UMM AL-QURA UNIVERSITY

College of Computer and Information Systems

Computer Engineering Department

14031201-4

Digital Logic Design

Lab Manual

Student Name: ____________________________________

Student ID: ____

Section: ________________ Group: __________________

Session (Fall / Spring / Summer) ____________________

This page is intentionally left blank

TABLE OF CONTENTS

Prepared By:	Engr. Zafar Zaheer-ul-Haque	2012
Revised By:	Dr. Khaled Khayyat and Dr. Ahmed Morgan	2016
Reviewed By:	______________________________	_____________
Approved By:	______________________________	_____________

Umm Al-Qura University
Computer Engineering Department

LABORATORY SAFETY GUIDELINES

A. General Laboratory Safety Rules

1. **Personal Safety**
    - Be familiar with the electrical and fire hazards associated with your workplace.
    - Be as careful for the safety of others as for yourself. Think before you act.
    - Be tidy and systematic.
    - Avoid bulky, loose or trailing clothes. Avoid long loose hair.
    - No one is allowed to enter in the lab area bare foot due to increased risk of electric shock.
    - Remove metal bracelets, rings or watchstraps when working in the laboratories.
    - Avoid working with wet hands and clothing.

2. **Food, Beverages and Smoking**
    - Due to the increased risk of electric shock, no drinking of beverages, consumption or storage of any kind of food is allowed in the laboratory.
    - Smoking is prohibited in all laboratories in all timings.

3. **Soldering**
    - No one is allowed to do soldering in any of the computer engineering laboratories except the graduation project design laboratory.
    - Anyone doing soldering in the graduation project design laboratory must wear appropriate apparel, socks, gloves, covered shoes and safety goggles to prevent the possibility of severe burns resulting from the splashing or dripping of hot liquefied solder into the face and eyes or on to the exposed skin on the chest, hands, legs, and feet.
    - Students who are not so properly attired for these tasks will NOT be allowed to perform any type of soldering in the graduation project design laboratory.

4. **Laboratory Operating Hours**
    - Students are never allowed to work alone in any lab area other than scheduled laboratory operating hours unless either a Lab T/A or Course Instructor is present inside that lab area.
    - The laboratory operating hours for students are posted on the entrance doorway and on the notice board of computer engineering department.

5. **Power Supply Related Safety**
    - Voltages above 50-VAC or 120-VDC are always dangerous.
    - Extra precautions should be considered as voltage levels are increased.
    - Never make any changes to circuits or mechanical layout without first isolating the circuit by switching off and removing connections to power supplies.

6. **Laboratory Equipment**
    - Lab equipment may not be removed from the Computer Engineering lab areas without the permission of the Laboratory Supervisors.
    - Laboratory bench equipment (except for some lab bench computers) must be turned off before closing down the lab area for the day.
    - Never open (remove cover) of any equipment in the laboratories.
    - Never "jump," disable, bypass or otherwise disengage any safety device or feature of any equipment in the laboratories.
    - Laboratories shall be locked when unoccupied.

7. **Waste Management Safety**
    - Know the correct handling, storage and disposal procedures for batteries, cell, capacitors, inductors and other high energy-storage devices.

8. **Equipment Safety**
    - Before equipment is energized ensure, circuit connections and layout have been checked by a Teaching Assistant (TA) and all colleagues in your group has given their consent.
    - Experiments left unattended should be isolated from the power supplies. If for a special reason, it must be left on, a barrier and a warning notice are required.
    - Equipment found to be faulty in any way should be reported to the lab supervisor and taken out of service until inspected and declared safe.

9. **Equipment Accessories**
    - Use extension cords only when necessary and only on a temporary basis.
    - Request new outlets if your work requires equipment in an area without an outlet.
    - Discard damaged cords, cords that become hot, or cords with exposed wiring.

B. Electrical and Fire Emergency Responses

1. **Police, Fire or Medical Emergency**
    - Use the telephone located in the laboratory area and press 0-996 to notify police, fire, and ambulance for emergency help.
    - Everyone present in the laboratory area shall be familiar with the locations and operation of safety and emergency equipment, including but not limited to, fire extinguishers, first aid kits, emergency power off system, fire alarm pull stations, and emergency telephones.

2. **Electric Shock**
    - When someone suffers serious electrical shock, he may be knocked unconscious.
    - If the victim is still in contact with electrical current, immediately turn off the electrical power source.
    - If you cannot disconnect the power source, depress the Emergency Power Off switch.
    - Do not touch a victim that is still in contact with a live power source; you could be electrocuted! Have someone call for emergency medical assistance immediately. Administer first-aid, as appropriate.

3. **Electrical Fire**
    - If an electrical fire occurs, try to disconnect the electrical power source, if possible.
    - If the fire is small and you are not in immediate danger; and if you have been properly trained in fighting fires, use the correct type of fire extinguisher to extinguish the fire.
    - When in doubt, push in the Emergency Power Off button.
    - NEVER use water to extinguish an electrical fire.

4. **Emergency Power Off**
    - Every lab is equipped with an Emergency Power off System.
    - When this switch is depressed, electrical power to the lab will shut off, except for lights.
    - Only authorized personnel are permitted to reset power once the Emergency Power Off system has been engaged.

5. **Building Evacuation in Emergency**
    - Everyone present in the laboratory should be familiar to emergency exits & way out plans.
    - Use the nearest exit doorway from lab area closest to the stairwell to exit the building.
    - Follow the Emergency Exit Signs posted in the hallways. Do not use elevators.
    - Lab Teaching Assistants (T/As) or Instructor shall make sure all persons are out of the laboratory area and follow the directions posted at each doorway to the laboratory area.

The above general laboratory safety rules are designed to safeguard you and your co-workers, fellow students and colleagues and are a minimum requirement for individuals working in the computer engineering laboratories at Umm Al-Qura University, Makkah Al-Mukarramah. Specialized training and rules may apply depending on type and scope of activities involved.

General Information

1. Introduction to the Lab:

- Each lab is a 3-hour session
- The lab objective is to introduce students to the practical world of digital system design and the various digital components. This includes how design digital circuits, how to simulates them using software packages, and how to construct them using standard Integrated Circuits (ICs) and other hardware that are readily available commercially. Students will also be familiarized with the procedure of designing, simplifying, simulating, implementing and testing many combinational and sequential circuits that explained and presented in lectures.
- Some parts of the labs do not give you step-by-step instructions on how to do them. They require you to think carefully about how you are going to do them.
- You will do the labs with the Logisim software as well as prototype boards and components that are offered to you. Logisim is a free graphical tool for designing and simulating logic circuits. You could download it from: http://www.cburch.com/logisim/. The website also includes some documents that help you getting started with the software.
- Every effort will be made to cover the lab materials in class before the scheduled lab period. However, if the lectures do lag behind the labs, please consult the textbook, the notes, the TA or the course instructor for any information you need.

2. Lab Rules:

- Be regular and disciplined: Students are expected to attend all lab sessions regularly.
- Come to the lab sessions on time. Action may be taken to late comers.
- The due date for any lab report is one week after the lab is done. Submit your lab reports on time. Any late report, for no valid reason, will not be accepted.
- Lab is yours, so help to keep it clean. Bringing foods, water or newspaper are forbidden.
- Come prepared. Read the background information and discuss the experiment with your colleagues before coming.
- Bring the Lab manual on every time you come to the laboratory.
- Do not make walking around, chatting, or noise during lab period.
- Do not change your lab group or bench without taking a permission from the Lab instructor.
- Do not do anything in a hurry. Ask the instructor if you don't understand any point or step.
- Each lab is to be demoed to the lab TA to confirm its operation. Furthermore, the TA will ask questions regarding the lab, to ensure the proper understanding of the lab materials.
- After finishing the experiment, disconnect the connection wires that you used during the experiment, and put them in the wire kit, take out all IC chips from the training kit and put them back in the IC Cabinet. Be sure to return all materials you used, to clean your table, and to inform the Lab instructor when you depart.
- The student's mark will be affected if these rules are not obeyed.

3. Grading :

- The Lab weights 25% of the whole course grade, the following grading scheme will be used:

Technical Reports	5%
Participation and Activities	5%
Experiment 7: Exam on Combinational Logic Modules	5%
Final Lab Exam	10%

- Discipline and preparedness will be taken into account when assessing the student.
- Two marks will be discarded for each unexcused absence.

- A student who has more than two unexcused absence will receive a final grade of zero.
- Students could discuss their lab experiment and results with each others. However, every student should write his/her report by himself/herself. No marks will be given if any cheating in the way of copying is noticed.

4. *Requirements for Lab Reports:*

- Lab reports should be submitted in a week from the date the lab is performed and should include all of the following sections:
  - Cover Page: The cover page of your lab report should include the experiment name and number, your name and number, your bench order and the submission date.
  - Discussion and Conclusions: In this section, you present the results you got during the lab, your observations about these results, your comments, and your conclusions.
  - Questions answers: if your lab manual includes questions to be answered, you should answer all the questions completely and accurately. Some questions require you to simulate certain circuits. Snapshots from the simulation should be included in your report.
- When writing the lab report, ensure that all sentences are grammatically correct & without mistakes in spelling. You can use a computer to make your report, the following powerful softwares may help you:
  - SmartDraw or Microsoft Visio: to create and draw diagrams.
  - Logisim: to enter and simulate logic circuits. (available at: http://www.cburch.com/logisim/). Similar tools are Logicly (available at http://logic.ly/). Circuit maker (available at http://circuitmaker.com/), Multisim (available at http://www.ni.com/multisim/), and Xilinx ISE (available at http://www.xilinx.com).
- Please ask the instructor if any point here is not clear to you.

This page is intentionally left blank

Weekly Schedule

(Based on the University Calendar)

Week	Lab/Tutorial
1	No lab nor tutorial
2	**Tutorial 1:** Number systems – Conversions
3	**Tutorial 2:** Number systems – Operations and Codes
4	**Experiment 1:** Introduction to logic gates
5	**Tutorial 3:** Implementation and simplification of Boolean functions using Boolean Algebra
6	**Experiment 2:** Implementation and Simplification of Boolean functions
7	**Experiment 3:** NAND and NOR implementation
8	**Tutorial 4:** Implementation and simplification of Boolean functions using K-maps
9	Reading Break
10	**Experiment 4:** Combinational logic modules – Adders and subtractors
11	**Experiment 5:** Combinational logic modules – Decoders
12	**Experiment 6:** Combinational logic modules – Comparators and multiplexers
13	**Experiment 7:** Exam on Combinational logic modules – Design of an arithmetic circuit
14	**Tutorial 5:** Sequential logic, flip flops, FSM analysis and design
15	**Experiment 8:** Latches, Flip flops and counters
16	**Final Lab Exam**

Week (2)

Tutorial 1: Number systems – Conversions

Reminder: Please, remember to print and bring the tutorial with you

Week (3)

Tutorial 2: Number systems – Operations and Codes

Reminder: Please, remember to print and bring the tutorial with you

Week (4)

Experiment 1: Introduction to Logic Gates

1.1 Objectives:

- To get familiar with the usage of the lab software and equipments.
- To describe and verify the operation of AND, OR, NOT, NAND, NOR, XOR, and XNOR gates.
- To introduce a basic knowledge of Integrated Circuits (ICs) operation.
- To practice how to simulate a simple digital circuit using a software package and to build it using ICs and other digital components.

1.2 Background Information:

Logic gates are the simplest component of any logic circuit. In order to understand the computer logic, you should understand and master the logic operators (i.e., gates). A gate is a digital electronic circuit having only one output, but one or more inputs. The output that appears at the output of the gate depends on the input combinations.

There are many types of logic gates; such as AND, OR, and NOT, which are usually called the three basic gates. Other popular gates are NAND and NOR gates; which are simply combinations of an AND or an OR gate followed by a NOT gate. Other gates include the XOR “Exclusive-OR” and the XNOR "Exclusive NOR" gates. In this experiment, we will investigate the above mentioned basic logic gates and study their operations according to their truth table, as prescribed during lectures.

1.3 Equipment Required:

1.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

1.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- 1x 7400 TTL QUAD NAND GATE
- 1x 7402 TTL QUAD NOR GATE
- 1x 7404 TTL HEX INVERTER GATE
- 1x 7408 TTL QUAD AND GATE
- 1x 7432 TTL QUAD OR GATE
- 1x 7486 TTL QUAD EXCLUSIVE OR GATE
- 1x 74266 TTL QUAD EXCLUSIVE NOR GATE
- 2x Toggle Switches
- 1x Carbon-film Resistor (470Ω)
- 1x LED

1.4 Prelab:

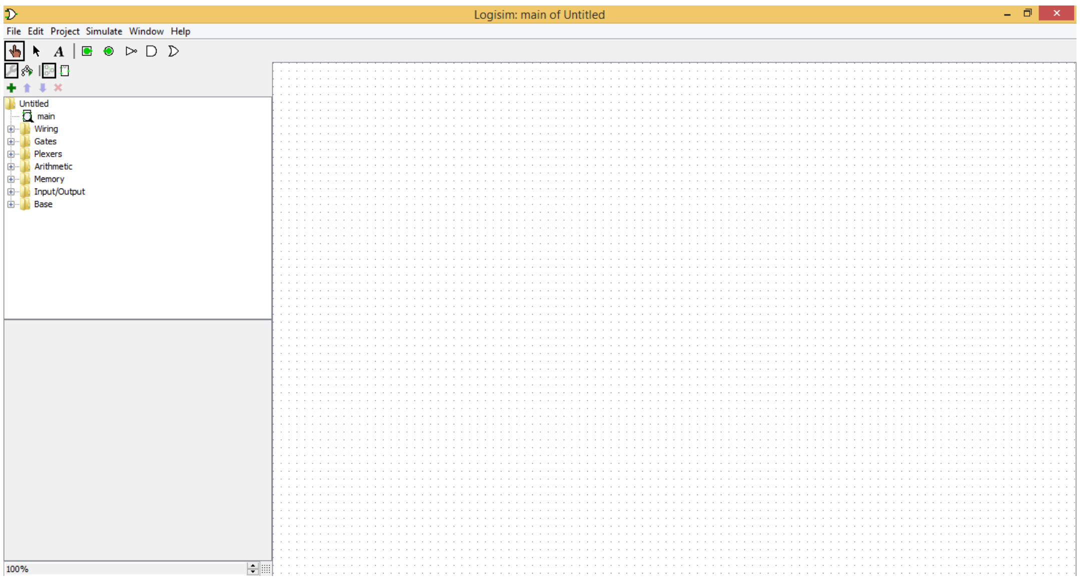
It is recommended that you go through the Logisim beginners tutorial before coming to the lab. The tutorial is available at http://www.cburch.com/logisim/docs/2.7/en/html/guide/tutorial/index.html.

1.5 Software Simulation:

1.5.1 Login into one of the PCs in the lab using your user name and password.

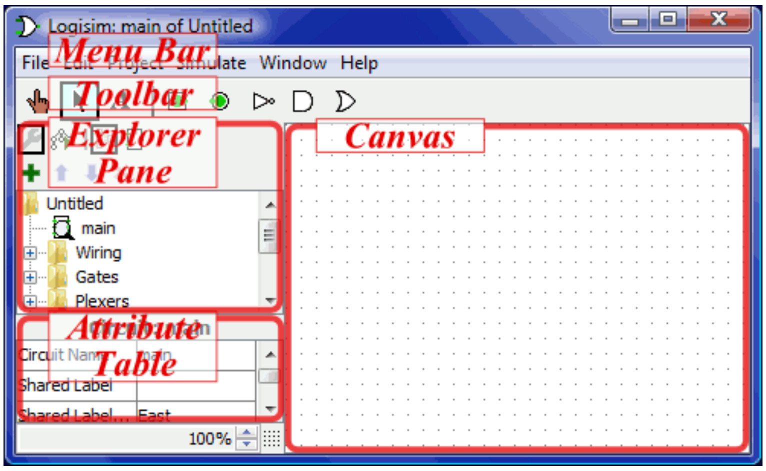
1.5.2 Double click on the *logisim-win-2.7.1* icon on the desktop of the computer.

1.5.3 The Logisim main window will open, as shown below



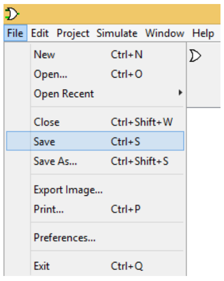
1.5.4 Logisim main window has five main parts:

- Explorer pane that includes the components, which you will use in building your design.
- Attribute table that allows you to adjust the attribute of the components, according to your design needs. If the attribute table is not apparent on your computer right now, it will show up in the following steps.
- Canvas, where you will draw your circuit
- Menu bar that allow you to handle you project and its associated files
- Toolbar that helps you to draw and simulate your circuit

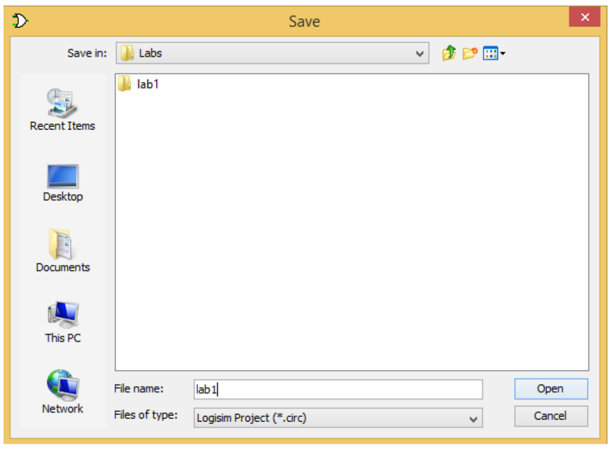


The picture is taken from
http://www.cburch.com/logisim/docs/2.7/en/html/guide/tutorial/tutor-orient.html.

1.5.5 From the File menu, click on _Save_ or simply click Ctrl+S.



1.5.6 The _Save_ window will show up. Browse to the folder, where you want to save you files, write lab1 on the _File name_, then click save.



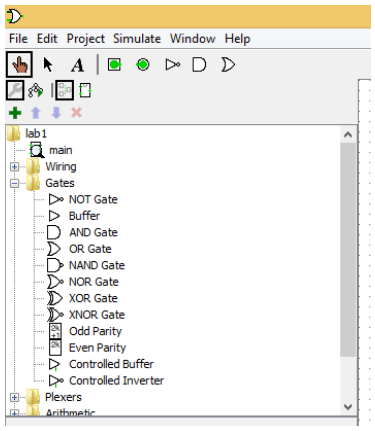
Save in: Labs

lab1

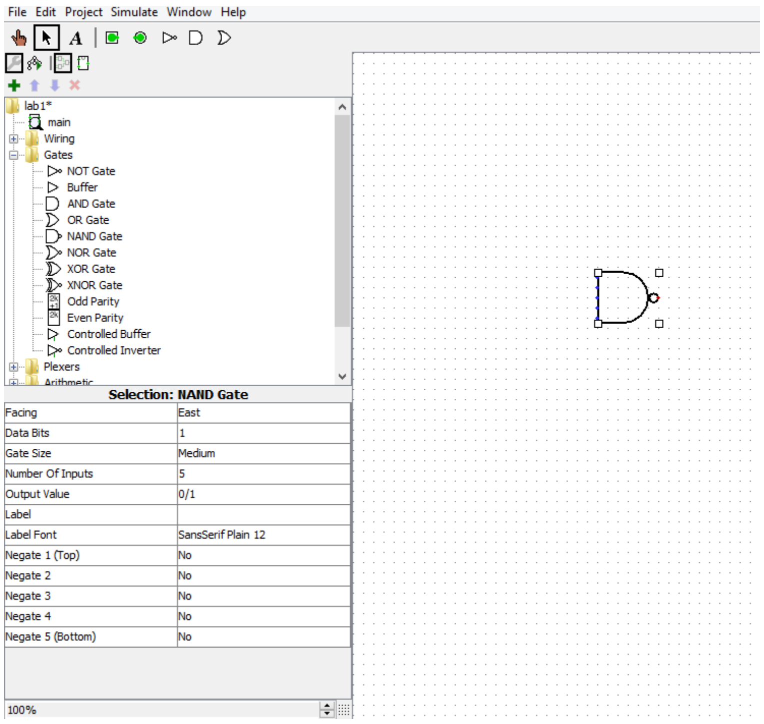
File name: lab1

Files of type: Logisim Project (*.circ)

1.5.7 Now, you are ready to verify the operation of all basic logic gates. Let's start with the NAND gate.
1.5.8 From the *explorer pane*, expand the *Gates* folder by clicking on the + sign beside it.



