CMPE-160
Digital System Design I
Laboratory Manual

Department of Computer Engineering
Rochester Institute of Technology

Spring 2021

Contents

Weekly Laboratory Schedule

Monday and Wednesday Lab

Week	Lab	Demonstration
1	Ex. 1	
2	Ex. 2	Ex. 1 grading sheet
3	Ex. 3	Ex. 2 grading sheet
4	Ex. 4	Ex. 3 report
5	Ex. 5	Ex. 4 report
6	Ex. 6	Ex. 5 report
7	Ex. 7	Ex. 6 report
8	Ex. 8	Ex. 7 report
9	Wed Recharge	
10	Ex. 9	Ex. 8 report
11	Ex. 10	Ex. 9 report
12	Ex. 11	Ex. 10 grading sheet
13	Ex. 12 Part 1	Ex. 11 report
14	Ex. 12 Part 2	Ex. 12 grading sheet

Tuesday Lab

Week	Lab	Demonstration
1	Ex. 1	
2	Ex. 2	Ex. 1 grading sheet
3	Ex. 3	Ex. 2 grading sheet
4	Ex. 4	Ex. 3 report
5	Tues Recharge	
6	Ex. 5	Ex. 4 report
7	Ex. 6	Ex. 5 report
8	Ex. 7	Ex. 6 report
9	Ex. 8	Ex. 7 report
10	Ex. 9	Ex. 8 report
11	Ex. 10	Ex. 9 report
12	Ex. 11	Ex. 10 grading sheet
13	Ex. 12 Part 1	Ex. 11 report
14	Ex. 12 Part 2	Ex. 12 grading sheet

Thursday Lab

Week	Lab	Demonstration
1	Ex. 1	
2	Ex. 2	Ex. 1 grading sheet
3	Ex. 3	Ex. 2 grading sheet
4	Ex. 4	Ex. 3 report
5	Ex. 5	Ex. 4 report
6	Ex. 6	Ex. 5 report
7	Ex. 7	Ex. 6 report
8	Ex. 8	Ex. 7 report
9	Ex. 9	Ex. 8 report
10	Ex. 10	Ex. 9 report
11	Ex. 11	Ex. 10 grading sheet
12	Ex. 12 Part 1	Ex. 11 report
13	Thurs Recharge	
14	Ex. 12 Part 2	Ex. 12 grading sheet

Laboratory Kit Parts List

3	74LS00	Quad 2-Input NAND
2	74LS02	Quad 2-Input NOR
4	74LS04	Inverter
2	74LS08	Quad 2-Input AND
2	74LS10	Triple 3-Input NAND
2	74HCT14	Schmitt Trigger Inverter
1	74LS20	Dual 4-Input NAND
2	74LS27	Triple 3-Input NOR
2	74LS32	Quad 2-Input OR
2	74LS86	Quad 2-Input Exclusive-OR
1	74LS283/74LS83	4-Bit Binary Adder with Fast Carry
1	76SB10	Dip Switch (10 switches)
1	4114R-2-472	Dip Resistors (4.7k x 10)
1	4114R-2-102	Dip Resistors (1k x 10)
8	LED (Light Emitting Diodes)–Red	
8	100 kΩ Resistors	
1	Breadboard	
1	IC Extractor	
1	Plastic Parts Box	
1	Jumper Wire Kit	
1	Plier Kit	

Laboratory Guidelines

These first few pages of the lab manual give guidelines for the lab procedure and the lab report. These guidelines apply to all CMPE-160 laboratory work, unless otherwise specified by the lab instructor.

Lab Procedure

- Labs start on the first week of classes, (see the schedule on page iii.) Lab work must be conducted, completed and demoed in the registered lab section to receive full credit. If work is not completed by the end of the registered lab section, it may be completed and demoed during scheduled mentoring hours with a penalty of 10 points. Exercises One and Two are exceptions and may be demoed during scheduled mentoring hours for full credit.
- If not explicitly stated otherwise, each lab exercise consists of three parts: the prelab work, the lab work, and the lab report. The grading sheet included at the end of each lab exercise gives a detailed grade distribution for these components. To receive any grading credit students must earn points for both the demonstration and the report. Furthermore, the earned demonstration points must be at least half of the available demonstration points for grading credit.
  - **Prelab work**. **Completed prelab work is due at the beginning of the registered lab session.** Any questions about it must be discussed ahead of time, either during TA mentoring hours, or during course instructor office hours.
  - **Lab work**. Lab work follows the procedure given in the lab manual and requires obtaining the instructor's or the TA's signature for each step, as indicated in the procedure.
  - **Lab report**. Technical communication is an essential part of engineering. The lab report guidelines that follow apply to all reports. Any questions regarding these guidelines should be discussed with TAs and/or instructors, especially early in the semester.

Lab Report

- **Reports are due one week after the lab is performed and must be submitted at the beginning of the registered lab session.** Both a paper copy of the report and its electronic version must be submitted. Paper copies will be collected in lab. Electronic submissions must be made in PDF format to the appropriate dropbox on MyCourses, prior to the registered lab session. For lab exercises 11 and 12, all VHDL source code must be submitted to the dropbox as well. Students should maintain an electronic copy of each report submitted.
- Plagiarism in any form is not tolerated. This includes but is not limited to: copying from the lab manual, copying someone else's work, copying your own previous work, or copying from the Internet.
- Late reports must be submitted to your TA's folder. For any late report, the student is responsible for submission and receipt: (1) informing the TA by email that the report has been placed in his or her folder, (2) confirming report receipt by the TA. There is a significant penalty for **late submission: 20 points per day.** Thus, lab work must be completed within one week to receive grading credit.
- All reports must be typed in 11-point serif font with full justification, (e.g., "justified" alignment in MS-Word) and 1" margins.
- Any information required for the report must be self-contained and computer generated; or else the report will be treated as not submitted. All figures should be integrated into the report, and they must be properly labeled, introduced, and explained in the text.

- Tense: Choose a tense appropriate to the content of the report, and avoid unnecessary tense shifts. For example, write about specific procedures or results from your work in past tense (since they are already completed).
- Perspective: Write the lab report only in the third person–not using "I," "me," "my," "we," "us," "our," "you," "your," etc. Also avoid a writing style which talks about any person–"one," "the student," or "the instructor." For example, avoid, "One derived the truth table for a full adder," or "The student designed a parking gate control system." Such problems are easily avoided if perspective is focused solely on the technical content of what was done in lab rather than on the classroom experience.
- In the manual, at the end of each lab exercise, there is a grading sheet for collecting signatures from demonstrations in lab. This sheet must be attached at the end of the lab report. The grading sheet serves both as a grading rubric as well as a checklist to make sure all of the necessary information is included in the lab report.
- The report is composed of the following sections (in order):
  - **Cover Sheet.** The provided cover sheet template must be used.
  - **Abstract.** The abstract presents the reader with a brief but complete summary of the intent of the lab exercise. From the abstract, the reader should know what the rest of the report entails–not only what was done, how it was done, and what were the results, but also "new" concepts that were explored in the exercise (e.g., K-maps, 2's complement, active high/low, etc.). Here are two common errors in writing an abstract:
    - ⊘ Simply copying the objective from the lab exercise–the abstract should begin by establishing the scope of the exercise and expressing its purpose in the writer's own words.
    - ⊘ Not stating the results–the abstract must be a concise but comprehensive synopsis.
  - **Design Methodology.** The design methodology presents much of the theory behind the lab exercise, which was confirmed during the activity with software simulation and/or hardware experimentation. Make sure that any of the figures included not only are labeled but also are both referenced and discussed in text, (e.g., "The final design of the two-bit adder is shown in Figure 5."). Within the report, integrate each figure next to the text that discusses it. Also, this section includes any circuit diagrams, K-maps, truth tables, etc. that were used in designing the integrated circuit. Common pitfalls include stating that "the design methodology is based on the lab manual." The lab report should stand on its own objective technical merit without any reference to class or lab.
  - **Results and Analysis.** The results and analysis section provides the reader with all results and insights obtained from completing the exercise: any graphs, tables, etc. that were collected from the lab work. Discuss how the data were obtained (e.g., using a multimeter to collect the voltage levels), the standard for comparison (e.g., truth tables), and both expected and unexpected results obtained from the lab activities. Once again, every figure, table, and schematic diagram must be properly labeled, introduced, and discussed in text. Here are some common pitfalls:
    - ⊘ Putting conclusions here.
    - ⊘ Not comparing results obtained with the corresponding aspect(s) in Design Methodology.
  - **Conclusions.** The conclusion section states whether the exercise as a whole was properly executed and completed. It discusses any facilitators and hurdles encountered in performing the exercise. In addition, it describes how theories and techniques (learned in class) were applied in the lab exercise, the lessons learned from the lab activities, and what could have been done, if anything, to make the exercise go smoother.

  - **Questions.** For each question in the manual at the end of the lab exercise, the questions section copies the question and gives an answer. Some of the questions can be challenging and require leveraging concepts learned from the lecture, the lab activities, and sometimes additional reference material. Students are encouraged to attempt answering the questions before the lab session ends, and may seek for help from the lab instructor and the TA. Attempting to solve a question is better than leaving it blank—even if the answer is wrong.
- Writing composition points will be awarded if the lab report is written with enough concise, self-contained information that someone with similar background could read only the lab report and be able to perform the same exercise to obtain similar results. For example, your lab report should contain enough detail so that you could reread it three years from now to know exactly how to repeat the exercise and get very similar results.
- Further details and tips on how to write a report can be found in the sample report that been posted on MyCourses. Failure to follow guidelines in the sample report or in this section will result in point deductions.

Exercise 1 Altera Quartus II and ModelSim Tutorial

Objective

The objective of this exercise is to introduce essential software tools used in the CMPE-160 Digital System Design I course. After completing this exercise, you will be able to create schematic of a simple digital circuit in Quartus II and perform simulation in ModelSim.

Lab Procedure

Part 1

In this tutorial, the Quartus II and Modelsim software are used to implement the following circuit:

$$\text{Y} = (\text{A NOR B}) \text{ XOR } (\text{NOT C})$$

where A, B, and C are the inputs and Y is the output. The tutorial consists of two parts: (I) designing the circuit in Altera Quartus II and (II) simulating the circuit in ModelSim.

I. Circuit Design Tutorial

A. Getting Started

1. Create a directory **on a flash drive** (If you do not have a flash drive, you may create this directory in My Documents, but make a copy once you have a flash drive available). Suggestion for the directory name–use the name and number of the exercise along with another identifier for the particular circuit since many exercises have more than one circuit. For instance, the first circuit in Exercise Two could be named Ex2_c1. *Warning*: Only use alphanumerics (A to Z and 0 to 9) or underscores for file and directory names. No spaces or special characters (.&-!#) should be used. (it works for *My Documents* as *My Documents* is only a link to a physical directory without spaces `C:\Users\username\Documents`). Avoid virtual directories such as the "Desktop" folder.
2. Launch Altera's Quartus tool: Start → Search for "Quartus II" → Quartus II 13.0sp1 64-bit. The version number does not need to match exactly.
3. On the startup dialog box, choose Create a New Project (New Project Wizard). (If the startup dialog box has been disabled, the same result can be achieved by going to File → New Project Wizard.) The New Project Wizard: Introduction window will open.
   a. Click the Next > button to go to the next new project wizard window.
   b. In the first text box, ("What is the working directory for this project?"), browse to find the full path to the directory created in step I.A.1.
   c. In the second text box, ("What is the name of this project?"), enter the name of the directory created in step I.A.1. The third text box will default to the same name as typed here.
   d. Click the Finish button to complete the new project wizard.
4. From the File menu, select New, which will launch the New dialog box. Under Design Files select Block Diagram/Schematic File, and then press OK. A new block design file (bdf) will open in the Block Editor.
5. From the File menu, select Save as, provide file name, and save the block design file in the project directory created in step I.A.1. (Because of the entries in the previous New

Project Wizard step, this directory will be the default in the Save As dialog box, and the default file name will be the same as the project name.)

B. Component Selection Process

1. On the toolbar at the top of the Block Editor, click the Symbol Tool icon, which looks like the symbol for an AND gate. A Symbol window listing the available Altera library components should appear. Under `c:/altera/13.0sp1/quartus/libraries`, there are megafunctions, primitives, and others. For this tutorial, the primitives library is used.
2. Select nor2 under primitives → logic, and a 2-input NOR gate will appear on the palette to the right. Press the OK button, and the Symbol window will disappear, returning to the Block Editor with a 2-input NOR gate at the mouse pointer. Click where the 2-input NOR gate should be placed. Press the escape key (Esc) to stop the Symbol Tool from entering more NOR gates.
3. Repeat steps 1-2 to place an inverter. Note: The inverter is designated as not in the Symbol Files field.
4. Finally, place a 2-input XOR gate (xor) on the palette.

C. Adding/Deleting Wires and Moving Components

1. Place the mouse pointer near the output of the inverter so that the pointer changes to a + symbol with a right angle wire symbol to the lower right.
2. Click the mouse and drag the pointer to the input of the XOR gate. This action creates a wire, as shown in Figure 1.1.

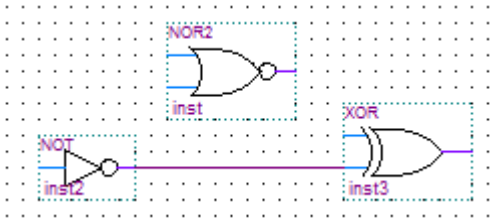


Figure 1.1: Block Editor after step I.C.2.

3. To delete a wire, simply click on it, (it should change color to indicate selection), and press the Delete key.
4. Try moving a component by selecting it with the mouse, holding down the left button, and moving it to another location in the Block Editor. (Note: if wires are connected to the component being moved, they drag and stay connected to the component.)

D. Adding Input and Output Ports

1. In the same manner that gates were placed, add an input port using the Symbol Tool by selecting primitives → pin → input.
2. Click on the input pin name and change it to A. (Note that VCC shown below the pin name is the default value of the input signal. Typically the value of an input will be specified explicitly during simulation.)
3. Repeat steps 1-2 to create input ports B and C.
4. Add an output port using the Symbol Tool by selecting primitives → pin → output. Change the pin name to Y on the output port.
5. Connect the gates and ports as shown in Figure 1.2.
6. Save the design. The design is now complete and can be used for simulation.

II. Circuit simulation

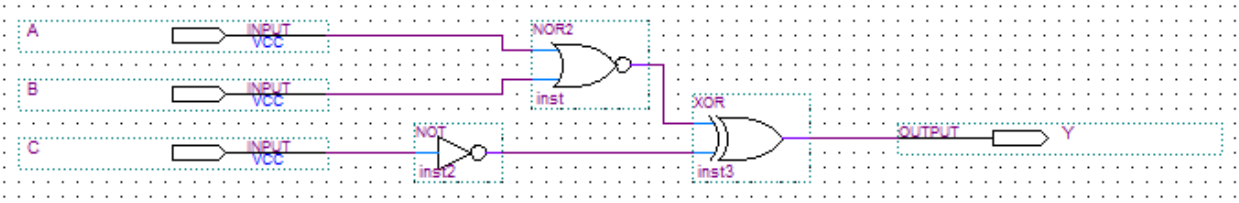


Figure 1.2: Block Editor after step 5.

A. Setting Project for Functional Simulation with ModelSim
Before simulating the design, a VHDL description of the Quartus design must be created.
1. From the File menu, select Create/Update → Create HDL File From Current File... which will result in a Create HDL Design File for Current File dialog box.
2. Select VHDL for the File type.
3. Press the OK button and wait for the Quartus II message window that shows "Create VHDL File was successful."
4. Press the OK button to close the Quartus II message window.

B. Creating a ModelSim Project
The VHDL file generated by Quartus in the previous step is used in ModelSim. It must be associated with a ModelSim project to simulate the design.
1. Launch ModelSim: Start → Search for "Modelsim" → Modelsim SE-64 10.1d.
2. From the File menu, select New → Project, which will launch the Create Project dialog box.
   a. In the Project Name text box, enter the same project name used in Quartus.
   b. For the Project Location, browse to the Quartus project directory created in step I.A.1.
   c. Click the OK button to create the project, which will launch the Add Items to the Project window.
   d. Click the Add existing file icon to launch the Add file to Project window.
   e. Click the Browse button to launch the Select files to add to project window.
   f. Select the .vhd file with the same name as the project and press Open to close the Select files to add to project window. This file will be located in the project directory. Windows may not show the .vhd extention. The .vhd file may look like a text document.
   g. Click the OK button to close the Add file to Project window.
   h. Click the Close button to close the Add Items to the Project window. One VHDL file should be listed in the Project pane and it should look similar to Figure 1.3.

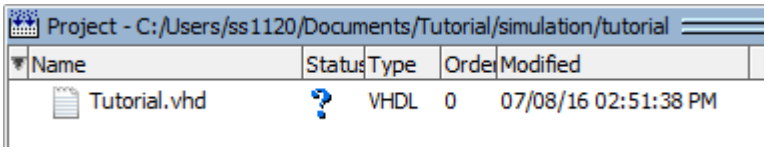


Figure 1.3: VHDL file loaded after step II.B.2.h.

3. From the Compile menu, select Compile All to compile the design. When the project has been successfully compiled, the "?" under Status in the Project pane will change to "✓" as shown in Figure 1.4.

C. Creating Waveforms for Simulation
Once the ModelSim project has been compiled, waveforms need to be defined to provide

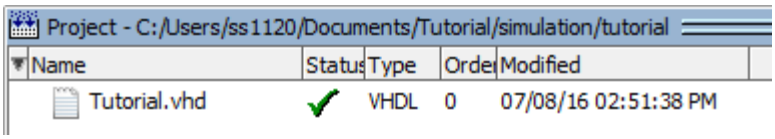


Figure 1.4: VHDL file compiled after step II.B.3.

stimulus to the design for simulation.

1. From the Simulate menu, select Start Simulation, which launches the Start Simulation window.
2. Under work (in the default Design tab), select the entity with the name of the project.
3. Click the OK button. The Start Simulation window will close, and simulation mode will start with a sim pane on the left, an Objects pane in the center, a Wave pane on the right, and a Transcript pane on the bottom as shown in Figure 1.5. If the ModelSim screen does not automatically launch those panes shown in Figure 1.5, use the View menu to open any missing windows.

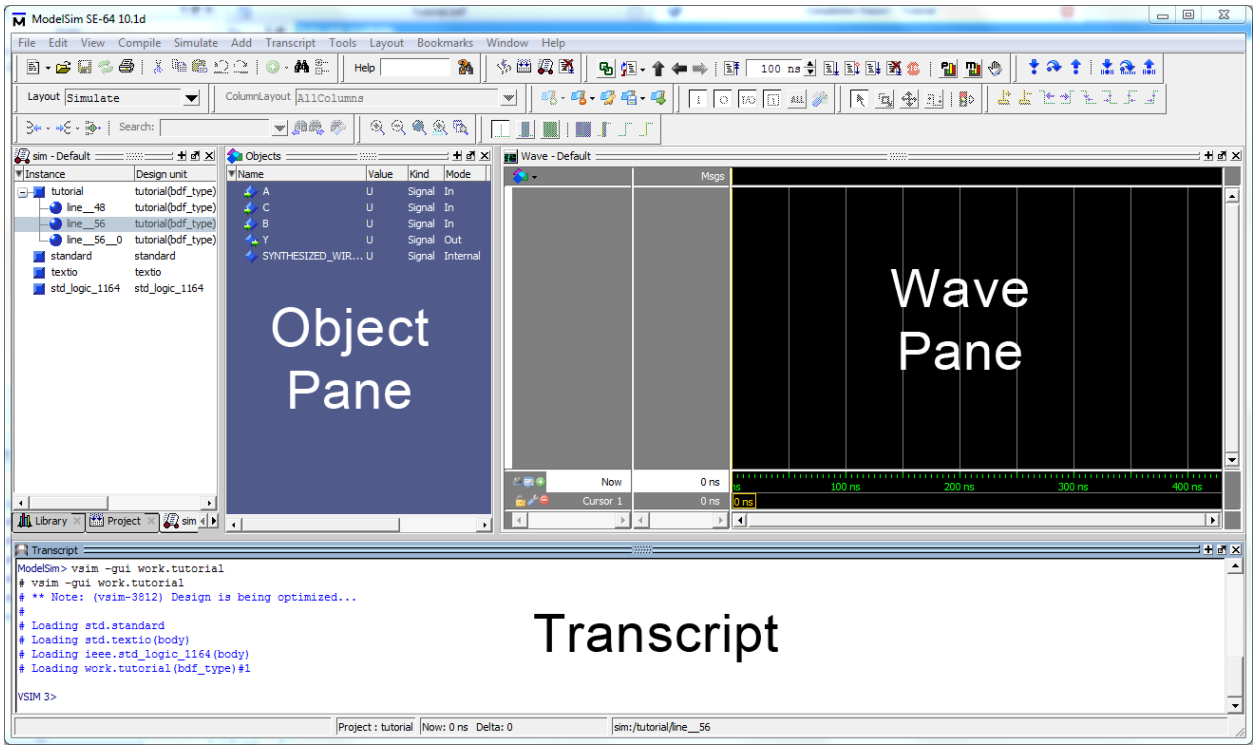


Figure 1.5: ModelSim screen after selecting Start Simulation.

4. Next, create waveforms for the inputs to the design by doing the following: To simulate all possible A, B, and C inputs requires eight different combinations. For this tutorial, these combinations are applied for 50 ns each and correspond to the order of the truth table for the design, where C varies in each combination, B varies in every other combination, and A varies half as often as B.
5. To see the simulation results, all signals must appear in the Wave window. In the Objects window, left-click on the top signal, hold shift, left-click on the bottom signal (this should select all signals) and select Add → To Wave → Selected signals. Figure 1.6

shows the resulting Wave pane sidebar. Drag the signal names around in the wave window to reorder them to the most logical order.

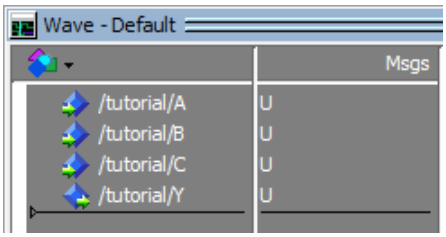


Figure 1.6: ModelSim Wave pane sidebar after adding signals

6. To zoom the view of the Wave pane, press the + or - magnifying glass icon while the Wave pane is active.
7. To adjust grid lines in the Wave pane, from the Wave menu, select Wave Preferences to launch the Wave Window Preferences window. Next select the Grid & Timeline tab, edit the Grid Period, and then press the OK button. Figure 1.6 shows the ModelSim screen after creating waveforms and adjusting the grid period to 50 ns.

