ECE218 Lab Manual

Electrical and Computer Engineering

TABLE OF CONTENT

TABLE OF CONTENT 2
Lab Introduction 3
Lab 1. Simple Circuits with Debugging Techniques 5
Lab 2. Functionality verification of Gates. 11
Lab 3. Introduction to Digital Circuits 19
Lab 4. Code Conversion 24
Lab 5. Finite State Machines 29
Lab 6. Serial Adder 36
Lab 7. Counters 38
Lab 8. Programmable Logic Arrays 42
Lab 9. Sequential Logic Design with PLDs 50
APPENDIX A. ECE GUIDE TO LABORATORY REPORT WRITING

Lab Introduction

Overview

The purpose of this lab is to apply the material learned in ECE 218 to design and debug digital electronic circuits. To accomplish this goal, you will perform experiments that involve the design and construction of circuits of increasing scope and complexity.

Labs are scheduled for 2 hours and 50 minutes. To make the best use of this time, you must come to lab prepared. Each experiment must be read ahead of time, preliminary questions must be completed and data sheets prepared. It is highly recommended that Schematic be constructed before coming to lab.

Your lab report and preliminary questions can be a mix of word processed text and neatly drawn/written circuit diagrams/equations. Drawings in both the preliminary and final report should be drawn neatly in ink if not computer generated. Component values, pin numbers, signal names, and part numbers must all be clearly shown. More discussion of the report and examples are given below.

Preliminary Work

The assignments to be completed before coming to the lab are:

1. **Reading Assignments** are taken from one of the required texts for ECE 218.
2. **Preliminary Questions** have two functions. First, like homework problems, they are designed to elucidate the theory involved in the current lab exercise. Second, calculations are required that give theoretical values of measurements to be made in the lab. The results of these calculations are usually organized in tabular form.
   The answers to the Preliminary Questions must be neatly done on 8.5x11, unlined, white paper. Sheets torn from a notebook are not acceptable. Unlike homework problems, which are normally handwritten, answers to the Preliminary Questions can be a mix of word processed text and neatly drawn/written circuit diagrams/equations.

Pre-Lab Discussion/Quiz

A quiz, modeled after the Preliminary Questions may be given at the beginning of the pre-lab. Punctuality is required. After the quiz, your instructor and/or your TA will conduct a pre-lab discussion. The answers to the Preliminary Questions will be turned-in at the beginning of the pre-lab. The lab will be conducted in room 311 SH.

Procedure

Instructions for measurements to be taken are given in the Procedure section of the lab assignment. Procedures that relate to explanation of unfamiliar and/or complicated instruments will usually be quite detailed. This type of instruction is referred to as “cookbook instructions”. More often, the procedures will be very brief. For example, the entire instruction may be “Confirm the results of Preliminary Question 1”.

Data Sheet

Before going to the lab, the students should carefully review the Procedures and prepare document called the Data Sheet. The data sheet provides an organized plan for completing the Proce-

dures. A well-organized data sheet will often include a tabulation of the preliminary results with space to record the measured data. This attention to detail will expedite performing the lab and can prevent the discovery after the lab that not enough data was taken or that the data is obviously incorrect. The data sheet should include a list of equipment used including name and model number.

Lab Report

A brief guide to preparing a student experiment report is found on the Appendix A of this lab manual. Your report should follow the outline on page 30. The components of an ECE 218 report include Title Page (include the TA's name), Introduction, Equipment, Results, Conclusion, and Appendix. The data sheet, the circuit lay out, a photo of the prototype, etc., should be put in the appendix.

The pages should be numbered and the document stapled. The report is due at the beginning of the next lab – normally, one week after the lab is completed. The week of an examination in the lecture portion of ECE 218 is an exception to this due date. The exceptions are noted in the lab syllabus.

If you have a lab partner, you may each turn-in either an individual lab report or a group lab report. If you elect a group Lab Report, a statement on the title page, signed by each group member, must attest **"that every member of the group has materially contributed to the intellectual content of this report"**. This means that one student is not allowed to "do the work" while the other student's contribution is to type or otherwise prepare a neat copy.

Credit Distribution

Component	*Credit*	*Credit*
Preliminary Questions	20 %	10 %
Schematic/Breadboard layout	N/A	10 %
Data Sheet	10 %	10 %
Correctly completing the lab	35 %	35 %
Lab Report	35 %	35 %
	100 %	100 %

Lab 1. Simple Circuits with Debugging Techniques.

Objective

In this experiment, you will be implementing and understanding simple circuits involving basic components like resistors, LEDs, switches and capacitors. You will also learn different trouble-shooting techniques which involves use of a multimeter.

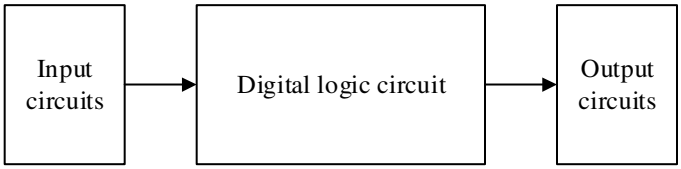
Keywords:

Multimeter, Resistors, LED, Single Pole Single Throw Switch, Resistor array, Digital Logic circuits, Open Circuit, Kirchhoff's Voltage Law, Ohm's Law and closed circuit, I-V Characteristics of LED.

Components: Handheld Multimeter, Single 1kΩ Resistors, 1 kΩ Resistor Array, DIP SPST Switch (containing 4 switches) and a 1µF Ceramic Capacitor.

Background

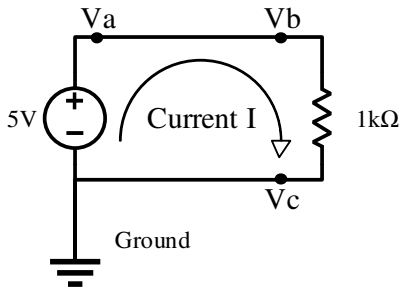
All the circuits implemented in this course can be generalized into a simple block diagram as shown below. Each digital circuit is characterized by inputs and outputs, with sometimes a dedicated circuit, and the actual circuit.



In this lab we will be focusing on the different input and output circuits which usually consists of resistors, switches or LEDs.

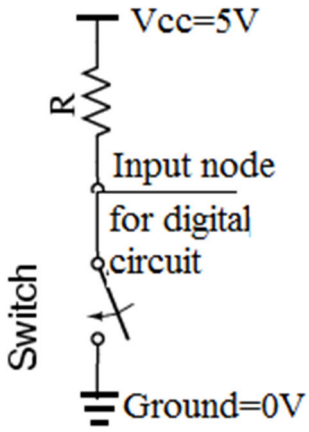
Input Circuit:

Consider the simple circuit shown below and calculate the potentials at the points Va, Vb and Vc with respect to ground and also determine the current I circulating in the circuit. What is the difference of potential between Va and Vb?

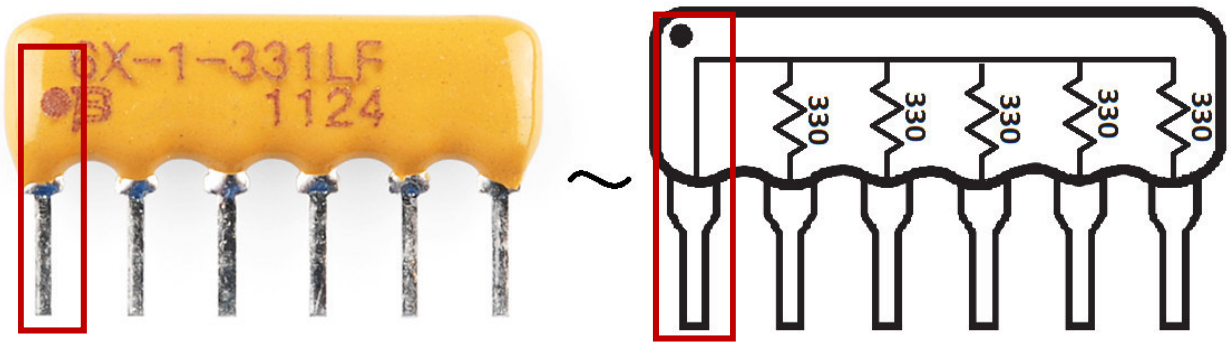


With respect to ground, since Vc is directly connected, we have Vc=0V. Va and Vb being on the same wire, we know that Va=Vb. They are also connected to the positive output of the power supply, so Va and Vb potential with respect to the ground is the potential at the negative output of the power supply, plus the voltage it produces. In this case, we get Va=Vb=5+0=5V. Using Ohm's law using potentials Vb and Vc, we can determine the current circulating in the loop: I=U⁄R=5⁄1000=5mA. The difference of potential between Va and Vb is 0 since the two nodes are directly connected. However, this is an ideal representation. In reality, wires act as very small resistances, and a small voltage drop can be expected.

This circuit can be broken up to accommodate a switch to provides a digital input to a circuit. In this case, if the switch is opened, the output will be 5V, which represent a logic high, or 1. If the switch is closed, the output is directly connected to the ground, 0V. This represents a logic low, or 0. This is the kind of circuit you will use to generate input signals for the digital circuits later on.



Most of the digital circuits use actually more than one input, and with many inputs, many resistors may be used. In order to reduce the space taken by the resistors, they are sometimes packed in a single component with several resistors in it, as shown below with 330Ω resistors. The red boxes indicate a common node for all the integrated resistors. If we take the previous schematic, the common node can be connected to the positive voltage supply (Vcc), and each individual terminal can be connected to a separate switch.

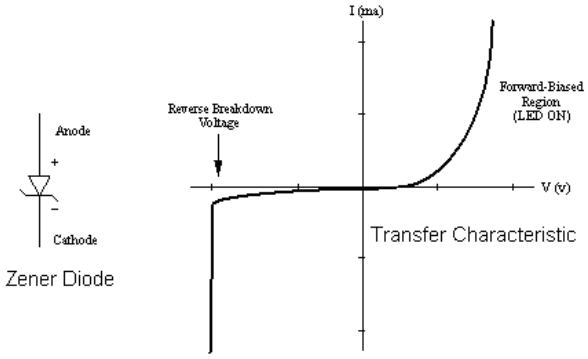


The voltages used in the input of the digital circuit can be generated in a multitude of ways. For instance, a pattern generator is a device that generates several inputs signals for the digital circuit, in sequence. The lab is equipped with a device called “Analog Discovery” connected to the computer. It is a multi-tool, and includes a pattern generator.

Output Circuit:

A very efficient way to visualize the outputs of a digital circuit is to use LEDs. The LEDs provide a very visual output and is usually very compact and convenient. But before going further, a little bit of theory is required.

Light-emitting diodes (LEDs) are semiconductor devices that have electrical characteristics similar to common diodes, but they also emit light when properly connected. Like the common diode, the LED has two terminals: an anode and a cathode. Figure below shows the schematic symbol for a LED and the LED's voltage-current characteristics.



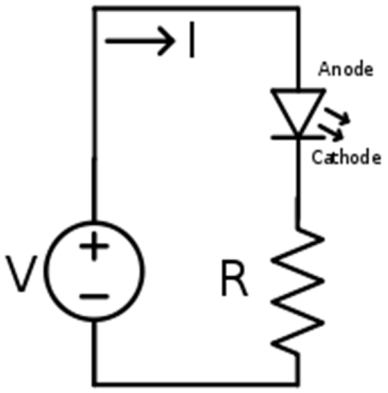
During normal operation, a diode acts as a "one-way" conducting device - when a positive voltage is placed between the anode and cathode, a positive current flows through the diode. This is known as the forward biased state. When a zero or small negative voltage is placed between the anode and cathode, no current is conducted. This is known as the off state. When a negative voltage exceeds a value known as the reverse breakdown voltage, the diode is reversed biased and conducts current in the opposite direction

LEDs are available in a variety of colors, including red, green, yellow, and blue, and are also available in the infrared range (infrared LEDs are often used in TV and VCR remote control

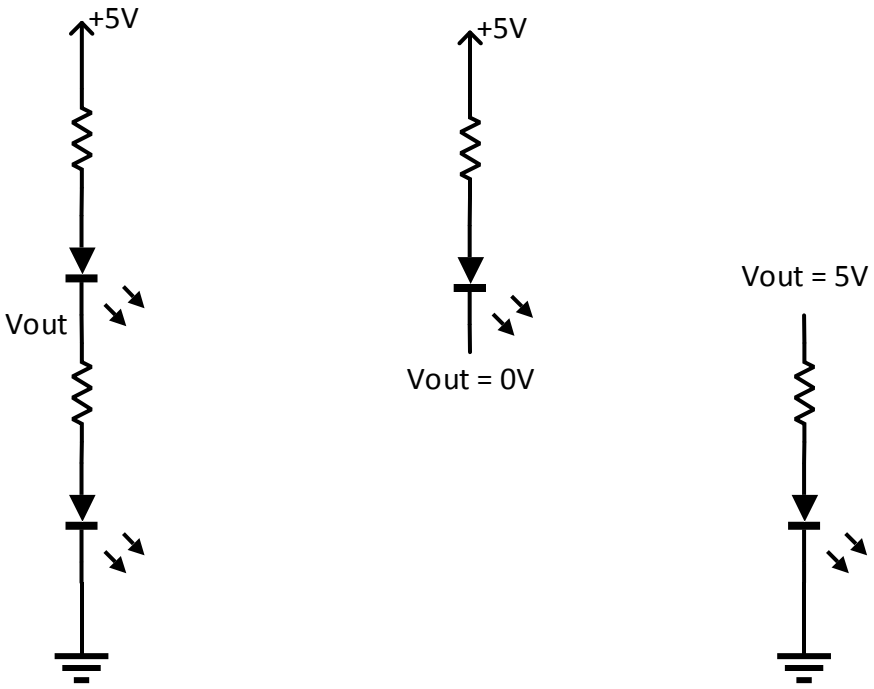
units). They are manufactured in a number of different packages, including round lamps, "bar graph" displays, and seven segment displays, which are used to display numerical values.

In the following schematic, we use a power supply to produce a voltage that we increase slowly from 0V to 5V. One thing that can be notice is that the LED will not appear to be lit until a certain voltage is reached. To understand more about LEDs, go the link given in **Reference for keywords section**.

The voltage at which the LED starts glowing is called Threshold Voltage Vth. The current flowing through the resistor R and LED can be calculated by using the formula I=(V-Vth)/R. This formula can be obtained by using Kirchhoff's Voltage law and Ohm's law. The threshold voltage of an LED is usually around 1.6V to 2.1V, but it is appropriate to refer to the datasheet of the LED to know exactly that value. However, the light emitted by the LED is directly related to the current going through it. A standard LED can be correctly lit at 10mA, but can be visible at currents as low as 1mA, or even less. Once again, the datasheet provides the optimal current. The resistance introduced in the schematic is to limit the current going through the LED.



The next circuit shows two LEDs, and an input signal Vout. When Vout is 0V, the voltage for the bottom part of the circuit is 0V, so no current is circulating, and the LED stays dark. However, the voltage is of 5V for the upper part of the circuit, lighting up the LED. On the contrary, if Vout is 5V, the voltage for the upper part of the circuit is reduced to 0V. However, the voltage for the bottom part of the circuit is 5V, and so the LED lights up. It is to be noted that the current flows in and out of the node Vout.



When connected to the output of a digital circuit, this can clearly display the output state, 1 or 0, represented by 5V or 0V. When the output is 0, the upper LED will be lit, and the lower one if the output is 1. The circuit can be restricted to a single LED to save components.

For this lab, you can wire the Vout directly to +5V or GND to see if the LEDs light up.

Additional Reading(Mandatory):

Chapter 1 from Guide to Design: This will help you in using desktop multimeter and Handheld multimeter.

Preliminary Assignment

1. For the input circuit, what is the voltage at the node mentioned as "input node" when the switch is open and the supply voltage is 10V instead of 5V?
2. With the same voltage of 10V, for the input circuit, what is the voltage at the node mentioned as "input node" when the switch is closed?
3. What is the current flowing through the resistor R=1kΩ in the input circuit when the supply voltage is 5V (give the answer for both switch open and switch closed condition)?

In the lab

1. Implement the input circuit with the resistor array and switch. (Refer to the guide or presentation to implement the circuit on breadboard/Remember Guide has a simple resistor circuit. So plan in advance by preparing schematic and grid layout for input circuit).

