Texas A&M University
Department of Electrical Engineering

ELEN 248

Introduction to Digital Design

Laboratory Manual

Prepared and Revised by

Daryl Reynolds
David Rigsby
Prof. Ray Mercer
Lifford McLauchlan

Dec. 2004

CONTENTS

ELEN 248 Laboratory Policies and Report Format 1

Lab 1: Introduction to Combinational Design 3

1.1 Introduction
1.2 Background
1.3 Prelab
1.4 Lab Procedure
    1.4.1 Experiment 1
    1.4.2 Experiment 2
    1.4.3 Experiment 3
    1.4.4 Experiment 4
    1.4.5 Experiment 5
1.5 Post lab

Lab 2: Introduction to Max+Plus II 11

2.1 Introduction
2.2 Background
2.3 Prelab
2.4 Lab Procedure
    2.4.1 Experiment 1
    2.4.2 Experiment 2
    2.4.3 Experiment 3
    2.4.4 Experiment 4
2.5 Postlab

Lab 3: Adders 17

3.1 Introduction
3.2 Background
3.3 Prelab
    3.3.1 Design 1
    3.3.2 Design 2
    3.3.3 Design 3
    3.3.4 Questions
3.4 Lab Procedure
    3.4.1 Experiment 1

3.4.2 Experiment 2
3.4.3 Experiment 3
3.4.4 Experiment 4
3.5 Postlab

Lab 4: Signed Numbers 21

4.1 Introduction
4.2 Background
4.2.1 Two's Complement Numbers
4.2.2 Signed Integer Arithmetic
4.3 Prelab
4.3.1 Design 1: Negation (Two's Complement)
4.3.2 Design 2: Signed Integer Adder
4.3.3 Design 3: Signed Integer Subtracter
4.3.4 Questions
4.4 Lab Procedure
4.4.1 Experiment 1
4.4.2 Experiment 2
4.4.3 Experiment 3
4.5 Postlab

Lab 5: 8-bit Counter with Debounced and Non-debounced Clock 26

5.1 Introduction
5.2 Background
5.2.1 The 8count 8-bit counter
5.2.2 Clock bounce
5.2.3 The SR latch
5.2.4 The seven segment display
5.3 Prelab
5.3.1 Design 1
5.3.2 Design 2
5.3.3 Design 3
5.3.4 Questions
5.4 Lab Procedure
5.4.1 Experiment 1
5.4.2 Experiment 2
5.5 Postlab

Lab 6: 4-bit Shift Adder 32

6.1 Introduction
6.2 Background
6.2.1 The 74179 shift register

6.2.2 Binary addition via shifting
6.3 Prelab
6.3.1 Design 1
6.3.2 Design 2
6.3.3 Questions
6.4 Lab Procedure
6.4.1 Experiment 1
6.4.2 Experiment 2
6.5 Postlab

Lab 7: 4-bit Shift Multiplier 36

7.1 Introduction
7.2 Background
7.2.1 The Altera internal clock and the clock divider circuit
7.2.2 4-bit shift multiplication
7.2.3 How to do it
7.3 Prelab
7.3.1 Design 1
7.3.2 Design 2
7.3.3 Questions
7.4 Lab Procedure
7.4.1 Experiment 1
7.4.2 Experiment 2
7.5 Postlab

Lab 8: The Priority Encoder 41

8.1 Introduction
8.2 Background
8.3 Prelab
8.3.1 Design 1
8.3.2 Design 2
8.4 Lab Procedure
8.4.1 Experiment 1
8.4.2 Experiment 2
8.5 Postlab

Lab 9: T-Bird Taillights 43

9.1 Introduction
9.2 Background
9.2.1 State Machines
9.2.2 T-bird Taillights

9.3 Prelab
9.3.1 Design 1
9.3.2 Design 2
9.3.3 Design 3
9.3.4 Design 4
9.3.5 Questions
9.4 Lab Procedure
9.5 Postlab

Lab 10: LED Pong 48

10.1 Introduction
10.2 Background
10.3 Prelab
10.3.1 Design 1
10.3.2 Design 2
10.3.3 Design 3
10.3.4 Design 4
10.4 Lab Procedure
10.5 Postlab

ELEN 248 Laboratory Policies and Report Format

Reports are due at the beginning of the lab period. The reports are intended to be a compete documentation of the work done in preparation for and during the lab. The report should be complete so that someone else familiar with digital design could use it to verify your work. Portions of the prelab needed during lab should be duplicated before you arrive since the TA will collect the original. The prelab and postlab report format is as follows:

1. A neat thorough prelab must be presented to your lab instructor at (or before) the beginning of your scheduled lab period. Lab reports should be submitted on 8.5" x 11" paper, typed on one side only. Your report is a professional presentation of your work in the lab. Neatness, organization, and completeness will be rewarded. Points will be deducted for any part that is not clear.

2. Each report will contain the following sections:

   a) **Cover Page:** Include your name, ELEN 248, Section No., Lab No., TA's name, and date

   b) **Objectives:** Enumerate 3 or 4 of the topics that you think the lab will teach you. DO NOT REPEAT the wording in the lab manual procedures. There should be one or two sentences per objective. Remember, you should write about what you will *learn,* not what you will *do*. These are not necessarily the same things.

   c) **Design:** This part contains all the steps required to arrive at your final circuit. This should include diagrams, tables, equations, K-maps, explanations, etc. Be sure to reproduce any tables you completed for the lab. This section should also include a clear written description of your design process. Simply including a circuit schematic is not sufficient.

   d) **Schematics:** As part of the design process for each lab (with the exception of the first two), you will create a gate-level schematic and turn it in with your report. The schematic must be complete. You should be able to copy it directly into Max+II and create a working circuit. Since the design process is completed before you come into lab, you may turn in a hand-drawn schematic, but it must be neatly drawn. Schematics that are difficult to read will receive no credit.

   e) **Questions:** Specific questions asked in the lab should be answered here.

3. Late prelabs will have 50% of the points deducted for being one day late. If a prelab is 2 days late, a grade of 0 will be assigned. The prelab for a two-week lab is due at the beginning of the first week's lab period.

4. Your work must be original and prepared independently. However, if you need any guidance or have any questions or problems, please do not hesitate to call your Teaching Assistant (TA) during office hours. Copying any prelab will result in a grade of 0. The incident will be formally reported to the University.

5. Each laboratory exercise (circuit) must be completed and demonstrated to your TA before the prelab for the subsequent lab is due (1-2 weeks later) in order to receive working circuit credit. This is the procedure to follow:

   a) **Circuit works:** If the circuit works during the lab period (3 hours), call your TA, and he/she will sign and date it. You should then save your circuits on your own Zip disk before leaving the lab. This is the end of this lab, and you will get a complete grade (100) for this portion of the lab.

   b) **Circuit does not work:** If the circuit does not work, you must make use of the open times for the lab room (Zachry 115D) to complete your circuit. When your circuit is ready, contact your TA to set up a time when the two of you can meet to check your circuit.

6. You are guaranteed a computer and workspace in 115D Zachry only during your lab period. If you need to work on your circuits at a time other than your regularly scheduled lab period, the lab in 115D either is open or can be opened any time the instrument room is open (8 am - 9 pm M-F and 9 am - 5 pm Saturday). However, if another lab section is in progress, ask the TA if he/she has any open lab stations.

7. Attendance at your regularly scheduled lab period is required. An unexpected absence will result in loss of credit for your lab. Your lab instructor may permit rescheduling if you arrange for the change ahead of time.

Lab 1: Introduction to Combinational Design

1.1 Introduction

The purpose of this experiment is to introduce you to the basics of circuit wiring, troubleshooting, positive/negative logic, threshold voltages, clock, delay concepts, and gate behavior. In this lab, you will test the behavior of several of the basic logic gates and you will connect several logic gates together to create simple circuits.

1.2 Background

None.

1.3 Prelab

None.

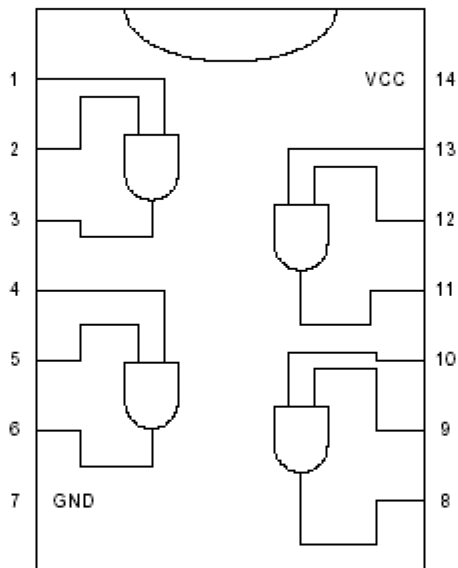
1.4 Lab Procedure

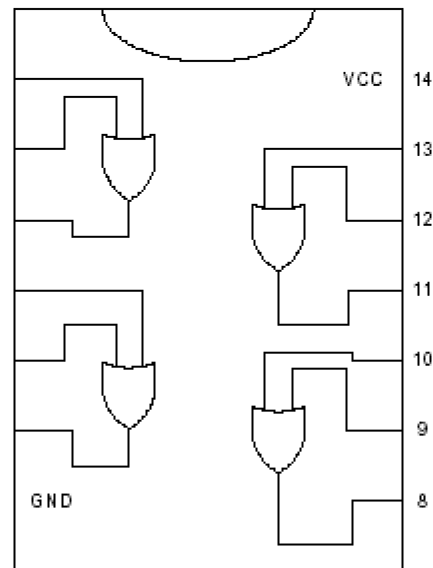
We will look at the behavior of some basic logic gates, including inverters, AND gates, OR gates, XOR gates, NAND gates, and NOR gates. Each of these gates is embedded in an integrated circuit package. See Figures 1.1 and 1.2 for pinouts for these circuits. More detailed specifications may be found in the TTL (Transistor-Transistor-Logic) book found in the instrument room (Zachry 111A). We will also investigate the concepts of positive and negative logic, threshold voltages, clock pulse, and delay.

1.4.1 Experiment 1

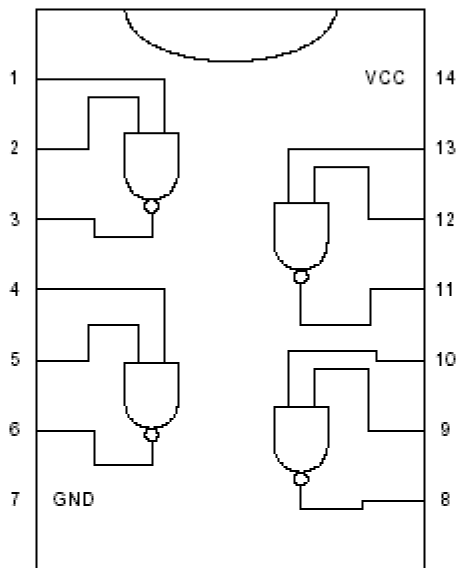
We will start by setting up the DC power supply and multimeter for our use. Be sure both are turned off. Then check to see that the multimeter is set to measure DC (button on far left), and be sure the red lead is connected to the red multimeter input that is marked for voltage (not current). Finally, set the scale to the 20V scale. Now, set the DC power supply voltage output to zero (turn the coarse adjustment counterclockwise until it stops). Connect the red lead of the power supply to the red lead of the multimeter. Likewise, connect the black lead of the power supply to the black lead of the multimeter.