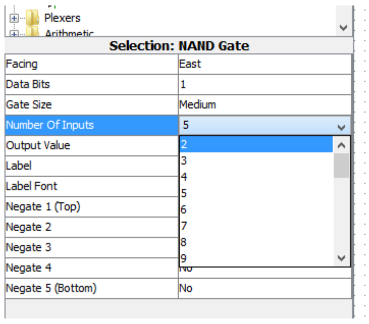
1.5.9 Select the NAND gate and place it onto the *canvas*. Notice that the attribute table of the NAND gate would show up.



Selection: NAND Gate

Facing	East
Data Bits	1
Gate Size	Medium
Number Of Inputs	5
Output Value	0/1
Label	
Label Font	SansSerif Plain 12
Negate 1 (Top)	No
Negate 2	No
Negate 3	No
Negate 4	No
Negate 5 (Bottom)	No

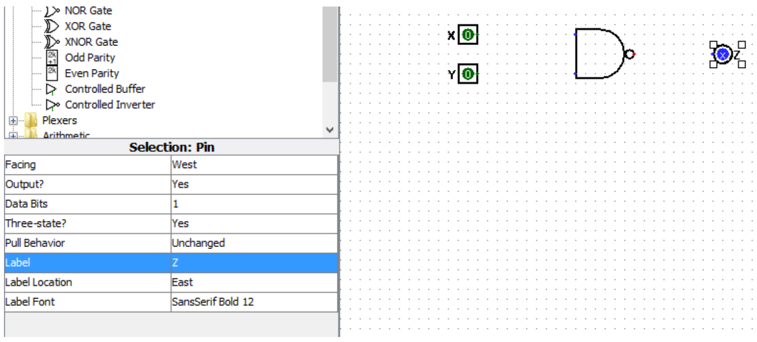
1.5.10 Practice changing one of the attributes of the NAND gate. Change the number of inputs from 5 to 2.



Selection: NAND Gate

Facing	East
Data Bits	1
Gate Size	Medium
Number Of Inputs	5
Output Value	2
Label	3
Label Font	4
Negate 1 (Top)	5
Negate 2	6
Negate 3	7
Negate 4	8
	9
Negate 5 (Bottom)	No

1.5.11 From the <u>*toolbar*</u>, select the *Input tool* ( ) and add two inputs on the canvas to the left of the NAND gate. Similarly, add an output to the right of the NAND gate using the *output tool* ( ). In the <u>*attribute table*</u>, give labels to your inputs and output (X, Y, and Z are used in the following figure).

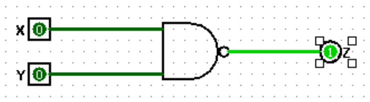


Selection: Pin

Facing	West
Output?	Yes
Data Bits	1
Three-state?	Yes
Pull Behavior	Unchanged
Label	Z
Label Location	East
Label Font	SansSerif Bold 12

1.5.12 If required, you could move your components through the *canvas* using the *edit tool* ( ). Any component could be deleted by selecting it using the *edit tool*, then click the delete key.

1.5.13 From the <u>*toolbar*</u>, select the *edit tool* to wire your circuit. Click on the input X and keep pressing while moving to the upper input of the NAND gate. Notice that when the mouse cursor is over a point that receives a wire, a small green circle will be drawn around it. Logisim allows only horizontal and vertical wires (i.e., inclined wires could not be drawn).



1.5.14 Now, let's verify the operation of the NAND gate by constructing its truth table. Currently, the two inputs, X and Y, are 0s (or colored in dark green), whereas the output Z is 1 (or colored in light green). This constitutes the first row of the 2-input NAND gate truth table.

1.5.15 To try other input combinations, select the *poke tool* ( ), from the <u>*toolbar*</u>, and click on the input you want to toggle. Try all input combinations and fill in the following truth table. Ensure that the operation agrees with what you have learned from the lecture and the textbook.

NAND gate truth table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

1.5.16 Repeat the previous steps to verify the operation of all other basic logic gates: NOR, NOT, AND, OR, XOR, and XNOR. Fill in the following truth tables for each gate. You could start a new file by selecting *NEW* from the *File menu* (or by clicking Ctrl+N)

NOR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

AND gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

OR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

XOR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

XNOR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

NOT gate

X	Z
0	
1	

1.5.17 After you are done with all gates, remember to back up your work onto a USB drive or to email it to yourself for future usage.

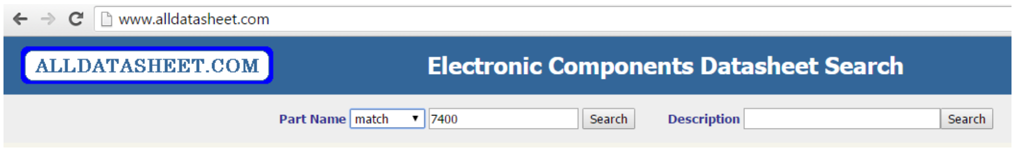
1.6 Hardware Implementation

1.6.1 Collect the components necessary to accomplish this experiment, as listed in Section 1.3. Each gate is fabricated as an Integrated Circuit (IC) chip with a unique number. For example, the 7400 IC chip has four 2-input NAND gates inside it.

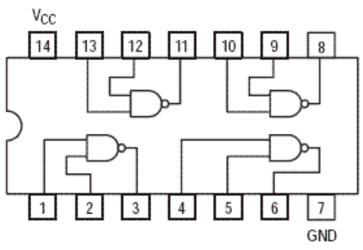
1.6.2 Plug the NAND gate IC chip (7400) into one slot of the IC Training Kit. Place the IC in the slot so that half of the legs are on one side of the middle line and half are on the other side. If you have trouble connecting your IC, please, ask your TA for help.

1.6.3 The connection of each IC is done according to its datasheet. You have to check the datasheet of any IC before connecting it to your circuit. Datasheets are available on the laboratory and they could be obtained from the Internet.

1.6.4 Let's practice getting the datasheet of the 7400 IC. Go to http://www.alldatasheet.com/, write 7400 in the part number and click search.



1.6.5 From the search result, open the datasheet of the 7400 IC. The pin-out should be as shown below. Notice that the legs of the IC are always counted in the anti-clock wise direction from the round notch, which appears to the center left of the IC in the following figure.



1.6.6 According to the datasheet, connect your power supply voltage and ground lines to their corresponding pins in the chips. (i.e., PIN 7 = 0V and PIN 14 = +5). Please, ensure that your wires are inserted all the way to the bottom of the IC Training Kit. Loose and disconnected wires prevent the circuit from working properly.

1.6.7 According to the pin-out diagram of the 7400 IC, wire only one gate to verify its truth table. For example, wire pins 1 and 2 of the IC to two switches and pin 3 to an LED through a 470Ω resistor.

1.6.8 Once all connections have been done, turn the power ON. If necessary, you might have your TA to check your circuit before turning the power ON.

1.6.9 By changing the two input switches and monitoring the LED, fill in the following truth table of the NAND gate.

NAND gate truth table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

1.6.10 Repeat the above steps for each IC package. (i.e., NOR, NOT, AND, OR, XOR, and XNOR). Fill in their truth tables and ensure that the operation agrees with what you have learned from the lecture and the textbook.

NOR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

AND gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

OR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

XOR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

XNOR gate

X	Y	Z
0	0	
0	1	
1	0	
1	1	

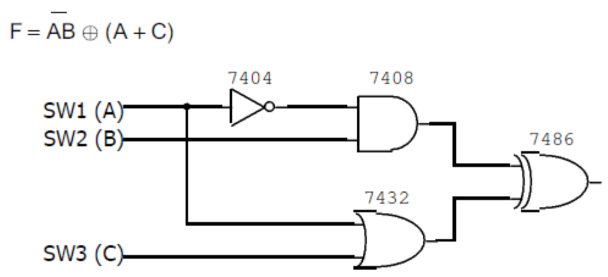
NOT gate

X	Z
0	
1	

1.6.11 After finishing the experiment, turn the power OFF, disconnect the wires, take out all IC chips from the trainer, put back everything you have used and clean your table.

1.7 Questions: Practice Simulation and hardware implementation

From what you have learned in the previous steps, consider the logic circuit shown in the following figure. Use Logisim to simulate it. Verify the circuit operation and fill in its truth table. Include snapshots from your simulation into your report.



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

Week (5)

Tutorial 3: Implementation and simplification of Boolean functions using Boolean Algebra

Reminder: Please, remember to print and bring the tutorial with you

Week (6)

Experiment 2: Implementation and Simplification of Boolean Functions

2.1 Objectives:

- To demonstrate the relationship between a Boolean function, its corresponding truth table, and logic diagram.
- To use the potential of Boolean algebra and K-maps to simplify complex logic circuits.

2.2 Background Information:

2.2.1 Representation of Boolean Functions

A Boolean function is an expression formed with binary variables, binary operators, and an equal sign. A binary variable can take either the value of 0 or 1. A binary operator specifies the logic operation between these variables.

Example:
Consider the following Boolean function consisting of two variables (x, y).

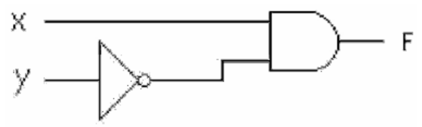
$$F = x\,y'$$

This function is equal to 1 if $x = 1$ and $y' = 1$ (i.e., $y=0$); otherwise, F is equal to 0. The above simple Boolean function is represented as an algebraic expression. Also, the function can be represented as a truth table. In this case, we need to list all the 2^n combinations of 1s and 0s of the n binary variables of the function, along with a column for the corresponding value of F. the truth table for the above function is as follows

X	Y	F
0	0	0
0	1	0
1	0	1
1	1	0

From the table, we can see that the function is equal to 1 only if $x = 1$ and $y = 0$.

The third representation of Boolean functions is by logic diagrams. In a logic diagram, binary variables are represented as inputs, binary operators are represented as symbolic draw for logic gates, and the function itself is represented as the output. The logic diagram representing the above function is as follows:



2.2.2 Simplification of Boolean Functions

An important issue when dealing with Boolean functions is the simplification of these functions. In this process, we try to simplify or reduce the function to obtain the same output with a simple and short form of the function. This simplification, in turn, reduces the required number of gates. The simplification of Boolean functions is done by applying the postulates and the theorems of Boolean algebra or by using Karnaugh maps (K-maps). In this lab, you will practice doing the simplification using these two methods. You will see how to implement and simplify the following Boolean function:

$$F = X'\,Y'\,Z + X'\,Y\,Z + X\,Y'$$

2.3 Equipment Required:

2.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

2.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- 1x 7404 TTL HEX INVERTER GATE
- 1x 7411 TTL TRIPLE 3-input AND GATE
- 1x 7432 TTL QUAD 2-input OR GATE
- 3x Toggle Switches
- 1x Carbon-film Resistor (470Ω)
- 1x LED

2.4 Prelab:

It is recommended that you review how to simplify Boolean functions using Boolean algebra and K-maps.

2.5 Software Simulation:

2.5.1 Login into one of the PCs in the lab using your user name and password.

2.5.2 Double click on the *logisim-win-2.7.1* icon on the desktop of the computer.

2.5.3 In the area below, use basic gates to draw the logic diagram of the Boolean function: F = X' Y' Z + X' Y Z + X Y'. If necessary, have your TA to check your circuit

2.5.4 As you did in Lab (1), enter your circuit into Logisim

2.5.5 Use the *poke tool (👆)* to simulate your circuit and fill in the following truth table.

X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2.5.6 Save you circuit under the name *Lab2original*. You will need it again during this lab.

2.5.7 Now, simplify the function using Boolean algebra. Show all the simplification steps and write the most simplified form in the following area. Have your TA to check your work.

2.5.8 Simplify the function using the K-map and ensure that you get the same simplified expression as the one resulted from the Boolean algebra simplification. Use the following area to draw your K-map and write the most simplified expression

2.5.9 In the area below, use basic gates to draw the logic diagram of the simplified expression.

2.5.10 Start a new Logisim project and enter the circuit you drew for the simplified expression into it.

2.5.11 Use the *poke tool* (👆) to simulate your simplified circuit and fill in the following truth table. The table should be identical to the one of the original circuit, in step 2.5.5

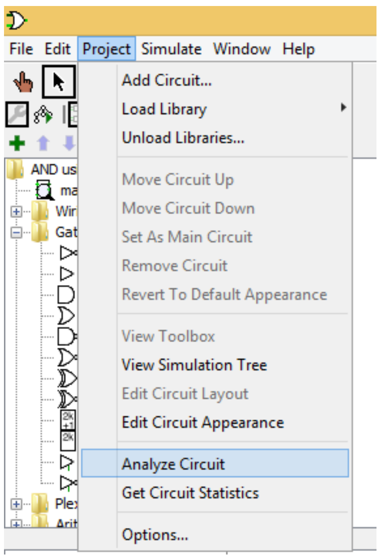
X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2.5.12 Count the number of gates in your original circuit and the number of gates in the simplified one. Consider the three-input AND gate to be equivalent to 2 two-input AND gates. How many gates could you save by simplification? …………………

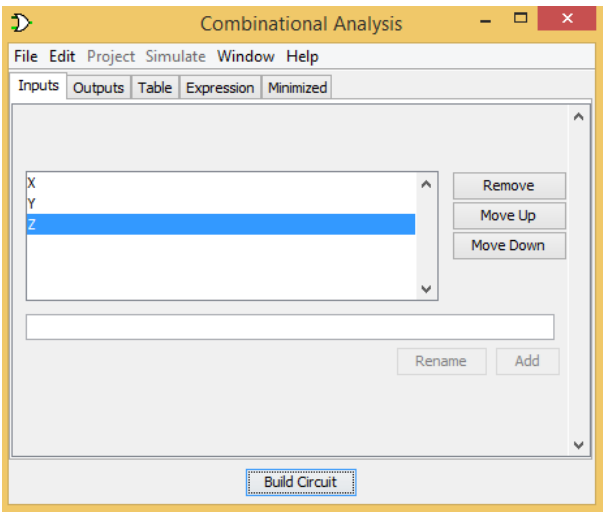
2.5.13 Now, let's use Logisim to carry out the simplification you did manually on steps 2.5.7 and 2.5.8

2.5.14 From the *File menu*, select *Open Recent* and click on your original file, Lab2original.

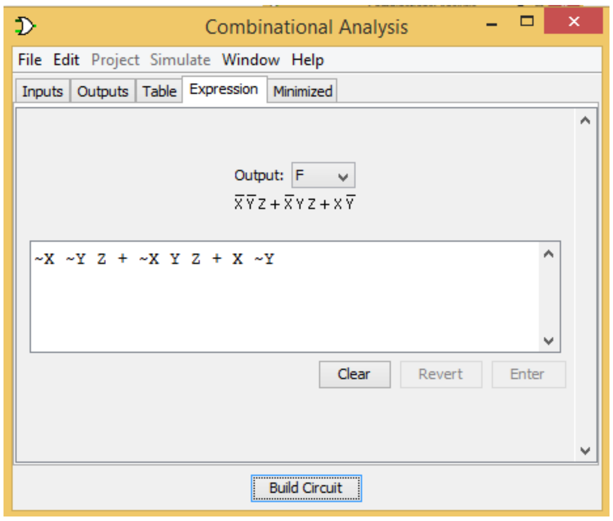
2.5.15 From the *Project menu*, click on *Analyze Circuit*



2.5.16 The <u>*Combinational Analysis*</u> window will show up


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
X
Y
Z
Remove
Move Up
Move Down
Rename
Add
Build Circuit

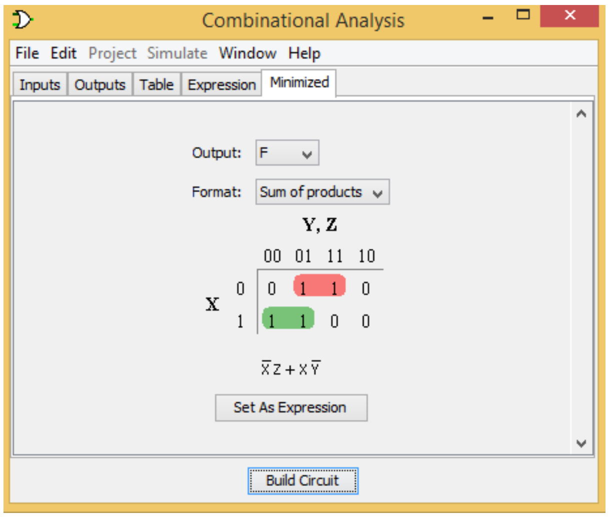
2.5.17 Go to the *Expression* tab, you will see your original expression as in point 2.5.3.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Output: F
Clear
Revert
Enter
Build Circuit

$\overline{X}\,\overline{Y}\,Z + \overline{X}\,Y\,Z + X\,\overline{Y}$

```
~X ~Y Z + ~X Y Z + X ~Y
```

2.5.18 Go to the *Minimized* tab, you will see the K-map simplification of the Boolean function and the simplified expression. It should be identical to the one you obtained in steps 2.5.7 and 2.5.8.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Set As Expression
Build Circuit

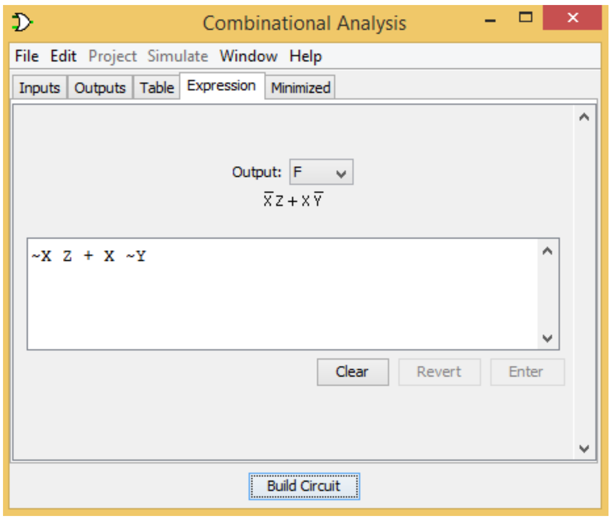
Output: F

Format: Sum of products

X \ Y, Z	00	01	11	10
0	0	1	1	0
1	1	1	0	0

$\overline{X}Z + X\overline{Y}$

2.5.19 Click on the *Set As Expression* button on the *Combinational Analysis* window. The simplified expression will be used instead of the original one. You could check that by going to the *Expression* tab again

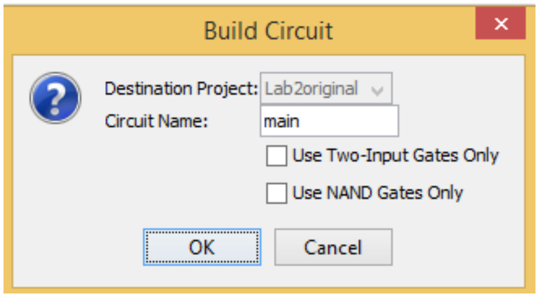

Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Clear
Revert
Enter
Build Circuit

Output: F

$\overline{X}Z + X\overline{Y}$

```
~X Z + X ~Y
```

2.5.20 Click on the *Build Circuit* button, at the bottom of the *Combinational Analysis* window. The *Build Circuit* window will show up.



