LABORATORY MANUAL

ECE 2090 – Logic and Computing Devices

Clemson University

Department of Electrical and Computer Engineering

Clemson, SC 29634

Compiled August 2009

Table of Contents

ECE 2090 – Lab Introduction

PURPOSE

To familiarize students with the basis of safety, lab procedures, and the equipment to be used throughout the course.

EQUIPMENT

ECE 2090 Lab
Kit

REQUIREMENTS

Each student should read and understand this introduction prior to the first lab meeting.

SAFETY

Whenever electricity is used in an experiment, some danger exits. This should always be on your mind. Most of the experiments in this course will use only low voltages, which are not inherently dangerous. However, it is possible to incorrectly wire almost any experiment such that dangerous voltage levels result. Subsequent lab courses require the use of high voltages. There is only one way to prevent accidents.

THINK

Plan what you are going to do. Understand what you are being asked to do. Ask questions. ***Never turn the power on until you are confident that everything is safe!***

The careless use of electricity can have two results. It can hurt you. It can hurt equipment. Obviously, you want to avoid hurting anything.

To project yourself, always treat electricity with respect. Don't handle "hot" lead wires.

Don't leave wire dangling about in space. An old rule of thumb is to keep one hand in your pocket at all times. This hopefully prevents the flow of electricity from one hand to another, potentially causing the heart to stop. Keeping one hand in your pocket also causes cramps, and decreases efficiency, so no one does it. But keep in mind that when you touch a wire that is electrified, the electricity will always want to flow, and it is to your advantage to keep it from flowing through you. Don't hold electrified wire, and make sure that no part of your body inadvertently comes into contact with a wire.

To protect equipment, make sure that all of the hookups between power supplies, oscilloscopes, volt-ohm meters, light emitting diodes (LED's), protoboards, chips, etc., are as you want them to be. Double check all wiring prior to turning on the power. One way to destroy an Integrated Circuit (I.C.) is to reverse power leads. You can tell because the abused I.C. will start smoking or will get so hot you cannot touch it. Then throw away the I.C.

When probing a circuit with a test lead, make absolutely certain that you are making contact with the exact points you wish to test, and ONLY those points.

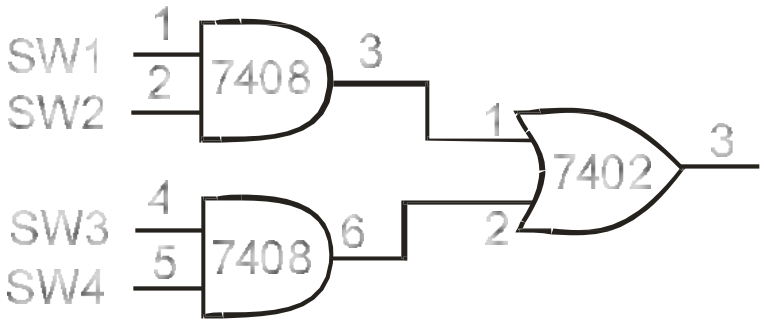
When using a particular piece of test equipment, understand the limitations of the equipment. Don't try to use it to test something it wasn't designed to test. Read the manual.

In summary, the way to keep yourself and the equipment from being damaged is to do everything slowly, cautiously, and carefully.

GOOD LAB PROCEDURES

In order to make your experiments go easier, there are some procedures which should be followed. Most are generally common sense.

- Before wiring a circuit, a circuit diagram should be drawn and simulated. This diagram should include pin numbers as shown below.



  You can then check off each connection as it is wired. For example, after the top `AND` gate is connected to the `OR` gate on the right, the line connecting them in the diagram should be checked off as having been wired.

  Without some system like this, it is very easy to forget which gate is which within a system. *If you don't have a properly labeled circuit diagram prior to the beginning of lab, your instructor will give you a zero for lab performance.*

- While wiring, rewiring, etc. *turn off the power*. This prevents the application of power to the circuit in unwanted places. Violation of this rule will result in decreased lab performance grades. The voltages used in this lab are generally on detrimental to the chips, not students. But it is very important to learn proper safety techniques *before* you enter the higher-voltage experiments of other classes.

- Try to avoid messy "rat's nest" wiring. It is almost impossible to trouble-shoot a messy wiring job. It is also hard to make changes to a disorganized board. Keep lead wires as short as possible, and make neat, flat bends. It is also smart not to wire across the top of the IC's.

- Wire the power leads of chips first, using a color scheme if possible. Traditionally, in DC applications

  RED = +5 Volts and **BLACK** = Ground

  *(Notice that this is different from AC applications where black is **hot**, white is **neutral**, and green is **ground**.)*

  You can expand on this, e.g. - use yellow wires for inputs and green for outputs. This makes trouble-shooting much easier.

- Wire circuits carefully. It is easier to wire it right than to spend hours tracking down an error.
- Make certain fragile leads are not bent excessively. They will break off after being bent back and forth several times.
- Observe polarity markings on equipment and components.
- Handle equipment carefully. *Don't drop anythin*g.
- Put all components back where they are supposed to be.
- Before leaving the lab, check that your bench position is neat and orderly, and check the floor for wires. Report any defective equipment to the instructor. Turn off all equipment, and make sure the bench power is off.

EQUIPMENT

The proper care of the equipment used in this lab is essential. If handled improperly, many of the devices used to perform the experiments will give erroneous readings or fail to operate altogether.

Protoboard

Your kit includes a plastic board used to wire together electric circuits. This is called a breadboard or a protoboard, since it is used to prototype circuits.

Figure 1 below shows how the terminals are connected internally inside the board. The horizontal connections X and Y are called buses and are usually for power and ground.

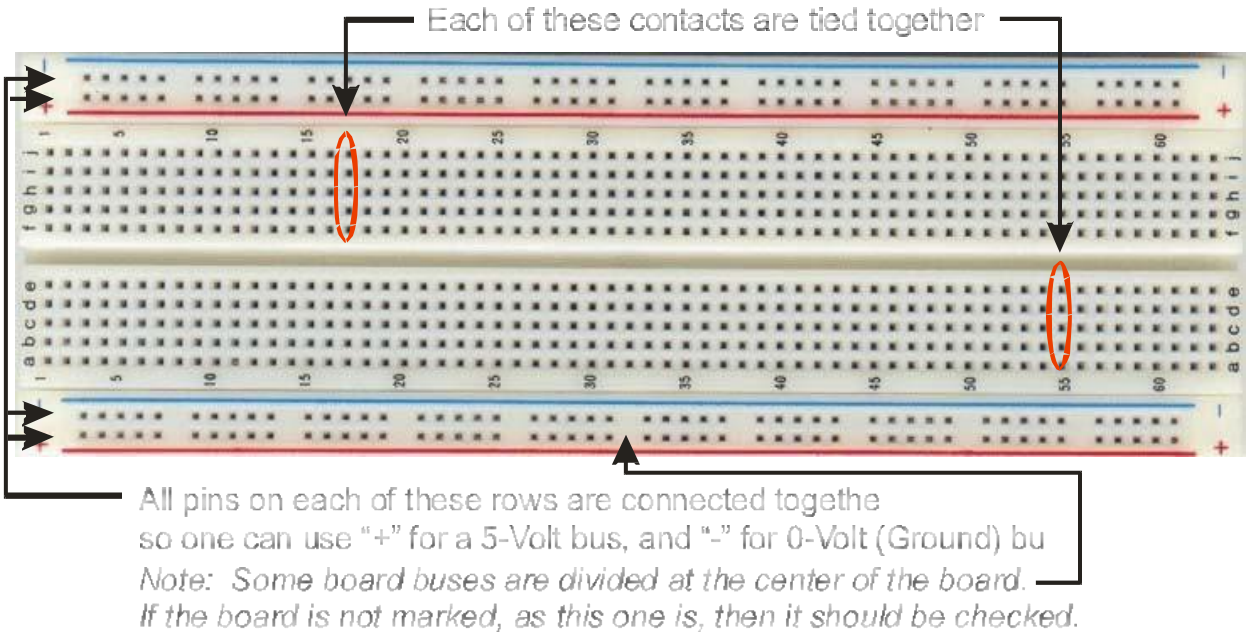


Figure 1. Protoboard Connections

Usually, the top row is connected to the +5V power supply and is called the power bus. The bottom row is connected to an external ground and is called the ground bus.

Figure 2 shows how the power and ground pins of IC1 and IC2 can be connected together by the red wires and black wires. It also shows how the output of pin 3 on IC1 can be connected to the input pin 1 of IC2 with a green wire.

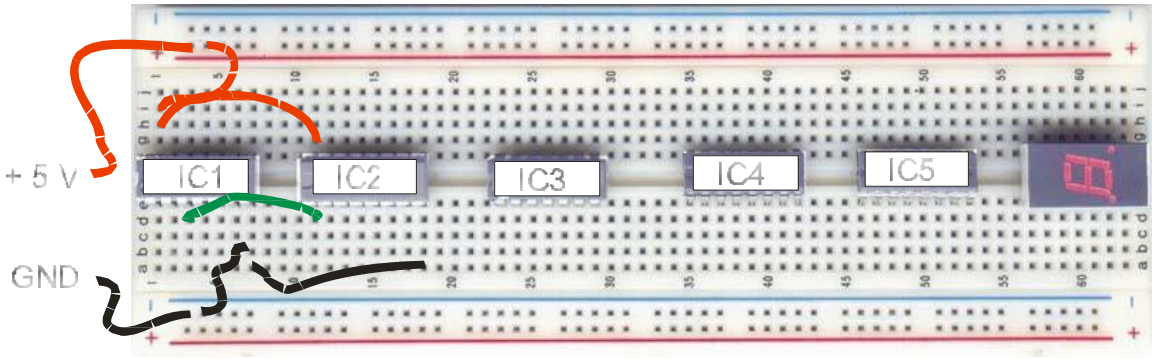


Figure 2. Connecting Wires

Special care is needed when using the protoboard. If a wire which is too large is forced into one of the holes, that particular contact point most probably will be damaged. As a result, the next time the protoboard is used and a wire is inserted into that hole, the wire may not make contact with the internal connection strip and the circuit will not operate properly.

An even more aggravating situation is when the wire makes contact only part of the time. This is called an "intermittent fault." Since the circuit will operate correctly part of the

time, and then mysteriously fail, this type of fault is very hard to find. A good rule to follow is: if the wire doesn't go in easily, find another wire. Pay particular attention to components such as resistors, capacitors - etc., since many have lead with diameters which allow insertion into the protoboard, but still cause damage to the contacts.

To avoid damage, do not insert wires too far.

You will have to strip the wire leads before using them to interconnect circuits on your protoboards. The proper way to do this is to use a pair of wire cutters to carefully strip ¼ inch of insulation off of each end of the wire, taking care not to nick the copper wire. If you take more than a quarter inch off, you risk having wire exposed above the protoboard, which will cause a short if it touches another lead or IC pin. If you cut less that a quarter off, the wire may not make a good connection within the protoboard hole. It also helps to cut the ends of the wire at an angle, which produces a point on the end of the wire. The wire will then slide into the protoboard easier.

NI ELVIS II

You will be using the *NI ELVIS II* workstation equipped with Freescale board during this lab. Each Freescale board has two protoboards in the middle. It is recommended that you place your protoboard on top of the one that is connected to the *NI ELVIS II (Freescale board)*. This way, you can use all of the functions provided by the *NI ELVIS II*, and still be able to pick up your protoboard when the lab is finished with all connections intact. Also, you should pre-wire the circuit called for by each lab in order to save valuable lab time.

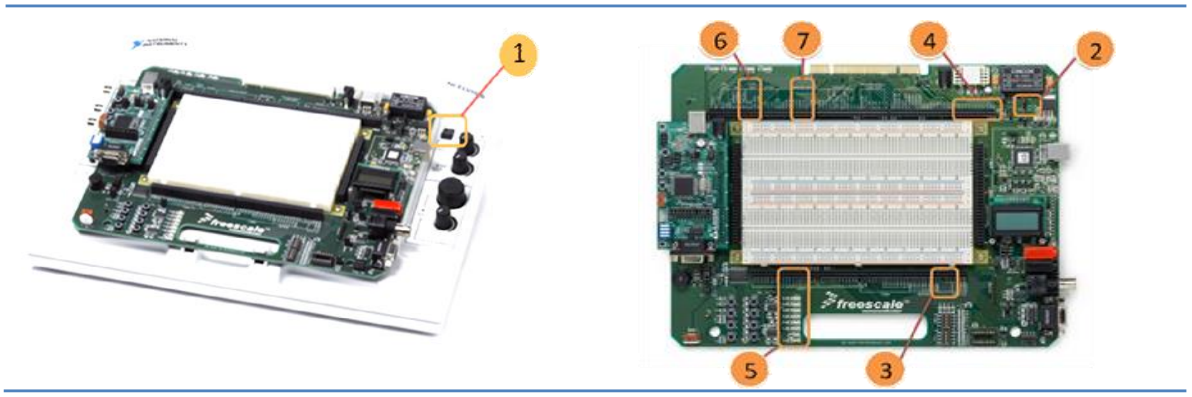
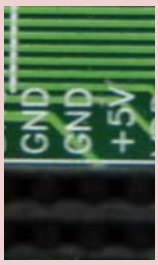
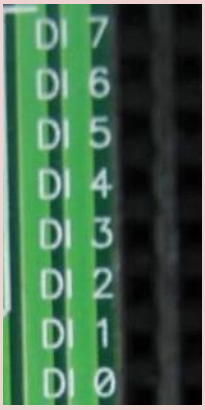


Figure 3. NI ELVIS II

There are several features of the *NI ELVIS II* workstation and Freescale board; however, we will just list and explain only those features that are required for this lab. These features are numbered 1 through 7 on Figure 3, and are briefly explained as follows:

1 **On-Off Switch**		Turns the power to the *NI ELVIS* board on and off. The power LED lights up when the switch is turned ON.
2 **PWR_SEL Jumper**		This allows to set the power input to the board by USB or Power Supply. We will set this jumper as shown in the adjacent figure to input 5V power to the board using the power supply.
3 **Power Supply**		Provides +5V and ground to the tie points associated with them. Therefore, if you connect a wire from one of the +5V holes on the *NI ELVIS (Freescale)* board to the top row of holes on your protoboard, all of the top row will be energized to +5V. We will use only +5V and Ground in this lab.
4 **Logic Inputs**		Digital inputs used to provide either +5 V or GRD to the associated tie points. You could move a wire between +5 V and GRD to provide the same function, but it is much easier to use the switches when a lot of changes are going to be required. These switches can be programmed. In this lab, we will program these switches using NI ELVIS software, as shown later, to give digital inputs to the protoboard. A wire from one of the holes of the switch in use is connected to the specific point on the protoboard. How to control these switches will be shown later.

5

Lamp Monitors

LED's that can be used to indicate ground (light off) or +5V (light on). You can insert small wires into the corresponding holes of the LED's and then insert the other end into a specific point on the protoboard.

6

Function Generator

Function Generator (FUNC GEN) is used to input different waveforms to the specific point on the protoboard. For instance, you can use FUNC GEN to input a square waveform of a desired frequency. This waveform can be described as a +5 V to ground to +5 V sequence (or vice-versa) on a wire. You could do the same thing by moving a switch (described above) from one position to the other and then back. However, the switch is a mechanical device and does not make each transition smoothly. The switch actually "bounces" on its contacts causing a series of pulses. The FUNC GEN performs the same function electrically, and are, therefore, called "de-bounced". A wire from one of the holes of the FUNC OUT is connected to the specific point on the protoboard. How to generate a waveform will be shown later.

7

Analog Inputs

Analog inputs holes are used to give inputs to an oscilloscope. These inputs will be used only if you intend to see the generated square waveform on the NI ELVIS virtual scope.

USING DIGITAL INPUTS SWITCHES AS LOGIC INPUTS

As mentioned above digital inputs are used to provide either +5 V or GRD to the associated tie points. The NI ELVIS board allows you to control these switches using NI ELVISmx Instrument software. Open the "NI ELVISmx Instrument Launcher" on the PC connected to the NI ELVIS board. You will see a GUI as shown in Figure 4.

Figure 4. NI ELVISmx Instrument Launcher

Click on the "DigOut" button. Another GUI, "NI ELVISmx Digital Writer", will pop up as shown in Figure 5.

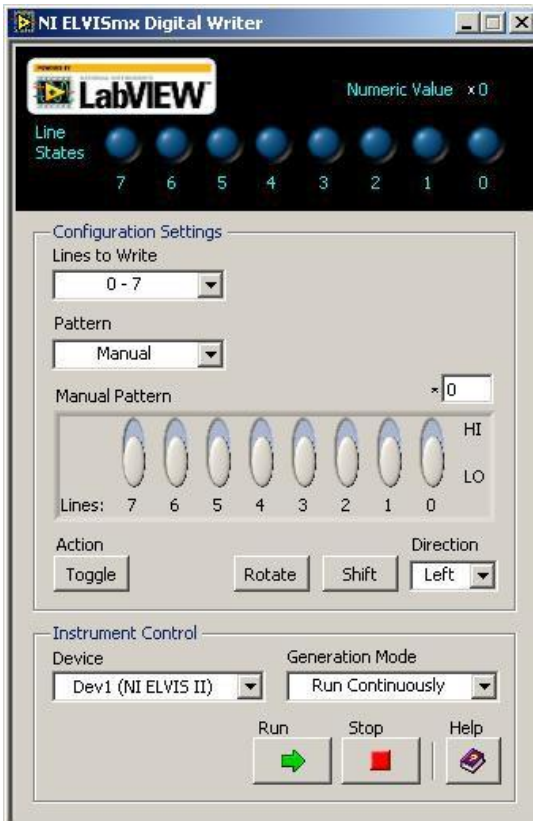


Figure 5. NI ELVISmx Digital Writer

You can now select the lines you wish to write using the drop-down menu under Configuration Settings. Leave the Pattern as "Manual" as this will allow you to manually switch between HI and LO for a specific line (or input). Hit the green arrow button to run the GUI. Now, you can manually switch between HI and LO inputs for any number of input lines you selected (0 - 7). For instance, if you select to write 0 – 7 lines and make

all the 8 inputs HI, DI 0 – DI 7 (circled as number 4 on Figure 3) will be set to HI (or logic 1) on the NI ELVIS board. A particular point on the protoboard connected to any of these DIs will receive a HI input.

USING FUNCTION GENERATOR

To generate a square or pulse waveform, press the "FGEN" button on the NI ELVISmx Instrument Launcher. A GUI as shown in Figure 6 will pop up.