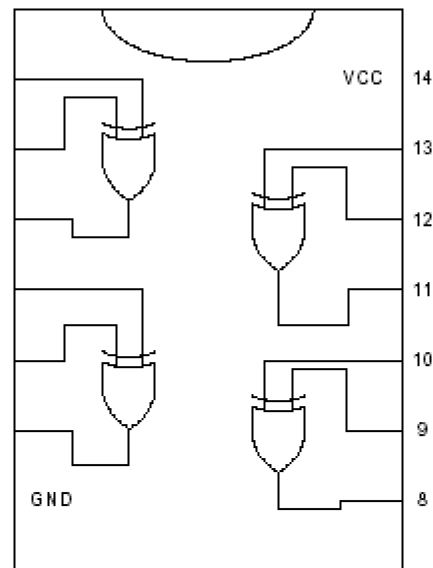
***** NOTE - DO NOT CONNECT POWER (RED) AND GROUND (BLACK) TOGETHER. This will cause a short.**





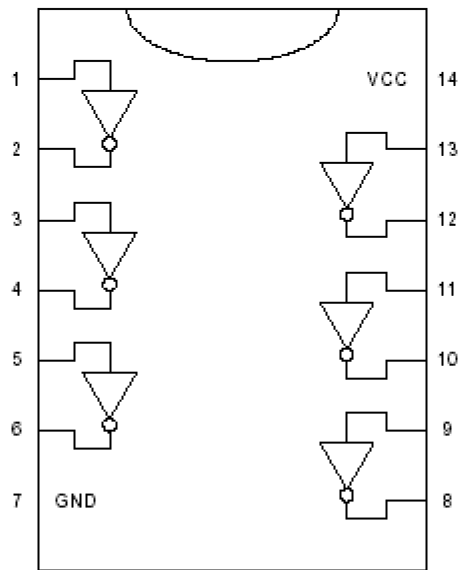
7408 7432

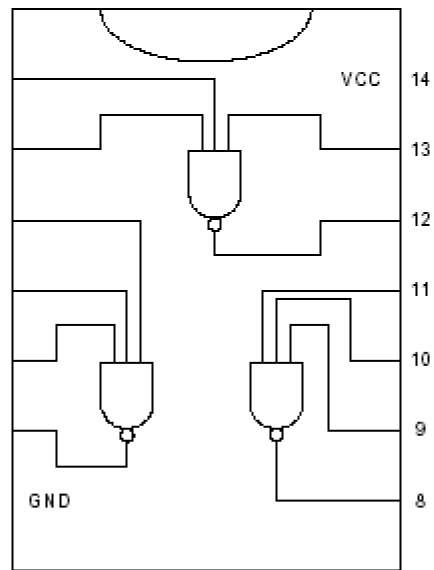


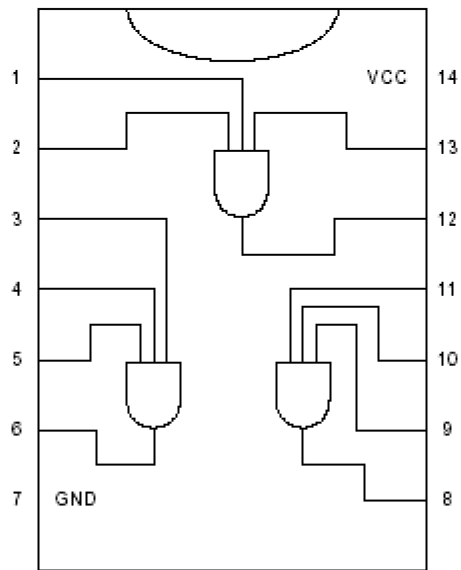


7400 7486

Fig. 1.1: Pinouts of the 7400 series TTL logic gates.







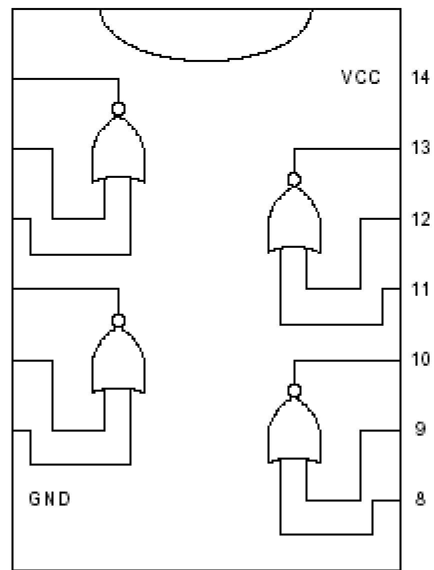


Fig. 1.2: Pinouts of the 7400 series TTL logic gates.

Turn on both the multimeter and the power supply. The multimeter should read very near zero. Turn the coarse adjustment clockwise until the multimeter reads 5V. If the multimeter display does not change significantly when you turn the coarse adjustment, turn the power supply off and recheck your connections. You may have a short. When the multimeter reads 5V, the adjustments are complete and you should turn off the power supply. You are ready to test your first gate. We will start by wiring a 74LS04 (inverter) gate. The pinouts for this chip may be found in Figure 1.1. Insert the 74LS04 chip onto the breadboard. Be sure you are not shorting pins together. **Identify the power (VCC) and ground (GND) pins for the 74LS04**. Connect the VCC pin to the red lead of the power supply and connect the GND pin to the black lead of the power supply (or multimeter). This chip (7404) contains 6 different inverter gates. Each inverter gate has an input pin and a corresponding output pin. Choose one of the gates and connect the red lead of the multimeter to the gate output. The black lead of the multimeter should always be connected to the black lead of the power supply (at the GND pin). Then connect a wire from either the VCC pin to the input (for a logic high input) or from the GND to the input (for a logic low input). Do not connect both at the same time, as this will cause a short. Turn on the power supply and observe the gate output.

Assume A is the input to the inverter (either H or L) and that Y is the output. Fill in Table 1.1 according to the logic behavior that you observe.

Note: First fill in the second column of the table using the readings from the multimeter. Then determine the answers to the last column based upon these readings. If the output is high (H), the multimeter will read approximately 4.4 volts; when it is low (L), the multimeter will read about 0.15 volts. If you read a voltage between these values, you have likely wired your circuit incorrectly.

A	Y (Volts)	Y (H/L)
L		
H		

Tab. 1.1: Inverter logic behavior

1.4.2 Experiment 2

In a circuit, logic variables (values 0 and 1) can be represented as levels of voltage. The most obvious way of representing two logic values as voltage levels is to define a threshold voltage; any voltage below the threshold, represents one logic value and voltages above the threshold correspond to the other logic value.

To implement the threshold – voltage concept, a range of low and high voltage levels is defined, as shown in Fig. 1.3. This figure indicates that voltages in the range Gnd to Vo,max represent logic value 0. Similarly, the range from V1,min to Vcc corresponds to logic value 1. Logic signals do not normally assume voltages in undefined range except in transition from one logic value to the other.

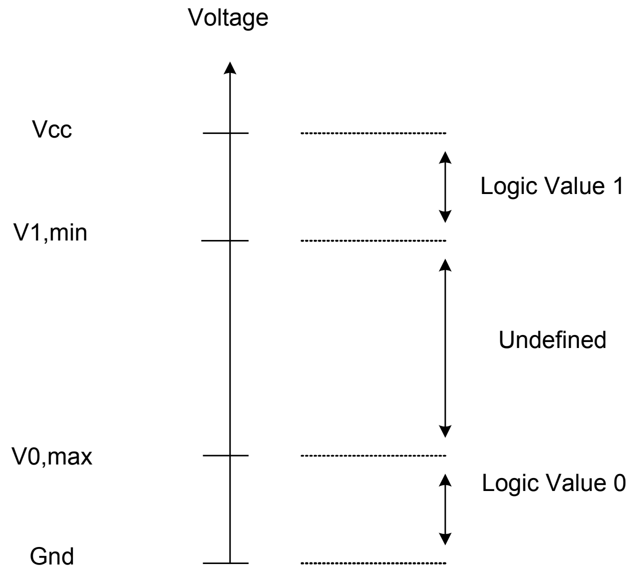


Fig. 1.3: Representation of logic values by voltage levels

Creating a variable input voltage from 0 to 5 volt, you will use a potential meter connected between Gnd and Vcc. Then connect its third pin (middle pin) to input of an inverter gate and measure its output voltage with the multimeter while the input voltage of the gate changes from 0 to 5 volt. Draw Vo–Vin curve for the inverter gate, what are the threshold values for high and low logic levels? Demonstrate the results to your TA.

1.4.3 Experiment 3

We are going to repeat the same experiment with the gates 74LS00 (NAND), 74LS02 (NOR), 74LS08 (AND), 74LS32 (OR), and 74LS86 (XOR). Note that each of the gates has two inputs and one output. Fill in Tables 1.2 and 1.3 to indicate the observed responses of these gates.

So far we have reviewed the voltage behavior (H/L) of various gates. These devices will always behave according to the logic described in Tables 1.1-1.3. There is an abstract way to interpret this voltage behavior. We call this *logic interpretation*. Logic interpretation can be of two classes: positive or negative. The logic interpretation assigns logic levels (0 or 1) to the voltage inputs and outputs (H or L) of the devices. The levels are assigned according to Table 1.4. Use Table 1.4 to fill out Tables 1.5 and 1.6. Note that the tables should be filled with 0's and 1's, not H's and L's. If you still have questions, see your TA.

A	B	and2 (V)	and2 (H/L)	or2 (V)	or2 (H/L)	nand2 (V)	nand2 (H/L)
L	L						
L	H						
H	L						
H	H						

Tab. 1.2: Basic gate logic behavior, part I

A	B	nor2 (V)	nor2 (H/L)	xor (V)	xor (H/L)
L	L				
L	H				
H	L				
H	H				

Tab. 1.3: Basic gate logic behavior, part II

Positive Logic	Negative Logic
H=1	H=0
L=0	L=1

Tab. 1.4: Logic Interpretation

A	B	7400 Pos.	7400 Neg.	7402 Pos.	7402 Neg.	7408 Pos.	7408 Neg.
L	L						
L	H						
H	L						
H	H						

Tab. 1.5: Positive/Negative logic

A	B	7432 Pos.	7432 Neg.	7486 Pos.	7486 Neg.
L	L				
L	H				
H	L				
H	H				

Tab. 1.6: Positive/Negative logic

1.4.4 Experiment 4

This experiment will introduce you to the procedures of wiring schematics. Connect the circuit in Figure 1.4 and complete logic behavior Table 1.7 using this circuit. Assume positive logic. Demonstrate this circuit to your teaching assistant.

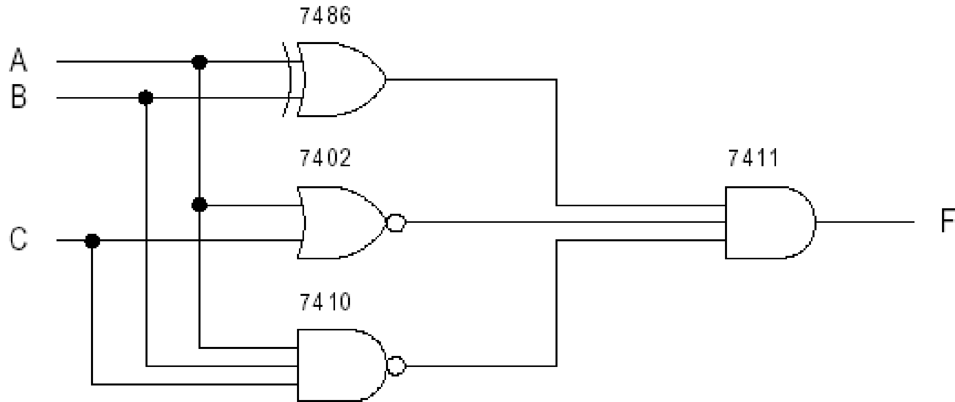


Fig. 1.4: Circuit A

A	B	C	F (V)	F (H/L)
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Tab. 1.7: Circuit A logic behavior

1.4.5 Experiment 5

This experiment will introduce you concept of clock signal and delay. Clock is a control signal that allows the changes in the states of digital circuits to occur only at well–defined time intervals, as if they were controlled by a clock. It should be noted that either rising clock edges, _|‾, or falling edges, ‾|_, is lead to changes in the states not its logic level (0 or 1). Actually clock is a periodic signal that is equal to 1 at regular time intervals to suggest that this is how the clock signal usually appears in a real system.
Connecting an odd number of inverter gates in series, you will make a clock signal. Fig 1.5 depicts the simple clock pulse circuit. You should connect the last gate output to first gate input. Then look at the output signal on oscilloscope and measure its frequency. Then calculate average delay of each gate and demonstrate your result to your TA. Keep in mind, more number of inverter gates, more accurate results.