2. Using handheld multimeter and desktop multimeter measure the voltage when the switch is open and closed. (Guide or presentation can be used as reference to learn handheld multimeter and desktop multimeter).
3. Implement the output circuit, using a green LED for the top, and a red LED for the bottom. Using the handheld multimeter, measure the voltages at the LEDs and resistors poles, both with the input to ground and to 5V. Record what LED is lit when the input is connected to either ground of Vcc, and calculate the current going through the LED using the voltage measurement at the resitors.

Reference for keywords:

Multimeter: https://learn.sparkfun.com/tutorials/how-to-use-a-multimeter
Waveform\signal Generator: https://en.wikipedia.org/wiki/Signal_generator
LEDs: https://en.wikipedia.org/wiki/Light-emitting_diode
SPST switch: http://www.learningaboutelectronics.com/Articles/What-is-a-single-pole-single-throw-switch-SPST
Resistor Array: https://www.youtube.com/watch?v=_DJ_xLOxnpE
Digital Logic Circuits (Digital Circuits):
https://en.wikibooks.org/wiki/Digital_Circuits/Logic_Operations
Kirchhoff's Voltage Law: https://en.wikipedia.org/wiki/Kirchhoff%27s_circuit_laws
Ohm's Law: https://en.wikipedia.org/wiki/Ohm%27s_law
Open and Closed Circuits: http://science.howstuffworks.com/environmental/energy/circuit2.htm
I-V Characteristics of Diode: http://www.electronics-tutorials.ws/diode/diode_3.html

Lab 2. Functionality verification of Gates.

Objective

In this experiment, you will get acquainted with the implementation of Basic Digital Circuits and verify the functionality of the gates.

Keywords:

Boolean Algebra, Binary numbers, Chips (Integrated Circuits), Datasheets and logic analyzer

Components used in this experiment: 74LS08 (TTL AND gate), 74LS04 (TTL NOT gates) and Analog Discovery (Logic Analyzer).

Other components include: 1 kΩ resistor array, 4-SPST switches.

Background

Modern computers function on the principles of Boolean algebra. Boolean algebra involves computation using only two states: 0 and 1. In order to be used, these states needs to be linked to a physical value, that can be measured and reproduced. One of the early standards to encode these states in digital circuits is TTL, Transistor-to-Transistor Logic. It represents the states as voltages, with 0 being 0V and 1 being 5V. In reality, there is a certain tolerance on the voltage defining 0 or 1, and a threshold voltage is usually provided to characterize that tolerance.

Every digital circuit is based on the same basic building blocks, or gates: AND, OR, NOT, XOR, NAND, NOR, XNOR. This translates into available Integrated Circuits (ICs), aka chips, implementing different functions. For the basic functions, the chips are usually following a standard, which is the case for the 74LS family that you are using in this lab, but can be produced by many different companies. Each company provides a datasheet defining the function of the chip, the pin layout, and also its electrical characteristics.

We are now going to present the AND gate as an example. The AND gate is a Boolean function that returns 1 only if all its inputs are 1, otherwise its output is 0. For our explanation we are taking a two input AND gate. The following truth table list the possible input combinations and the resulting output:

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

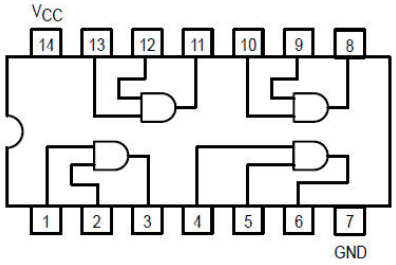
Binary logic in terms of electronic circuit.

Input 1	Input2	Output
0V	0V	0V
0V	5V	0V
5V	0V	0V
5V	5V	5V

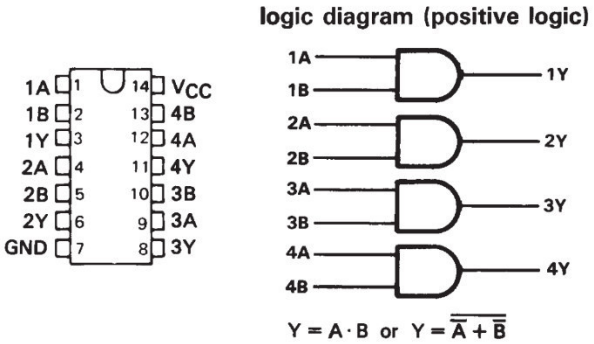
The electronic behind the implementation of the AND gate is very similar to the use of switches. The transistors are actually used as electronically controlled switches, but without mechanical part. They are also very small, allowing to implement a lot of them in one single chip (billions in current computers). The AND chips we are using in the lab, the 74LS08 includes for example 4 AND chips.

For TTL logic, a convention on the reference number exists. The 74 refers to the standard TTL family. The following number defines the function of the chip, for example here, 08 refers to the AND gates. The letters in between indicate special characteristics of the chip: L is low power, H is high speed, S is high-speed with Schottky diodes, and so on. You can also find letters before the 74, which indicates the funder that produced that chip specifically. In this case, the SN74LS08 is produced by Texas Instrument.

Now let's have a brief glance into the datasheet of the AND gate chip:



The pin layout indications vary from one vendor to another. The above figure shows the four AND gets contained in the chip by overlaying the gates on the chip view from top. In the datasheet from Texas Instrument however, the ship is shown with the pin names, and a second figure shows the schematic with the gates and the corresponding pin names.



The tolerance on the input voltage, as well as on the power supply, or the maximum current that can be drowned from the outputs are all defined in the "recommended operating conditions" of the datasheet, as shown below:

recommended operating conditions

	SN5406			SN7408			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
V_{CC} Supply voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH} High-level input voltage	2			2			V
V_{IL} Low-level input voltage			0.8			0.8	V
I_{OH} High-level output current			−0.8			−0.8	mA
I_{OL} Low-level output current			16			16	mA
T_A Operating free-air temperature	−55		125	0		70	°C

The datasheet includes much more information, such as switching delay. Switching delay is the time that it takes for a change on an input to affect the output. This information is really important when implementing circuit that requires certain time constraints. To get this information about the delay refer to the "switching characteristics" section of the datasheet. For more information on "how to read datasheet?" refer to the link given under the Datasheet in the **Reference for keywords** section.

P.S. To find the datasheet go to http://www.alldatasheet.com/ type part name/ number in the search box and select the company which supply the IC. For this experiment the IC's which we use belong to Texas Instruments.

We are now going to use the circuits we saw in the previous lab, and apply them to test the functioning of the AND gate.

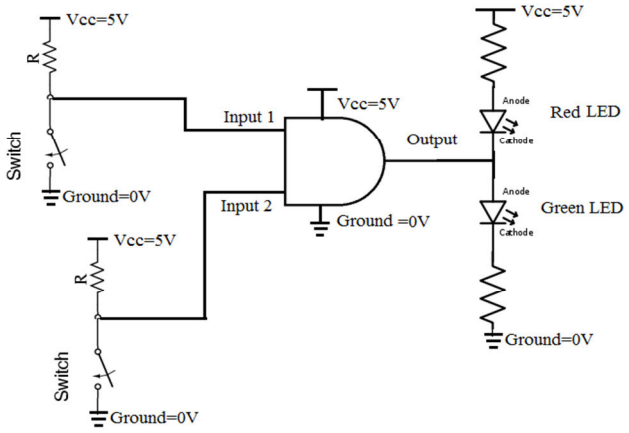
For the input circuit, we are going to use a resistor and a switch for each input, and for the output, we are going to use two LEDs and two resistors.

Connection of VCC and Ground to the AND gate:

The Vcc and Ground pins of any IC needs to be connected to a power supply to ensure correct operations. The datasheet we saw earlier defined that the Vcc pin needs to receive between 4.75V and 5.25V in relation to ground (so we assume that the ground is the ground of the power

supply, 0V). In this lab, the power supply on your bench should be set to 5V. Ask your lab instructor how to properly set the power supply if you are not sure how to.

The overall schematic of the circuit looks as shown below:



Grid Layout

Grid layout diagram show the connections between the IC pins and other components. Use heavy black lines to represent jumpers between adjacent pins and colored lines to represent long connections. The connection to +5V must be shown in red and the ground connection shown in black. Show "hair-pin loops" where the input and the output cables are to be connected. Remember, each cable requires two hair-pin loops; one for the red lead and one for the black lead. It is not good practice to connect two black mini-hook gripper leads to one hair-pin. A sample of Grid layout with schematic equivalent is given in Appendix

Logic analyzer for functionality verification:

Background on Logic Analyzer:

You may already be familiar with the use of an oscilloscope as a tool for displaying voltage versus time. The oscilloscope can be used as a tool for logic circuits as well as analog circuits since different voltage ranges represent different logic values. The biggest drawback of doing this is that oscilloscopes have a limited number of channels (in our case two). In complex logic circuits, we may want to observer several signals simultaneously.

Logic analyzers alleviate this problem. They display logic values (i.e. 1 and 0) that represent the value of digitals versus time. Logic analyzers can display of several these signals simultaneously, making them ideal for debugging complex circuits. The Digilent Analog Discovery can display up to 16 signals simultaneously (high-end logic analyzers can display many more). Logic analyzers are no substitute for oscilloscopes when debugging tricky analog problems because they can display only logic 1 and 0 values. However, they excel when you are attempting to debug complex sequential circuits.

Logic analyzers operate by repeatedly sampling data inputs and temporarily storing them while searching for a trigger condition. If no trigger condition is detected, stored values are overwritten

by new values as they sampled. However, if a trigger condition is detected, then the stored data is shown on the display so that the user can see the values of the signals before, during, and after the trigger condition. In the Digilent Analog Discovery, these values are transferred to the computer that will display the waveforms. The software allows to display the status of each individual channel of course, but also allows to combine different channels in "buses".

A major advantage of logic analyzers is that they can collect this data at high speed, making it possible to test circuits at full clock speed where errors are most likely to occur.

Logic analyzers are extremely useful when debugging sequential circuits, which may contain many signals, which change at different times. Examining these signals with a logic analyzer allows you to see if the circuit is behaving properly and, if not, to isolate the source of the problem. Unlike an oscilloscope, which triggers only on a single signal, logic analyzers can trigger only arbitrary patterns of multiple inputs, making it possible to specify exactly the sequence of events that you wish to examine.

Figure 2.1 shows a picture of the Digilent Analog Discovery device. Figure 2.2 shows the functionality of all the pins of the front connector. For this lab, we are going to focus on the Digital I/O Signals and the ground.

Figure 2.1: Digilent Analog Discovery Device

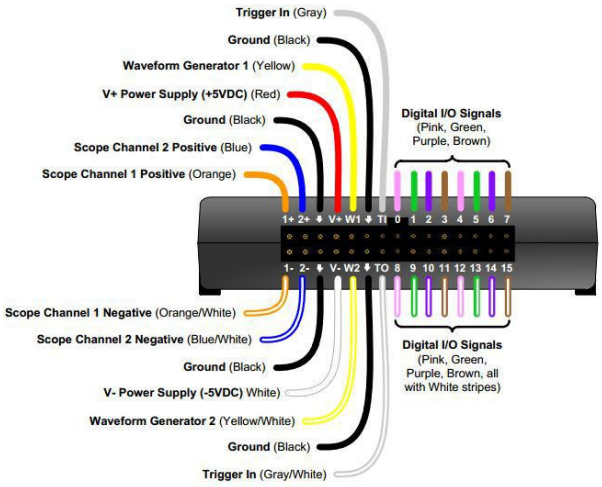


Figure 2.2: Digilent Analog Discovery Connections

To connect a probe to a test point in your circuit, first turn off the power. Then, connect the wire to the circuit in one of the following way: Cut and strip the ends of a short piece of wire. Plug one end into the breadboard at your test point, and the other end to the wire coming from the digital analyzer. It is important to tum off the power so that you do not short out your circuit when making the connections.

The Analog Discovery device is to be used with the WaveForms software from Digilent. The device includes the functionality of different lab devices such as oscilloscope, signal generator, power supply, logic analyzer and pattern generator. Figure 2.3 shows the WaveForms selection

screen corresponding to the different functions of the device. In this lab we are only interested in the logic analyzer function.

Figure 2.3. WaveForms Selection Window

Figure 2.4 shows the main window for the WaveForms logic analyzer function.

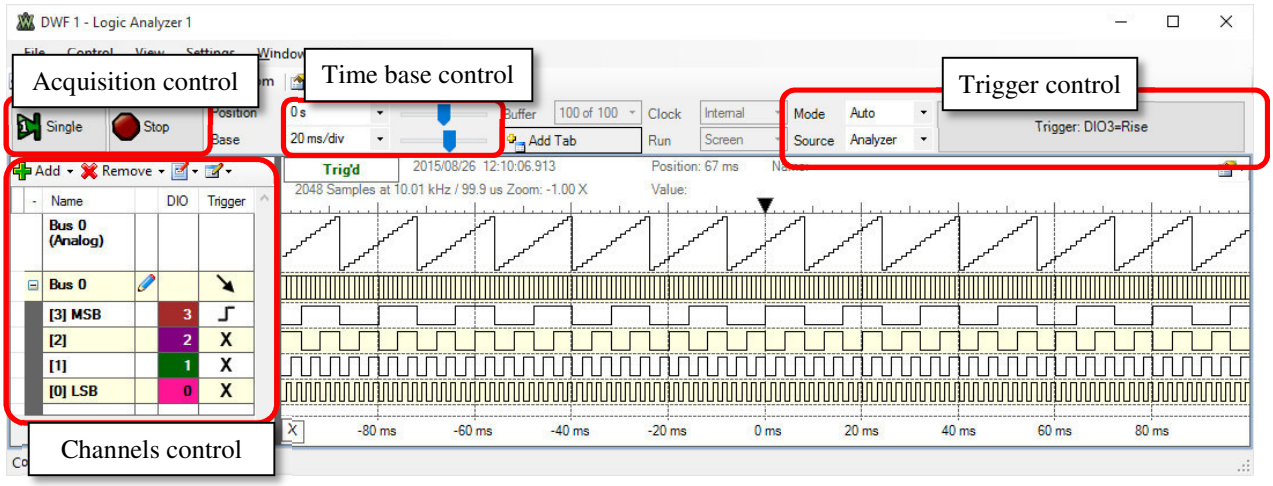


Figure 2.4. WaveForms Logic Analyzer Window

Any logic analyzer implements a set of functionalities that are similar to those of an oscilloscope:

- **Channel Selection:** It allows to select the different channels to be displayed. Some logic analyzers such as this one allows you to group individual channels into buses.
- **Time Base:** It controls the sampling rate. The standard unit for oscilloscope is the time per division.

- **Acquisition Control:** It allows you to start/stop the acquisition and also to start a "single shot" that is triggered once the trigger conditions are met and acquire the signals for a predetermined time.
- **Trigger Selection:** The trigger selection allows you to select on what event the acquisition of the signals should start. In the case of a logic analyzer, for each logic channel, several options are available:
    - **Rising Edge:** In this mode, the acquisition is triggered when the signal switches from the state 0 to state 1.
    - **Falling Edge:** In this mode, the acquisition is triggered when the signal switches from the state 1 to state 0.
    - **=1:** If a channel is selected to be =1, the trigger will happen only when this channel is in the state 1.
    - **=0:** If a channel is selected to be =0, the trigger will happen only when this channel is in the state 0.
    - X: In this configuration, the channel is ignored for the trigger

At least one channel, and only one, is to be configured as rising edge or falling edge. The trigger has two modes: in Auto mode, if the acquisition is not triggered after a certain time, the acquisition is automatically triggered. In Normal mode, the acquisition happens only on a trigger event.

In this lab you will be verify the function of AND gate using the switch-resistor input and resistor led output and you will also be verifying functionality of NOT gate using logic analyzer.

Preliminary Assignment

1. Write the truth tables for the NOT and OR gates.
2. What is the maximum low level input voltage for the NOT gate in the chip SN74LS04?
3. What is the minimum high level input voltage for the OR gate in the chip SN74LS32?
4. What is the allowed range of the supply voltage for the AND, OR and NOT gates from Texas Instrument (SN74LS08, SN74LS32, SN74LS04)?
5. What is the minimum and nominal output high voltage for all the three gates mentioned in preceding question (V_{OH} for Vcc=Min, I_{OH}=-0.8 mA and V_{IH}=2V; Refer the datasheet of all three gates)?
6. What is the maximum frequency of a square wave that can be provided to the NOT gate (see T_{plh} and T_{phl} values given in datasheet)?