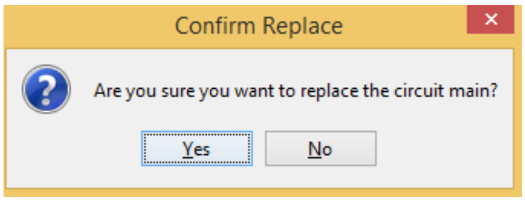
Destination Project: Lab2original

Circuit Name: main

☐ Use Two-Input Gates Only

☐ Use NAND Gates Only

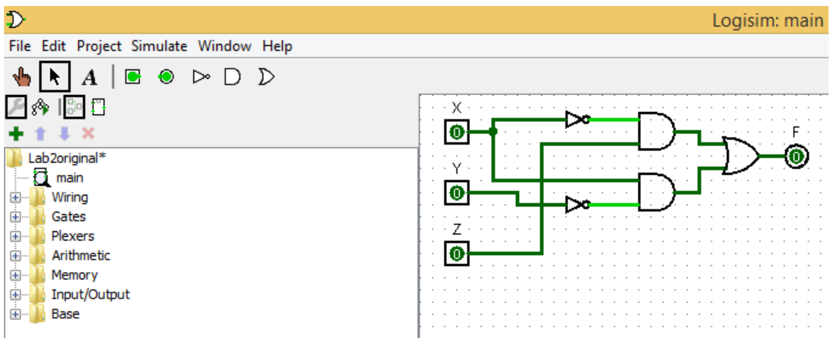
2.5.21 Click OK, the *Confirm Replace* window will show up



Are you sure you want to replace the circuit main?

2.5.22 Click Yes on the *Confirm Replace* Window.

2.5.23 Go to Logisim main window. You will find that the original circuit is replaced by the simplified one. Ensure that it is the same as the simplified circuit you generated before.



2.5.24 After you are done, remember to back up your work onto a USB drive or to email it to yourself for future usage.

2.6 Hardware Implementation:

2.6.1 Collect the components necessary to implement the original circuit, as listed in Section 2.3. Notice that you have drawn the logic diagram of the circuit in step 2.5.3.

2.6.2 As you did in Lab (1), plug each IC chip into the IC Training Kit.

2.6.3 Check the datasheet of each IC and according to the pin-out diagram of the IC chips, wire the logic circuit. Insert your wires carefully and avoid any loose connection.

2.6.4 Be sure to connect the supply voltage and ground lines (5V and 0V) to all chips.

2.6.5 Once all connections have been done, turn on the power switch of the IC Training Kit.

2.6.6 Use the three switches for X, Y, and Z to enter all possible input combinations in order to fill in the truth table below:

X	Y	Z	X' Y' Z	X' Y Z	X Y'	F
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

2.6.7 After you have completely done with the original circuit, use necessary IC chips to construct the simplified circuit. Notice that the figure of step 2.5.23 shows the logic diagram of the simplified circuit

2.6.8 Once all connections have been done, turn on the power switch of the IC Training Kit.

2.6.9 Use the three switches for X, Y, and Z to enter all possible input combinations in order to fill in the truth table below. Ensure that the truth table of the simplified circuit is identical to that of the original one.

X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2.6.10 After finishing the experiment, turn the power OFF, disconnect the wires, take out all IC chips from the trainer, put back everything you have used and clean your table.

2.7 Questions: Practice Simplification of Boolean Functions

For each of the following Boolean functions it is required to do all of the following and include your work onto the report.

- Draw the logic diagram corresponding to each function
- Simplify the function manually using Boolean algebra and K-map
- Enter the original logic diagram onto Logisim. Then, using the Combinational Analysis window, ensure that the simplification you did is correct. Snapshots from Logisim should be included in your report

$F = A' B' C' + A' B' C + A B' C$

$F = A' (B C + B C') + A (B C + B C')$

Week (7)

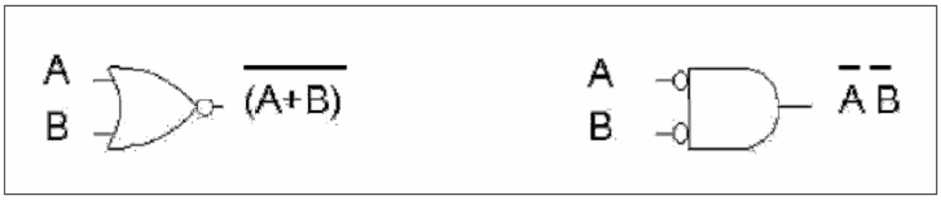
Experiment 3: NOR and NAND Implementation of Logic Functions

3.1 Objectives:

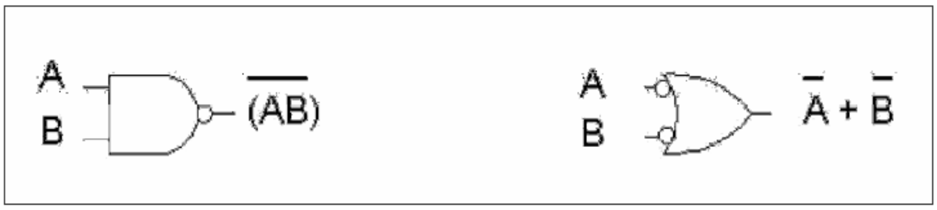
To demonstrate the implementation of digital systems using NOR and NAND gates.

3.2 Background Information:

NOR and NAND gates are said to be universal gates because any digital system could be implemented using only one type of these gates. As these gates are easier to fabricate with electronic components, digital circuits are frequently constructed with only NOR or only NAND gates. Because of the importance of NOR and NAND gates in the design of digital circuits, rules and procedures have been developed for the conversion from Boolean functions in terms of AND, OR, and NOT into equivalent NOR or NAND-based logic diagrams. From DeMorgan theorem, we can see two expressions and symbols of NOR and NAND gates as follows:



Two expressions and symbols for NOR gate



Two expressions and symbols for NAND gate

3.3 Equipment Required:

3.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

3.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- (1) 7400 TTL QUAD NAND GATE
- (1) 7402 TTL QUAD NOR GATE
- 2x Toggle Switches
- 1x Carbon-film Resistor (470Ω)
- 1x LED

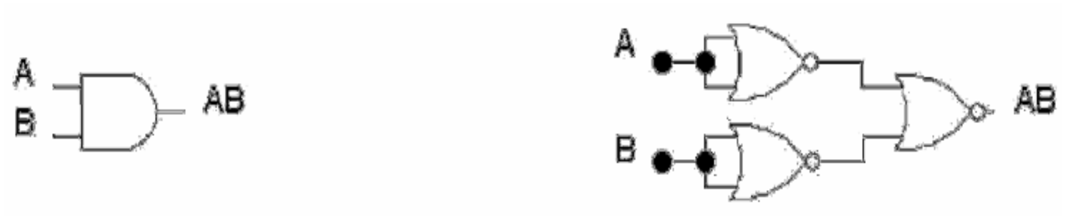
3.4 Prelab:

It is recommended that you review how to implement logic functions using the universal NOR and NAND gates

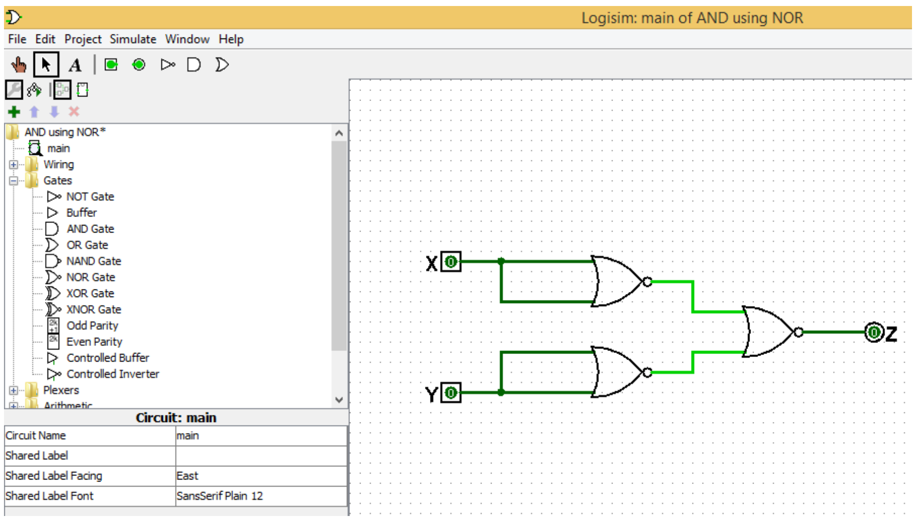
3.5 Implementation of Basic Gates using NOR gate:

3.5.1 Software Simulation

3.5.1.1 Let's start by implementing the basic AND using universal NOR gates. The NOR implementation of the basic AND gate is shown in the figure below



3.5.1.2 As you did in Lab (1), start a new Logisim project and enter the NOR-based circuit, to the right of the above figure, into it.



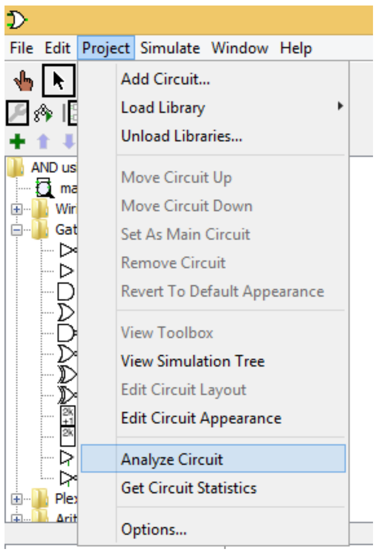
Circuit: main

Circuit Name	main
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

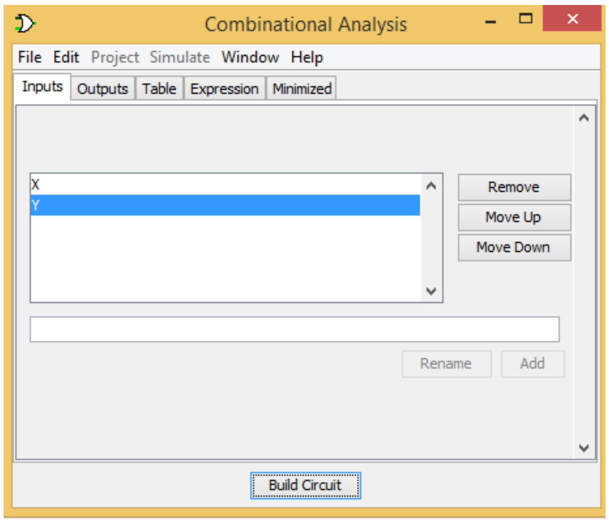
3.5.1.3 Use the *poke tool(* *)* to simulate the circuit and fill in the following truth table. The truth table should be identical to that of the basic AND gate.

X	Y	Z
0	0	
0	1	
1	0	
1	1	

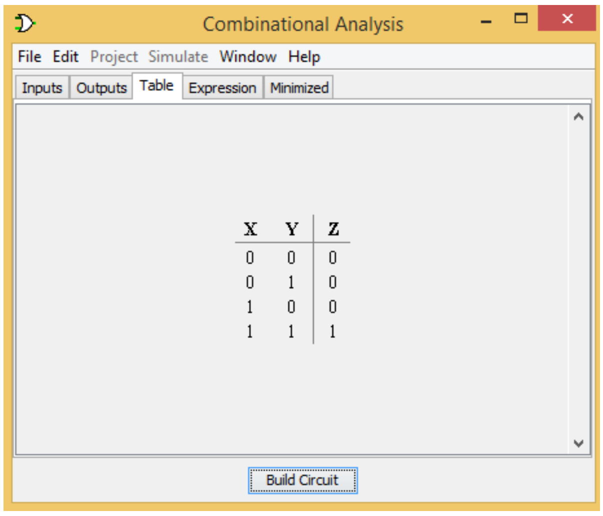
3.5.1.4 Now, let's see another method to generate the truth table of any circuit other than the *poke tool*.
3.5.1.5 From the *Project menu*, click on *Analyze Circuit*



3.5.1.6 The *Combinational Analysis* window will show up


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
X
Y
Remove
Move Up
Move Down
Rename
Add
Build Circuit

3.5.1.7 Click on the *Table* tab. The truth table will show up and it should be identical to the one you generated using the *poke tool*. The table ensures that the NOR implementation of the AND gate is correct.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Build Circuit

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

3.5.1.8 Remember to save your work for future usage.

3.5.2 <u>Hardware Implementation</u>

3.5.2.1 As you did in Lab (1), use the 7402 IC chip to implement the NOR-based circuit of the basic AND gate. Remember to check the datasheet of the IC chip before connecting your wires.

3.5.2.2 Turn your power ON an Fill in the following truth table.

X	Y	Z
0	0	
0	1	
1	0	
1	1	

3.5.2.3 The truth table should be identical to the one you got from the simulation and the one of the basic AND gate. This ensures that basic AND could be implemented using only NOR gates.

3.5.3 Now, come up with your implementation of the basic NOT and OR gates using the universal NOR gate. Draw the circuits in the area below, enter them into Logisim, and build them using the required IC chips. Generate the truth table and ensure that your circuits are correct. Have your TA to check your work.

3.6 *Implementation of Logic Functions using NOR gates:*

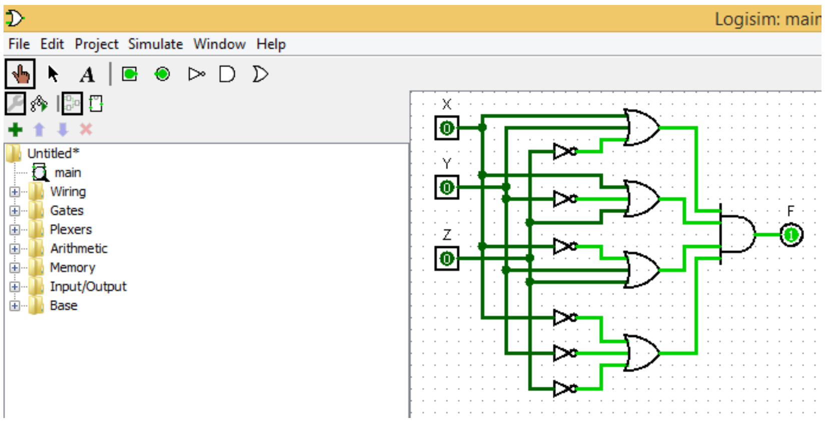
3.6.1 Software Simulation

3.6.1.1 In order to implement any logic function using NOR gates, the function should be in the Product of Sum (POS) representation. POS could be obtained by applying DeMorgan Theorem on the Sum of Product (SOP) representation.

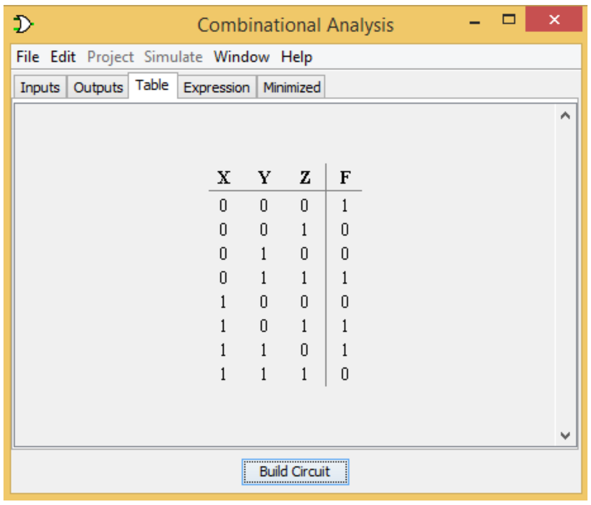
3.6.1.2 Consider the following POS function:

$$F = (X + Y + Z')(X + Y' + Z)(X' + Y + Z)(X' + Y' + Z')$$

3.6.1.3 Create a new Logisim project and construct the above function using OR-AND gates.



3.6.1.4 Use the *poke tool* or the *Combinational Analysis* window to generate the truth table of the circuit.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Build Circuit

X	Y	Z	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

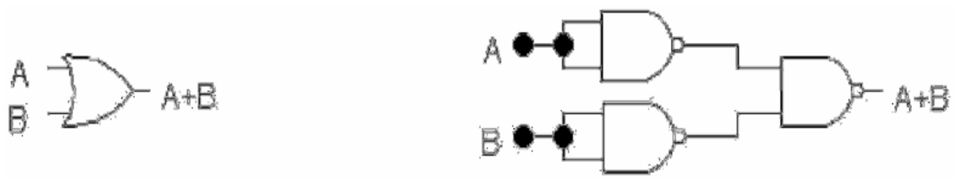
3.6.1.5 Now, Construct the same function using only NOR gates. Draw your circuit in the area below and have your TA to check it.

3.6.1.6 Enter the NOR-based circuit you constructed into Logisim and generate its truth table. The generated table should be identical to that of the OR-AND implementation. This ensures that any logic function could be implemented using NOR gates only.

3.7 Implementation of Basic Gates using NAND gate:

3.7.1 Software Simulation

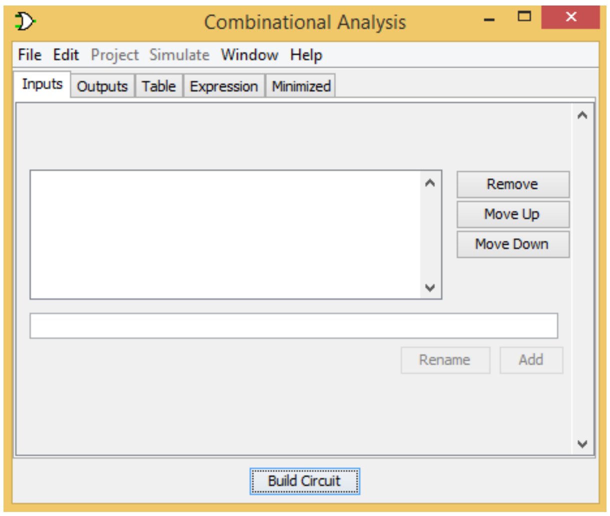
3.7.1.1 Let's start by implementing the basic OR using universal NAND gates. The NAND implementation of the basic OR gate is shown in the figure below



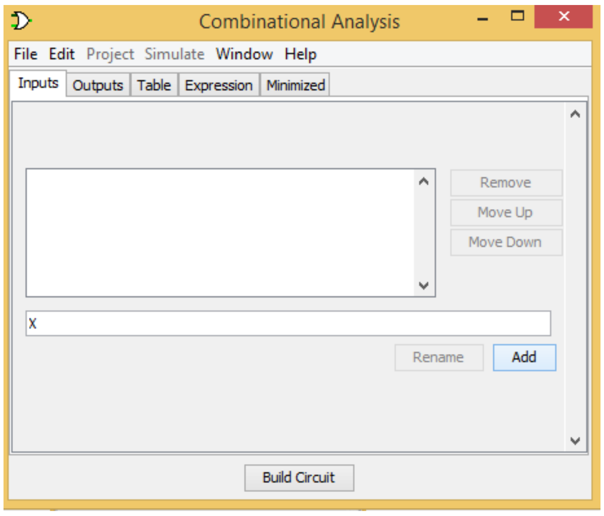
3.7.1.2 You could enter the NAND-based circuit into Logisim and simulate it, as you did with the NOR-based circuit, in Section 3.5. However, let's see an another method to enter logic expressions and implement them using universal NAND gates.

