CSCE 312 Lab manual

Lab 4 - Computer Organization and Data Path Design

Instructor: Dr. Yum

Fall 2016

Department of Computer Science & Engineering
Texas A&M University

Chapter 5: Computer Organization and Data path design

Various generic and specialized hardware and software components work with each other to make your computer systems run. Conceptually they are often represented in form of a stack (Fig. 1). Each block or layer is actually an abstraction of the layers below it. This means that a layer at the top depends on the layers below it for its materialization and operation. In this course you are learning about all these components and their operations from bottom to top, starting with logic gates and digital circuits. The later advanced courses will teach you about more complex software layers/components in this stack.

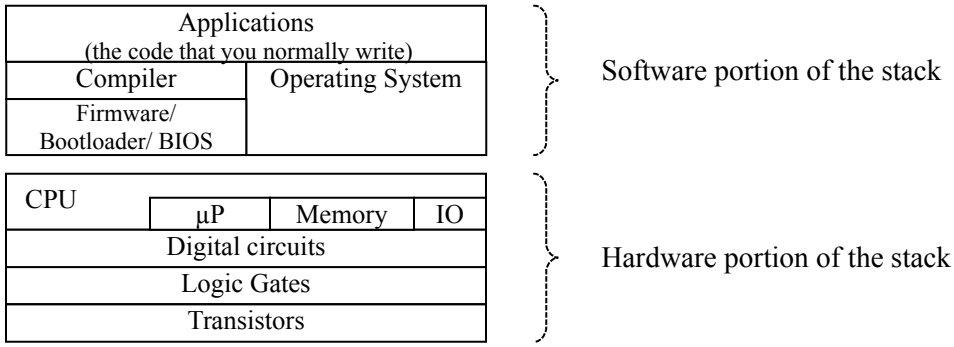

Fig. 1: Hardware and software stack: A conceptual model of the Computer system components.

In computer systems, hardware component designs have been standardized in form of specific design patterns. Data paths and control blocks refer to two of such classes of design patterns which have well defined functions. Inside the CPU, microprocessor (µP) is a generic hardware block which integrates a sub-set of the required data path and control block components. The software that runs inside (or on) it is the customization mechanism that allows us to apply microprocessors to solve a large variety of problems. All this together makes a microprocessor based system a generic digital system solution for a very wide variety of design problems. Sometimes by the term CPU we may mean the microprocessor.

As the course proceeds you will gradually learn about all these basic data path and control block components, their role, and how they interact with each other to finally make programs (in machine language representation) actually execute on the hardware components. In later labs and lecture classes you will also get to know how the machine language code, which can actually execute on the hardware, is generated from the "C" source code.

In this chapter, we focus on designing data path components which sit both outside and inside of the microprocessor chip in a computer system. In the previous labs you have designed IO interface circuitry, address/data/control bus, encoders, decoders, counters, registers, latches, etc. These components sit outside the processor and thus they are generally known as "peripheral components". In industry we use more advanced vocabulary, where the term peripheral denotes more advanced circuitry and functional blocks such as "DMA controller",

"north/south bridge", "memory controller", "programmable interrupt controller (PIC)", etc. All these systems are composed of basic data path components. It is just that these peripherals evolved to become quite complex themselves, therefore they had to be integrated as chips. For time being we will continue to use the term "peripheral" to actually mean the basic components: IO circuitry, bus, etc.

Arithmetic logic units unit (ALU), which is used for integer computation or the floating point unit (FPU), which is used for floating point numbers, are the key data path components that sit inside the processor. Adders, Subtractors, Dividers, Comparators, Shifters, etc. are some components of the ALU/FPU sub-system. In addition to ALU and FPU, there are other data-path components like Register file, Latches, etc., which sit inside the processor chip. Some basic components like Bus, Counters, etc., are often used inside and outside the processor chip. The control circuitry that are present inside the processor chip orchestrates and co-ordinates the operation of the data path. This control circuitry is implemented by sequential logic which you learned in last lab. You will learn more about processor control blocks later.