In the lab

1. Implement the AND circuit. (See the Guide for breadboard implementation)
2. Record the output voltages and the led that glows for all the different input combinations.
3. Use the waveform generator and logic analyzer to verify the functionality of the NOT gate. Provide a square wave to the circuit using the waveform generator and observe both the input and the output. Make a screenshot of the logic analyzer window showing both

input and output. (Refer the guide to work on logic analyzer and waveform generator). Try to vary and frequency and note the changes in the logic analyzer.

Reference for keywords:

Boolean algebra and Binary numbers: http://www.i-programmer.info/babbages-bag/235-logic-logic-everything-is-logic.html

Chips/Integrated Circuits: https://learn.sparkfun.com/tutorials/integrated-circuits

Datasheet: https://www.sparkfun.com/tutorials/223

Lab 3. Digital Circuits

Objective

The object of this experiment is to measure the glitches caused by the delay circuit and to learn the transfer characteristics of the Inverter using Desktop oscilloscope.

Keywords: Transfer Characteristics, Logic families and TTL logic families.

Components used in this lab: 74LS04(NOT gate), waveform generator, oscilloscope and logic analyzer.

Background

Propagation Delay

When the input to a logic inverter make a transition from a H → L, the output goes from a L → H. And vice-versa. The output transitions are delayed compared to the input transitions. This delay is called propagation delay. Page two gives these delay times to be about 10ns. Figure 3.1 shows the output response of an inverter to an input pulse. Note that neither the input nor the output are "perfect" square waves. Both take a small amount of time to rise or fall. t_r and t_f are called the rise-time and the fall-time. Since it is hard to tell exactly when the pulse start to rise and exactly when it gets to its final value, the rise time, t_r, is defined as the times it takes to get from 10% of the initial value to 90% of the final value. The fall time is similarly defined. A visual definition of the turn-on delay and turn-off delay given on page two of the spec sheet (AC Characteristics) are shown pictorially n in Figure 3.1Figure. The propagation delay is usually defined as the average of the turn-on delay and the turn-off delay.

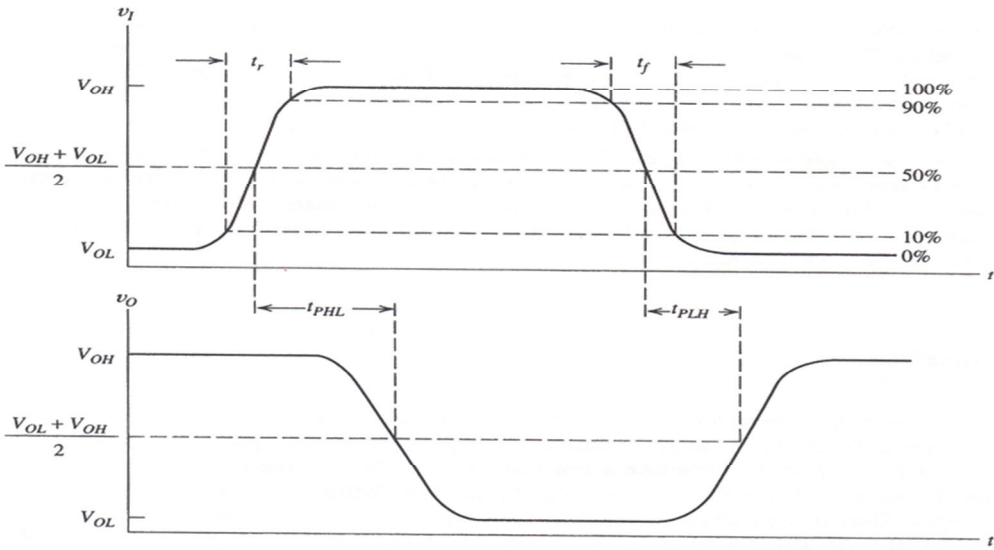


Figure 3.1: Input pulse and output of a typical inverter

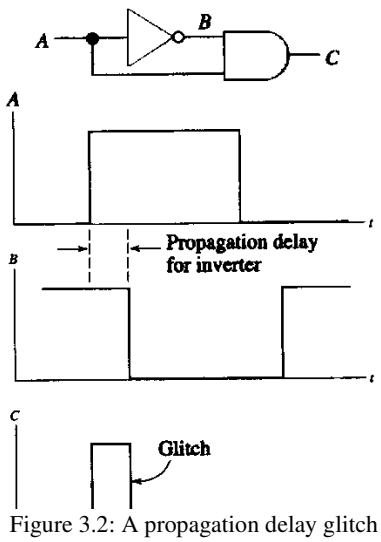


Figure 3.2: A propagation delay glitch

Glitches

Propagation delays can cause "glitches" in logic circuits as shown in Figure 3.2. In this figure we have assumed zero delay in the AND circuit and zero rise and fall times. Combinatorial

logic leads us to expect the output of the AND circuit would be LOW regardless of the input. However, when the input switches from a LOW to a HIGH, the change get to the two inputs of the AND circuit at different times. The bottom input of the AND gate "sees" the change immediately but the change to the top input of the AND gate is delayed by the propagation delay. The short pulse at the output of the AND gate has width equal to the propagation delay. The situation depicted in Figure 3.2 is highly idealized. Some experienced (and clever) designers use glitches as part of a "clever" design, More often than not, glitches are unexpected and cause problematic results. Poor (sloppy) wiring can also cause glitches.

Power Supply Filtering

In the experiment dealing with op-amps, you saw that unexpected circuit behavior sometimes results from glitches on the power supply busses. Good wiring practice demands you filter the power supply buss. TTL handbooks recommend that you use a $0.1\mu F$ ceramic capacitor. It is also a good idea to use a $47\mu F$ electrolytic in parallel with the ceramic to reduce the voltage variations from the power supply. Use that combination here and in all future prototypes.

Vin/Vout (Transfer) Characteristics of Inverter:

A plot of input vs output voltage characteristics of NOT gate/ Inverter gives the Input output characteristics. It is known that the Inverter gives an output of 0V when the input is 5V and vice versa but what happens if intermediate voltages are given to the inverter? This kind of question can be answered by the Vin/Vout Characteristics of inverter. This curve is also used in determining, the electrical characteristics of Input and Output voltage ranges given in the datasheet. This transfer characteristics vary for different logic families.

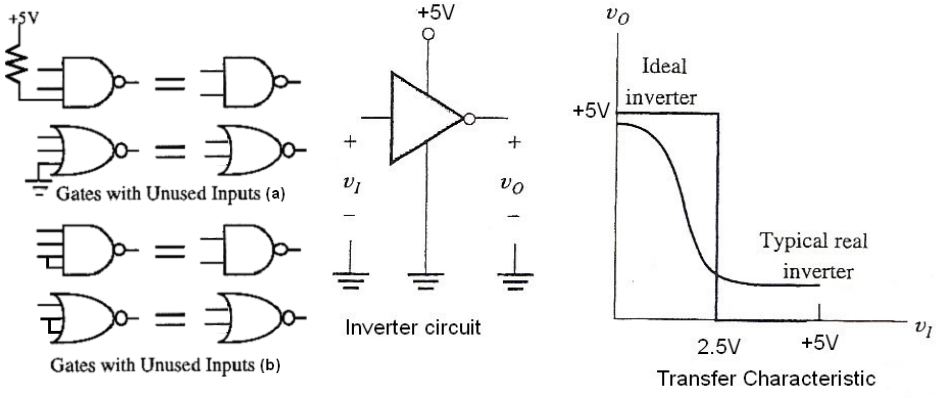


Figure 3.3: Connecting unused gates/inputs

Unused Inputs

Unused inputs should never be left floating because unconnected inputs are susceptible to pick-up noise. Figure 3.3 shows the recommended method of dealing with unused gates and/or unused inputs.

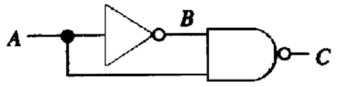
Sometimes you may have a gate with more inputs than you need. Figures (a) and (b) both effectively reduce the number of inputs by one. The equivalent reduced-number-of input gates in connection (a) would be preferred when fan out is important in the design, for example when switching speed is important. Connection (b) is probably easier to wire because the inputs are probably on adjacent pins. However,

with this connection, whatever is connected to the doubled-up input is driving two inputs. Having FO = 2 may or may not be important.

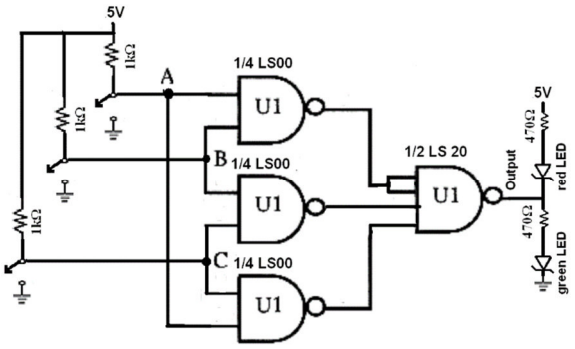
Sometimes you don't need all the gates on a multi-gate chip. You cannot just leave the inputs of the unused gates floating. Figure (c) shows a method that draws the least amount of current from the supply. Figure (d) can be used but draws much more current from the supply. Page 2 of the data sheet gives two specs:

- $I_{IH} = 20\mu A$. This is the current that flows into an input that is held HIGH. Thus in (c), the total current is $3 \times 20 = 60\mu A$.
- $I_{IL} = -0.4mA$. This is the current sunk by an input that is held LOW. Thus in (d), the total current is $3 \times 0.4 = 1.2mA$.

Preliminary Assignment
Answer to the following questions:



1. The glitch described in the Introduction's Figure 3.2 is very narrow. We will daisy-chain 3 inverters to get a wider glitch. Since 3 is an odd number, 3 inverters daisy-chained will function like an inverter with a longer propagation delay. In the figure to the left, the +5V supply and the ground connections are not shown and the inverter symbol represents 3 daisy-chained inverters.
Sketch a figure similar to Figure 3.2 for our circuit. The input at A is a +5V pulse. Label the voltage axis with numerical values. Indicate the glitch's width – the delay for each individual inverter in can be found on the data sheet.



2. The circuit to the left is a **majority circuit**. Make a truth table to describe the majority circuit's operation. The entries **A**, **B**, **C** and **Output** are either **HIGH** or **LOW**. Include columns for the red LED and the Green LED where the entries are **LIT** or **OFF**.

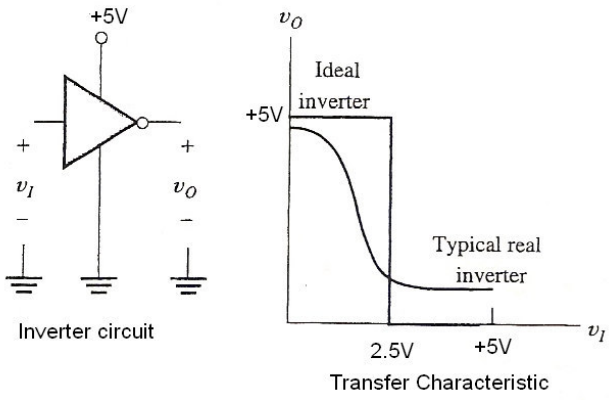
One last point; you were advised to ground all the unused inputs. Notice in the layout of the majority circuit, there are 4 bare wires that connect a node cluster on the bottom of the board to a node cluster on the top of the board. Those wires are the same wires used to ground the unused inputs in the inverter and glitch circuits.

Tip: strip about 10" of wire. Make eight of the bare wires described in the paragraph above. Once you get the first one "just right", the other "copies" go very quickly.

Before coming to the lab, you should:

- Put the pin numbers on the circuit diagram at the bottom of the layout pages. Use colored highlighters to identify the circuit diagram lines that correspond to the layout connections. Keep this for use in the lab. Turn it in with the final report.

In the lab



1. Use the function generator with a ramp wave of 1Hz. The ramp is the input to inverter1. Use a BNC tee to connect the inverter1's input to the scope's channel1. Use a 10:1 probe to observe the output of inverter1 on channel2. Switch the scope to X-Y mode. Do you get the inverters transfer function?

When a TTL inverter is used as a digital circuit, the input will be either a LOW or a HIGH. When we use a ramp as the input, the input slowly changes from a low voltage to a high voltage. Between the low and the high, the inverter acts as a high-gain inverting amplifier.

2. Breadboard the "glitch" circuit of Preliminary Question 1. Build the circuit in stages.

- Leave the LED circuit and switch circuit in place but disconnect them from the inverter. Daisy-chain one more inverter on the 7404. Use a 1kHz, +5V to 0V, 20% duty cycle pulse as input to inverter1.
- Observe the inverter's outputs with a 10:1 probe. Hint: clip the scope probe's black lead to ground. Remove the probe's mini-clip. The exposed probe tip will fit into the breadboard's holes.
- Hook up the NAND gate.
- Observe the output. It should be HIGH – you can't see the glitch yet because the scope's sweep speed is too slow.

Trigger the scope on the input. Set the sweep speed such that the scope trace shows two cycles of the input and output. You won't see the glitch at this sweep speed. Capture the trace. Increase the scope's sweep speed until you see the glitch. Capture the scope trace.

3. **(Optional/ Grace points)** Use the methodical procedure to breadboard the majority circuit of the preliminary question 3. Open/close the switches and observe the state of the LEDs. Record data in truth table form where the entries are OPEN or CLOSED for the switches and RED or GREEN for the LEDs… Is it a "majority circuit"?

Point to be highlighted in report

Compare the data of the Procedure to the Preliminary Questions results. Comment if they don't agree.

Reference for Keywords

Transfer Characteristics: https://en.wikipedia.org/wiki/Inverter_(logic_gate)

Logic Families and TTL logic families: https://en.wikipedia.org/wiki/Logic_family

Lab 4. Code Conversion

Objective

This experiment will introduce you to the digital design of a simple BCD to Excess-3 BCD and reverse.

Keywords: Karnaugh Map, Binary Coded Decimal, Excess-3 Code, Gray Code, ASCII, Unicode.

Components used in this lab: 74LS04, 74LS32 and 74LS20. Include 3 switches for the inputs and 3 LEDs

Background

Converting data from a human-friendly format into a form that is understandable by a machine is a key concept in any digital logic design environment. Computers and digital logic circuits in general, are only able to interpret data in binary form. Every form of data, from integers to characters, must be converted into a binary string in order for a computer to be able to manipulate and store them. This process is called encoding the data for use in a computer. Literally any data-encoding scheme will work as long as data is converted into a binary form that the computer is programmed to understand. For this reason, there are numerous different binary codes that may all represent the same thing. For instance, the ASCII code (American Standard Code for Information Interchange) utilizes a seven-bit string to represent 128 basic text characters, numbers, and control characters used in word processing. More recently, with the introduction of the Internet and globalization, a new code has been required to handle a vastly greater number of characters from languages throughout the world. Unicode was introduced to handle the vast number of characters and symbols now required for worldwide communication. It uses up to 24-bits and several encoding algorithms to achieve this. Clearly, a machine using ASCII codes would need some form of data conversion processing to understand Unicode, and vice-versa. In fact, many web pages containing foreign languages may not be displayed correctly depending on the Unicode support a given web browser allows.

This laboratory experiment will be focused on data conversion, but nothing as complicated as ASCII to Unicode. When all that is required is to encode integers from 0 through 9, encoding data becomes much simpler, although there are still a great number of different encodings that are possible. Binary Coded Decimal (BCD) is a code that uses four bits to represent each of the digits from 0 to 9. Each integer value is encoded directly in its binary form, producing several unused four-bit combinations. Excess-3 code is similar to BCD in that it uses a four-bit encoding for each integer value, however, when an integer is encoded, its value is first incremented by three, then converted to its binary form. The overall effect is to add three to all the Natural BCD codes. While Natural BCD encoding has six unused codes at the high end of the binary conversion (10 and above), Excess-3 BCD has three unused codes at the low end (2 and below) and three at the high end (13 and above). Table 4.1 shows the integer values from 0 through 9 and their corresponding Natural BCD and Excess-3 BCD encodings.