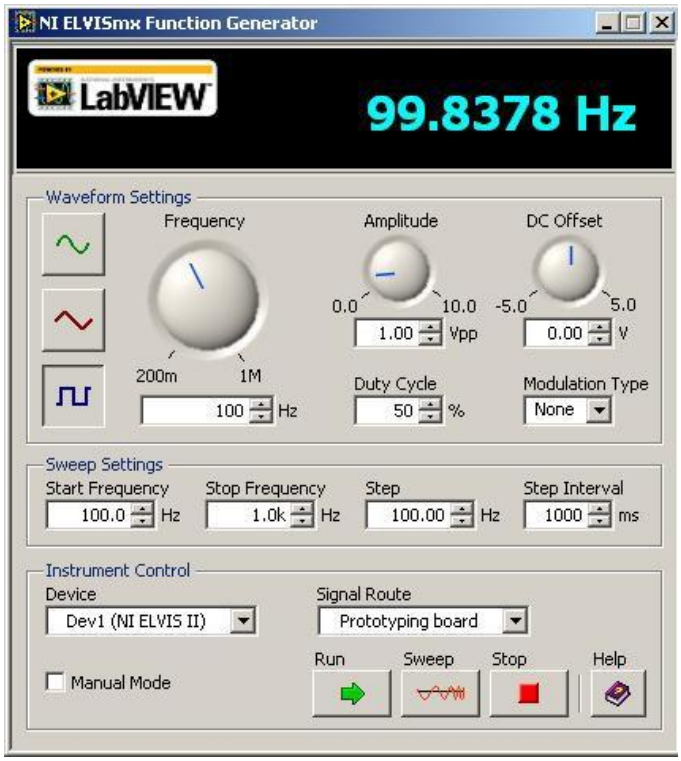


Figure 6. NI ELVISmx Function Generator

Press the square waveform button (third button given in blue under Waveform Settings). Set the Amplitude to 5 Vpp, and keep DC Offset as zero. Press the "Run" button, and you will get the desired waveform on the FUNC OUT hole on the NI ELVIS board. You can adjust the frequency using the GUI. You can also use the Sweep Settings to vary the frequency automatically with a predetermined step. For instance, if you want to start a waveform with a frequency of 100 Hz, increment the frequency by 100 Hz after every 10 seconds, and stop the waveform generation when the frequency reaches 1 kHz, you can use the sweep settings as shown in Figure 6. Press the "Sweep" button, and you will get the desired waveform on FUNC OUT hole on the board.

USING OSCILLOSCOPE

Additionally, you can use the oscilloscope tool to watch the waveform that you generated using the Function Generator. Connect a wire from FUNC OUT hole to one of the

positive Analog inputs (for example, ACH0+), and connect ACH0- to the GND hole on the board. Press the "Scope" button on the NI ELVISmx Instrument Launcher, an oscilloscope GUI will pop up. Select the channel and the source, and press the "Run" button, you should see the waveform output of the FUNC OUT as shown in Figure 7.

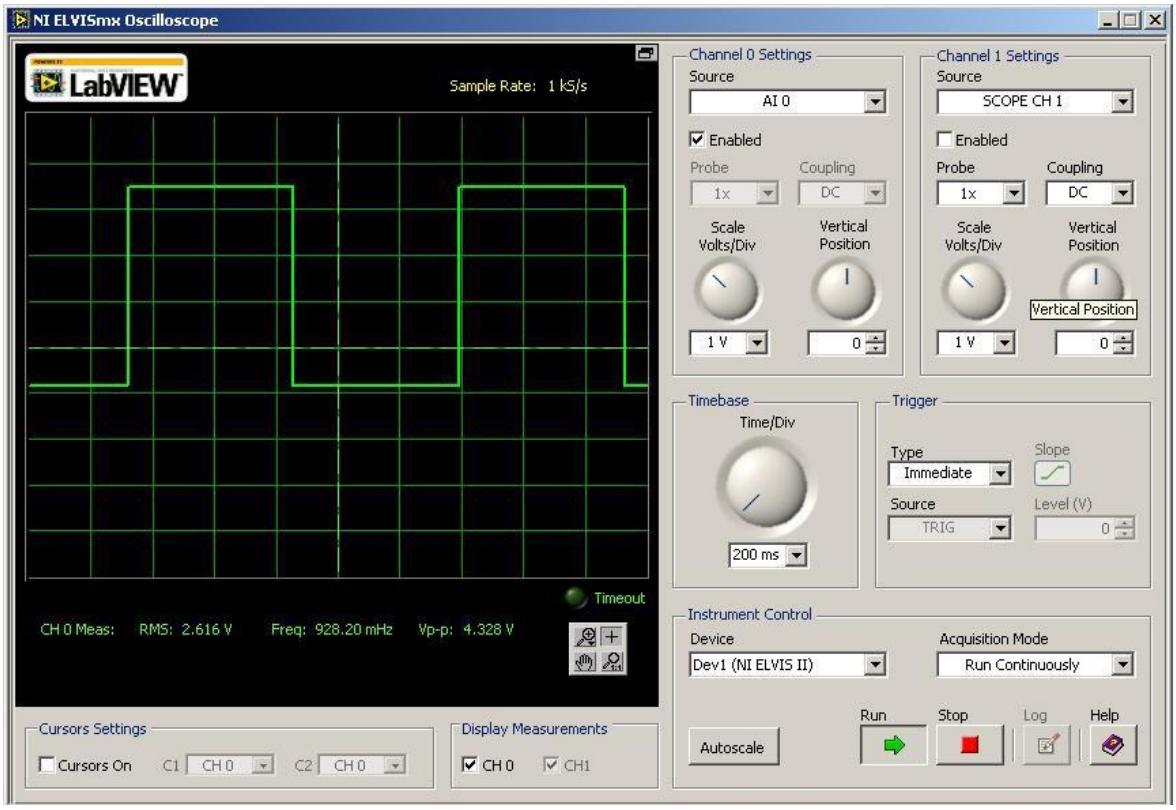


Figure 7. NI ELVISmx Oscilloscope

The *NI ELVIS II* is a powerful (and expensive) design aid. Do not abuse it. Any questions you may have should be directed to the instructor.

Integrated Circuits (IC's)

The IC's supplied with the kit are used in a variety of experiments. Each looks similar to the one shown below:

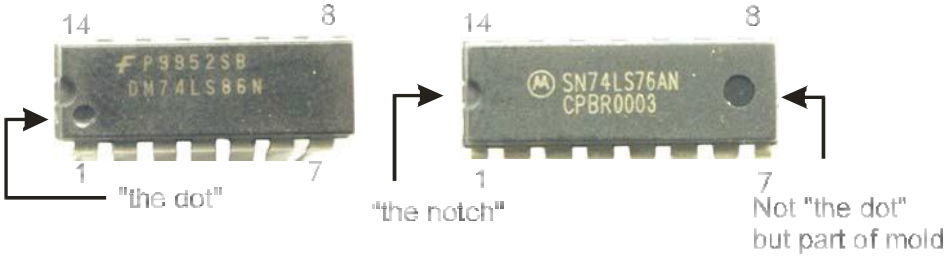


Figure 8. IC Pin Numbering

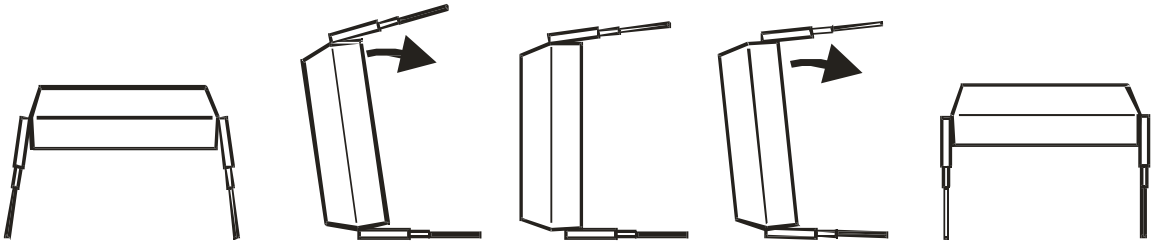
Note the numbering system, which progresses counter-clockwise from pin 1. Pin 1 is designated by a dot or a notch Usually on TTL logic (the type of chips supplied with the kit), the last pin is to be connected to the +5 V supply, and the pin diagonally opposite is connected to ground. These are pins 14 and 7, respectively, for 14-pin IC's and 16 and 8 for 16-pin IC's.

There are some exceptions to this rule, so always check!

Special caution should be taken when inserting and removing IC's from the protoboard.

When shipped from the factory, the leads (legs) of the IC's are slightly bent apart to aid in machine insertion.

It is necessary to straighten the legs prior to insertion into the protoboards. This can be done easily by flattening the legs on a table top as shown below. Ask you instructor to show you how this is done if you need help.



From factory. **Bend one side, then the other, till legs are parallel.**

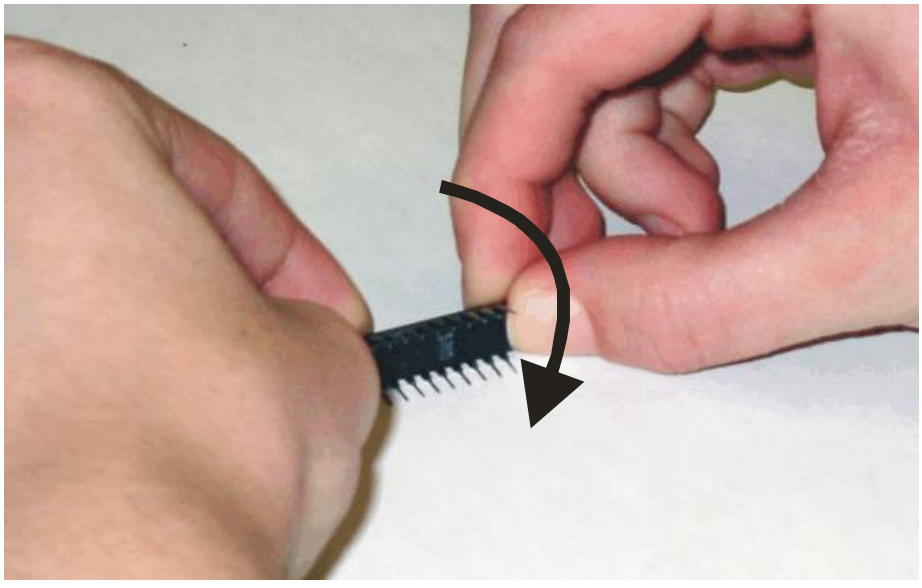


Figure 9. Straightening IC Legs

When you remove a chip (IC) from the protoboard, it is *very easy* to bend the legs by not exercising caution. You should not simply pull on the chip with two fingers to remove it.

One end will invariably rise before the other causing the legs on the other end to bend. It may even result in a puncture wound to one of your fingers because the legs of the IC's are very sharp. *Be careful*! If an IC extractor is not available, you should use the tip of a pen or pencil to gently pry up the legs on one end of the chip, and then the other, as shown below.

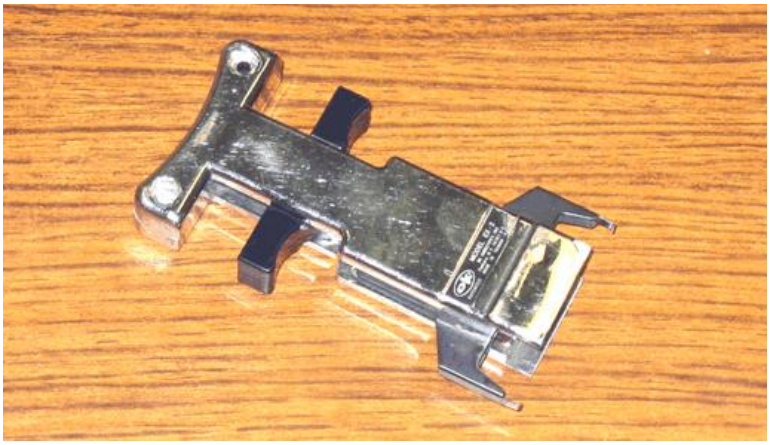


Figure 10. An IC Extractor

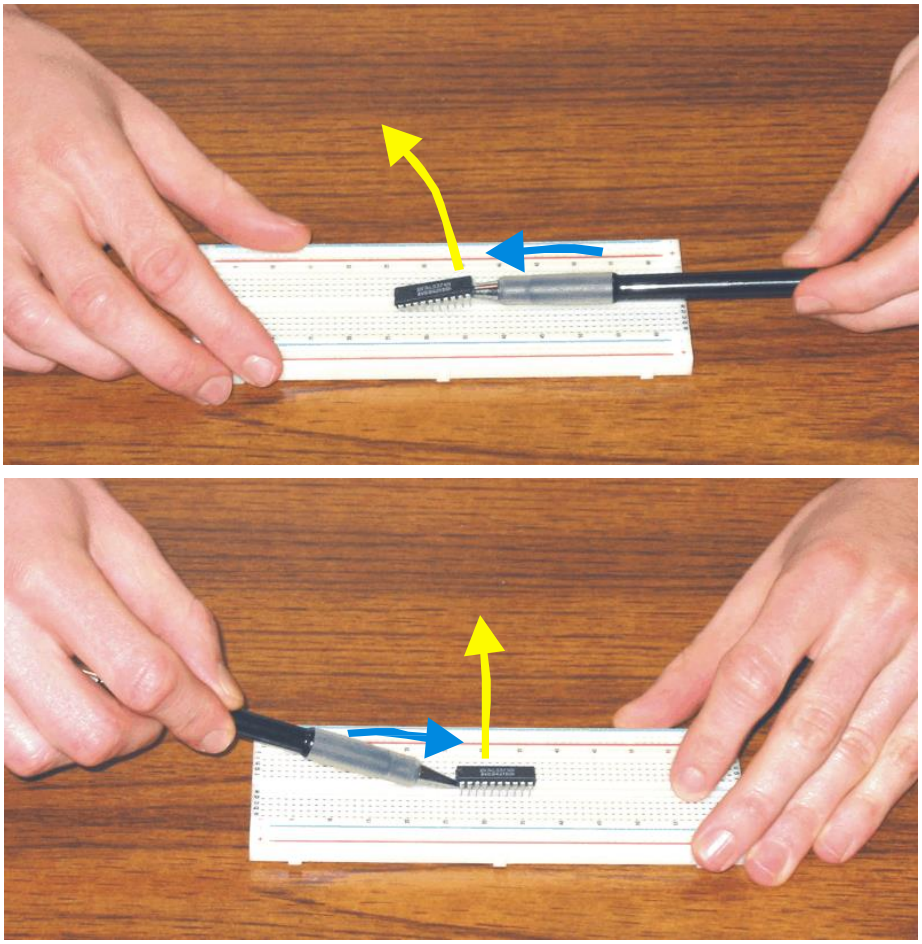


Figure 10. Removing an IC

Again, ask your instructor to show you if you need some help. Usually, if a leg is bent twice, it will break off easily and the IC is virtually useless so.

LAB REPORTS

Each report should, as a minimum, include:

- Objectives
- Circuit diagram/Simulation
- Explanation of circuit operation
- Results
- Conclusions and suggestions

Failure to include these lab sections will result in a reduced report grade.

WORKING WITH THE SIMULATOR

See the ***Logisim Getting Started*** guide for an introduction to the simulator software before your first lab.

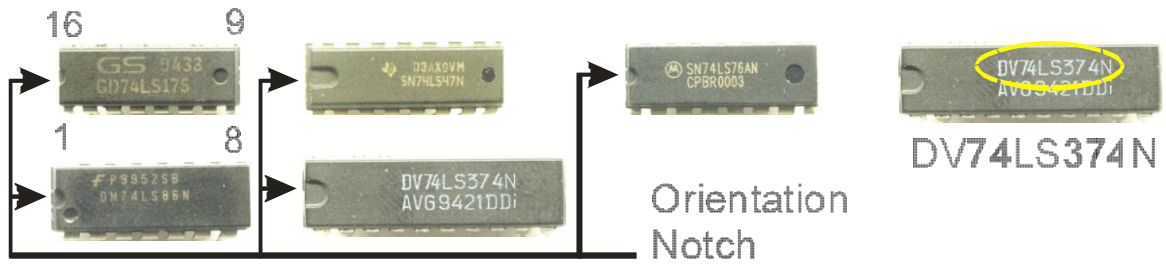
CONCLUSION

The labs you are going to perform are fun! Take your time. Ask lots of questions so that you can learn as much as possible. You've paid for it. Understand everything that an experiment can show you. Try variations of circuits. The amount of information you learn from these labs is directly proportional to the amount of thought and effort you put into them.

If you have any suggestions on how you could make this lab more informative or interesting, please see the professor in charge.

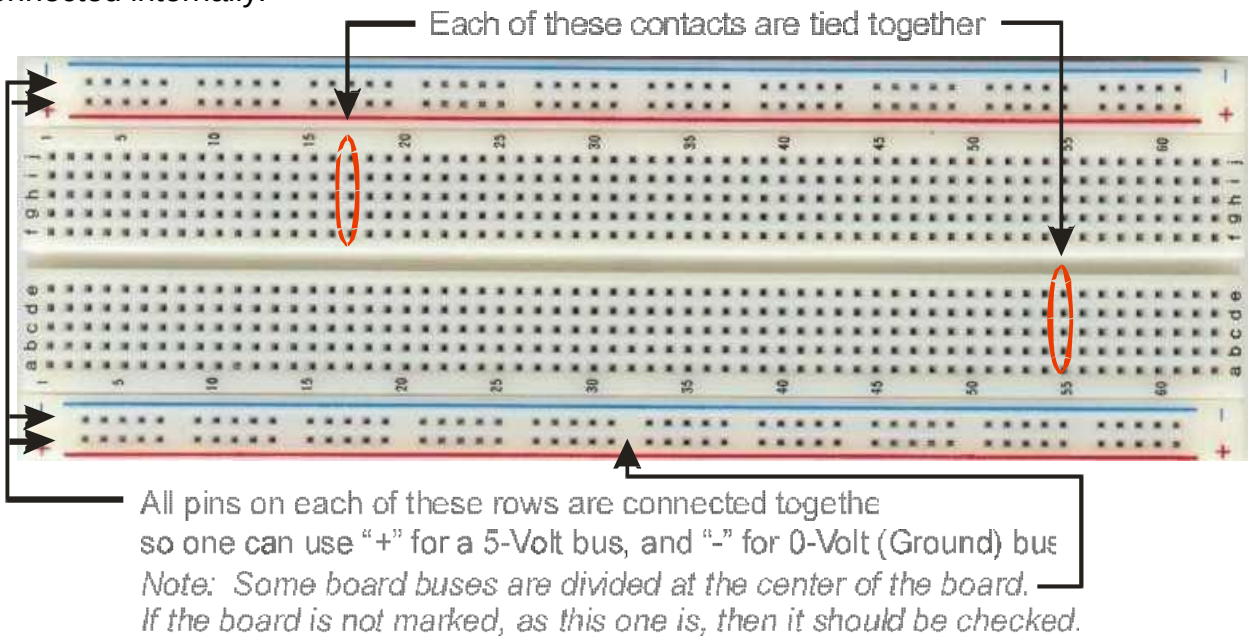
Bill Reid, Dan Stanzione, and Nitendra Nath

To interconnect your Integrated Circuits (IC's) properly, the pin numbers must be known. Each chip has an orientation "notch" which defines where pin 1 is located. When the notch is on the left, pin one is on the bottom left. The pins are numbered counterclockwise from 1.