3.7.1.3 Start a new Logisim project, simply by clicking Ctrl + N.

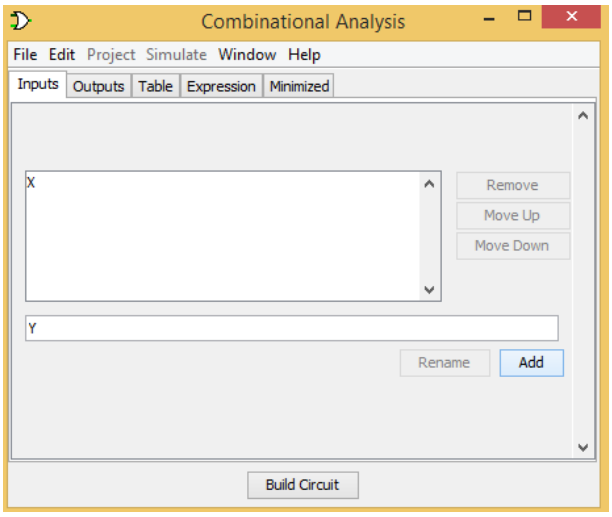
3.7.1.4 From the <u>*Project menu*</u>, click *Analyze Circuit*. The <u>*Combinational Analysis*</u> window will show up.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Remove
Move Up
Move Down
Rename
Add
Build Circuit

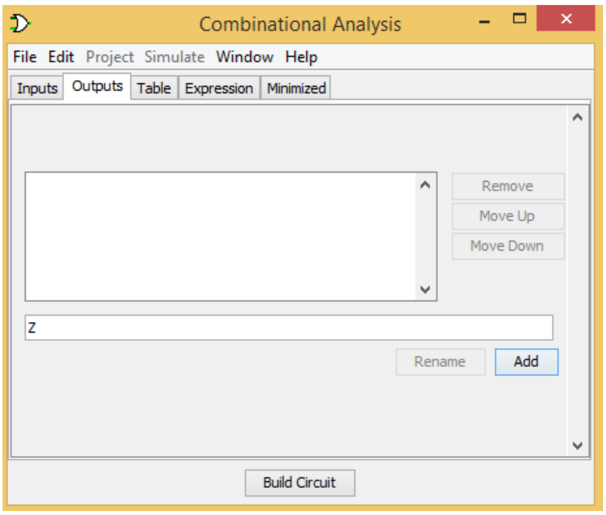
3.7.1.5 Ensure that you are in the *Inputs* tab. Then, enter X in the bottom Edit Box and click on the <u>*Add*</u> button.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Remove
Move Up
Move Down
X
Rename
Add
Build Circuit

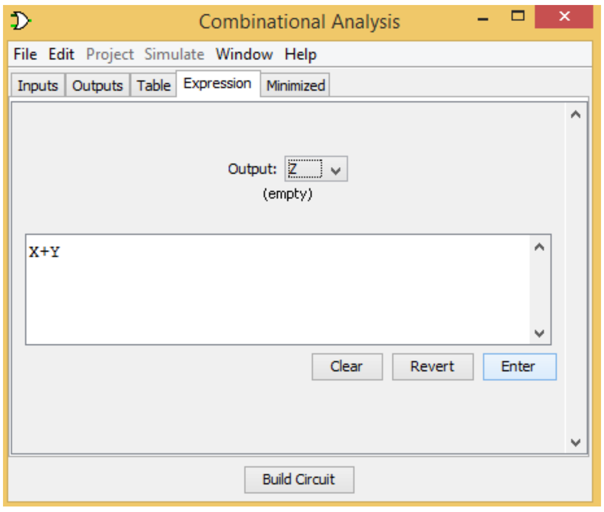
3.7.1.6 Variable X will be added to the upper Edit Box as an input. Similarly, add Y as another input.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
X
Remove
Move Up
Move Down
Y
Rename
Add
Build Circuit

3.7.1.7 Go to the *Outputs* tab, and add Z as an output of your circuit


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Remove
Move Up
Move Down
Z
Rename
Add
Build Circuit

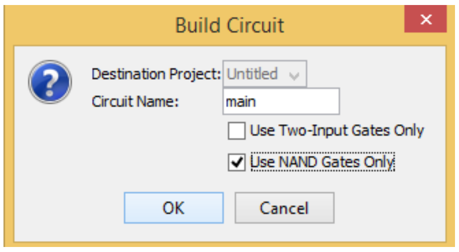
3.7.1.8 Go to the *Expression* tab. Write the basic OR gate expression (i.e., X+Y) into the Edit Box and click on the *Enter* button.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Clear Revert Enter
Build Circuit

Output: Z
(empty)

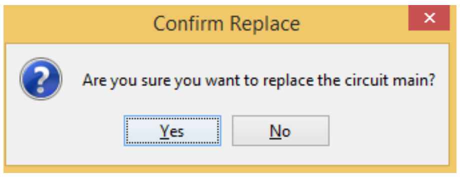
```
X+Y
```

3.7.1.9 Click on the *Build Circuit* button, at the bottom of the *Combinational Analysis* window. The *Build Circuit* window will show up. Check the box beside *"Use NAND Gates Only"*. Then, click *OK*.



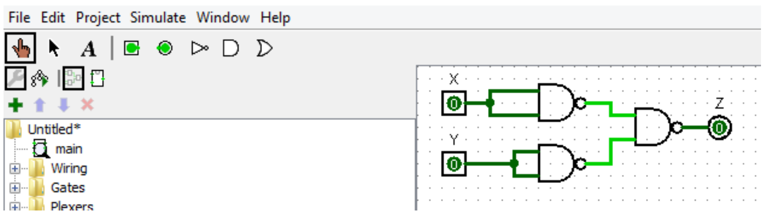
Destination Project: Untitled
Circuit Name: main
☐ Use Two-Input Gates Only
☑ Use NAND Gates Only

3.7.1.10 Click Yes on the *Confirm replace* Window.

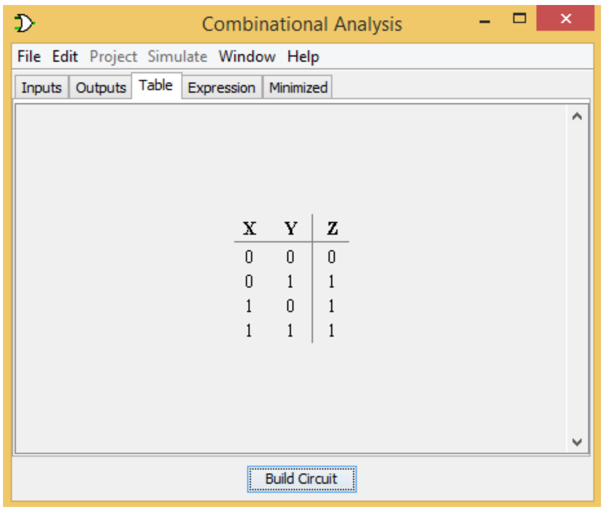


Are you sure you want to replace the circuit main?

3.7.1.11 Go to Logisim main window. The NAND-implementation of the basic OR gate will be automatically generated for you.



3.7.1.12 Ensure that the circuit is identical to the NAND implementation of the OR gate, as in point 3.7.1.1. Generate the truth table, using the *poke tool* or the *Combinational Analysis* window, and ensure that it is identical to that of the OR gate.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Build Circuit

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

3.7.1.13 Remember to save your work for future usage

3.7.2 <u>Hardware Implementation</u>

3.7.2.1 As you did in Lab (1), use the 7400 IC chip to implement the NAND-based circuit of the basic OR gate. Remember to check the datasheet of the IC chip before connecting your wires.

3.7.2.2 Turn your power ON an Fill in the following truth table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

3.7.2.3 The truth table should be identical to the one you got from the simulation and the one of the basic OR gate. This ensures that basic OR could be implemented using only NAND gates.

3.7.3 Now, come up with your implementation of the basic NOT and AND gates using the universal NAND gate. Draw the circuits in the area below, enter them into Logisim, and build them using the required IC chips. Generate the truth table and ensure that your circuits are correct. Have your TA to check your work.

3.8 Implementation of Logic Functions using NAND gates:

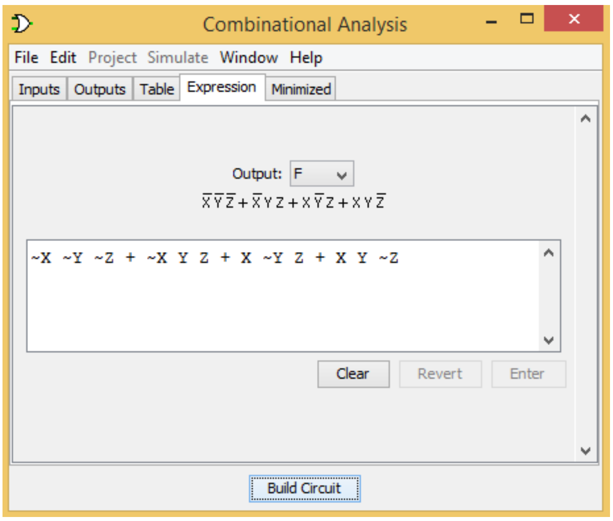
3.8.1 <u>Software Simulation</u>

3.8.1.1 In order to implement any logic function using NAND gates, the function should be in the Sum of Product (SOP) representation.

3.8.1.2 Consider the following POS function:

F = X' Y' Z' + X ' Y Z + X Y' Z +X Y Z'

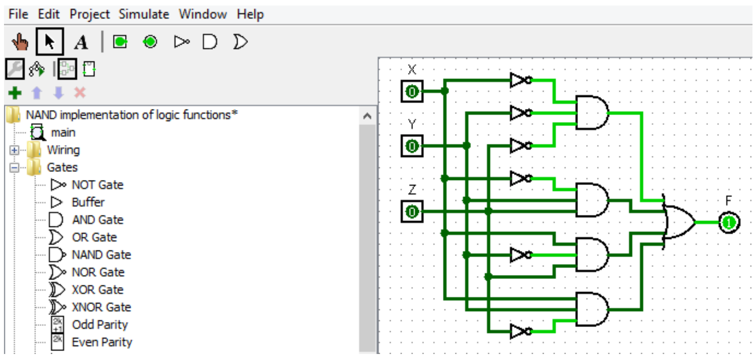
3.8.1.3 Create a new Logisim project. As you did through Subsection 3.7.1, enter the above expression using the *Combinational Analysis* window. Remember to define your inputs and outputs before writing the expression. In order to complement any Boolean variable, the character '~' should be written before the variable. Moreover, a space should be left between any two adjacent variables.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Output: F
Clear Revert Enter
Build Circuit

$\overline{X}\,\overline{Y}\,\overline{Z} + \overline{X}\,Y\,Z + X\,\overline{Y}\,Z + X\,Y\,\overline{Z}$

```
~X ~Y ~Z + ~X Y Z + X ~Y Z + X Y ~Z
```

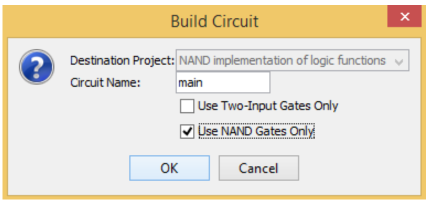
3.8.1.4 Click on the *<u>Build Circuit</u>* button, the *Build Circuit* window will appear. Click OK. The *Confirm Replace* window will appear, click Yes. The logic function will be generated using AND-OR gates.



3.8.1.5 Use the *poke tool* or the *Combinational Analysis* window to generate the truth table of the circuit.

3.8.1.6 Now, construct the same function using only NAND gates. Draw your circuit in the area below and have your TA to check it.

3.8.1.7 Now, use the *Combinational Analysis* window again to rebuild the circuit. However, in this time, check the *"Use NAND Gates Only"* box


Build Circuit

Destination Project: NAND implementation of logic functions

Circuit Name: main

☐ Use Two-Input Gates Only

☑ Use NAND Gates Only

3.8.1.8 A NAND-based implementation of the logic function is generated in Logisim main window. The implementation should be identical to the one you generate in point 3.8.1.6. Moreover, check the truth table of the NAND-based circuit. It should be also identical to that of the AND-OR representation. This ensures that any logic function could be implemented using NAND gates only.

3.9 Questions

3.9.1 Implement a NOR gate using NAND gates; then implement a NAND gate using NOR gates.

3.9.2 Convert the POS expression in Subsection 3.6.1 into its corresponding SOP expression. Implement the SOP expression using NAND gates only and simulate it using Logisim. Snapshot from Logisim should be included into your report.

3.9.3 Convert the SOP expression in Subsection 3.8.1 into its corresponding POS expression. Implement the POS expression using NOR gates only and simulate it using Logisim. Snapshot from Logisim should be included into your report.

Week (8)

Tutorial 4: Implementation and simplification of Boolean functions using K-maps

Reminder: Please, remember to print and bring the tutorial with you

Week (10)

Experiment 4: Combinational Logic Modules – Adders and Subtractors

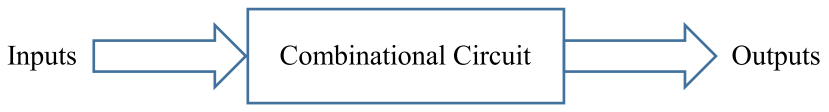
4.1 Objectives:

- To describe the difference between combinational and sequential logic circuits.
- To describe the operation and the construction of binary adders and subtractors
- To design a hierarchical combinational circuit and simulate it using Logisim.

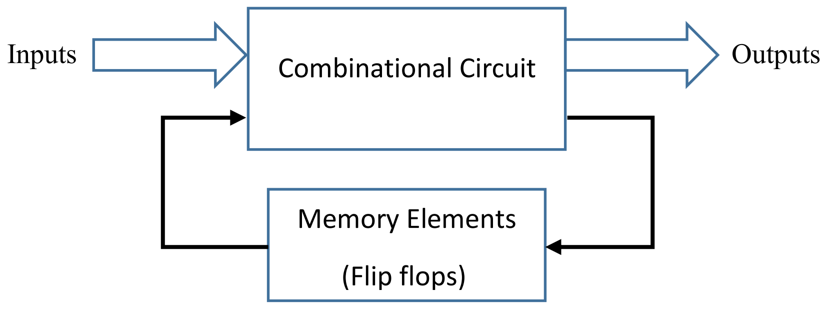
4.2 Background Information:

4.2.1 Combinational vs Sequential Circuits

Digital logic circuits can be classified into two main types: combinational and sequential. A combinational circuit is a logic circuit that is made up of combinations of logic gates. The application of inputs into combinational circuits generates the output instantaneously. Some examples of typical combinational circuits are binary adders, subtractors, comparators, decoders, encoders, multiplexers, and demultiplexers. All these digital components will be covered and discussed practically starting from this lab.



A sequential circuit, on the other hand, is made up of a combinational circuit and memory elements, called flip-flops. The outputs of the sequential circuits depend not only on the inputs but also on the output of the memory elements. Some examples of sequential circuits are counters and shift registers.



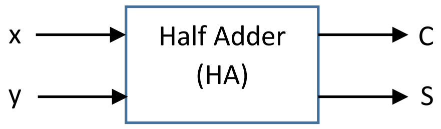
4.2.2 Hierarchical Design

Hierarchical design is a design technique in which pre-designed modules are used to build larger ones. In this lab, you will hierarchically build the full adder using half adders. The full adder will then be used to build a larger adder-subtractor circuit. Historically, in the field of digital design, Small Scale Integrated circuits (SSI) are used to build Medium Scale Integrated circuits (MSI), which in turn are used to build Large Scale Integrated circuits (LSI). LSI modules are finally used to build Very Large Scale Integrated circuits (VLSI) and Ultra Large Scale Integrated circuits (ULSI).

4.2.3 Binary Adders and Subtractors

Binary adders and subtractors are combinational circuits that can perform the operations of addition and subtraction of binary numbers, respectively. In this lab, we will study and construct various adder and subtractor circuits.

a) **Half Adder (HA) Circuit**
The combinational circuit that adds only two bits is called half adder.



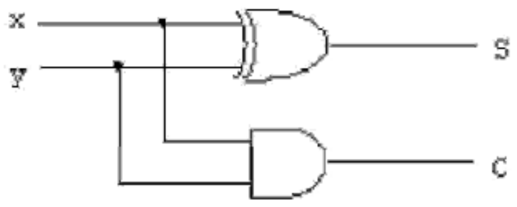
Since there are two inputs (x and y), only four possible combinations of inputs can be applied. These four possibilities, and their resulting Sum (S) and Carry (C) are shown in following truth table.

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

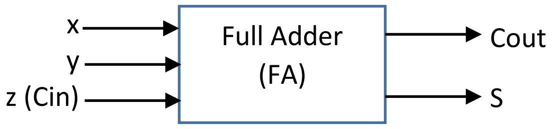
From the truth table, we could get the Boolean expression of C and S as

$$C = x.y$$
$$S = x \oplus y = x'.y + x.y'$$

Accordingly, the logic circuit of the half adder is drawn below.



b) **Full Adder (FA) Circuit**
Full Adder (FA) is a combinational circuit that adds three bits. It generates two outputs: Sum (S) and carry (C). Full adders allow for the addition of multi-bit numbers. A Designer just needs to provide a way for carries to propagate between bit positions.



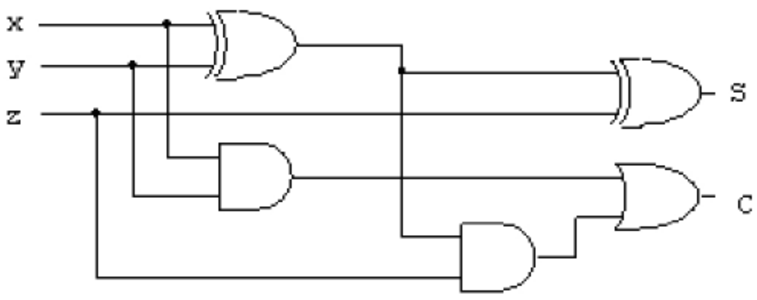
The truth table of the full adder is as follow.

X	Y	z (Cin)	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

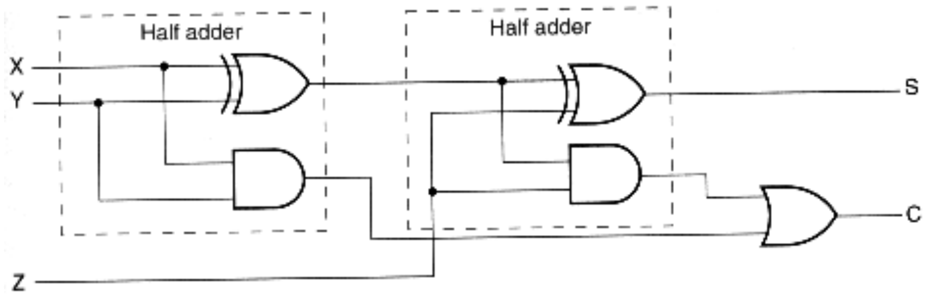
From the truth table, we could get the Boolean expression of Cout and S as

$$Cout = x'\,y\,Cin + x\,y'\,Cin + x\,y\,Cin' + x\,y\,Cin = x\,y + x\,Cin + y\,Cin$$
$$S = x'\,y'\,Cin + x'\,y\,Cin' + x\,y'\,Cin' + x\,y\,Cin = x \oplus y \oplus Cin$$

Accordingly, the logic circuit of the full adder is drawn below.



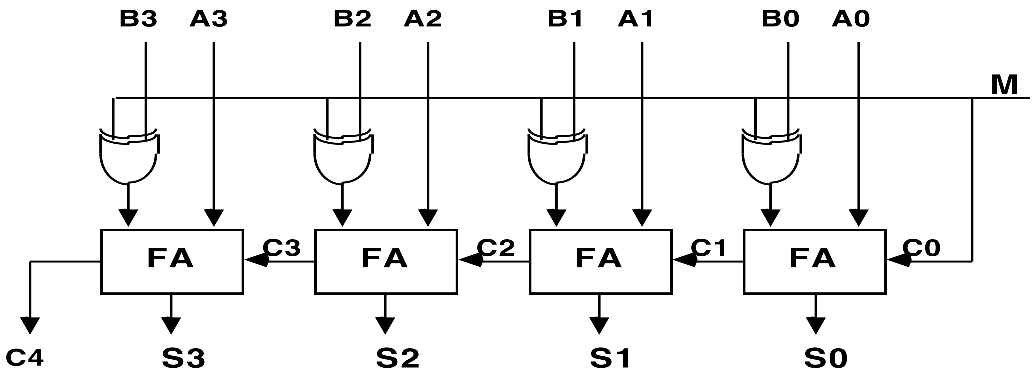
The previous figure also shows that the full adder could be built using half adders. By comparing the full adder circuit to the half adder one, you could simply implement the full adder circuit using half adders as follow



c) **Adder-Subtractor Circuit**
The subtraction of two binary numbers can be done by taking the 2's complement of the subtrahend and adding it to the minued. The 2's complement can be obtained by taking the 1's complement (i.e., inverting all the bits) and adding 1. For example, to perform A – B, we complement the four bits of B, add them to their corresponding four bits of A, and add 1. The simplest way to add a binary 1 is to insert it through the input carry.

An XOR gate could be used as an inverter if we place logic 1 at one of the inputs. This helps in getting the 1's complement of the subtrahend. Then, we add 1 to get the 2's complement. The 2's complement is finally added to the minued to get the final result of the subtraction.