Decimal	Natural BCD			Excess-3 BCD		
	A2	A1	A0	Y2	Y1	Y0
0	0	0	0	0	1	1
1	0	0	1	1	0	0
2	0	1	0	1	0	1
3	0	1	1	1	1	0
4	1	0	0	1	1	1
5	1	0	1	0	0	0
6	1	1	0	0	0	1
7	1	1	1	0	1	0

Table 4.1: BCD Codes

A sample design procedure for 3-bit BCD to Gray code conversion design:
Step 1: Find the code conversion which you need to make and list out the inputs and corresponding outputs. The truth table is of 3-bit BCD to Gray code is given in Table 4.2 and here the A2, A1 and A0 constitute inputs while Y2, Y1, Y0 constitute outputs.

Decimal	Natural BCD			Gray BCD		
	A2	A1	A0	Y2	Y1	Y0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

Table 4.2: Binary to Gray code

Step2: Derive the equations for output literals in terms of input literals using K-map. Examples of K-map and simplification are given in Figure 4.1. Observe that the outputs Y2 to Y1 are expressed in terms of inputs A2 to A1.

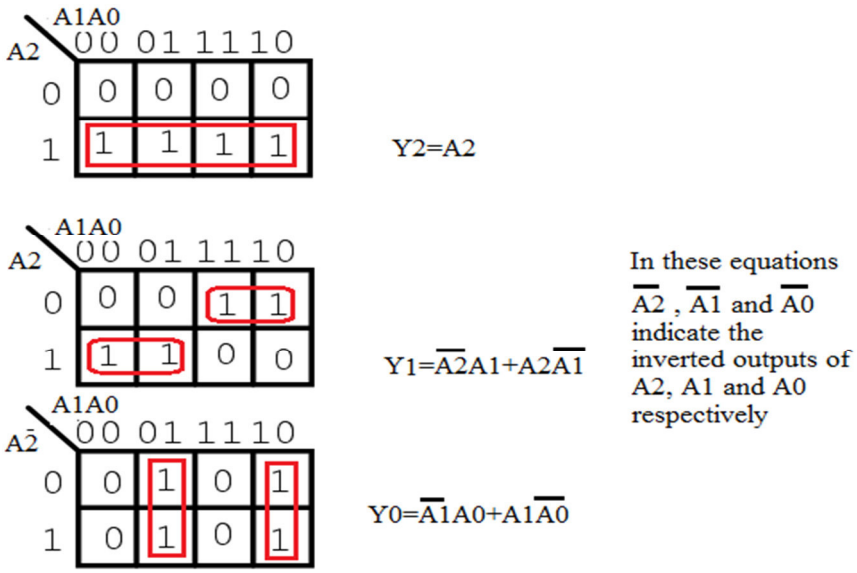


Figure 4.1

Step 3: Using the derived equations of outputs the schematics need to be drawn for each output as shown in figure 4.2.

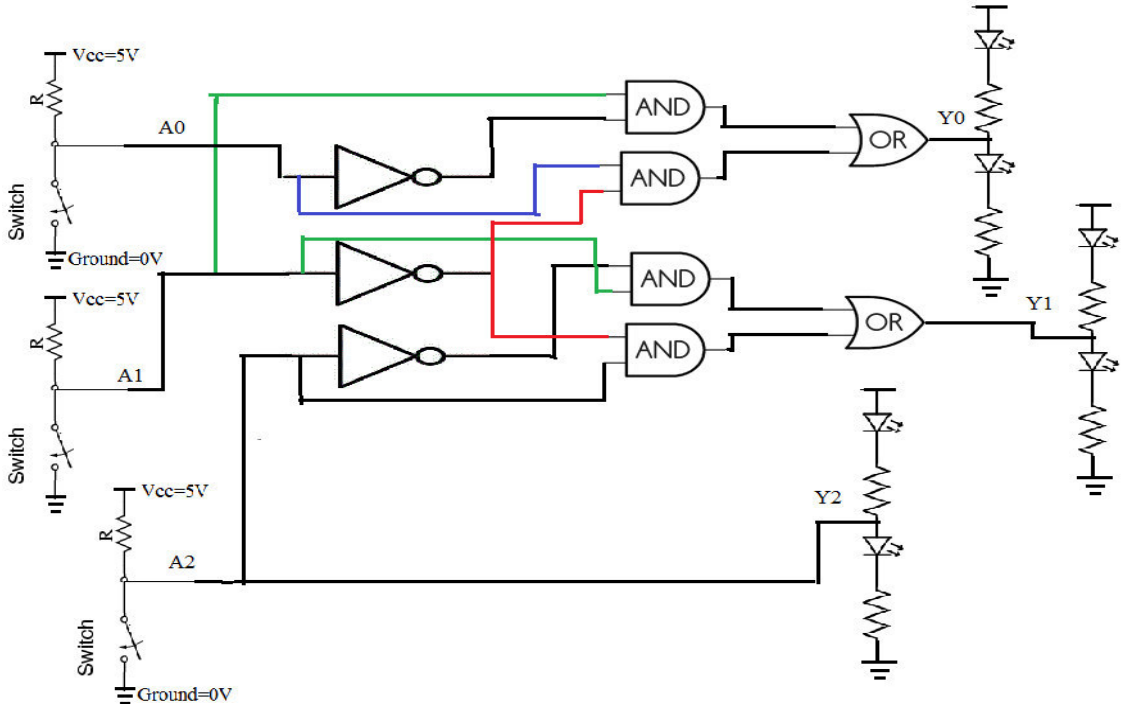


Figure 4.2

Preliminary Assignment

In this laboratory, two different code conversion circuits will be designed based upon the data encodings in Table 4.1. The first circuit will capture an input number from the user in Natural BCD form, and will output its Excess-3 BCD representation. The second circuit that will be built will perform the reverse operation and convert the Excess-3 BCD input into natural BCD representation. Figure 4.3 below contains block diagrams for both of these circuits.

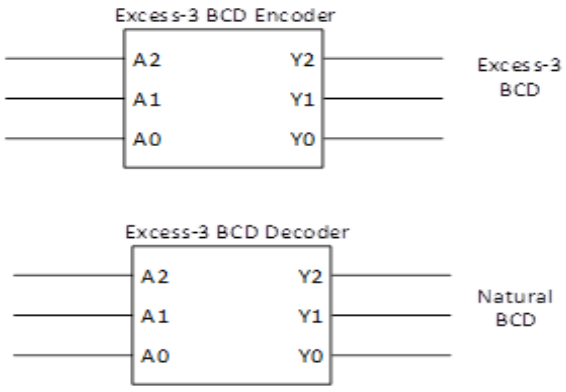


Figure 4.3. Converter Circuit Block Diagrams

With the exception of not including the unused codes for Natural BCD, Table 4.1 above is essentially the truth table for the first circuit to be designed for this laboratory. Using this truth table, the logic functions for each output of the encoder and decoder can be relatively easily obtained and minimized using Karnaugh Maps.

Before coming to the lab, you should:

1. Develop the logic functions for each output of the Excess-3 BCD encoder. Create the full truth table, and minimize the logic using Karnaugh Maps or another preferred method of your choice. Be sure to capitalize upon DON'T CARE conditions where possible.
2. Create the full truth table for the Excess-3 BCD encoder circuit. Be sure to include DON'T CARE terms where they apply.
3. Develop the logic functions for each output of the Excess-3 BCD decoder. Create the full truth table, and minimize the logic using Karnaugh Maps or another preferred method of your choice. Be sure to capitalize upon DON'T CARE conditions where possible.
4. Create the full truth table for the Excess-3 BCD decoder circuit. Be sure to include DON'T CARE terms where they apply.
5. For both circuits, draw the schematic representation using inverters, OR and AND gates. Hint: you can reuse the output of some gates using the De Morgan's law $\overline{A+B} = \bar{A} \cdot \bar{B}$ and $\overline{A \cdot B} = \bar{A} + \bar{B}$.
6. For both circuits, draw the breadboard layout using 74LS04, 74LS32 and 74LS20. Include 3 switches for the inputs and 3 LEDs for the outputs, with appropriate resistors.

In the Lab

1. Build and test the Excess-3 BCD encoder circuit.
2. Demonstrate the circuit to the TA.
3. Build and test the Excess-3 BCD decoder circuit.
4. Demonstrate the circuit to the TA.

Post-Lab Questions

1. Draw the schematic of a simple logic function that returns 1 if the input is from 0 to 7, and 0 otherwise.
2. Draw the schematic of a simple logic function that returns 1 if the input is from 3 to 10, and 0 otherwise.
3. Using a 3-bits adder, draw the schematic of the Excess-3 BCD encoder
4. Using a 3-bits adder, draw the schematic of the Excess-3 BCD decoder

Reference for Keywords

Karnaugh Map: https://en.wikipedia.org/wiki/Karnaugh_map

Binary Coded Decimal: https://en.wikipedia.org/wiki/Binary-coded_decimal

Excess-3 Code: https://en.wikipedia.org/wiki/Excess-3

Gray Code: https://en.wikipedia.org/wiki/Gray_code

ASCII: http://www.ascii-code.com/

Unicode: http://unicode.org/charts/

Lab 5. Finite State Machines

Objective

This experiment will review Finite State Machines (FSMs). Students will design and build a FSM using the classical design method.

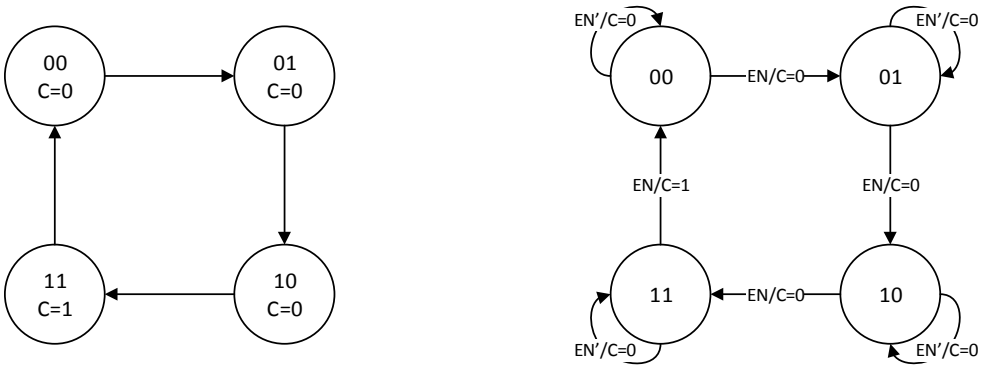
Keywords: State Diagrams, Flip Flops, Moore Machines, and Mealy Machines.

Components used in this lab:74LS175 (D-flip flop); other gates[AND(74LS08), OR(74LS32) NOT (74LS04), etc..] LED's, Switches and Resistors according to your design.

Background

State diagrams are used to specify other sequential logic circuits that implement a specialized logic function. They allow designers to specify the behavior of the FSM (i.e. the next state and outputs) as a function of current state and inputs.

Next state values are specified by transition arrows, while outputs are specified in one of two ways. First, some output values have a fixed value in each state. Outputs specified this way are called *unconditional outputs* since they depend only on the current state and not the value of the input. Unconditional outputs are specified in a state diagram by labeling the state bubbles with the output values that will appear in each state. Figure 5.1 (a) shows a two-bit binary counter with a carry output. The carry output C is true only when the counter is in state "11". FSMs that contain only unconditional outputs are called *Moore machines*.



(a) Simple Counter (Moore Machine)

(b) Counter with Enable (Mealy Machine)

Figure 5.1: Conditional and unconditional outputs in state diagrams

On the other hand, some output values depend on an input condition during a particular state. Outputs specified this way are called *conditional* outputs since they depend on an input condition as well as the current state. Conditional outputs are specified in a state diagram by labeling the arrows between states with the condition that causes the transition and the output value that should be asserted if that condition is true. Figure 5.1 (b) shows the state diagram of a two-bit counter with enable and a carry output. Since the carry output should only be true when the enable input EN is true and the current state is "11", it is a conditional output and the arrows be-

tween states are labeled with output values. FSMs that contain conditional outputs are called *Mealy machines*.

In the state diagrams we have worked with so far, we have used binary values for each state. However, it is sometimes useful to give states symbolic names that make it easier to keep track of what each state is doing. For example, if a FSM performs a sequence of functions, each state is given a name that refers to the function that it is currently performing. This makes the state diagram easier to understand. Figure 5.2 shows a state diagram with symbolic state names S0, S1, S2, and S3 that is similar to Figure 5.1 (b).

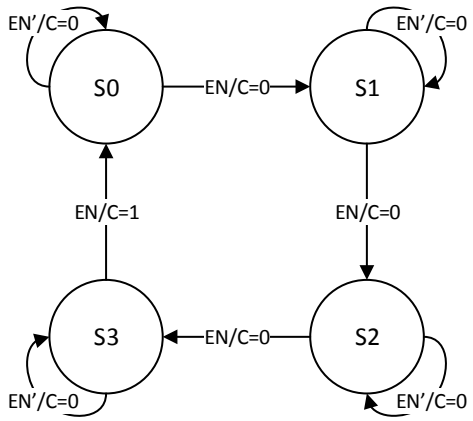


Figure 5.2: State diagram with symbolic state names

Each symbolic state name still represents a unique state value, although this value may not initially be specified. This value, called a *state code*, is supplied by the designer during a process known as *state assignment*. During state assign1nent, a state code is chosen for each symbolic state name. Different state assignments for the same state diagram can result in implementations that differ in cost. For this reason, computer-based tools are often used to perform state assignment for large FSMs. In smaller FSMs where cost is not a factor, arbitrary state assignments are often used.

Implementing Finite State Machines

Once a state diagram has been completed, the process of designing an implementation is a straightforward, mechanical process. The textbook [1] describes this process in Section 5.7. In short, the implementation process consists of the following steps:

1. Create a *state transition table* (STT) that corresponds to the state diagram. Each row of the STT corresponds to one transition "arrow" in the state diagram. It specifies an input condition, present state, next state, and output value. Table 5.1 (a) shows the STT that is created from the state diagram of Figure 5.2.
2. (Optional) Reduce the number of states in the state diagram by combining any equivalent states (i.e. states with the same outputs and same next state transitions). Section 5.7 of the textbook [1] describes the details of state minimization.
3. Perform state assignment. This involves choosing the number of flip-flops to be used in the implementation (at least log2 of the number of states) and assigning a state code to

each state that is a unique binary number. Arbitrary state assignments can be used on small state machines, but larger state machines may need a computer-based state assignment tool to reduce the implementation cost. After state assignment is complete, the state codes are substituted for the state names, creating a truth table for the next state and output values. For example, since the state diagram of Figure 5.1 has four states, two flip-flops are needed. Assigning the states to arbitrary state codes results in Table 5.1 (b).

4. Choose flip-flop type to be used in the implementation. JK flip-flops sometimes result in lower-cost implementations but are more complicated to work with. State machines using D flip-flops are easier to implement and will be used in this lab.
5. Create an excitation and output logic table. The excitation and output table acts as a truth table for the excitation inputs of each flip-flop and the outputs. For D flip-flops, this is equivalent to the state transition table created in step 3. For JK flip-flops, a new table must be created that specifies the values for the J and K inputs of each flip-flop (see section 5.7 of the textbook [1] for more details).
6. Simplify excitation and output logic using Karnaugh Maps, Boolean Algebra, or a computer-based logic minimizer. As discussed in class, Karnaugh Maps can be used to easily minimize logic functions with four or fewer inputs. Five and six input logic functions can be minimized using multiple Karnaugh maps, but this process is more difficult. Since the inputs to the logic functions include both state variables and FSM inputs, the excitation logic often has more than six inputs. In this case, small functions can be simplified using the rules of Boolean Algebra. For functions that are more complex a computer-based minimizer is essential.
7. Draw schematic diagram of the simplified logic and flip-flops required to implement the FSM.

IN	PS	NS	OUT
0	S0	S0	0
1	S0	S1	0
0	S1	S1	0
1	S1	S2	0
0	S2	S2	0
1	S2	S3	0
0	S3	S3	0
1	S3	S0	1

(a) State Transition Table (STT)

IN	Q0 Q1	D0 D1	OUT
0	0 0	0 0	0
1	0 0	0 1	0
0	0 1	0 1	0
1	0 1	1 0	0
0	1 0	1 0	0
1	1 0	1 1	0
0	1 1	1 1	0
1	1 1	0 0	1

(b) STT after State Assignment

Table 5.1: State Transition Table

FSM Implementation Example

In this section, we walk through an example FSM implementation using the state diagram shown in Figure 5.2 as a starting point. The first step is to create a state transition table (STT), which is shown in Table 5.1 (a). Since there are no equivalent states, in this state machine, state minimization is not attempted. Next, an arbitrary state assignment is performed. Since there are four states, a minimum of two flip-flops are required. If we choose two flip-flops, we can arbitrarily

assign state S0 the code “00”, state S1 the code “01”, state S2 the code “10”, and state S3 the code “11”. When these codes are substituted into the STT, the result is the table shown in Table 5.1 (b). If D flip-flops are used, we can name the outputs of these flip-flops Q0 and Q1 and the inputs of these flip-flops D0 and D1. In Table 5.1 (b), we see each value of IN, Q0, and Q1 specifies a desired value for D0, D1, and OUT. Thus, this table is a truth table for logic functions producing D0, D1, and OUT. Since each of these functions has three inputs, they can easily be simplified using Karnaugh Maps, as shown in Figure 5.3. Finally, Figure 5.4 shows a schematic diagram of the FSM implementation. Note that the product term $IN' \cdot Q1$ is common to two different logic functions (D0 and D1) and is reused.