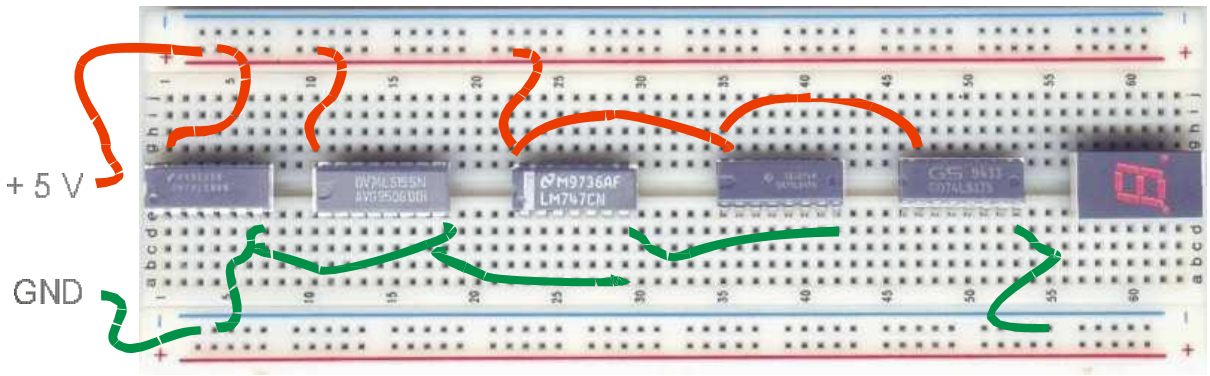


Each chip has a part number stamped on it which gives the chip's manufacturer, function, and performance. As shown above, the chip DV74LS374N is a 74374 logic chip which is an "octal D-type tri-state flip flop." The LS means it is a "Low-power Schottky."

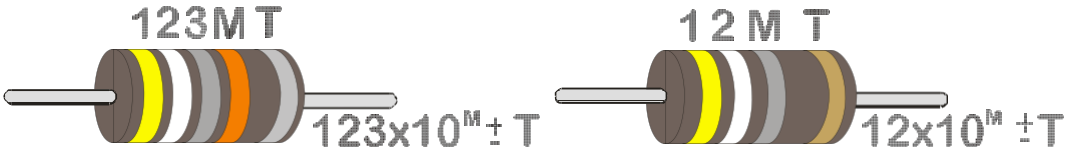
The "Breadboard" contains a grid of holes in which wires can be inserted to make an electrical connection with the pins ("legs") of the IC. The diagram below shows how contacts are connected internally.



Chips should be placed across the center groove of the breadboard as shown below:



Resistors and Capacitor Codes



Band 1	
Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

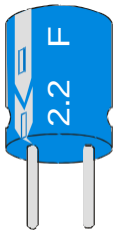
Band 2	
Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

Band 3	
Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Gray	8
White	9

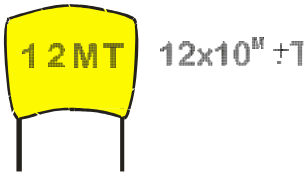
Multiplier	
Black	10^0
Brown	10^1
Red	10^2
Orange	10^3
Yellow	10^4
Green	10^5
Blue	10^6
Silver	10^{-2}
Gold	10^{-1}

Tolerance	
Brown	1%
Silver	10%
Gold	5%

Capacitors come in many shapes, sizes, and types. Some large electrolytic capacitors have the explicit value printed on them.



Some capacitors use the following code shown below, where M is a multiplier and T is the tolerance. The table below shows the values for the given code numbers and letters.



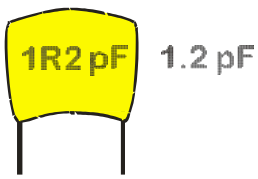
151 K = 15˙10 = 150 pF

759 = 75˙0.1 = 7.5 pF

M	Multiply By
0	1
1	10
2	100
3	1000
4	10,000
5	100,000
8	0.01
9	0.1

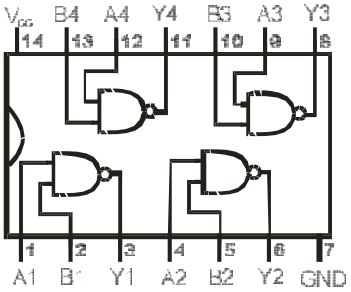
:5 10 pF	Letter	> 10 pF
0.1 %	B	
0.25 %	C	
0.5 %	D	
1.0 %	F	1 %
2.0 %	G	2 %
	H	3 %
	J	5 %
	K	10 %
	M	20 %

Sometimes the letter “R” may be used to signify a decimal point:



or 2R2 = 2.2 (pF or F)

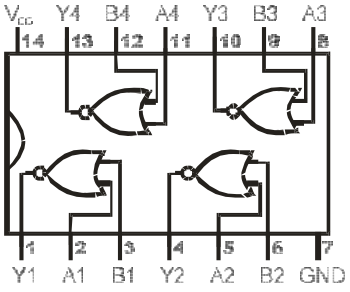
7400 Quad 2-Input NAND Gate



Y = (AB)′

Input		Output
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

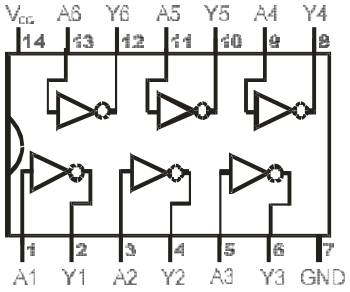
7402 Quad 2-Input NOR Gate



Y = (A+B)′

Input		Output
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

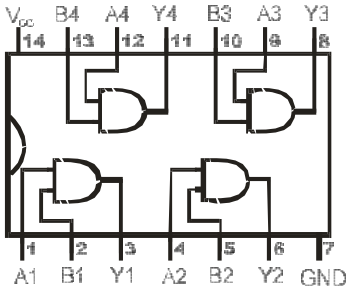
7404 Hex Inverter



Y = A′

Input	Output
A	Y
L	H
H	L

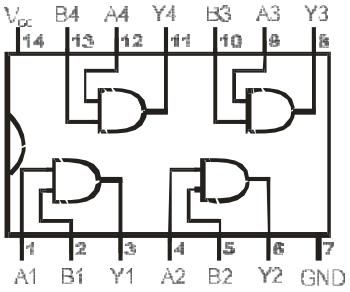
7408 Quad 2-Input AND Gate



Y = AB

Input		Output
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

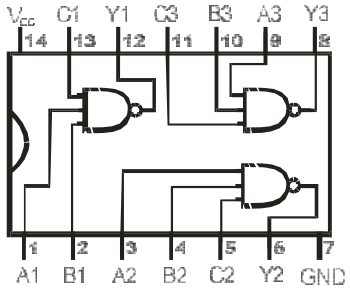
7409 Quad 2-Input AND Gate (Open Collector)



Y = AB

Input		Output
A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

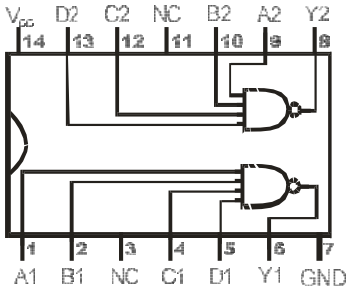
7410 Triple 3-Input NAND Gate



Y = (ABC)′

Input			Output
A	B	C	Y
X	X	L	H
X	L	X	H
L	X	X	H
H	H	H	L

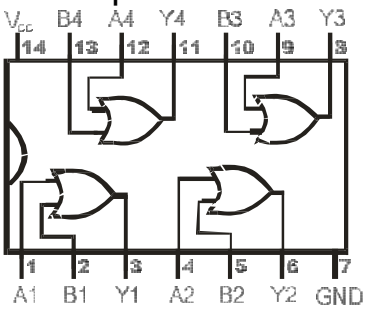
7420 Dual 4-Input NAND Gate



Y = (ABCD)′

Input				Output
A	B	C	D	Y
X	X	X	L	H
X	X	L	X	H
X	L	X	X	H
L	X	X	X	H
H	H	H	H	L

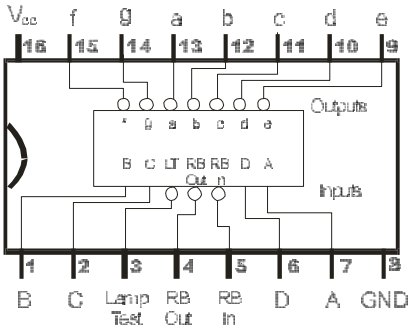
7432 Quad 2-Input OR Gate



Y = (A+B)′

Input		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

7447 Binary-to-BCD Converter

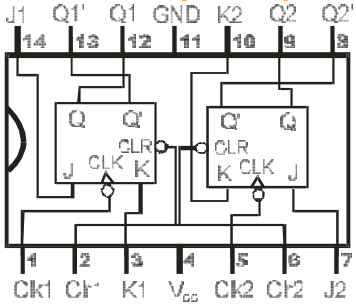


MOTOROLA

TEXAS INSTRUMENTS

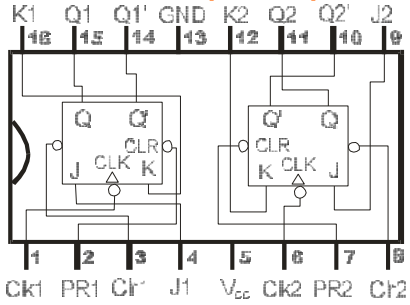
FAIRCHILD SEMICONDUCTOR

7473 Dual J-K Flip-Flop



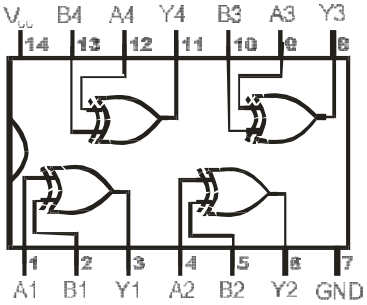
Inputs				Outputs	
Clr	Clk	J	K	Q	Q′
L	X	X	X	L	H
H		L	L	Q_0	Q_0'
H		H	L	H	L
H		L	H	L	H
H		H	H	Toggle	
H	H	X	X	Q_0	Q_0'

7476 Dual J-K Flip-Flop



Inputs					Outputs	
PR	Clr	Clk	J	K	Q	Q′
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H	H
H	H		L	L	Q_0	Q_0'
H	H		H	L	H	L
H	H		L	H	L	H
H	H		H	H	Toggle	

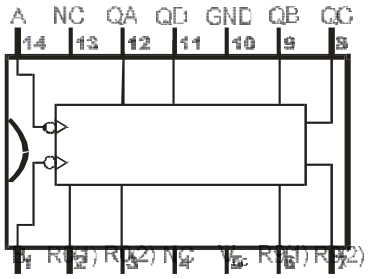
7486 Quad 2-Input XOR Gate



$Y = A \quad B = A'B + AB'$

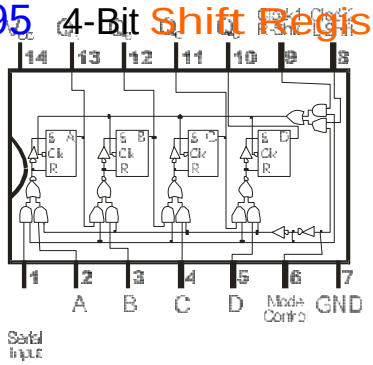
Input		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

7490 Decade and Binary Counter



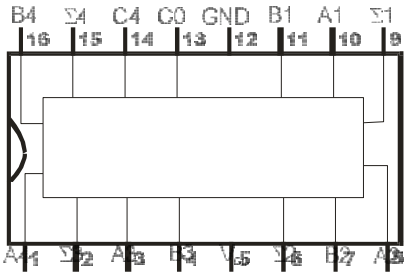
Inputs				Outputs			
R_01	R_02	R_91	R_92	J	K	Q	Q′
H	H	L	X	L	L	L	L
H	H	X	L	L	L	L	L
X	X	H	H	H	L	L	H
X	L	X	L	Count			
L	X	L	X	Count			
L	X	X	L	Count			
X	L	L	X	Count			

7495 4-Bit Shift Register



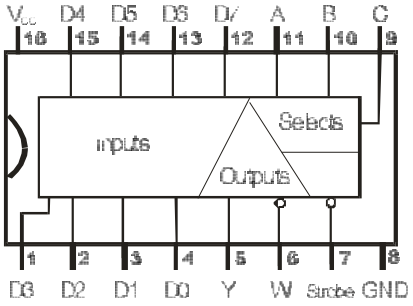
Operating Mode	Inputs					Outputs			
	S	CP1	CP2	DS	PN	Q0	Q1	Q2	Q3
Shift	L	↓	X	L	X	L	Q0	Q1	Q2
	L	↑	X	H	X	H	Q0	Q1	Q2
Load	H	X		X	PN	P0	P1	P2	P3
Mode Change	↑	L	L	X	X	No Change			
	↓	L	L	X	X	No Change			
	↑	H	L	X	X	No Change			
	↓	H	L	X	X	Undetermined			
	↑	L	H	X	X	Undetermined			
	↓	L	H	X	X	No Change			
	↑	H	H	X	X	Undetermined			

7483 4-Bit Full Adder with Fast Carry



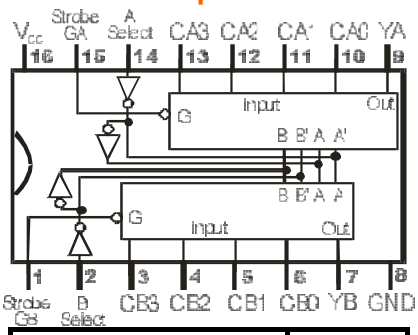
Inputs				Outputs					
				When C0 = L/C2 = L			When C0 = H/C2 = H		
A1/A3	B1/B3	A2/A4	B2/B4	1/ 3	2/ 4	2/ 4	1/ 3	2/ 4	2/ 4
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	H	L	H	H	L	H
L	L	H	H	H	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

74151 8-to-1 Selector/MUX



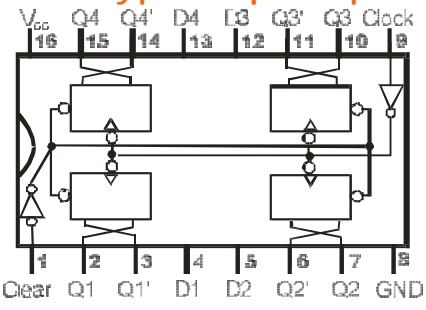
Inputs		Outputs		
Clear	Clock	D	Q	Q'
L	X	X	L	H
H		H	H	L
H		L	L	H
H	L	X	Q_0	Q_0'

74153 Dual 4-to-1 Multiplexer



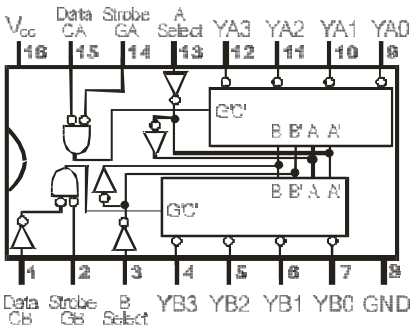
Inputs				Outputs	
Selects			Strobe		
C	B	A	S	Y	W
X	X	X	H	L	H
L	L	L	L	D0	D0
L	L	H	L	D1	D1
L	H	L	L	D2	D2
L	H	H	L	D3	D3
H	L	L	L	D4	D4
H	L	H	L	D5	D5
H	H	L	L	D6	D6
H	H	H	L	D7	D7

74175 Quad D-type Flip-Flop



Inputs							Output
Selects		Data				Strobe	
B	A	C0	C1	C2	C3	G	Y
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

74155 Decoders Demultiplexers



3-to-8 Decoder or 1-to-8 Demultiplexer

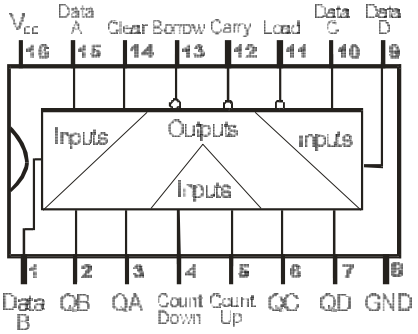
Inputs				Outputs							
Selects			Strobe	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
C	B	A	G	YB0	YB1	YB2	YB3	YA0	YA1	YA2	YA3
X	X	X	H	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	H
L	H	L	L	H	H	L	H	H	H	H	H
L	H	H	L	H	H	H	L	H	H	H	H
H	L	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H
H	H	L	L	H	H	H	H	H	H	L	H
H	H	H	L	H	H	H	H	H	H	H	L

3-to-8 Decoder or 1-to-8 Demultiplexer

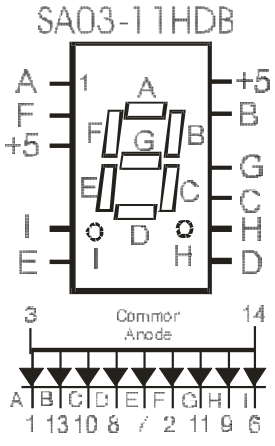
Inputs				Outputs			
B	A	GA	CA	YA0	YA1	YA2	YA3
X	X	H	X	H	H	H	H
L	L	L	H	L	H	H	H
L	H	L	H	H	L	H	H
H	L	L	H	H	H	L	H
H	H	L	H	H	H	H	L
X	X	X	L	H	H	H	H

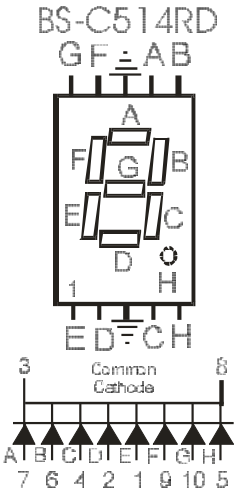
Inputs				Outputs			
B	A	GB	CB	YB0	YB1	YB2	YB3
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

74193 Dual Sync. Counter w/ separate UP/Down Clocks



Seven-Segment Displays





Lab Kit: Other resources

Fairchild Data Sheets on the Web

Texas Instruments Data Sheets on the Web

ECE 2090 - Lab 1

Logic Gates: A Smart Lighting System

PURPOSE

In this experiment, you will explore the notion of combinational circuits and basic combinational design.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS II

REQUIREMENTS

- Circuit diagrams for all three circuits with pin numbers clearly labeled.
- Verbal description of the function of the final circuit.
- Truth table for the first function (the light controller).