Before simulating, open up a text editor and create a data file in the Quartus project directory created in step I.A.1.

```
force objectname value1 time1, value2 time2 ...  -repeat duration
```

The force is defined in value-time pairs, where the inputs will change to the value at the given time. The repeat part is optional, and is commonly used to generate a clock. So, for example:

```
force A 0 0, 1 200ns -repeat 400ns
force B 0 0, 1 100ns -repeat 200ns
force C 0 0, 1 50ns -repeat 100ns
```

creates three clocks with 50% duty cycles which generate all of the eight possible inputs to the circuit over a period of 400 ns. When all the inputs have been assigned values, save the file (any filename will do) and quit the text editor. Next, the forces data has to be loaded into the simulator by typing:

```
do forcesfilename
```

This executes the force commands stored in the *forcesfilename* file. The forces can also be entered at the prompt in the Transcript pane, instead of from a file, for quick changes in the time-value pairs. If ModelSim cannot find the file, type `pwd` in the Transcript box to check which directory it is looking in.

D. Running the Simulation and Analyzing the Results

1. First, set the length of simulation. For this tutorial, enter 400 ns as the time for simulation in the Run Length text box at the center of the top row of ModelSim tool bars.
2. Reset the simulation by pressing the Restart button immediately to the left of the Run Length text box at the center of the top row of ModelSim tool bars, which will launch a Restart dialog box. After making sure all dialog box options are selected, click the OK button to reset the simulation. This clears any previously performed `force` or `do` commands.
3. Load the forces data by typing `do` *forcesfilename* in the Transcript box.
4. Next run the simulator by clicking the Run button, immediately to the right of the Run Length text box. The resulting simulation waveforms are shown in Figure 1.7.

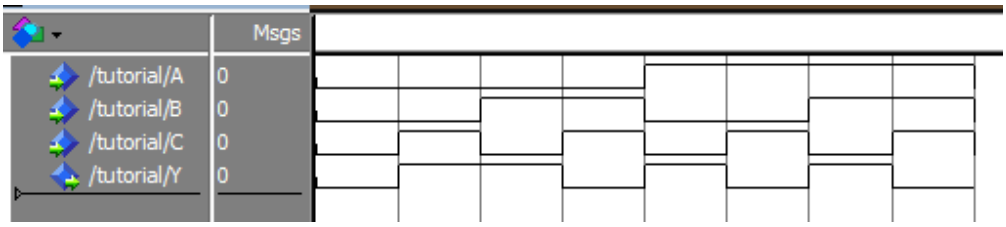


Figure 1.7: Simulation results in the Wave pane

E. Saving Simulation Results
   1. From the File menu, select Print.
   2. Check the "Print to file" box.
   3. Select the appropriate time range. Usually "Full Range" is best.
   4. Be sure to add ".xps" to the end of the file name.

Demonstrate results to the lab TAs or instructor.

Part 2

Follow the same steps as presented in Part 1 to implement and simulate the following Boolean expression:

$$Y = (A \text{ NAND } (\text{NOT } B)) \text{ AND } (C \text{ OR } D)$$

Note that this will require modifying the force file to include `D`. When considering how to do so, be sure that all test cases are covered, much like a truth table. Demonstrate results to the lab TAs or instructor.

Exercise 1: Altera Quartus II and ModelSim Tutorial

Student's Name:_____ Section:_____

Demo		Point Value	Points Earned	Date
Part 1	Circuit Schematic	20		
	Circuit Simulation	20		
Part 2	Circuit Schematic	20		
	Circuit Simulation	40		

Total	100		

A report is not required for this exercise.

Exercise 2 Simple Digital Inputs and Outputs

Objective

The objective of this exercise is to build and test the input and output (I/O) circuits that will be used for verifying digital logic circuits of subsequent exercises. By doing this you will become familiar with parts placement and schematic annotation of pin numbers and reference designators for gate/IC selection.

Introduction

What is a digital circuit? Digital circuits usually consist of 3 stages: the input, the logic, and the output, as shown in Figure 2.1. The input stage, as presented on the left side of the diagram, provides binary 0s and 1s for the implemented logic circuit in the middle. The output stage on the right side of the diagram is used to present the outputs in a way that is easy to interpret. For this, LEDs will be used.

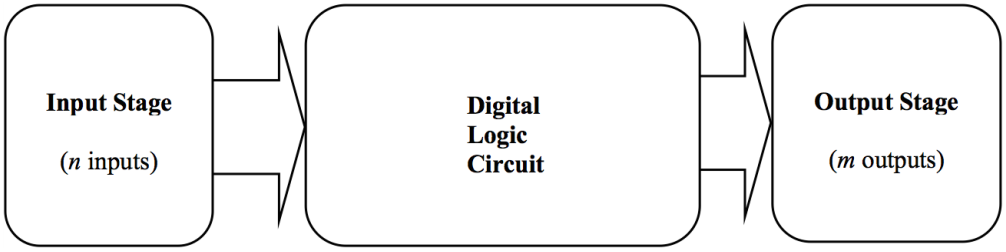


Figure 2.1: Digital logic circuit and its input/output stages

A single input/output pair is presented in Figure 2.2. The input stage consists of a resistor and a switch. The position of the switch (ON or OFF) determines whether a low (logic 0) or high (logic 1) input voltage (Vin1) is applied to the logic circuit through terminal S1. The output stage, on the right side, consists of an inverter, LED, and resistor. When the output voltage (Vout1) at the terminal L1 is high, a difference in voltage potential between V+ and V- will result in a flow of current from Vcc and thus light the LED. By contrast, a low Vout1 will result in no voltage potential difference between V+ and V- and no current flow thus the LED will be off.

Although the primary focus of the course is the design of digital logic circuits, this exercise consists of building the input and output (I/O) stages shown in Figures 2.3 and 2.4, using the IC components and wires provided in the lab kit. The input stage of Figure 2.3 consists of ten circuits, which will be used to provide digital inputs (i.e., 0s and 1s) to circuits designed later in this course. The output stage of Figure 2.3 has six output circuits with LEDs, which will allow visual inspection of future digital circuit outputs.

The input and output stages are separate, unconnected circuits. Even though they consist of more than just digital components, the planning required to construct these I/O circuits is representative of digital circuits designed in subsequent exercises. Therefore, the process of building the circuits for this exercise serves as a worked example of requirements for future circuits in this course.

Circuits in this course are typically built from gates inside integrated circuits (i.e., ICs or chips). A data sheet for each IC in the lab kit is provided on MyCourses. Select your lab section and then "content". The data sheets are in the "Reference IC data sheets" module.

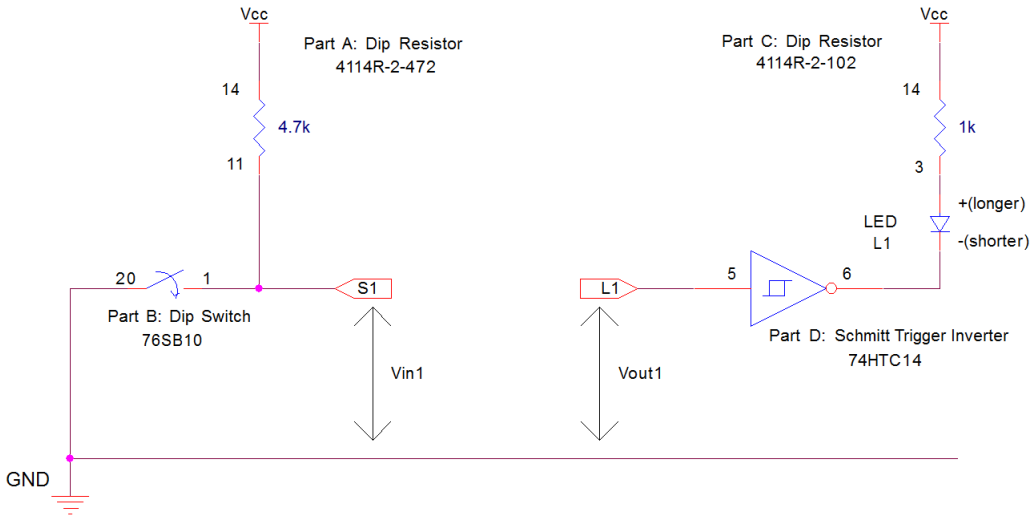


Figure 2.2: A single input/output pair.

The data sheet for an IC includes a diagram that indicates how each pin of the IC relates to the internal gates or circuit of the IC.

For each circuit implemented on the breadboard, each gate in the circuit is obtained from an IC. Thus, it is necessary to select the correct types and number of ICs needed to implement the circuit. This selection results in a mapping of circuit gates to ICs, and this mapping is documented for circuit construction with two diagrams: a completely labeled circuit diagram and a parts placement diagram. For any digital circuit constructed in this course, a completely labeled circuit diagram and a carefully thought parts-placement diagram are required. They not only serve as a blueprint for building the circuit but also significantly reduce chances of wiring error and facilitate debugging. The rest of this section explains these two diagrams and presents examples for the I/O circuits to be constructed in lab.

Completely labeled circuit diagram. In addition to depicting a circuit, a completely labeled circuit diagram, (also called a schematic diagram), indicates three things about each gate or component.

1. The part number of the IC, (e.g., 74HCT14).
2. The reference designator for the IC, (e.g., A)–a letter assigned by the circuit designer that identifies this particular IC for the circuit.
3. The pin number for each connection to the IC, (e.g., 1)–determined from the data sheet for the IC.

A digital circuit may require fewer or more gates than are packaged in an IC. Regardless of the number of gates used in circuit ICs, the circuit diagram includes only the gates used in the circuit.

Figures 2.3 and 2.4 are completely labeled circuit diagrams for the I/O circuits. Building the circuits consists of connecting the components as indicated. The IC for each component is indicated by its reference designator, (A and B in Figure 2.3 and C, and D in Figure 2.4), and part number. Once the ICs to be connected are identified, a wire is connected between the pins indicated on the diagram. In prelab work of subsequent exercises, you will draw a completely labeled circuit diagram for each circuit to be built on the breadboard. (Refer to Appendix D for more information about completely labeled circuit diagrams.)

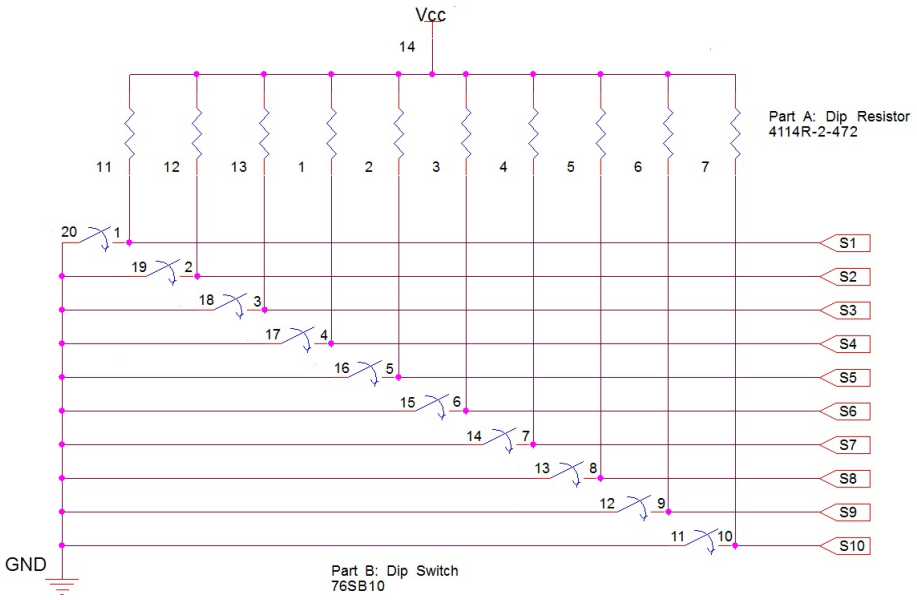


Figure 2.3: Digital logic input stage with 10 inputs

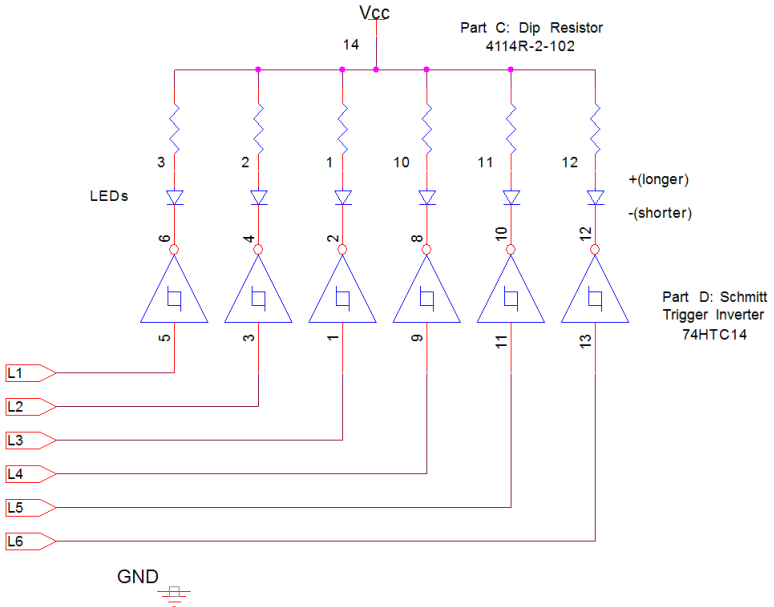


Figure 2.4: Digital logic output stage with 6 outputs

Parts placement diagram. In addition to the connectivity of circuit components, information about the location of the ICs on the breadboard is needed to build and test circuits. Another diagram, the parts placement diagram, has this information. It is a map of where each IC is located on the breadboard. In addition to mapping part locations, a parts placement diagram gives other information about each gate or component.

1. The part number of the IC, (e.g., 74HCT14).
2. The reference designator for the IC, (e.g., A)-a letter assigned by the circuit designer that identifies this particular IC for the circuit.
3. The location of pin number one, which indicates the orientation of the IC.
4. The number and location of the Vcc pin(s) not indicated on the circuit diagram–determined from the data sheet for the IC.
5. The number and location of the GND pin(s) not indicated on the circuit diagram–determined from the data sheet for the IC.

Placement of parts on the breadboard is determined by connections required among ICs. As a practical matter to reduce wiring errors and protect ICs, all ICs should be oriented the same way on the breadboard-in other words, they should be arranged so that the location of pin number one is the same for each IC. Another consideration is the connectivity provided by the breadboard. Figure 2.7 is an example that shows a parts placement diagram superimposed on a diagram of the internal connections of breadboard holes.

Although it indicates placement of parts on the breadboard, it is not a complete parts placement diagram because it lacks the reference designators needed to build the final circuit. For example, there are two 74HCT14 ICs; this lack of distinction between identical components could lead to wiring errors and debugging confusion. In subsequent exercises, you will draw a parts placement diagram for each circuit to be built on the breadboard. (Refer to Appendix D for more details on parts placement diagrams.)

Prelab

Verify the pin numbers in the completely labeled circuit diagrams (Figures 2.3 and 2.4) for the two dual inline package (DIP) resistors (4114R-2-472 and 4114R-2-102), the Schmitt-triggered inverter (74HCT14), and DIP switch (76SB10). Data sheets for these components are provided on MyCourses. Select your lab section and then "content". The data sheets are in the "Reference IC data sheets" module.

Note that there are ten input circuits (S1-S10) presented in Figure 2.3. Complete the parts placement diagram in Figure 2.5 to show connections that will be needed to implement all of them using one DIP resistor and one DIP switch. Make sure to clearly indicate connections on crossing wires with a dot at the intersection.

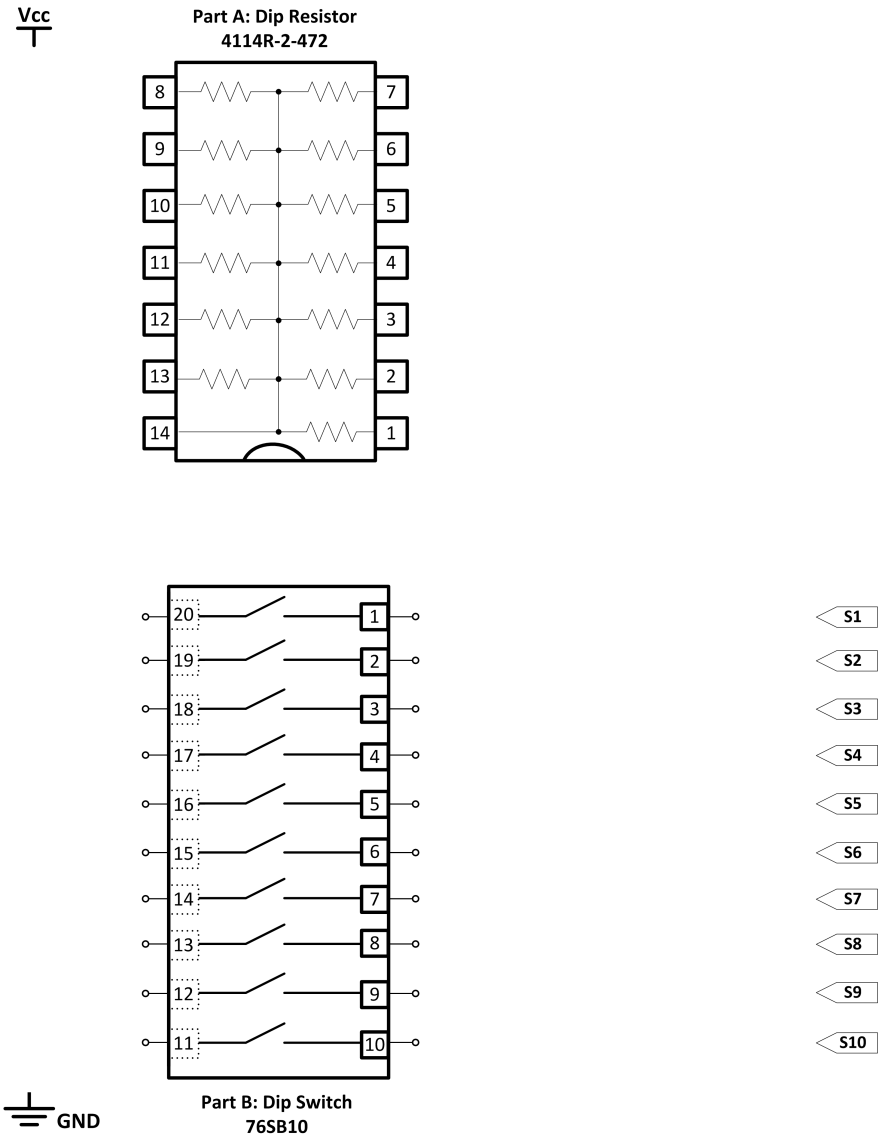


Figure 2.5: Connections of the input stage.

Note that there are six output circuits (L1-L6) presented in Figure 2.4. Complete the parts placement diagram in Figure 2.6 to show connections that will be needed to implement all of them using one DIP resistor, one package with six Schmitt trigger inverters, and six individual LEDs. Again, make sure to clearly indicate connections on crossing wires with a dot at the intersection.

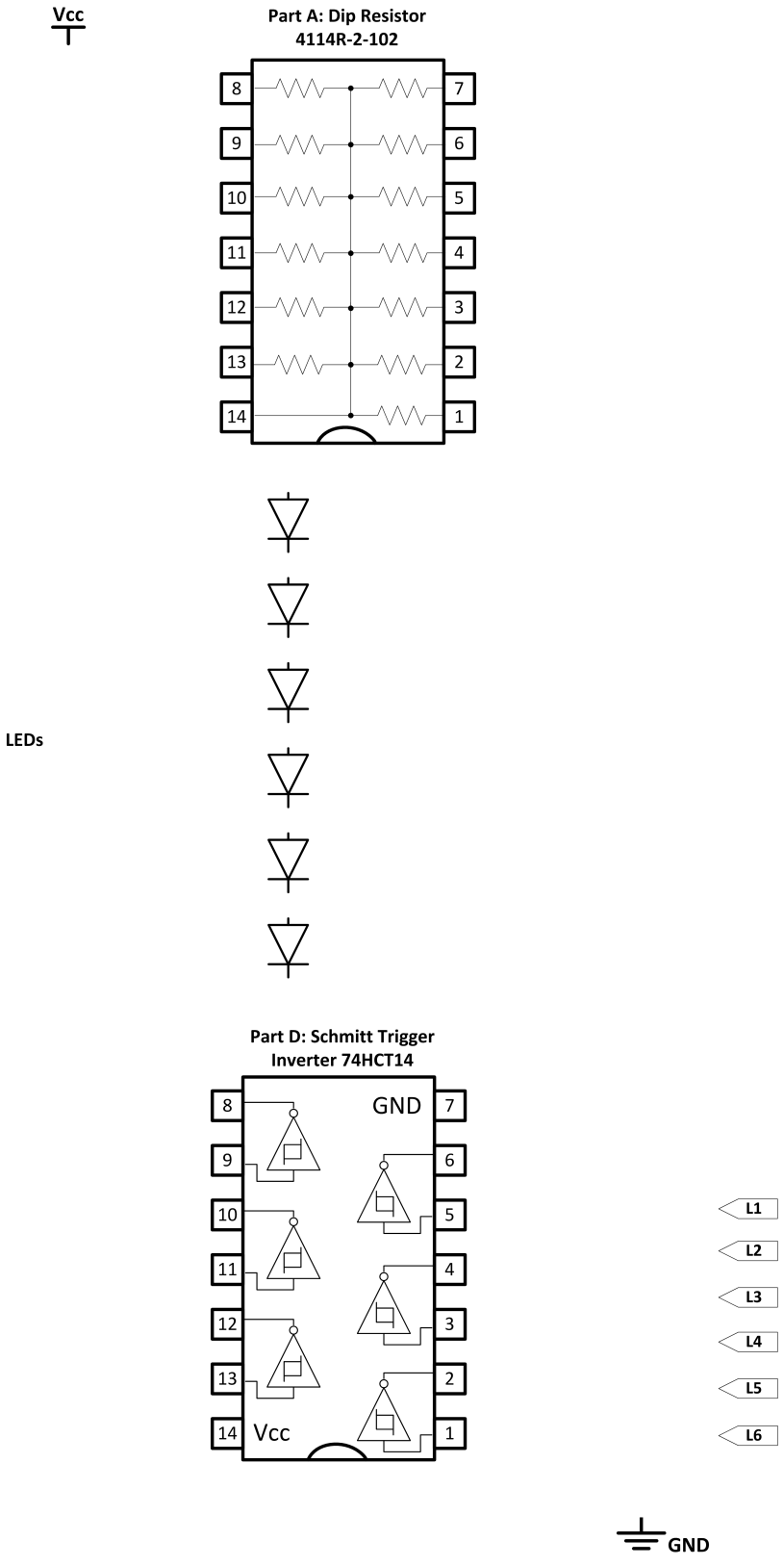


Figure 2.6: Connections of the output stage.

