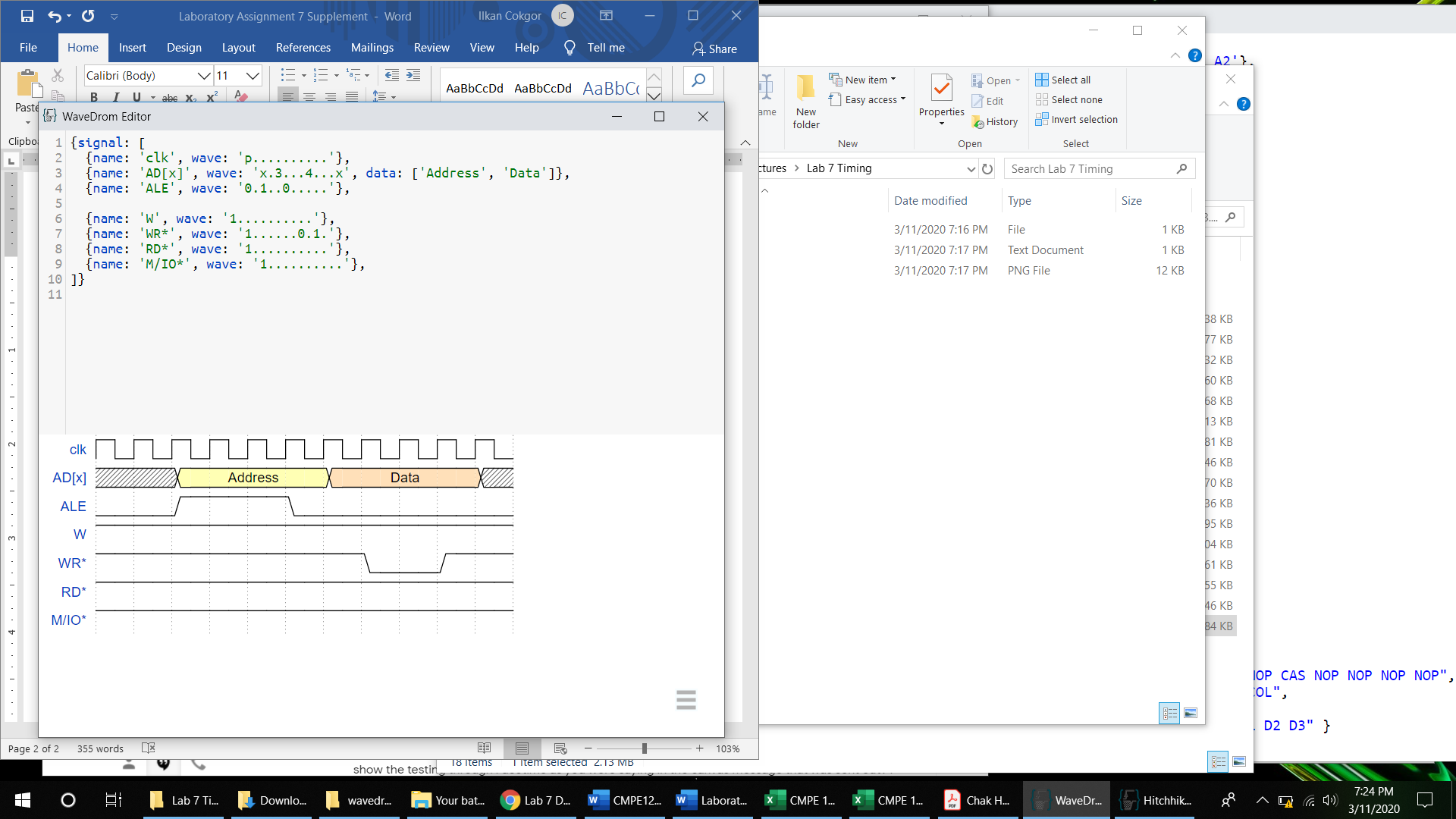
* Write an 8-bit data to a memory location in your address map in the region of SRAM1.
* Write an 8-bit data to a memory location in your address map in the region of SRAM2.
* Read the data you wrote to SRAM1.
* Read the data you wrote to SRAM2.
* Print data on Telemetry terminal.

Use the LOG\_INFO() function to output on Telemetry. LOG\_INFO() works just like printf()in C.