PROCEDURE

Section 1 – Designing and Building a Digital Light Control

Consider the problem of constructing a light controller for a certain room in a house. It is desirable for the light to be switched on if:

1) A Burglar Alarm detects an intruder
2) A Master Light Switch is on, **or**
3) An Auxiliary Switching system is active, **and** a person(s) is/are present in the room.

Item 3 requires further consideration. First of all, how is the system going to know if a person is in the room? A motion and/or sound detector could be used to produce a "logic 1" (`Boolean True`) if a person is detected.

The auxiliary switches mentioned above could be the wall switches already found in the room. Assuming that the room has two doors, then a (*three-way*) switch at each door would be convenient. In this configuration, the light is off if both switches are up or both are down, and it is on if one switch is up and the other is down. This allows for the light to be switched no matter what the state of the switches is.

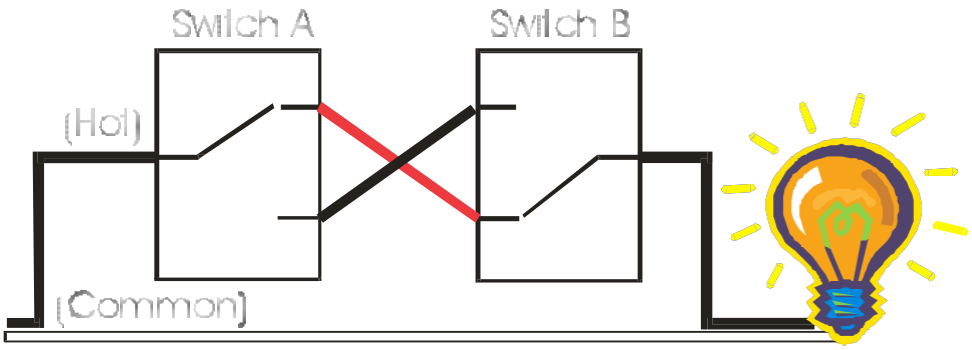


Figure 1. Wiring of a Three-Way Switch

For the lights to come on in our design, not only must one switch be up and one down, but a person must also be detected (unless one of the other conditions [Burglar Alarm or Master Switch] turns them on). Note that the "person detector" would probably have a timer which keeps the output high for a designated time after a person is detected.

Basically, what we need is a circuit which will switch the lights on if (and only if)

The Burglar Alarm is On

OR

The Master Switch is On

OR

A person is detected AND one but not both auxiliary switches is up.

Now let's assign some variable names to the various switches (inputs) so that we can write an equation to describe the desired binary function.

Let

B = Burglar Alarm
M = Master Switch
P = Person Detector
A_1 = Auxiliary Switch 1, and
A_2 = Auxiliary Switch 2

Note that the necessary condition of A_1 and A_2 to activate the lights is an *Exclusive OR* function (one, but not both). Using the XOR symbol $\oplus$, we can write the desired lighting function, F_L as

$$\mathbf{F_L = B + M + P \cdot (A_1 \oplus A_2)}$$

A circuit for this function can be drawn in the following manner:

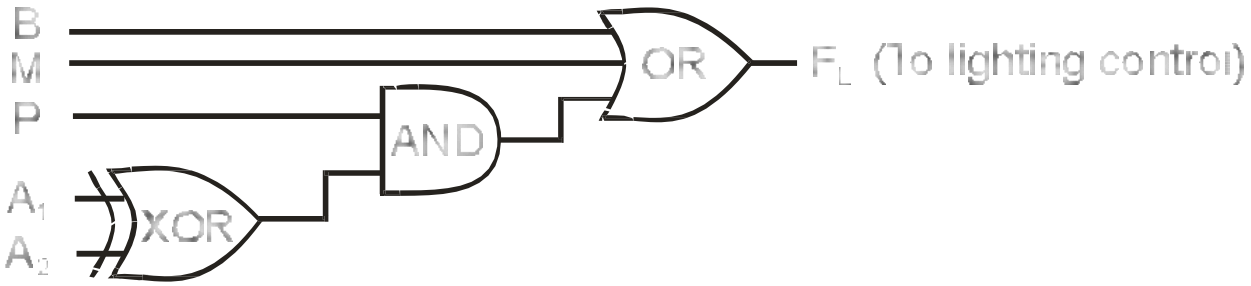


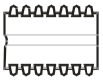
Figure 2. Circuit Diagram for F_L

Note that the circuit above would not switch the light on directly. The Binary signal that is 5 Volts DC from the output would need to go to a device (say a relay) which could switch on the 120-Volt AC power to the lights. In lab, however, we will wire the output directly to a light (LED) since these require only 5 Volts DC. If you use the red LED provided to you, use a 220 ohm resistor along with it. Connect the anode terminal (long terminal) of the red LED to the output of the circuit through the resistor, and connect the cathode terminal (short terminal) to the GND.

Note also that the wall switches in lab are only switching 5 Volts in this case, not the 120 Volts AC as would be found in the actual home.

One immediate problem of implementing this lab is there is no 3-input OR gate in your lab kit. Recall that the OR operation is associative allowing you to make a 3-input OR from two 2-input OR gates.

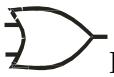
$$A \text{ OR } B \text{ OR } C = A + B + C = (A + B) + C = (A + B) + C$$

 Now connect this circuit and check to see that it works properly. Use the switch 0 and 1 for A_1 and A_2, respectively. Switch 2 for P, switch 3 for M, and switch 4 for B. Verify the functioning of the circuit. Now, instead of using a switch for B, use the function generator to generate pulse (or a square waveform) for input B. Adjust the frequency (≈ 1 Hz) for the pulse so that the LED (F_L) lights on and off when the pulse goes high and low, respectively. Then have your instructor verify the count.

Section 2 – Implementing a Function with Different Gates.

It's possible to implement this same function using only AND, OR, and NOT gates by using the definition of the XOR function. Using Boolean Algebra,

$$\begin{aligned} \mathbf{F_L} &= \mathbf{B + M + P(A_1 \oplus A_2)} \\ &= \mathbf{B + M + P(A_1'A_2 + A_1A_2')} \\ &= \mathbf{B + M + P(A_1'A_2 + A_1A_2')} \end{aligned}$$


Draw a diagram for this circuit, showing how to use 2-input AND gates to make the 3-input AND's. *Wiring and checking this circuit is optional.*

Section 3 – Realizing an Arbitrary Boolean Function

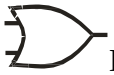
Now that we have looked at a real life example, let's look at an arbitrary function to see how we might realize it. Rather than specifying the logic with words as in the previous example, we will use a truth table. This will be a function of four variables (A, B, C, D), giving the truth table will have 16 entries.

A	B	C	D	F
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0

A	B	C	D	F
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

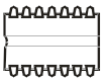
Using your Boolean algebra skills, you can write an equation for this expression and simplify it to *Minimal Sum Of Products* form (MSOP), which gives you:

F = B'D' + A'B'C + ABC + CD'


Draw a circuit for this function using only AND, OR, and NOT gates. The function calls for two 3-input AND gates and one 4-input OR gate. All you have available, however, are 2-input AND and OR gates, thus you must discover how to make

- A 3-input AND from two 2-input AND's, and
- A 4-input OR from three 2-input OR's.

Remember that both the AND and OR operations are associative.


Wire this circuit and have the lab instructor verify its proper function.

ECE 2090 - Lab 2

Encoding/Decoding: The Seven-Segment Display

PURPOSE

To familiarize the student with the seven-segment LED display, and the process of converting one type of binary information to another (encoding/decoding). A good understanding of BCD (Binary Coded Decimal) should also result.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS

II Simulation Software

REQUIREMENTS

- Circuit diagrams with pin numbers labeled.
- Verbal description of the function of the final circuits.
- Truth Table for all seven segments, and all seven functions in MSOP.
- Simulation of functional seven-segment display circuit.

PROCEDURE

Section 1 – About the Seven-Segment Display

! *Improper connections to the seven-segment display can destroy it. Double check your connections* **before** *applying power.*

The seven-segment LED (Light Emitting Diode) display is a common device in consumer electronics, from calculators to clocks to microwave ovens. In this lab, you will learn the basic principles of operation of the seven-segment display and the process of converting BCD values to the proper signals to drive this display. The display has seven separate bar-shaped LED's arranged as shown. In addition, many seven-segment displays have one (or two) circular LED used as a decimal point.

Figure 1. A Seven-Segment Display

Inside the seven-segment display, one end of each LED is connected to a common point. This common point is tied either to ground or to the positive supply, depending on the specific device. If your seven-segment display is designed to have the common connection tied to the positive supply, +5V, it is called a *common anode* configuration as shown below. To light these LED segments, the inputs must be a logic low.

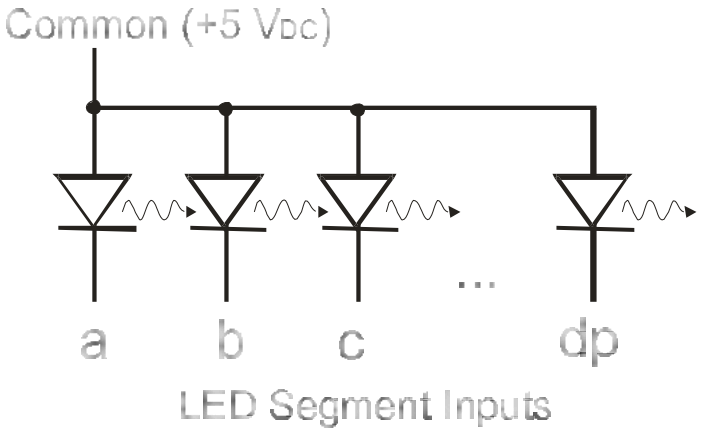


Figure 2. Common Anode Seven-Segment Display Circuitry

To actually light up a single LED segment, a resistor *must* be added to limit the current through the LED.

! *This resistor is critical! If you connect the LED between +5V and ground without the resistor, the LED will momentarily glow bright and then* **never** *glow again.* For this display use, a 220-Ω resistor as shown below.

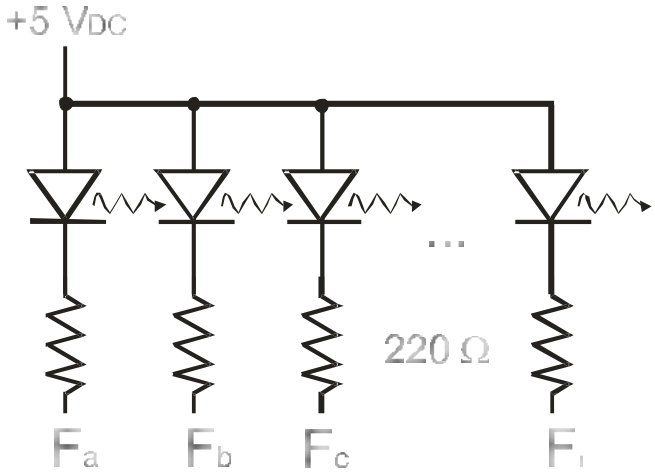


Figure 3. Resistors used with a Common Anode Seven-Segment Display

If F_a, F_b, etc... are +5 V, there is no voltage drop across the LED and resister resulting in no current flow through them, (and the LED remains dark.) If the inputs are 0 Volts, a current is produced and the diode glows.

If your lab kits contains a *common cathode* display, the common point is *ground* instead of +5 V as shown below. To light these segments, a logic high must be supplied.

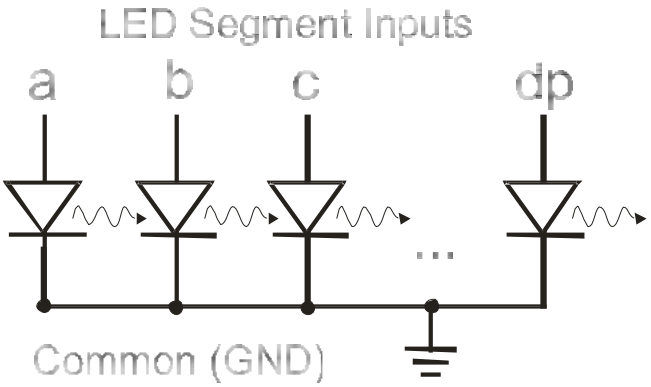


Figure 4. Common Cathode Seven-Segment Display Circuitry

These LED circuits can be used in the last lab as a *logic indicator* to troubleshoot circuits. Consider verifying the operation of the XOR gate of Lab 1 using a logic indicator shown below. The LED functions as a logic test probe which lights up when the test point is a logic zero and does not light up when the test point is at a logic one.

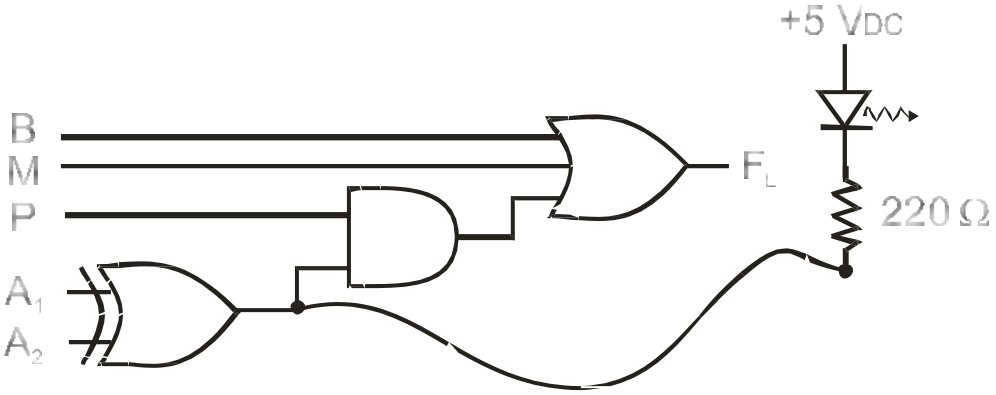


Figure 5. LED used as Logic Test Probe

NOTE: You can use the LED's built into the NI ELVIS for trouble-shooting as discussed above. Consider that they (more appropriately) turn on with a logic one instead of a logic zero.

Now consider how the ten decimal numerals can be formed using the seven-segment display. The figure below shows these digits 0 through 9.

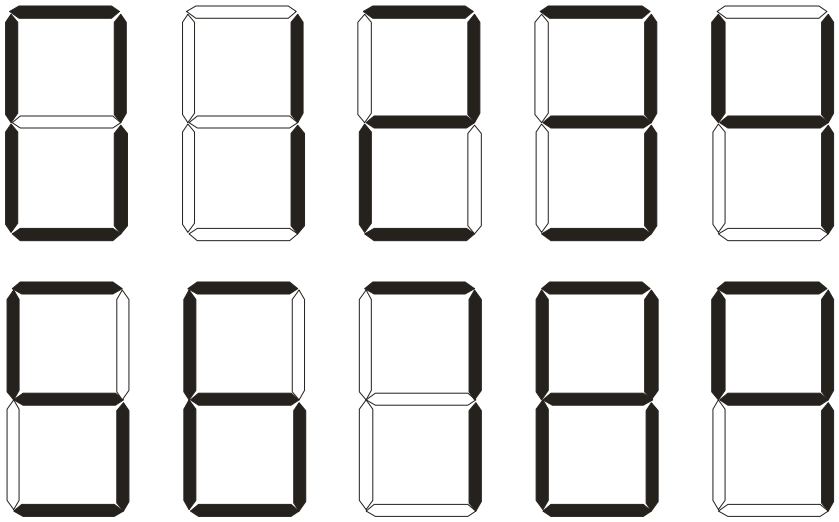


Figure 7. Seven-Segment Display Decimal Representations

Section 2 – The BCD to Seven-Segment Converter

What we wish to do is input a BCD (4-bit) binary number to some combinational circuit which causes the appropriate segments of the display to light up. For example, if a `0000` is input to the circuit, all of the LED pins on the seven-segment display should go low (to light the segments) except the pin connected to the center (horizontal) LED. That is, segments **a** through **f** (See Figure 1.)

What we need to do now is determine the appropriate combinational circuit to light each segment. That is, the each segment is turned on by certain Boolean function. For example, segment **a** is lit for 0, 2, 3, 5, 6, 7, 8 and 9. Therefore the circuit must produce a *logic zero* for these numbers to light segment **a**. That is, it must produce a *logic one* for the numbers 1 and 4.

Remember that BCD numbers use only ten of the sixteen possible combinations of four bits (0-9). Therefore, we do not care what comes out of the circuit for the last six inputs (`1010` through `1111`.) since these inputs should never occur. The resulting symbol for these inputs should be whatever it takes to produce the least complicated circuit.

For the function of segment **a**, we have the following truth table.

D	C	B	A	F_a	D	C	B	A	F_a
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	X
0	0	1	1	0	1	0	1	1	X
0	1	0	0	1	1	1	0	0	X
0	1	0	1	0	1	1	0	1	X
0	1	1	0	0	1	1	1	0	X
0	1	1	1	0	1	1	1	1	X

The "X's" in the truth table above indicate the "don't care" conditions: that is, rows in the table where we do not care what the output of the function is. Using Boolean algebra, we can write the minimum sum-of-products (MSOP) expression for F_A as:

$$F_a = D'C'B'A + CB'A'$$

? *What comes out of this circuit for each of the six invalid inputs? For example, what would F_A be if the input were 1010?*

Now, make up a truth table for all seven segments and find a function (MSOP) for each.

! *You need not make a separate truth table for each segment - just list the inputs once, and have seven output columns as shown below.*

D	C	B	A	F_A	F_B	F_C	F_D	F_E	F_F	F_G
0	0	0	0	0	0	0	0	0	0	1
	...						...			
1	1	1	1	X	X	X	X	X	X	X

We need not build all of these individual circuits with separate IC's, however, in order to use the seven-segment display. Because this function is so common to electronics, a single chip has been standardized to perform this conversion. This chip (which you have in your lab kit) is the 7447 and is called a BCD-to-seven-segment display. A block diagram for this chip is show below.