The following figure shows the adder-subtractor circuit. The mode input M controls the operation. When M=0, the circuit is an adder. Whereas, when M=1, the circuit becomes a subtractor. Although the figure is for only four bits, this circuit can be cascaded for any number of inputs.



Note: During subtraction, if A ≥ B, the result is a positive number and the output carry is equal to 1. If A < B, the subtraction gives a negative difference represented in the 2's complement format and the output carry is equal to 0. Therefore, during subtraction, the carry actually represents the inversion of the borrow.

4.3 Equipment Required:

4.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

4.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- 1x 7408 QUAD 2-INPUT AND
- 1x 7432 QUAD 2-INPUT OR
- 1x 7486 QUAD 2-INPUT XOR
- 1x 74283 4-BIT FULL ADDER
- 9X Toggle switches
- 5x Carbon-film Resistors (470Ω)
- 5x LEDs

4.4 Prelab:

It is recommended that you read Section 4.2 of this lab to review how to implement adder and subtractor circuits.

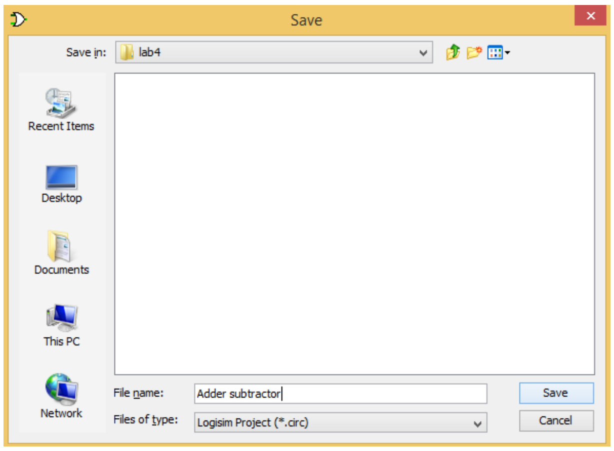
4.5 Software Simulation

4.5.1 Designing a Half Adder using Basic Gates

4.5.1.1 Login into one of the PCs in the lab using your user name and password.

4.5.1.2 Double click on the *logisim-win-2.7.1* icon on the desktop of the computer. Logisim will be launched and a new project would be created for you.

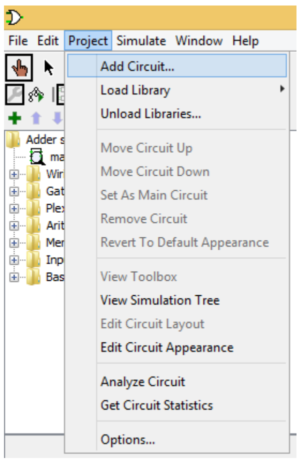
4.5.1.3 Save the project by clicking Ctrl+S. The *Save* window will appear. As you will do hierarchical design in this lab, the project should take the name of the top level. Therefore, enter *Adder subtractor* in the File name and click Save.



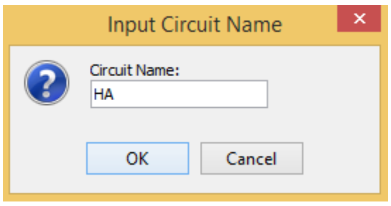
File name: Adder subtractor

Files of type: Logisim Project (*.circ)

4.5.1.4 Now, let's start to design the most inner module, the Half Adder (HA). From the *project menu*, click on *Add Circuit*.

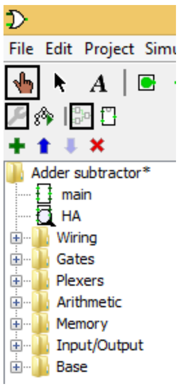


4.5.1.5 The *Input Circuit Name* window will appear. Enter *HA* as the name of the circuit and click OK

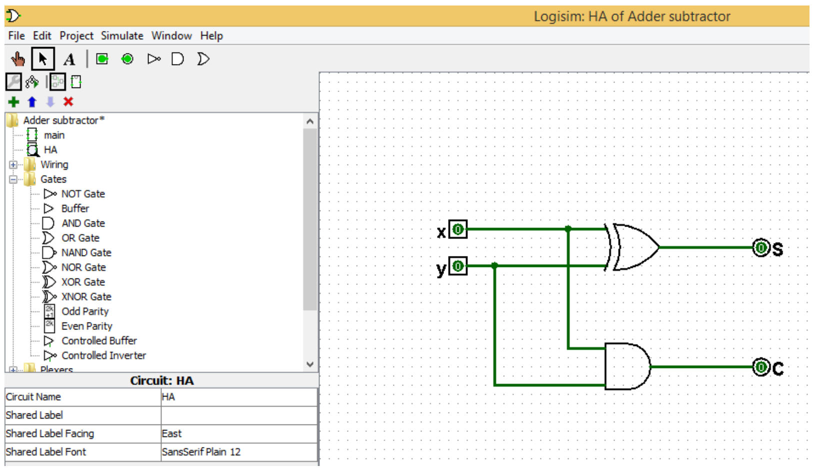


Circuit Name: HA

4.5.1.6 A new module named HA is created for you and added to the *explorer pane*. The magnifying glass over the HA indicates that it is the module that is currently viewed on the *canvas*.



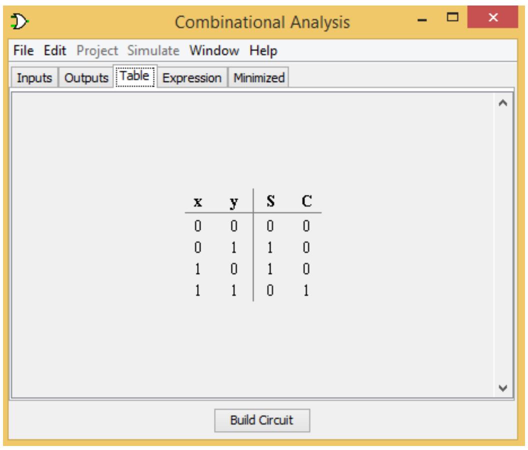
4.5.1.7 As you did in previous labs, enter the circuit of the half adder, as shown in subsection 4.2.3.



Circuit: HA

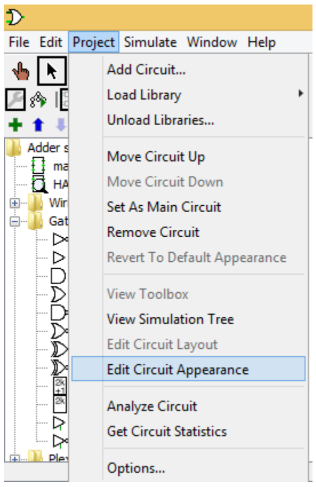
Circuit Name	HA
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

4.5.1.8 Check the operation of the HA module, either by using the *poke tool* or the *combinational analysis* window. The truth table should agree with the one in subsection 4.2.3.

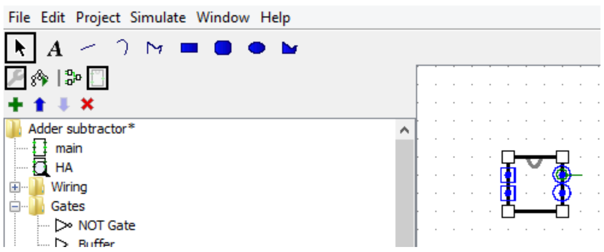


x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

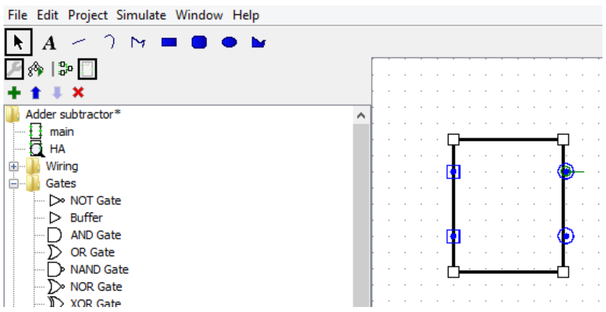
4.5.1.9 Before using the half adder hierarchically in larger designs, let's adjust it appearance first. From the <u>*project menu*</u>, click on *Edit Circuit Appearance*.



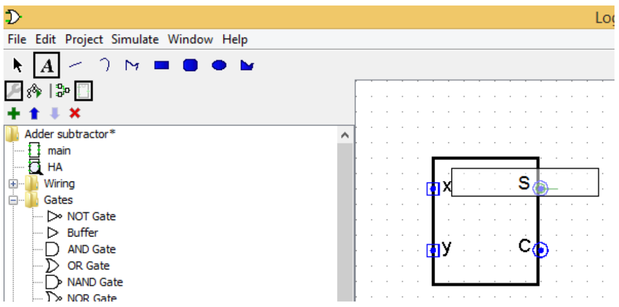
4.5.1.10 The module of the half adder will appear as a rectangle with two inputs and two outputs.



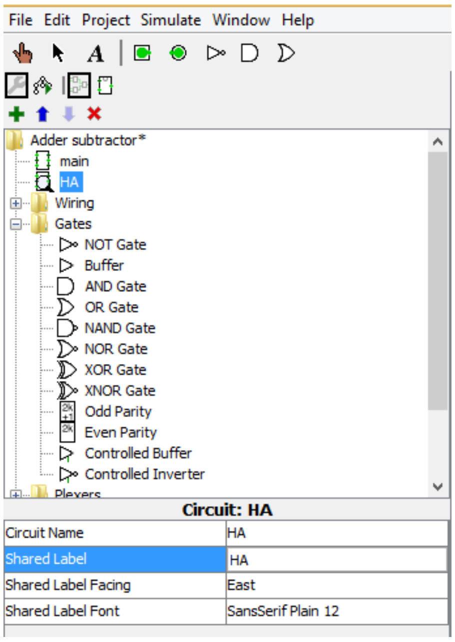
4.5.1.11 Enlarge the module and move input and output ports to proper locations



4.5.1.12 From the *toolbar*, click on the *Add text button* (**A**). Then, click beside the two inputs and name them x and y. Similarly, name the two outputs S and C, as shown below.



4.5.1.13 Finally, click on the HA module on the *explorer pane*. Then, in the *attribute table*, write HA beside the <u>*Shared Label*</u> attribute. This would be the name of the module when used in larger designs. Also, change the <u>*Shared Label Facing*</u> attribute to North.



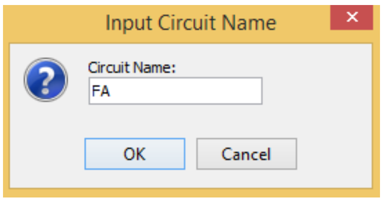
Circuit: HA

Circuit Name	HA
Shared Label	HA
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

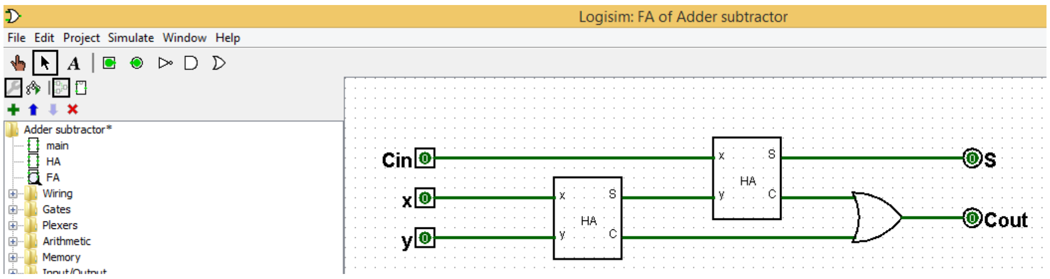
4.5.1.14 You are completely done with the half adder. You are ready to use it to build the full adder circuit. Before doing that, remember to save your work by clicking Ctrl+S.

4.5.2 <u>Designing a Full Adder using Half Adders</u>

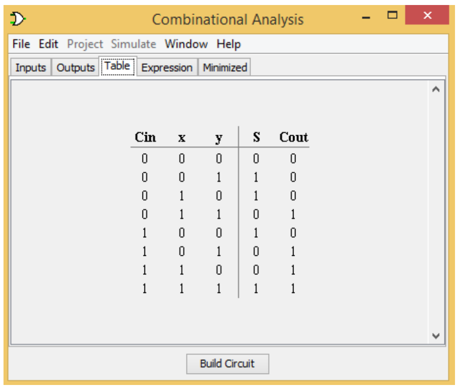
4.5.2.1 From the <u>*Project menu*</u>, click on *Add Circuit*. In the *Input Circuit Name* window, name the new circuit FA and click OK.



4.5.2.2 The new module is created for you in the *explorer pane* and an empty sheet is opened in the *canvas* for it.
4.5.2.3 Now, let's build the FA using two HAs, as discussed in subsection 4.2.3
4.5.2.4 From the <u>*explorer pane*</u>, click on the *HA* module and insert two of them into the *canvas.*
4.5.2.5 From the <u>*toolbar*</u>, click on the OR gate button and insert one of it into the *canvas*. Change the *Number of Inputs* attribute to 2.
4.5.2.6 Wire the circuit, according to the circuit diagram in subsection 4.2.3 and as shown below.

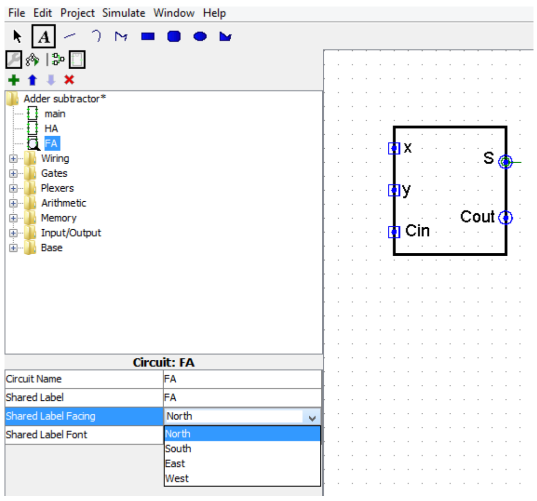


4.5.2.7 Check the operation of the FA module, either by using the *poke tool* or the *combinational analysis* window. The truth table should agree with the one in subsection 4.2.3



Cin	x	y	S	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

4.5.2.8 As you did in from step 4.5.1.9 to step 4.5.1.13, edit and adjust the appearance of the FA module.



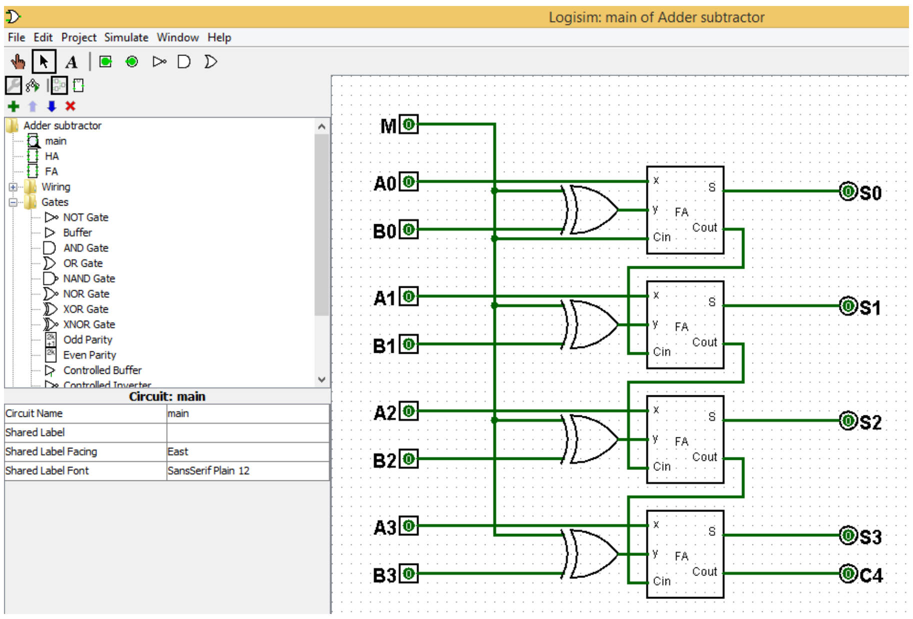
Circuit: FA	
Circuit Name	FA
Shared Label	FA
Shared Label Facing	North
Shared Label Font	

4.5.2.9 Now, you successfully build the FA using HAs. Remember to save your work before hierarchically use the FA to build the adder subtractor circuit.

4.5.3 <u>Designing an Adder Subtractor using Half Adders and XORs</u>

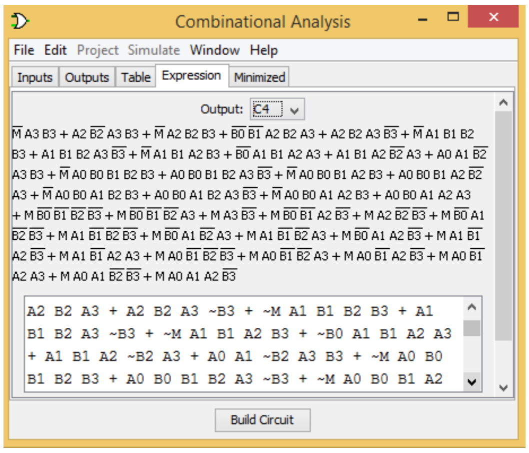
4.5.3.1 Now, we reach the top level module, the adder subtractor. Therefore, in the *explorer pane*, double click on the main file.

4.5.3.2 Use four FAs and four XORs to build the adder subtractor circuit, as discussed in subsection 4.2.3.



Circuit: main	
Circuit Name	main
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

4.5.3.3 To understand the benefit and the power of using the hierarchical design technique. Imagine building the adder subtractor circuit using the traditional method of building the truth table and minimizing the logic expression of the outputs. The 9-input truth table would have 512 rows and the logic expression of the output would be very hard to minimize. In order to visualize this complexity, use the *combinational analysis* window to see the truth table and the logic expressions of the output. The following figure gives an example of the logic expression for C4.


Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Build Circuit

Output: C4

$\overline{M}$ A3 B3 + A2 $\overline{B2}$ A3 B3 + $\overline{M}$ A2 B2 B3 + $\overline{B0}$ $\overline{B1}$ A2 B2 A3 + A2 B2 A3 $\overline{B3}$ + $\overline{M}$ A1 B1 B2
B3 + A1 B1 B2 A3 $\overline{B3}$ + $\overline{M}$ A1 B1 A2 B3 + $\overline{B0}$ A1 B1 A2 A3 + A1 B1 A2 $\overline{B2}$ A3 + A0 A1 $\overline{B2}$
A3 B3 + $\overline{M}$ A0 B0 B1 B2 B3 + A0 B0 B1 B2 A3 $\overline{B3}$ + $\overline{M}$ A0 B0 B1 A2 B3 + A0 B0 B1 A2 $\overline{B2}$
A3 + $\overline{M}$ A0 B0 A1 B2 B3 + A0 B0 A1 B2 A3 $\overline{B3}$ + $\overline{M}$ A0 B0 A1 A2 B3 + A0 B0 A1 A2 A3
+ M $\overline{B0}$ $\overline{B1}$ $\overline{B2}$ $\overline{B3}$ + M $\overline{B0}$ $\overline{B1}$ $\overline{B2}$ A3 + M A3 $\overline{B3}$ + M $\overline{B0}$ $\overline{B1}$ A2 $\overline{B3}$ + M A2 $\overline{B2}$ $\overline{B3}$ + M $\overline{B0}$ A1
$\overline{B2}$ $\overline{B3}$ + M A1 $\overline{B1}$ $\overline{B2}$ $\overline{B3}$ + M $\overline{B0}$ A1 $\overline{B2}$ A3 + M A1 $\overline{B1}$ $\overline{B2}$ A3 + M $\overline{B0}$ A1 A2 $\overline{B3}$ + M A1 $\overline{B1}$
A2 $\overline{B3}$ + M A1 $\overline{B1}$ A2 A3 + M A0 $\overline{B1}$ $\overline{B2}$ $\overline{B3}$ + M A0 $\overline{B1}$ $\overline{B2}$ A3 + M A0 $\overline{B1}$ A2 $\overline{B3}$ + M A0 $\overline{B1}$
A2 A3 + M A0 A1 $\overline{B2}$ $\overline{B3}$ + M A0 A1 A2 $\overline{B3}$

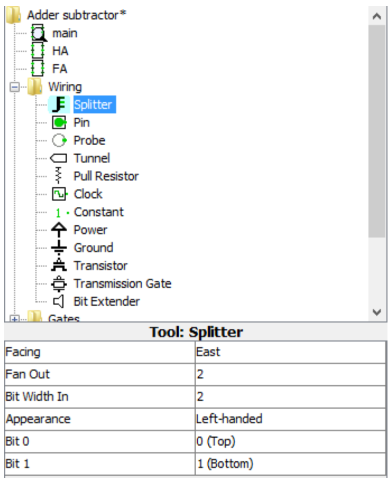
```
A2 B2 A3 + A2 B2 A3 ~B3 + ~M A1 B1 B2 B3 + A1
B1 B2 A3 ~B3 + ~M A1 B1 A2 B3 + ~B0 A1 B1 A2 A3
+ A1 B1 A2 ~B2 A3 + A0 A1 ~B2 A3 B3 + ~M A0 B0
B1 B2 B3 + A0 B0 B1 B2 A3 ~B3 + ~M A0 B0 B1 A2
```

4.5.3.4 Use the *poke tool* to simulate your circuit. Apply to following values though A's and B's and record their corresponding values of S's and C_4. Interpret your results (i.e., Explain why S and C have these values). Have you TA to check your work.