1. Learning duration: 2 weeks. **Required Tools:** Logisim 2.7.x

2. Group size: This is a group exercise. Ideal group size should be 2 or 3 people.

3. Objective: *To learn* -

Primary topics

1. Application of finite state machine concept to design sequential circuits to generate complex timing requirements.
2. A basic design pattern (Von Neumann architecture) for digital systems.
3. How to design basic IO interface and peripheral circuitry for a microprocessor based digital system.
4. When and how to use memory components (ROM, RAM) in a circuit.
5. How to design basic ALU components – adder, subtractor, comparator, shifters.
6. Role of software as the timing and sequence generation mechanism.
7. Understanding how software will actually work along with the hardware.
8. What are the basic design considerations for a microprocessor based system.

4. Instructions:

1. Form your lab group for this lab and later course project.
2. The group should submit a single common lab report, but each member individually should submit an additional 1 page report to explain what their individual role and contribution in the design assignment was and must critique the group design process.

The format for that 1 page report is given below. For a single person group the individual report is not required.

< Format for the individual 1 pg report >

1) **Your name & Group members:**
2) **List of the parts of the design that you alone did:**
3) **List of the advantages/disadvantages that you perceived in this team based design assignment:**
4) **Which your group member's design was superior and how their designed parts can be improved:**

5. Exercises to do

5.1 Problem 1: Design a sequential circuit to implement the following requirement for the car's blinker system e.g. "….There will be row of LEDs at the back of the car, when the left blinker switch is activated, the LEDs will start glowing in sequence from right to left to manifest a running light effect. The right blinker switch will show the running light effect from left to right (opposite direction). There would be 20 LEDs in the back of the car. Each LED should be on for half second...."

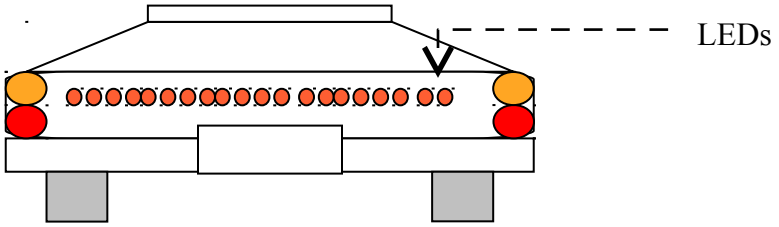

Fig.2: LEDs at the back of the car for the running light effect

Activities to do-

1. Draw the finite state machine representation of the required sequential circuit.
2. Use flip-flops and logic gates to implement the required circuit. The design should have a clearly identifiable data bus.
3. Instead of a decoder, use a ROM and re-implement the same circuit.

Notes:

1. To understand running light effect, see http://www.youtube.com/watch?v=RaEeeaf4ooM
2. Build the required sequencer with a counter implemented with flip-flops.

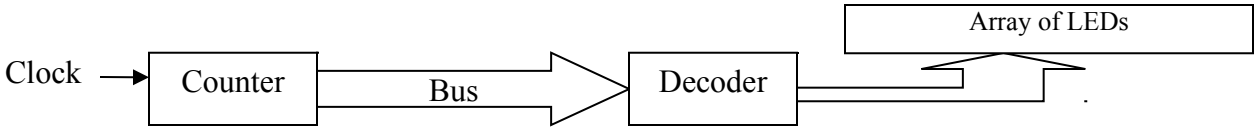

Fig.3a : System architecture of the required circuit designed with a decoder

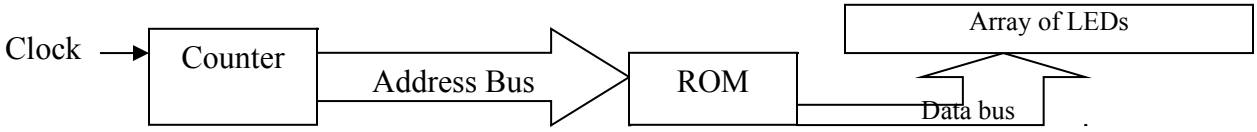

Fig.3b : Alternative system architecture using a ROM

3. To learn about implementing a combinational logic with a ROM, read the section under ROM (Lecture 4), here - http://cs.nyu.edu/~gottlieb/courses/2000s/2000-01-fall/arch/class-notes.html