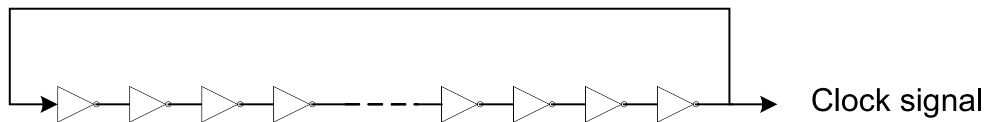


Fig. 1.5: Construction of a clock signal using inverter gates

1.5 Post lab

None.

Lab 2: Introduction to Max+Plus II

2.1 Introduction

In this lab you will repeat the experiments from Lab 1; however, this time you will be using the Altera MAX+plus II software to program the Altera EPM7128S **P**(rogrammable) **L**(ogic) **D**(evice). This is the simulation technique we will use for the rest of the semester.

2.2 Background

The only background material you need is an understanding of combinational circuits, covered in Lab 1.

2.3 Prelab

None.

2.4 Lab Procedure

To begin this lab, log on to the computer using your Novell Network login. A unique username and password is automatically assigned to every student that is registered for an electrical engineering laboratory. The username has the format “fml6789” where “fml” represent your initials and “6789” represents the last 4 digits of your student ID number. Your password is your entire student ID number, including dashes. Your account may not be active during the first couple of weeks of lab, so you may have to obtain a temporary login from your TA.

After you have logged in, find the Max+plus Icon and run the program (double-click the icon). Once in Max+, create a new file by clicking on the toolbar icon at the far left. A dialog window will appear and will ask what type of file you wish to create. Choose the .gdf file. This should be the default choice. Now proceed to the first experiment.

2.4.1 Experiment 1

As in Lab 1, we will look at the behavior of some basic logic gates. We will first program an inverter (not) gate. To do this, left click on your .gdf file at the location where you would like the inverter to be placed. You can always move it later. Then go to the **Symbol** pull down menu and select **Enter Symbol**. A dialog window will appear (see Figure 2.1). Type “not” on the top line of the dialog window and hit <enter>. An inverter gate will appear in your .gdf file. We now need to create an input pin and an output to connect to the inverter gate. Repeat the procedure above, but type “input” in the dialog window. This will give you an input pin. Now create an “output” pin. To create the wires to connect the pins to the gate, click on the line tool button, which is the

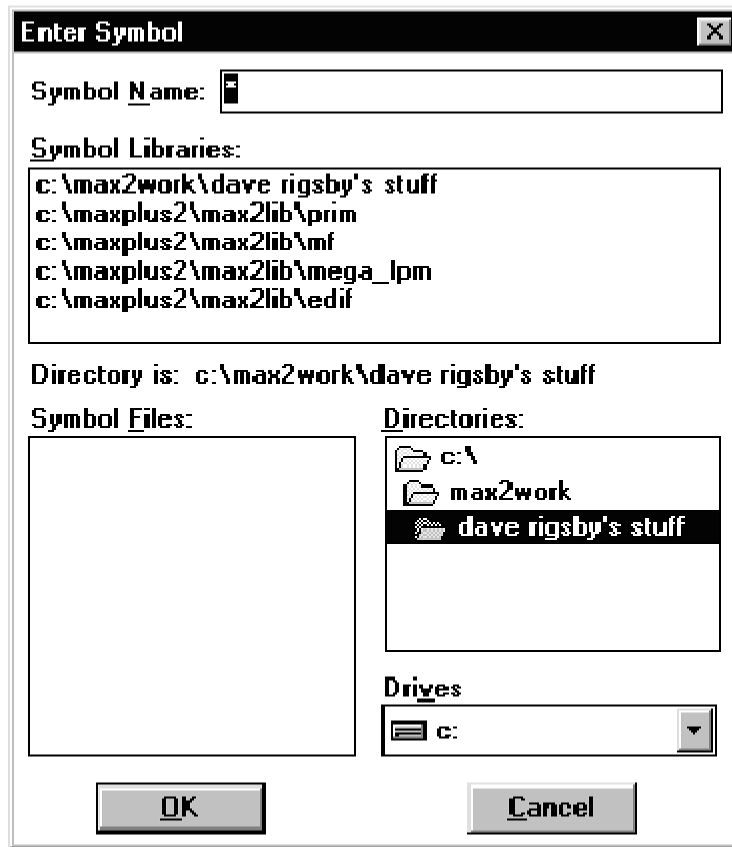


Fig. 2.1: Enter Symbol dialog box

button on the left with a picture of a right angle. You can then drag a straight line with one 90-degree bend to connect any two gates or wires in the schematic. There are two important things to remember when creating schematics in Max+II. First, always look for a dot to appear whenever you connect two wires together. If there is no dot, you cannot be sure that there is (isn't) a connection. Second, make sure that your wires are not laid across the dotted-line borders of your symbols, as this can cause errors.

Next, you must name your input and output pins. You can do this by double-clicking on the name shown (which defaults to PIN NAME) and typing a new name. Not that all pin names must be unique. Before you can compile your simulation file and program the PLD, you must make your .gdf file the "current project". The command to set the project name to your current file is found in the File menu. Scroll down to Project, which opens a submenu. Inside this submenu, scroll down to Set Project to Current File. Note that if you have not yet saved your program, you must do so at this point. All files must be saved to your Zip disk, which is the E: drive.

Now you can compile your circuit. This command can be found in the File menu, under Project, as before. In the same submenu, click Save and Compile. The compiler will open a window containing messages about your program's status. At this point, one of two things will happen. Either you will see messages in red, indicating errors, or the program will compile successfully.

Note that you may see warnings, but these can generally be ignored. If your file has errors then you will need to go back and correct them. Under some circumstances, you may double (left) click on the red message and the program will return to the .gdf file and highlight, in red, the part of the circuit that the program believes is in error.

Before you can program the EPM7128S PLD that is embedded in the Altera circuit board, you must assign specific PLD pin numbers to the input and output pins you have placed in your .gdf file. Many of the pins on the PLD have been prewired to switches, LEDs, or pushbuttons. Page 16 provides you with a pin-out that will describe these connections. To assign a pin number to one of your input/output pins, first right click on the pin. Choose **Assign,** then **Pin/Location/Chip** from the resulting pull down menu. A new dialog box will appear. The first time you specify a pin number, you have to push the **Assign Device** button and choose the PLD part number EPM7128S. Click **OK** to return to the first dialog box. Now place a pin number in the **Pin** box and click **OK**. Repeat this procedure for each pin in your .gdf file. Remember, you only have to **Assign Device** once.

You will find the Programmer in the MAX+plus II pull down menu; click on Program, and your program will be written to the chip.

After programming is complete, observe the behavior of the inverter circuit. Suppose A is the input to the inverter and Y is the observed response (H or L). Fill in Table 2.1. Remember that if an LED is on, it is indicating a logic L.

A	Y (H/L)
L	
H	

Tab. 2.1: Inverter logic behavior

2.4.2 Experiment 2

Perform the above experiment with gates and2, or2, nand2, nor2, and xor. Since each of these gates has two inputs and one output, you will need to create another input pin. Fill in Table 2.2.

A	B	and2	or2	nand2	nor2	xor
L	L					
L	H					
H	L					
H	H					

Tab. 2.2: Basic logic gate logic behavior

2.4.3 Experiment 3

Create the circuit in Fig. 2.2 using the Max+II software and complete the logic behavior Table 2.3. Assume positive logic. Demonstrate this circuit to your TA.

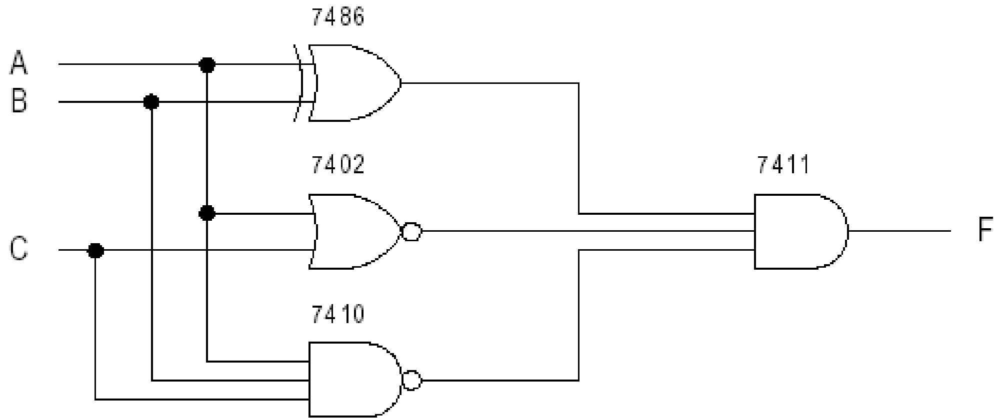


Fig. 2.2: Circuit A

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

Tab. 2.3: Circuit A logic behavior

2.4.4 Experiment 4

This experiment will introduce you one out of huge number of blocks that Max+II provide you to use in constructing digital circuits. Each of these blocks has its own special function. Accessing to each block, repeat the procedure in experiment 1 and type the name of the block in the dialog window. In this experiment you will connect three inputs and one output to the 21mux block and then fill in its truth table shown in Fig. 2.3. Guess its function and demonstrate to your TA.

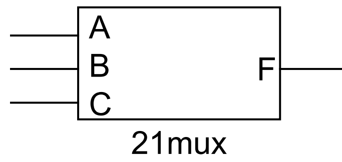


Fig. 2.3: 21mux block

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

Tab. 2.4: Multiplexer 2:1 logic behavior

2.5 Postlab

There is no postlab requirement for Lab 2. The prelab for Lab 3 is due at the beginning of lab next week.

Lab 3: Adders

3.1 Introduction

The purpose of this experiment is to introduce the design of simple combinational circuits, in this case half adders and full adders. The student should also be able to design an *n*-bit ripple carry adder using n full adders.

3.2 Background

We will be using two types of adders for this lab and throughout the semester. A half adder adds two bits together and therefore has possible sums of 0, 1, or 2. A full adder adds a carry-in bit to two bits and has possible sums of 0, 1, 2, or 3. See Figure 3.1

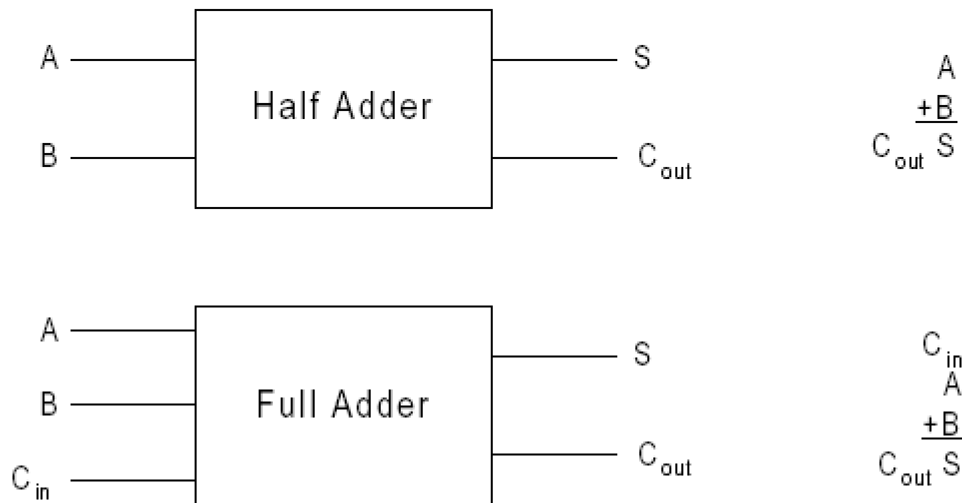


Fig. 3.1: Adders

To create an *n* bit ripple carry adder, you may connects *n* full adders together in the fashion of Figure 3.2. The carry-out from a full adder becomes the carry-in for the next significant bit in the next full adder.