Lab Procedure

Build the Circuit

First, construct a single input and a single output circuit as presented in Figure 2.2 and Figure 2.7.

Make sure that both I/O stages fit on the left band of the breadboard. (There are three bands on your breadboard). The center and rightmost bands of the breadboard will be used for building logic circuits in future lab exercises.

Based on your prelab work, finish wiring the remaining inputs and outputs. The idea of this exercise is for you to design a clean and clear layout of the I/O circuits. "Neat" wiring and labels of the inputs (S1-S10) and outputs (L1-L6) will facilitate future uses of these circuits, which will serve as your I/O stages for the rest of the term.

Connect the Power Supply

Using Figure 2.8 as a reference, follow the instructions to connect the power supply to your circuit.

1. Get one red and one black banana-banana cable.
2. Insert one end of each cable into the same colored 6V terminals (1 and 2) on the power supply.
   - Red is power (1) and black is ground (2).
3. Insert the other end of each cable into the same colored terminals on the breadboard.
   - Red is power and black is ground.
4. Turn on the power supply by pressing the power button (3).
5. Ensure the display says "OUTPUT OFF", if the output is not off, press the "Output on/off" button (4).
6. To view the voltage and current limits, press the "Display Limit" button (5). A small "Lmt" will be shown near the bottom right of the display indicating that the display is in the Display Limit view.
   - After a few seconds of being idle, the display will revert to showing the output. If this happens before finishing setting the limits, press the "Display Limit" button again.
7. Ensure that the limits for the 6V output are being displayed. A small "+6V" should be shown near the bottom left of the display. If this is not shown or "+25V" or "-25V" is shown instead, press the "+6V" select button (6).
8. In the Display Limit view, the voltage limit is shown on the left and the current limit is shown on the right. The active digit will be flashing. Rotating the control knob (7) will change the value of the active digit. Use the arrow buttons (8) to change which digit is active. Use the "Voltage/Current" button to toggle between changing the voltage and current limits.
9. Set the voltage limit to 5V. Set the current limit to 0.1A.
   - These limits prevent too much voltage or current from being provided to a circuit. The power supply will provide just enough power to reach one of the limits.
10. Once the limits are set properly, press the "Output on/off" button (4) to provide power to your circuit. The display will now show how much voltage and current the power supply is providing.
    - For a properly wired ciruit, the Output view will show that approximately 5.0V and a few milliamps are being provided to the circuit. If the display shows that 0.1A (the current limit) is being provided, the circuit was not wired correctly. In this case, press the "Output on/off" (4) button to stop providing power to the ciruit. Repair the ciruit before providing power again.
    - It is easy to confuse the Output view with the Display Limit view. If there is a small "Lmt"

near the bottom right of the display, the Display Limit view is being shown, not the Output view.

Do you see something strange in the behavior of your circuit? Most likely, your diodes are either always on or periodically turning on and off. Turn off the power supply and then fix this undesired behavior by adding six discrete 100 kΩ resistors to pull down (connect to GND via a resistor) inputs of inverters in the 74HCT14 IC - one resistor for each inverter that is used in your circuit. Turn the power supply back on and make sure all of the LEDs are now off.

Test the Circuit

Test your input stage by using one wire to connect each one of the ten inputs circuits (S1-S10)–individually, one at a time–to one of the outputs. (Choose one output circuit, and use it to test each input circuit.) For each input, when you close its switch, an input of *low voltage* will be applied to the output stage, and the LED will turn *off*. By contrast when you open the switch, an input of *high voltage* will be applied to the output stage, and the LED will turn *on*. For future reference, you may label the *open* position as *high* or "1" and the *closed* position as *low* or "0," to represent the voltage as well as the logical level.

Similarly, test your output stage by using one wire to connect each one of the six output circuits (L1-L6)–individually, one at a time–to one of the inputs. (Choose one input circuit, and use it to test each output circuit.)

Demonstrate your correctly operating input and output circuits to the lab instructor with the inputs and outputs labeled.

Question

1. Explain the purpose of the additional 100 kΩ pull down resistors that you added after first powering on the circuit.

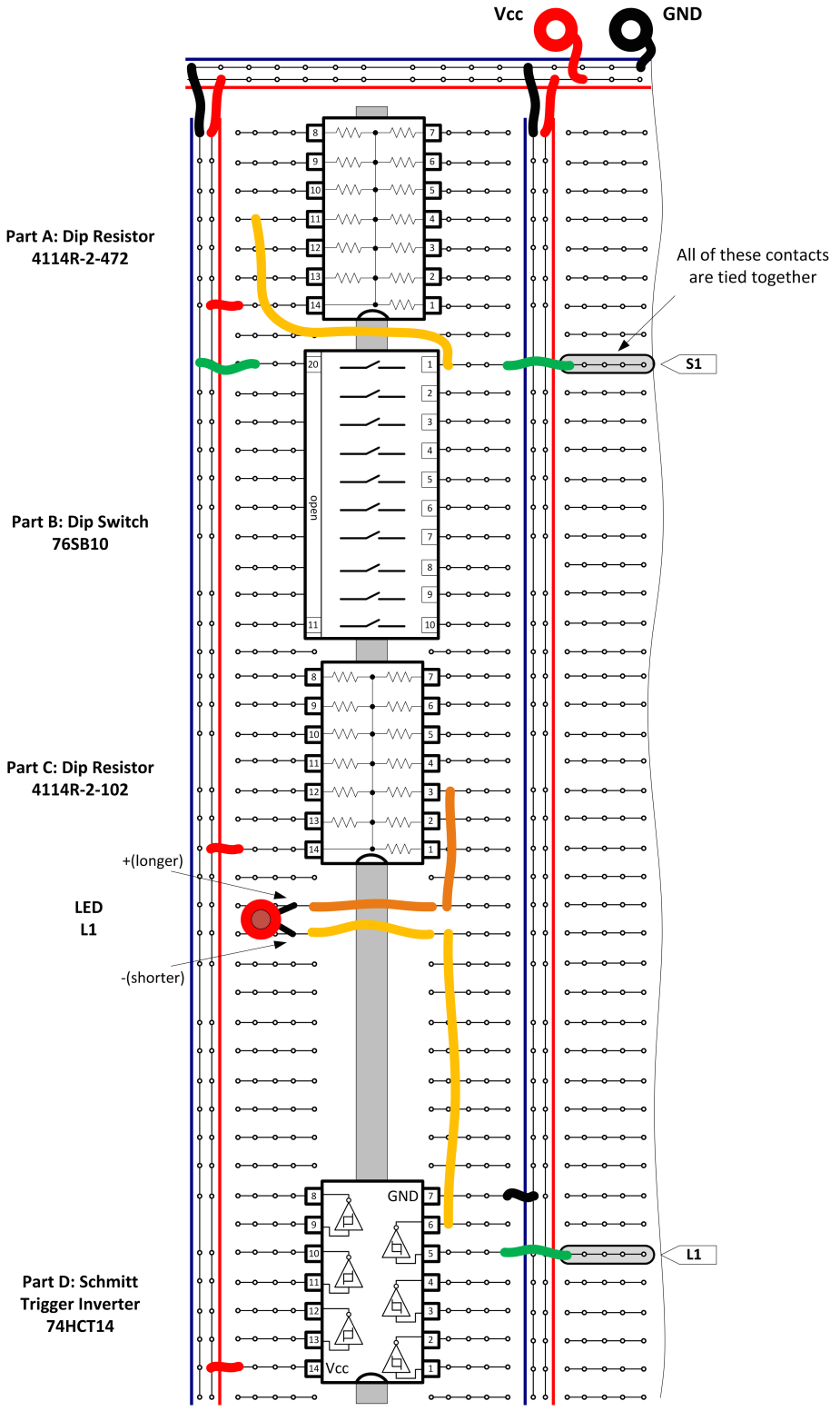


Figure 2.7: Connections of a single input and a single output circuit shown on a breadboard.

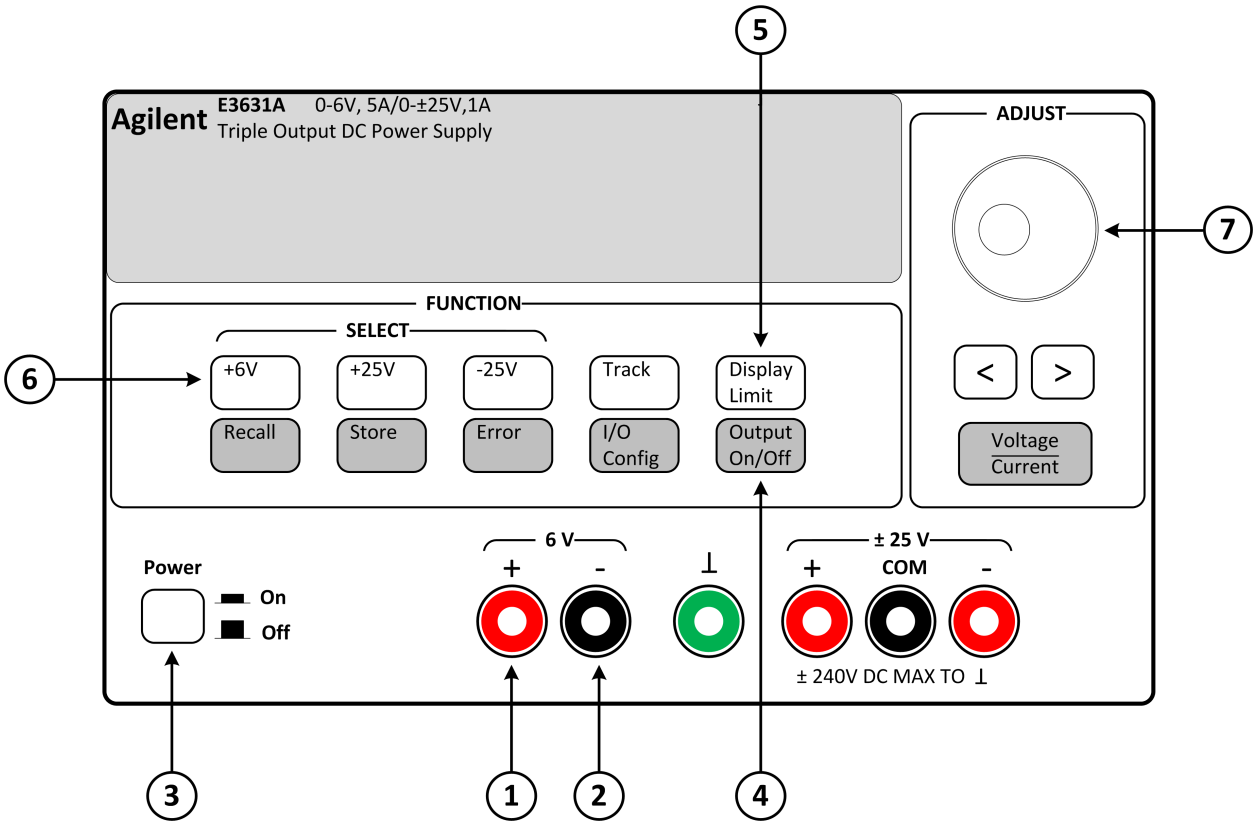


Figure 2.8: An Agilent E3631A Power Supply.

Exercise 2: Simple Digital Inputs and Outputs

Student's Name:_____ Section:_____

Prelab	Point Value	Points Earned	Comments
Prelab	20		

Demo		Point Value	Points Earned	Date
Demo	Demo	60		
	Question	20		

Total	100		

A report is not required for this exercise.

Exercise 3 Electrical and Logical Characteristics of Gates

Objective

The objective of this exercise is to investigate some of the electrical and delay characteristics of combinational logic gates.

Prelab Work

Part 1

Using the data sheet for a quad two-input AND gate (74LS08), complete Table 3.1.

Table 3.1: Electrical characteristics

	Voltage (V)
$V_{IL,max}$	
$V_{IH,min}$	
$V_{OL,max}$	
$V_{OH,min}$	

Based on the circuit presented in Figure 3.1 and using your own judgment without actually conducting the experiment, complete Table 3.2.

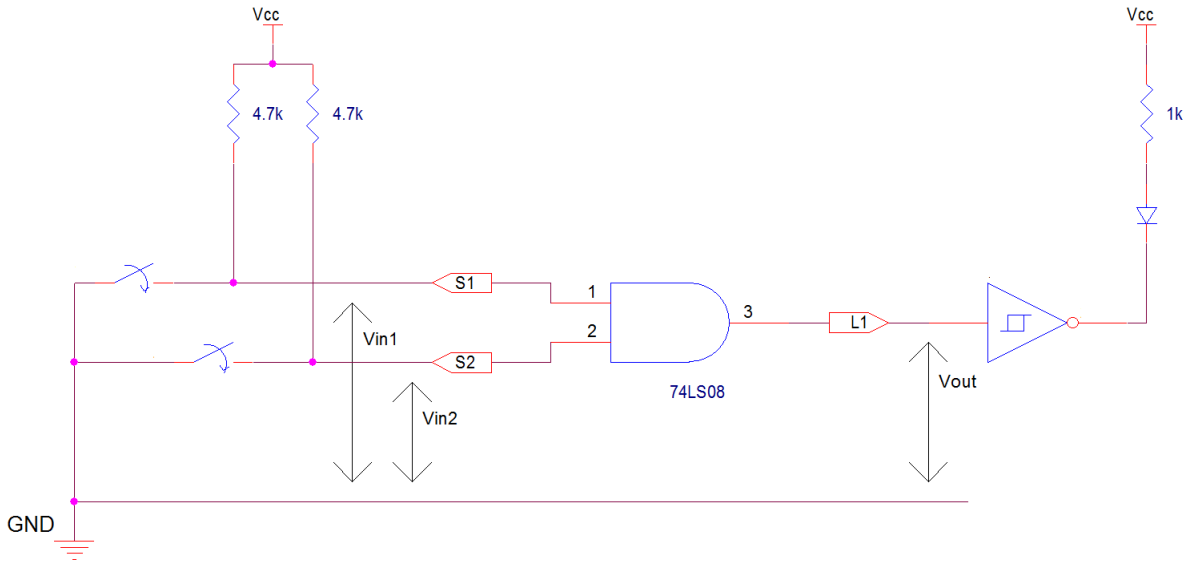


Figure 3.1: Circuit diagram for an AND gate circuit.

Table 3.2: Voltage characteristics of a two input AND gate circuit

Switch 1 Position	Switch 2 Position	$V_{in,1}$ (V)	$V_{in,2}$ (V)	V_{out} (V)	LED (on/off)
Closed	Closed				
Closed	Open				
Open	Closed				
Open	Open				
Logical function of gate:				AND	
Lab kit part number for gate:				74LS08	

Complete Figure 3.2 to show the internal connections of the AND gates inside the package. Also show how one of these gates would be connected for measurements as presented in Figure 3.1. Showing the components of the I/O circuit is not necessary.

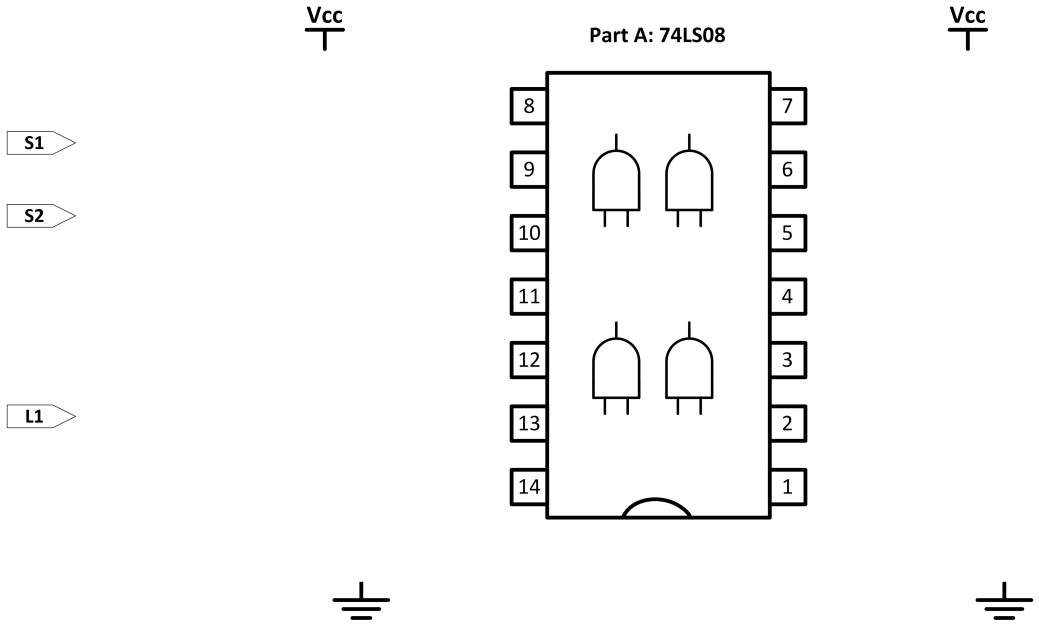


Figure 3.2: Parts placement diagram for AND gate circuit

Part 2

Using the data sheet for a hex inverter (74LS04), complete Table 3.3.

Table 3.3: Timing characteristics

		$R_L = 2\text{k}\Omega$, $C_L = 15\text{pF}$	
Symbol	Parameter	Min (ns)	Max (ns)
T_{PLH}	Propagation delay time Low to high level output		
T_{PHL}	Propagation delay time High to low level output		

Based on the circuit presented in Figure 3.3 and using your own judgment without actually conducting the experiment, complete Table 3.4 with the times you expect to find when you take the actual measurements in lab.

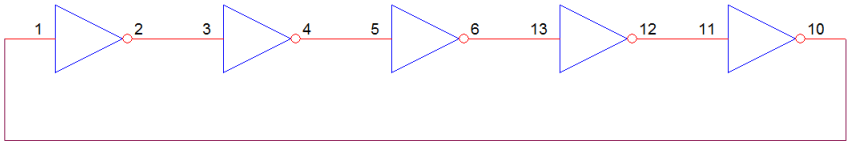


Figure 3.3: A ring oscillator

Table 3.4: Effect of propagation delay

74LS04 (pin to pin)	Number of Gates of Delay	Propagation Delay (ns)
2 to 4		
2 to 6		
2 to 12		
2 to 10		

Complete Figure 3.4 to show the internal connections of the inverters inside the package. Also show how five of the six available inverters will be used to construct the oscillator circuit as presented in Figure 3.3. Show connections that will be needed to conduct all of the measurements.

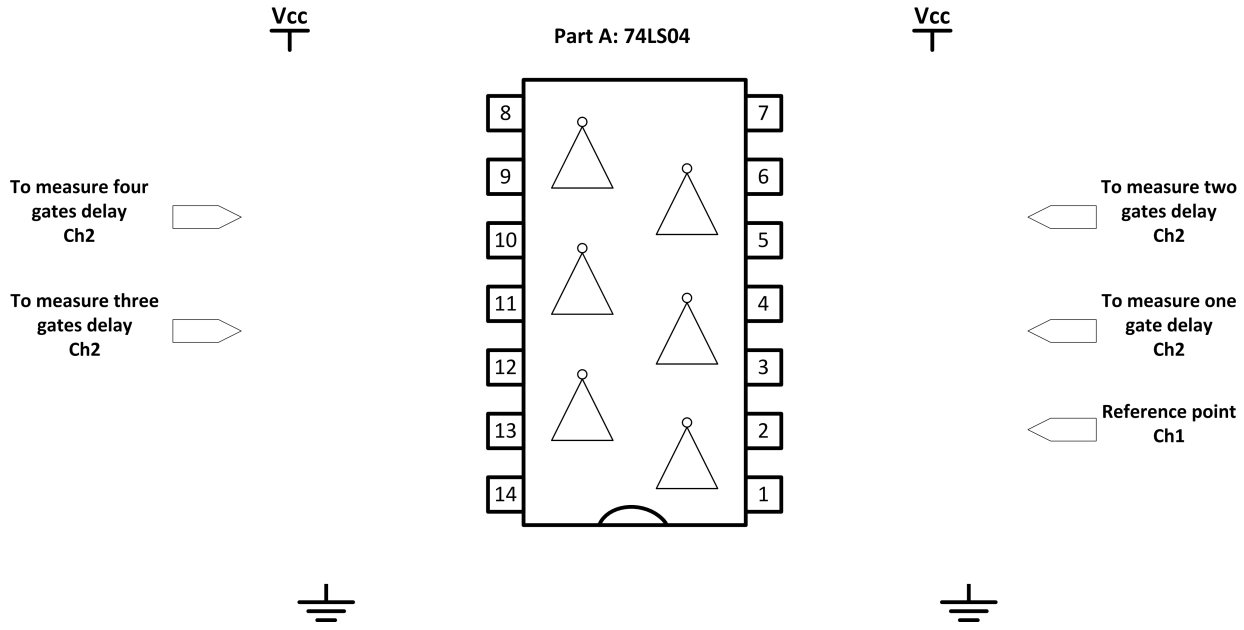


Figure 3.4: Parts placement diagram for ring oscillator

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session. Your prelab grade is based on reasonable effort and logical reasoning demonstrated for your answer.

Lab Procedure

Use the I/O circuit board you built in Lab Exercise Two to test three of the two-input chips in your lab kit, the NAND, OR, and XOR. Be sure that you have selected the quad 2-input version of the chip. Place the first IC on the breadboard separately and wire it properly to the switches and LEDs of your I/O circuit board. You may assume that it has the same pinout as a two-input AND gate. Using Figure 3.5 as a reference, follow the instructions to measure voltage using the multimeter.

1. Get one red and one black banana-banana cable.
2. Insert the banana end of the black cable into the rightmost black terminal (1).
3. Insert the banana end of the red cable into the upper rightmost red terminal (2).
4. Turn on the multimeter by pressing the power button (3)
5. Press the "DC V" button (4) to make sure DC voltage is being measured.
6. Attach the black and red alligator clips across where voltage is to be measured.

Complete Table 3.5 to characterize the functionality of the NAND gate. Repeat the process for the OR and XOR gates using Table 3.6 and Table 3.7, respectively. To make the process faster, you may reuse the connections you made for the NAND gate. Demonstrate your circuit and results to the lab instructor. Note that Vin,1 and Vin,2 are the voltages which appear on the two input pins of the gate, and Vout is the voltage that will appear at the output of the gate. Measure all voltages relative to GND of the IC.