4. **Do not use the built-in counter or decoder module provided by Logisim**. You may use the built-in adder/subtractor modules if you wish, or you could build the adder/subtractor from basic gates as required by problem-3 first and then re-use them to build your counter. You may use the built-in register module.

5.2 Problem 2: So far you have been implementing various features of the car's control system with combinational and sequential circuits. But for the next generation of the cars, your manager wants you to implement them all with a single small microprocessor based circuit and software code, instead of using a separate hardware logic circuit for each requirement.

***Design rationale*:** This new design paradigm would cut down the design and manufacturing costs and allow design and manufacturing flexibility. Small microprocessors have become very cheap (a few dollars), its cost is comparable to the small scale integration (SSI) logic gate chips. The requirements for which speed is not critical (faster than nanoseconds) can be implemented with microprocessor based systems. One single microprocessor based circuit will be able to do more control jobs, deliver multiple features and thus obviate multiplicity of logic circuits. The other rationale is that any feature can be added later on by simply reloading a new version of the software long after the hardware has been designed and fitted in.

Operation mechanism: The microprocessor will replace the sequential logic which you would traditionally use to generate complex timing and sequential signals (as in Lab 3 and Lab 4, prob # 1). The timing signals will be generated sequentially as each program statement runs sequentially in the processor. To generate a specific timing sequence you have to design the software code appropriately.

***Requirements*:** In addition to "logical I/O" based control requirements as in Lab 1/Chap 2, Prob # 4 and Lab 3/ Chap 4, Prob# 5, your boss has asked to provision the design for "analog I/O". Such a hardware design will be able to cater to many advanced control requirements that may come up in the near future. For example, an intelligent and energy efficient air conditioning (A/C) system will pose such requirements. In this system, there is a need to read the temperatures inside the car, outside the car, the solar insolation and then turn on/off or operate the A/C compressor motor with different speed levels. The microprocessor based system can read temperature sensors which give output in analog form (0 to 5 V to indicate -30 to +70 deg centigrade). The microprocessor based system can also vary the compressor speed by providing a range of analog signal (0 to 5 V).

Activities to do-

1. Implement **a unidirectional 8 bit address and control bus**, and **bidirectional 8 bit data bus**, which are suitable for the 8 bit microprocessor (like Intel 8088).
2. Interface these buses with a **ROM and RAM** as found in Logisim, and verify the combined operation of these busses, ROM and RAM.
3. **Design the IO system circuitry** (an addressable I/O system) for the required microprocessor based system (see the tips below). The design should be modular. At this point of time you don't have to understand in details how the program will actually execute in side the microprocessor. All you have to understand is what signals the microprocessor has to generate to read from the input interface circuitry and to write to the output interface circuitry as part of the control task.
4. **Integrate these busses to the designed IO system**.
5. Provide brief text description, truth table and signal timing diagram to illustrate the operation of each sub-system or modules in your design.
6. The microprocessor will orchestrate the reading, writing from the digital and analog I/O system and the memory. These reading and writing operations are the tasks that have to be executed in the system. Write down the sequence of operations/tasks in- (see tip #13)

    (I) Pseudo-code. (II) Task timing diagram for the following tasks:

    a) Reading from ROM
    b) Reading from RAM
    c) Writing into RAM
    d) Reading from digital input line

e) Writing to a digital output line

7. Transform the pseudo code as designed above to a C code. The code should be setup to run in an infinite loop to repeatedly execute these tasks. (**See tip # 13**). Note: While you write appropriate C language statements, since the hardware does not exist in real life, this code will not be executed on the CS machines. You can model your template similar to the templates we provided for Lab-1.