In Figure 3.3 you can see the interaction of the input and output signals shown in Figure 3.2.

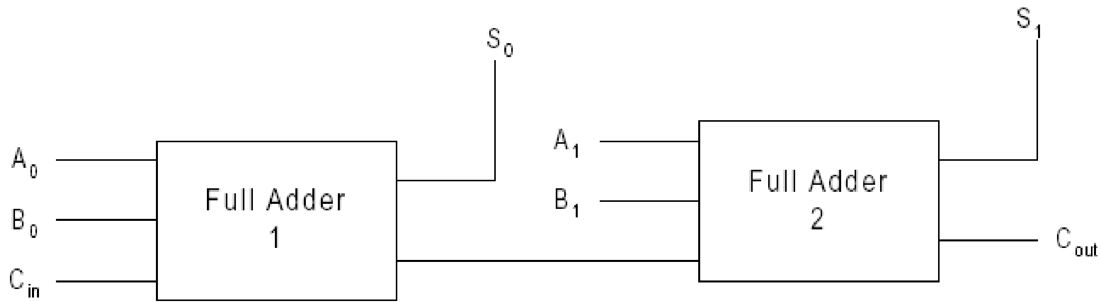


Fig. 3.2: 2-bit ripple carry adder

$$\begin{array}{rrr} & & cin \\ & a_1 & a_0 \\ + & b_1 & b_0 \\ \hline cout & s_1 & s_0 \end{array}$$

Fig. 3.3: 2-bit addition

3.3 Prelab

Complete Designs 1, 2, and 3, answer the questions, and follow the prelab report format carefully.

3.3.1 Design 1

Construct a truth table for a half adder circuit, including the sum and carry-out outputs. Write equations for the outputs of the half adder that are minimal in terms of the number of gates required for implementation. **Note that XOR and NOT gates count as one gate each.** Draw a gate-level schematic for the half-adder, including Max+II input and output pin numbers.

3.3.2 Design 2

Construct a truth table for a full adder circuit, again including the sum and carry-out outputs. Write the equation for each output that is minimal in terms of the number of gates required for implementation. Draw a gate-level schematic including Max+II input and output pin numbers.

3.3.3 Design 3

Design a 3-bit ripple carry adder using full adders as building blocks. Include the carry-in to the LSB. There should be 7 inputs: ($A_2A_1A_0$) one 3-bit operand, ($B_2B_1B_0$) another 3-bit operand, and C_{in}, the carry-in to the LSB of the adder. There should be 4 outputs:

($S_2S_1S_0$), which is the sum, and C_{out}, which is the carry-out of the MSB. Calculate the expected results for each test in Table 3.1

A_2	A_1	A_0	B_2	B_1	B_0	C_{in}	C_{out}	S_2	S_1	S_0
0	0	0	0	0	0	0				
0	0	0	0	0	0	1				
0	1	0	1	0	1	0				
0	1	0	1	0	1	1				
0	1	0	0	1	0	0				
0	1	1	0	1	1	0				
1	0	0	1	0	0	0				
1	1	1	1	1	1	0				
1	1	1	1	1	1	1				
0	0	1	1	1	1	1				

Tab. 3.1: 3-bit ripple carry adder logic behavior

3.3.4 Questions

1. How would you make a 6-bit ripple carry adder using **only** 3-bit adder blocks like the one in Figure 3.4? (Do not use individual full adders as building blocks.) Draw the schematic.

Fig. 3.4: 3-bit adder block

2. How would you make a 4-bit ripple carry adder using **only** 3-bit adders? Hint: use one 3-bit adder for the least significant 3 bits and use the most significant bit from the other 3-bit adder for the fourth bit. (Again, do not use individual full adders as building blocks.) Draw the schematic.

3. Using your design for the full adder, determine the worst case propagation delay from the inputs to the carry-out output bit. Assume each gate has the same delay, 1 gate delay.

4. Using your designs for the full adder and the 3 bit ripple carry adder (Design 3), determine the worst case propagation delay from the least significant input bit (of either input number A or B) to the carry-out output bit. Again, assume each gate has the same delay, 1 gate delay.

3.4 Lab Procedure

Complete Experiments 1-4.

3.4.1 Experiment 1

Implement and test your half adder design using the MAX+plus II software.

3.4.2 Experiment 2

Implement and test your full adder design using the MAX+plus II software. When it is working, demonstrate it to your TA.

3.4.3 Experiment 3

You will create symbols (modules) for your adder circuits so that they may be used easily in later labs. You can do this by opening the File menu and selecting Create Default Symbol. From this point on, when you open the Enter Symbol window, you should see your full adder in the ”Symbol Files” area.

3.4.4 Experiment 4

Implement your design for the 3-bit ripple carry adder from prelab design 3. Test it via Table 3.1 and demonstrate it to your TA.

3.5 Postlab

None.

Lab 4: Signed Numbers

4.1 Introduction

A digital system is one that manipulates numbers. Hence, in digital systems design three concepts play an important role:

- how to *represent* numbers
- how to *manipulate* the numbers
- how to represent and manipulate numbers *conveniently* and *efficiently*

You've already gained a taste for the first two concepts in previous labs. In Labs 1 through 3, you extended the numbers you dealt with from single-digit *bits* to multiple-digit *words*. As well, you experimented with manipulating these numbers with various operators, from the Boolean functions of Labs 1 and 2, to the arithmetic addition operation of Lab 3.
In this Lab, we are going to extend the representation of numbers from the positive integers to *signed* integers (i.e. both positive and negative integers). By doing so, we'll also introduce the concepts of *convenience* and *efficiency* into digital systems design.

4.2 Background

4.2.1 Two's Complement Numbers

There are many ways to represent numbers. Here, we will develop one possible way called the *Two's Complement* representation. We will find that this is a convenient and efficient manner of representation.

To begin, let's consider 3-bit numbers. With 3 bits, we can represent $2^3 = 8$ numbers. Table 4.1 shows the straightforward *unsigned* integer representation that you've explored in Lab 3. Now, the whole process of assigning a numerical magnitude to a particular representation is arbitrary and up to the designer's discretion. It is the *convenience* and *efficiency* of the assignment that makes a designer prefer one assignment over the other. For instance, in Table 4.1 we could have assigned "011" to represent 0, "111" to represent 1, etc. But, on inspecting Table 4.1, you can see that

1. the normal ordering of the positive integers is maintained by our assignment (*i.e.*subsequent assignments are greater than the preceding assignment by 1). This leads to an *efficient* assignment, as we can use the normal adders of Lab 3 to manipulate the numbers.

2. by assigning "000" to represent 0, we have a common representation for 0, regardless of the number of bits (*e.g.* for 3-bits we have 0 being represented by "000", for 7- bits we

would have 0 being represented by "0000000", and so on). This makes our assignment *convenient*; if we had to extend our design by scaling (*i.e.* increasing or decreasing) the numbers of bits, we would not need to worry about re-assigning the 0.

word	unsigned magnitude
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Tab. 4.1: One Possible Scheme for Unsigned Integer Representation

With this in mind, we can now go ahead and try to represent both positive and negative (*i.e.* signed) integers; Table 4.2 shows one possible representation.

word	signed magnitude
000	
001	
010	
011	
100	-4
101	-3
110	-2
111	-1
000	0
001	+1
010	+2
011	+3
100	
101	
110	
111	

Tab. 4.2: One Possible Scheme for Signed Integer Representation

On inspecting this representation we notice some characteristics:

1) the *ordering* of integers is maintained
2) the representation is *symmetrical* about zero, as with integers
3) the zero maintains its representation regardless of the number of bits
4) using a regular adder (*e.g.* the one from Lab 3), we can add complementary pairs (*e.g.* *n* and -*n*) to get zero
5) using only inverters and a regular adder, we can obtain the negative of any represented number

The first three points refer to the *convenience* of the representation; it maintains the properties of the mathematical set of integers. The last two points attest to the *efficiency* of the representation; no new hardware elements are required to manipulate these numbers. For these reasons and more, the representation of Table 4.2 is widely-used and is called the *Two's Complement Representation.*

4.2.2 Signed Integer Arithmetic

We mentioned that a convenient property of this representation is that we can add complementary pairs of numbers to obtain zero. What about any two pairs of numbers in general?

The result of addition or subtraction is supposed to fit within the significant bits used to represent the numbers. If n bits are used to represent signed numbers in 2's complement scheme, then the result must be in the range -2^{n-1} to $2^{n-1}-1$. If the result does not fit in this range, then we say that arithmetic overflow has occurred. To ensure the correct operation of an arithmetic circuit, it is important to be able to detect the occurrence of overflow.

Figure 4-1 presents the six different cases where 2's complement numbers are added. Because we are using four-bit numbers, there are three significant bits, b_{2-0}. When the numbers have opposite signs, there is no overflow (V=0). But if both numbers have the same sign, based on the magnitude of added numbers maybe overflow occurs. The key to determining whether overflow occurs is the carry-out from the MSB position, called C_3 in the figure, and from the sign-bit position, called C_4. It can be proved that:

$$V = C_{n-1} \oplus C_n \qquad (1)$$

where
V is overflow and C_{n-1}, the carry-out from the MSB position, and C_n the carry-out from the sign-bit position (Fig. 4-2).

In (a), (c), (e) and (f), $-8 \le S \le +7$ where S stands for sum, so V=0 and the addition can be shown by 4 bits regardless of C_n and V values. It is apparent that in this case the computation is correct and it doesn't need correction. On the other hand, in (b) and (d) value of V is 1, it means that 4 bits are not enough for showing the computation S and it should be corrected. To correct it, we should use carry-out (C_n) along with S. In this case C_n shows sign bit and the 4-bit number represents magnitude.

Breifly, when V is equal to zero, the n-bit number S shows the correct computation; otherwise, C_n shows the sign bit and S shows the magnitude.

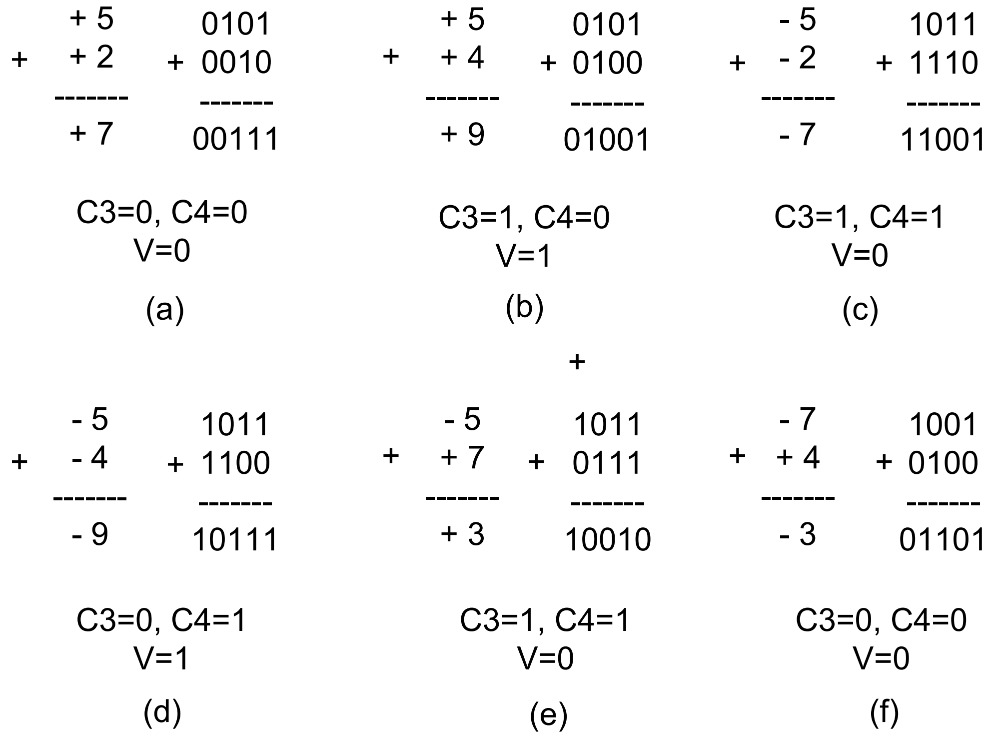


Fig. 4-1: Addition of a pair of two's complement numbers

```
      Cn-1  Cn-2 ...  C0  0
       An    An-1 …   A1  A0
 +     Bn    Bn-1 …   B1  B0
    --------------------------------
   Cn  Sn   Sn-1   ... S1  S0
```

Fig. 4-2: Addition of a pair of two's complement numbers

4.3 Prelab

Complete Designs 1-3, answer the questions, and follow the prelab report format carefully.