In addition to the voltages, record the logical function and the lab kit part number of the gate.

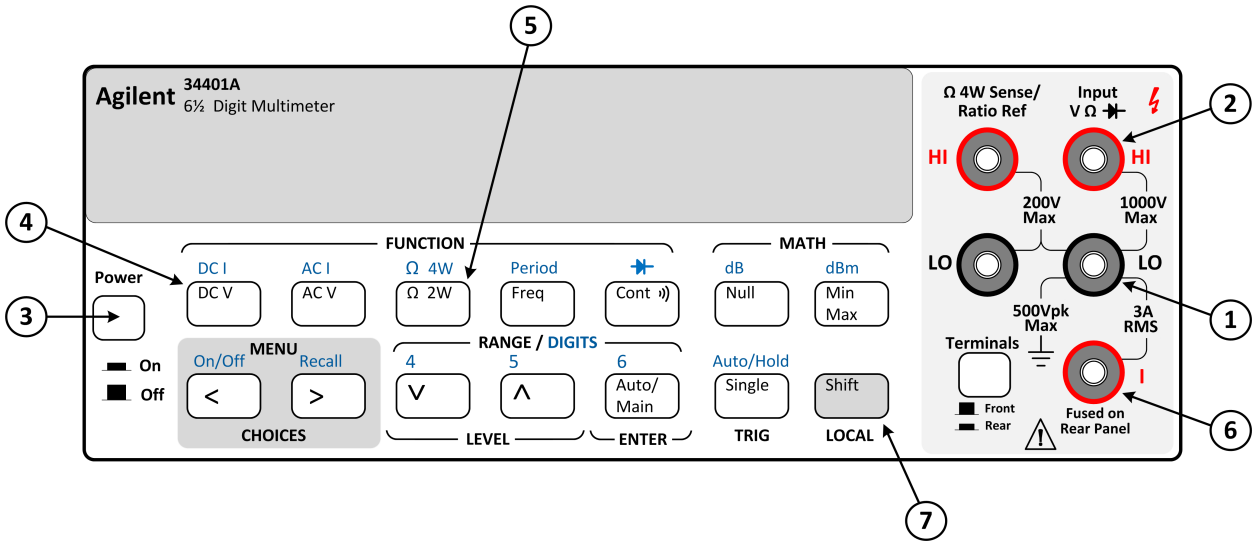


Figure 3.5: An Agilent 34401A Multimeter

Table 3.5: Measurements of a two input NAND gate circuit

Switch 1 Position	Switch 2 Position	$V_{in,1}$ (V)	$V_{in,2}$ (V)	V_{out} (V)	LED (on/off)
Closed	Closed				
Closed	Open				
Open	Closed				
Open	Open				
			Logical function of gate:		
			Lab kit part number for gate:		

Table 3.6: Measurements of a two input OR gate circuit

Switch 1 Position	Switch 2 Position	$V_{in,1}$ (V)	$V_{in,2}$ (V)	V_{out} (V)	LED (on/off)
Closed	Closed				
Closed	Open				
Open	Closed				
Open	Open				
			Logical function of gate:		
			Lab kit part number for gate:		

Table 3.7: Measurements of a two input XOR gate circuit

Switch 1 Position	Switch 2 Position	$V_{in,1}$ (V)	$V_{in,2}$ (V)	V_{out} (V)	LED (on/off)
Closed	Closed				
Closed	Open				
Open	Closed				
Open	Open				
			Logical function of gate:		
			Lab kit part number for gate:		

Use the circuit shown in Figure 3.3 to study the effect of propagation delay. Notice that each 74LS04 inverter output connects to the next 74LS04 input. Keeping wiring as short as possible, construct circuit from Figure 3.3 as shown in the parts placement diagram presented in Figure 3.4. Be sure to attach power and ground to the IC.

Using the oscilloscope, connect CH1 to pin 2 and CH2 to pin 4, ensuring that both leads are grounded. Create a new measurement using the Meas button menu, followed by selected the Type Delay. Determine the time difference between the rising edge of CH1 and falling edge of CH2. To change which edges are measured between, press the Settings softkey, followed by the necessary softkeys. Press Back followed by Add Measurement.

Leave CH1 at pin 2 and move CH2 to pin 6. This time, measure the time difference between the rising edge of CH1 and rising edge of CH2. Each time you measure across an additional inverter, you must change which edges you measure between because the new inverter inverts the signal. This is illustrated in Figure 3.6. Demonstrate the circuit and time delay measurements to the lab instructor. Repeat the measurement, leaving CH1 on pin 2 and moving CH2 to pins 12 and 10. Include an oscilloscope capture of the delay measurement in the report.

Record the results in Table 3.8, and plot propagation delay versus number of TTLs gate on a graph. Include this graph in your report, along with a line of best fit for the data. Be sure to explain what the slope and intercept mean. Also indicate on the graph the T_{PLH} minimum and the T_{PLH} maximum time delay specifications given in the 74LS04 data sheet.

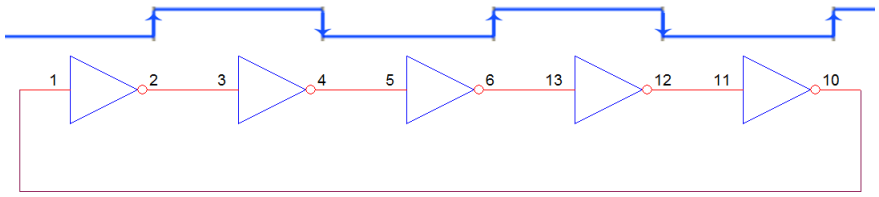


Figure 3.6: Illustration of which edges to measure between.

Using the oscilloscope, connect CH 1 to pin 2 (turn CH2 off by pressing the 2 button) and use the scope to measure the period of the square wave. This measurement can be selected by pressing the Meas button, Type softkey, and the scrolling to find Period. Consider the number of gates the signal must travel through for the pin to go both high and low. Use the period and number of gates to

Table 3.8: Effect of propagation delay

74LS04 (pin to pin)	Number of Gates of Delay	Propagation Delay (ns)
2 to 4 (rising to falling)		
2 to 6 (rising to rising)		
2 to 12 (rising to falling)		
2 to 10 (rising to rising)		
Period:		

determine the average propagation delay of the TTL inverter. Include a oscilloscope capture of the period measurement in the report. If desired, this can be combined with the delay measurement, as long as both measurements are shown clearly.

Look at the questions for this exercise, and ask your instructor about the meaning of any question(s) you do not understand.

Lab Report

If you do not get above an 80% on the lab report, counting only the points from the report, not those from the prelab or demo, you will be required to meet with your TA to receive your corrected report. This meeting will occur before the next report is due, so that you can incorporate the feedback into following report.

Questions

1. What are these voltage values for the 74LS27? (Hint: Check MyCourses for the data sheet)
   (a) Minimum input voltage for logic "1."
   (b) Maximum input voltage for logic "0."
   (c) Minimum output voltage for logic "1."
2. Define propagation delay, and discuss how it impacts the output from a gate. Why is it a consideration in digital designs?
3. Design two different circuits that implement a 2-input XOR function using a single 74LS08 (quadruple 2-input AND gate package), a single 74LS32 (quadruple 2-input OR gate package), and a single 74LS04 (6-inverter package). Draw the circuit diagrams with the pin numbers properly labeled. You do not need to draw the I/O circuit–use S1 to refer to input 1, L1 as output 1, and so on. (Hint: XOR can be implemented with signals going into AND gates first or going into OR gates first.)

Exercise 3: Electrical and Logical Characteristics of Gates

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Prelab	Part 1	10		
	Part 2	10		

Demo		Point Value	Points Earned	Date
Demo	Voltage measurements	20		
	Ring oscillator	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 3: Electrical and Logical Characteristics of Gates

Report		Point Value	Points Earned	Comments
Abstract		3		
Design Methodology	Voltage measurement circuit schematic	4		
	Ring oscillator circuit schematic	4		
Results and Analysis	Voltage tables	4		
	oscilloscope capture(s)	2		
	Delay table	3		
	Delay graph	4		
Conclusion		3		
Questions	Q1	3		
	Q2	3		
	Q3	4		
Writing Composition		3		
Total for prelab, demo, and report		100		

Exercise 4 Combinational Logic Circuit Design Using Boolean Algebra Simplification

Objective

The objective of this exercise is to apply Boolean algebraic simplification to a Boolean expression and to implement the simplified expression with a combinational logic circuit.

Design Description

Combinational logic circuits can be designed to perform various mathematical operations as well as to provide control signals for devices. Design a logic circuit that will read a single two-bit binary number, $N = (N1, N0)$, and will produce a four-bit number, $F = (W, X, Y, Z)$ as an output, which depends on the selection control signal (C). This output value will be the square of N, ($F = N^2$), whenever C is at logic state "0," and the output value will be five times the input, ($F = 5 \times N$), whenever the C is at logic state "1."

Prelab Work

Part 1

Complete the truth table, Table 4.1, for the operations described.

Table 4.1: Truth table for mathematical operations N^2 and $5N$

C	$N1$	$N0$	W	X	Y	Z

For each output, write the sum of products Boolean expression, where each product term will include all three input variables.

W (C, N1, N0) = _____

X (C, N1, N0) = _____

Y (C, N1, N0) = _____

Z (C, N1, N0) = _____

Part 2

Apply Boolean algebraic properties to simplify the Boolean expressions from Prelab Part 1 so that they can be implemented with the fewest parts from the lab kit. Show each step in the simplification, and list the Boolean property applied in that step.

Part 3

For the simplified expression from Prelab Part 2, design a circuit to implement it using basic gates. Gates with more than 2 inputs can be used, but be sure to only use gates provided in the lab kit. For instance, because there is no 3-input AND gate in the lab kit, consider using a 3-input NAND gate followed by an inverter or two 2-input AND gates. Use Altera Quartus II to draw a *completely labeled circuit diagram* (i.e., with pin numbers) for your design. Remember to refer to the data sheets and manually indicate the pin numbers on the logic gates you use. Note that the pin numbers do not automatically appear in Quartus II; therefore, you will need to enter them manually. Next, draw a parts placement diagram. Finally, simulate your design in ModelSim to generate a waveform that verifies it is correct.

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Lab Procedure

Construct the circuit from Prelab Part 3 by using four LEDs to represent W, X, Y, and Z, and by using three switches for inputs C, N1, and N0. Test and demonstrate your correctly built circuit to the lab instructor or TA.

Look at the questions for this exercise, and ask your instructor about the meaning of any question(s) you do not understand.

Questions

There are various metrics to evaluate the efficiency of an implementation of a Boolean expression. For each of the metrics listed, give its value a) for direct implementation of the expressions from Prelab Part 1 with AND, OR, and inverter gates and b) for your simplified expressions as implemented in lab.

1. Equivalent gate count (EGC)–the total number of 2-input gates required to implement the expression.
2. Chip count from basic gates (i.e., ANDs, ORs, and inverters)–the total number of ICs (chips) required to implement the expression.

Exercise 4: Combinational Logic Circuit Design Using Boolean Algebra Simplification

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Part 1	Table completed	3		
	Expressions correct	2		
Part 2	Correct simplified equations	1		
	Steps shown	2		
	Boolean properties shown	2		
Part 3	Correct schematic	3		
	Pin and chip numbers	1		
	Correct simulation	4		
	Parts placement diagram	2		

Demo		Point Value	Points Earned	Date
Demo	Two fully functional output pins	20		
	Additional two fully functional output pins	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 4: Combinational Logic Circuit Design Using Boolean Algebra Simplification

Report		Point Value	Points Earned	Comments
Abstract		5		
Design Methodology	Truth table	4		
	Boolean expressions	4		
	Correct simplified equations	1		
	Steps shown	2		
	Boolean properties shown	2		
	Schematic	4		
Results and Analysis	Discussion	5		
Conclusion		3		
Questions	EGC	3		
	Chip count	3		
Writing Composition		4		
Total for prelab, demo, and report		100		

Exercise 5 Combinational Logic Circuit Design Using Karnaugh Map Simplification

Objective

The objective of this exercise is to apply combinational logic design principles, including Karnaugh-map (K-map) techniques, to the implementation of two different design problems.

Design Description

Any Boolean expression can be implemented using various combinational logic circuits whose outputs are equivalent. This exercise investigates circuits to implement the Boolean expression $F = \Sigma_{ABCD}(0, 2, 3, 4, 6, 7, 13, 14, 15)$.

Prelab work

Part 1

Use K-maps below to determine the minimum SOP and POS expressions for F. Clearly circle 1s and 0s to indicate the reduced function.

AB \ CD	00	01	11	10
00				
01				
11				
10				

$F_{SOP} =$ _____

AB \ CD	00	01	11	10
00				
01				
11				
10				

$F_{POS} =$ _____

Part 2

1. Locate and download this lab's VHDL test bench from MyCourses, `tb_ex5.vhd`. Save the test bench in the project directory with the circuit design created in prelab.
2. Read the comments at the top of the test bench so your pins are named correctly in step 3.
3. Use Altera Quartus II to draw circuit diagrams for both representations of function F. Both circuits should be in the same file. Each of the four inputs will be used twice, once per circuit. Different outputs, one per representation, are needed. Simulate your design in ModelSim to generate a waveform that verifies that both implementations are correct.
4. Calculate EGC and chip count for each expression using only two input gates for the EGC. Include all required gates (AND, OR, inverters) in the chip count.
5. Completely label (pin numbers and chip numbers) the diagram of the circuit with the lower cost. Remember to refer to the data sheets and manually indicate the pin numbers on the logic gates you use. (Note that the pin numbers do not automatically appear in Quartus II; therefore, you will need to enter them manually.)
6. Finally, draw a parts placement diagram.

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Lab Procedure

A second method for circuit simulation are VHDL test benches. Used in industry, test benches have significantly more functionality than force statements. Because VHDL has not been covered in lecture, the VHDL test bench code will be provided in a file found on MyCourses. The commands that are similar to the force statements will be discussed before VHDL is covered in lecture. The following procedure will produce the same results generated in the prelab using force statements with a VHDL test bench.

1. Open the file and look through it. Find the lines near the bottom that look like "`A <= not A after 50 ns;`". This is equivalent to the force statement, "`force A 0 0, 1 50ns -repeat 100ns`". The force statement explicitly tells the A signal to be 0 for 50 ns and then 1 for 50 ns and then repeat. The VHDL only tells A to become the opposite ("not A") after 50 ns. Both of these have the same outcome.
2. Check to ensure that the names of the signals in the Quartus file match that of those in the test bench. If they do not, rename the signals in Quartus and re-export the HDL file.
3. As in prelab, launch ModelSim and create a new project.
   (a) In the Project Name text box, enter the project name used in Quartus ADDING the letters "tb" to the end of the name.
   (b) Use the project directory for the Project Location and click OK.
   (c) Click the Add existing icon to launch the Add to Project window and click Browse to add files.
   (d) Select the .vhd created by Quartus AND the .vhd test bench file downloaded from MyCourses. Press Open to close then Select to add to project window. Both files will be located in the project directory. Windows may not show the .vhd extention. The .vhd files may look like text documents.
   (e) Click the OK button to close the Add to Project window.
   (f) Click the Close button to close the Add Items to the Project window. Two VHDL files should

be listed in the Project pane.

4. From the Compile menu, select Compile All to compile the design. When the project has been successfully compiled, the "?" under Status in the Project pane will change to "✓" as shown in Figure 1.4.
5. Because a test bench is in the project, it will provide input stimulus. Do not force the inputs.
   (a) From the Simulate menu, select Start Simulation, which launches the Start Simulation window.
   (b) Under work (in the default Design tab), select the test bench file.
   (c) Uncheck the Enable Optimization checkbox at the bottom of the Start Simulation window.
   (d) Click the OK button. The Start Simulation window will close, and simulation mode will start with a sim pane on the left, an Objects pane in the center, a Wave pane on the right, and a Transcript pane on the bottom. If the ModelSim screen does not automatically launch those panes, use the View menu to open any missing windows.
6. To see the simulation results, all signals must appear in the Wave window. In the Objects window, right-click on the top signal, hold shift, right-click on the bottom signal (this should select all signals) and select Add → To Wave → Selected signals. Drag the signal names around in the wave window to reorder them to the most logical order.
7. Set the length of simulation. For this tutorial, enter 800 ns as the time for simulation in the Run Length text box at the center of the top row of ModelSim tool bars.
8. Reset the simulation by pressing the Restart button immediately to the left of the Run Length text box at the center of the top row of ModelSim tool bars, which will launch a Restart dialog box. After making sure all dialog box options are selected, click the OK button to reset the simulation.
9. Next run the simulator by clicking the Run button, immediately to the right of the Run Length text box.

Save the simulation results and demonstrate results to the lab TAs or instructor.

Construct the circuit from Prelab Part 2 with the lower EGC cost by using one LED to represent F and four switches to represent inputs A, B, C, and D. Test and demonstrate your correctly built circuit to the lab instructor or TA.

Look at the questions for this exercise, and ask your instructor about their meaning if you do not understand.

Questions

1. For the SOP expression determined in Prelab Part 1, design a 2-level NAND-NAND implementation. Inverters are not counted towards the total and may only be used before the first level. Draw a schematic of this circuit in Altera Quartus II.
2. For the POS expression determined in Prelab Part 1, design a 2-level NOR-NOR implementation. Inverters are not counted towards the total and may only be used before the first level. Draw a schematic of this circuit in Altera Quartus II.

Exercise 5: Combinational Logic Circuit Design Using Karnaugh Map Simplification

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Part 1	K-maps	4		
	Correct Boolean expressions	4		
Part 2	Schematic diagram	4		
	Simulations	4		
	EGC	4		

Demo		Point Value	Points Earned	Date
Demo	Test Bench	20		
	Circuit Construction	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 5: Combinational Logic Circuit Design Using Karnaugh Map Simplification

Report		Point Value	Points Earned	Comments
Abstract		4		
Design Methodology	Karnaugh maps	5		
	Reduced SOP expression	5		
	Reduced POS expression	5		
	Schematic diagram	4		
Results and Analysis	SOP waveforms	2		
	POS waveforms	2		
Conclusion		5		
Question		5		
Writing Composition		3		
Total for prelab, demo, and report		100		

Exercise 6 Binary Addition and Subtraction Circuits

Objective

The objective of this exercise is to design, implement, and study binary arithmetic circuits. A full adder is designed and constructed from basic logic gates, and a four-bit adder/subtracter is built from a four-bit full adder. One-bit and four-bit arithmetic operations are tested.

Prelab Work

Part 1

Derive the Boolean expressions for the sum and carry out outputs of a one-bit full adder in the form of a minimum sum of products using Karnaugh maps. Use Boolean transformations to express the sum output in a form that uses only XOR operations. Based on those expressions, design a full-adder circuit using basic logic gates available in your lab kit. Show all of your design work clearly, including the Karnaugh maps and all steps of the Boolean transformations used to obtain the expressions. Draw a schematic diagram of your circuit with Quartus II. Remember to show the pin numbers on your schematic diagram. Simulate your design in ModelSim using force statements to generate a waveform that verifies it is correct. Then, download the test bench file for this exercise and look through it and compare what has changed between this new file and the test bench for Exercise 5. Run simulations using this test bench.

Part 2

Draw a diagram of a four-bit binary adder/subtracter (assuming two's complement representation) using the 74LS83 or 74LS283 IC (4-bit binary adder) and any other necessary logic gates. Be sure to use the chip number that matches the IC in your lab kit. A single control line should be used to select circuit operation between addition (Control=0) and subtraction (Control=1). In your schematic/block diagram, show the 74LS83 or 74LS283 IC as a single black box component (search for '74283' in Quartus II).

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Lab Procedure

Part 1

For the circuit from Prelab Part 1, build, test, and demonstrate proper operation to your laboratory instructor or TA.

Part 2

The 74LS283 and 74LS83 ICs are MSI four-bit binary full adders, whose operations are shown below. Inputs A and B and output F are to be interpreted as 4-bit two's complement numbers.

```
                    C0  ←  Carry in (bit 0)
     (  A3  A2  A1  A0  )
 +   (  B3  B2  B1  B0  )
-------------------------
 C4  (  F3  F2  F1  F0  )
```

Build the 4-bit adder/subtracter circuit from Prelab Part 2 with proper arrangements of your input and output signals. Make sure you look at the data sheet for the adder IC (74LS83 *or* 74LS283) and note that the power and ground pins may not be in the usual location, depending on which chip is in your lab kit. By hand, compute the additions and subtractions listed in Tables 6.1 and 6.2, write your results in the tables, and indicate which operations produce an incorrect two's complement result. Verify proper operation of your circuit by adding and subtracting the numbers indicated in Tables 6.1 and 6.2 and comparing with your hand-computed results. You may wish to test more examples, since the instructor or TA may ask you to demonstrate other values. Show that the results are correct by converting A, B, and the results to decimal values. Demonstrate proper circuit operation to your instructor or TA.

Look at the questions for this exercise, and ask your instructor about the meaning of any question(s) you do not understand.

Table 6.1: Four-bit binary addition

$Augend(A) + Addend(B) + CarryIn(C0) \rightarrow\ Sum(F)$ and $CarryOut(C4)$

$A3\ A2\ A1\ A0$	$B3\ B2\ B1\ B0$	$C4$	$F3\ F2\ F1\ F0$	$A + B = F$
$S7\ S6\ S5\ S4$	$S3\ S2\ S1\ S0$	$L4$	$L3\ L2\ L1\ L0$	(Decimal)
1 0 1 0	0 0 1 1			
1 1 1 1	1 1 1 1			
1 1 1 1	0 1 1 0			
0 1 1 1	0 0 1 1			
0 1 0 1	1 0 1 0	0	1 1 1 1	$(-6) + (5) = -1$

Table 6.2: Four-bit binary subtraction

$Minuend(A) - Subtrahend(B)$ implemented as $A + (-B)$

$A3\ A2\ A1\ A0$	$B3\ B2\ B1\ B0$	$C4$	$F3\ F2\ F1\ F0$	$A - B = F$
$S7\ S6\ S5\ S4$	$S3\ S2\ S1\ S0$	$L4$	$L3\ L2\ L1\ L0$	(Decimal)
1 0 1 0	0 0 1 1			
1 1 1 1	1 1 1 1			
1 1 1 1	0 1 1 0			
0 1 1 1	0 0 1 1			
0 1 0 1	1 0 1 0			

Questions

1. In Part 1, you designed one full adder. What would be the simplest way (i.e., fewest modifications) to combine two full adders to make a two-bit carry-ripple adder? (Note that the carry into the least significant bit is always zero.) For your answer, give a block diagram, where a block represents the circuit from Part 1.
2. How can overflow be detected for addition and subtraction of two's complement numbers? Give a Boolean expression for overflow of n-bit numbers. (Hint: Think about how an XOR gate could be used to detect overflow.)
3. Why does the 4-bit adder/subtracter circuit from Part 2 produce incorrect two's complement results, even though the circuit is correct?
4. What happens to the "incorrect" results found in Question 3 if the result register is extended by one bit to use the carry-out bit as the MSB in the register? If this change were implemented, would the "correct" results found in Question 3 still be "correct" results (in five bits)? Justify your answer.
5. Design an additional circuit for a 4-bit adder/subtracter so that the overall circuit will produce a 5-bit result that is always correct even in the case of overflow.