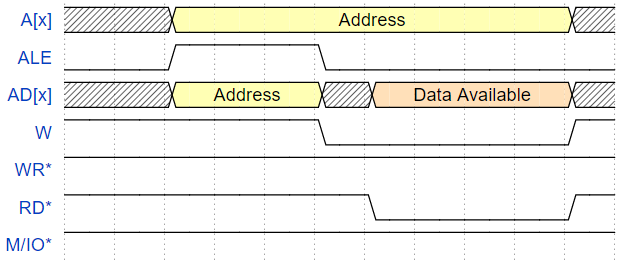
e.g. LOG\_INFO("Read from %08b: %08b", address, data);

1. Timing Diagrams for MemWrite() and MemRead():

*Simplified timing diagram for writing to SRAM:*



*Simplified timing diagram for reading from SRAM:*



1. Laboratory 7 Submission Requirements:
   1. Picture of the board (back and front) with permanent identification.
   2. Screen capture of the Telemetry terminal showing the addresses and data written and read back.
   3. main.cpp and Bus.hpp
2. Always keep your main.cpp and Bus.hpp intact. Remember that you may be required to do an online or a live demo upon request.
3. The timing diagrams above were generated using an online tool <https://wavedrom.com/>.

Other timing diagram generation tools that you might want to check out are:

timing-diagrams.com

timingeditor.sourceforge.net

**Lab 8: Programmable I/O Interface**

The purpose of this lab is to integrate a programmable peripheral interface, the 82C55, so that various input and output peripherals can be connected and can communicate with the microprocessor.

Prelab:

Study the datasheet for 82C55, focusing on programming the 82C55 and Mode 0 Operating Mode.

Laboratory Assignment:

1. Design the logic circuit to interface an 82C55 to your microprocessor. Refer to the 82C55 datasheet for pin configurations and logic diagrams. Note that each port of the 82C55 needs to have a unique port address. You will need to use the m\_io control signal in your design so that your I/O addresses are isolated from your memory addresses.

Laboratory Report:

1. Submit your logic circuit schematic.
2. Submit the truth table for 82C55 chip-select input pin. This table will have address lines and all relevant control lines as inputs, and 82C55 chip-select signal as the output.
3. Submit the port addresses for each I/O port of the 82C55, including the address of the command register.
4. Create a WRITE and READ timing diagrams that show the logic levels and timing of the control signals that you use to write and read to and from the 82C55. Submit this chart.

**Lab 9: Programmable I/O Interface**

This laboratory session continues from Lab 8.

In this Lab, we will program the 82C55 to operate in Mode 0 – Basic Input/Output.

Prelab:

1. Build the circuitry you designed in Lab 8 on your prototype board. Use wire-wrapping socket for the 82C55. Glue the socket on the board (anchoring the four corner pins is sufficient to hold the socket in place) before starting to wire-wrap.

Laboratory Assignment:

1. Develop your I/O write (IOWrite)and I/O read (IORead)functions for your Bus.hpp.
2. Develop a main.cpp program so that Port A of the 82C55 is programmed as an Output port and Port B is programmed as an Input port. Add to your Bus.hpp code so that you can write and read from a specific I/O port.

Laboratory Report:

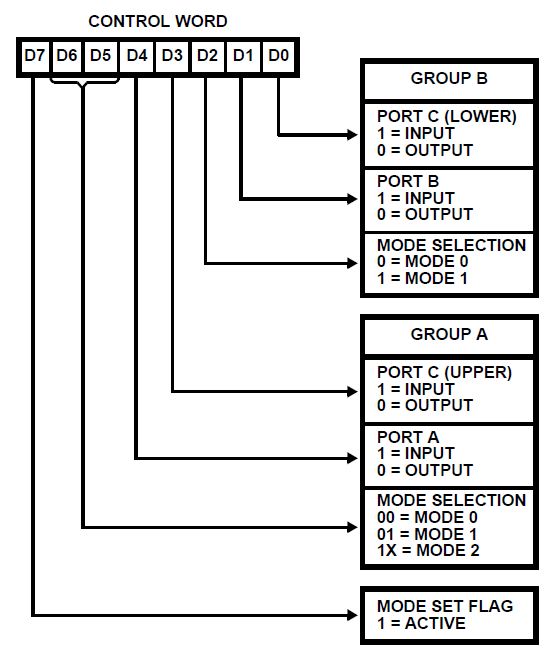
1. Connect Port A to Port B with jumper wires. Write to Port A and read Port B. Demonstrate to the instructor that you can write to the Output port A and read from Input Port B.
2. Submit your main.cpp and Bus.hpp code in Canvas.