4.3.1 Design 1: Negation (Two's Complement)

We mentioned that one can obtain the negative of any number using only inverters and an adder. The function that performs this conversion is

$$-p = \overline{p} + 1 \quad (2)$$

where $\overline{p}$ is called the **One's Complement** of *P* and is obtained by inverting each bit of *P* number and –*P* is two's complement of *P* number. Design and Test a 4-bit block that gives the negative of any 4-bit input applied to it. Create a symbol for your block.

4.3.2 Design 2: Signed Integer Adder

Design and Test a 4-bit Two's Complement Adder (*i.e.* one that adds two 4-bit numbers in Two's Complement form), and create a symbol for your block.

4.3.3 Design 3: Signed Integer Subtracter

Design and Test a 4-bit Two's Complement Subtracter (*i.e.* one that subtracts one 4-bit number in Two's Complement form from another 4-bit number in Two's Complement form), and create a symbol for your block.

4.3.4 Questions

1- Demonstrate that equation (1) is valid, either by deriving it or proving it.

2- Demonstrate that equation (2) is valid, either by deriving it, proving it, or using some sample numbers.

4.4 Lab Procedure

Complete Experiments 1-3.

4.4.1 Experiment 1

Implement and test your Two's Complement Negation design using the MAX+plus II software. When it is working, demonstrate it to your TA.

4.4.2 Experiment 2

Implement and test your 4-bit Two's Complement Adder design using the MAX+plus II software. When it is working, demonstrate it to your TA.

4.4.3 Experiment 3

Implement and test your 4-bit Two's Complement Subtracter design using the MAX+plus II software. When it is working, demonstrate it to your TA.

4.5 Postlab

None.

Lab 5: 8-bit Counter with Debounced and Non-debounced Clock

5.1 Introduction

The purpose of this experiment is to introduce the student to the to the 8-bit counter and to the concept of debounced and non-debounced clock signals. Specifically, you will use an 8-bit counter with both a non-debounced and debounced clock circuit to measure the number of bounces in the clock input signal. In addition, you will learn how to write your results as a hexadecimal number to the 7-segment display.

5.2 Background

5.2.1 The 8count 8-bit counter

A counter is a circuit that uses sequential logic to count clock pulses (either rising clock edges, ↑, or falling edges, ↓). The counter can count up or down, depending on the logic level of the control inputs. You will use an 8-bit binary counter called 8count that is included with the Max+II software. Figure 5.1 is the Max+II module for the counter and Table 5.1 describes its input/output behavior. Note that 8count is a rising edge counter, *i.e.* it will only count when the inputs are set correctly *and* there is a rising edge in the clock input signal.

Inputs						*Outputs*			*function*
CLK	CLRN	SETN	LDN	DNUP	GN	QH	QG..QB	QA	
X	L	X	X	X	X	L	L..L	L	async clear
X	H	L	X	X	X	h	g..b	a	async set
↑	H	H	L	X	X	h	g..b	a	sync load
↑	H	H	H	H	L	previous-1			count down
↑	H	H	H	L	L	previous+1			count up
↑	H	H	H	X	H	previous			hold count

Tab. 5.1: 8count input/output behavior

As mentioned, the 8count counter is able to count up or down. This is controlled by the DNUP input. In this lab, we will be counting the number of pushbutton bounces; for this reason, we need the counter to count up. To do this, DNUP must be set L.

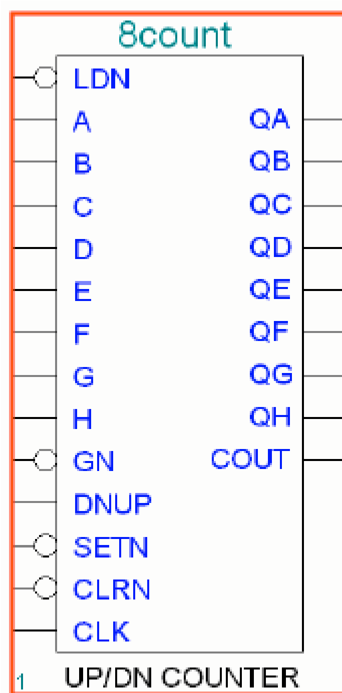


Fig. 5.1: 8Count Max+II module

Note that QH is the highest-order bit (most significant bit, or **MSB**), and QA is the lowest-order bit (least significant bit, or **LSB**). Here is a brief description of the function of the 8count inputs. See Table 5.1 for details.

- **CLK:** clock input; device requires a rising edge for all functions except asynchronous *clear* and asynchronous *set*.
- **CLRN:** asynchronous *clear*; when CLRN is L, this will clear all outputs to logic L (for the *clear* function, the other inputs are *don't-care* inputs; this is denoted on Table 5.1 by the don't-care value, 'X', on inputs CLK, SETN, LDN, DNUP, and GN).
- **SETN:** asynchronous *load*; when CLRN is H and SETN is L, this will set outputs QA-QH to the values on A-H regardless of the logic values on the other inputs (CLK, LDN, DNUP, GN are all X)
- **LDN:** synchronous *load*; when CLRN is H, SETN is H, LDN is L *and* there is a rising clock edge, this will load values on inputs A-H to outputs QA-QH regardless of the logic values on the other inputs (DNUP, GN are all X)
- **DNUP:** determines direction of count; if DNUP is L, 8count counts up; if DNUP is 1, 8count counts down
- **GN:** hold count; if GN is H, the counter outputs will not change, even if there is a rising clock edge; if GN is L, the counter counts normally

In addition to the count output, QH...QA, there is also a carry-out signal, **COUT**, that is set to 1 on counter overflow (*e.g.* when QH...QA = "11111111", and the counter counts up once).

Note: it is necessary to "buffer" (*i.e.* add a redundant gate, such as an AND gate with both inputs tied together) the input signals to the 8count. See the note in Section 5.4.2.

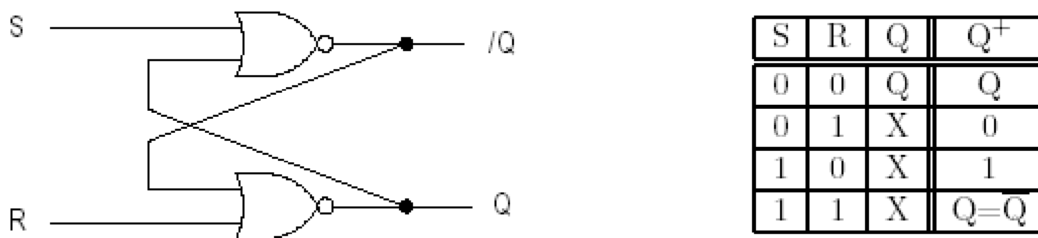
5.2.2 Clock bounce

Each pushbutton (pins 55 and 54) is active-low, which means that it produces a logic low "pulse" when pushed and is logic high otherwise. **For this reason, you will want to invert your clock signal.** When you push the button, it will not instantly give a 0 (or a 1 when inverted); instead, the voltage will fluctuate in a middle region at first. Voltages in the middle region are still interpreted (quantized) as 1's and 0's, so the output will "bounce" between 0 and 1 until the value settles. Hence, a pushbutton is a non-debounced clock signal.

Often you will want to have access to a clock signal that does not bounce. You can create a very simple sequential circuit (see next Section) to accomplish this. This circuit should use both of your board's pushbuttons; one of the buttons should cause your clock signal to go high, and the other should cause it to go low. This is called a "debouncing" circuit; it removes any bounce from the clock signal, so that the signal stays at the correct level.

5.2.3 The SR latch

In order to create a debounced clock signal you will use a very simple sequential circuit called the S-R (set-reset) latch. Sequential circuits are characterized by the fact that their outputs depend not only on the current inputs, but on past inputs. This is different from combinational circuits whose outputs depend only on the current inputs. You will study sequential circuits in detail later in the semester, but for the purposes of this and future labs, you need to understand the function of the S-R latch. The S-R latch has two inputs and two outputs. One output is the complement of the other. The S input "sets" the output (to logic H) while the R "resets" the output to logic L. Note that this is the kind of behavior we require for a debounced clock signal. If both S and R are asserted, the latch tries to force $Q = \overline{Q}$; obviously, this state should be avoided. See Figure 5.2 for a schematic and truth table for the S-R latch. Assume positive logic unless noted. When using this latch as a debounced clock signal, note that the pushbuttons are high when they are not being pushed. To avoid this condition, you will need to use inverters between the pushbuttons and the latch inputs.



S	R	Q	Q^+
0	0	Q	Q
0	1	X	0
1	0	X	1
1	1	X	$Q=\overline{Q}$

Fig. 5.2: SR Latch

5.2.4 The seven segment display

Also in this lab, you will be converting the binary output of the counter to a hexadecimal number in order to show the number on the Altera board's 7-segment display. We will use the digits A - F to represent the numbers 10 - 15 in hex format. The hex digits A - F are represented in uppercase, with the exception of B and D, which are represented in lowercase to avoid confusion with 8 and 0. Your Altera board wiring diagram has the pin numbers for the individual segments of the display.

5.3 Prelab

Complete Designs 1-3, answer the questions, and follow the prelab report format carefully.

5.3.1 Design 1

The output of 8count is an 8-bit binary number (QH...QA). Create functions and schematics to map this output to the 7-segment display in the form of hexadecimal numbers. For example, if the number of bounces was, in binary, 00011011, then the hex display should read 1b (not 1B because of possible confusion between 8 and B). Note that you can create a single module that maps 4 bits to a 1 digit hexadecimal number and use this module once to map the 4 most significant bits to the hex most significant digit and again to map the 4 least significant bits to the hex least significant digit. Hence there are 7 functions to create, not 14 (see Figure 5.3). Remember that these segments are also active-low, so that you will want to use inverters or otherwise alter your design to take this into account. Use Karnaugh maps or other techniques to minimize your expressions. Be sure to include truth tables, functions, schematics, and all your minimization work for each segment in your report.

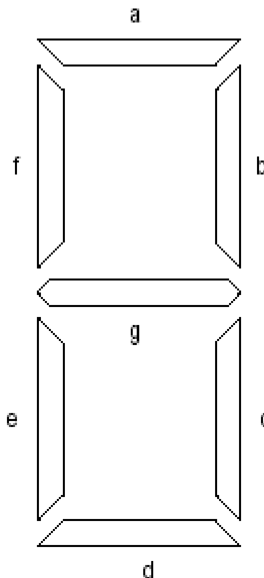


Fig. 5.3: 7-segment display

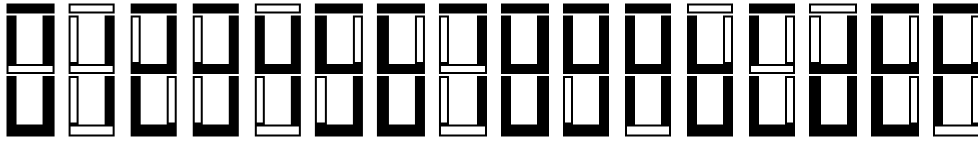


Fig. 5.4: All 7-segment digits

numeral	a	b	c	d	e	f	g
0	0						
1	1						
2	0						
3	0						
4	1						
5	0						
6	0						
7	0						
8	0						
9	0						
A	0						
B	1						
C	0						
D	1						
E	0						
F	0						

Tab. 5.2: 7-segment display "truth table"

5.3.2 Design 2

Use the 7-segment display module you created in Design 1 with the 8count counter to create a circuit that will count the number of bounces when a pushbutton is depressed. Remember to include input pins as needed for the control signals; however, do not include unnecessary input pins; some of the control signals of the 8count can be held at a certain logic level, and so you can connect them directly to ground or VCC. Be sure to connect the outputs of the 8count counter to your display modules in the correct manner (remember which is the most significant bit).