Important tips:

1. Read Chap 1 of Frank Vahid's book, section 1.4 (only 1.4.1 & 1.4.2) from Byrant's book.
2. Refer the materials provided for Lab 1.
3. Understand what is "logical I/O" and "analog I/O" and their differences.
4. Implement the bidirectional data bus by the tri-state buffers as available in Digital works or by employing MUX blocks. To learn about tri-state logic read the following links –
   http://en.wikipedia.org/wiki/Three-state_logic
   http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/CompOrg/tristate.html
5. The following reference system architecture for a microprocessor based system will give you the needed big picture (Fig.4) to give a head start. You have to design all the components except the CPU, AD bus DMUX, ROM and RAM, which are showed in dotted lines. The block arrows indicate the direction of data flow.

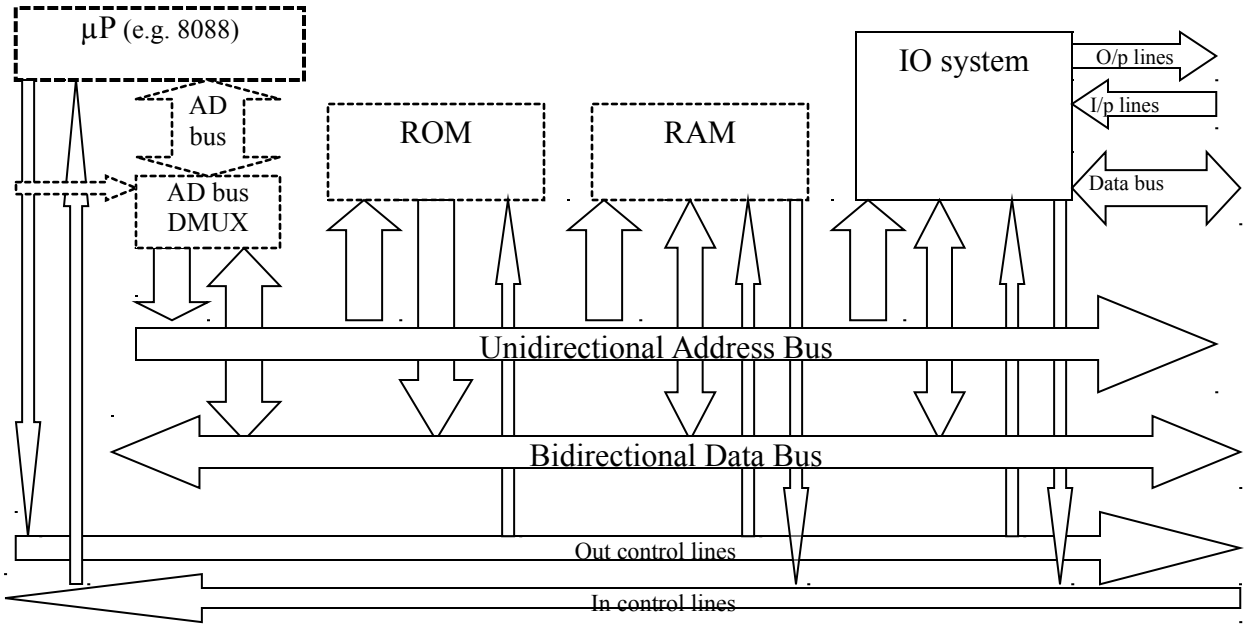

Fig. 4: System architecture for a microprocessor based system

6. A reference design for an addressable IO system is given below (Fig 5). Correlate this with the figures given in 8088 CPU data sheet. Only one logical output and analog input are shown.