**CMPE-127 SP20**

**Laboratory Assignment 9 Supplement**

**Dr. Ilkan Çokgör, Dr. Haluk Özemek**

1. After building the circuit you designed in Lab 8, do a continuity check between all connections to ensure circuit integrity.
2. The 82C55 is programmed by writing a Control Word to its Command Register. You are required to develop the Control Word to write to the Command Register. The bit definitions of the Control Word are given as follows (source: Intersil 82C55 data sheet.)



In Lab 9, we will operate the 82C55 in Mode 0 – Basic Input/Output. The relevant bits (i.e. D6, D5 and D2) of the Control Word should be configured to set the ports to the correct Mode.

You will need to configure Port A as an OUTPUT port and Port B as an INPUT Port. For now, Port C is don't-care.

It is essential that you refer to the 82C55 datasheet to understand Mode 0 operation (Intersil 82C55 datasheet pages 6-9) and the timing diagrams (Mode 0 timing diagrams: page 7). Not obeying the timing diagram requirements will result in incorrect or non-operation of the 82C55.

***IMPORTANT NOTE:*** 82C55 operation is very sensitive to the timing of the control signals. It is recommended that a delay is inserted after each control signal to ensure all timing requirements are met. We are not able to control the timing of our signals in great precision. However, the use of sjsu::Delay(1ms) should be sufficient.

1. The 82C55 needs to be configured before its ports can be used as intended. First, the Control Word must be written into the Command Register. The Command Register, Port A and Port B addresses have been determined by the circuit you designed in Lab 8. Select the Command Register address and write the Control Word.

After this step, data can be written to the output port (in our case, Port A), and can be read from the input port (Port B), by selecting their respective addresses.

1. Write 0x55 to Port A, and test Port A pins with a multimeter or a logic analyzer to confirm that the write operation has been successful (Note that an 82C55 port that is configured as an Output port will act as a latched output. This is the reason why you can observe the output with a multimeter.)
2. After you have verified that write has been successful, connect Port A pins to Port B pins with jumper wires. These wires are in your parts list (see type of wire below).



Since Port A is a latched Output and Port B is a buffered Input, you can connect Port A pins to Port B pins without damaging the 82C55.

1. Read at Port B address after writing to Port A address. You should be able to read back what you have written to Port A. Try different data values to make sure you can consistently write and read to the 82C55 ports.
2. Submission requirements: you are required to submit your main.cpp and Bus.hpp code files on Canvas.

**Lab 10: Keypad interface and Polled I/O**

The purpose of this lab is to connect a keypad to your programmable peripheral interface and detect key presses by using the Polled I/O technique.

Prelab:

1. Glue an 8-pin male header on your prototype board as shown in Figure 12 below to be used as the keypad connection.
2. Glue the 2 x 4 pin wire-wrapping DIP socket on your prototype board to be used for resistors (Refer to Figure 13 below).

|  |  |
| --- | --- |
|  | Figure 12:  8-pin male header soldered on to the prototype board. 4 of the pins are for keypad *column* connections and 4 of the pins are for keypad *row* connections. |

Laboratory Assignment:

1. Connect the keypad to the 82C55 as shown in the circuit below (Figure 13).

|  |  |
| --- | --- |
|  | Figure 13 |

1. Using the main.cpp you developed in Lab 9, read the input port (Port B). What signal level do you read? Is it consistent with the circuit design?
2. What signal level do you need at the output port (Port A) to be able to detect key presses at the input port? Develop your main.cpp to demonstrate that you can detect a single key press.

Laboratory Report:

1. Submit in Canvas as a text document:
   1. What signal level do you read when you read the input port when no key is pressed?
   2. What signal level do you need to output to be able to detect a key press?
   3. What signal level do you read at the input of the corresponding key when it is pressed?
2. Demonstrate to the instructor that you can detect a single key press.
3. Submit your main.cpp and Bus.hpp code in Canvas.