	Addition (M=0)			Subtraction (M=1)		
$A_3A_2A_1A_0$	Ten= $(1010)_2$	Eleven= $(1011)_2$	Six= $(0110)_2$	Ten= $(1010)_2$	Eleven= $(1011)_2$	Six= $(0110)_2$
$B_3B_2B_1B_0$	Six= $(0110)_2$	Fourteen =$(1110)_2$	Six= $(0110)_2$	Six= $(0110)_2$	Fourteen =$(1110)_2$	Six= $(0110)_2$
$C_4S_3S_2S_1S_0$						

4.5.3.5 Now, let's use another convenient method when dealing with multiple related bits, like A's, B's and S's. A common practice in the digital logic field is to combine these bits in a wire bundle, called the bus. So, let's see how we could create these busses in Logisim.

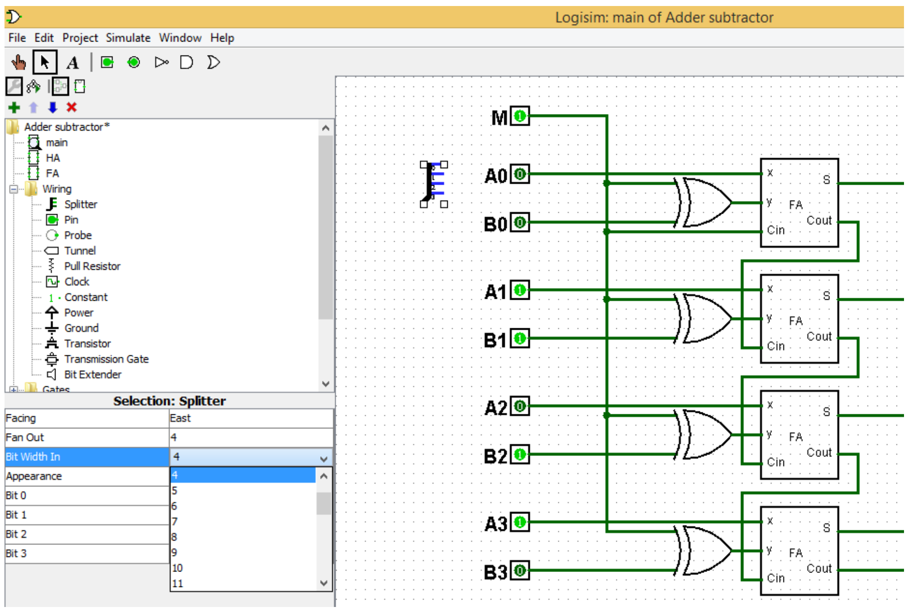
4.5.3.6 From the *Wiring* folder in the *explorer pane*, click on *Splitter* and insert it into the *canvas*.



Tool: Splitter

Facing	East
Fan Out	2
Bit Width In	2
Appearance	Left-handed
Bit 0	0 (Top)
Bit 1	1 (Bottom)

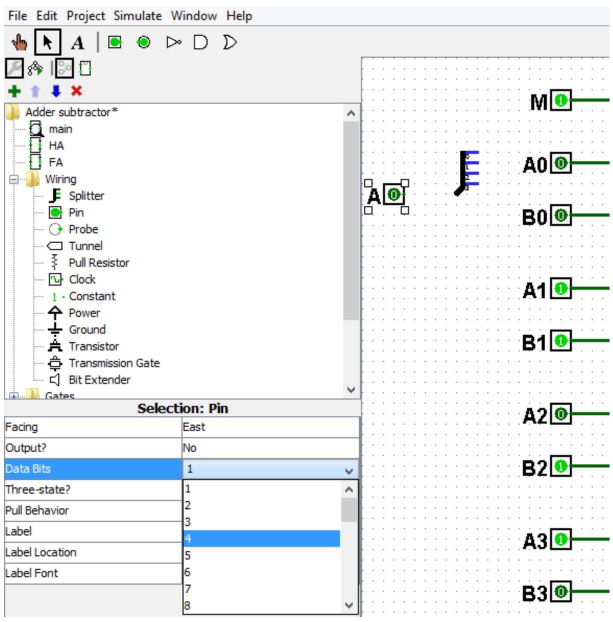
4.5.3.7 The *splitter* you just inserted will be used to create the bus of input A. Input A has four bits, $A_3A_2A_1A_0$. Therefore, in the *attribute table*, change the *Bit Width In* attribute to 4. Similarly, change the *Fan Out* attribute to 4 (i.e., the splitter would split the 4-bit bus into 4 separate wires)


Logisim: main of Adder subtractor

Selection: Splitter

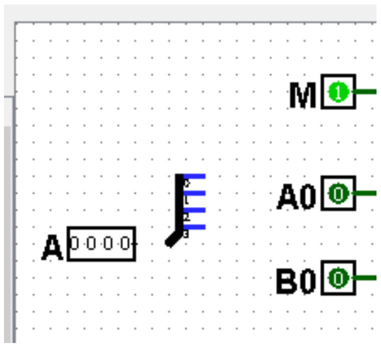
Facing	East
Fan Out	4
Bit Width In	4
Appearance	
Bit 0	
Bit 1	
Bit 2	
Bit 3	

4.5.3.8 From the <u>*toolbar*</u>, insert an *input pin*. In the *attribute table*, label it as A and change its *Data Bits* attribute to 4.


Selection: Pin

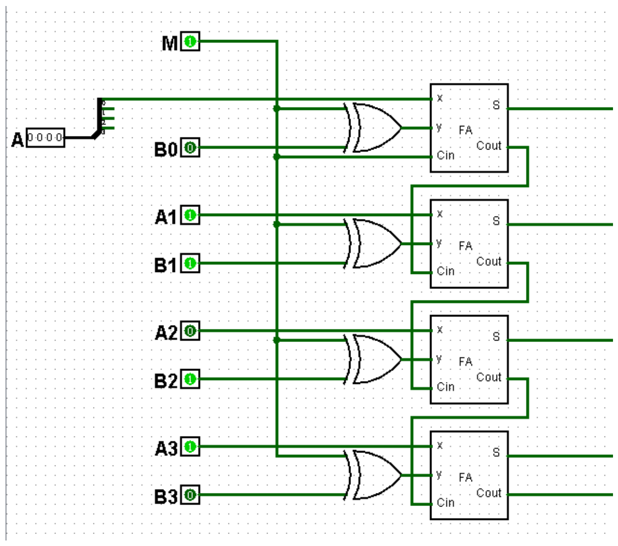
Facing	East
Output?	No
Data Bits	1
Three-state?	
Pull Behavior	
Label	
Label Location	
Label Font	

4.5.3.9 The input pin should have extended in the *canvas* to have 4 bits.

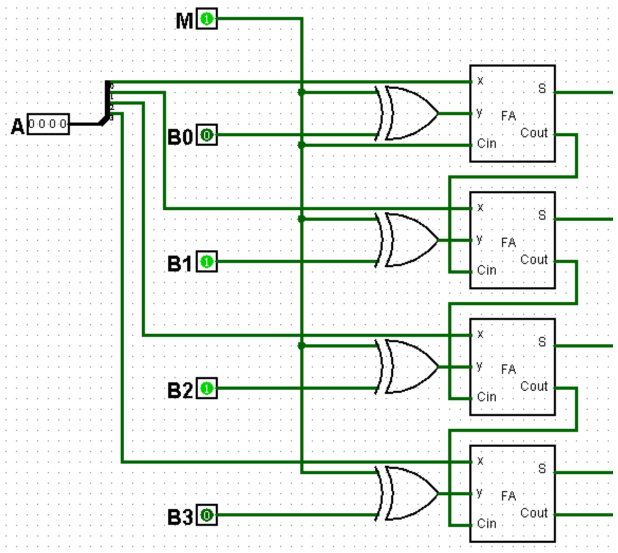


4.5.3.10 Wire the new 4-bit port, A, to the left side of the *splitter*.

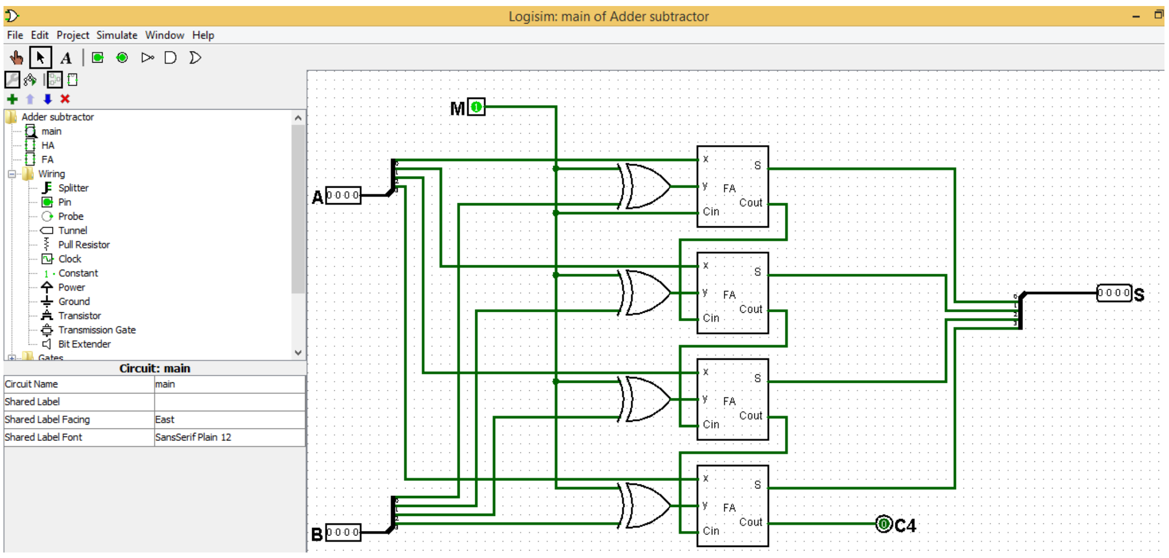
4.5.3.11 Remove the old input pin, A0, and wire bit 0 (i.e., the top right bit) of the *splitter* instead of it.



4.5.3.12 Repeat step 4.5.3.11 to remove input pins A1, A2, and A3 and connect bit 1, bit 2, and bit 3 of the *splitter* instead of them.



4.5.3.13 Finally, insert two more *splitters* for bus B and Bus S. For each bus, repeat steps 4.5.3.6 to 4.5.3.12. Change the *Facing attribute* to the S bus to West instead of East. Your final circuit should appear as follow.



Circuit: main

Circuit Name	main
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

4.5.3.14 Use the *poke tool* to practice simulating your final circuit and fill in the following table again.

	Addition (M=0)			Subtraction (M=1)		
$A_3A_2A_1A_0$	Ten= $(1010)_2$	Eleven= $(1011)_2$	Six= $(0110)_2$	Ten= $(1010)_2$	Eleven= $(1011)_2$	Six= $(0110)_2$
$B_3B_2B_1B_0$	Six= $(0110)_2$	Fourteen $=(1110)_2$	Six= $(0110)_2$	Six= $(0110)_2$	Fourteen $=(1110)_2$	Six= $(0110)_2$
$C_4S_3S_2S_1S_0$						

4.6 Hardware Implementation

4.6.1 Use basic logic gates to build the full adder, according to the circuit diagram in subsection 4.2.3.

4.6.2 Once all connections have been done, turn on the power switch of the training kit. Apply all possible input combinations and fill in the following truth table. Ensure that the truth table agrees with the one in subsection 4.2.3.

x	y	z (Cin)	Cout	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

4.6.3 Use the full adder IC chip, 74283, and the XOR IC chip, 7486, to build the circuit diagram of the adder-subtractor circuit, as shown in subsection 4.2.3.

4.6.4 Once all connections have been done, turn on the power switch of the training kit. Apply the following values through A's and B's and record their corresponding values of S's and C_4. Have your TA to check your work.

	Addition (M=0)			Subtraction (M=1)		
$A_3A_2A_1A_0$	Ten= $(1010)_2$	Eleven= $(1011)_2$	Six= $(0110)_2$	Ten= $(1010)_2$	Eleven= $(1011)_2$	Six= $(0110)_2$
$B_3B_2B_1B_0$	Six= $(0110)_2$	Fourteen $=(1110)_2$	Six= $(0110)_2$	Six= $(0110)_2$	Fourteen $=(1110)_2$	Six= $(0110)_2$
$C_4S_3S_2S_1S_0$						

4.7 Questions

- Design a 4-bit binary decrementer circuit using four half adders. Simulate the circuit using Logisim to ensure that it works properly. Include your design and snapshots from Logisim to your report.

Week (11)

Experiment 5: Combinational Logic Modules – Decoders

5.1 Objectives:

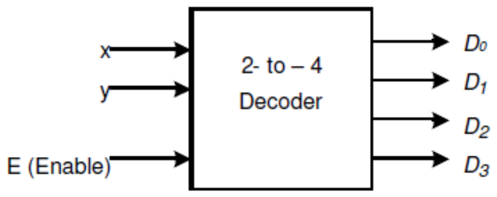
- To study the basic operation and design of decoder circuits.
- To describe the concept of active low and active high logic signals.
- To learn how to use the n-to-2^n type decoders to implement a given Boolean function.
- To learn how to use 7-segment LED display along with a seven-segment decoder to create decimal digits.

5.2 Background Information:

5.2.1 <u>Decoders</u>

Decoder is a combinational circuit that converts certain coded inputs to another coded outputs. Famous examples of decoders are binary n-to-2^n decoders and seven-segment decoders. A binary decoder has n inputs and a maximum of 2^n outputs. As we know, an n-bit binary number provides 2^n minterms. This type of decoder produces one of these 2^n minterms at the outputs, based on the input combinations. The following table shows the block diagram and the truth table of an example 2-to-4 decoder.

E	x	y	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1



From the truth table, you can observe the basic operation of n-to-2^n decoders. For each input combination, there is only one active output D_i, which is an acronym of the minterm (m_i). The module has an active high Enable. Accordingly, it will not work unless the Enable is connected to logic 1.

An example of a commercial n-to-2^n line decoder is the 74139 chip. This chip has two 2-to-4 decoders with active low enable for each.

5.2.2 <u>Implementation of Boolean Functions using Decoders</u>

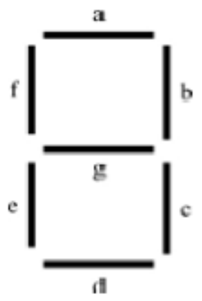
Any Boolean function can be expressed as a sum of products (or sum of minterms). A binary decoder provides all the minterms as outputs. Therefore, any Boolean function could be implemented by ORing its minterms (D_i), which are generated by a decoder. For example, consider the full-adder circuit that we discussed and designed in lab (4). The Boolean expressions for the outputs S and C_{out} (in terms of the inputs x, y, and C_{in}) are:

$$S = x'\,y'\,C_{in} + x'\,y\,C_{in}' + x\,y'\,C_{in}' + x\,y\,C_{in} = D1 + D2 + D4 + D7$$
$$C_{out} = x'\,y\,C_{in} + x\,y'\,C_{in} + x\,y\,C_{in}' + x\,y\,C_{in} = D3 + D5 + D6 + D7$$

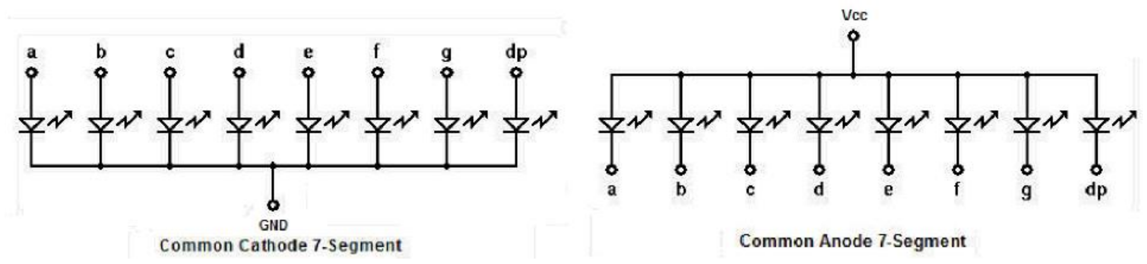
The above expressions can be implemented by ORING the appropriate combination of output minterms of a 3-to-8 decoder.

5.2.3 <u>Seven-Segment Decoder and Seven-Segment Display</u>

Another common type of decoder is the seven-segment decoder. This decoder is used along with seven-segment LED display to create a decimal or hexadecimal digits. The seven-segment LED display is commonly used as a numerical display in multimeters, calculators …etc. It contains seven independent LEDs arranged as shown in the following figure.

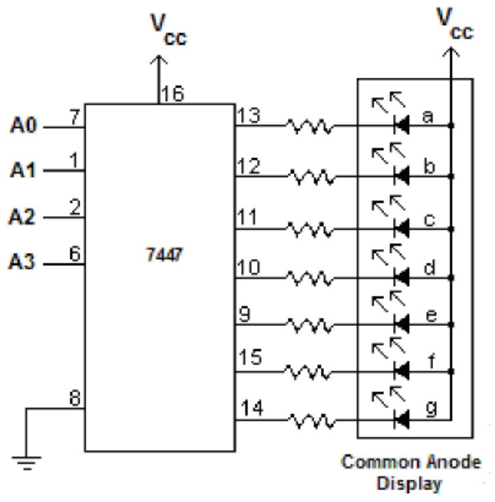


There are two main types of seven-segment LED displays: the common cathode (CC) and the common anode (CA). In the CC type, the cathodes of all segments (i.e., LEDs) are internally joined in a single node. In contrary, in the CA type, the anodes are internally joined together in a single node. The other sides of LEDs (a, b, c …etc.) are left for the designer to connect them externally. CC display is said to be active high, as a connection of logic 1 causes the designated segment to be ON. On the other side, a connection of logic 0 to a segment in the active low CA display causes that segment to turn ON. The following figure shows the internal connection of both types.



All decimal or hexadecimal numbers can be displayed by controlling the state of the appropriate segments ON or OFF. This can be done using a seven-segment decoder. A seven-segment decoder accepts four binary inputs. According to these inputs, it provides seven outputs that determines which segments of the seven-segment LED display should be ON in order to create the corresponding decimal or hexadecimal digits.

A widely used commercial common anode 7-segment decoder is the 7447 chip. This chip changes a BCD number to its corresponding CA seven-segment code. The generated seven-segment code could be used for displaying numbers from 0 to 9, based on the corresponding input BCD number. The circuit that displays any input BCD number on a common anode 7-segment display is shown in the following figure. Similar connection could be used for common cathode display. However, the 7447 decoder should be replaced by the 7448 one.