5.3.3 Design 3

Repeat Design 2, but use your debounced clock circuit instead of the non-debounced clock.

5.3.4 Questions

1. What is the highest possible number that can be generated by the 8count counter (in binary, decimal, and hex)?

2. Discuss the similarities and differences between the functions of the SETN and LDN inputs for the 8count device.

3. How would you wire up an 8count 8-bit counter such that it functions as a 7-bit counter that counts backwards.

5.4 Lab Procedure

5.4.1 Experiment 1

Create a simulation file to implement Design 2. First you should create 7 separate modules for the display segments and test each one of them individually. Then create a module containing these 7 small modules. Finally, you will use two of these large modules in your circuit, one for each hex digit. The instructions for creating modules (symbols) are found in Lab 3. The 7-segment displays are hardwired to certain output pins of the Altera board. These pins are enumerated in the Altera board wiring diagram that you saw in lab 3.

Once you have compiled and programmed your circuit, you should reset your counter so that you can start from zero. Then use the pushbutton to clock the counter. The count shown on the display should increase by more than one; record the number of bounces, which is the number the counter increased minus one. You should take at least fifteen to twenty trials to get an accurate result for your board's pushbutton. Then determine the average number of bounces. Once you have determined this average, demonstrate the circuit and show your results to the TA.

5.4.2 Experiment 2

Repeat Experiment 1, but use your debounced clock circuit. The clock bounce you saw in Experiment 1 should disappear, and your counter should increment by exactly 1 each time you cycle your debounced clock. Demonstrate this circuit to your TA.

NOTE: When you are compiling, MAX+plus II may give you an error stating "Project doesn't fit", along with an error saying "Illegal Assignment-global clock "clock" on pin 34", where the name "clock" and the pin number are whatever you have assigned them to be. If this happens, a simple solution is to fool the compiler into thinking that your pin is not a global clock. To do this, you must buffer by signal simply by ANDing it (that is, feed it through an AND2 module) with VCC (or with itself). This operation will always have the same result as the value of the input pin.

5.5 Postlab

None.

Lab 6: 4-bit Shift Adder

6.1 Introduction

In this lab you will extend your work from Lab 3 to create a 4-bit shift adder using your full adder and shift registers.

6.2 Background

6.2.1 The 74179 shift register

A register is a device that stores its input(s) when it sees either a rising or a falling clock edge (hence, its outputs do not change unless the input is changed *and* there is a clock pulse). A shift register is a device that can shift values among the storage elements, one bit at a time, when it sees either a rising or falling clock.

In this lab, you will be working primarily with the 74179 "4-bit shift register with clear". This register has nine inputs and five outputs, which are shown in the Figure 6.1 and Table 6.1. The 74179 shift register can load, shift, and hold values, depending on the value of its control input signals LD and ST. The inputs A - D are used for loading, and the value of SER is shifted into QA, the most significant bit, when the register is shifting. CLRN and CLK are the clear and clock inputs, respectively. Note that CLRN is an asynchronous input. This means that an assertion of CLRN (a logic L since it is active-low) will set all outputs to zero regardless of the state of the CLK signal.

Two or more of these registers can be cascaded to create an 8-bit (or longer) register. This is done by connecting QD (the least significant bit) of the first register to the SER input of the next register.

For more information about the 74179 shift register, go to the **Help** menu, select **74- series macrofunctions,** and choose the 74179 shift register. Be sure you understand the operation of this register as we will be using it in the next several labs.

6.2.2 Binary addition via shifting

In this lab you will use a single full adder with shift registers to create a 4-bit binary adder. 4-bit binary addition is done in the same way as decimal addition as you can see from Figure 6.2. Since you are restricted to using a single full adder, your implementation will require shifting of the binary numbers A and B such that the correct bits of A and B are placed at the inputs of the full adder at the correct time. For example, you want to add $a_0 + b_0$, $a_1 + b_1 + c_0$, $a_2 + b_2 + c_1$, etc. Both A and B should be shifted at the same rate, so that the correct bits are added.

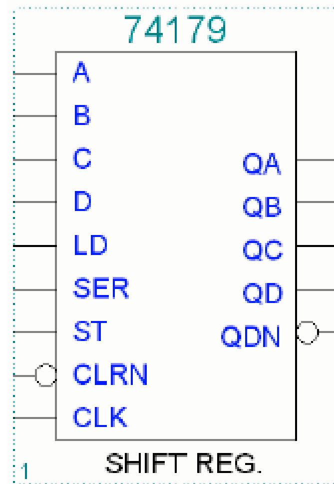


Fig. 6.1: 74179 4-bit shift register

Inputs									*Outputs*				
					Parallel								
CLRN	ST	LD	CLK	SER	A	B	C	D	QA	QB	QC	QD	QDN
L	X	X	X	X	X	X	X	X	L	L	L	L	H
H	X	X	H	X	X	X	X	X	QA_0	QB_0	QC_0	QD_0	$\overline{QD_0}$
H	L	L	↓	H	X	X	X	X	QA_0	QB_0	QC_0	QD_0	$\overline{QD_0}$
H	L	H	↓	X	a	b	c	d	a	b	c	d	$\overline{d}$
H	H	X	↓	H	X	X	X	X	H	QA_0	QB_0	QC_0	$\overline{QC_0}$
H	H	X	↓	L	X	X	X	X	L	QA_0	QB_0	QC_0	$\overline{QC_0}$

Tab. 6.1: 74179 shift register function table

In Figure 6.2, c2, c1, and c0 are the carry bits. You must account for these carry bits in the design of your 4-bit adder by feeding the carry bit from the previous addition back into the carryin of our full adder. However, you will want to put the carryout of the adder through a D flip-flop before feeding it to the carryin. This will provide a one-clock- cycle delay so that the carry bits are added to the correct bits of A and B. The D flip-flop (DFF) is shown in Figure 6.3 and its operation is specified in Table 6.2.

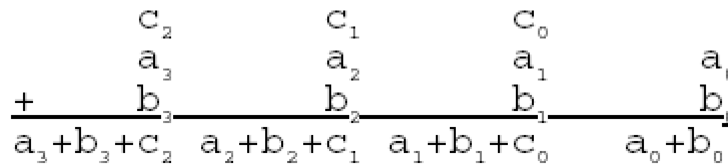


Fig. 6.2: 4-bit addition

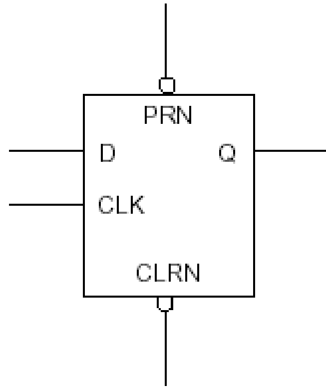


Fig. 6.3: D flip-flop

Inputs				*Outputs*
PRN	CLRN	CLK	D	Q
L	H	X	X	H
H	L	X	X	L
L	L	X	X	illegal
H	H	↑	L	L
H	H	↑	H	H
H	H	L	X	Qprevious
H	H	H	X	Qprevious

Tab. 6.2: DFF D flip-flop function table

6.3 Prelab

Complete Designs 1 and 2, answer the questions, and follow the prelab report format carefully.

6.3.1 Design 1

Design a 4-bit shift adder using 74179 shift registers and one full adder using the information given above and in Labs 3 and 5. Be sure to connect something to LD, ST, and SER so that you can control the shifting of the 74179 register. You should use a debounced clock signal. Draw a complete schematic for your design. Connect the output number to your seven segment display modules. Keep in mind that the output number is only 5 bits, so the most significant digit of the display will be either 1 or 0.

6.3.2 Design 2

The shift adder you created in Design 1 will probably require 5 clock cycles to complete the addition/shifting process. When you demonstrate Design 1 you will simply clock the circuit

manually 5 times and stop. If you clock more than 5 times, you will probably destroy your answer. For Design 2, modify Design 1 such that the circuit will not allow you to clock the circuit more than the number of times that is required to arrive at the answer. You will probably want to use a counter to count the number of times you use the clock. The counter outputs (via some simple combinational logic) can then be used as a control signal to enable/disable the clock. The clock can be disabled by ANDing the output of your debouncing circuit with the proper control signal.

6.3.3 Questions

1. Why do you need a D flip-flop between the carryout and carryin of the full adder?

2. How would you modify the clock before it is input into the flip-flop, and why?

3. Normally, this lab requires three 74179 shift registers: two for the addends and one for the sum. Do you see a way to implement this lab using only two registers? If so, what is it? If not, why not?

6.4 Lab Procedure

6.4.1 Experiment 1

Create a simulation file to implement Design 1. Use your 7-segment display modules to represent the output number. Demonstrate the working circuit to your TA.

6.4.2 Experiment 2

Create a simulation file to implement Design 2 and demonstrate it to your TA.

6.5 Postlab

None.

Lab 7: 4-bit Shift Multiplier

7.1 Introduction

In this lab you will combine designs from several previous labs to build a 4-bit shift multiplier. In addition, you will begin to learn how to use the Altera board's internal clock to run your circuits at any speed you like.

7.2 Background

You may wish to review the background material from previous labs, especially shift registers, full adders, and counters.

7.2.1 The Altera internal clock and the clock divider circuit

The Altera University Program board contains a 25.175-MHz clock, which can be used to automatically clock your circuit's components. This clock is accessed internally by the board and the EPM7128 chip through pin **83**. Once you include the clock pin in your circuit, you can connect it to the CLK inputs for all registers and flip-flops in your circuit.

The inherent problem with using the Altera board's internal clock is that it is too fast for results to be viewed as they occur. To solve this problem, we use a divider circuit to slow the clock pulse down. A divider circuit is basically a counter, whose output increments by 1 every time the CLK input sees either a rising or falling clock edge. Each of the binary outputs of the counter divides the clock signal by a factor of 2. For example, the least significant bit of the clock output changes at half the rate of the input signal, and the next bit of the output changes at one quarter the rate of the input clock signal. Each bit of the clock output is available, so that the clock frequency can be divided by any power of 2 that you wish. More than one counter may be used; wiring the most significant bit of the first counter to the CLK input of the second one (via an inverter) connects the counters. In this lab, you should use the 16cudslr 16-bit counter to construct your divider circuit. The 16cudslr is shown in Figure 7.1. The function of the inputs is very similar to that of the 8count 8-bit counter you have used previously.

7.2.2 4-bit shift multiplication

The circuit you will build in this lab is a 4-bit (shift) multiplier; that is, it takes two 4-bit inputs and multiplies them together. Referring to Figure 7.2, we see that binary multiplication is done in the same way as decimal multiplication:

- the terms a_ib_j , $3 \geq i, j \geq 0$ represent a_i AND-ed with b_j

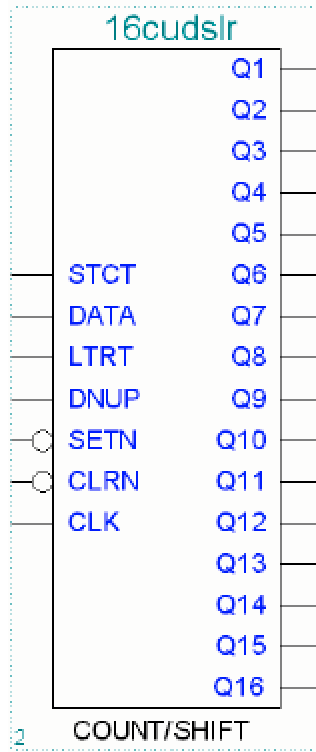


Fig. 7.1: 16cudslr 16-bit counter

- the 0's are placeholders. As in regular decimal multiplication, placeholders are necessary because multiplying by different bits of B means multiplying by a different power of 2

Shifting is required for multiplication. Each bit of B $\{b_3b_2b_1b_0\}$ must be multiplied by all four bits of A $\{a_3a_2a_1a_0\}$, as well as by some zeroes. For example:

- **at step 0**, b_0 is multiplied with the four bits of A to produce the products $\{a_3b_0, a_2b_0, a_1b_0, a_0b_0\}$. This is shown on line 3 of Figure 7.2.