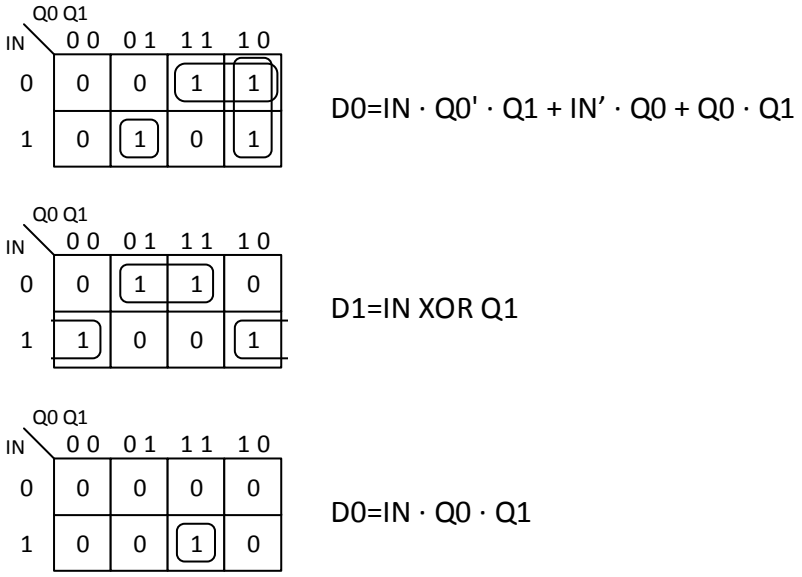


Figure 5.3: Simplified Next-State and Output Functions

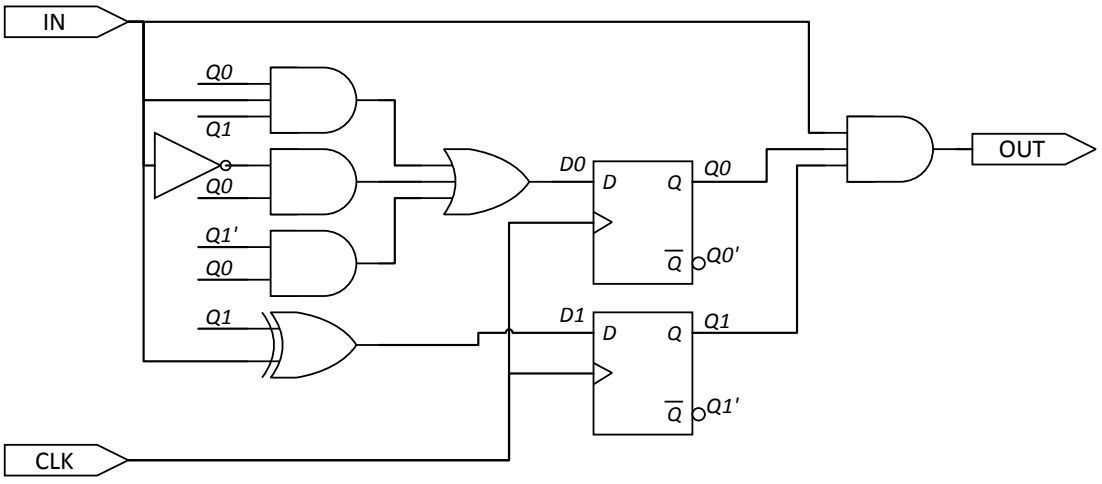


Figure 5.4: FSM Implementation

The Turn Signal FSM

For the design part of this experiment, we will implement an FSM that controls a turn signal and hazard flasher for a car. As shown in Figure 5.5, the rear of a car has three lamps labeled LL, HL, and RL. The FSM that controls these lamps has two inputs, L and R and three outputs that drive lamps LL, HL, and RL. It is also connected to a clock signal (not shown) that has a period equal to the rate at which signals should flash.

When L is high and R is low, the FSM should indicate a left turn by alternately making the LL output high and low during successive clock cycles until L is no longer high. Similarly, when L is low and R is high, the FSM should indicate a right turn by alternately making the RL output high and low during successive clock cycles until R is no longer high, When L and R are both high, the FSM should initiate a hazard flashing sequence in which LL and RL are both high, followed by LL and RL low and HL high, followed by LL, HL, and RL all low in successive clock cycles until L and R are no longer high.

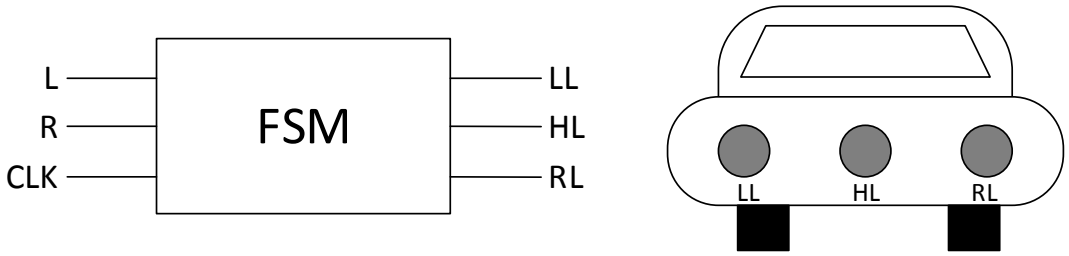


Figure 5.5: A turn signal FSM

Figure 5.6 shows a state diagram for the FSM. Each “bubble” in the state diagram shows the symbolic state name and the values of outputs LL, HL, and RL, respectively. Transition “arrows” are labeled by the input condition that must be true for the transition to be taken (transitions labeled with “1” are always taken).

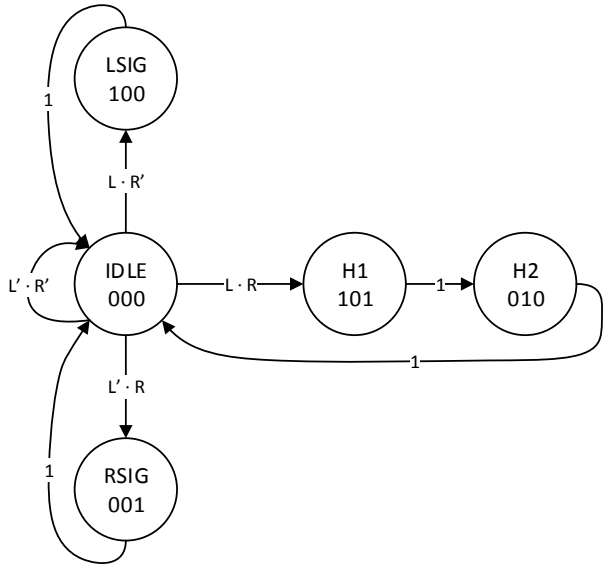


Figure 5.6: State diagram for turn signal FSM

In the IDLE state, nothing is happening and all lights are off, as shown by the output values "000". The FSM stays in the IDLE state as long as both inputs are false. If the L or R input is asserted true while the other remains FALSE, then the FSM begins a flashing sequence through states either LSIG or RSIG. After entering one of these states, the FSM returns to the IDLE state in the next clock cycle so that the signal flashes on and off. If L and R are both true, then the FSM enters the H1 state and turns on both the LL and RL outputs. In the next clock cycle, it enters state H2 and turns on the HL output before returning to the IDLE state to implement the hazard flashing sequence. It should be noted that this FSM is a "Moore machine" in which the output values depend only on the present state and not the current input values.

Preliminary Assignment

1. Implement the state diagram of Figure 5.2 using the state assignment S1="10", S1="11", S2="01", and S3="00". Compare the cost of this implementation to the implementation discussed in the previous section.
2. Using the procedures described in the previous section, create a State Transition Table for the turn signal state diagram shown in Figure 5.6.
3. Since there are five states in this FSM, the implementation of this circuit will require at least three flip-flops. We will implement this circuit using three D flip-flops. Assume that the flip-flop outputs are named Q0, Q1, and Q2 and the flip-flop inputs are named D0, D1, and D2. Given a state assignment of IDLE = 000, LSIG = 100, RSIG = 001, H1 = 101, and H2 = 010, write a truth table for the flip-flop excitation inputs D0, D1, and D2 and outputs LL, HL, and RL in terms of the L and R inputs and the present state values Q0, Q1, and Q2.
4. If you examine the values of the outputs in the truth table, you will see that the values of the outputs LL, HL, and RL are equal to the values of Q0, Q1, and Q2 in every state. This is done intentionally during state assignment and is called an "output-coded" state assignment. Output-coded state assignments are popular with designers because they eliminate output logic. However, they can only be used in Moore-style FSMs because the outputs cannot depend on input values. What else restricts the use of output-coded state assignments in FSM implementation?
5. Simplify the excitation logic for the turn signal FSM and design an implementation using parts available to you in the lab kit. You may wish to simplify the logic using Boolean Algebra instead of a Karnaugh Map. Draw a detailed schematic diagram of your circuit showing all connections and pin numbers and bring it to the lab.

In the lab

Build and debug the Turn Signal FSM that you designed in step 2 of the preliminary. Demonstrate its operation to the instructor.

References

[1] M. Morris Mano & Michael D. Ciletti, *Digital Design*, Fifth Edition, pp. 231-245, Prentice-Hall, Inc., 2012

References for Keywords

State Diagram: http://www.allaboutcircuits.com/textbook/digital/chpt-11/finite-state-machines/

Moore Machine: https://en.wikipedia.org/wiki/Mealy_machine

Mealy Machine: https://en.wikipedia.org/wiki/Moore_machine

Lab 6. Serial Adder

Objective

This experiment will introduce the concept of shift registers and their use in parallel-to-serial and serial-to-parallel applications.

Keywords: Shift Registers, Most Significant and Least Significant Bit and Serial Subtraction.

Components used in this lab: 74LS194 (x2) (Shift Registers), 74LS08(AND gate), 74LS32(OR gate) and 74LS175(D flip flop)

Background

Most adders used in digital circuits are implemented in parallel, the simplest example being the carry ripple adder, where each bit has a full adder block and where the carry is propagated from one block to another. However, another architecture of adder can be used to save space: a serial adder. This type of adder requires to send the two values to be added bit after bit, starting from the Least Significant Bit (LSB).

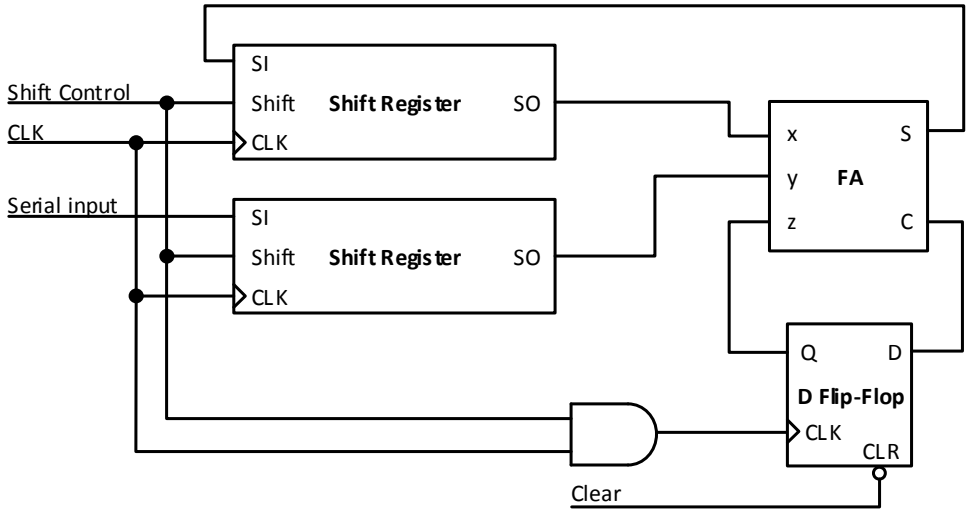


Figure 6.1. Serial adder using a single full-adder cell

Shift registers are used to store the values and transmit them serially, while a flip-flop is used to retain the carry for the computation of the next bit. For each new operation, the carry needs to be reset and the values needs to be reloaded in the shift registers. For it to work, the circuit needs to be provided with a clock, synchronizing the computation and the shifting. The computation time depends on the size of the data to be computed. For example, a 4-bit addition will require 4 clock cycles.

Preliminary Assignment

1. Using the following ICs: 74x194 (x2), 74x08, 74x32 and 74x175, design the schematic of a 4-bit adder based on Figure 6.1. You need to include switches to be able to load the data in the shift registers, and LEDs to display the result. Use appropriate resistors.
2. Draw the breadboard layout of the circuit.
3. List the set of actions to do to operate your serial adder properly

In the lab

1. Build the serial adder circuit.
2. In order to generate 4 clock cycles, we are going to use the signal generator.
   Check the signal generator manual on the computer to set the following parameters:
   - Pulse
   - HiLevel = 5V
   - LoLevel = 0V
   - High-Z Output
   - Pulse width = 1s
   - Duty cycle = 50%

   We are going to use a special functionality of the signal generator called Burst. It will generate a defined number of pulses on a trigger. Set it for 4 pulses on manual trigger. With this configuration, the signal generator will generate 4 pulses of 1 second each and every time you hit the Trigger button (and the Output is enabled).
3. Test the functionality of your circuit using the following values:
   - A = 0111, B = 0001
   - A = 0111, B = 1000
   - A = 0110, B = 0101
4. Test your circuit using 3 more values, chosen according to your own criteria (justify your choice).
5. Demonstrate the operation of your serial adder to the TA.

Post-Lab Questions

1. Adders are usually used to compute subtractions as well. Modify your schematic to allow your circuit to compute subtractions as well.

References

[1] M. Morris Mano & Michael D. Ciletti, *Digital Design*, Fifth Edition, pp. 261-266 & 466-467, Prentice-Hall, Inc., 2012

References for keywords:

Shift Registers: http://www.allaboutcircuits.com/textbook/digital/chpt-12/introduction-to-shift-registers/

Least Significant Bit and Most Significant Bit: http://www.buczynski.com/Proteus/msblsb.html

Serial Subtraction: http://www.asic-world.com/digital/arithmetic3.html

Lab 7. Counters

Objective

This experiment will review counters. Students will design counters from flip-flops and combinatorial logic and explore the capabilities of TTL counter chips.

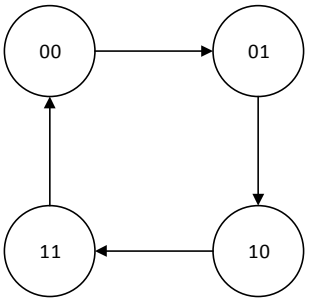
Keywords: Counter.

Components used in this lab: 74LS163, other gates resistors, switches and LEDs as per your design.

Background

A *counter* is a sequential logic circuit that goes through a specific sequence of states. The *binary counter* is the best-known form of counter. It goes through a sequence of states in which the flip-flop values represent increasing binary values. For example, a two-bit counter goes through states "00", "01", "10", and "11" before returning to state "00". In general, an n-bit binary counter goes through a sequence of states with values 0 to 2^n-1 before returning to the 0 state. Other counting sequences are also possible. For example, a *Gray code* counter goes through states in a way such that only one bit changes at a time. A two-bit Gray code goes though states "00", "01", "11", and "10" before returning to state "00". Most counters used today are *synchronous* - all flip-flops in the counter are connected to a common clock. Asynchronous counters such as "ripple counters" can be easier to implement but are difficult to use reliably. For this reason, this experiment deals only with synchronous counters.

