CSCE 312 Lab manual

Lab 4 - Computer Organization and Data Path Design

Instructor: Prof. Eun Jung Kim

Spring 2024

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 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.

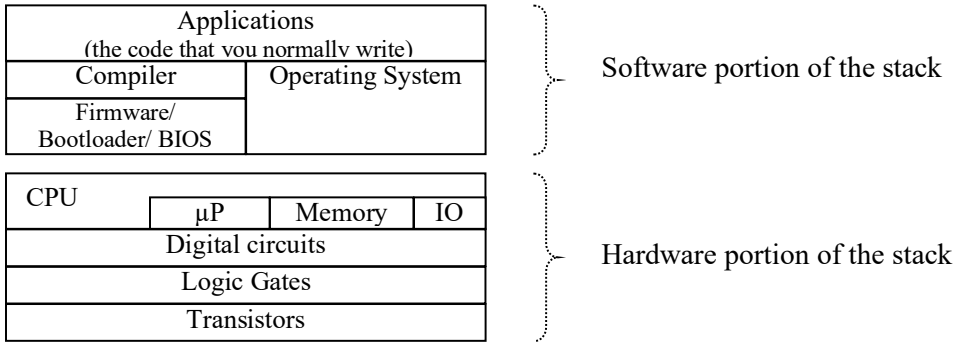

Fig. 1: Hardware and software stack: A conceptual model of the Computer system components.

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) execute on the hardware components. In later labs and lecture classes you will also get to know how the machine language code, which can 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 mean the basic components: IO circuitry, bus, etc.

Arithmetic logic 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 3.8.x

2. Group size: This is a group exercise. Max group size: 3

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 works along with the hardware.
8. The basic design considerations for a microprocessor-based system.

4. Instructions:

1. Form your lab group for this lab.
2. The group should submit a single common lab report along with all Logisim files that you made.
3. 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-page report >

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

Problem 1 (65 points)

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 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). The timing signals will be generated sequentially as each program statement runs sequentially in the processor.

***Requirements*:** In addition to "logical I/O" based control requirements as in Lab 1 Prob #4 and Lab 3 Prob #3, your boss has asked to provision the design for "analog I/O". Such hardware design will be able to cater to many advanced control requirements that may come up soon. 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 5V to indicate -30 to +70 degree C). The microprocessor-based system can also vary the compressor speed by providing a range of analog signal (0 to 5V).

Activities to do

(demo: 50 pts, Report: 15 pts) -

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 buses, ROM, and RAM.
3. **Design the IO system circuitry with 4 input devices(numbered from 00000001 to 00000100) and 4 output devices(numbered from 00000101 to 00001000)** (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 detail how the program will execute inside the microprocessor. All you must understand is what signals the microprocessor must 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 buses to the designed IO system along with a data buffer that stores all data passed via the data bus**.

5. The microprocessor will **orchestrate 5 operations**: read from ROM, read from RAM, write to RAM, read from I/O device, write to I/O device and the control signals for which are numbered from 00000001 to 00000101.
6. Also implement a one-bit **enable input** that controls the whole system and turning it down will not change the values stored in any of the buffers/inputs/outputs/memory elements but will disable the control signals from working.
7. In the report, please give explanation for:
    - How you built each of the individual components of the system.
    - How were the connections amongst all the components made, what issues were faced and how were they tackled.
    - How to use your system to verify its correctness for each of the five requirements.
    - **10 different configurations** of inputs to test your design, Two tests per requirement.
    - **Timing Diagrams** for all the inputs/outputs that are affected when testing the 10 configurations.
8. **Include the Logisim file in your submission folder.**

Extra tips/Resources:

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 4, especially the slides.
3. Understand what "logical I/O" is and "analog I/O" and their differences.
4. Implement the bidirectional data bus by the controlled buffers available in Logisim.
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 must 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.

6. A reference design for an addressable IO system is given below (Fig 7). Correlate this with the figures given in 8088 CPU data sheet.