- **at step 1**, b_1 is multiplied with the four bits of A to produce the products $\{a_3b_1, a_2b_1, a_1b_1, a_0b_1\}$. This is shown on line 4 of Figure 7.2.

- ... and so on for all bits of B ...

7.2.3 How to do it

Figure 7.2 shows how to generate a series of "partial products" (*i.e.* lines 3 to 6 of Figure 7.2), but how do we obtain the final result of the multiplication of A and B?

First, we consider the partial products found on line 3 of Figure 7.2. Table 7.1 illustrates how these partial products are added with zero's to produce five Sum bits. The LSB of the sum, $S_{0,0}$

forms the 2^0 bit of the final result.

```
                          a3     a2     a1     a0
                     x    b3     b2     b1     b0
                        a3b0   a2b0   a1b0   a0b0
                 a3b1   a2b1   a1b1   a0b1     0
          a3b2   a2b2   a1b2   a0b2     0      0
   a3b3   a2b3   a1b3   a0b3     0      0      0
```

Fig. 7.2: 4-bit multiplication

	0	0	0	0
$+$	a_3b_0	a_2b_0	a_1b_0	a_0b_0
C_0	$S_{0,3}$	$S_{0,2}$	$S_{0,1}$	$S_{0,0}$

Tab. 7.1: Operands and Result of First Addition. $S_{0,0}$ forms the 2^0 bit out the result

Next, we consider the partial products found on line 4 of Figure 7.2, and the *Sum* generated in the previous step. Table 7.2 illustrates how the these partial products are added with the previous Sum to produce five Sum bits. The LSB of the new sum, $S_{1,0}$ forms the 2^1 bit of the final result.

	C_0	$S_{0,3}$	$S_{0,2}$	$S_{0,1}$
$+$	a_3b_1	a_2b_1	a_1b_1	a_0b_1
C_1	$S_{1,3}$	$S_{1,2}$	$S_{1,1}$	$S_{1,0}$

Tab. 7.2: Operands and Result of Second Addition. $S_{1,0}$ forms the 2^1 bit out the result

Next, we consider the partial products found on line 5 of Figure 7.2, and the Sum generated in the previous step. Table 7.3 illustrates how these partial products are added with the previous Sum to produce five Sum bits. The LSB of the new sum, $S_{2,0}$ forms the 2^2 bit of the final result.

	C_1	$S_{1,3}$	$S_{1,2}$	$S_{1,1}$
$+$	a_3b_2	a_2b_2	a_1b_2	a_0b_2
C_2	$S_{2,3}$	$S_{2,2}$	$S_{2,1}$	$S_{2,0}$

Tab. 7.3: Operands and Result of Third Addition. $S_{2,0}$ forms the 2^2 bit out the result

Next, we consider the partial products found on line 6 of Figure 7.2, and the Sum generated in the previous step. Table 7.4 illustrates how these partial products are added with the previous Sum to produce five Sum bits. The LSB of the new sum, $S_{3,0}$ forms the 2^3 bit of the final result.

The bits $\{C_3, S_{3,3}, S_{3,2}, S_{3,1}\}$ from the previous addition form the 2^7, 2^6, 2^5, 2^4 bits of the final result, respectively. Table 7.5 illustrates how the previous Sum is added with zeros to reproduce the Sum.

	C_2	$S_{2,3}$	$S_{2,2}$	$S_{2,1}$
+	a_3b_3	a_2b_3	a_1b_3	a_0b_3
C_3	$S_{3,3}$	$S_{3,2}$	$S_{3,1}$	$S_{3,0}$

Tab. 7.4: Operands and Result of Fourth Addition. $S_{3,0}$ forms the 2^3 bit out the result

	C_3	$S_{3,3}$	$S_{3,2}$	$S_{3,1}$
+	0	0	0	0
	C_3	$S_{3,3}$	$S_{3,2}$	$S_{3,1}$

Tab. 7.5: Operands and Result of Fifth Addition which produces the upper four bits of the final result.

Hence, the final result of multiplying $a_3a_2a_1a_0 \times b_3b_2b_1b_0$ is $\{C_3\ S_{3,3}\ S_{3,2}\ S_{3,1}\ S_{3,0}\ S_{2,0}\ S_{1,0}\ S_{0,0}\}$.

7.3 Prelab

Complete Designs 1 and 2, answer the questions, and follow the prelab report format carefully.

7.3.1 Design 1

Design the clock divider circuit discussed in the background section of this lab. Your final clock output should have a frequency that is as close as possible to 1 Hz.

7.3.2 Design 2

Design your shift multiplier using your 4-bit ripple carry adder, several 74179 shift registers, your clock divider, and your 7-segment display modules. Your result should appear as a hexadecimal number. You should note that this lab is very open-ended; that is, a lot of the decisions in programming this multiplier are left to you.

NOTE: Remember to incorporate clock stopping logic (*i.e.* the counter-based circuit from Lab 6 that was used to suspend the clock after a certain number of clock cycles).

7.3.3 Questions

1. Since each output from the 16cudslr counter has a frequency that is half that of the previous output, how many of these counters should you use if you want your final clock output to pulse as close to once a second as possible? Which output (1-16) of your final counter do you want to use as your final clock output?

2. What is the effect of placing an inverter between Q16 of the first counter and the CLK input of the second counter? Under what circumstances do you need the inverter? Is it important for this lab?

3. How many bits are necessary for the output of your 4-bit multiplier? What is the largest product you can obtain (in hex and decimal)?

7.4 Lab Procedure

Complete Experiments 1 and 2.

7.4.1 Experiment 1

Create a simulation file for your clock divider circuit and demonstrate it to your TA. Create a module for this circuit as you will be using it in next experiment. Be sure and make the 8 most significant output bits of the last counter available as you may need different clock frequencies in your multiplier design.

7.4.2 Experiment 2

Create a simulation file for your shift multiplier. Test it by inputting several sets of multiplicands and multipliers, and computing the result. Demonstrate your working circuit to the TA.

NOTE: To easily identify and debug problems, first test your circuit with a manual clock generator (*i.e.* two push buttons and an SR latch).

7.5 Postlab

None.

Lab 8: The Priority Encoder

8.1 Introduction

The purpose of this experiment is to provide the student with experience in sequential circuit design. Specifically, you will build a priority encoder that will examine 8 interrupt bits and determine the order of the highest active interrupt.

8.2 Background

No further background material is necessary for this lab; though, you may wish to review the function of the 74179 shift register and the 8count counter shown in Fig. 8.1. You will use both of these in your design.

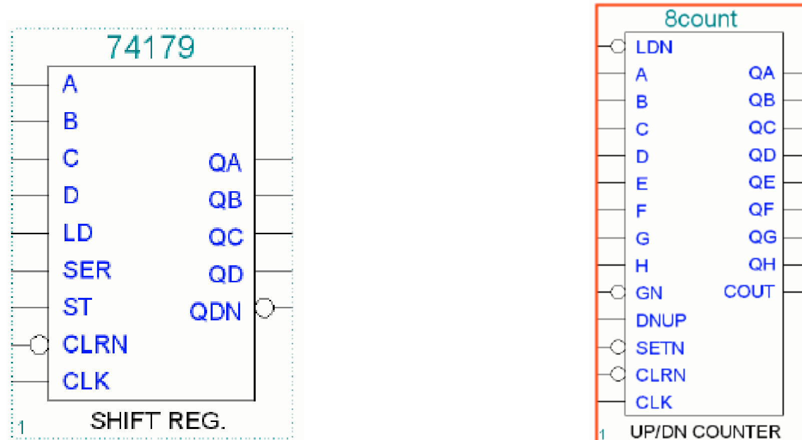


Fig. 8.1: 74179 4-bit shift register and 8count Max+II modules

8.3 Prelab

Complete Designs 1 and 2 and follow the prelab report format carefully.

8.3.1 Design 1

You are to design what is called an 8-bit priority encoder. A priority encoder is used to determine the relative importance of interrupt requests in a computer, so that the computer will know which request should be given top priority. The circuit will have one 8-bit input (switches) and will use

one digit of the 7-segment display for the output. The function of the circuit will be to display the order of the highest order input bit that is a logic 1. If, for example, your input is 00101011, the display should read 5 because 5 is the order of the highest-order bit equal to 1 (00101011). In our notation, the least significant bit has order zero and the most significant bit has order 7. If your input byte does not contain a 1, then the output digit should display a dash, *i.e.* only the center segment should be turned on. Once the output has been determined, the clock should be disabled so that you cannot destroy your answer with additional clock pulses.

You will use two cascaded 74179 shift registers for your input byte. The inputs of these registers should be connected to switches (8 of them). You will also likely use a counter in your design. You will also need to use a debounced clock circuit to connect to the clock inputs of the shift registers and the counter.

Most designs will have each of the components listed above, but the specifics of the design are up to you. One way to complete the design is to first load the input byte in "backwards" so that the first bit shifted out of the register will be the most significant bit (MSB) of the input byte. Simultaneously load the counter with 7 and set it to count down. After the load operations, (and before the first shift) you should check to see if the value of the bit to be shifted out (the MSB of the input byte) is high. If so, the answer is 7 and you are done. If not, shift once, decrement the counter, and check bit 6. Proceed in this fashion until you find the first bit that is high. Since we are shifting "backwards", the first bit that is a 1 is the highest order bit of the input byte that is 1. Draw a comprehensive schematic integrating all of the above pieces of your design.

8.3.2 Design 2

During the testing phase in Max+II, your clock signal should be the debounced clock you implemented in a previous lab. When your circuit is working correctly, replace this clock signal with your automatic signal from your clock divider circuit. Your final circuit should function correctly with a total of **9** switches, eight for the input bytes and one for clear/reset. This means you will have to devise a way for the circuit to automatically switch the shift registers (counters) from load operation to shift (count) operation after the first clock cycle. Draw a comprehensive schematic integrating all of the above pieces of your design.

8.4 Lab Procedure

Complete Experiments 1 and 2.

8.4.1 Experiment 1

Create a simulation file to implement your prelab design with the debounced clock for testing purposes.

8.4.2 Experiment 2

Add the automatic clock to your last design and demonstrate the working circuit to your TA.

8.5 Postlab

None.

Lab 9: T-Bird Taillights

9.1 Introduction

In this lab you will learn about **state machines**. You will use state machines to design a circuit that will cause two sets of 3 LED's to light up and blink in a manner that mimics the taillight behavior of an old Ford Thunderbird automobile.

The lab is a **one-week** lab; however, an optional extra credit portion can be turned in during the (optional) second week.

9.2 Background

9.2.1 State Machines

A state machine is a type of sequential circuit structure which can allow you to create more elaborate types of digital systems.