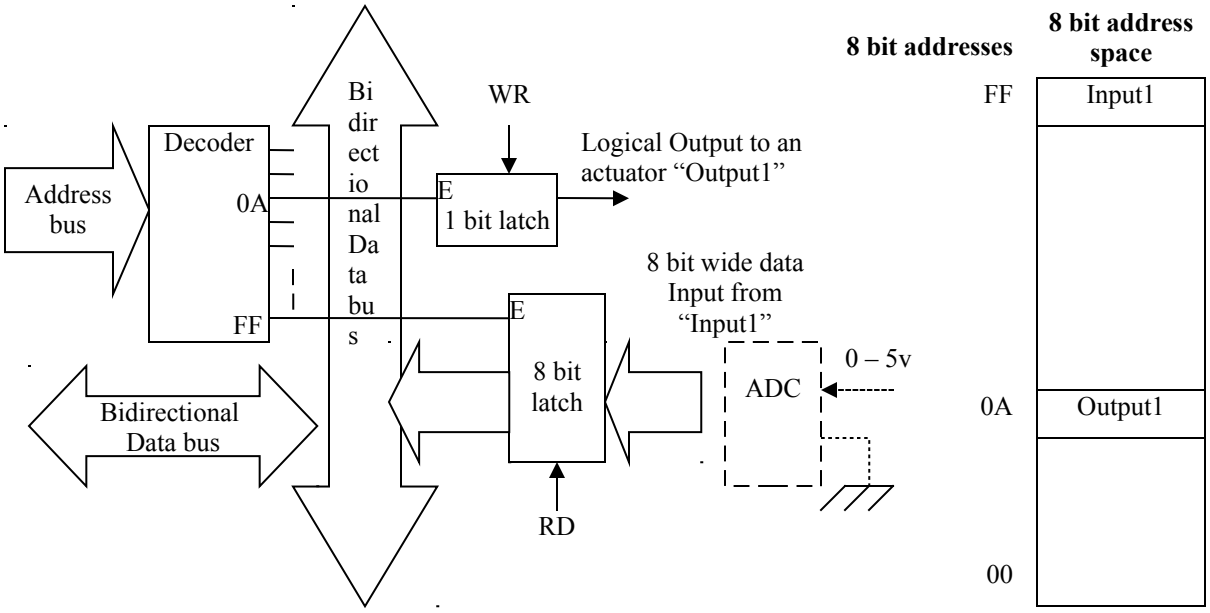

Fig. 5: System architecture for addressable IO and 8 bit address space representation

7. To know about a popular computer organization pattern (Von Neumann architecture), read http://en.wikipedia.org/wiki/Von_Neumann_architecture
8. The actual analog to digital or digital to analog signal conversion is done by analog to digital (ADC) or digital to analog (DAC) converters, which can be interfaced to the microprocessor through the data, address and control bus. Don't bother about how to exactly interface ADC or DAC with a microprocessor at this point, just provide the address and data bus. Don't worry about implementing an ADC or DAC.
9. To know about timing diagrams, bus interfacing and bus protocols, read this -http://esd.cs.ucr.edu/slides/ch6_030702.ppt (**very important**), read the required slides from here - http://www1.cs.columbia.edu/~sedwards/classes/2015/4840/hw-sw-interfaces.pdf
10. To learn how to draw standard timing diagram (either task or digital signals) see the following links. For digital signals – http://en.wikipedia.org/wiki/Digital_Timing_Diagram. For UML interaction diagram see http://en.wikipedia.org/wiki/Interaction_diagram
11. About relationship between "C' code and timing diagram (using hardware as in Fig. 5) –

Example "C" code fragment for the 8 bit system (code for IO permission are not shown) –(This reads in a value *i* through port 0xFF and prints out *i*+1 through 0x0A.)

```
unsigned int i, j;
#define INPORT 0xFF
#define OUTPORT 0x0A
....
for(; ; ) {
      i = inb(INPORT);
      j = i +1;
      outb(j, OUTPORT);
  }
```

*Example task sequence in **low level pseudo** code for the "C" code inside the above loop* –

.....
Task 1: Read from an in port which is addressable with address "FF"
Task 2: Write the value to memory address corresponding to "i"
Task 3: Read from memory location "i"
Task 4: Read from memory location "j"
Task 5: Add the values received from location "i" to the value got from "j"
Task 6: Write the added value to location "j"
Task 7: Read from memory location "j"
Task 8: Write the value got from "j" to an out port which is addressable with 0x0A
.....