Exercise 6: Binary Addition and Subtraction Circuits

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Part 1	K-maps	2		
	Boolean transformations	4		
	Correct Boolean expressions	2		
	Schematic diagram	4		
	Simulations	4		
Part 2	Schematic diagram	4		

Demo		Point Value	Points Earned	Date
Demo	1-bit full adder	20		
	4-bit adder/-subtracter	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 6: Binary Addition and Subtraction Circuits

Report		Point Value	Points Earned	Comments
Abstract		4		
Design Methodology	K-maps / Boolean transformations	3		
	1-bit FA circuit diagram	2		
	Discussion of add/subtract control	3		
	4-bit adder/-subtracter circuit diagram	2		
Results and Analysis	1-bit full adder Simulation	2		
	4-bit adder/-subtracter result tables	3		
Conclusion		4		
Questions	Q1	4		
	Q2	2		
	Q3	3		
	Q4	2		
	Q5	3		
Writing Composition		3		
Total for prelab, demo, and report		100		

Exercise 7 Sequential Circuit Elements

Objective

The objective of this exercise is to design, implement, and study two basic sequential circuit elements: D latch with active-low enable and rising edge-triggered, master-slave D flip-flop.

Prelab Work

Part 1

Based on Figure 7.1, draw a completely labeled circuit diagram of a D latch with active-low enable in Quartus II. Simulate your design in ModelSim to generate a waveform that verifies it is correct. Make sure to show complete operation by including one case where the D input changes while keeping enable at 0 (active) and one where D changes while keeping enable at 1 (inactive). Complete the diagram presented in Figure 7.3 to show the implementation of two such latches using components from your lab kit.

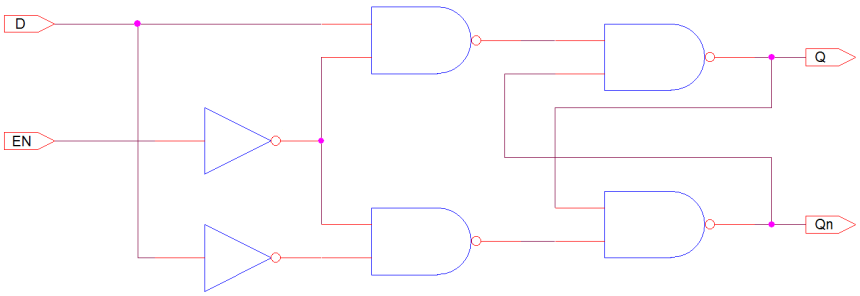


Figure 7.1: Circuit diagram of an active-low enable D latch using NAND gates

Part 2

Based on Figures 7.1 and 7.2, draw a completely labeled circuit diagram of a rising edge-triggered D flip-flop using two D latches in Quartus II. Simulate your design in ModelSim using force statements to generate a waveform that verifies it is correct. Make sure to show complete operation by including one case where D changes between two rising edges of the clock and one where D remains the same across two rising edges of the clock. *Do not allow D to change at the same time as the rising edge of the clock.* See page 94 for more details. Then, download the test bench file for this exercise and look through it and compare what has changed between this new file and the test benches for Exercise 5 and 6. Run simulations using this test bench.

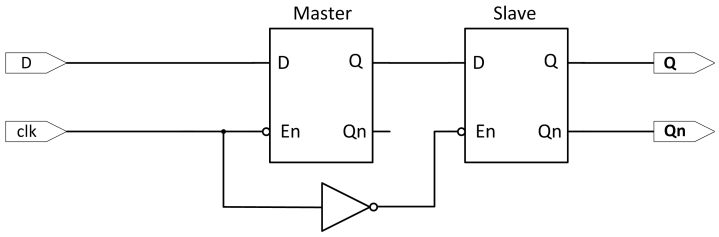


Figure 7.2: Circuit diagram of a master-slave D flip flop made of D latches

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Lab Procedure

Part 1

Complete Table 7.1 to describe the functionality of the D latch with active-low enable. Based on Figure 7.3, build two D latches with active-low enable from Prelab Part 1. Verify their functionality. Connect D1 to switch S1, En1 to switch S2, Q1 to LED L1, and Qn1 to LED L2. Connect D2 to switch S3, En2 to switch S4, Q2 to LED L3, and Qn2 to LED L4. Demonstrate proper operation to your laboratory instructor or TA. Keep this circuit for Part 2.

Table 7.1: D latch with active-low enable

En	D	Q	Qn
0	0		
0	1		
1	X		

Part 2

Complete Table 7.2 to describe the functionality of the D flip-flop. Use the two D latches from Part 1 to build the D flip-flop from Prelab Part 2. Configure the function generator to generate a square wave at 1 Hz with a 50% duty cycle, 5 V peak-to-peak amplitude, and 2.5 V offset. Check this signal on the oscilloscope, then connect it to the clock input of the circuit. Connect one oscilloscope probe to this clock signal and a second probe to Q, the final output of the flip-flop. D should be connected to one of the switches. Check to ensure that both probes and the function generator are properly grounded. Toggle D quickly until several transitions of Q can be seen, then press Run/Stop. Demonstrate your results to your laboratory instructor or TA, and save the results to your USB drive.

Table 7.2: Rising edge-triggered D flip flop

clk	D	Q	Qn
↑	0		
↑	1		
otherwise	X		

Look at the questions for this exercise, and ask your instructor about the meaning of any question(s) you do not understand.

Questions

1. Design a falling edge-triggered D flip-flop using D latches with active-low enable (from Part 1).
2. The 74LS175 IC has four D flip-flops. Find these timing values for a 74LS175 D flip-flop using the MyCourses data sheet.

(a) Maximum time for clear operation
(b) Maximum time for output to change after triggering clock edge
(c) Minimum setup time for D input
(d) Minimum hold time for D input

Extra Credit

The extra credit is intended to explore the workings of a D flip-flop in more detail, as well as measure some of its characteristics. The following procedure requires the exercise to have been finished. The oscilloscope probes should remain connected to clock and Q, and the function generator should still produce the clock.

1. Change the x-axis scale to be 50 µs/div and set the trigger for rising edge of the Q channel.
2. Put the oscilloscope in single shot mode by pressing the Single button located in the top-right corner. Toggle the switch until the waveform appears. If this does not occur, ensure that the voltage in the upper right corner of the display is around 2.5 V. Because this is the trigger voltage, much larger or smaller values will result in the oscilloscope not detecting the edge.
3. Zoom in until a delay between the two signals is clear. Add a delay measurement where clock is the reference and Q is the signal. Repeat for a falling edge on Q. Figure 7.4 shows the required measurement.

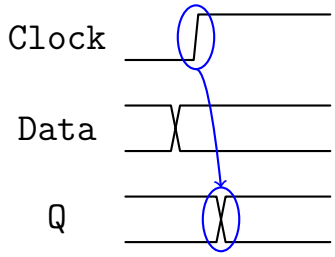


Figure 7.4: Clock to Q delay

4. Capture a screenshot of one of the two cases and include it in the report.
5. Answer the following questions in the report: What was the maximum clock to Q delay measured? Compare this to the answer for Question 2b. What frequency can the flip-flop operate at, assuming no other delays are present?

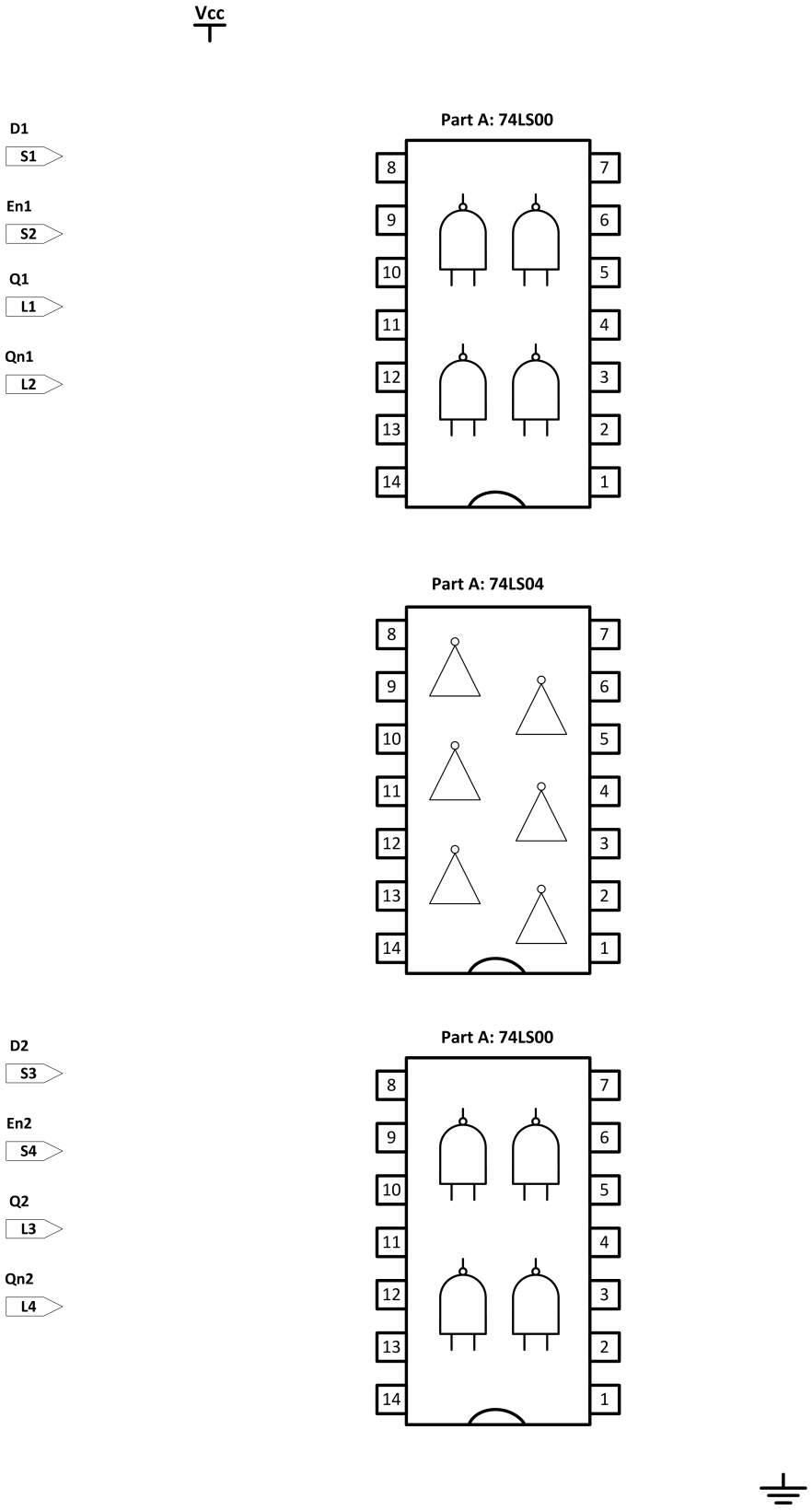


Figure 7.3: Parts placement diagram for two D latches

Exercise 7: Sequential Circuit Elements

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Part 1	Schematic	4		
	Simulations	4		
	Parts placement diagram	4		
Part 2	Schematic	4		
	Simulations	4		

Demo		Point Value	Points Earned	Date
Demo	Latches	20		
	Flip-flop	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 7: Sequential Circuit Elements

Report		Point Value	Points Earned	Comments
Abstract		4		
Design Methodology	D latch function table	3		
	D latch circuit diagram	3		
	D flip-flop function table	3		
	D flip-flop circuit diagram	3		
Results and Analysis	D latch simulation	3		
	D flip-flop simulation	3		
	D flip-flop oscilloscope capture	2		
Conclusion		4		
Questions	Q1	4		
	Q2	4		
Extra Credit	Oscilloscope capture	5		
	Question	5		
Writing Composition		4		
Total for prelab, demo, and report		100		

Exercise 8 Analysis and Simulation of Sequential Circuits

Objective

The objective of this exercise is to analyze and simulate sequential circuit—4-bit shift register. This will yield a better understanding of the design and operation of sequential circuits.

Prelab

1. Analyze the four-bit shift register shown in Figure 8.1, and label the inputs and outputs. Use names given in the list of "input and output signals."
2. Create a state/output table that describes the functionality of the shift register. Note that there is no need to create a large table that lists all possible combinations of 0s and 1s. Think of how to create a table that represents both the load and shift-right functionality.
3. Draw the waveforms of *all input and output signals* for the following sequence of operations.
   - Reset before the first operational clock cycle.
   - Clock Cycle 1: parallel load of the number B_{16} (hex) to the register.
   - Clock Cycles 2-3: shift right two times while serial input is equal to 0.
   - Clock Cycles 4-5: shift right two times while serial input is equal to 1.

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Input and output signals

- (SIN): Serial input
- ($A\ B\ C\ D$): Parallel input (4 bits)
- ($QA\ QB\ QC\ QD$): Parallel output (4 bits), where QA is the most significant bit, and QD is the least significant bit
- (QD): Serial output (from LSB flip-flop, which is QD)
- (CLK): Clock input
- (SL): Control line to switch the mode between loading and shifting right
- (RST): Reset input (asynchronous, active low)

Control signal description

These are the effects of the control signals.

- The register is triggered at the rising edge of the clock.
- The reset signal (RST) is asynchronous and active low. When this signal is low (logic 0), the register is reset (all flip-flops' outputs are 0), independent of the clock. When reset (RST) is high (logic 1), the functionality depends on the control lines.
- If the signal SL is low (0), the register is shifting right every clock period. During the shift-right mode, the value of SIN will be appended as the MSB of the new number. If SL is high (1), the register performs a parallel load, sending (A B C D) to (QA QB QC QD). These operations happen at the rising edge of the clock.

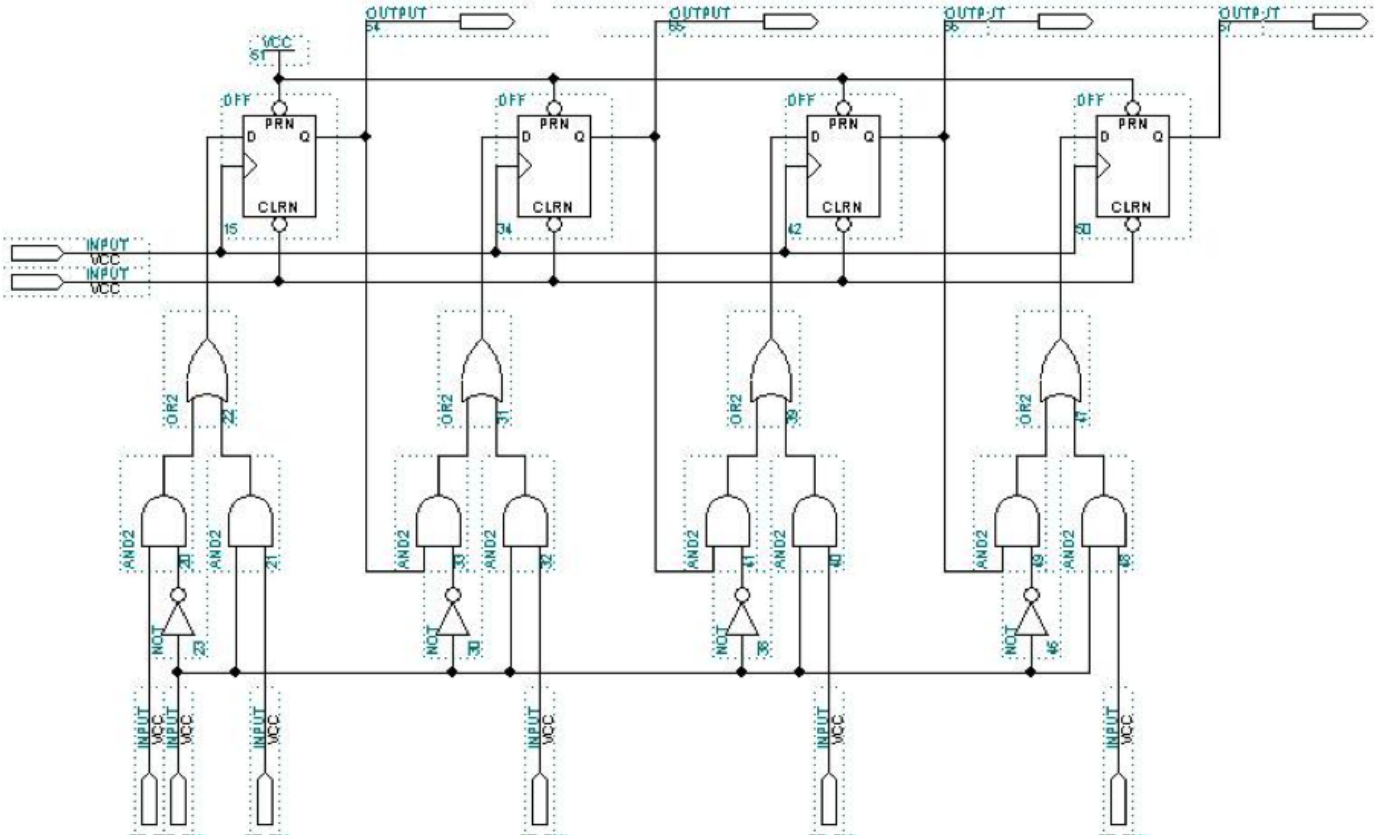


Figure 8.1: A 4-bit shift register, (needs labels for all inputs/outputs)

Lab Procedure

Create a schematic for the four-bit shift-register using Altera Quartus II CAD tool. Using ModelSim, verify your design to make sure it works correctly. Download the incomplete test bench from MyCourses and modify it as explained in the comments and use it to simulate your design. When completed, the test bench should perform the same operations listed in the prelab. You may consult the lab instructor or the TA if unexpected results appear on your simulation waveform. Also complete Table 8.1 based on the results you obtain.

After the circuit is correctly compiled and simulated, save the schematic diagram and the simulation results and include them in your report. Determine the minimum clock period and maximum clock frequency for your design using the following procedure. This does not work on personal computers.

1. With the schematic open in Quartus, find the Tasks Pane.
2. Locate the "Time Quest Timing Analysis" section and expand it. Double click on "View Report". This may take a few minutes.
3. Find the Table of Contents Pane and select the TimeQuest Timing Analyzer. Expand it to find the Multicorner Datasheet Report Summary.
4. This will have several tables. The relevant ones are the Setup Times and Clock to Output Times. Choose the largest value from each table.

Plug the values found from Quartus into the following equation:

$$T_{min} = t_{co} + t_{pd} + t_s$$

T_{min} is the minimum clock period that the circuit can operate correctly at. t_{co} is the Clock to Output time, and similarly, t_s is the Setup time. Finally, t_{pd} is the propagation delay through the logic, which is assumed to be 0 ns. The result is also in nanoseconds. Show your results to the lab instructor to obtain signatures.

Create a new schematic of an equivalent 4 bit register in which you will replace the multiplexor logic elements implemented as AND-OR network with multiplexors implemented using tri-state buffers as shown in Figure 8.2. Using ModelSim, verify your design to make sure it works correctly. After the circuit is correctly compiled and simulated, save the new schematic diagram and the simulation results and include them in your report.

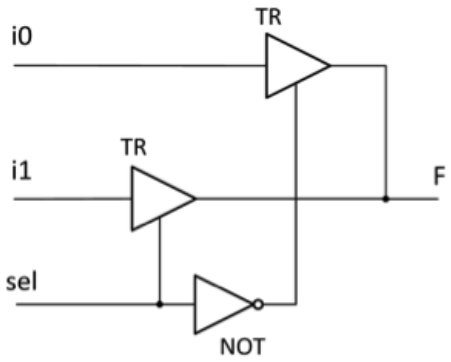


Figure 8.2: A 2:1 multiplexer built using two tri-state buffers and one inverter

Table 8.1: Four-bit shift register operation

Time	Register Contents		
	Binary ($QA\ QB\ QC\ QD$)	Hex	Decimal
After Reset			
Clock Cycle 1			
Clock Cycle 2			
Clock Cycle 3			
Clock Cycle 4			
Clock Cycle 5			

Lab Report

Discuss the functionality of shift register circuit. Give the minimum clock period and maximum clock frequency; discuss the timing delays used to determine them.

Questions

1. What kind of arithmetic operation occurs when a value is shifted right one bit and a zero is introduced into the most significant bit position?
2. What kind of arithmetic operation occurs when a value is shifted left one bit and a zero is introduced into the least significant bit position?
3. Assume the flip-flops are sensitive to the rising edge of the clock. Explain why it is not a good idea to change the input signals at exactly the same time as the rising edge of the clock.

Exercise 8: Analysis and Simulation of Sequential Circuits

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Prelab	Part 1	5		
	Part 2	5		
	Part 3	10		

Demo		Point Value	Points Earned	Date
Demo	Part 1 Simulation	15		
	Part 2 Simulation	15		
	Timing Analysis	10		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 8: Analysis and Simulation of Sequential Circuits

Report		Point Value	Points Earned	Comments
Abstract		3		
Design Methodology	Function table	5		
	Circuit diagram	5		
	Circuit diagram with multiplexors	4		
Results and Analysis	Simulation results	6		
	Timing analysis	3		
Conclusion		4		
Questions	Q1	2		
	Q2	2		
	Q3	2		
Writing Composition		4		
Total for prelab, demo, and report		100		

Exercise 9 Design and Simulation of a Moore State Machine

Objective

The objective of this exercise is to design and simulate a Moore state machine based on a functional description. Techniques learned from class are to be applied to design the sequential circuit: developing a state transition diagram, a state table, the various equations, and the circuit diagram.