**Lab 11: Keypad interface and Polled I/O**

This laboratory session continues from Lab 10.

Prelab:

1. Develop a polling algorithm to detect and identify a key on the keypad when it is pressed.

Laboratory Assignment:

1. Implement the polling algorithm in your main.cpp to detect a key pressed.

Laboratory Report:

1. Demonstrate to the instructor that you can detect and identify the key pressed by displaying the corresponding character on the SJTwo display. Your program should be able to accept repetitive key presses without having to restart.

**Lab 12: Interrupt I/O**

The purpose of this lab is to interface a keypad by using the interrupt function of the 82C55 peripheral interface. Rather than constantly polling to see if a key has been pressed, this system will utilize an external interrupt to notify the microprocessor when a key has been pressed. The polling algorithm you have developed in the previous lab will be a part of the interrupt service routine that will be called when an interrupt is detected.

Pre-lab:

1. Study the datasheet for 82C55, focusing on Mode 1 Operating Mode.
2. Glue a 2 x 7 wire-wrapping DIP socket to be used for the 74LS32.

Laboratory Assignment:

1. Implement the circuit in Figure 14 below on your prototype board by adding the circuit for the interrupt signal.

|  |  |
| --- | --- |
|  | Figure 14 |

1. Add the circuit in Figure 14 to your overall schematic.
2. Develop your main.cpp so that 82C55 is configured in Operating Mode 1. Configure the 82C55 so that it is programmed to receive an interrupt from the keypad and generate an interrupt signal to the processor.

Laboratory Report:

1. Demonstrate to the instructor that pressing a key on the keypad results in an interrupt signal to be generated.
2. Submit your circuit schematic.

**Lab 13: Interrupt I/O**

This laboratory session continues from Lab 12.

Prelab:

There is no prelab work for Lab 13.

Laboratory Assignment:

1. Develop your main.cpp so that the polling of the keypad is now part of an Interrupt Service Routine (ISR). You will need to add the following code in your main.cpp to be able to detect the interrupt. In addition, you may have to move the GPIO interrupt line out of private in bus.hpp.

sjsu::lpc40xx::Gpio interrupt = sjsu::lpc40xx::Gpio(0,11);

interrupt.GetPin().SetPull(sjsu::Pin::Resistor::kPullUp);

interrupt.AttachInterrupt(insert\_here\_your-interrupt\_service\_routine\_name, sjsu::Gpio::Edge::kEdgeRising);

1. Develop your main.cpp so that while an interrupt is not triggered, i.e. outside of the ISR function, it periodically prints a message to console showing that a key has not been pressed. When a key is pressed, the ISR should be called. When first entering your ISR function, print a message to console showing that the ISR has in fact been called.
2. Your ISR should detect the key pressed and print the corresponding character.

Laboratory Report:

1. Submit your main.cpp and Bus.hpp code in Canvas.
2. Demonstrate to the instructor your interrupt I/O system. Periodically print a message to console showing that a key has not been pressed. When a key is pressed, the ISR should be called. When first entering your ISR function, print a message to console showing that the ISR has in fact been called. Your ISR should detect the key pressed and print the corresponding character.

**Lab 14: LCD Interface**

This lab is for extra credit. The purpose of this lab is to interface an LCD screen to the microprocessor. The student is given the freedom to design their own circuit. Refer to the addendum and to the LCD datasheet before starting your lab.

Prelab:

There is no prelab work for Lab 14.

Laboratory Assignment:

1. Design the circuit to interface the LCD to the microprocessor.
2. Build the LCD interface circuit on your prototype board.
3. Develop a main.cpp that initializes the LCD and writes characters on it.

Laboratory Report:

Demonstrate the operation of your LCD by detecting characters from your keypad and printing them on the LCD screen.