Example task timing diagram for first 3 tasks of the pseudo code -

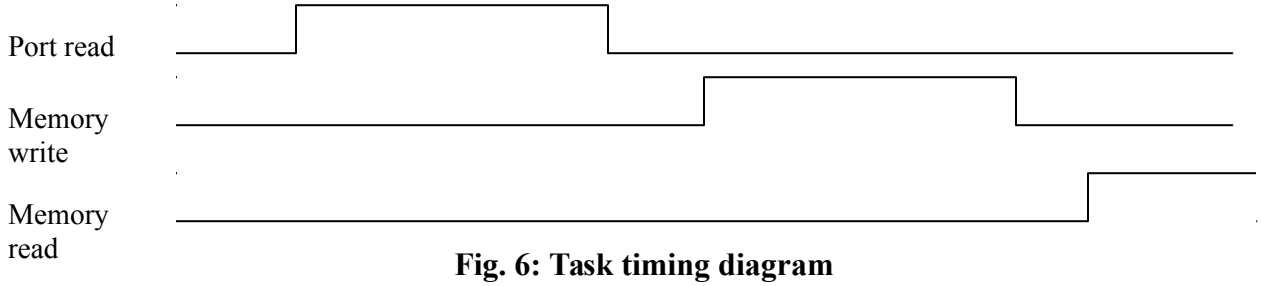

Fig. 6: Task timing diagram

Example digital signal timing diagram for the first task (port read, few signals are shown) -

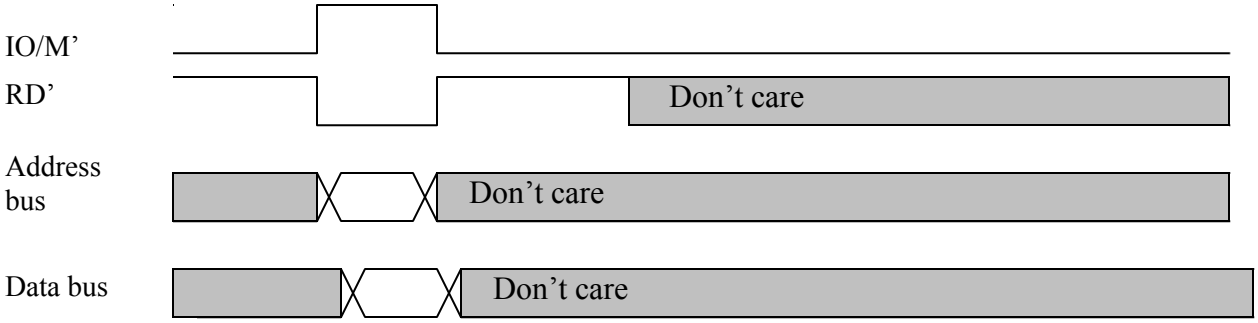

Fig. 7: Signal timing diagram

12. Additional reading materials:

To understand how a program actually gets executed inside the microprocessor:

section 4.3.2 of the Byrant's book (text book).

5.3 Problem 3: *Activities to do-*

Design and verify the following components. Each component should be able to handle 8-bit inputs. (Use basic logic gates for the design. You may use the built in modules for registers, and mux/demux)

i. Adder.
ii. Subtractor (2's compliment)
iii. Magnitude comparator.
iv. Left and right - barrel shifter. (A barrel shifter shifts multiple bits at a time)

as discussed in Frank Vahid's book in Chap 4. These blocks should be designed in way so that they can read data from two (or one) separate 8 bit latched input data buses, and write to a third (or second) latched output data bus. The idea is that you would reuse these designs to implement a small simple microprocessor in future.

Additional material: www-inst.eecs.berkeley.edu/~cs150/fa00/Lectures/04-CombEx.ppt