Functional Description

The state machine is a sequence detector. It has two data inputs (A and B), a single data output (Z), a rising-edge triggered clock (CLK), and an asynchronous active-low reset signal (RST). After reset, the sequence detector tracks the data sequence of inputs A and B at each clock cycle. If $A = 1$ and $B = 1$ in one clock cycle, $A = 1$ and $B = 0$ in the next clock cycle, and then $A = 0$ and $B = 0$ in the next clock cycle, it produces an output of $Z = 1$. Afterward, if the inputs remain unchanged, Z remains 1; otherwise Z changes to 0, and the detector tracks the inputs again for the next occurrence of the sequence $(A, B) = (1, 1), (1, 0)$, and then $(0, 0)$. For all other cases, $Z = 0$.

Prelab Work

Utilize the sequential circuit design techniques discussed in class to design the sequence detector using rising edge-triggered flip-flops. The design should include a state diagram, a state table, the various equations, and the circuit diagram—(manually drawn will suffice for prelab). All work should be presented in an organized and clean fashion. Use the following names to represent the states of your state machine: Idle, Got11, Got10, GotAll and the following encoding of the states:

Table 9.1: States and encodings for the Moore state machine

State name	State	
	$Q1$	$Q0$
Idle	0	0
Got11	0	1
Got10	1	1
GotAll	1	0

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Lab Procedure

Use Quartus II to build the sequence detector. Test your sequence detector by following the operation sequence below in a ModelSim simulation. Download the incomplete test bench from MyCourses and modify it as explained in the comments and use it to simulate your design. Note: RST, A, and B should not change at exactly the same time the clock cycle changes, (i.e., at the rising edge of CLK). Two separate simulations should be performed, using the following two sequences:

1. $(A, B) = (1, 1), (1, 0), (0, 0), (1, 0), (0, 0), (1, 1), (1, 0), (0, 0), (0, 0)$
2. A different test sequence of your own design that also verifies the functionality of your circuit.

Be sure the second test exercises state transitions that the first did not.

Verify the results by comparing them to the problem description. After verifying your results, save your circuit diagram and both simulation results. From the Quartus timing analysis generated during compilation, determine the minimum clock period and maximum clock frequency for your design. Show your saved output to the lab instructor for discussion and signatures.

Questions

1. Repeat the synthesis process of your state machine for the following state encoding style:

Table 9.2: Alternative states and encodings for the Moore state machine

State name	State	
	$Q1$	$Q0$
Idle	0	0
Got11	0	1
Got10	1	0
GotAll	1	1

2. How many different ways are there to encode a state machine with four states that is implemented using two D-flip flops? Note that the answer should be a number, not "one-hot" or "gray".
3. What is "one-hot" encoding style?

Exercise 9: Design and Simulation of a Moore State Machine

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Prelab	State diagram	5		
	State table	5		
	Equations	5		
	Circuit diagram	5		

Demo		Point Value	Points Earned	Date
Demo	Part 1 simulation	15		
	Part 2 simulation	15		
	Timing analysis	10		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 9: Design and Simulation of a Moore State Machine

Report		Point Value	Points Earned	Comments
Abstract		4		
Design Methodology	State diagram	3		
	State transition table	3		
	Equations	3		
	Circuit diagram	3		
Results and Analysis	Test sequence	3		
	Simulation results	3		
	Timing results	2		
Questions	Q1	4		
	Q2	2		
	Q3	2		
Conclusion		4		
Writing Composition		4		
Total for prelab, demo, and report		100		

Exercise 10 ModelSim and VHDL Tutorial

Objective

The objective of this exercise is to introduce essential VHDL statements, modeling approaches, and using test benches in functional verification.

Prelab Work

Do the step-by-step tutorial that is given in Appendix E and the following:

(a) Before copying the test bench code and simulating both designs, draw a timing diagram showing the expected inputs waveforms the test bench will generate.

(b) After copying and filling in the missing code, simulate the data flow adder design and draw its high level diagram. Draw logic gates to represent the data flow VHDL statements. Label the drawing with the correct input, output and internal signal names.

(c) After copying and filling in the missing code, simulate the structural adder design and draw its high level diagram. Include instance labels (ex. `xor3_instance1)`, component names (ex. `xor3`), and I/O signal names (ex. `A`, `B`, `C`, `Y`) on all instantiated logic gates. Label the high level drawing with the correct input, output and internal signal names.

XX

Lab Procedure

Using knowledge gained during the tutorial, create a third full adder design.

(a) Design a half adder (`hadder`) entity and architecture using the data flow style. Save the design in hadder.vhd.

(b) Design a two input or gate (`or_gate`) entity and architecture using the data flow style. Save the design in or_gate.vhd.

(c) Add a third architecture to the full adder using the structural style. As shown in the full adder block diagram, Figure 10.1, instantiate two `hadder` circuits and one `or_gate` circuit.

(d) Add the new full adder design to the test bench, modifying the test bench as necessary, and verify correct operation.

Demonstrate your results to the lab instructor.

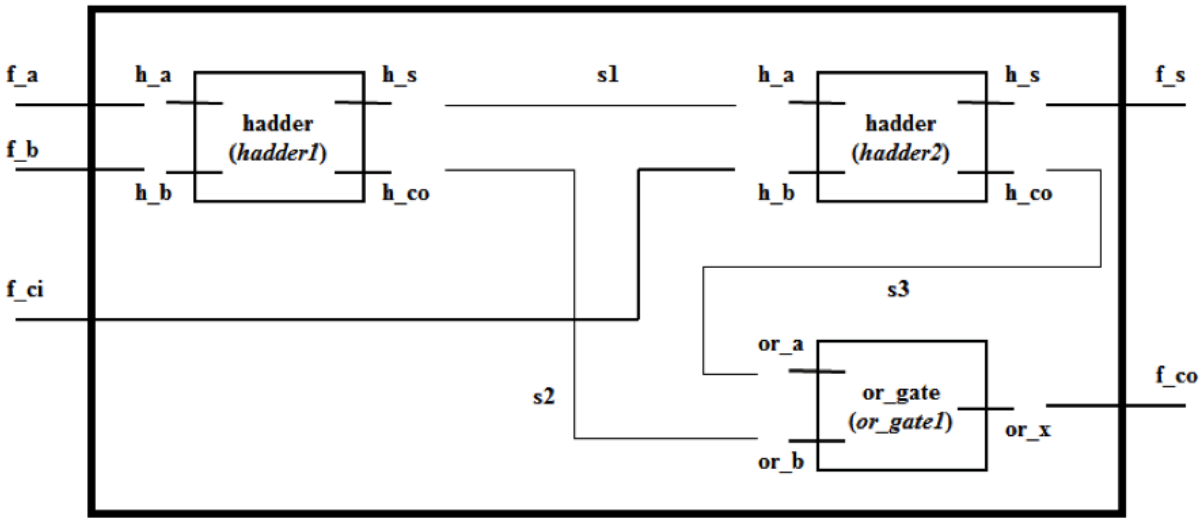


Figure 10.1: Full adder composed of half adders

Exercise 10: ModelSim and VHDL Tutorial

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Prelab	Dataflow	10		
	Structural	10		
	Testbench	10		
	Drawings	10		

Demo	Point Value	Points Earned	Date
Demo	60		

Total	100		

A report is not required for this exercise.

Exercise 11 Modeling of Combinational Circuits Using Concurrent and Sequential Statements

Objective

The objective of this exercise is to learn how to write structural, data flow and behavioral models in VHDL. Three models of the 74LS153 circuit (two 4:1 multiplexers) are constructed and simulated in ModelSim to verify their correctness.

Prelab Work

Part 1

Based on the data sheet of 74LS153 circuit and Table 11.1, write entity declaration of a dual 4-line to 1-line data multiplexers circuit.

Table 11.1: Inputs and outputs of a 74LS153 circuit

Inputs		Outputs	
G1, G2, A, B	std_logic	Y1, Y2	std_logic
C1, C2	std_logic_vector(3 downto 0)		

Part 2

Write data flow architecture (`df`) to model this circuit using two selected signal assignment statements, (one for each multiplexer). This model shall not include any propagation delays. (Assume that, in the data sheet, "L" is equivalent to logic "0", "H" to logic "1," and "X" denotes any value).

```
-- architecture df (data flow)
architecture df of DM74LS153 is
--Declare internal control signals for each of the multiplexers

begin
    -- model of the first multiplexer
    sControl1 <= _________________;
    with ______ select
        Y1 <= ______ when "000",
        ...

    -- model of the second multiplexer
    ...
    with ______ select
        Y2 <= ______ when "000",
        ...

end architecture df;
```

Part 3

For the same entity, write a behavioral architecture (`behv`) that will utilize two case statements and two processes, (one set for each multiplexer). This model shall include the worst case delay of 22 ns from all inputs to all output.

Part 4

For the same entity, write a structural architecture (`struct`), that will be a three-level hierarchical model. First write models (separate entity-architecture pairs) of an inverter, 4-input AND, and 4-input OR gates. In your models, include 4 ns as a propagation delay of the inverter and 7 ns as a propagation delay of the AND and OR gates. Use these as components to structurally model a single multiplexer, (another separate entity-architecture pair). Use the model of a single multiplexer to construct the struct architecture of the 74LS153 circuit.

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

Lab Procedure

Prepare a single VHDL test bench that will allow simulation of **all three architectures in sync**. Use ModelSim simulator to verify correctness of your models. Make your test vectors (inputs) to change every 100 ns.

Your tests will need to consider the following cases (for each of the multiplexers):

1. Keeping `G` (Strobe) input at active value (logic '0') and constant value on select inputs `A` and `B` verify the output by changing the data inputs in the way presented in Table 11.2.

Table 11.2: Test set 1

G	B	A	C3	C2	C1	C0
0	0	0	1	1	1	0
0	0	0	0	0	0	1
0	0	1	1	1	0	1
0	0	1	0	0	1	0
0	1	0	1	0	1	1
0	1	0	0	1	0	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0

An example output for the 3rd and 4th tests listed in Table 11.2 is shown in Figure 11.1.

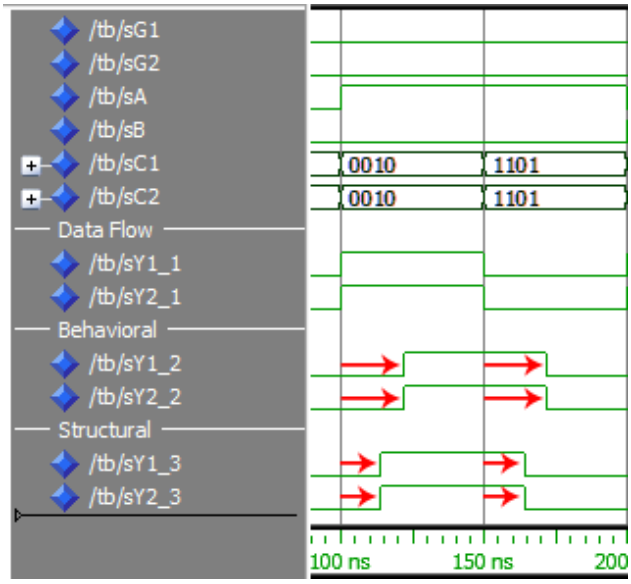


Figure 11.1: Sample test output.

Take note that the delays coded into the behavioral and structural models are highlighted on this diagram with red arrows. Also note how the differing bit is being selected.

2. Keeping G (Strobe) input at active value (logic '0') and constant value on data inputs verify the output by changing select inputs A and B in the way presented in Table 11.3.

Table 11.3: Test set 2

G	B	A	C3	C2	C1	C0
0	0	0	1	0	1	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	1	0	1	0
0	0	0	0	1	0	1
0	0	1	0	1	0	1
0	1	0	0	1	0	1
0	1	1	0	1	0	1

3. Repeat either test set 1 or 2 with the G input receiving an inactive value (logic '1').

Save separate simulation results (waveforms) for test cases 1, 2 and 3. Show your output to the TAs or instructor for discussion and signatures.

Lab Report

Code should not be included in the lab report. However, a submission of your well commented VHDL source code to MyCourses is required. Your submission should include VHDL files, not zipped, as well as the report.

Questions

Explain the meaning of delta delays and concurrent statements in the context of VHDL, using the code below as an example. Draw appropriate waveforms neatly using a ruler and attach them to the hard copy of your report. Hand drawn is acceptable for this diagram only. Show how changes on inputs (signals) trigger the execution of other statements–make delta cycles visible. Draw the schematic of corresponding circuit in Quartus II. Be sure to label all signals, internal and external.

```
entity delta is
    port(a,b,c,d : in  bit;
    z            : out bit);
end delta;
architecture df of delta is
    signal u,v,w,x,y : bit;
begin
    z <= not y;
    y <= w or x;
    x <= u or v;
    w <= u and v;
    v <= c or d;
    u <= a and b;
end df;
```

ns	+delta	a	b	c	d	u	v	w	x	y	z
0	+0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	+1	0	0	0	0	0	0	0	0	0	**1**
100	+0	**1**	0	0	0	0	0	0	0	0	1
200	+0	1	**1**	0	0	0	0	0	0	0	1
200	+1	1	1	0	0	**1**	0	0	0	0	1
200	+2	1	1	0	0	1	0	0	**1**	0	1
200	+3	1	1	0	0	1	0	0	1	**1**	1
200	+4	1	1	0	0	1	0	0	1	1	**0**
300	+0	**0**	1	0	0	1	0	0	1	1	0
300	+1	0	1	0	0	**0**	0	0	1	1	0
300	+2	0	1	0	0	0	0	0	**0**	1	0
300	+3	0	1	0	0	0	0	0	0	**0**	0
300	+4	0	1	0	0	0	0	0	0	0	**1**
400	+0	0	**0**	0	0	0	0	0	0	0	1

Exercise 11: Modeling of Combinational Circuits Using Concurrent and Sequential Statements

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Prelab	Entity declaration	5		
	Data flow architecture	5		
	Behavioral architecture	5		
	Structural architecture	5		

Demo		Point Value	Points Earned	Date
Demo	Test case 1	15		
	Test case 2	15		
	Test case 3	10		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 11: Modeling of Combinational Circuits Using Concurrent and Sequential Statements

Report		Point Value	Points Earned	Comments
Abstract		3		
Design Methodology	Discussion of circuit functionality	5		
Results and Analysis	Test methodology	3		
	Simluation results	10		
Conclusion		3		
Questions		6		
Writing Composition		3		
Source code	Comments	3		
	Style	4		
Total for prelab, demo, and report		100		

Exercise 12 Serial Adder with Control Unit

Objective

The objective of this exercise is to design, model, and test (through ModelSim simulation) a serial adder circuit as an example of a complete digital system with data path and a control unit. Testing procedure includes self-checking test bench with array of records to store test vectors. The course instructor will discuss this design problem in class to provide a starting point for the exercise.

Part 1: Serial Adder

Prelab Work

Write a behavioral model (22-ns delay) of a 4-bit bidirectional universal shift register based on the function table of the DM74LS194A component as presented in Table 12.1. In your model, use one process, one if-elsif, and one case statements. All inputs and outputs need to be declared as single bit ports of `std_logic` type.

Table 12.1: Functionality of the DM74LS194A circuit

Inputs										Outputs			
Clear	Mode		Clock	Serial		Parallel				Q_A	Q_B	Q_C	Q_D
	$S1$	$S0$		SL	SR	A	B	C	D				
0	X	X	X	X	X	X	X	X	X	0	0	0	0
1	X	X	0	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
1	1	1	↑	X	X	a	b	c	d	a	b	c	d
1	0	1	↑	X	sr	X	X	X	X	sr	Q_{An}	Q_{Bn}	Q_{Cn}
1	1	0	↑	sl	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	sl
1	0	0	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

Write models of a 1-bit full adder (8-ns delay), rising-edge sensitive D flip-flop with `clk`, with an active low `clear`, and active high `enable` inputs (6-ns delay), 2-input AND gate (4-ns delay), and an inverter (2-ns delay).

Download the shift register test bench from MyCourses (`tb_DM74LS194A.vhd`). Fill in the missing assert statements and ensure that all signal names match those in your shift register entity, as well as the entity name itself. If you have not done so already, create a new ModelSim project and add both the test bench and the shift register. Use this to ensure that the shift register is working properly.

Use ModelSim simulator to verify correctness of your other models. Force commands can be used for this purpose.

Complete Figure 12.1, showing the connections between blocks of serial adder.

Lab Procedure

Use your components to structurally model the data path of a serial adder–two instances of a 4-bit shift register (`regA`, `regB`), one D flip-flop, one 1-bit full adder, one 2-input AND gate, and one inverter.

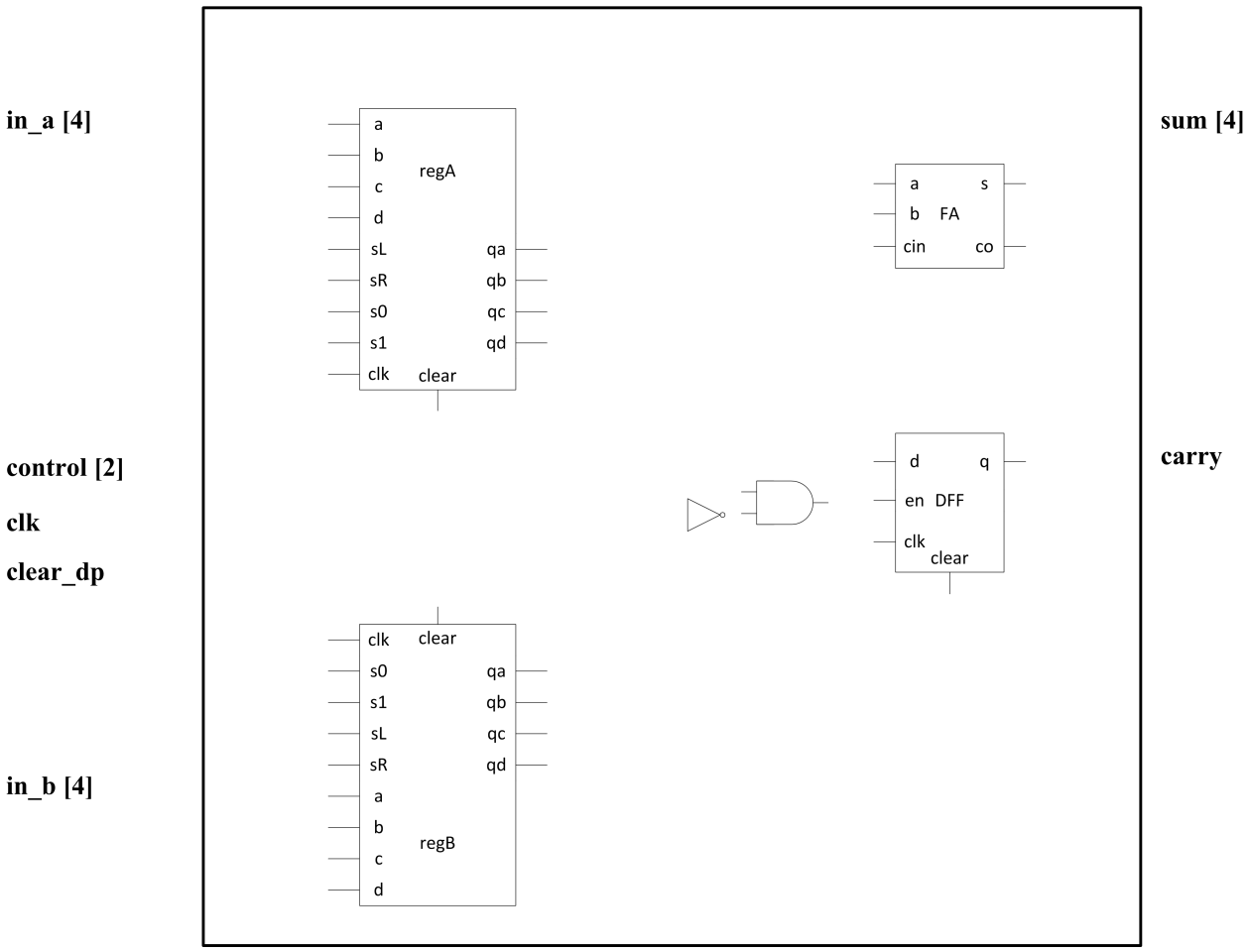


Figure 12.1: Serial Adder Block Diagram

Functional Details

On the rising edge of the clk input, when mode is "11," the registers are loaded with `in_a` (`regA`) and `in_b` (`regB`).

On the rising edges of the `clk` input, when mode is "01," the content of the registers is shifted into the 1-bit full adder. Recall that when adding in binary, start with the Least Significant Bit (Rightmost bit). The full adder's sum bit is shifted into the register holding `in_a`. After 4 rising edges on the `clk` while the mode remains "01," the `regA` register contains the value that is equal to `in_a` + `in_b`, and carry should be the output of the DFF which corresponds to the carry out bit.

Testing

The test bench will apply inputs (`in_a`, `in_b`, `control`, `clk` and `clear_dp`) and verify the outputs, using **assert statements** to flag errors. A constant array of records will be used to store the values for `in_a`, `in_b`, expected sum and expected carry. The values to be used for `in_a` and `in_b` are shown in Table 12.3.

Fill out the rest of the table and include it in your report (using "hex" notation). Your test bench checks the result, and if any error occurs, an assert statement should give the appropriate message.

Table 12.2: Implementation details for the entity of the serial adder data path

Inputs		Outputs	
`in_a`	`std_logic_vector`		
`in_b`	`std_logic_vector`	`sum`	`std_logic_vector`
`control`	`std_logic_vector`	`carry`	`std_logic`
`clk`	`std_logic`		
`clear_dp`	`std_logic`		

Table 12.3: Test cases for the serial adder

in_a	in_b	sum	carry
X"0"	X"4"		
X"C"	X"E"		
X"8"	X"A"		
X"F"	X"F"		
X"F"	X"1"		
X"A"	X"5"	**X"2"**	**'0'**
X"8"	X"7"		

Timing is a crucial thing in your test bench. Remember to reset the circuit before you start a new addition, and explain why it is important to do so. Do not change the result $A+5$ (see the 6th row of Table 12.3); it is intentionally wrong to test the response of your test bench. Save the simulation results (waveforms) and show them to the lab instructor or TAs for discussion and signatures.

Hint

Since the full adder is not clocked by itself, we use the DFF to align the output to the clock. When reporting the carry out, be sure to use the output of the DFF. In addition, if you do not recall the structure of a full adder, be sure to review Lab 6 and Lab 10.

Part 2: Control Unit

Prelab Work

Write a behavioral model (10-ns delay) of the control unit (state machine) presented in Figure 12.2. Use enumeration type to define states (using the same names as in the diagram). Use ModelSim simulator to verify correctness of your model. Force commands can be used for this purpose. Save the simulation results.