5.3 Equipment Required:

5.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

5.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- 1x 7447 BCD-TO-SEVEN SEGMENT DECODERS/DRIVERS
- 1x Common Anode (CA) Seven-Segment LED Display
- 4x Toggle Switches
- 7x Carbon-film Resistors (470Ω)

5.4 *Prelab:*

It is recommended that you read Section 5.2 of this lab in order to review binary decoder, 7-segment decoder, and 7-segment display.

5.5 *Software Simulation*

5.5.1 Implementation of Boolean Functions using Decoders

5.5.1.1 Consider the following Boolean function. It is required to implement it using 3x8 decoder.

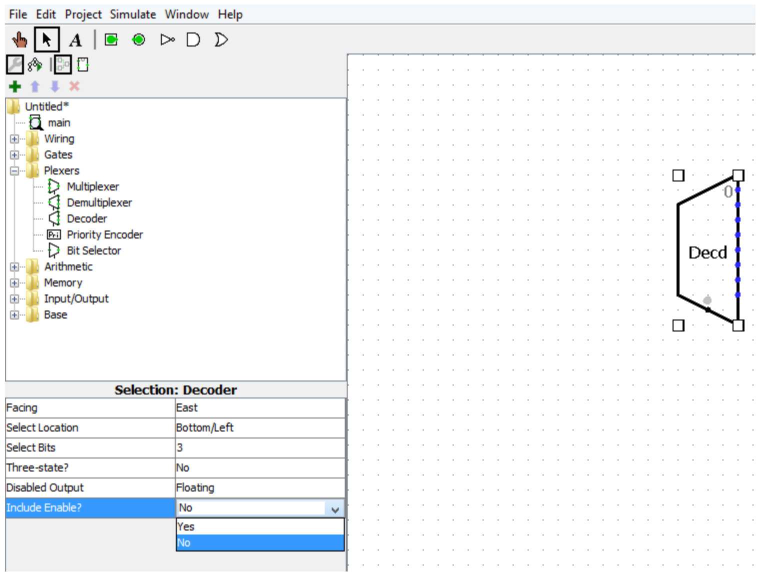
$$F(A, B, C) = A'B'C + AB'C' + ABC$$

5.5.1.2 Draw the circuit that implements the Boolean function in the area below. If necessary, have your TA to check your implementation.

5.5.1.3 Start a new Logisim project.

5.5.1.4 From the *Plexers* folder, select a decoder and insert in onto the *canvas*

5.5.1.5 In the *attribute table*, change the *Select Bits* attribute to 3 and the *Include Enable* to No



Selection: Decoder

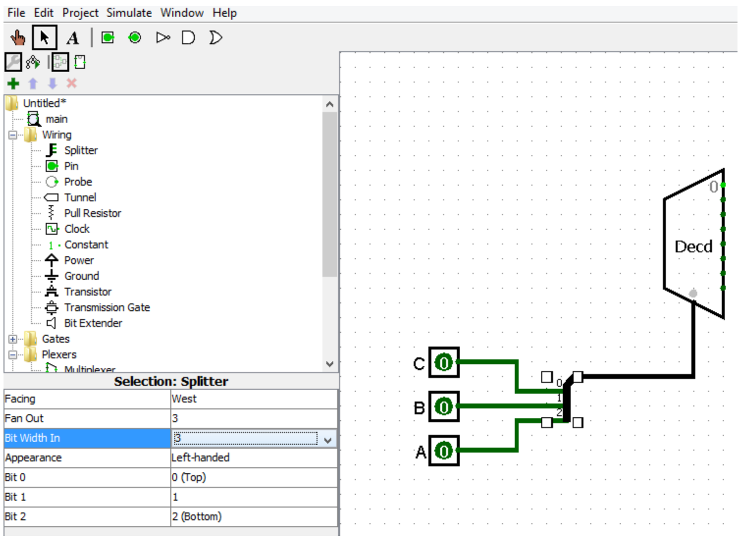
Facing	East
Select Location	Bottom/Left
Select Bits	3
Three-state?	No
Disabled Output	Floating
Include Enable?	No

Yes [✓ No]

5.5.1.6 Insert three *input pins* and name them A, B, and C, respectively.

5.5.1.7 From the *Wiring* folder, select a *Splitter* and insert it onto the *canvas*.

5.5.1.8 As you did in lab (4), use the *splitter* to collect the three inputs A, B, and C in a bus. Ensure that inputs A, B, and C are connected to bits 2, 1, and 0 of the bus, respectively. Connect this bus to the input of the decoder.



Selection: Splitter

Facing	West
Fan Out	3
Bit Width In	3
Appearance	Left-handed
Bit 0	0 (Top)
Bit 1	1
Bit 2	2 (Bottom)

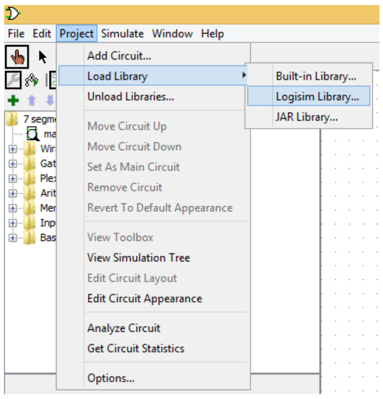
5.5.1.9 The decoder has 8 outputs. According to the circuit you draw in point 5.5.1.2, implement the Boolean function using proper decoder outputs. Use the *poke tool* to check the circuit for proper operation. Have your TA to check your work.

5.5.2 Seven-Segment Decoder and Seven-Segment Display

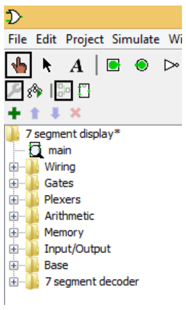
5.5.2.1 Start a new Logisim project. Save the project with the name "7 segment display"

5.5.2.2 The 7-segment decoder is not a part of the built in Logisim modules. Therefore, we have to add it to Logisim first. The 7 segment decoder is already implemented for you. You could find it in the desktop under the name "7 segment decoder"

5.5.2.3 From the *Project menu*, select *Load library/Logisim Library*



5.5.2.4 Browse to the file "7 segment decoder" on the desktop, select it, and click *Open*. You will see a new folder added to your *explorer pane* named "7 segment decoder"

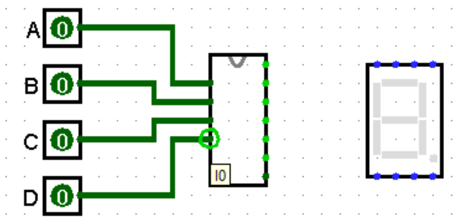


5.5.2.5 From the "7 segment decoder" folder, select main and insert it into the *canvas*

5.5.2.6 From the <u>*Input/Output*</u> folder, select "7-segment display" and insert it into the *canvas*.

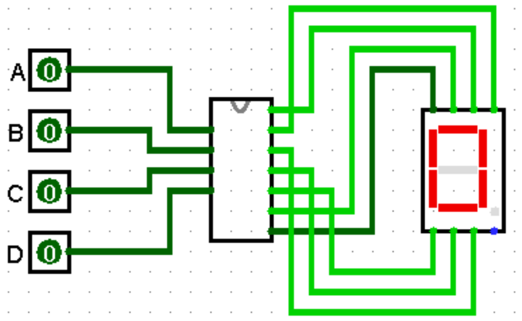
5.5.2.7 Insert 4 input pins to the *canvas*, to the left of the decoder, and name them A, B, C, and D, respectively.

5.5.2.8 Connect A, B, C, and D to I3, I2, I1, and I0 of the decoder, respectively. To know the name of any decoder input, stop the mouse over that input. A hint box will appear with the name of that input. For example, the following figure shows the hint of decoder input I0.

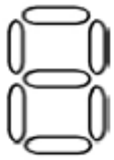
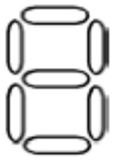
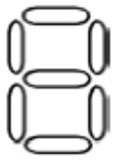
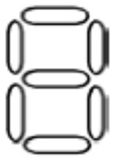
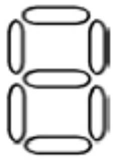
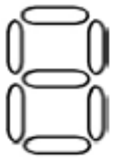


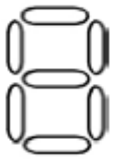
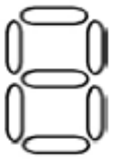
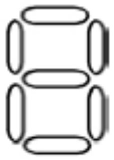
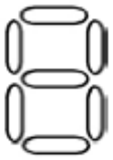
5.5.2.9 The 7 segment display has 8 inputs corresponding to the 7 segments, a to g, and the dot in the bottom right. If you start from the top right and go counter-clockwise, the order of these inputs is: b, a, f, g, e, d, c, and the dot. You could verify that yourself by connecting every input to logic 1 and see which segment will be ON.

5.5.2.10 Connect every output of the decoder to its corresponding input of the display. Leave the input of the dot unconnected.



5.5.2.11 Use the *poke tool* to verify the operation of the circuit. Enter different BCD numbers and monitor the corresponding decimal digits on the 7 segment display. Fill in the following table (i.e., color the segments that are ON)

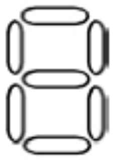
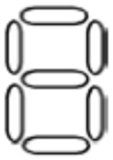
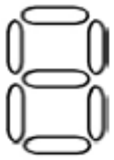
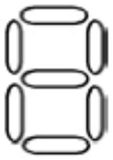
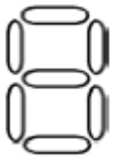
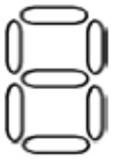
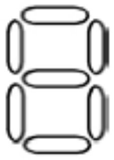
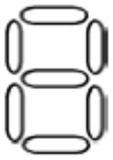
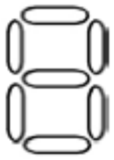
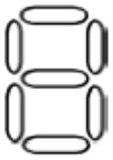
A	B	C	D		A	B	C	D	
0	0	0	0		0	1	0	1	
0	0	0	1		0	1	1	0	
0	0	1	0		0	1	1	1	

0	0	1	1		1	0	0	0	
0	1	0	0		1	0	0	1	

5.6 Hardware Implementation

5.6.1 Construct the circuit that shows BCD numbers on a 7-segment display, as shown in subsection 5.2.3.

5.6.2 Verify the operation of the circuit. Enter different BCD numbers and monitor the corresponding decimal digits on the 7-segment display. Fill in the following table (i.e., color the segments that are ON). Have your TA to check your work.

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

5.7 Questions

5.7.1 Implement the following Boolean function using 3x8 decoder. Draw the circuit diagram and simulate your design using Logisim to ensure its proper operation. Snapshots from Logisim should be included in your report.

F(A, B, C) = A' B' C' + A B' C + A B'

Week (12)

Experiment 6: Combinational Logic Modules – Comparators and Multiplexers

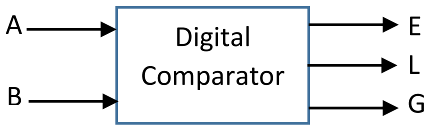
6.1 Objectives:

- To understand the function, design, and operation of the digital comparator.
- To learn practically how to compare two binary numbers using the 7485 4-bit magnitude comparator chip.
- To understand the function and operation of multiplexers
- To learn how to use multiplexers to build any Boolean expression

6.2 Background Information:

6.2.1 Single-bit Comparator

Digital Comparator "also called Magnitude Comparator" is a combinational circuit that compares two inputs binary numbers (A and B) and generates outputs to indicate whether the two inputs are equal or which input is greater than the other. Accordingly, the circuit has three outputs to indicate whether A=B, A>B, or A<B. At any given input quantities, only one output should be equal to logic '1'. The following figure shows a block diagram of the magnitude comparator.



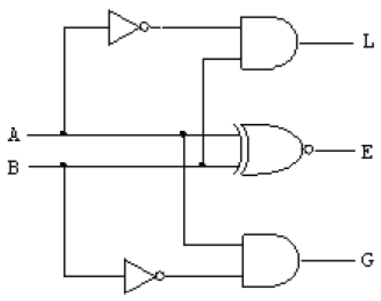
The truth table for the single bit comparator is as shown below:

A	B	E (A=B)	L (A<B)	G (A>B)
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

From the truth table, we observe that:

$E = A \odot B$
$L = A'.B$
$G = A.B'$

We can construct the logic circuit of the single-bit magnitude comparator as shown below:



6.2.2 n-bit Comparators

In order to understand how to compare numbers of more than one bit, consider two 4-bit unsigned numbers A and B.

$A = A_3\,A_2\,A_1\,A_0$ $B = B_3\,B_2\,B_1\,B_0$

The two numbers are equal if all pairs of corresponding bits are equal. i.e., $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$. To check for this equality, we XNOR each pair of corresponding bits, as we did in a single-bit comparator.

$$E_3 = A_3 \odot B_3$$
$$E_2 = A_2 \odot B_2$$
$$E_1 = A_1 \odot B_1$$
$$E_0 = A_0 \odot B_0$$

So, for A = B, E_3, E_2, E_1 and E_0 altogether should be true. Thus,

$$E\,(A = B) = E_3\,E_2\,E_1\,E_0$$

To determine if A > B or A < B, we check the relative magnitudes of pairs of digits starting from the most significant position. If the pair of digits of the same weight is equal, we compare the next lower significant pair of digits, until a pair of unequal digits is found. If the corresponding digit of A is 1 and that of B is 0, we conclude that A > B; but If the corresponding digit of A is 0 and that of B is 1, we conclude that A < B. This comparison can be expressed as Boolean functions:

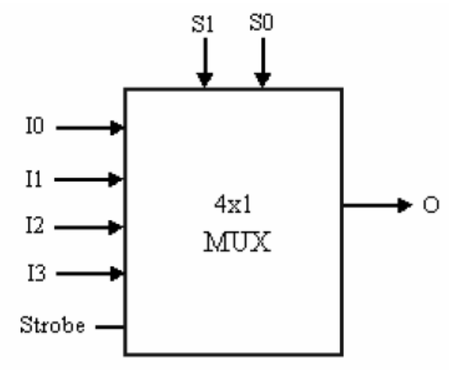
$$G\,(A > B) = A_3\,B_3' + E_3\,A_2\,B_2' + E_3\,E_2\,A_1\,B_1' + E_3\,E_2\,E_1\,A_0\,B_0'$$
$$L\,(A < B) = A_3'\,B_3 + E_3\,A_2'\,B_2 + E_3\,E_2\,A_1'\,B_1 + E_3\,E_2\,E_1\,A_0'\,B_0$$

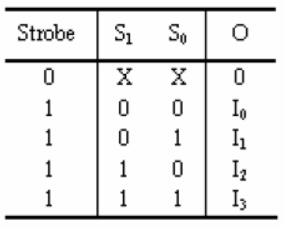
The logic circuit of four-bit magnitude comparator can be constructed using the various logic gates, based on to the Boolean expressions we have obtained. Furthermore, the comparison algorithm has a regular pattern, we can follow the same procedure to obtain a magnitude comparator circuit for binary numbers with more than four bits.

Commercially, the 7485 chip is a 4-bit magnitude comparator. This chip provides three decoded outputs that indicates which of the two 4-bit inputs is larger than the other or if both numbers are equal. It also includes three cascade inputs to allow two or more chips to compare larger numbers of eight or more bits.

6.2.3 Multiplexers

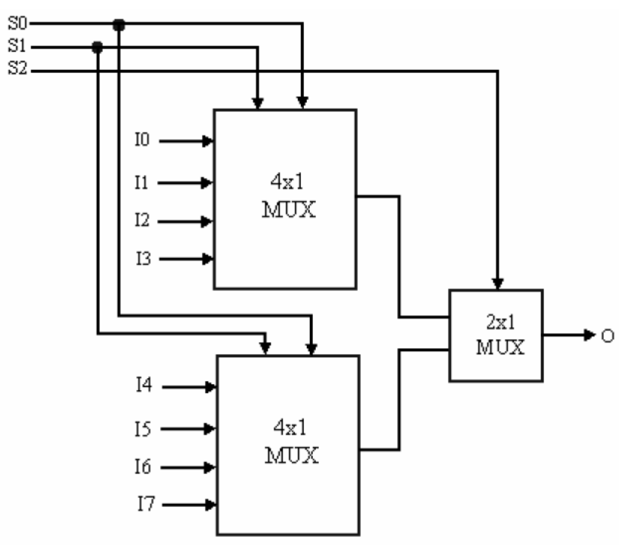
Multiplexers (MUX), sometimes called data selector, is a combinational logic circuit that selects one of 2^n inputs and routes it to the output. It has 2^n inputs, only one output, and n select lines (or simply called selectors) that identify which input will be provided to the output. A multiplexer might also has a strobe (sometimes called an enable) input that enables or disables the operation of the whole module. The following figure shows the block diagram and the truth table of a 4x1 MUX.





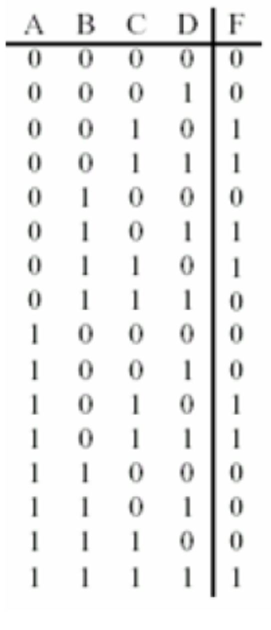
Strobe	S_1	S_0	O
0	X	X	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

Smaller multiplexers could be used to build larger ones. Select lines should be connected properly to ensure the correct routing of a certain input to the output. The following figure shows how 4x1 and 2x1 MUXs could be used to build an 8x1 MUX. Note that the strobe of the multiplexers are not drawn, which means that the MUXs are enabled forever.

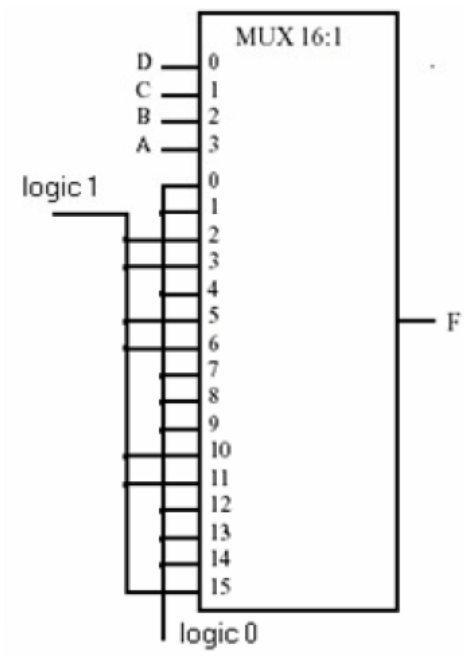


6.2.4 <u>Implementation of Boolean Functions using Multiplexers</u>

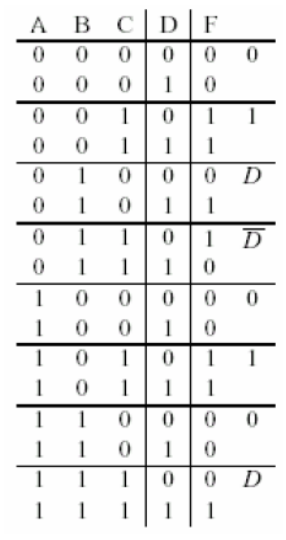
While MUXs are primarily thought of as data selectors, they can also be used to implement Boolean functions. Boolean functions of x variables could be implemented using a MUX with 2^x inputs. Consider a 4-variable Boolean function described by the truth table in the following figure. The function could be implemented by a 16x1 MUX. Variables A, B, C, and D should be connected to the select lines S_3, S_2, S_1, and S_0, respectively. Thereafter, logic 1 or logic 0 should be connected to the MUX inputs, according to the truth table. The following figure shows the implementation of the Boolean function using a 16x1 MUX.