It is convenient to represent the behavior of sequential logic circuits such as counters using either a *state diagram* or *state transition table*, as shown in Figure 7.1. A state diagram represents each state as a "bubble" that shows the flip-flop values for the state. Arrows between bubbles represent *state transitions* that occur when going from one clock cycle to the next clock cycle. Figure (a) shows the state diagram for a two bit binary counter. It shows that if the current state is "00" then the next state will be "01", if the current state is "01" the next state will be "10", etc. The state transition table provides the same information in a table, as shown in Figure 7.1 (b). Given a current state that is specified by flip-flop values QA and QB, it specifies the next state values of the flip-flop, which are labeled QA* and QB*.



(a) State Diagram

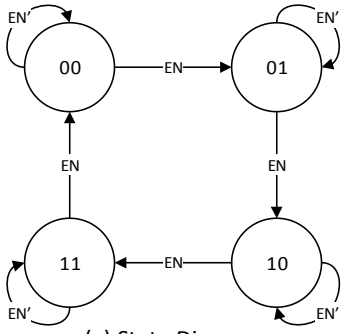
Present State		Next State	
QA	QB	QA*	QB*
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

(b) State Transition Table

Figure 7.1: Two-bit binary counter

Often, state transitions in sequential logic circuits depend on an input as well as the current state. For example, a *binary counter with enable* only counts when an enable input EN is true. When the EN input is false, it stays in the current state. Figure 7.2 shows the state diagram for a two bit binary counter with enable.

When state transitions depend on input values, they are labeled with a Boolean expression that specifies the conditions under which the transition is taken. For example, when the counter in Figure 7.2 is in the "01" state and EN is true, the next state is "10". Thus, the arrow between state "01" and "10" is labeled with the expression EN. On the other hand, if the counter is in state "01" and EN is false, then the next state will remain "01". Thus the arrow from state "01" back to itself is labeled with the expression EN' (EN prime). The state transition table describes the information in tabular form. Thus next state values QA* and QB* are described as functions for EN, QA, and QB, as shown in Figure 7.2.



(a) State Diagram

	Present State		Next State	
EN	QA	QB	QA*	QB*
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

(b) State Transition Table

Figure 7.2: Two-bit binary counter with enable

If D flip-flops are used to implement a sequential logic circuit, then the state transition table becomes a truth table for the D inputs of the two flip-flops. These logic functions can be minimized and used to create an implementation of the circuit. If other flip-flop types are used (e.g. JK flip-flops), then an additional step must be taken to determine the values of the flip-flop inputs that will result in the proper flip-flop values. These input values are known as excitation values. Section 5.7 of the textbook [1] describes this process in more detail. You should review this section before beginning the preliminary.

Since counters perform a very useful function, the TTL logic family provides several different counters. Figure 7.3 shows the pinout of the 74LS163, a 4 bit binary counter that is provided in the lab kit. In addition to performing the basic counting sequence, the 'LS163 performs several other useful functions. It has four outputs QA, QB, QC, and QD that are the outputs of the counter flip-flops (QD is the most significant bit).

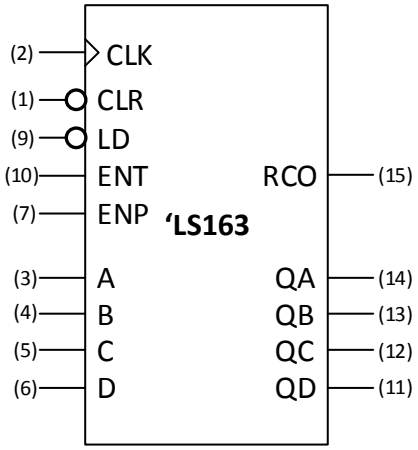


Figure 7.3: 74LS163 binary counter

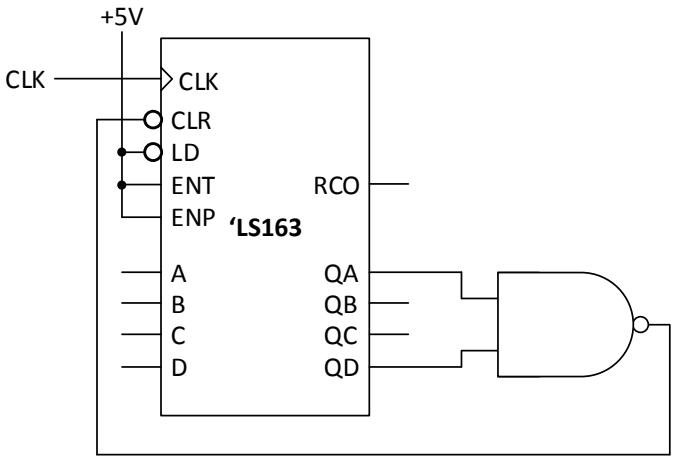


Figure 7.4: Decade counter using 74LS163

An additional output called RCO is a "carry" output that is true when the counter is in state "1111" and the ENT input is logic H. ENP ("Enable Parallel") and ENT ("Enable Trickle") act as enable inputs. When either ENT or ENP is logic L, the counter stops counting and remains in the current state. In addition, when ENT is logic L, the RCO output is disabled (logic L) regardless of the current state. Two counters can be combined to create an 8-bit counter by connecting the RCO output of one counter to the ENT input of the second counter.

The CLR input is an active-low "synchronous clear". It is called synchronous because the flip-flops in the counter are reset on the next rising clock edge after CLR is asserted to logic L. This is in contrast to asynchronous clear inputs, which reset the flip-flops immediately when CLR is asserted. Because it is synchronous, it can be used without worrying about the timing problems associated with asynchronous reset inputs.

The LD input loads the A, B, C, and D inputs into QA, QB, QC, and QD, respectively. Like CLR, this signal is active-low and synchronous.

With all of these capabilities, the 'LS163 can be adapted to perform many functions besides the simple binary counter function. For example, the CLR input can be used to "cut short" a counting sequence by returning the counter to state "0000". Figure 7.4 shows the use of the CLR input to create a "decade counter" which counts from 0 to 9. When the counter reaches "1001", the output of the NAND gate becomes logic L. This forces the next state of the counter to be "0000". The LD input can be used to start counting sequences at other values besides "0000". For example, if the output of the NAND was connected to LD instead of CLR, C & D grounded, and A, B & CLR tied to Vcc then counter would count from 3 ("0011") to 9 ("1001").

Preliminary Assignment

1. What range of values does a 10-bit binary counter go through?
2. Draw the state diagram and state transition table for a 2-bit Gray code counter with enable input EN. When EN is logic H, the counter should go through the counting sequence for a Gray code. When EN is logic L, the counter stays in the current state.

3. Design the 2-bit Gray code counter with enable using D flip-flops and combinational parts available to you. Use the state transition table developed in step 3 as a truth table for the combinational logic as described previously.
4. Draw a schematic showing how three 74LS163 counters can be connected to create a 12-bit counter with enable.
5. Using a 74LS 163 and other parts in the lab kit, design a specialized counter with two inputs C0 and C1 (Table 7.1). Depending on the values of C0 and C1, it should go through the counting sequences shown below. Draw a state diagram for this counter.

C0	C1	COUNTING SEQUENCE
0	0	0 to 15
0	1	0 to 12
1	0	3 to 15
1	1	3 to 12

Table 7.1: Counting sequence for specialized counter with two inputs C0 and C1

In the Lab

1. Build the 2-bit Gray code counter with enable designed in the preliminary assignment. Demonstrate its operation to the instructor.
2. Build the specialized counter designed in the preliminary assignment. Demonstrate its operation to the instructor.

References

[1] M. Morris Mano & Michael D. Ciletti, *Digital Design*, Fifth Edition, pp. 231-245, Prentice-Hall, Inc., 2012

References for keywords:

Counters: https://en.wikipedia.org/wiki/Counter_(digital)

Lab 8. Programmable Logic Arrays

Objective

This experiment will introduce the use of a PLD (Programmable Logic Device) to implement logic functions. In the previous experiment logic functions were implemented with TTL chips.

Keywords: Programmable Logic Devices, rest of the description is given in the background section.

Components used in this lab: 74LS47(Seven Segment Display Driver), GAL22VPD-25QP (Programmable Logic Device), Resistors, Switches and Seven Segment Display(Part Number: SA56-11SRWA).

Background

Since their introduction in the late 1970's, programmable logic devices have become very popular for use in implementing logic functions. PLDs can implement a wide variety of logic circuits often in fewer packages than would be possible using TTL.

The PLD we will be using is re-programmable, i.e., if we need to re-design our circuit, either because we made a mistake or if we just want to add a new feature to our circuit, all we need to do is change a few lines of code and then reprogram the PLD. Re-designing a circuit implemented with TTL usually is a major project.

Use of a PLD requires programming with a HDL (Hardware Description Language). We will be using ABEL (Advanced Boolean Equation Language), which was developed by Data I/O Corporation. ABEL is now owned by Xilinx, Inc.

ABEL

There are many ABEL tutorials available on websites. A particularly concise one is a 22-page primer used at the University of Pennsylvania. This primer can be found on Black Board; ECE 218→Lab→ABEL Primer.

There are enough examples in what follows that mastery of the ABEL language is not required for this lab. Thus, you may choose to "skim read" the ABEL primer rather than digest all 23 pages.

A basic ABEL program consists of the following four sections:

- **Module**: each source file must start with the key word module followed by a module name. For this lab, name your module Your_Name_ECE218_Lab8_Project1.
- **Pin**: pin declarations are used to define the symbolic names for your inputs and outputs and the pin numbers they are associated with.
- **Equations**: Use the keyword equations to begin the logic descriptions. Equations specify the relation between the inputs and the outputs using Boolean logic expressions. The ABEL syntax or the important logical operators are:

Operator	=	!	&	#	$
Syntax	Assignment	NOT	AND	OR	XOR

- **Test Vectors**: Test vectors are the "solution" of the Boolean equations written in the equations section. They express the outputs as a function of the inputs in a truth table format. Test vectors are optional but provide a means to verify the correct operation of the circuit. The program checks the Boolean equations against the test vectors. If they don't agree, the program gives an error message.

Comments can be inserted anywhere in a file and begin with a double quote and end with another double quote or the end of a line whichever comes first.

An example ABEL code for the half adder is given below:

```
module YourName_ECE218_Lab8_Project1;
" The Title is optional. The title's name must be enclosed in single quotes. The title line is ignored by the
"compiler, but is useful in identifying projects"
title 'Half_Adder'

"x and y are inputs variables. x is associated with pin3 and y with pin5
x, y pin 3, 5;.

" output pins are specified next. ; S stands for Sum and C stands for Carry. The istype is an optional assignment
"to an output pin. 'com' indicates a combinational signal. 'reg' indicates a clock signal (registered with a flip
flop)
S, C pin 15, 18 istype 'com' ;

equations
S = (x & !y) # (!x & y) ;
C = x & y;

Test_vectors
( [x, y] -> [S, C] )
[ 0, 0 ] -> [0, 0];
[ 0, 1 ] -> [1, 0];
[ 1, 0 ] -> [1, 0];
[ 1, 1 ] -> [0, 1];
End
```

Table 8.1: Half-adder ABEL code

In the example above, the pin numbers must be chosen to be consistent with the PLD we will use. The GAL22VPD chip is a 24 pin chip. The inputs are pins 2 through 11 and pin 13. Pin 1 is an input but is a clock (if used). The outputs are pin numbers 14 through 23.

There are two ways to "tell" the ABEL program what combinational logic circuit we want to implement. We can either write the Boolean equations describing the circuit or we can describe the circuit with a truth table. We will repeat the above example using a truth table instead of Boolean equations.

```
module YourName_ECE218_Lab8_Project2;
x, y pin 3, 5;.
S, C pin 15, 18 istype 'com' ;
truth_table ( [x, y] -> [S, C] )
[ 0, 0 ] -> [0, 0];
[ 0, 1 ] -> [1, 0];
```

```
[ 1, 0 ] -> [1, 0];
[ 1, 1 ] -> [0, 1];
End
```

Table 8.2: Half-adder ABEL code without Boolean equations

This example is looks so compact because we have omitted the optional items such as comments, title and test vectors.

ispLEVER Classic/Project Navigator Software

Lattice Semiconductor, the manufacturer of the PLD chips, has a software program named "ispLEVER Classic". The program uses ABEL to create a file that is used as the input to a hardware device that will program our PLD. The ispLEVER Classic program is installed on the computers in our lab. Although it is not necessary, you may want to download the program to your laptop.

To download/install a copy of the program installed on the ECE 218 lab computers go to

latticesemi.com >DOWNLOAD>SOFTWARE>ispLEVER-Classic.

This should lead you to a 3-step process. On step-1, you will be given the choice of several downloads. Download only the ispLEVER Classic Base Module. Step-3 informs you that you need a valid software license. Don't panic – the license is free. On the software license request form for Company use "IIT Student". The license is valid for one year and is renewable at no cost.

Part of ispLEVER Classic is a program named Project Navigator. This is the program that is used to create the file need by the hardware device that will program the PLD. The use of Project Navigator is outlined below. It will also be demonstrated in the pre-lab session.

Using Project Navigator

1. Open the Lattice Semiconductor software from Start/Programs/Lattice Semiconductor/ispLever Project Navigator
2. Open a new project from File/New Project. Type in the project name and select Schematic/ABEL as "Design Entry Type". You should also specify the project location path where the files will be saved. If you are using the computers in the lab, specify your thumb drive. Then click "Next" to continue. In the "Select Device" window, "tell" the program what PDL we will be using. Select the following parameters if you are using a GAL22VPD-25QP chip:

   Family: GAL Device
   Device: GAL22VPD
   Speed Grade: -15
   Package Type: 24 PDIP

Part Name: GAL22VPD-25QP
Click Next/Next/Finish. This will bring you back to the project navigator window.

3. On the left hand side, you should see a section called "Sources in Project". Your project name should be there. Right click on the project name and select new. In the new source window, select ABEL-HDL Module and click OK. Type in the Module Name, File name (no spaces allowed) and the Title (title is optional).
4. A text file will open with the following format:

   MODULE your_module_name

   TITEL 'your_title_name'

   END

   Write the ABEL code for the given circuit between TITLE and END. You may also paste the code previously written using a word processing program; Notepad works.

5. Once the code is written in the text file and saved, there should be two files in sources in window section. One is the ABEL source code file (having the extension .abl) and the other one is the test vectors file that ABEL created.
6. Click on the .abl file and double click on Check Syntax. If there are any syntax errors then ABEL will show it in the log window.
7. After removing the syntax errors, click on the vectors file and double click on "Compile Test Vectors". If there are any syntax errors in the test vectors, the test vectors will not compiled.
8. After compiling the test vectors, double click on JEDEC Simulation Report. This will show whether all the test vectors passed. This will also create a file project_name.jed in the current directory that you specified when you created new project.
9. This .jed file is used to program a GAL chip. Save it on your USB drive.

Summary of Steps to Program Chip using <u>DATA i0 Model PLUS 48</u>

The printer port of a computer in the SH 311 lab is connected to a piece of hardware - a DATA i0 model PLUS48 programmer. There is in icon on the desktop entitled "Shortcut to PaINT.exe".

Opening that icon will bring up a windows program that checks the status of the printer port. If you forgot to turn on the power to the DATA i0 hardware or someone disconnected the hardware, you will get a warning with instruction as what to check. If everything is OK with the hardware, a new window will open which prompts you to select the chip that you will program. If you are using the GAL22VPD-25QP chip, select:

Vendor: National Semi
Family: GAL
Device: GAL20V8A

Click OK to proceed.

A new window titled "Windows Software: Logic Programmer" appears. Select file then select load. This is the step where you "tell" the program where to find your *.jed file (which you copied to your thumb drive).

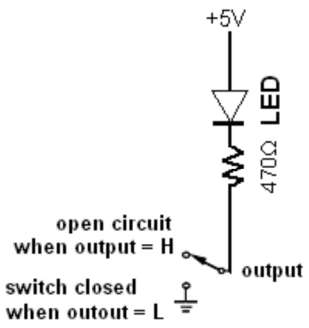
After you have loaded your *.jed file, select the Program button. The program will check that the correct chip is in the hardware's socket and if OK the program will "burn-in" the logic and flash PASS or FAIL. Fail could mean the chip is bad, you inserted it in the wrong pins of the socket, or ….

The 7-segment display and the BCD decoder/driver

In a previous lab, we turned on/off an LED with a logic gate. We used a resistor to limit the current.