A state machine consists of:

- memory elements (*e.g.* D flip-flops) to store the *current state*
- combinational logic to compute the *next state*
- combinational logic to compute the *output*

There are two classes of state machines: *Mealy and Moore.*

- a *Mealy* machine uses both the *current state* **and** the *current inputs* to form the *output*
- a *Moore* machine only uses the *current state* to form the *output*

Fig. 9.1 illustrates this difference. The figure shows the basic elements of a state machine:

- the inputs (*e.g.* switches)
- the memory elements ("D flip-flops") that store the *current state*
- the combinational logic ("Next State Logic") that computes the *next state*
- the combinational logic ("Output Logic") that computes the *output*

With a Mealy machine, a connection exists between the "Inputs" and both the "Next State Logic" and the "Output Logic". With a Moore machine, a connection only exists between the "Inputs" and the "Next State Logic". (The dashed line from the "Inputs" to the "Output Logic" represents

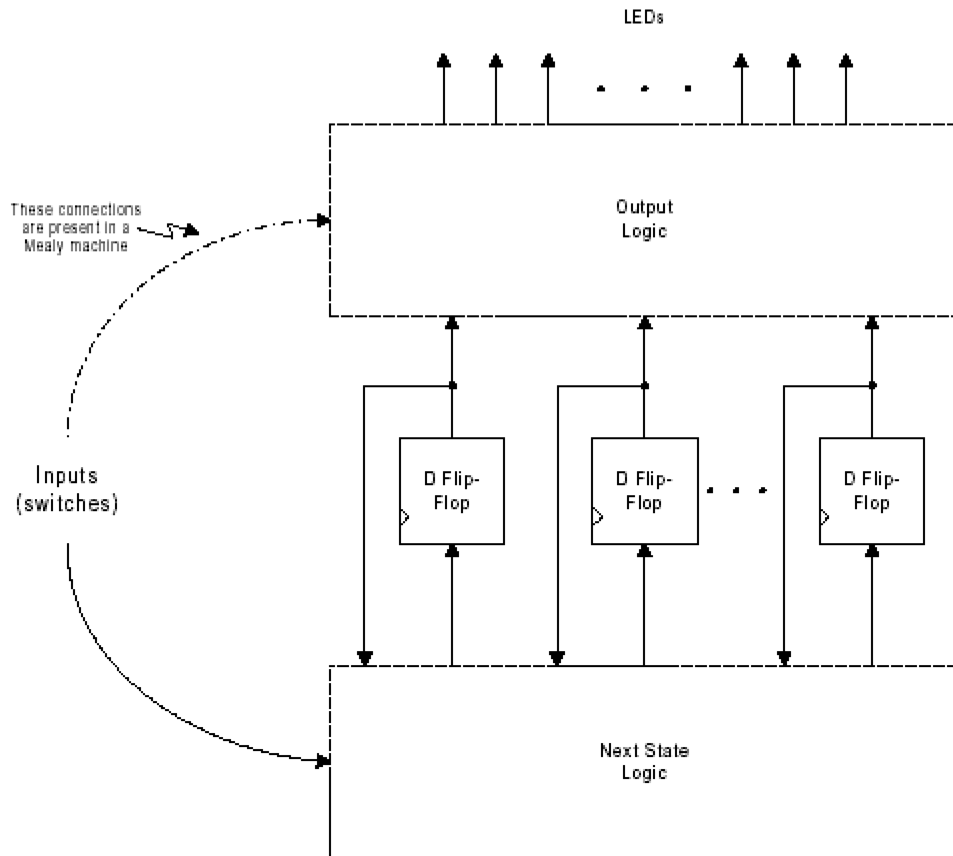


Fig. 9.1: State machine as implemented in Max+II

the sole difference between Mealy and Moore machines.)

In order to design a state machine, you must:

- determine the “states” of the machine (if there are *n* states, then you will need $log_2(n)$ memory elements)
- design the “Next State Logic” block
- design the “Output Logic” block

To design this, you must use *state diagrams*, *state tables*, and the combinational logic design techniques that you’ve already used in the previous labs. **For more information about state machines, see your text.**

9.2.2 T-bird Taillights

The T-bird taillights were distinctive in the way they lit up. There were three lights on each side of the car. When one of the turn signals came on, the innermost light would come on, then the next light, and finally the outermost light. Then all the lights would go off, and the sequence would repeat. Figure 9.2 shows the sequence of lights for a left turn.

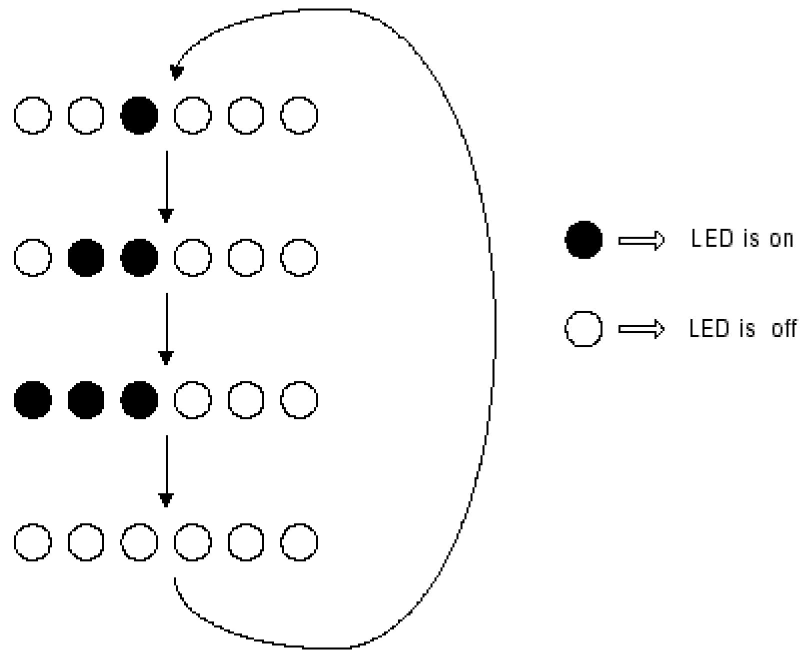


Fig. 9.2: LED sequence for a left turn

A circuit to control T-bird taillights must support:

- left turns
- right turns
- hazard lights
- braking (*i.e.* the application of the brake)

What states are needed? You will need:

- a state where all six lights are off
- a state where all six lights are on
- three states to represent the LED sequence for a left turn
- three states to represent the LED sequence for a right turn

Note: the “hazard” condition does not need any additional states. It can be implemented by bouncing back and forth between the “all on” and “all off” states.

What inputs are needed? You will need three inputs:

- "LT" for left turns
- "RT" for right turns
- "BRK" for the brake

Note: when "LT" **and** "RT" are **both** on, the "hazard" condition should be activated.

9.3 Prelab

The prelab for Designs 1, 2 and 3 must be turned in at the beginning of the lab session. Design 4 is for extra-credit.

9.3.1 Design 1

Enumerate the states for a state machine, which will run the basic functions of the T-bird taillights (left and right-turn signals, brake lights, and hazard lights). Assuming you will build a *Moore machine*, draw a state diagram showing all of the inputs and outputs you are using. For this design, you do not need to worry about the brightness of the lights; you may simply turn the lights on and off.

9.3.2 Design 2

Now assume you will build a *Mealy machine*. Draw the state diagram for your machine, correctly indicating all the inputs and outputs you are using. Again, do not worry about dimming the lights yet.

9.3.3 Design 3

Choose a machine type (Moore or Mealy) and complete the design process. You should end up with a gate level schematic. You will want to use your clock divider from the previous labs as your clock signal.

9.3.4 Design 4

Once you have developed the basic circuit, you will want to add extra features. The only *required* feature is that you dim the brake lights so that you can see a (brighter) turn signal on top of the brake lights. You may add other features for extra credit. Examples include:

- Headlights: these can be implemented in various ways and often they have some effect on the brightness of the LEDs.
- Use the 7 segment display modules as a speedometer that increases whenever the brake is not applied (or when some other criteria is satisfied).
- Use the 7 segment display modules as a gas gauge that decreases whenever the brake is not applied.
- Use the LEDs to simulate the arrows that appear in the side mirrors of some new cars

when the turn signal is activated.

9.3.5 Questions

1. Suppose a state machine has k states. Find an exact expression for the number of D flip-flops (memory elements) required to hold the state.

2. Which of your first two designs (Mealy and Moore) do you think is more realistic for an actual car? Why?

9.4 Lab Procedure

Create a simulation file, using the design you chose in Question 1, and demonstrate the basic functions of the T-Bird taillights to your TA. Once you have finished that portion of the lab, implement your own improvements and demonstrate them to the TA.

9.5 Postlab

None.

Lab 10: LED Pong

10.1 Introduction

This is a two-week lab in which you will combine all the information you have learned this semester to develop a game of LED Pong. Once the basic game has been designed and tested you will add more innovations to your game as time allows.

10.2 Background

There is no new material for this lab.

10.3 Prelab

This lab is intended to be more open-ended than the previous labs have been. There is more than one correct way to implement this lab. You may turn in the prelab in two parts. During the first week you should turn in everything except for Designs 2-4. During the second week you may turn in an addendum containing Designs 2,3,4. Since each design will be unique, it is especially important for this lab that you include a detailed written description of how your design is to work. **Read the game instructions carefully before you start. Circuits that do not work within the rules specified will not receive full credit.**

10.3.1 Design 1

Design a circuit that will play LED Pong in the following fashion. Two players are assigned pushbuttons. Two columns of LED's on the Altera board will be the game board. A lit LED represents the "ball"; as the "ball" is traveling, one LED will deactivate and another will activate. Only one light is lit at any given time. One of the players hits his/her pushbutton to "serve" the ball and begin the game. The ball travels up one column and down the other as in Figure 10.1.

The second player must hit his/her pushbutton to return it but can only do so when the ball is in the lowest position of the column. If the player hits the pushbutton at the correct time, the ball returns up the column and back toward the first player; however, if the returning player hits the button either after *or* anytime before the last LED is lit, the game resets and the *winning* player serves. Note that if a server tries to re-serve before the ball reaches the other player, it should have no effect on the circuit.

To do this, you may use either a shift register, a counter, a state machine, or some combination of these. You will also need your clock divider module to set the game speed.

If you decide to use a state machine, keep in mind that a state machine requires that inputs, once

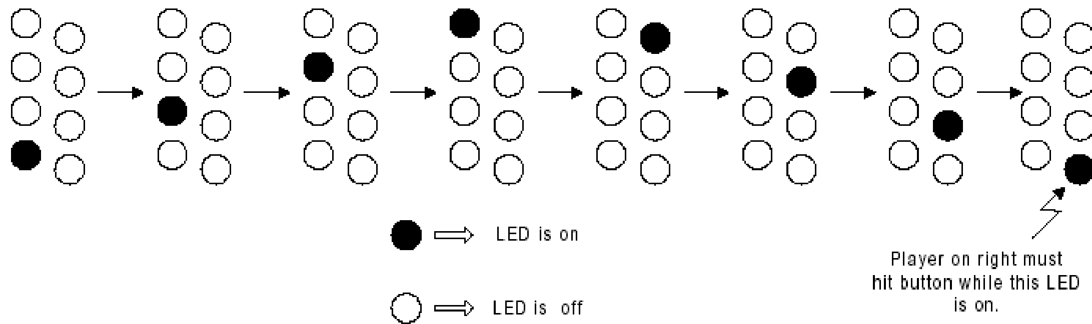


Fig. 10.1: LED sequence when the left player has served

they are asserted, must remain asserted until the next rising clock edge (until the next state transition). Since pushbuttons do not work in this fashion, you will have to use a self-resetting latch to hold the pushbutton value until the next rising clock edge. You can use an SR latch and a D-flip-flop in the following manner: attach the Q latch output to the flip-flop input and attach the flip-flop output to the Reset latch input. The clock signal for this flip-flop should be the same one used for the game itself.

10.3.2 Design 2

Once the original LED Pong game works, add two 4-bit counters and your hexadecimal display modules from Lab 4, so that a one-digit score may be displayed for each player. Be sure when you are adding this portion that the clock signals of these counters have no bounce, so that the scores only increase by 1 at a time.

10.3.3 Design 3

Next, you can make the game more difficult by adding acceleration to the "ball". This acceleration will depend on the length of time a player waits to hit the ball while it is in the last position. If the ball is hit when it first arrives, the return will be slowest; if it is hit just before it goes out of play, the return will be fastest. Your design will use lower-order clock bits to determine at what fraction of the time period the ball was hit. However, there will be several different speeds at which the ball may be traveling; therefore, you must use a decoder (demultiplexer) to determine which clock bits you use to determine the speed of the return.

10.3.4 Design 4

You are welcome to design your own innovations for LED Pong. Each extra innovation will result in a higher lab score.

10.4 Lab Procedure

Create a simulation file for your circuit in Design 1 and demonstrate it to the TA. Once the TA

has approved your design, add as many of the improvements listed in Designs 2 through 4 as possible before the end of the lab period.

10.5 Postlab

None.