**CMPE-127 LCD Display Interface Lab Addendum**

1. In order to complete this lab, you will need the display controller datasheet. The datasheet will be specific to the display controller of your particular display. However, most display controllers are similar. I attached the datasheet of a typical display controller along with this document.
2. The controller has 3 control pins that are used for initialization and control: RW (W is low active), RS, and E. E is Active High.
3. The controller supports two modes: 4-bit mode and 8-bit mode. The circuit for 4-bit mode is simpler, as shown in the lab manual, however it requires more complex coding. The circuit for 8-bit mode is more complex, however programming an 8-bit mode device is simpler. The choice is yours. If you use the 4-bit mode, you need to use DB4-DB7 as the only data lines and connect DB0-DB3 to ground.
4. The display controller has two registers that are used for programming: 1) Data Register (DR); 2) Instruction Register (IR). The logic level of RS pin determines the register selection as follows:

|  |  |
| --- | --- |
| **RS pin level** | **Selected Register** |
| H | DR |
| L | IR |

1. DDRAM, Display Data RAM, stores display data in 8-bit character codes. Writing to the DR register writes to DDRAM automatically, and the address counter is incremented automatically.
2. RS and RW pin logic levels versus programming operation is as follows:

|  |  |  |
| --- | --- | --- |
| **RS** | **RW** | **Operation** |
| L | L | Write instruction code into IR |
| L | H | Read busy flag (DB7) |
| H | L | Write data into DR |
| H | H | Read data from DR |

1. An instruction code example: Clear Display

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RS** | **RW** | **DB7** | **DB6** | **DB5** | **DB4** | **DB3** | **DB2** | **DB1** | **DB0** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

(Note: In 4-bit mode, DB3, DB2, DB1 and DB0 pins are the same as DB7, DB6, DB1 and DB4, respectively. First DB7-DB4 values are written, then DB3 to DB0 values are written. $-bit mode is a two-cycle write sequence as shown in the timing diagram below.)

1. Write sequence:





1. Initialization: The LCD controller needs to be initialized upon power on. The initialization process is very specific and need to be followed very carefully. See the initialization routine for the 4-bit mode on page 25 of the attached datasheet. The initialization routine for the 8-bit mode is given on page 23.
2. When initializing the controller, first start working with a 1-line display. Select 5x8 dot characters.
3. There are 80 segments (16 characters) on the display. The DDRAM address for each character position is as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Char. Pos. | 1 | 2 |  | 8 | 9 |  |  | 16 |
| DDRAM Addr. | 0x00 | 0x01 |  | 0x07 | 0x08 |  |  | 0x0F |

1. Instruction code to set the DDRAM address:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RS** | **RW** | **DB7** | **DB6** | **DB5** | **DB4** | **DB3** | **DB2** | **DB1** | **DB0** |
| 0 | 0 | 1 | x | x | x | x | x | x | x |

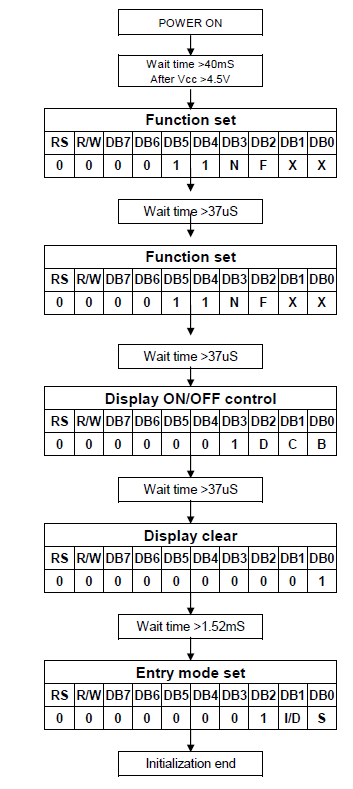
where DB6 to DB0 is the 7-bit DDRAM address.

Instruction code to write data to RAM:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RS** | **RW** | **DB7** | **DB6** | **DB5** | **DB4** | **DB3** | **DB2** | **DB1** | **DB0** |
| 1 | 0 | x | x | x | x | x | x | x | x |

where DB7 to DB0 is the character code to be written at the address (i.e. character location) selected above. Character codes are given starting from page 13 of the attached datasheet.

Initialization:



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Set DDRAM Address to Character Location 1

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Write ‘0’ to DDRAM (i.e. print character)

**Tips:**

Lab 7: If you are experiencing a case in which you are changing the memory address and data values, but reading some past values after make flash, then try the following:

1) Try `make clean`. Make clean will whip out the executables in your local directory.

2) If that does not work try `make purge` then attempt to do `make application` + `make flash`.  make purge will purge all of the pre-compiled files that short cut the build process.