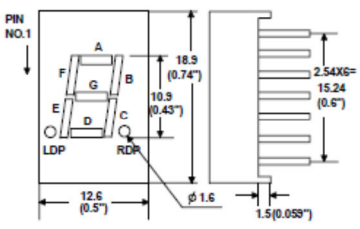
In this experiment we will use a TTL SN74LS47 seven-segment decoder/driver to control the LED's in a 7-segment display. The SN74LS47 has four inputs (A, B, C, D) and 7 outputs (a, b, c, d, e, f, g). The comments below refer to the attached 7447 data sheet:

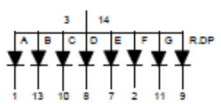
- Page 1 – VCC = 5V on pin 16 and ground pin 8.
- Page 3 – For now ignore inputs BI, RBI, and LT in the Function Table. If an L input represents 0 and an H input represents 1, then A, B, C and D are the binary-coded-digits for the decimal numbers in the left most column. The outputs column indicates which of the outputs will turn on the various LED segment connected to them. Note for example, a decimal 6 at the input doesn't turn on segments a and b; thus the 6 doesn't have a "tail".
- Page 3 - Surprisingly, for the "illegal" decimal numbers 10 thru 15, the display is not blank – it displays some odd looking characters which could easily be misread. Another decoder/driver made by ON Semiconductor, the MC14511, decodes the "illegal" inputs so that the display is blank.
- Page 2 – The table indicates that the SN74LS47's output has active level low and has an open-collector output configuration. "Active" refers to the voltage that turns on a segment. This means an LED is turned on by a low output.



"Open-collector output" tells the designer to model the output as a switch that either grounds the output or open-circuits the output. The designer would use the circuit on the left as the equivalent circuit looking into the output of the decoder/driver. The LED represents one of the segments of the 7-segment display. The LED is ON when the switch is grounded and OFF when the switch is open. The 470Ω limits the current in the LED.

The other six segments are connected to the other six decoder/driver outputs. Each segment requires its own current limiting resistor.





Common anode pins 3 & 14
Pin 6 not connected
No pin in position 4, 5 & 12

The 7-segment display we will use is shown on the left. There are 7 LED segments and 1 LED decimal point. All 8 anodes are con-

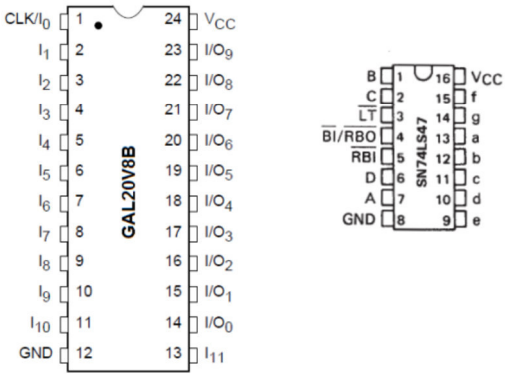
nected together. This is called a "common anode" display. Pin3 or pin14 should be connected to +5V. A current limiting resistor must be connected between the output of the decoder/driver and the display.

We stock a resistor array that can be used as the current limiting resistors. The array has eight isolated 470Ω resistors in a 16pin DIP. The resistors are connected between pin1-pin16, pin2-pin15, etc.

Displays also come in a "common cathode" configuration. (See the Appendix section below.)

Go back to the SN74LS47 datasheet for a few more comments.

- The function BI (blanking Input), if L, turns-off all the segments. The LT (lamp test) function, if low, turns on (tests) all the segments. For normal operation, keep both BI and LT high. BI/RBO is identified as a wired AND connection. See page 97 of your textbook [1] for information about wired AND. We are going to use pin4 as an input and hold it high.



In this experiment, we are going to use both a PLD and a SN74LS47 as decoder/drivers. Plan ahead. The TTL chip specifies where the input and output pins are with respect to one another. The PLD lets you assign the pin locations. As you write the code for the PLD, assign the pin numbers in the same pattern as the TTL chip. That allows you to unplug the TTL chip and plug in the PDL w/o much (if any) rewiring of the current limiting resistors and the display. Compare the pin-out diagram of the SN74LS47 with the pin-out diagram of the GAL22VPD. Fortunately, both have inputs are on the left and outputs on the right. You can keep the pin layout pattern of the TTL and PLD the same by assigning the input pins of the PLD pins B→pin3, C→pin4 D→pin8 A→pin9 and assign the output pins of the PLD f→pin21, g→pin20, a→pin19, b→pin18, c→pin17, d→pin16, e→pin15. **Plan ahead; every minute spent planning is time well spent**.

The Programmable Logic Device

The PLD we will be using is a GAL20V8. A datasheet or "spec sheet" can be found attached to this experiment. The datasheet is 16 pages in length. Page 7 pages gives a functional description of the chip and may be worth reading. Page 8 gives the pin-out configuration which is essential. The remaining pages are detailed technical specifications – not necessary for this lab.

Some necessary items from the datasheet:

- 24 pin DIP. V_{CC} = 5V on pin 24 and ground on pin 12.
- Pins 2 – 11 (on the left hand side) and 13 (on the right hand side) are inputs.
- Pins 14 – 23 are outputs (are on the right hand side).
- Unused inputs pins should be tied to VCC by a pull-up resistor or should be grounded.

- Inputs/outputs are TTL compatible.

Preliminary Assignment

1. Copy/paste the ABEL half-adder program in the introduction into a Notepad (or other word processing program). If you are going to use a two-wide DIP switch to toggle the input from high to low, change the input pin assignments to consecutive numbers. Name the file Half_Adder_Program. Save on your thumb drive.
2. In the lab, you will use the Project Navigator program to check the syntax and test vectors of an ABEL program. If your program is OK, you won't get any errors. If you don't get any error, you won't "see" how to debug the program. Put a couple errors in the program of preliminary question 1 and save the file as Half_Adder_w_Errors. Some common errors you can try are:
   - Leave-off the ";" after a line of code
   - Misspell the keyword "equations"
   - Incorrectly assign a pin#, eg, assign an output to pin24. (Pin24 is V_{CC} = 5V.)
   - Interchange is & and the # in the Boolean expression. It is still a valid Boolean expression but the test vectors won't match the equations.
3. Write an ABEL program that will allow the PLD to replace the SN74LS47, except blank the display for inputs decimal 10 thru 15. Chose the pin assignments so that you can easily swap the TTL and PLD chips w/o extensively rewiring the breadboard. Save the program on your thumb drive. After you do question 6, check the pin assignments and change if necessary.
4. Layout the GAL22VPD as programmed in preliminary question 1. To verify the outputs S and C, use a discrete 470Ω and red LED on S and a discrete 470Ω and green LED on C. Use a 10 pin 1kΩ common SIP resistor array to pull up the unused inputs of the PLD chip to a high and to accommodate the DIP switches used to toggle the inputs from a high to a low. If you don't have enough pull-up resistors in the array or if one of the unused inputs is on the "wrong" side, ground it. Don't leave any inputs floating.
5. Layout the SN74LS47 → 470Ω isolated resistor array DIP → display. Don't forget to include a 1kΩ SIP array to pull up the unused inputs of the TTL chip to a high and to accommodate the DIP switches used to toggle the inputs from a high to a low.
6. Modify the layout of preliminary question 5 to accommodate the GAL22VPD.

In the lab

1. Use Project Navigator to debug the "bad" program of preliminary question 2. After debugging, save the .jed file to your thumb drive. Program the PLD using the procedure described on page 4 of this experiment. Plug the PLD into the breadboard of preliminary question 4 and verify the half-adder's operation.
2. Breadboard as per preliminary question 5. Test the display by grounding the TTL's TL pin. All seven segments should go on. If OK, verify the BCD decoding function by toggling the DIP switches.
3. Use Project Navigator to program the decode/drive function of preliminary question 3. Use the .jed file to program the PLD. Toggle the DIP switches to verify the circuits operation.

Report

Include pictures of your breadboarded circuits in your report. Comment on the usefulness of PLDs.

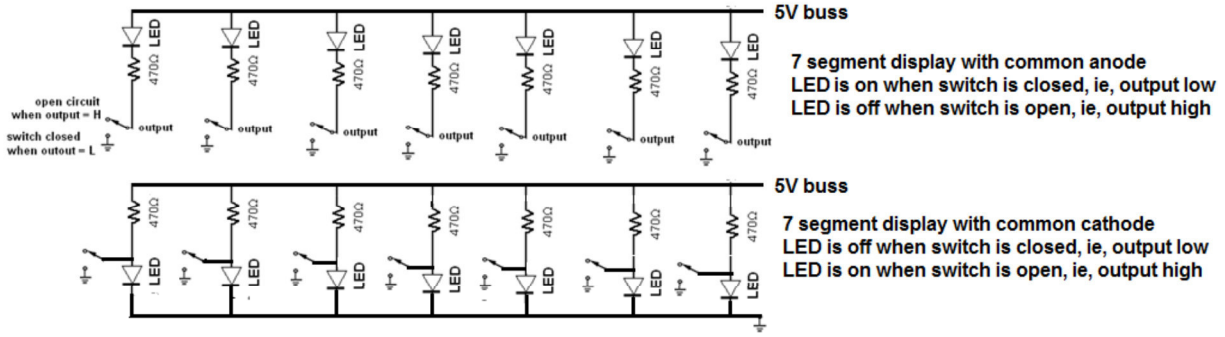
References

[1] M. Morris Mano & Michael D. Ciletti, *Digital Design*, Fifth Edition, Prentice-Hall, Inc., 2012

Appendix

The SN74LS47's output has active level low. This means an LED is turned on by a low output. The 7447 is intended to be used with a common anode display. The SN74LS49's output has active level high. This means an LED is turned on by a high output. The 7449 is intended to be used with a common cathode.

We normally stock only the common anode displays. If a common cathode display is used, the hookup is different as shown below.



If a 7447 seven-segment decoder/driver is used to control the segments of a **common cathode** 7-segment display, the display will be "backwards". A 0 turns just the middle segment on. An 8 turns all segments the off, etc. As an exercise, fill out the "backwards" display shown below.

The 7448 seven-segment decoder/driver has active level high with built-in 2kΩ pull up resistors. Advantage: you don't need external pull up resistors to limit the current. Disadvantage: LED won't be as bright because the current is only about ¼ of the current when using 470Ω pull up resistors.

References for Keywords

Programmable Logic Devices: https://en.wikipedia.org/wiki/Programmable_logic_device

Hardware Description Language: https://en.wikipedia.org/wiki/Hardware_description_language

Lab 9. Sequential Logic Design with PLDs

Objective

In this lab, you will design and use sequential circuits and FSMs with programmable logic devices (PLDs). The concepts of digital system data path and controller elements are introduced. To illustrate these concepts, you will design and build a simple "ping-pong" game using a FSM and a specialized shift-register.

Background

In previous experiments, we used programmable logic devices (PLDs) to synthesize combinational logic. PLDs with “registered” outputs (i.e. with flip-flops on their outputs) can be used to implement sequential circuits. Figure 9.1 shows the basic structure of a registered PLD. Sum-of-product outputs from the OR array are connected to flip-flop inputs. The flip-flop outputs are connected to output pins and are internally fed back to the AND array. Thus, each flip-flop can be programmed to be a function of both inputs and current flip-flop outputs, making PLDs ideal for implementing sequential logic circuits. PLDs with configurable output “macrocells” (as the GAL22VPD used in this lab) can be set to use both registered AND combinational outputs.

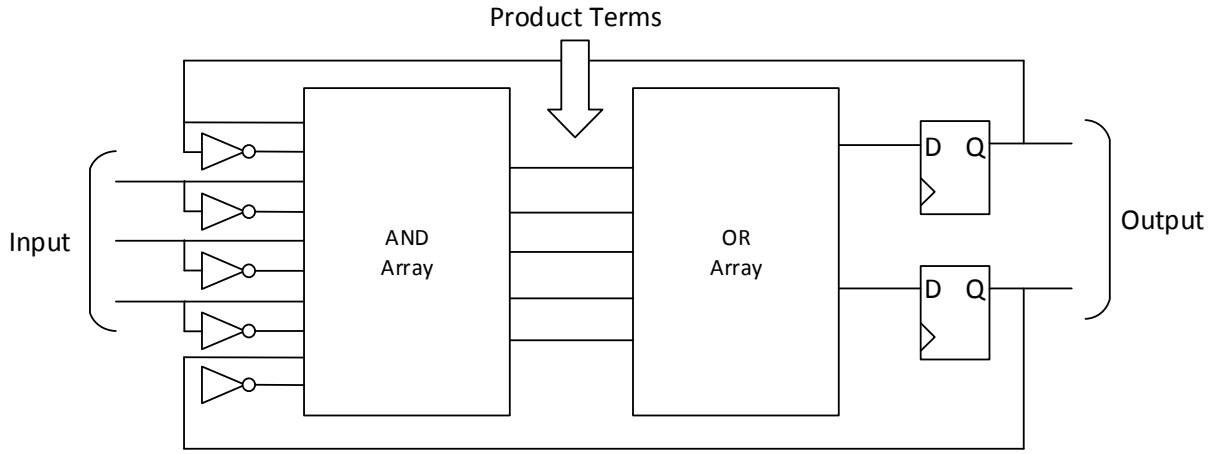


Figure 9.1: Structure of a sequential PLD

To program sequential PLDs using ABEL, logic equations are written in the same way that they are for combinational circuits except that the “=” combinational assignment operator is replaced with the sequential assignment operator “:=”. This specifies that the value on the right hand side of the equation will be latched into the flip-flop at the rising edge of the clock.

Table 9.1 shows an example of a sequential logic circuit implemented using ABEL. This circuit implements a four-bit “universal shift register” similar to the 74LS194 TTL part. Registered outputs QA, QB, QC, and QD are used as the shift register's flip-flops. Depending on the values of the select signals S0 and S1, the circuit will either hold its current value, shift data left using LIN as an input, shift data right using RIN as an input, or load a new value into all four flip-flops using the A, B, C, and D inputs. The EQUATIONS section specifies the logic equations for each flip-flop.

```
MODULE ShiftReg
"Inputs
CLOCK, S0, S1, LIN, RIN, A, B, C, D PIN;
"Ouputs
QA, QB, QC, QD PIN ISTYPE 'REG';

EQUATIONS
QA:= (!S0 & !S1 & QA) # (!S0 & S1 & RIN) # (S0 & !S1 & QB) # (S0 & S1 & A);
QB:= (!S0 & !S1 & QB) # (!S0 & S1 & QA) # (S0 & !S1 & QC) # (S0 & S1 & B);
QC:= (!S0 & !S1 & QC) # (!S0 & S1 & QB) # (S0 & !S1 & QD) # (S0 & S1 & C);
QD:= (!S0 & !S1 & QD) # (!S0 & S1 & QC) # (S0 & !S1 & LIN) # (S0 & S1 & D);

QA.CLK=CLOCK; QA.OE=1;
QB.CLK=CLOCK; QB.OE=1;
QC.CLK=CLOCK; QC.OE=1;
QD.CLK=CLOCK; QD.OE=1;

TEST_VECTORS
([CLOCK,S0,S1,RIN,LIN,A,B,C,D] -> [QA,QB,QC,QD])
[.C.,1,1,0,0,0,0,0,0] -> [0,0,0,0];
[.C.,0,1,1,0,0,0,0,0] -> [1,0,0,0];
[.C.,0,1,1,0,0,0,0,0] -> [1,1,0,0];
[.C.,1,0,0,1,0,0,0,0] -> [1,0,0,1];
[.C.,1,0,0,1,0,0,0,0] -> [0,0,1,1];
[.C.,0,0,0,0,0,0,0,0] -> [0,0,1,1];
[.C.,1,1,0,0,1,0,1,1] -> [1,0,1,1];

END
```

Table 9.1: ABEL file for a four-bit universal shift register

PLDs can be used to implement FSMs using the methods described in the "Finite State Machine" lab to translate the state diagram into excitation logic for the flip-flops and writing equations for this logic using the ":=" operator. However, ABEL supports a more direct method of generating implementations directly from state diagrams.

In the direct implementation method, ABEL allows the user to specify state variables, a state assignment, state transitions, and FSM outputs. Table 9.2 shows an ABEL specification for the two-bit counter with enable described in the "Finite State Machines" lab.