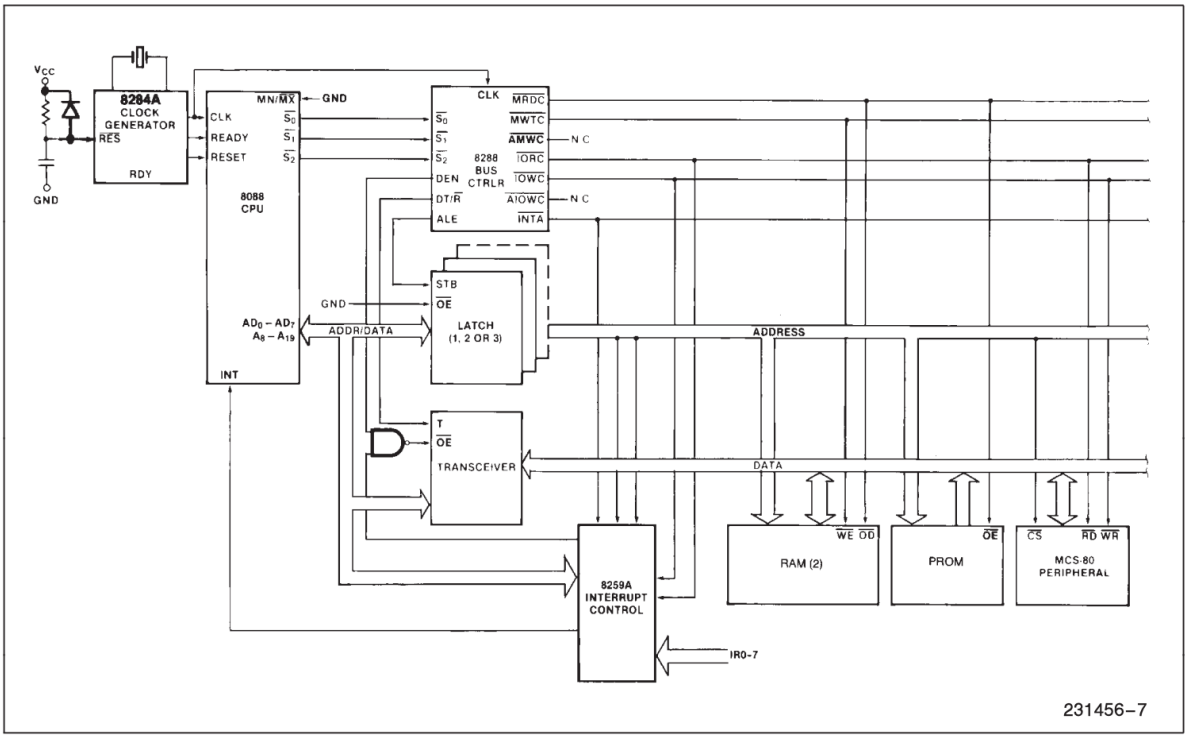

Figure 7. Fully Buffered System Using Bus Controller

7. For a quick tutorial on embedded system organization that will give some idea about how software codes and hardware work together, see here –
   https://www.codrey.com/embedded-systems/embedded-systems-introduction/
   https://www.electronicsforu.com/resources/embedded-systems-overview
8. To know about a popular computer organization pattern (Von Neumann architecture), read

   Von Neumann Architecture
9. 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.
10. To know about timing diagrams, bus interfacing and bus protocols, read this –
    Hardware Software Interfaces
    Data Bus, Address Bus, Control Bus
11. **Additional reading materials:**

    To understand how a program gets executed inside the microprocessor, please read section 4.3.2 of the Byrant's book (textbook).

Problem 2 (30 points)

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, decoders, muxers, demuxers). **Include the Logisim files in your submission.**

i. (10 pts) Adder (with the correct overflow/carry_out bit as output as well).
ii. (10 pts) Subtractor (using 2's complement)
iii. (5 pts) Magnitude comparator.
iv. (5 pts) Left - 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 input data buses and write to a third (or second) output data bus.

In your group report, please provide screenshot of one working example from each component.

Problem 3 (Extra Credit – 20 points)

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 **25 LEDs** in the back of the car. **Each LED should be on for half second**...."

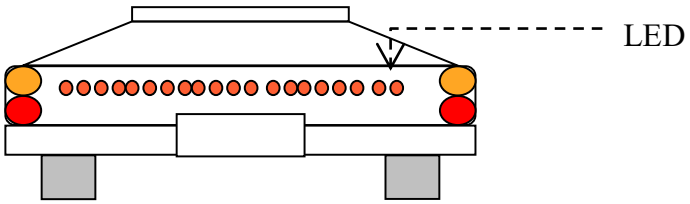

Fig.2: LEDs at the back of the car for the running light effect

Activities to do-

1. (2 pts) Draw the finite state machine representation of the required sequential circuit.
2. (8 pts) Use flip-flops and logic gates to implement the required circuit. The design should have a clearly identifiable data bus.
3. (10 pts) 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.

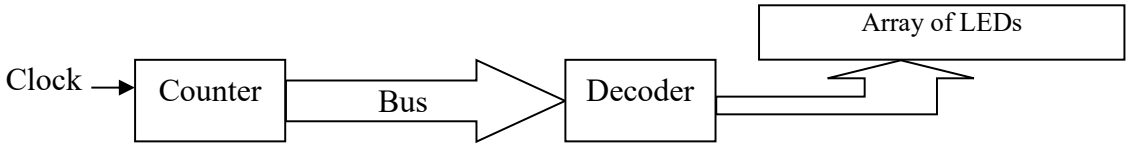

Fig.3a: System architecture of the required circuit designed with a decoder

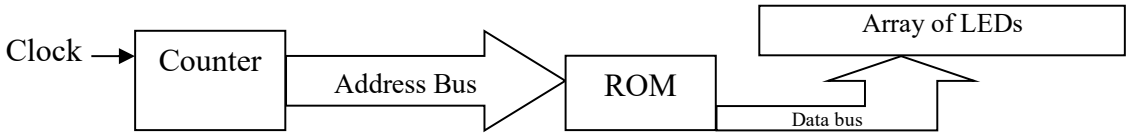

Fig.3b: Alternative system architecture using a ROM

3. To learn about implementing a combinational logic with a ROM, read the section under ROM (Lecture 4), here –
   Link

4. **You may use the built-in adder/subtractor/counter/decoder modules if you wish, or you could build the adder/subtractor from basic gates as required by problem 2 first and then re-use them to build your counter. You may use the built-in register module.**

INDIVIDUAL LAB REPORT : 5 POINTS

SUBMISSION CHECKLIST

Group Submission:

1. Report with:
    a. required explanations and Timing Diagrams for Problem 1
    b. Screenshots for working examples for each part of problem 2
    c. Diagram of Finite State Machine for Problem 3 (if attempted)
    d. Appendix: Stating work distribution amongst team members
2. Logisim Files:
    a. Working file for Problem 1
    b. Working File(s) for Problem 2
    c. Working File(s) for Problem 3 (if attempted)
3. Compress into zipped file as Group<Group_Number>.zip

Individual Submission (1 page submission):

1. Your name and Group Details (Number and Team members)
2. List of Parts of the design you did individually
3. List of advantages/disadvantages/difficulties you faced.
4. Which team member's design impressed you and what did you learn from their help/implementation.
5. Submit as <LastName>_<UIN>.pdf