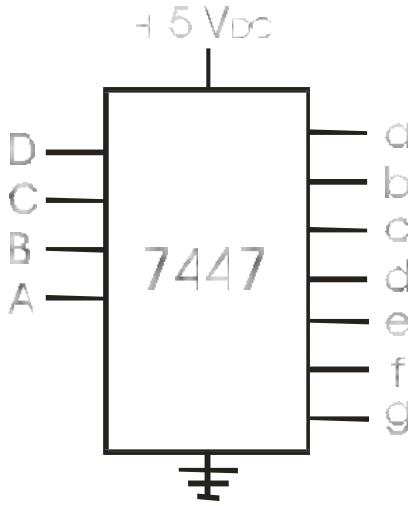


Figure 8. 7447 BCD to Seven-Segment Display

The 7447 has common collector outputs which can sink much more current than they can source (supply). Therefore, the 7447 is designed for a common anode type display which needs a logic low to turn on the segments. If you have a common cathode display, then you must use an inverter on each input. The inverter should be able to supply enough current to light a segment.

Now, using the 4 switches of the *NI ELVIS* as the BCD input, connect the following circuit and verify that it functions properly.

Be sure to include a description of the circuit operation in your report.

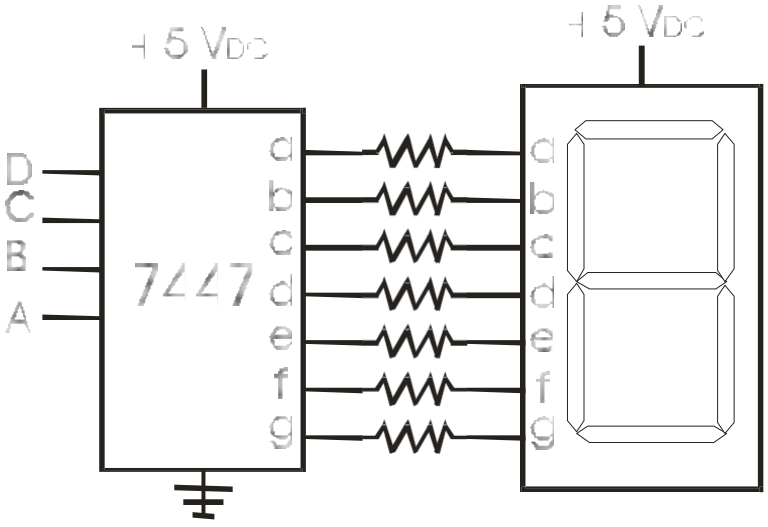


Figure 8. Using the 7447 BCD to Seven-Segment Display

! Remember to be very careful wiring this circuit, making sure that no resistor leads are shorted anywhere and that the power to the seven-segment display is connected to the correct pin. **Do Not** connect any pin of the common anode seven-segment display directly to ground! This will short out the segment forever. Likewise, do not connect any pin of the common cathode display to + 5 V or it to will be shorted out forever.

Check the six unused input combinations (1010 through 1111) and report which segments light up.

? Does this match what you would expect from the seven equations you got for the decoder? If not, can you think of one reason why the output might not match your equations?

ECE 2090 - Lab 3

Combinational Circuits: Parity Generation and Detection

PURPOSE

To familiarize the student with combination circuits by studying methods of parity generation and detection.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS

II Simulation Software

REQUIREMENTS

- Karnaugh Map for Parity Generator and Detector.
- Truth Table for Parity Detector.
- Verbal description of the function of the Parity Generator/Detector.
- Simulation of functional Parity Generator/Detector.

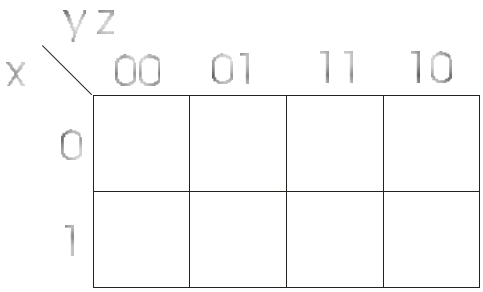
PROCEDURE

Section 1 – Parity Generator

In this part of the lab, we will design and build circuits to generate and detect odd parity for three-bit words. Our parity generator circuit will take three input bits (x, y, and z) and produce one output bit (P). The truth table for this parity generator is shown below:

x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Use the Karnaugh Map to produce an MSOP representation of this function by grouping the 1's of the function.



P = _____

? How many 2 input AND and OR gates and how many inverters would be required to implement the equation for P above? (Remember, it takes two 2-input gates to make one 3-input gate)

ANDs - ORs - NOTs -

This circuit can be implemented with the chips in your lab kit, but it would leave only one OR gate and no AND gates to implement our entire parity detection circuit. (Not to mention that it would be a pain to wire up!). Can we simplify this function in order to simplify the hardware? This is one of those examples where you see why engineers can't be replaced by computers (yet).

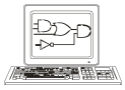
The K-map guarantees us MSOP form, but that's not the simplest form for this problem. Neither is the MPOS form we'd get from grouping the zeroes. (Try it and see.) Notice that the Karnaugh map shows a checkerboard pattern (every other square). When this pattern exists, the function can be implemented in either an XOR or XNOR operation. The equation obtained for P can be simplified using the properties of Boolean algebra as follows:

$$P = (x'y' + xy)\,z' + (x'y + xy')\,z$$
$$= (x \oplus y)'\,z' + (x \oplus y)\,z$$

If we let $A = x \oplus y$ then, we have

$$P = A'z' + Az = (A \oplus z)' = (x \oplus y \oplus z)'$$

Therefore, we can implement P with a three-variable XNOR gate. Of course, we don't have a three-input XNOR gate in our lab kits, but we can easily build one from two XOR gates (7486) and single inverter (Recall that XOR is associative, like AND or OR, so that $(x \oplus y) \oplus z = x \oplus y \oplus z$).



Use Logisim to create a macro of your parity generator circuit.

Wire up the circuit on your breadboard and test its function.

Section 2– Parity Detector

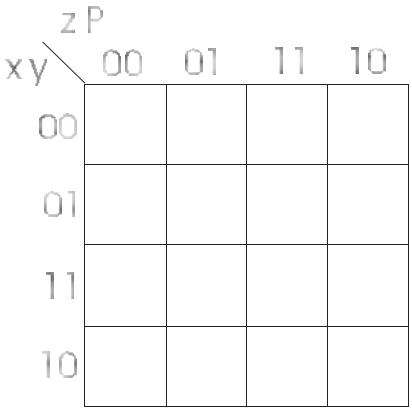
Next, we need to implement a Parity Detector circuit. It will have four inputs: the three information bits, x, y, and z and the newly created parity bit, P. It will have one output bit, the error, E, which will be high whenever there is a parity error.

Fill in the value for E in the truth table for this circuit below. Remember, E will be 1 whenever P is not the correct odd parity bit for the values of x, y, and z:

x	y	z	P	E
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	

x	y	z	P	E
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Now, find an MSOP equation for E using the Karnaugh map below:



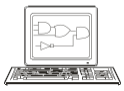
E = _____

Once again, you'll notice that this equation would be a nuisance to wire up since we cannot form any groups on the map. And, once again, we notice the familiar checkerboard pattern.

If we look back at our truth table, we'll notice that E is true whenever there is an even number of 1's in the four input variables (just as P was 1 whenever there was an even number of 1 's in the input in the table for our generator circuit). Once again, we can implement this function as an XNOR of the four input variables,

$$E = (x \oplus y \oplus z \oplus P)'$$

Because of this property, XNOR is also known as the *even function*, and XOR is also known as the *odd function*. If we changed our truth table so that our output was true whenever there was an odd number of 1s, the resulting function would be an XOR, and the Karnaugh map would still look like a checkerboard, but the first one would be in square 0001 instead of 0000.



Create your parity detection circuit using the equation for E above as a macro in Digital Works with four inputs and 1 output and verify its function. To test your simulation, embed your generator and detector macros in the same circuit as shown in the figure below:

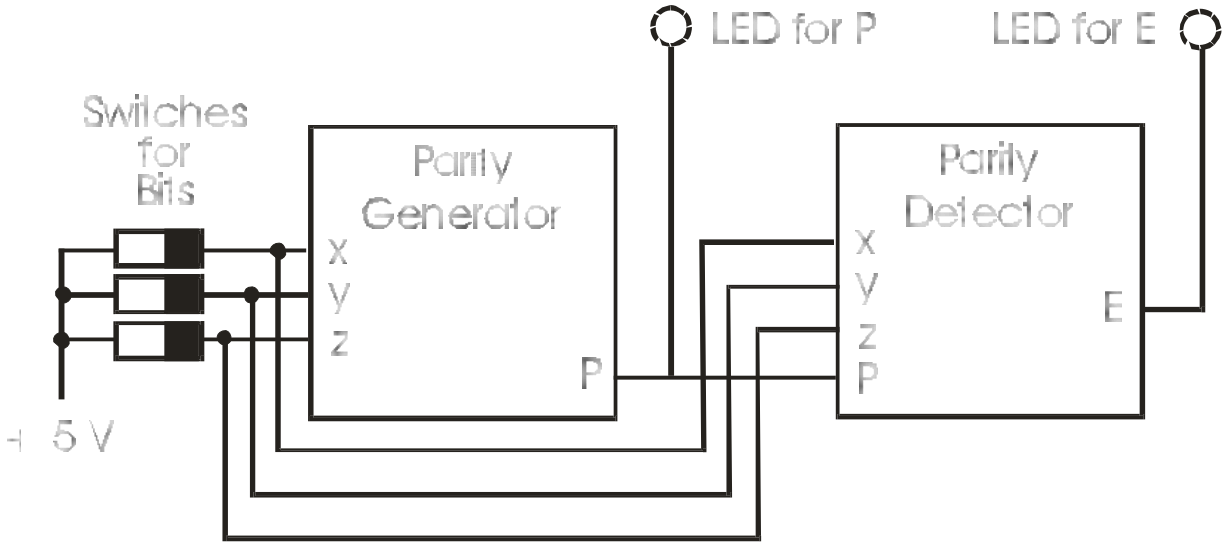
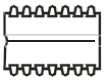


Figure 1. Parity Generator and Detector



Wire up your circuit on your breadboard. This will require three more XOR gates and one more inverter. You should have two 7486 chips in your kit. Connect your parity generator to your parity detector and verify that it works correctly.

? How might you change the circuit above to simulate a communication where a single bit error may be introduced to one of the four inputs to the parity detector?

ECE 2090 - Lab 4

Binary Arithmetic - Adders

PURPOSE

The student should demonstrate knowledge of simple binary arithmetic and the mechanics of its use. Each student is required to design, simulate, build, and test a two-bit full adder.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS

II Simulation Software

REQUIREMENTS

- Simulation of functional Full Adder.
- Functional Full Adder Circuit.

PROCEDURE

Section 1 – Adders

Consider the problem of adding two single-bit numbers, A and B, resulting in a single two-bit answer. The truth table for this operation is shown below:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The two output functions are labeled ‘C’ and ‘S’, where ‘S’ stands for ‘sum’ is the low order bit of the output. The ‘C’ stands for ‘carry’, and is the high order bit of the output. The functions for S and C can be written as the two MSOP equations below:

C = AB **S = A′B + AB′**

A circuit that implements these two functions is known as a *half adder*. This adder is referred to as a half adder because it only solves half the general problem of adding numbers with more than one bit. Let's take a look at an example of what happens when we add two 8-bit numbers:

```
Carry     1 0 1 1 1 0 0 0
A           1 0 1 1 1 0 0 1
B           1 0 1 0 1 1 0 0
```

Sum **0 1 1 0 0 1 0 1**

Note that, except for the right most column, we are actually adding *three* bits: a bit from each of the 2 numbers and a carry bit from the bits immediately to the right. Note also that each addition produces 2 bits - the result bit (S), and the carry bit (C). Now, let's make a truth table for this addition process. The truth table will have three variables, one bit from each of the numbers A and B, and a carry in bit, C_{in}, which represents the carry from the previous position. The two outputs are the sum bit and the carry out bit, C_{out}, which will be used in the next position.

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

We'll use Karnaugh maps to simplify the two functions in the table above into MSOP form:

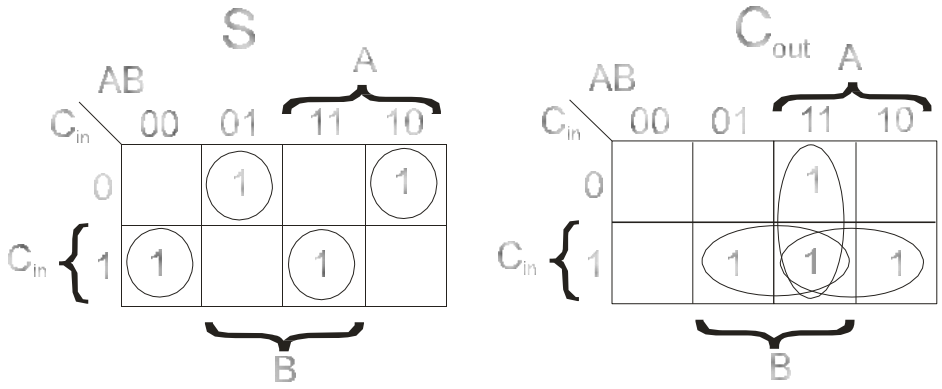


Figure 1. Full Adder Karnaugh Maps.

As shown above, the MSOP functions for S and C_{out} are:

Sum = A′B′C + AB′C′+ A′BC′ + ABC **C_{out} = AB + AC +BC**

We can implement the function for C_{out} in a straightforward manner, as shown below:

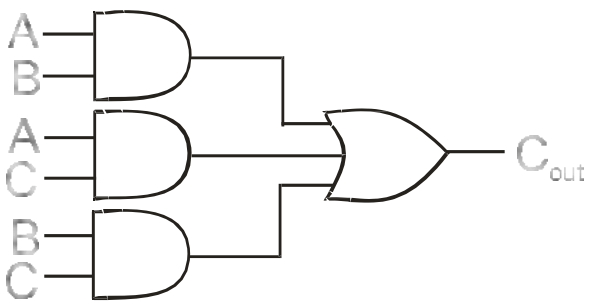


Figure 2. Full Adder Carry Circuit.

The MSOP form S is a bit more complex, however. If we examine this function a bit more closely, though, we will see the now familiar checkerboard pattern in the K-map, and notice that S is only equal to 1 when an *odd* number of the input variables are 1 in the truth table. The function for S can therefore be easily implemented with an XOR function, as shown below:

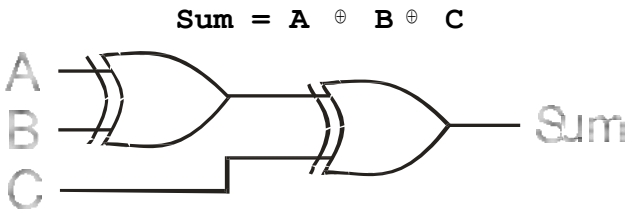
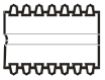


Figure 3. Full Adder Sum Circuit.

These two circuits together are called a *full adder*.

Section 2 – Building a 2-bit Full Adder



You are going to build a device which will add two unsigned 2-bit numbers. Use a pair of switches for each input value. The results will be displayed on LEDs. You will need to build two copies of the full adder. The carry input to the right most adder will be tied to GND. The carry in of the left adder will be tied to the carry out of the right adder.

? What happens to the carry out of the left adder?

In block diagram form, the 2-bit Full Adder looks like:

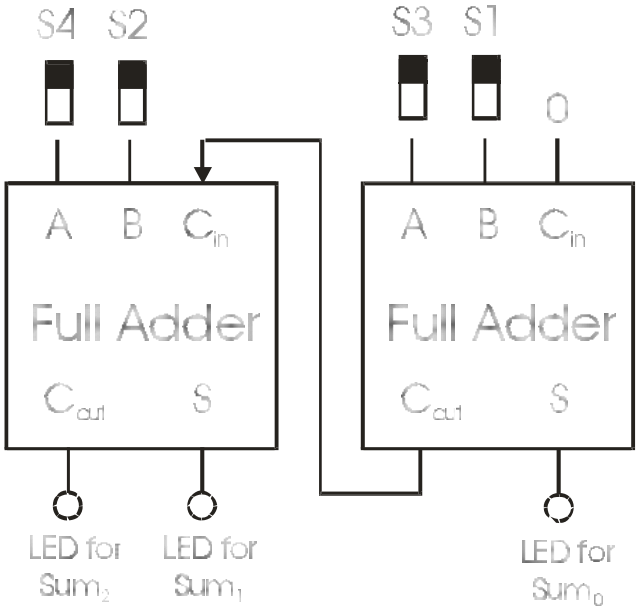


Figure 4. Full Adder Circuit.

You will need one 7486, two 7408's and two 7432's to build the above circuit.

In this simulation, you will build some simple circuits with gates and then use multiple copies of those circuits to build a larger, more complicated device. This is typical of how system design is done. We use this same kind of procedure with Computer Aided Design (CAD) tools, like your digital simulator.

Ideally, you would like to enter the circuit for a full adder once, test and debug it, then turn it into a part we can use over and over again. Then you could simply use two of those parts and draw the connections between them to build your two bit adder. This is called a *hierarchichal* design.

In Logisim, hierarchy is achieved by using *macros*. You used a macro in the previous lab for the 7447 driver chip. This week, you will want to create your own macro for the full adder circuit. See the online Logisim *Getting Started* guide for details on how.

Once your macro is created, you will create *another* schematic that uses two copies of the macro, and adding the switches and lights. Your resulting schematic should look like the block diagram of the circuit shown above.

HINT: It will probably be easier if you use one 7408 and one 7432 for each carry generator rather than using one chip for parts of both full adders. This way the two carry circuits can have identical pin assignments and also be physically separate to help avoid confusion. This also makes it possible for you to label the pins correctly inside your full adder macro.

You are to turn in copies of the schematic for you full-adder macro as well as your final circuit in lab.

ECE 2090 - Lab 5

MSI Circuits - Four-Bit Adder/Subtractor with Decimal Output

PURPOSE

To familiarize students with Medium Scale Integration (MSI) technology, specifically adders. The student should also become familiar with 1's complement arithmetic.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS
II Simulation Software

REQUIREMENTS

- Simulation of functional Full Adder.
- Functional Full Adder Circuit.

PROCEDURE

Section 1 – Adders

In the last experiment we built a pair of adders and used them to add two 2-bit numbers. In this lab le will use an MSI chip containing four full adders to add and subtract two 4-bit signed numbers using one's complement arithmetic.