If you do not know how to complete the prelab work, it is your responsibility to consult with a course instructor or TA well before your scheduled lab period so that you have your prelab work completed and ready for grading at the beginning of your lab session.

The output, control_output, is a concatenated vector of signals `ready`,`clear_dp`, and `mode` respec-

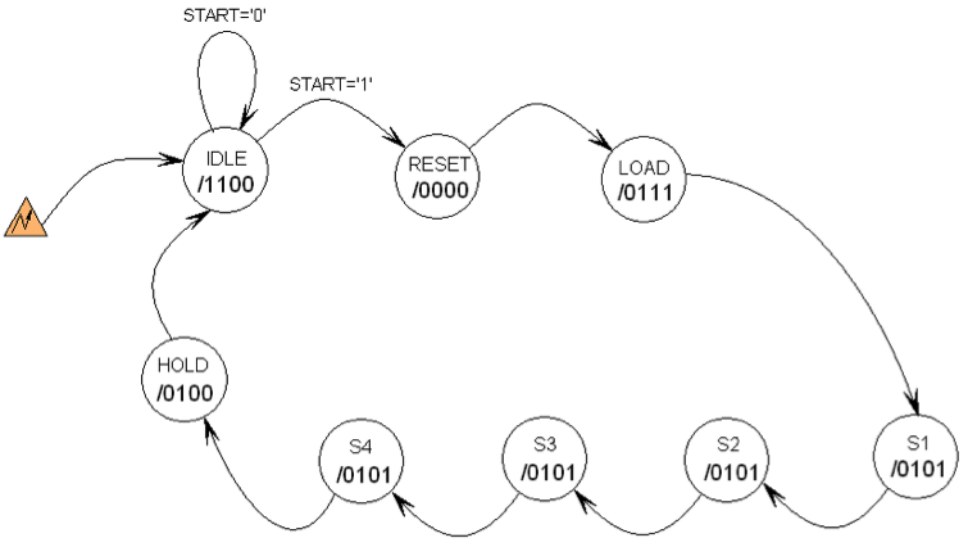


Figure 12.2: State machine diagram

Table 12.4: Implementation details for the entity of the serial adder control unit

Inputs		Outputs	
`start`	`std_logic`		
`clk`	`std_logic`	`control_output`	`std_logic_vector`
`clear_sm`	`std_logic`		

tively. It must done in this order for the state machine to work.

Lab Procedure

Connect your data path and control unit (state machine) to create a complete serial adder design. Table 12.5 shows the input and output signals of the final design. Note that this design should be in a new entity. Modify the test bench from Part 1 of this exercise to test the new complete design. Output signal `ready` should be used to recognize when the final result of the addition is ready for verification. Show your simulation results to the instructor or TAs.

Table 12.5: Implementation details for the entity of the complete serial adder

Inputs		Outputs	
`clk`	`std_logic`	`ready`	`std_logic`
`start`	`std_logic`	`cout`	`std_logic`
`clear_sm`	`std_logic`	`sum`	`std_logic_vector`
`inA, inB`	`std_logic_vector`		

Lab Report

Submit your well commented and organized source code to MyCourses. There is no formal report expected for this exercise.

Exercise 12: Serial Adder with Control Unit

Student's Name:_____ Section:_____

Prelab		Point Value	Points Earned	Comments
Part 1	Shift Register	4		
	D flip-flop	3		
	Adder	2		
	AND Gate	1		
	Inverter	1		
	Test bench	5		
	Force files	4		
Part 2	State Machine	5		
	Force file	5		

Demo		Point Value	Points Earned	Date
Demo	Part 1	40		
	Part 2	20		

To receive any grading credit students must earn points for both the demonstration and the report.

Exercise 12: Serial Adder with Control Unit

Report		Point Value	Points Earned	Comments
Source Code	Comments	3		
	Style	7		
Total for prelab, demo, and report		100		

Appendix A Resistors

Figure A.1 and Table A.1 show the resistor color code, and Table A.2 lists the nominal resistor values readily available in lab.

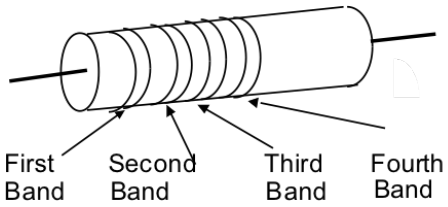


Figure A.1: The bands on a resistor

Table A.1: Resistor color codes

Bands 1-3 Color	Bands 1-2 Values	Band 3 Multiplier	Band 4 Color	Band 4 Tolerance
Black	0	1	None	±20%
Brown	1	10	Silver	±10%
Red	2	100	Gold	±5%
Orange	3	1,000		
Yellow	4	10,000		
Green	5	100,000		
Blue	6	1,000,000		
Violet	7			
Grey	8			
White	9			

Table A.2: Resistors Available in Lab Bins (Values in Ohms)

10	100	1.0K	10K	100K	1.0M
12	120	1.2K	12K	120K	4.7M
15	150	1.5K	15K	150K	10M
18	180	1.8K	18K	180K	
22	220	2.2K	22K	220K	
27	270	2.7K	27K	270K	741
33	330	3.3K	33K	330K	1N4004
39	390	3.9K	39K	390K	1N4148
47	470	4.7K	47K	470K	1N4735
56	560	5.6K	56K	560K	1N3904
68	680	6.8K	68K	680K	1N3906
82	820	8.2K	82K	820K	2N5459

Appendix B TTL Data Sheet Information

TTL data sheets are available on MyCourses. Select your lab section and then "content". The data sheets are in the "Reference IC data sheets" module.

Table B.1 gives the data sheet of a 7400 IC device, which contains four 2-input NAND gates in a single 14-pin package. The parameters to note from any data sheet are the maximum ratings. The absolute maximum V_{cc} of 7 V warns you that, if you have a variable power supply, you should adjust the voltage to +5 V before connecting it to any TTL devices. To make sure that you do not exceed the maximum input rating of 5.5 V, do not forget to adjust the level of any input source, such as a pulse generator, before connecting it to the input pin of a TTL device. Fanout is the maximum number of 7400 series gate inputs that can be connected to a single gate output without preventing the output from reaching valid high and low logic levels.

Under operating conditions, the ranges of supply voltages and ambient temperature over which the manufacturer guarantees the performance of the circuit are given. The 5400 series is functionally the same as the 7400 series, but it is guaranteed to operate over a wider range of supply voltage and ambient temperatures.

The electrical characteristics give the specific voltage levels and currents you should try to fix in your mind for reference. Then when you are troubleshooting, you will be able to recognize that an output is overloaded and is not reaching a legal level. Most manufacturers' data sheets use the positive logic convention. Hence logic "1" refers to the physical HIGH state, and logic "0" refers to the physical LOW state is referred to as logic "0."

The minimum logic HIGH input voltage ($V_{IH,MIN}$) of 2.0 V means the manufacturer guarantees that the gate will recognize an input voltage of 2.0 V or more as logic HIGH and will use HIGH in computing the output state of the gate. The maximum logic LOW input voltage ($V_{IH,MAX}$) of 0.8 V is recognized by the gate as a logic LOW. On the output of the gate, the given minimum logic HIGH output voltage ($V_{OH,MIN}$) of 2.4 V guarantees that an output HIGH will always be equal to or greater than 2.4 V. Typically an output HIGH will be 3 V or greater.

Note that the output is 0.4 V higher than the minimum voltage required for an input high on the following gate (Figure B.1). This 0.4-V difference is known as a "noise margin." It assures that a small noise transient on a connecting line cannot change the state of the next gate. A transient has to have greater than a 400-mV peak before it can swing the input of the next gate below the 2.0-V input high minimum. A similar noise margin of 400 mV exists between the minimum logic LOW output voltage ($V_{OL,MAX}$) of 0.4 V and the maximum logic LOW input voltage ($V_{IL,MAX}$) of 0.8 V (see Figure B.1). As is observed in a lab exercise, excessive loading or incorrect input signals can eliminate one or both of these noise margins and give erratic operation.

Continuing down the column of electrical characteristics in Table B.1, observe that an output must be able to source up to 40 µA of current while still producing a proper input logic HIGH of 2.4 V, but an output must be able to sink up to a maximum of 1.6 mA while still producing logic a proper LOW input. A look in Table B.1, at the conditions following the logic HIGH output voltage and the logic LOW output voltage, shows that outputs are designed to supply 400 µA in the high state and sink 16 mA in the low state. There is enough source and sink current for as many as 10 standard TTL inputs, or a fanout of 10 as mentioned previously. If you connect more than 10 gates to an output, the saturation voltage of the current sinking transistor in the gate will increase above the 0.4-V maximum specified by the manufacturer. Increasing this voltage decreases the noise margin for all the connected inputs and makes them more susceptible to noise transients.

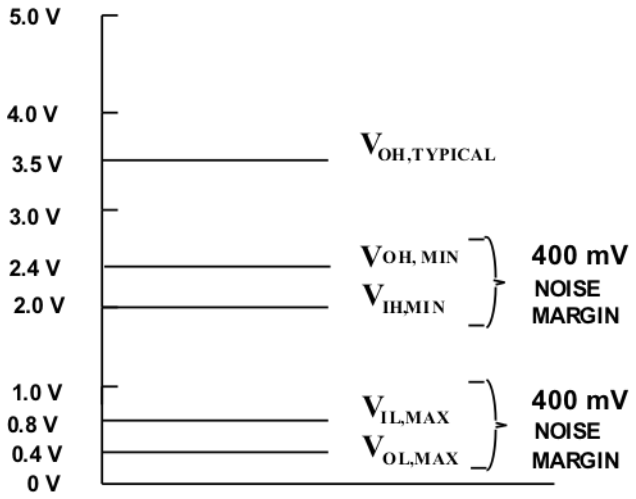


Figure B.1: TTL voltage levels

The next parameter to consider is the supply current per gate. Notice that a logic LOW output requires from 3.0 to 5.1 mA, while a logic HIGH output requires only 1.0 to 1.8 mA. When the gate switches states, the change in required supply current is another cause of transients on the Vcc line. When you are initially estimating the power requirements for a circuit, use the maximum supply current value for each IC and double it. This approach gives a margin of safety and makes power available for additional circuitry you may need.

The final parameter is propagation delay, which is the amount of time it takes an input pulse to change the output. It is measured from the 50 percent point of the corresponding output pulse edge, (see Figure B.2). Loading on the output during the test used to determine the propagation delay is the capacitive and resistive equivalent of 10 standard TTL gate inputs. Notice that the propagation delay time tPD is longer for the output to go from LOW to HIGH than from HIGH to LOW.

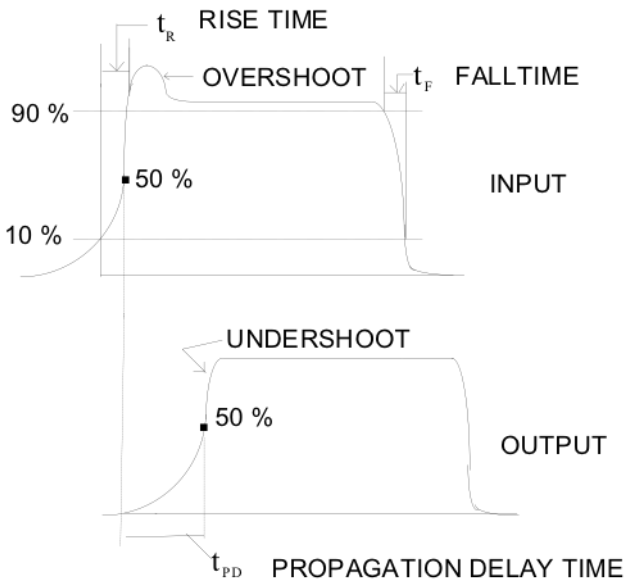


Figure B.2: Pulse parameters

Points to Remember When Working with TTL gates:

1. Tie unused inputs directly to ground or to V_{cc} with a 1-kΩ resistor.
2. The outputs of TTL gates should not be connected together. Exceptions to this rule are open collector and tri-state TTL.
3. Maximum V_{cc} is 7 V.
4. The maximum input signal is 5.5 V at V_{cc} of 5 V.
5. There is a maximum fanout of 10 gates within each TTL family.
6. Install a 0.1-µF bypass capacitor between V_{cc} and GND next to each IC.
7. Open collector leads should not be greater than 12 to 14 in. for standard TTL and 3 to 4 in. for Schottky TTL.

Table B.1: TTL Family Characteristics

DM5400/DM7400, DM5410/DM7410, DM7420 Data Sheet

Absolute Maximum Ratings	
Supply Voltage (V_{cc})	7.0 V
Input Voltage (V_{IN})	5.5 V
Storage Temperature Range	-65 °C to +150 °C
Fanout	10
Lead Temperature (soldering 10s)	300 °C

Operating Conditions	MIN	MAX	UNITS
Supply Voltage (V_{cc})			
DM54XX	4.5	5.5	V
DM74XX	4.75	5.25	V
Temperature (T_A)			
DM54XX	-55	+125	°C
DM74XX	0	+70	°C

Electrical Characteristics (Note 1)

Parameters	Conditions	MIN	TYP	MAX	UNITS
Input Diode Clamp Voltage	V_{cc} = 5.0 V, T_A = 25 °C, I_{IN} = -12 mA			-1.5	V
Logic "1" Input Voltage	V_{cc} = Min	2.0			V
Logic "0" Input Voltage	V_{cc} = Min			0.8	V
Logic "1" OutputVoltage	V_{cc} = Min, V_{IN} = 0.8 V, I_{OUT} = -400 mA	2.4			V
Logic "0" Output Voltage	V_{cc} =Min, V_{IN} = 2.0 V, I_{OUT} = 16 mA			0.04	V
Logic "1" Input Current	V_{cc} =Max, V_{IN} = 2.4 V			40	μA
Logic "1" Input Current	V_{cc} =Max, V_{IN} = 5.5 V			1	mA
Logic "0" Input Current	V_{cc} =Max, V_{IN} = 0.4 V			-1.6	mA
Output Short Circuit Current (Note 2)	V_{cc} =Max, V_{IN} = 0 V, V_{OUT} = 0 V				
	DM74XX	-20		-55	mA
	DM54XX	-18		-57	mA
Supply Current Logic"0" (Note 3	V_{cc} =Max, V_{IN} = 5.0 V		3	5.1	mA
Supply Current Logic"1" (Note 3)	V_{cc} =Max, V_{IN} = 0 V		1	1.8	mA
Propagation Delay Time to Logic"0", t_{PD0}	V_{cc} = 5.0 V, T_A = 25 °C, C = 50 pF		8	15	ns
Propagation Delay Time to Logic"1", t_{PD1}	V_{cc} = 5.0 V, T_A = 25 °C, C = 50 pF		13	25	ns

Note 1: Unless otherwise specified, min/max limits apply across the -55 °C to +125 °C temperature for the DM54XX and across the 0 °C to 70 °C range for the DM74XX. All typical values are given for V_{cc} = 5.0 V and T_A = 25 °C.

Note 2: Not more than 1 output should be shorted at a time.

Note 3: Each gate.

Source: National Semiconductor

Appendix C General-Purpose Breadboard Instructions

There are a number of socket strips on the logic breadboards for inserting integrated circuits (ICs) and other components, such as resistors, power supply leads, lamps, LEDs, and switches. The contacts provided on the prototyping breadboards are of two kinds: (1) bus strips, which distribute power around the board and provide easy access to it from any location on the board and (2) quick test sockets, which allow easy insertion and removal of standard DIPs (Dual-In-Line Packages—a standard packaging of ICs) and other components.

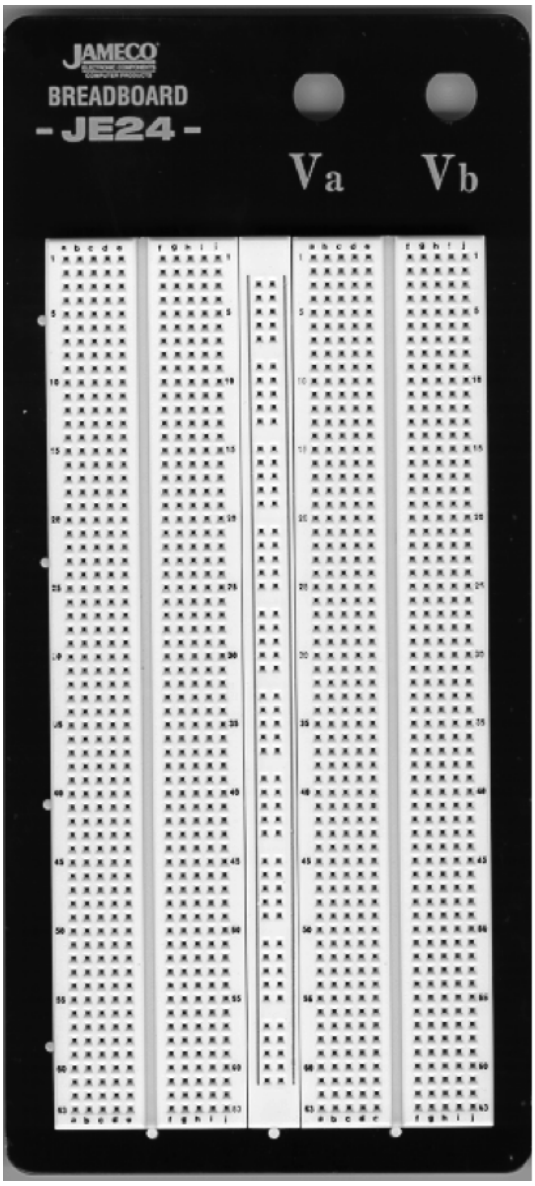


Figure C.1: A simple breadboard

Quick test sockets are intended to accommodate ICs and other components, and their internal connections are shown in the bottom view in Figure C.1. Parallel groups of five contacts are connected together. The ICs are inserted across the center insulating strip, and connections are made by wires inserted into the contacts adjacent to the pin to be connected.

Most general purpose breadboards are designed to accept a range of solid conductor wire sizes, but wire jumpers included in the kit work best. *Do not force component leads that have a larger diameter than those provided in the kit into the breadboard socket; otherwise, the socket spring clip into which the lead is inserted may be damaged.*

To insert new IC packages, the leads may have to be bent inward slightly so their spacing will match the spacing on the breadboard. An extraction tool should be used to remove IC packages. If an extraction tool is not used, the leads of an IC may get bent or even broken. Connection wires may be inserted by hand, but a pair of long-nosed pliers can be used for tight places. In general, it may be easier to insert leads which have been cut at an angle of about 45 degrees with respect to the axis of the wire. It is best to run wires around IC packages rather than over IC packages. This construction will facilitate easy removal and replacement an IC if necessary. Wires should be kept close to the surface of the breadboard, and they should be as short as possible. In order to help prevent a jumper wire from breaking inside its breadboard socket, the end of any jumper wire that is bent or appears overly flexed should be trimmed and re-stripped.

All ICs should be inserted with the same orientation to facilitate wiring and debugging. It does

not pay to reverse the orientation of some ICs to minimize wire lengths. A locating notch or small circle identifies pin one. For most (but not all) ICs, if pin one is oriented on the lower left side of an IC package, the IC ground pin is on the lower right, and the IC power pin is on the upper left side. Since breadboards have ground buses on the left and power buses on the right, it is best to insert all ICs with pin 1 on the same side of the insulating strip. Power and ground connections for each IC on the breadboard should be wired first–red wire for V_{cc} and black wire for ground (GND). After wiring the circuit, each pin of each IC should be examined sequentially to verify its connection; this procedure will avoid problems later on.

IC devices may be damaged if the supply voltage is removed while an input voltage is still active. Also, when power is applied to IC packages, there is no protection against reversed V_{cc} and GND connections. If any IC runs too hot to touch, the power supply should be turned off, and the power supply leads should be checked. It is good practice to shut off the power while wiring or making any modifications to a circuit. In addition, the power supply voltage used must be below the absolute maximum supply voltage rating for the IC family. Transistor-transistor-logic (TTL) devices are designed to operate from a 5-volt DC supply, and the absolute maximum supply rating of 7 V should not be exceeded. A voltage greater than 5.5 V applied to an input terminal of a TTL device may also damage the device.

The importance of neat wiring cannot be overemphasized. Neat wiring is easier to debug and is more reliable than messy wiring. With messy wiring, removing or inserting one wire can have an unpredictable and often untraceable effect of removing (or worse, merely loosening) other wires tangled with it in a wiring maze. On a neatly-wired board, the instructor has at least some chance of successfully helping the student debug.

Appendix D Instructions for Drawing Circuit Diagrams

Preparing a neat, easily readable circuit wiring diagram is a vital preliminary task that must be done before a circuit can be constructed. All drawings or diagrams must be prepared using the computer drawing program Altera Quartus II. The instructions for using Altera Quartus II can be found in the appendices. In a lab report, a figure label consisting of both a number and caption is required for each drawing (e.g., Figure D.1 on the next page) and should be placed below the drawing. The pins used on every IC should be numbered, except those pins used for V_{cc} and GND. Figure D.1 gives an example of a circuit wiring diagram. Altera Quartus II is available in CE labs. The CE labs are open and available for use every day of the week, except during scheduled lab times. A lab schedule is posted outside of each lab.

For circuits consisting of more than one package, a parts-placement diagram is required to show the location of each IC package on the breadboard. Also, the parts-placement diagram gives the pin numbers for V_{cc} and GND on each of the IC packages. Identifying letters, numbers, or a combination of both (i.e., reference designator) may be used inside (or beside) each IC on both the parts-placement diagram and the circuit diagram to allow proper package (i.e., specific gate within an IC) identification. Within each gate's symbol, Figure D.1 uses a letter to identify each IC package and a number to identify the gate within each package. Figure D.2 shows one possible parts-placement diagram that could be used for the wiring diagram shown in Figure D.1. *It is not necessary to show gate assignments on the parts-placement diagram*, but pin one of each package is should be shown to indicate its alignment. If each IC package is oriented in the same way, it is easy to remember the pattern, and thus fewer wiring mistakes would be expected.