Digital systems are often divided into two parts: the *datapath*, which stores and manipulates information, and the *controller* - a FSM that sequences the operation of the datapath. Each of the datapath elements has control lines that determine which function it will perform. In addition, it may have one or more status lines that represent a particular condition. The controller FSM has outputs that connect to the control lines of datapath elements, while status lines become FSM inputs. One metaphor for thinking about how the controller and the datapath work together is that of a puppeteer with a marionette puppet. The puppeteer (controller) pulls the strings to make the puppet (datapath) perform certain functions.

```
MODULE TwoBitCounter
"Inputs
CLOCK, EN PIN;
"Outputs
Q0, Q1 PIN ISTYPE 'REG';

QSTATE=[Q0,Q1];
S0=[0,0];
S1=[0,1];
S2=[1,0];
S3=[1,1];

STATE_DIAGRAM QSTATE
STATE S0: IF (EN) THEN S1 ELSE S0;
STATE S1: IF (EN) THEN S2 ELSE S1;
STATE S2: IF (EN) THEN S3 ELSE S2;
STATE S3: IF (EN) THEN S0 ELSE S3;

EQUATIONS
QSTATE.CLK=CLOCK; QSTATE.OE=1;

TEST_VECTORS
([EN, CLOCK] -> [Q0, Q1])
[1, .C.] -> [0, 0] ;
[1, .C.] -> [0, 1] ;
[1, .C.] -> [1, 0] ;
[1, .C.] -> [1, 1] ;
[1, .C.] -> [0, 0] ;
[0, .C.] -> [0, 0] ;
[1, .C.] -> [0, 1] ;

END
```

Table 9.2: ABEL file for a two-bit counter with enable

To illustrate this idea, consider the following design problem: We wish to design a simple "Ping-Pong" game shown in Figure 9.2. Eight LED's represent the location of the ball on a ping-pong table, while two switches represent the two paddles. Initially, the ball is placed in the leftmost position and the left player "serves" by pressing the "left paddle" button. The ball moves from left to right until it reaches the rightmost LED. If the right player presses the "right paddle" button at the same time that the ball reaches the rightmost position, the ball "bounces" back to the left side. On the other hand, if the right player presses the button too early or too late, he or she "misses" the ball, which goes back to the leftmost position for another serve by the left player. Similarly, when the ball bounces back to the leftmost position, the left player must press the "left paddle" button at exactly the right time to hit the ball back to the right side. If he or she misses, then the ball goes to the rightmost position for a serve by the right player.

Figure 9.3 shows a possible implementation of the ping-pong game using a controller FSM and a datapath consisting of a special-purpose shift register. The outputs of this shift register are tied to

LEDs to display the ball position. Control lines STARTL and STARTR set the shift register to display the ball in the leftmost and rightmost positions, respectively. SHIFTL and SHIFTR control the direction of shifting in the shift register. The FSM has inputs from the left paddle (LP) and right paddle (RP) switches and status line inputs from the datapath that specify that the ball is in either the leftmost (LLAMP) or rightmost (RLAMP) positions.

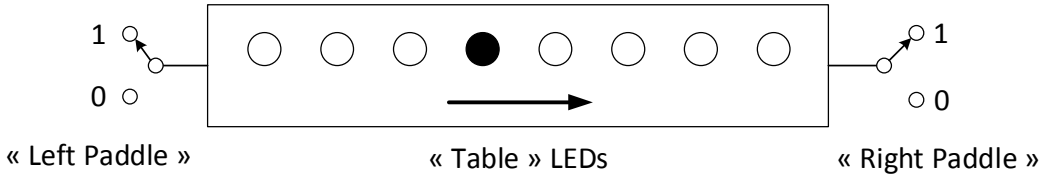


Figure 9.2: "Ping-Pong" game

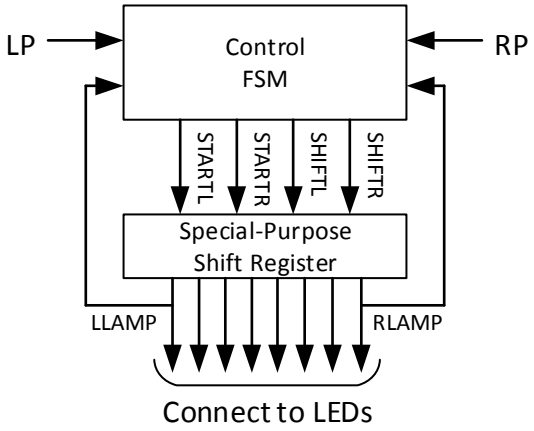


Figure 9.3: Organization of Ping-Pong game implementation

Figure 9.4 shows a state diagram for the control FSM *without* specifying the outputs. There are four states in this FSM. In state STOPL, the ping-pong game is waiting for the left player to serve by pressing the LP button. When this happens, it shifts to the RUNR state, which moves the ball from the left to the right until it reaches the rightmost position (as specified by the RLAMP input). If the right player presses the RP button at exactly this time, it goes to state RUNL to move the ball back to the l eft side. Otherwise, it goes to the STOPL state since the right player "missed the ball". The STOPR and RUNR states work in a similar way, but in the opposite direction.

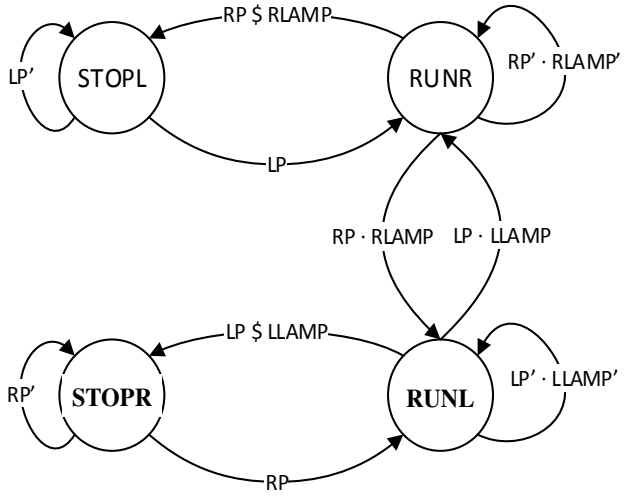


Figure 9.4: State diagram for Ping-Pong game

Preliminary Assignment

1. Review the ABEL file for the universal shift register shown in Table 9.1 and write a "function table" that describes what function it performs for each value of S0 and S1.
2. By using a PLD, design an 8-bit specialized shift register for use in the "Ping-Pong" game. This shift register should have four inputs: STARTL, STARTR, SHIFTL, and SHIFTR. When "STARTL" is true, the shift register should be loaded with the value "10000000" to indicate that the ball starts on the left side of the table. Similarly, when "STARTR" is true, the shift register should be loaded with the value "00000001" to indicate that the ball is on the right side of the table. When SHIFTL is true, the shift register should shift left. When SHIFTR, the shift register should shift right. You should be able to derive the equations for the shift register by inspection after studying Table 9.1. Simulate your ABEL description to show its proper operation.
3. The control line outputs of the FSM are not specified in Figure 9.4. Determine the proper values for the control lines and label the state diagram with the proper output values.
4. By using a PLD, design an implementation of the FSM including outputs. Simulate your ABEL description to show its proper operation.
5. Draw a schematic diagram showing how the specialized shift register and FSM should be connected to implement the state machine. Be sure to show all pin numbers.
6. When power is turned on in a sequential circuit, the flip-flops may hold random values. Explain what will happen to your circuit when it is turned on if it is initialized to a random value.

In the lab

1. Show the instructor your simulation for the specialized shift register and have him or her program the PLD for you. Test the circuit in the breadboard and demonstrate its proper operation to the instructor.

2. Show the instructor your simulation for the control FSM and have him or her program the controller PLD for you. Test the circuit in the breadboard with the specialized shift register and demonstrate the proper operation of the ping-pong circuit to the instructor.

APPENDIX A. ECE GUIDE TO LABORATORY REPORT WRITING

Introduction

This ECE Guide describes how students should prepare laboratory reports for the following undergraduate courses:

- ECE 218 Digital Lab I
- ECE 213 Analog Lab II
- ECE 311 Engineering Electronics
- ECE 312 Electronic Circuits
- ECE 319 Fundamentals of Power Engineering
- ECE 405 Digital and Data Communications with Laboratory
- ECE 407 Introduction to Computer networks with Laboratory
- ECE 411 Power Electronics
- ECE 412 Electric Motor Drives
- ECE 419 Power Systems Analysis with Laboratory
- ECE 423 Microwave Circuits and Systems with Laboratory
- ECE 429 Introduction to VLSI Design
- ECE 436 Digital Signal Processing I, with Laboratory
- ECE 441 Microcomputers
- ECE 446 Advanced Logic Design

Some of these courses require shorter reports than others, and some of them require a special format. Nevertheless, you need to turn in well-structured and complete engineering lab reports. This guide describes the structure of a good engineering report, explains the need for each section, and outlines the content of these sections. It introduces some standard conventions and rules for writing reports of professional quality. It also includes a checklist to help you be sure that your report is complete.

You are expected to follow these instructions and prepare reports in the prescribed format. Reports will be reviewed on the basis of instructions contained in this document. Grades will be strongly affected by the quality of your reports.

Need for Report Writing

According to recent nationwide surveys, engineers spend at least fifty percent of their time writing reports and memoranda. The quality of oral and written reports presented by working engineers is invariably one of the criteria used by their superiors in performance evaluations. The ability to write a good, professional-quality report is an essential, marketable skill. Therefore, all engineering education must include training and practice in report writing. The ECE Department is committed to providing its students with the incentives, opportunities, and guidance to develop this skill.

Advantages of a Standardized Format

This guide describes the various kinds of engineering reports, outlines the sections, and describes the material that should be included in each section. This is a format that has evolved over time in engineering practice. While there is no single perfect format, several very good approaches exist that are broadly similar. By using a checklist and a standardized format, an engineer can ensure that the final report is well-written and complete, and that readers who have different interests and needs can find the information they seek from the report with minimum effort. Using standard formats also cuts down on the time required to write a report.

Engineering professionals write several different kinds of engineering reports. The form, length, content, and emphasis are determined by the purpose of the report and the intended audience[s]. However, all engineering reports are similar, and include sections that describe objectives, methods and procedure, results, and conclusions.

A large number of references are available on engineering report writing. However, every effort has been made to ensure that this guide is the primary reference on report writing that is needed for the ECE laboratory courses.

Intended Readership

Although as a student you can expect that the grader will read your report in its entirety, engineering professionals know that only a few experts in their own field will read their complete report – and only if they continue to be impressed by the relevance of each individual part of the report. An engineering report prepared in a typical organization is read by a number of different people, with differing backgrounds, interests, and needs. Different parts of a report may be read by different people.

> Some individuals might be interested in the comparison of theory with experimental results. Others might be interested in the significance of the results and the conclusions that are drawn from them. Still others would want to know the calculations and both the accuracy and the precision of the results.
>
> At yet another level, a manager who seeks specific information to make important decisions about a project might look at only a brief summary of the report, together with the conclusions and recommendations.
>
> Engineering information is also frequently reported to non-technical readers who need to assess the impact of the report without having to understand all the technical details.
>
> Senior executives in a company frequently look for a summary of the report that extracts the information of principal importance, and usually for the conclusions and recommendations.
>
> Someone next in the chain of command would probably examine the report more closely for a description of the results.
>
> Only those with a direct interest in or immediate contact with the project or a similar project elsewhere would read the report completely.

Professional laboratory reports are written to meet the needs of all these individuals. Because they are an important part of your pre-professional training, laboratory reports written for ECE courses should also be written to satisfy the needs of this varied readership. Thus you will find that some repetition of information in different sections of the report is often necessary, perhaps with a difference in emphasis or detail.

As in all professional writing, clarity and precision in both language and calculations are essential in an engineering report. Figures, charts, tables and graphs should be used whenever they would be helpful. They should be labeled with an identifying number as well as a title. This Guide describes a report structure that satisfies these various requirements.

Time Required for Report Writing

Students in laboratory courses often feel that an excessive amount of time is needed for the preparation of laboratory reports, and that the return on this investment of time (in terms of the GPA) is inadequate. In fact, students who can report on their laboratory work in clear, organized reports receive higher grades than those who cannot. While report writing can indeed be time intensive, the time is well spent, because it provides you with the opportunity to develop or improve a skill that will be extremely valuable in your future career. Frustrations and problems related to report writing can be minimized by proper planning. Try to schedule your weekly activities to allow enough time for writ-

ing laboratory reports. As you become more proficient with lab reports, the time required decreases. Remember: quality and completeness is much more important than quantity and excess wordiness.

Use of Computer-based Word Processors

All laboratory reports for ECE courses must be prepared using a computer-based word processor. This is the standard practice today in most organizations. PC-based word processors with printers are available for use throughout the campus. Most word processors have useful features that significantly enhance the capability to produce a professional-quality report. These features include formatting, outlining, and spelling and grammar checkers.

Using a word processor and one standard format in all the laboratory courses will increase your writing efficiency. For longer reports, time is usually spent most efficiently by working on a report in more than one session. A rough draft is written first and set aside. The rough draft is then reconsidered, edited and polished into the final version in one or more revisions. The final version must be proofread carefully before submission. Allow time to write, edit, and proofread the reports before the final versions are printed. You are professionals whose successful careers will be based in part on how well you can communicate in writing. Start practicing now!

Language and Style

As with all other modes of communication, engineering reports are most effective if the language and style are selected to suit the background of the principal readers. Engineering reports are judged not only on technical content, but on clarity, structure of the report, ease of understanding, word usage, and grammatical correctness.

In general, reports should be written in the third person, past tense, in an impersonal style. The entire report should be written in continuous prose: don't expect figures or equations to serve where sentences and paragraphs are needed. For example “Figure 1 shows the voltage-current relationship of the XYZ diode.”

As you edit your report, delete unnecessary words, rewrite unclear phrases, and clean up grammatical errors. Use separate headings for each section. Allow space between sections. Place tables, schematics, or graphs logically, label and number them clearly, and execute them neatly. Aim for a clear, easy-to-read, professional-looking report, with the **essential** information contained therein. [If necessary, put additional material in an appendix.]

Laboratory Report Checklist for ECE Courses

A checklist for laboratory reports is attached to this document. It provides a list of items that must be included. The grader will expect you to follow the checklist carefully. Significant penalties may be imposed for failure to do so.

The length or volume of a report does not determine your grade. Your grade will be based on your adherence to the prescribed format, technical correctness, completeness of the report, completeness of each section therein, overall clarity and conciseness, writing mechanics and style, and timely submission.

Outline of ECE Laboratory Report

An outline of a typical laboratory report is also attached. Use it along with the checklist to make sure that your report is complete, correct, and in the prescribed format.

Grading

Laboratory reports will be graded separately for technical content, and for writing quality and style. The weight of each component in computing the overall score for the report will be determined by the course instructor. The Teaching Assistant for the course will assist in the conduct of the laboratory experiments and grade the reports for technical content and writing quality. In most ECE lab courses, experiments are usually carried out by groups of

students. It is expected that each member of a group will follow an identical procedure in the laboratory and use the same set of data. Members of a group are also encouraged to discuss the analysis of data with one another.

OUTLINE OF ECE LABORATORY REPORT

1. **Cover page**
    a. Your name
    b. Your laboratory partners' names
    c. Course number and section
    d. Your TA's name
    e. Experiment number and title
    f. Date of experiment
    g. Due date of laboratory report

Depending upon the course, you may each turn-in either an individual lab report or a group lab report. If allowed by your Instructor and you elect a group Lab Report, a statement on the cover page signed by each group member, must attest **"that every member of the group has materially contributed to the intellectual content of this report".** This means that one student is not allowed to "do the work" while the other student's contribution is to type or otherwise prepare a neat copy.

2. **Introduction**
    a. Purpose What is the purpose of this experiment?
    b. Scope What are the limitations of this experiment?
3. **Theory**
    a. What is the theoretical basis of this experiment? [Brief statement]
    b. What preliminary work was required before the lab session itself?
4. **Experimental procedure**
    a. Schematic: Neatly drawn schematic, block diagram, etc. of the circuits used in the experiment. Include the test equipment.
    b. Procedure: Outline the methods used in the experiment.
    c. All apparatus:
        - Manufacturer's name
        - Model number
    d. Results:
        - Original data sheet: Include date, experiment number, and signatures of the students running the experiment.
        - Graphs
        - Sample calculations
5. **Interpretation**
    a. Compare your results with the expected results.
    b. Discuss possible sources of error.
6. **Conclusions**
   What can you conclude from the results?

7. **Appendix** [If necessary]