All of the chips used thus far have been SSI (Small Scale Integration) chips which consist of single gates. MSI chips combine dozens of gates into a single function on a chip--in this case, a 4-bit full adder, the 7483. LSI (Large Scale Integration) and VLSI (Very Large Scale Integration) combine hundreds or thousands of gates into very complex devices on a single chip. Microprocessors and related components fit into these categories.

Remember that by using one's complement arithmetic we can both add and subtract with the same circuitry. The problem remains of how to complement a number so that subtraction can be performed.

Section 2 – Adder/Subtractor

Part I

Let us recall the operation of an XOR gate. Note that if one input is 0, the output equals the other input. On the other hand, if one input is 1, then the output equals the complement of the other input.

Figure 1. Using an XOR Gate as an Inverter

We can thus use XOR gates to perform a one's complement on command.

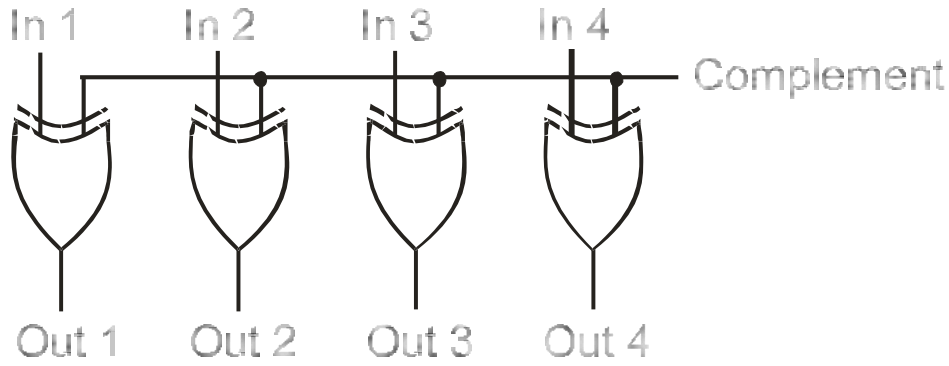


Figure 2. Using XOR Gates as a One's Complementor

In block diagram form, the 7483 would appear as shown:

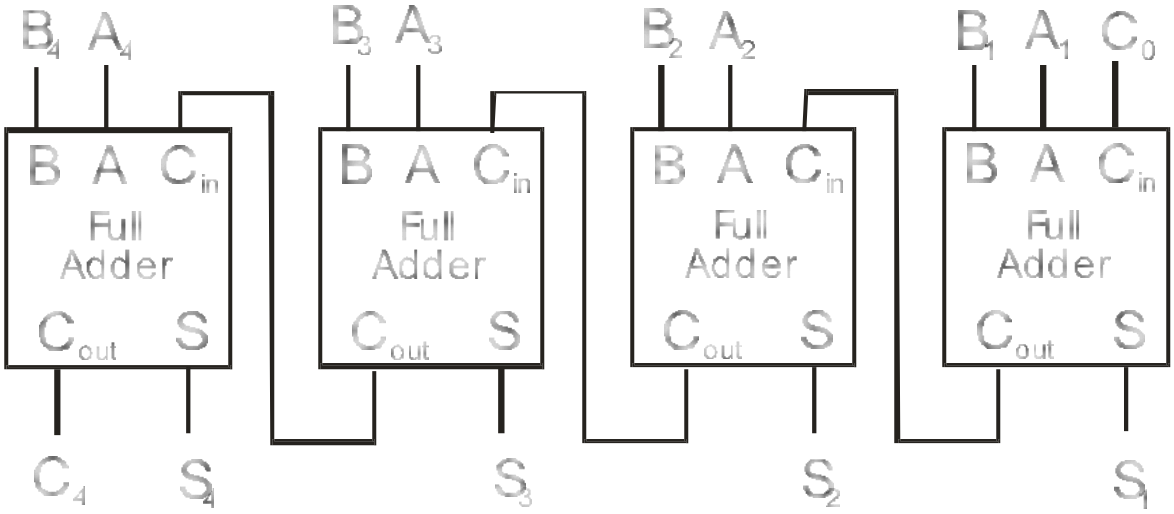


Figure 3. Block Diagram of 7483

Note that the carries are already interconnected within the chip. The right-most carry in, C_0, and left-most carry out, C_4, are available for cascading to other 7483's or other uses. The A's and B's are the inputs (addends) and the S's are the outputs (sum).

Label the pin numbers on the following circuit, construct it, and verify that it both adds and subtracts A and B (correctly).

! ***Check pin numbers for power and ground.***

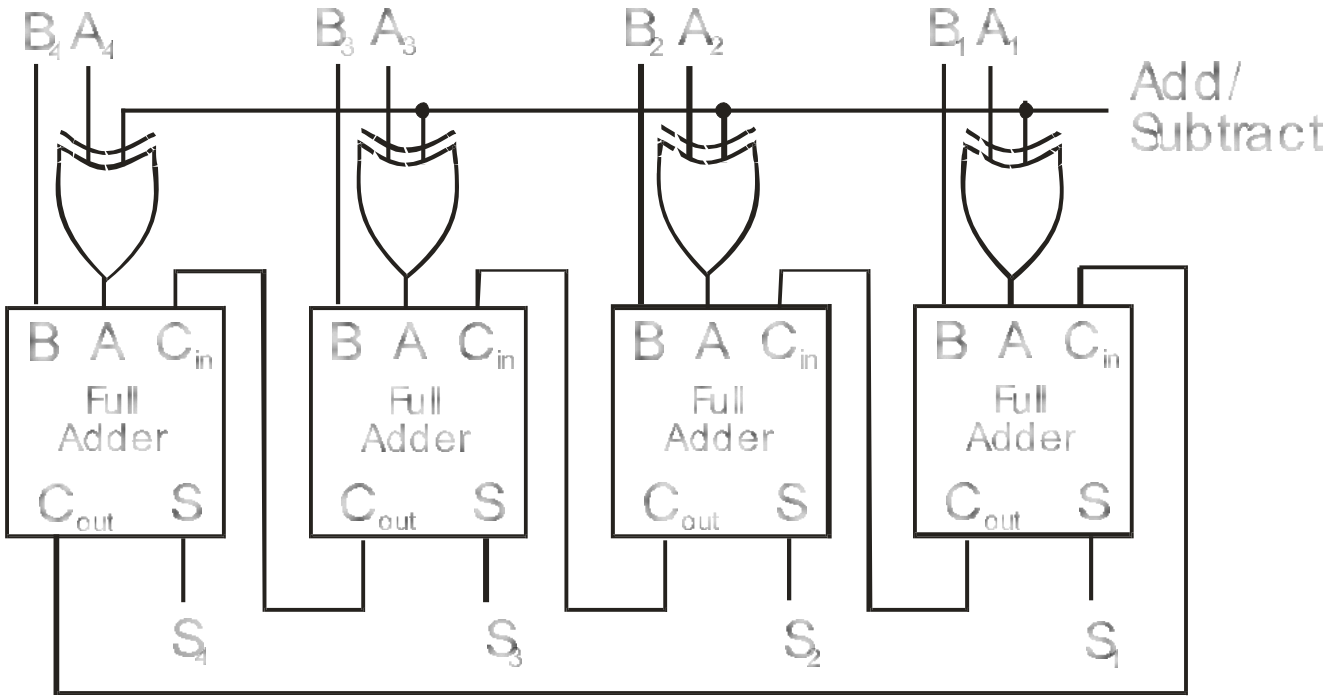


Figure 4. Using the 7483 for Addition and Subtraction

If only four switches are present, you can use these for the bits of `A` and simply plug the `B`'s into either +5 V or ground to produce `B`. (e.g. For B = 3, let B_4 = GND, B_3 = GND, B_2 = +5 V, B_1 = +5 V.

? What happens if we add 0011 and 0111? Is the result correct? Why or why not?

Why is C_4 connected to C_0?

! ***Be sure both chips are at one end of the breadboard to facilitate the further expansion of the circuit.***

Part II

It would be handy if we could display our results in a more easily readable form. Using another XOR package, a seven-segment display, a few resistors (to limit current so the seven-segment display won't smoke), and a BCD to seven-segment decoder (an extension of your second Lab), this goal may be achieved by means of the following circuit. How are the XOR's used to switch between Addition and Subtraction.

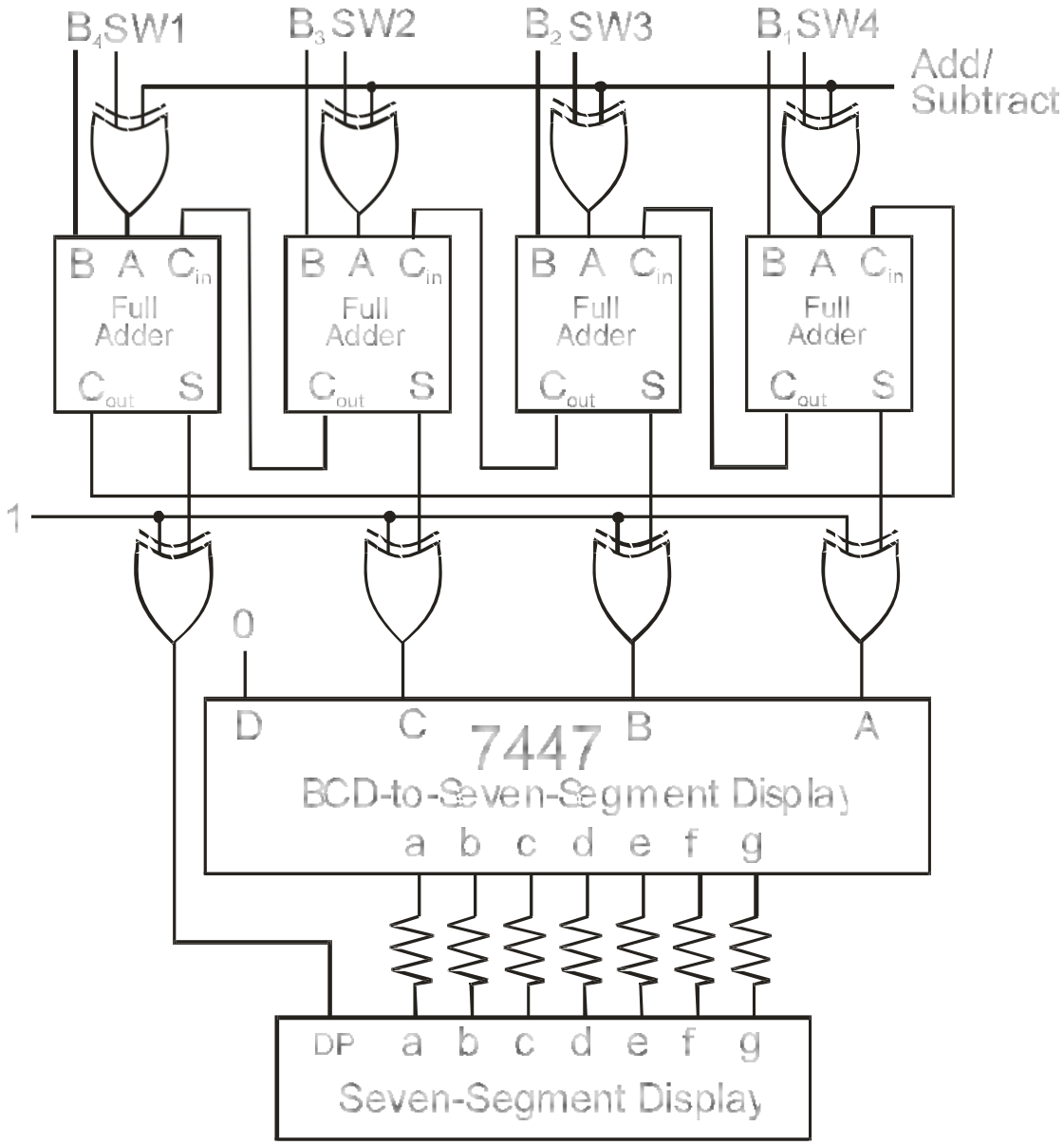


Figure 5. Adder/Subtractor with Display

Construct and test the operation of the circuit above.

The XOR's above may not be necessary, depending on the type of seven-segment display you are using. If the seven-segment display is a common anode design, then the sum, S_4 - S_0, must be inverted by `XOR`'s because the segments are lit by sending segment inputs **a** through **g** low, 0, instead of high. This might seem strange, but it will make perfect sense once you have studied the internal structure of TTL gates. If you are using a common cathode type device, then these gates are not necessary.

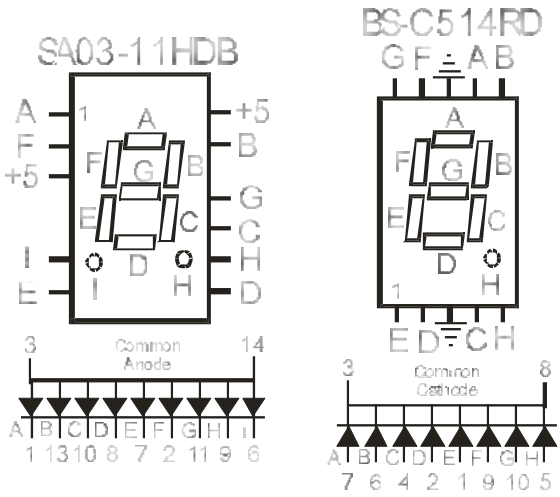


Figure 6. Common Anode and Common Cathode Displays

! ***Remember that the seven-segment display must use resistors to limit the current through the LED's.***

Be careful not to short the resistor leads together. This may result in a decrease in resistance and an thus an increase in current through the led segment causing it to quickly burn out and* never *shine again.

You need only to disconnect the wires to the lights in the first circuit. The rest can be left intact.

? Questions to turn in with the lab report:

- Do you think the 7447 is classed as SSI, MSI, or LSI?
- In both circuits, why are C_0 and C_4 connected together?
- Why is the D input of the 7447 always 0?
- What is the purpose of the 3 `XOR`'s connected to the A, B and C inputs of the 7447? Explain fully in your own words.
- Explain how to convert the above circuit to perform *two's*-complement arithmetic. Draw the circuit and explain its use. (Hint - C_0 can be used to form two's-complement.)

ECE 2090 - Lab 6

Multiplexers and Serial Communication

PURPOSE

To familiarize students with the internal realization of multiplexers, and to show an application of multiplexers and demultiplexers for use in serial communications.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS

II Simulation Software

REQUIREMENTS

- Circuit diagram with pin numbers labeled.
- Verbal description of the function of the final circuit.
- Simulation of functional seven-segment display circuit.

PROCEDURE

Section 1 – The Realization of a 4-bit Multiplexer

A 4-to-1 multiplexer functions like a four-position switch such as the one shown below. The switch contact can be moved to any one of the four positions. The switch can not be moved anywhere else, i.e. it must be in contact with one of the four inputs at any time. The output signal of the rotary switch equals the signal on the input to which the switch rotor has been positioned.

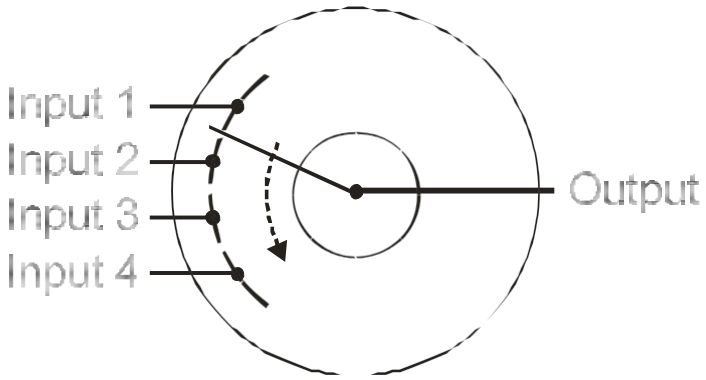


Figure 1. Four-Position Rotary Switch.

By now, you may be wondering what this has to do with multiplexers, which are little electronic

gizmos having no actual switch contacts, no knob to turn or lever to position, or anything of the sort. It

is helpful to think of a multiplexer as a rotary switch whose position is controlled by a binary number input to the device.

How many control bits would we need to select one of sixteen positions?

The control inputs will henceforth be called *select* lines since they are used to select one (and only one) input to be routed to the output. We can redraw our rotary switch as follows:

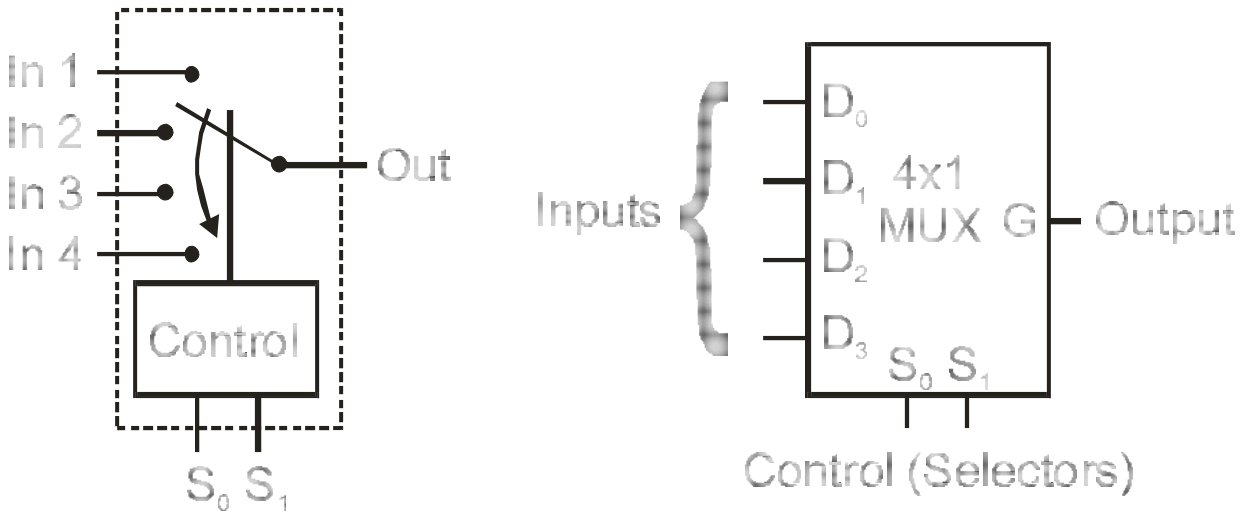


Figure 2. 4-to-1 Multiplexer.