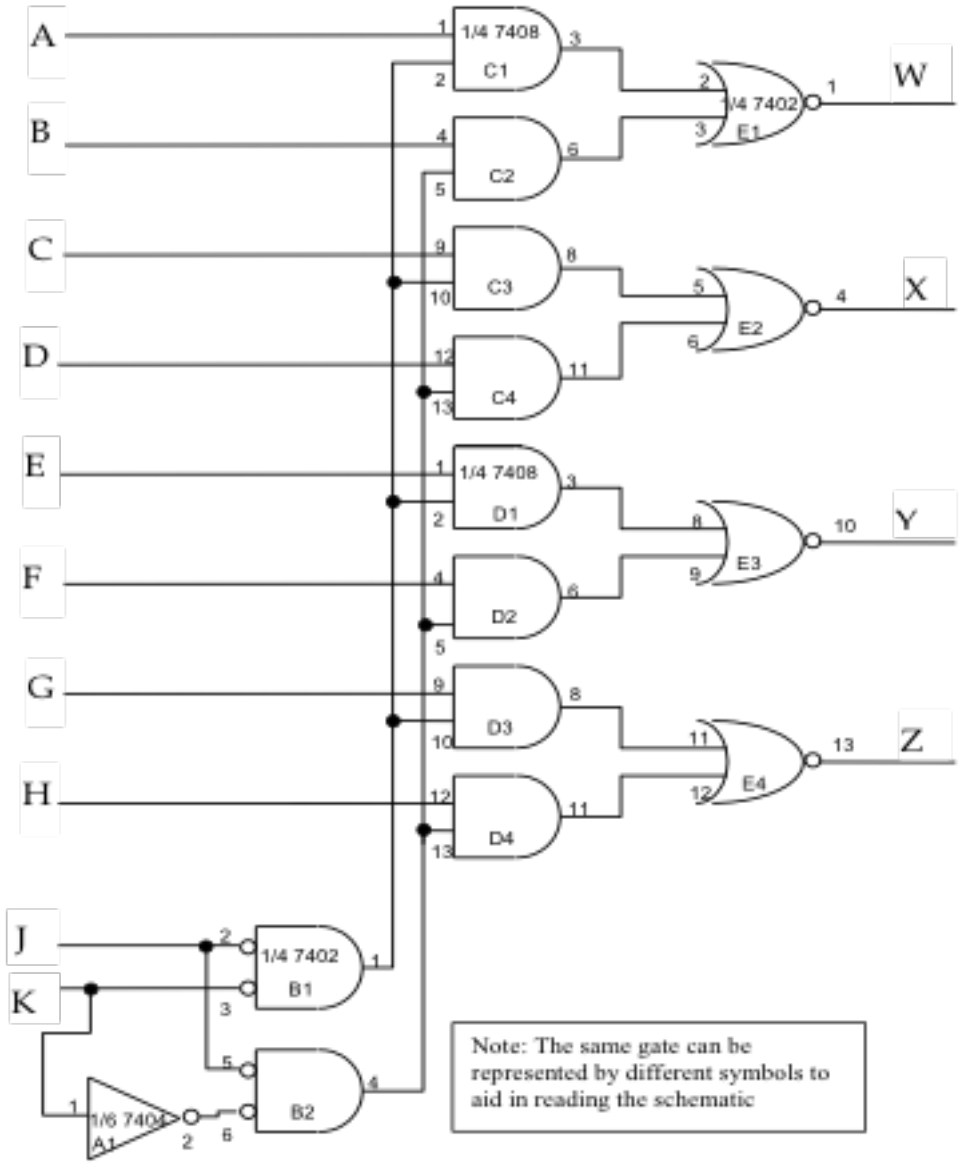


Figure D.1: Circuit diagram

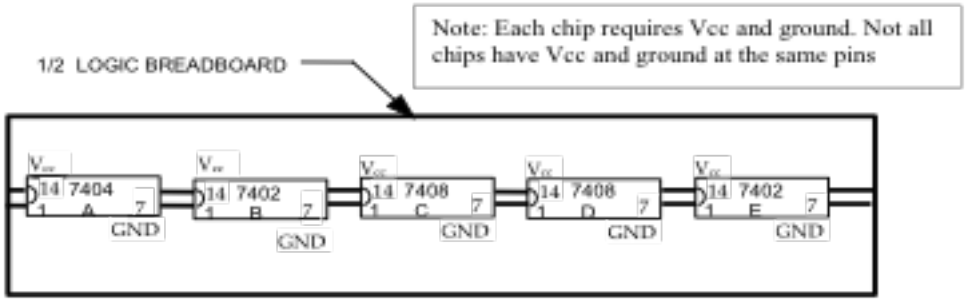


Figure D.2: Parts-placement diagram

Appendix E ModelSim and VHDL Tutorial

In this tutorial, the ModelSim software is used to model behavior and structure of a 1-Bit Full Adder (1-Bit FA or simply FA) circuit. The FA circuit has three inputs (`A`, `B`, `Cin`) and two outputs (`Sum` and `Cout`). The truth table of a FA is presented in Table E.1.

Table E.1: Truth table for 1-bit full adder

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Output `Sum` can be calculated as

$$\text{Sum} = \text{A XOR B XOR Cin}$$

and `Cout` as

$$\text{Cout} = (\text{A AND B}) \text{ OR } (\text{A AND Cin}) \text{ OR } (\text{B AND Cin})$$

The tutorial consists of two parts: (1) modeling the circuit in ModelSim using data flow and structural approaches and (2) simulating the circuits using force commands and a simple test bench.

Modeling using data flow

Start ModelSim simulator and create a new project: File → New → Project... Name the project FA, set the project location (remember to avoid spaces in the names of directories and files), and click OK. A new window should appear to add files to the project. Choose Create New File. Enter full_adder.vhd as the file name and click OK. Then close the "add new files" window.

Double click the full_adder.vhd source found under the Workspace window's project tab. This will open up an empty text editor configured to highlight VHDL syntax. Copy the source code presented in Listing E.1. After writing the code for the entity and its architecture, save and close the source file.

Listing E.1: Entity declaration and data flow model of a 1-Bit FA circuit.

```
-------------------------------------------------------------------------
-- Company       : RIT
-- Author        : Marcin Lukowiak
-- Created       : xx:xx:xx xx/xx/xxxx
--
-- Project name : lab#
```

```
-- File          : full_adder.vhd
--
-- Entity        : full_adder
-- Architecture  : df (data flow)
--
-- Revision
-- Rev 0.01      : xx:xx:xx xx/xx/xxxx file created
--
-- Tool version  : VHDL'93
-- Description   : The following is the entity and
--                 behavioral description of a
--                 1-bit full adder.
--
-- Notes         :

library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
    port (                  -- full adder inputs:
    A    : in std_logic;    -- A, active high
    B    : in _________;    -- B, active high
    Cin  : __ std_logic;    -- Carry In, active high

                            -- full adder outputs:
    Sum  : out _________;   -- Sum, active high
    Cout : ___ std_logic);-- Carry Out, active high
    end full_adder;

architecture df of full_adder is
    -- internal signal declarations
    signal s_1, s_2, s_3 : std_logic:='0';
begin
    -- use one signal assignment statement to model
    -- 3-input XOR gate that calculates value for the Sum output

    Sum <= ________________________________;

    -- write three statements to model 2-input AND gates
    -- calculating partial values for Cout output

    s_1 <= ______________________;
    s_2 <= ______________________;
    s_3 <= ______________________;

    -- use one signal assignment statement to model
    -- 3-input OR gate that calculates the final value for Cout output
```

```
    Cout <= ______________________;

    -- please note alternative model for calculating Cout could e.g. use
    -- just one signal assignment statement
    -- Cout <= (A and B) or (A and Cin) or (B and Cin);
end;
```

Compiling files

Select the file from the project files list frame and right click on it. Select Compile then Compile Selected to just compile this file or Compile All for the files in the current project. If there are errors within the code or the project, a red failure message will be displayed. Double click these red errors for more detailed errors. Otherwise, if all is well then no red warning or error messages will be displayed.

Simulation with force command

To simulate, first the entity design has to be loaded into the simulator. Do this by selecting from the menu: Simulate → Start Simulation. A new window will appear listing all libraries. Unroll library work to see the entities (not filenames) that are in the work library (the place where all your designs will be found after successful compilation). Select **full_adder** entity, **df** architecture for simulation and click OK. Uncheck Enable optimization - it is best way to maximize visibility within all parts of the design

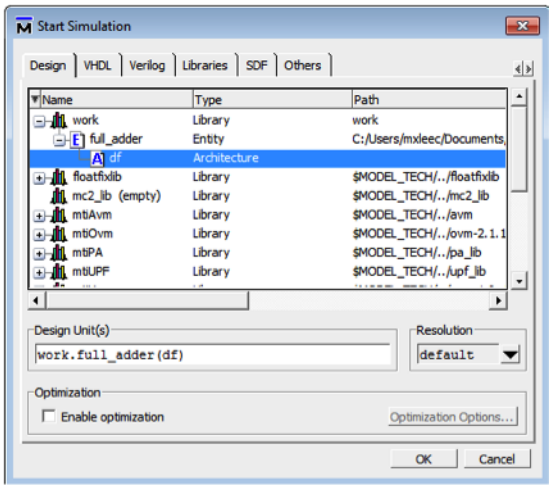


Figure E.1: The Start Simulation window

Often times it will be necessary to create entities with multiple architectures. In this case the architecture has to be specified for the simulation – you will see it later in this tutorial.

After the design is loaded, clear up any previous data and restart the simulation timer by typing in the prompt: `restart`

Now, before simulating, the input signals or forces for the tested design must first be initialized. To do this, open up a text editor and create data file in the ModelSim project directory using the following format:

```
force objectname value1 time1, value2 time2 ...  -repeat duration
```

The force is defined in value-time pairs, where the inputs will change to the value at the given time. The repeat part is optional, and is commonly used to generate a clock. So, for example:

```
force    A     0 0ns, 1  50ns -repeat 100ns
force    B     0 0ns, 1 100ns -repeat 200ns
force    Cin   0 0ns, 1 200ns -repeat 400ns
force    D     1 0ns, 0 150ns, 1 200ns, 0 255ns
```

creates three clocks with 50% duty cycles which generate all of the eight possible inputs to the circuit over a period of 400ns. The fourth signal, `D`, starts at "1", transitions low at 150ns, returns to high at 200ns, finally going low at 255ns, and remaining there until the simulation is finished. In general, the values have to be set according to the input type required by the entity. For example, if the input required a hexadecimal input, the value can be set as hexadecimal:

```
force A 16#0 0, 16#C 20, 16#8 100
```

The "16" denotes the radix for hexadecimal, and the "#" separates the base and the value. Other examples:

`10#9` - decimal radix

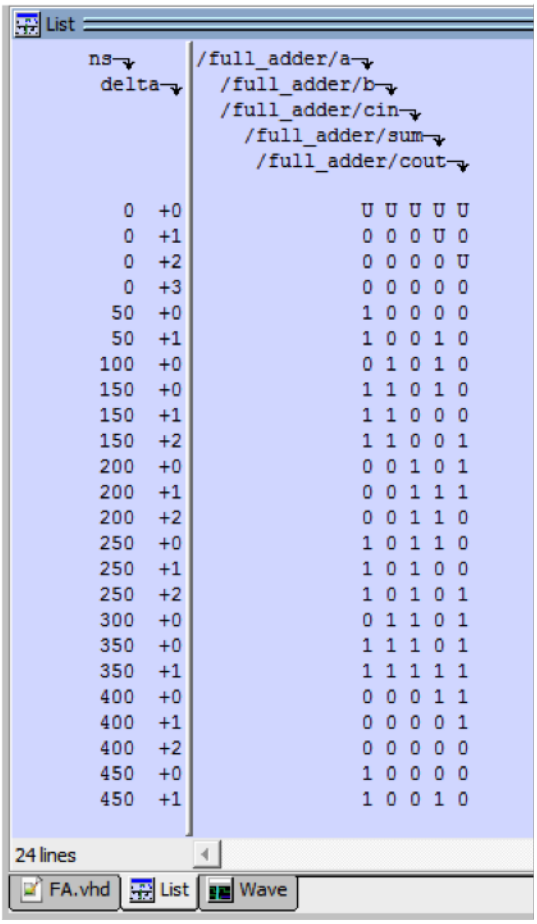
`2#111` - binary radix

When all the inputs have been assigned values, save the file (any filename will do) and quit the text editor. Next, the forces data has to be loaded into the simulator by typing:

```
do forcesfilename
```

This executes the force commands stored in the *forcesfilename*. The forces can also be entered at the prompt, instead of from a file, for quick changes in the time-value pairs.

To see the simulation results, names of the inputs, outputs and intermediate signals must appear in the Wave window and/or List window. In the Objects window, Ctrl+right-click on all inputs and outputs, select Add → To Wave → Selected signals, and later Add → To List → Selected signals.



The other options are: "Signals in region" which displays all the signals listed in the signals window, and "Signals in design" displays all the signals in the project (which can be numerous when the design is dealing with other libraries. The waveform and list windows are now displayed, and the project can now be run. Run the program by typing in the ModelSim prompt:

```
run duration
```

where *duration* is a numerical value denoting time. For this full adder, run the simulation for 450 ns. The waveform and list windows now display the signal results of the run for the du-

ration.

Listing form is used to see all events of the simulation or to see changes occurring to a value at what time. The left column is time elapsed, along with the delta delay. The right columns are the values at the given time and delta delay.

Modeling using structural style

Create a new VHDL file by choosing from the menu File → New → Source → VHDL. Save it as xor3.vhd. Content of this file is presented in Listing E.2. Click on the Project tab to make it active. Add the xor3.vhd file to the project by choosing Project → Add to Project → Existing File. Compile this file.

Listing E.2: Entity and architecture pair for a three-bit XOR gate

```
library ieee;
use ieee.std_logic_1164.all;

entity xor3 is
    port (_______ : in std_logic;
          Y       : out std_logic);
end xor3;

architecture df of xor3 is
begin
    -- this model needs to account for 4 ns propagation delay

    __ <= A xor B xor C ___________;
end;
```

Repeat this process for the file shown in Listing E.3.

Listing E.3: Entity and architecture pair for a two-bit AND gate

```
library ieee;
use ieee.std_logic_1164.all;

entity and2 is
    port (A, B : in std_logic;
          _____: out std_logic);
end and2;

architecture df of and2 is
begin
    Y <= ____________after 2 ns;
end;
```

Create a third file, or3.vhd, and fill in the missing content from Listing E.4.

Listing E.4: Entity and architecture pair for a three-bit OR gate

```
library ieee;
use ieee.std_logic_1164.all;

-- Entity/architecture pair for
-- 3-input OR gate with 3 ns
-- propogation delay should be here
--------------------------------
--------------------------------
--------------------------------
--------------------------------
--------------------------------
--------------------------------
--------------------------------
--------------------------------
--------------------------------
--------------------------------
```

Modify full_adder.vhd by adding a second architecture as shown in Listing E.5.

Listing E.5: Architecture for 1-bit FA structural circuit.

```
architecture struct of full_adder is
    -- component declaration for the full adder circuit
    component xor3 is
        port (A, B, C : in std_logic;
              Y       : out std_logic);
    end component;

    component and2 is
        port (A, B : in std_logic;
              Y    : out std_logic);
    end component;

    component or3 is
        port (A, B, C : in std_logic;
              Y       : out std_logic);
    end component;

    -- signal declarations
    signal s_1, s_2, s_3 : std_logic:='0';

begin
    -- component instantiation for calculating Sum
    xor3_instance1 : xor3 port map (A => A, B => B, C => Cin, Y => Sum);

    -- component instantiations for calculating Cout
    and2_instance1 : and2 port map (----------------------);
    and2_instance2 : and2 port map (----------------------);
```

```
    and2_instance3 : and2 port map (_____________________);

    or3_instance1 : or3 port map (_____________________);

end;
```

Simulate this model by following the same steps as for architecture **df** - this time however, you will select architecture **struct** of the FA as shown in Figure E.3.

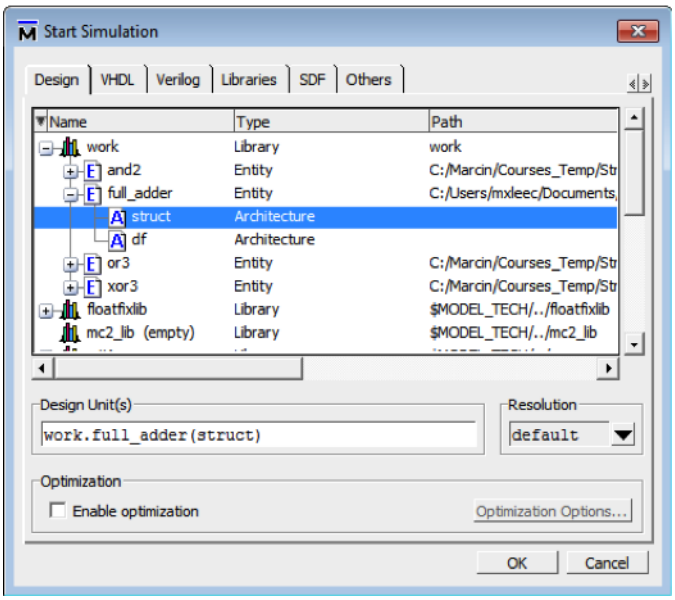


Figure E.3: Start simulation window with the struct architecture selected

Simulation with a test bench

VHDL test benches are VHDL design entities that describe simulation input and perform verification using standard VHDL language statements — a test bench is a top level structural model that instantiates the Unit Under Test (UUT)/Design Under Test (DUT) and drives it with a set of test vectors and compares the generated results with expected responses.

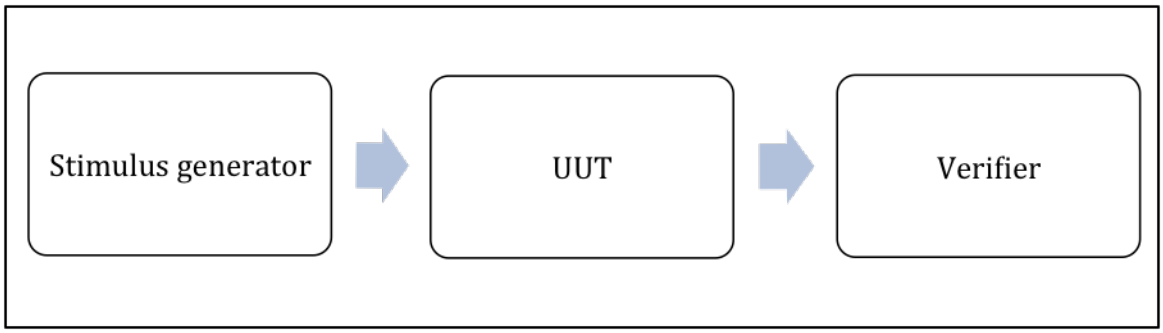


Figure E.4: Test bench

Create a new VHDL file by choosing from the menu File → New → Source → VHDL. Save it as

tb.vhd. The content of this file is presented in Listing E.6. Click on the Project tab to make it active. Add the tb.vhd file to the project by choosing Project → Add to Project → Existing File. Compile this file and load tb entity for simulation. Open the Wave window and add all signals of interest for inspection. This time you don't need to run any *force* commands to stimulate inputs of the FA circuit — typing `run` ***`duration`*** inside ModeSim prompt will be sufficient.

Modify your test bench to include automatic verification of both architectures at the same time — see Listing E.7 for details.

If something is not clear discuss it with your lab instructor.

Listing E.6: Simple test bench for the 1-bit FA circuit

```
library ieee;
use ieee.std_logic_1164.all;

entity tb is
end tb;

architecture behavior of tb is
    -- component declaration for the Unit Under Test (UUT)
    component full_adder
        port (                          -- full adder inputs:
                A    : in std_logic;  -- A, active high
                B    : in std_logic;  -- B, active high
                Cin  : in std_logic;  -- Carry In, active high

                -- full adder outputs:
                Sum  : out std_logic; -- Sum, active high
                Cout : out std_logic);-- Carry Out, active high
    end component;

    -- signals to stimulate UUT inputs
    signal sA: std_logic :='0';
    signal sB: std_logic :='0';
    signal sCin: std_logic :='0';

    -- signals to read UUT outputs
    signal sSum: std_logic :='0';
    signal sCout: std_logic :='0';

    -- component binding - this tells the simulator
    -- which entity-architecture pair will be simulated as UUT

    -- use this to simulate architecture df

    for UUT : full_adder use entity work.full_adder(df);

    -- use this to simulate architecture struct
    -- for UUT : full_adder use entity work.full_adder(struct);
```

```
begin
    -- Unit Under Test (UUT) instance
    UUT: full_adder port map (A=>sA, B=>sB, Cin=>sCin,
                              Sum=>sSum, Cout=>sCout   );

    -- stimulus
    sA <= not sA after 50 ns;
    sB <= not sB after 100 ns;
    sCin <= not sCin after 200 ns;

end;
```

Listing E.7: Self checking test bench for the 1-bit FA circuit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity tb is
    end tb;

architecture behavior of tb is
    -- component declaration for the Unit Under Test (UUT)
    component full_adder
        port (                          -- full adder inputs:
              A    : in std_logic;      -- A, active high
              B    : in std_logic;      -- B, active high
              Cin  : in std_logic;      -- Carry In, active high

                                        -- full adder outputs:
              Sum  : out std_logic;     -- Sum, active high
              Cout : out std_logic);-- Carry Out, active high
    end component;

    -- signals to stimulate UUT inputs
    signal sA: std_logic :='0';
    signal sB: std_logic :='0';
    signal sCin: std_logic :='0';

    -- signals to read UUT outputs
    signal sSum1, sSum2: std_logic :='0';
    signal sCout1, sCout2: std_logic :='0';

    -- component binding
    for UUT1 : full_adder use entity work.full_adder(df);
    for UUT2 : full_adder use entity work.full_adder(struct);

begin
```

```
    -- Unit Under Test (UUT) instance
    UUT1: full_adder port map (A=>sA, B=>sB, Cin=>sCin,
                               Sum=>sSum1, Cout=>sCout1   );

    UUT2: full_adder port map (A=>sA, B=>sB, Cin=>sCin,
                               Sum=>sSum2, Cout=>sCout2   );
    -- stimulus
    process
        variable vIn : std_logic_vector (2 downto 0):="000";
    begin
        (sCin, sB, sA) <= vIn;
        wait for 50 ns;
        vIn := vIn + 1;
    end process;

    process
        variable vA, vB, vCin, vOut : std_logic_vector (1 downto 0):="00";
    begin
        wait on sA, sB, sCin;
        vA := '0'& sA;
        vB := '0'& sB;
        vCin := '0'& sCin;
        vOut := vA + vB + vCin;  -- expected outputs
        wait for 20 ns;          -- propagation delay

        assert (sSum1 = vOut(0) and sCout1 = vOut(1))
        report "wrong output for df architecture at time:" & time'image(now)
        severity warning;

        if (sSum2/=vOut(0) or sCout2/=vOut(1)) then      -- alternative coding style
            assert false
            report "wrong output for struct architecture at time:" & time'image(now
            severity warning;
        end if;
    end process;
end;
```

Notes