The selectors S_1 and S_0 determine which "position" the "switch" is in. Note that the *order* of S_1 and S_0 is important; S_1 is the most significant bit whereas S_0 is the least significant bit. If we want to be slightly more technical, we can formulate the Boolean equation for the output function of a four input multiplexer as:

$$\mathbf{f} = \mathbf{S_1' S_0' I_0} + \mathbf{S_1' S_0 I_1} + \mathbf{S_1 S_0' I_2} + \mathbf{S_1 S_0 I_3}$$

Note that when a binary zero (00) is on the select lines, the output `f` is equal to I_0, when a binary three (11) is on the select lines, **f** equals $\mathbf{I_3}$, etc.

Section 2 – Application - Serial Communication

Some notes before we get started: in this section, in addition to using multiplexers and demultiplexers, we will use a 74193 counter chip. You probably have not seen counters yet in your 201 lecture. Don't worry, you don't have to know how counters work in order to complete this lab. You just have to know what they do: they count (in binary). Connect the counter as shown at the end of this lab procedure,

and it will count repeatedly through the numbers 000 to 111 on the "Q" outputs, incrementing once on each clock cycle (toggle). (We'll discuss counter design after introducing sequential logic circuits.)

You'll also need to make use of your 74151 8-to-1 multiplexer chip, and your 74155 chip.

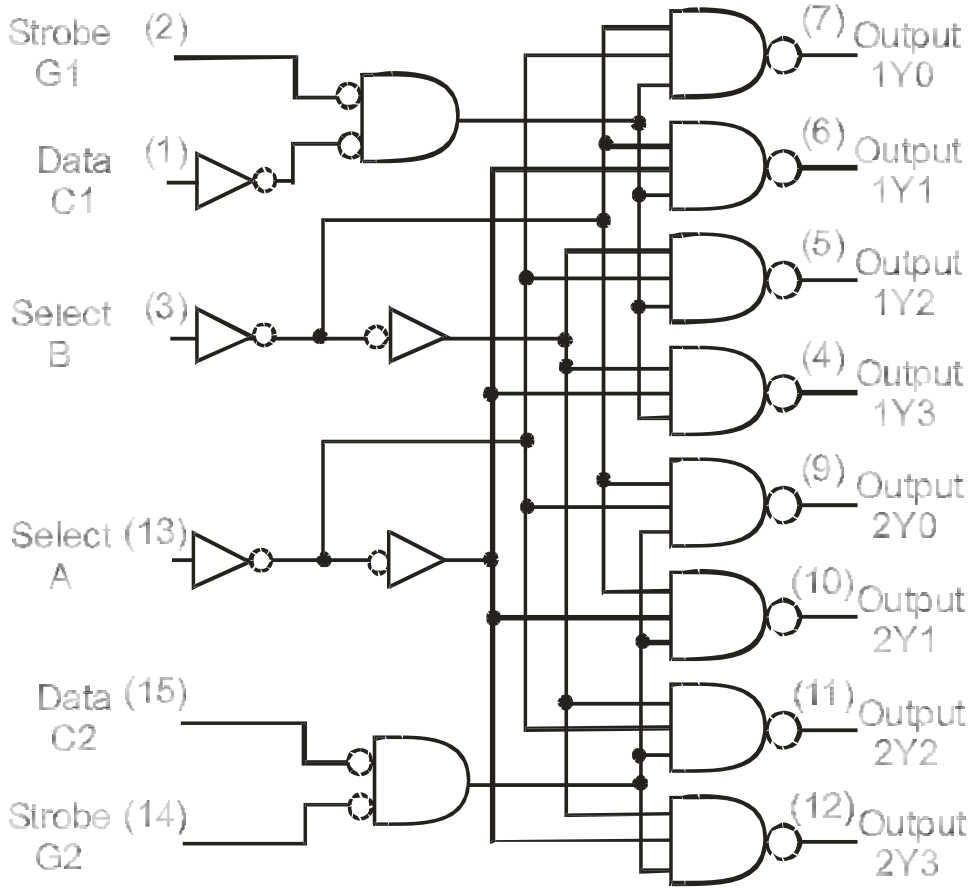


Figure 3. 74155 Logic Diagram.

Time multiplexing is often used with LED displays on calculators to reduce the amount of current the battery must supply to light the LED's. Rather than lighting all of the segments at once, one segment (or groups of segments for multiple digits) will be lit for a short time, perhaps 1 ms. Then the next segment (or group) will be lit for an equal time, and so forth until all the segments have been lit. Then the cycle repeats, making each segment of display flash on and off so quickly that the display appears continuous to the human eye, only slightly dimmer. The battery is then required to supply a much smaller average current than when all the segments are displayed continuously. Since LED's use a lot of current, around 10mA (all digits = 8), this can greatly prolong battery life.

Time Multiplexing is also often used in communication systems where independent data streams must be sent over a single line or channel. The phone company does this on its lines.

We are going to time-multiplex the seven segments of a 7-segment display. The 74193 counter will be used again, where the three least significant bits driving both the Multiplexer and the Demultiplexer. The Demultiplexer is essentially a backwards multiplexer - one input and 2^n outputs which are selected by n select lines. We will construct a 1-to-8 demultiplexer from the 74155 Dual 1-to-4 demultiplexer. The 74155 has two 1-to-4 demultiplexers, one of which has an inverting input (just to make life more difficult). A functional diagram of the 74155 is shown below.

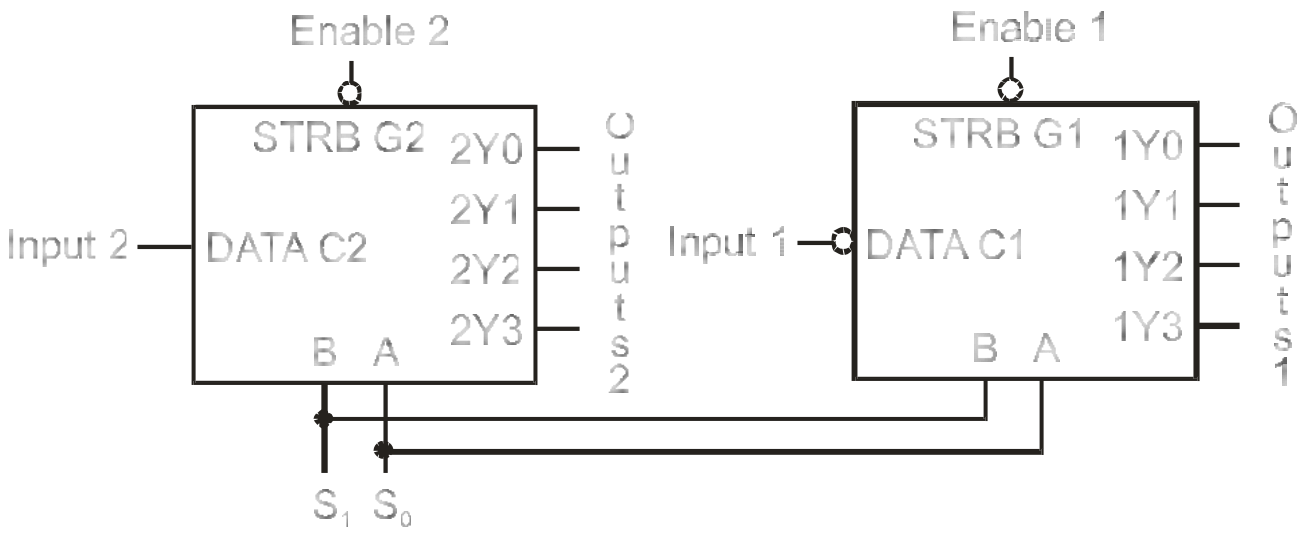
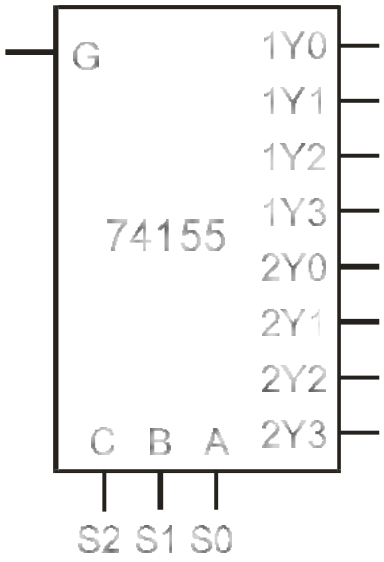


Figure 4. 74155 Functional Diagram.

To use as a 1-to-8 demultiplexer, connect as shown below:



G = Inputs 1G and 2G connected together.

C = Inputs 1C and 2C connected together.

Figure 5. 74155 Used as 1-to-8 Demultiplexer.

Now connect the following circuit:

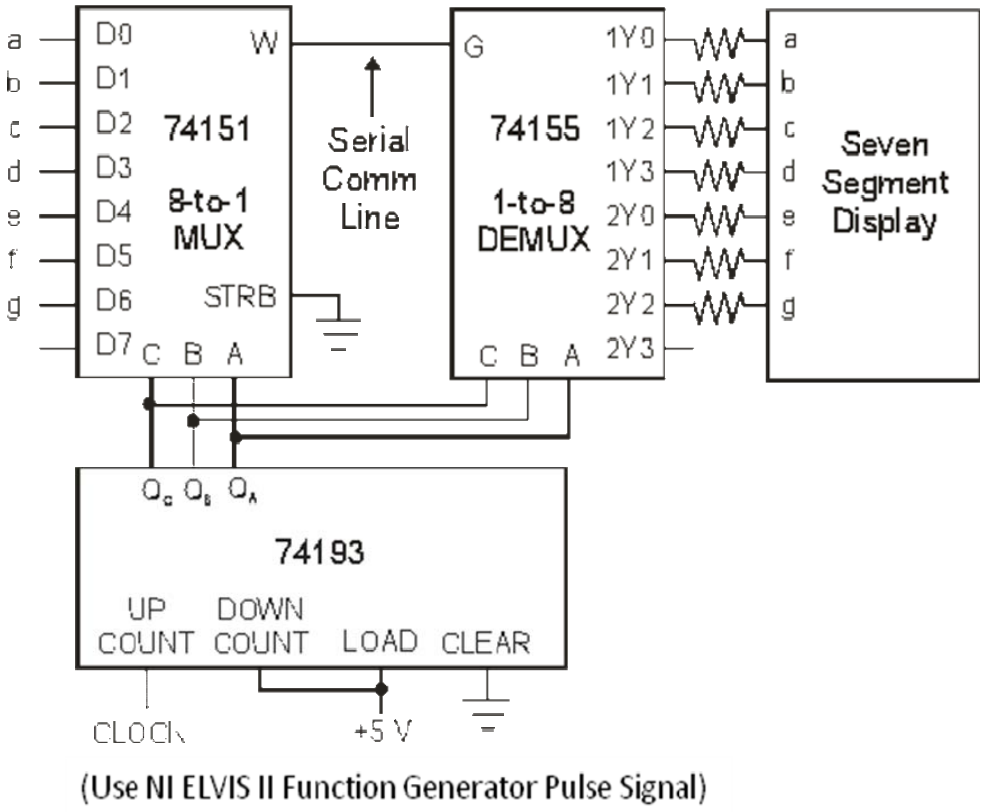


Figure 6. Parallel to Serial Communication.

! **WARNING!** Do not wire any pin of the seven-segment display to ground. You can destroy it. Also be sure that the bare leads of the resistors do not touch before turning on power.

To operate: wire the inputs to the 74151 as appropriate to light the proper segments for the number "5". Set the pulse/square waveform (clock) on 1 Hz and check that the proper segments light. Once all segments light (or remain dark) as they should—it will take 8 seconds for the number to be displayed—speed the clock up (by increasing the frequency of the pulse) and observe the effect at each speed. At 1 kHz, the display will appear constant though a bit dimmer. You can also use sweep settings of the function generator to increase the frequency automatically.

? Questions to turn in with the lab report.

1. Explain why the circuit connection on page 2 acts like a 1:8 demultiplexer
2. If this circuit were being used to transmit data over a single wire, which connection on the final circuit (page 3) corresponds to the data wire? (Ignore timing problems with the counter.)
3. How do you think the phone company uses multiplexing to put many conversations over a single line?

4. What other types of multiplexing can you think of?

Some notes on the simulation of this lab

Your simulation package does not have 74151 and 74193 chip models. Therefore, you will have to make them from smaller parts that do exist.

The 74151 MUX can be made in one of two ways. One approach would be to simply draw the AND-OR diagram of an 8-to-1 MUX. It's pretty simple, you just need eight 4-input AND's (each of which AND's the appropriate input line with the correct Minterm of the three select lines) and an 8-input OR function. Hint: Once you place a gate in Logisim, you can right click on it to increase the number of inputs to three or four.

Another approach would be to take advantage of the 4-to-1 MUX you made for Section 1 of this lab. Two 4-to-1 MUXes can easily be connected to form an 8-to-1 MUX as shown below:

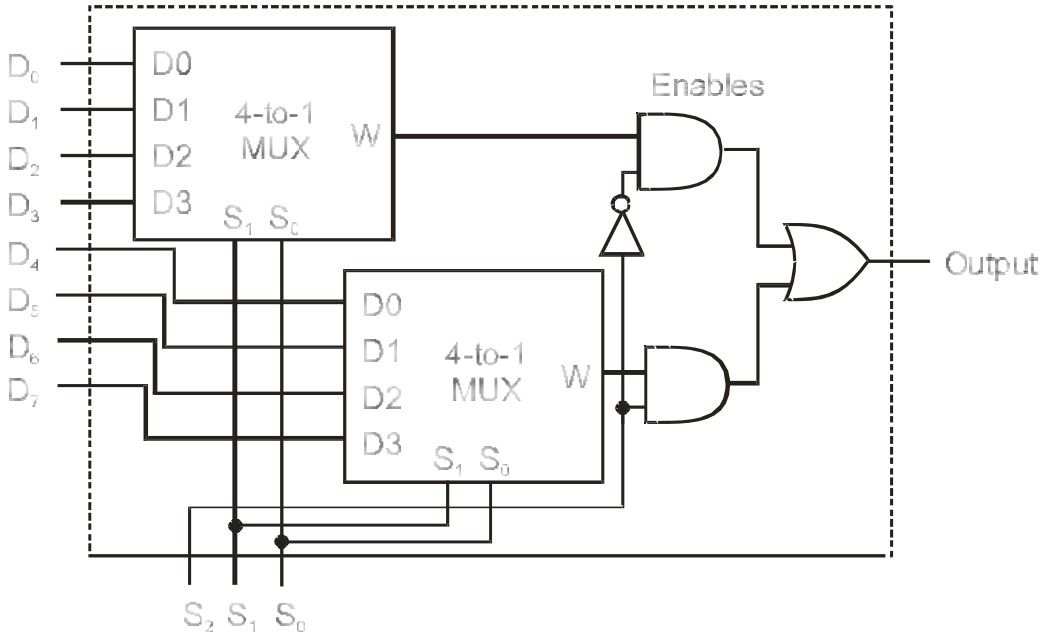


Figure 7. 8-to-1MUX from two 4-to-1 MUXes.

Logisim does supply a 4-bit counter macro which could be used for the 74193, but none of the pins are labeled making it hard to use. Since we haven't covered counters yet, a counter macro is supplied to you on the lab web page.

You will also find a macro for an 8-to-1 DEMUX on the lab page. It is not exactly like the 74155 (which can also be used as two 4-to-1 DEMUXes), but it does have the low-active outputs for this lab.

ECE 2090 - Lab 7

Four-Bit Combinational Multiplier

PURPOSE

To practice the combinational design process through the design of a 4-bit multiplier.

EQUIPMENT

Simulation Software

REQUIREMENTS

- Electronic copy of your design.
- Schematic of final design.
- Printout(s) of any macro(s) you used in your design.
- Brief report describing your circuit design and answering questions asked in the procedure.

INTRODUCTION

This lab will be unlike previous labs in two very significant ways. First, we're going to design circuits that are too complex to fit on our breadboard with the parts in the lab kit, so this lab will be done *entirely* with the simulator. Second, though this lab procedure will provide some insights into a possible design of a multiplier, no circuit diagrams will be provided—you will do the design completely on your own.

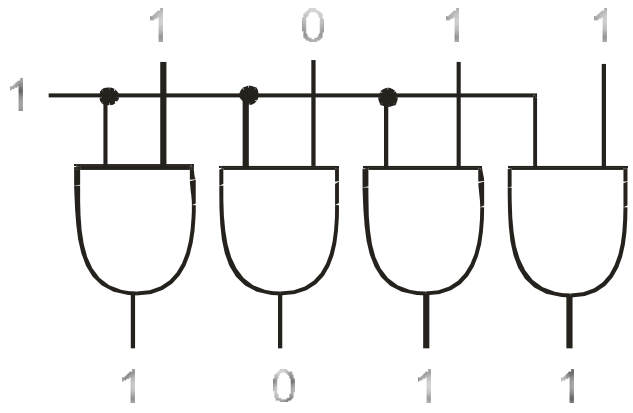
The circuit you will be designing is a four-bit multiplier, that is, a circuit which inputs two 4-bit numbers, and outputs their 8-bit product.

A truth table for this circuit would have eight inputs, eight outputs, and 256 rows. You would need eight, 8-variable Karnaugh maps to directly produce Boolean equations. Since 8-variable Karnaugh maps would be quite unwieldy, and we have not learned the Quine-McCluskey algorithm, we're going to have to think some about this one. The following section describes the multiplication process in detail, using steps that we've already figured out how to do. After that, it's up to you.

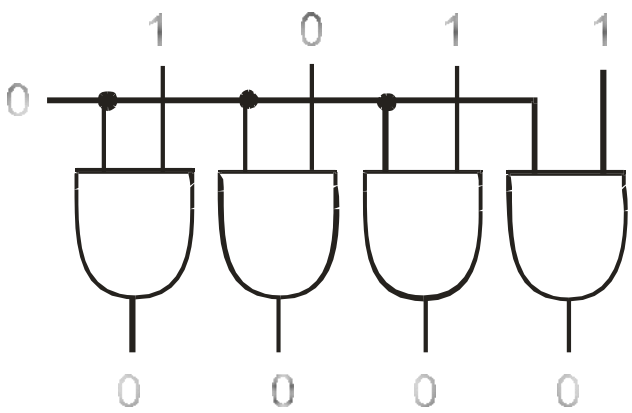
BACKGROUND

The first thing you need to know about multiplication is that the AND operation works just like decimal multiplication: 1•1 = 1, and 0•0 = 0•1 = 1•0 = 0. The AND operator even has the same symbol as decimal multiplication. So, if you need to multiply a number by a single bit, you can simply use AND gates. Consider the example below:

```
  1 0 1 1
x       1
  1 0 1 1
```




```
  1 0 1 1
x       0
  0 0 0 0
```



Things get a little more complicated when your multiplier has more than one bit. The Boolean AND operator functions only with Boolean variables, so we can only AND bits together, not binary integers. So, how do we multiply multi-bit numbers together? One way is to do things the same way we would perform the multiplication on paper: multiply the first number by the least significant bit of the second number, then add that result to the one you get by multiplying by the next bit, and so on, as shown in the example below:

```
          1 0 1 1          11
x       1 1 0 1        x  13
        1 0 1 1          33
      0 0 0 0           11
    1 0 1 1
  1 0 1 1
1 0 0 0 1 1 1 1   128+15 = 143
```

Note that we get an eight bit number from multiplying two, four-bit numbers.

In the general case, we can rewrite the problem symbolically below, where P_{ij} is the product (AND) of bit i of the multiplicand (top number) with bit j of the multiplier, and S_i is the sum of the column of numbers above it.

				A_3	A_2	A_1	A_0
			X	B_3	B_2	B_1	B_0
				P_{30}	P_{20}	P_{10}	P_{00}
			P_{31}	P_{21}	P_{11}	P_{01}	
		P_{32}	P_{22}	P_{12}	P_{02}		
	P_{33}	P_{23}	P_{13}	P_{03}			
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

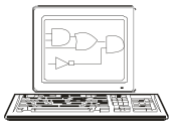
That's pretty close to something we can build. We can multiply by single numbers, and we can add numbers together. The only complication is the adders we know how to build can only add two numbers at a time. In this case we need to add up to five bits at a time, so we might want to create some partial sums to make use of the adders we have. A possible way to do this is shown below:

				A_3	A_2	A_1	A_0
			X	B_3	B_2	B_1	B_0
				P_{30}	P_{20}	P_{10}	P_{00}
		+	P_{31}	P_{21}	P_{11}	P_{01}	
		S_{05}	S_{04}	S_{03}	S_{02}	S_{01}	S_{00}
	+	P_{32}	P_{22}	P_{12}	P_{02}		
	S_{16}	S_{15}	S_{14}	S_{13}	S_{12}	S_{11}	S_{20}
+	P_{33}	P_{23}	P_{13}	P_{03}			
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

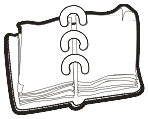
where S_{ij} represents the j^{th} output of the i^{th} adder.

Notice that S_{05} is the carry of the first adder, S_{16} is the carry of the second adder, and S_7 is the carry of the third adder. Also note that S_{00} is simply P_{00}, S_{11} is equal to S_{01}, and S_2 is S_{12}. Therefore, we need only three four-bit adders, which we have already built in lab.

PROCEDURE

 Bring an electronic copy of your design to lab with you, and be prepared to demonstrate the operation of your functional multiplier. Add appropriate switches and lights so that you can input two 4-bit numbers and see your 8-bit output.

 Turn in a printout of your final schematic, plus printouts of any macros you used in your designs as well.

 Turn in a brief report describing your circuit design. In your report, consider if and how well this circuit design would scale up to multiplying bigger numbers.

? Answer the following questions in your report:

- What would be the strengths and weaknesses of a large multiplier built in this fashion?
- What would the propagation delay be for the circuit in your simulation, assuming a 1 ns delay for each gate?

A Note on Multipliers

The multiplier you just built is pretty close to the kind used in high-end arithmetic circuits: a kind of multiplier known as an *array multiplier*. Real array multipliers pass the carry through the adders to the last one, rather than using ripple or look-ahead adders to add the partial sums. A special look-ahead adder is designed for the last stage to speed up the process. Multipliers built in this way are produce about the highest performance possible. Many multipliers, however, are not built in this fashion. A typical chip will use a much smaller, slower, but more complicated multiplier design based on registers which will be discussed in class.

ECE 2090 - Lab 8

Logic Design for a Direct-Mapped Cache

PURPOSE

To understand the function and design of a direct-mapped memory cache.

EQUIPMENT

Simulation Software

REQUIREMENTS

- Electronic copy of your design.
- Schematic of final design.
- Printout(s) of any macro(s) you used in your design.
- Brief report describing your circuit design and answering questions asked in the procedure.

BACKGROUND

Memory

A computer's memory is organized as a linear collection of storage locations. Each location has an *address.* Typical computer memories are *byte-addressable,* meaning that each byte in the memory has its own unique address (even though you usually read the memory a word at a time, and a word is typically greater than a byte).

The size of the address limits the amount of memory the computer can see. Most computers use at least a 32-bit address bus, meaning they can address 2^{32} Bytes, or 4 GB of memory. Naturally, the first Byte of memory is located at address 00000000_{16} and the last one is located at $FFFFFFFF_{16}$.

One of the most fundamental things a computer has to do is retrieve information from memory. The main memory of a computer stores both the programs the computer will run and the data the programs need. During operation, a computer's processor continuously generates a stream of addresses, first to get instructions out of the memory and then to fetch the data (operands) that those instructions need.

Memory Hierarchy

Unfortunately, memory is relatively slow and expensive, when compared to processors. As a result, a

CPU can spend a lot of its time doing nothing while it waits for a memory request to complete. In fact,

bandwidth (the amount of data you can transmit in a given time interval) to memory is one of the primary limiting factors in a computer's efficiency. (It is called the *von Neumann bottleneck*). To deal with this problem cost effectively, a *memory hierarchy* has been created. The basic concept is that data a processor needs most often will be kept in a small, fast, but expensive memory; and less frequently used data will be kept in progressively larger, slower, and cheaper memories. The fastest memory are *registers* on the CPU constructed of very fast transistors—the so-called *on-chip* or *Level 1 cache*. The second level of the hierarchy is typically an off-chip, or *L2 cache*, made up of a fast static RAM. The third level is main memory generally made up of slower, less-expensive dynamic RAMs, and the fourth level is the very large, slow, and inexpensive secondary storage such as a disk drive.

When the computer needs to read a byte of memory, it generates an address. The next step is to figure out where the data associated with that address is currently residing - is it in one of the caches, main memory, or on disk? The first thing to check is the L1 cache. If the location desired is there, this is known as a cache *hit*, and the value can be read immediately. If the location is not there, a cache *miss* occurs, and the memory hierarchy must be searched through until the data (and that around it) is located and brought into the L1 cache.

Caches

The purpose of this project is to build the logic that determines if the data located at a given address is or isn't in the cache. The output of your circuit will be a signal indicating whether we have a cache hit or miss.

We will be looking at a scheme known as *direct-mapped* caching. So, before we can get started, we need to examine how a direct-mapped cache works. A cache is divided into a number of *lines.* A line is a contiguous block of bytes in the memory that are stored at a single cache address. We will be designing circuits for a fictional computer which has a 16-bit address bus and a cache made up of 16 lines, containing 256 Bytes each. This means our computer will be able to address a total of 2^{16} or 64 KB of RAM, and at any time a maximum $16 \cdot 256 = 4096$ or 4 KB of this memory will be in the cache.

To use our direct-mapped cache, we will break the address into three fields. Since the 256 bytes that make up a line/block are contiguous, the last 8 bits (2^8 =256) of any address is the *offset* into the cache line. Since there are 16 lines in the cache, we need four bits of the address to determine which line in the cache the address will be in. The upper four bits of the address are the cache *tag* telling us which block is present. The decomposition of the address into fields is shown in the following figure :

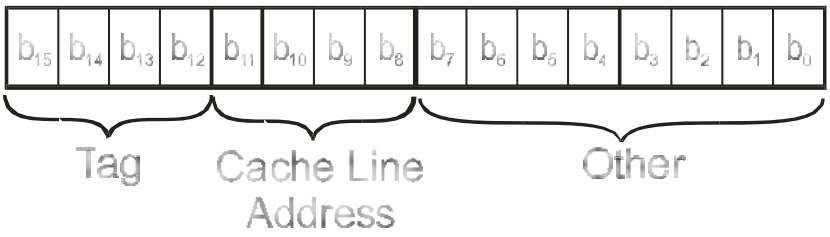
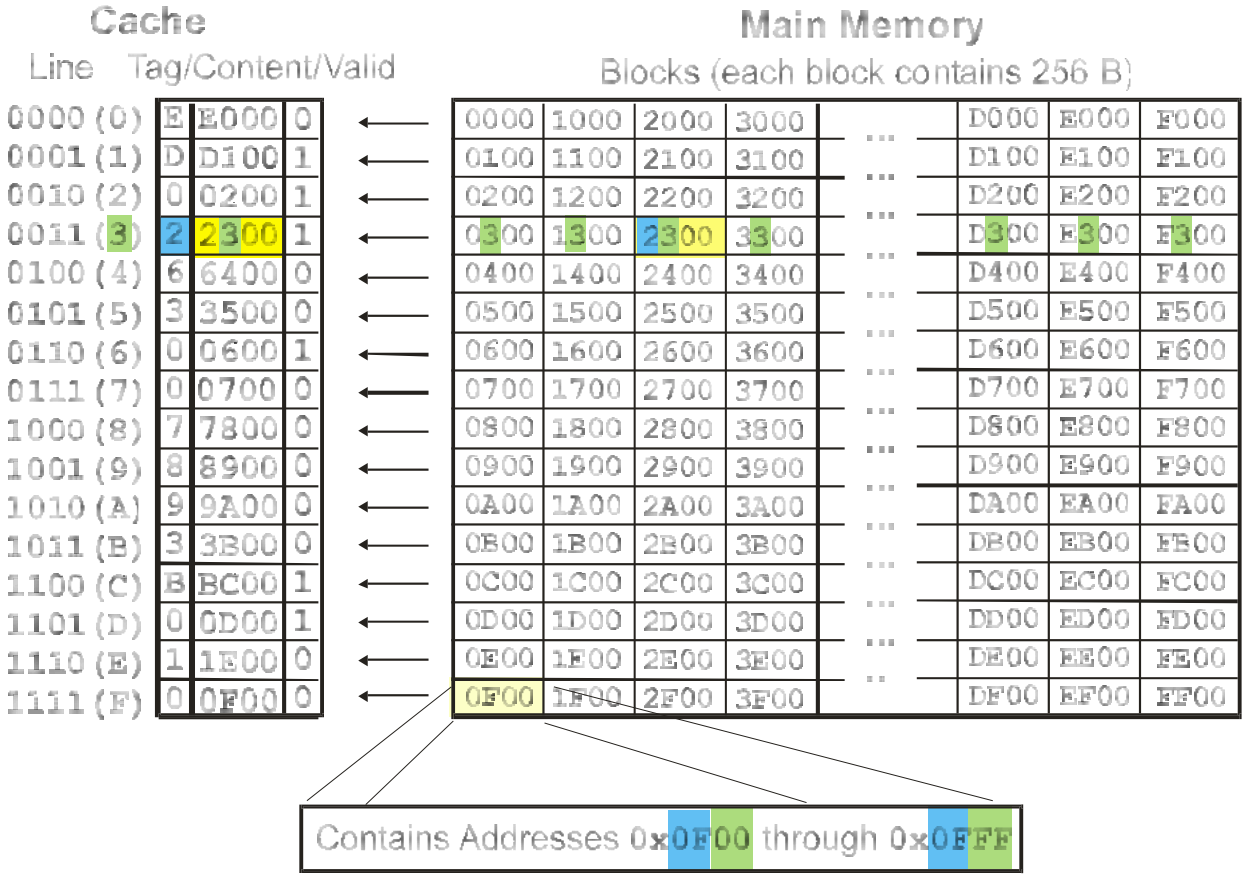


Figure 1. Fields of a Memory Address

For example, an address of **1111**0010**00110010** (0xF232) has a tag of 0xF, a line address of 0x2, and an offset of 0x32.

This type of cache is called "direct-mapped" because there is only one place in the cache where each address in memory may reside. However, since the cache is only 4 kB, and the whole memory is 64 kB, only 16 of the possible 256 lines can be in the cache at any one time. The mapping of 256 bytes of memory to the cache lines is shown below. Only memory addresses whose cache line address field matches can be in a particular cache line. Therefore, if you are looking at cache line address 3, only memory addresses who have the bits 0011 in bits b_{11} – b_8 can go into that cache line. That is, the only 256-byte blocks whose addresses start at 0x0**3**00, 0x1**3**00, 0x2**3**00,... 0xF**3**00 can go into cache line 3. Blocks that fit into a particular cache line are shown across a row in the figure below.



Note that for every address that goes into a single cache line, the line address is always the same, and there is exactly one block possible for each of the 16 tags. Therefore, to see if you have a cache hit, you can ignore the offset, and simply compare the tag to the tag currently stored in the appropriate line of the cache. To make this easier, associated with the cache there is a *tag memory*, a small memory that stores the tag of each block currently stored in the cache line. There is also one additional "valid bit" with each tag entry which indicates if any valid block is stored at that cache line (at power on, for instance, none of the lines are filled yet - there are several other cases when cache lines are invalid that we won't go into). For the cache described above, a 16x5 memory would be needed to store the 16 4-bit tags plus their corresponding valid bits.

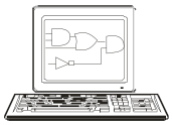
PROCEDURE

You are to design and simulate a circuit which determines if a cache hit or a miss is made for the system described above. You will have a 16x5 tag memory, which you will implement using a ROM (or RAM, since they're exactly the same in Logisim). You will initialize the tag memory according to the table below.

Address	Valid/Tag
0000	1 0001
0001	1 0001
0010	1 1111
0011	1 1110
0100	0 0001
0101	0 0101
0110	0 1010
0111	0 1010
1000	1 0000
1001	1 0000
1010	1 0000
1011	1 0111
1100	0 0000
1101	0 1111
1110	1 0010
1111	1 0011

The most significant bit in each word will be the valid bit, and the other four bits will be the tag. You will receive as input a 16-bit address. You will use the cache line address bits to retrieve the

appropriate tag from your memory. Then you will compare the tag you got from the memory to the tag in your address using a 4-bit comparator (you are expected to design a 4-bit comparator macro - see below). You will have one output, “hit”, which will be a 1 if the tags are equal and the valid bit is 1 and a 0 otherwise. Connect eight switches to the inputs of your simulation (corresponding to address bits 8-15) so your instructor can enter addresses to test the design. Wire the hit signal to an LED.



Bring an electronic copy of your design to lab with you, and be prepared to demonstrate the operation of your functional memory cache. Add appropriate input switches and an output light so that you can input the eight high-order bits of the 16-bit address (consisting of a 4-bit tag and 4-bit cache line) and see your LED output. A cache hit is lit and a cache miss is unlit.

A Note on Comparators

Recall that the XNOR function is also called equivalence, where

$$(x \oplus y)' = xy + x'y'$$

Also recall that two binary numbers would be equivalent if the first bits were equivalent *and* the second bits were equivalent *and* the third bits were equivalent, and so on.

ECE 2090 - Lab 9

Sequential Design – Three Bit Counter

PURPOSE

To understand the design and restrictions of Sequential Circuits.

EQUIPMENT

ECE 2090 Lab Kit & NI ELVIS

II Simulation Software

REQUIREMENTS

- Electronic copy of your design.
- Schematic of final design.
- State Transition Tables.
- Karnaugh Maps with Boolean reductions for each variable.
- Brief report describing your circuit design and answering questions asked in the procedure.

PROCEDURE

Assume that you have just designed and built a sequential circuit for a robot which will complete the remainder of your engineering classes for you. All you need to complete it is an up-down three-bit counter. It is Sunday night and you have three tests on Monday. Unfortunately, Radio Shack is closed so you will have to make do with your 2090 Lab Kit, or you will have to take those tests yourself.

Looking in your parts bag you find the following parts left:

- 1 - 7476 Dual J-K Flip-flop.
- 1 - 74175 Quad D Flip-flop.

With several of each of the following:

- 7408 - Quad 2-input AND
- 7432 - Quad 2-input OR
- 7486 - Quad 2-input XOR

Next, you realize that the only thing remaining for you to construct the counter on is your *NI ELVIS II* board which limits you to only five chips.

It is immediately obvious that the 7476 is insufficient since you need three flip-flops, thus you start by choosing the 7415 which contains four flip-flops. Next you do a quick design, and using C as the control bit variable (C = 0 means count up, C = 1 means count down) and $Q_3Q_2Q_1$ as the counter bits you obtain the following equations for the three inputs to the D flip flop:

$$D_3 = CQ_3'Q_2'Q_1' + C'Q_3'Q_2Q_1 + C'Q_3Q_2' + CQ_3Q_1 + Q_3Q_2Q_1'$$

$$D_2 = CQ_2'Q_1' + CQ_2Q_1 + C'Q_2'Q_1 + C'Q_2Q_1'$$

$$D_1 = Q_1'$$

With only two-input AND and OR gates, the decoding for D_3 alone would require four packages, thus the *NI ELVIS II* is not large enough. Leaving that for the moment, you move on to D_2. This looks bad superficially but after a moment's reflection you realize that can be implemented with dual 2-input XOR gates. D_1 must have Q'_1, but this is available on the 74175. Well, two out of three implemented and less than two full chips used isn't bad for a start.

If you could implement the remaining bit with three or fewer chips (plus two XOR gates) you'll have the rest of the semester off! You next try using a J–K flip-flop for bit three. (You decide trying to figure out how to use some XOR's to implement D_3 would take too long and might not work anyway.) As it turns out, the J and K inputs for Q_3 can be realized using one 7408 and one OR gate.

Determine the appropriate circuitry for J and K, and connect the whole mess using a switch for the control bit. In addition, connect one of the switches to the *clear* inputs of the 7476 and the 74175 to allow presetting of the counter to 000. (This should be the switch that goes low when the button is pressed – you can use pushbutton (PB) switch provided on the NI ELVIS board. These switches are pre-configured for active-low operations.)

Note that the preset pin of the 7476 *must* be tied to +5V.

Use three lights for the output. (Which bit goes to the left most light?) Initially, use the generated pulse (or square waveform) at 1 Hz to clock the circuit.

HINTS

- Make out the complete state transition table to get J and K. This is not absolutely necessary but may avoid confusion and errors. Compare the equations for J and K!
- If you don't immediately see how to implement D_2 with two XOR's, factor out the c and c' terms to recognize the XOR.
- Use an extra XOR to invert c as you did for the adder/subtractor circuit.

Verify that the circuit counts up if c is 0 and down if c is 1. Also verify that the active-low switch clears the counter.

Note: The clock of the 74175 must be inverted relative to that of the 7476 - invert the clock with an XOR.

! Watch the power connections on the 7476. If you hook up pin 13 to +5V and pin 5 to Ground, You will destroy the chip!

? Is it possible to implement D_3 within the stated limitations? If so, how?