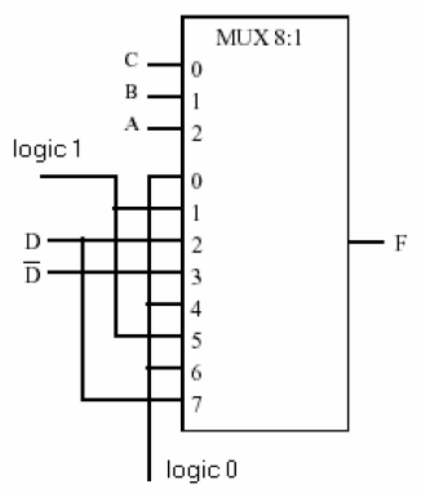
A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



Furthermore, we could implement the above Boolean function using an 8x1 MUX. Three of the variables should be connected to the select lines. According to the truth table, MUX inputs would be connected by logic 0, logic 1, the fourth variable, or the complement of the fourth variable. Assume that we connect A, B, and C to the select lines. Then, the truth table should be partitioned into sections that have the same value of A, B, and C, as shown in the following figure. The value of the Boolean function (F) for that section indicates whether its corresponding MUX input should be connected to logic 0 "F is always 0", logic 1 "F is always 1", D "F is similar to D", or D' " F is the complement of D". The following figure shows the implementation of the Boolean function using 8x1 MUX.



A	B	C	D	F	
0	0	0	0	0	0
0	0	0	1	0	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	0	*D*
0	1	0	1	1	
0	1	1	0	1	$\overline{D}$
0	1	1	1	0	
1	0	0	0	0	0
1	0	0	1	0	
1	0	1	0	1	1
1	0	1	1	1	
1	1	0	0	0	0
1	1	0	1	0	
1	1	1	0	0	*D*
1	1	1	1	1	



6.3 Equipment Required:

6.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

6.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- 1x 7404 Hex INVERTERS
- 1x 7408 QUAD 2-INPUT AND
- 1x 74266 QUAD 2-INPUT EXCLUSIVE-NOR GATES
- 1x 7485 4-BIT MAGNITUDE COMPARATORS
- 1x 74153 DUAL 4-To-1 LINE DATA SELECTORS/MULTIPLEXERS
- 1x 74151 8-To-1 LINE DATA SELECTOR/MULTIPLEXER
- 8x Toggle switches
- 3x Carbon-film Resistors (470Ω)
- 3x LEDs

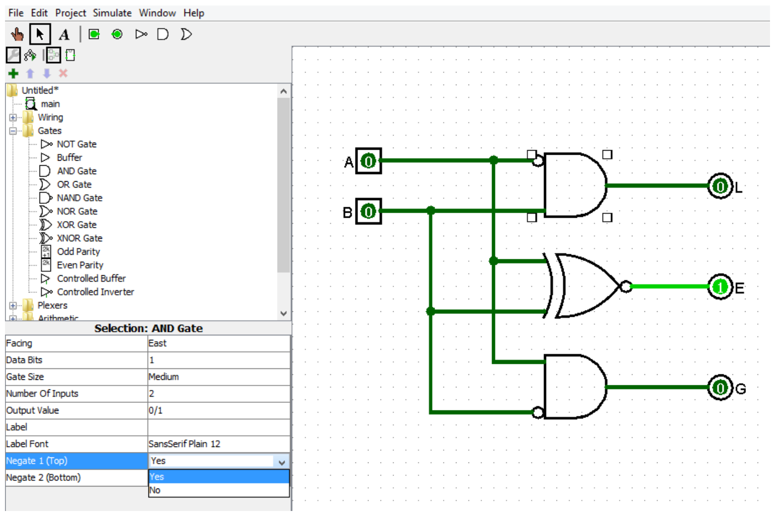
6.4 Prelab:

It is recommended that you read Section 6.2 of this lab in order to review digital comparators and multiplexers.

6.5 Software Simulation

6.5.1 <u>Single-bit Comparator</u>

6.5.1.1 Launch Logisim and enter the single-bit comparator circuit, as shown in subsection 6.2.1. Instead of using inverters, you could simply invert inputs of AND gates using the *Negate* attribute from the <u>*attribute table*</u>, as shown below for the upper AND gate.



Selection: AND Gate

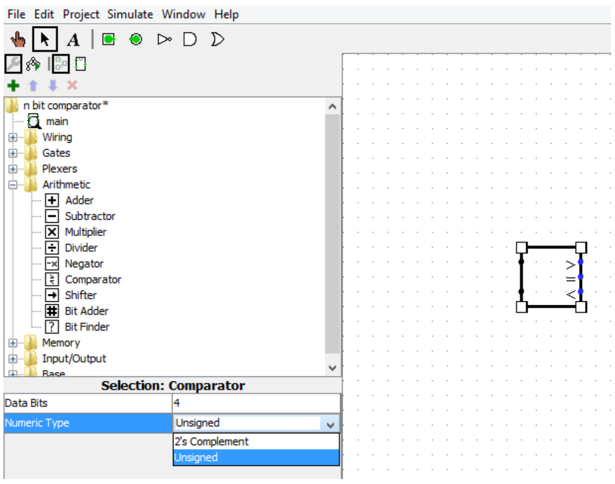
Facing	East
Data Bits	1
Gate Size	Medium
Number Of Inputs	2
Output Value	0/1
Label	
Label Font	SansSerif Plain 12
Negate 1 (Top)	Yes
Negate 2 (Bottom)	Yes / No

6.5.1.2 Check the operation of the comparator either by using the *poke tool* or the *combinational analysis* window. The truth table should agree with the one in subsection 6.2.1.

6.5.2 n-bit Comparator

6.5.2.1 Start a new Logisim project by clicking Ctrl+N

6.5.2.2 From the *arithmetic* folder in the *explorer pane*, select Comparator and insert it onto the *canvas*. Change its attributes in the *attribute table* such that *Data Bits* = 4 and *Numeric Type* = Unsigned. This makes it to be a 4-bit unsigned magnitude comparator.

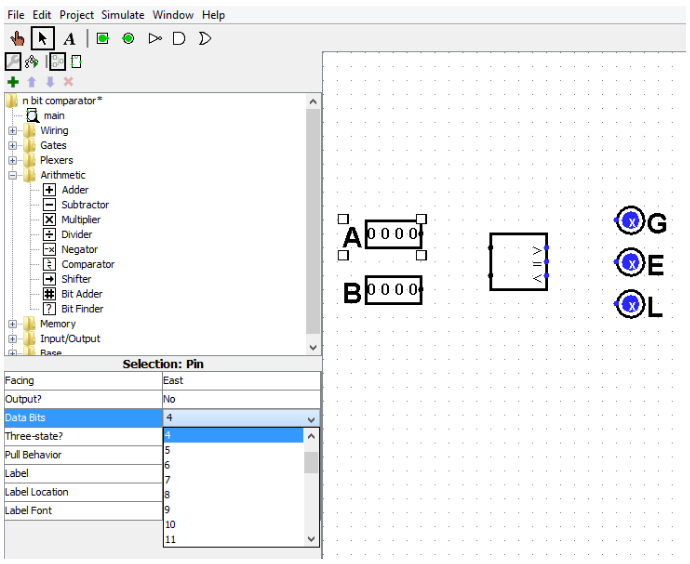


Selection: Comparator

Data Bits	4
Numeric Type	Unsigned

2's Complement
Unsigned

6.5.2.3 Connect two *input ports* and label them A and B. In the *attribute table*, change their *Data Bits* to 4. Furthermore, insert three 1-bit *output ports* and label them G, E, and L.



Selection: Pin

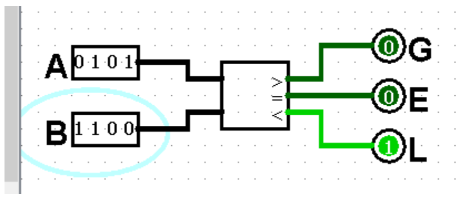
Facing	East
Output?	No
Data Bits	4
Three-state?	
Pull Behavior	
Label	
Label Location	
Label Font	

4
5
6
7
8
9
10
11

6.5.2.4 Connect input port A, to the upper input of the comparator and input B to the lower one.

6.5.2.5 The Comparator module has three outputs labelled >, =, and <. Connect these comparator outputs to ports G, E, and L, respectively.

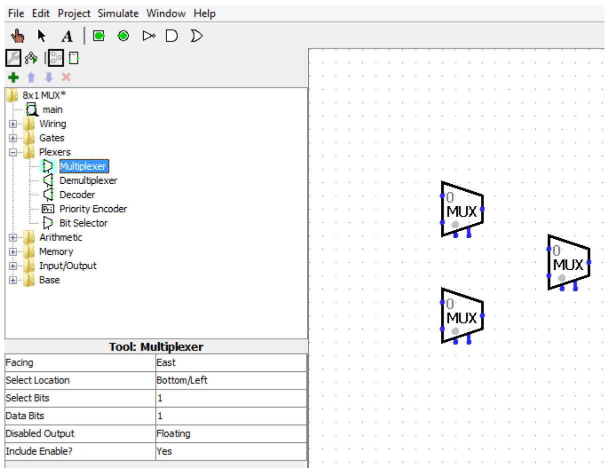
6.5.2.6 Use the *poke tool* to enter different numbers to A and B in order to check the operation of the comparator. Have your TA to check your work.



6.5.3 Multiplexers

6.5.3.1 Start a new Logisim project by clicking Ctrl+N

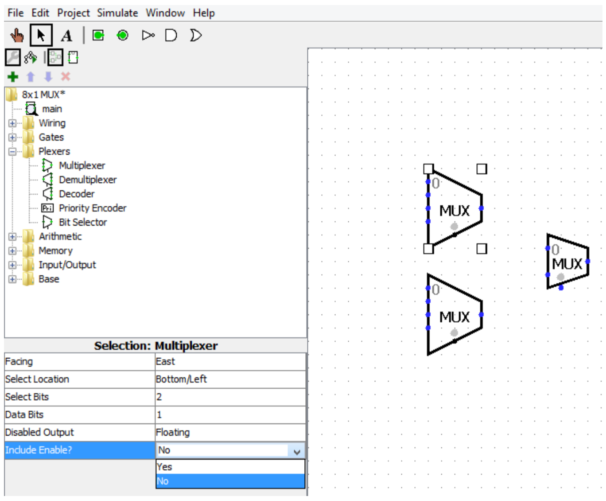
6.5.3.2 From the *Plexers* folder in the *explorer pane*, select multiplexer and insert it three times onto the *canvas*.



Tool: Multiplexer

Facing	East
Select Location	Bottom/Left
Select Bits	1
Data Bits	1
Disabled Output	Floating
Include Enable?	Yes

6.5.3.3 In the *attribute tables* of the two multiplexers on the left, change the *Select Bits* to 2 (i.e., 4x1 MUX) and the *Include Enable* to "No" (i.e., no strobe and the MUX is always working). For the MUX into the left, only change its *Include Enable* attribute to "No".

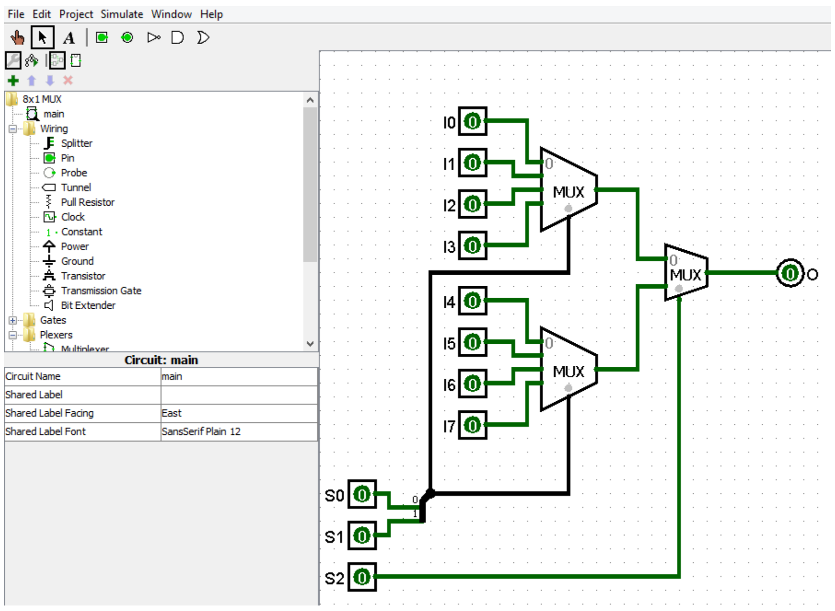


Selection: Multiplexer

Facing	East
Select Location	Bottom/Left
Select Bits	2
Data Bits	1
Disabled Output	Floating
Include Enable?	No

Yes [✓ No]

6.5.3.4 Now, construct an 8x1 MUX using these three MUXs, according to the figure in subsection 6.2.3. Use a *splitter* from the *Wiring* folder, as you did in lab (4), to combine the two select lines S_0 and S_1 into a 2-bit bus that would be connected to the selectors of the two left MUXs.



Circuit: main

Circuit Name	main
Shared Label	
Shared Label Facing	East
Shared Label Font	SansSerif Plain 12

6.5.3.5 Use the *poke tool* to enter different inputs through I_0 to I_7 and S_0 to S_2 in order to check the operation of the circuit. For example, clear all the select lines to 0, change I_0 between 0 and 1 and monitor the output. The output should follow the changes in the input. Repeat for other inputs and have your TA to check your work.

6.5.4 <u>Implementation of Boolean Functions using Multiplexers</u>

6.5.4.1 Consider the following Boolean function. It is required to implement it using 8x1 MUX and 4x1 MUX

$$F(x, y, z) = x' y' z' + x' y z' + x y' z + x y z$$

6.5.4.2 For the 8x1 MUX implementation, construct the truth table and draw the implemented circuit in the area below. If necessary, have your TA to check your implementation.

6.5.4.3 Enter the circuit you designed into Logisim and ensure its correct operation. Use *Power* and *Ground* from the *<u>Wiring</u>* folder to enter logic 1 and logic 0, respectively.

6.5.4.4 For the 4x1 MUX implementation, construct the truth table and draw the implemented circuit in the area below. Partition the truth table into sections based on identical values of x and y. If necessary have your TA to check your work.

6.5.4.5 Enter the circuit you designed into Logisim and ensure its correct operation.

6.6 Hardware Implementation

6.6.1 <u>Single-bit Comparator</u>

6.6.1.1 Use basic logic gates to build the single-bit comparator, according to the circuit diagram in subsection 6.2.1

6.6.1.2 Once all connections have been done, turn on the power switch of the training kit. Use different sets of inputs for A and B to check each of the outputs L, E, and G. Have your TA to check your work.

6.6.2 <u>n-bit Comparator</u>

6.6.2.1 Construct a 4-bit magnitude comparator circuit using switches, LEDs , and IC chip 7485. Verify the function of the circuit by applying different sets of inputs and monitoring the outputs. Have your TA to check your work.

6.6.3 <u>Implementation of Boolean Function using Multiplexers</u>

6.6.3.1 Construct the circuits you designed in points 6.5.4.2 and 6.5.4.4 and ensure that they correctly implement the Boolean function. Have your TA to check your work.

6.7 Questions

6.7.1 Assume that you already have a 4-bit subtractor. Use it to design a 4-bit magnitude comparator. Draw the block diagram of your circuit and explain how it works.

6.7.2 Construct a 16x1 MUX using 8x1 and 4x1 MUXs. Draw the block diagram of the circuit and clearly highlight how the select lines are connected.

6.7.3 Implement the following Boolean function using 8x1 and 4x1 MUXs. For each implementation, construct the truth table and draw the circuit diagram

$$F(A, B, C) = A'\,B\,C' + A'\,B\,C + A\,B'\,C + A\,B\,C'$$

Week (13)

Experiment 7: Exam on Combinational Logic Modules – Design of an Arithmetic Circuit

7.1 Objectives:

- To apply the combinational logic design techniques, covered throughout the course and the labs, in building a simple combinational module.
- To design, simulate, construct, and test an arithmetic circuit that has four operations: Add, Subtract. Decrement, and Increment

7.2 Introduction:

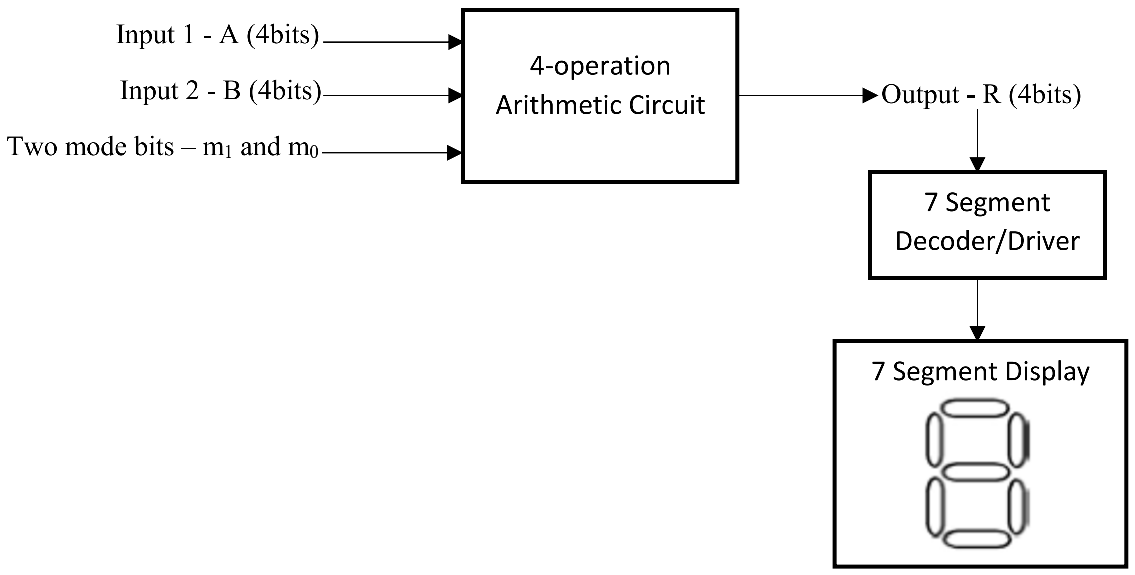
In this lab, you are required to use the combinational logic knowledge that you acquire in previous labs to build a simple new combinational module. The module that you are to build is a 4-operation arithmetic circuit. The arithmetic circuit has two 4-bit inputs (A and B) and one 4-bit output (R). The output (R) should be further displayed on a 7-segment LED display. Accordingly, it passes through a common-anode BCD to 7-segment decoder/driver to a common-anode 7-segment display. For the sake of simplicity, assume that the output of the arithmetic circuit will never exceed 9; and hence, there is no need to convert the binary output of the circuit to BCD. Moreover, ignore the generation of a carry or a borrow. According to input mode bits: m_0 and m_1, the arithmetic circuit is to do one of four arithmetic operations: Add, Subtract, Decrement, and Increment. The following table describe the operation of the circuit.

Arithmetic Circuit Operations

(A=A_3 A_2 A_1 A_0 and B=B_3 B_2 B_1 B_0 are 4-bit inputs; whereas, R=R_3 R_2 R_1 R_0 is the output)

Mode Bits		**Operation**	**Output**
m_1	**m_0**		
0	0	Add A and B	R = A + B
0	1	Subtract B from A	R = A – B = A + B' + 1
1	0	Decrement A	R = A – 1 (Hint, Add the 2's complement of 1 to A)
1	1	Increment A	R = A + 1

Throughout the lab, you are required to design the arithmetic circuit using the IC chips, listed in Section 7.3. Thereafter, the circuit you designed should be simulated using Logisim, constructed using IC chips, and tested for proper operation. The following block diagram shows the interface of the circuit.



7.3 Equipment Required:

7.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

7.3.2 Hardware implementation: The following modules are only allowed to design and implement your circuit:

- IC Training Kit
- Jumper wire kit
- 1x 7404 HEX INVERTERs
- 1x 74153 DUAL 4-To-1 LINE DATA SELECTORS/MULTIPLEXERS
- 1x 74283 4-BIT FULL ADDER
- 1x 7447 BCD-TO-SEVEN SEGMENT DECODER/DRIVER
- 1x Common-Anode Seven-Segment LED Display
- 2x 4-bit DIP switches (For A and B)
- 2x Toggle Switches (For m_1 and m_0)

7.4 Procedure

7.4.1 Design the arithmetic circuit using the modules listed in Section 7.3. Draw a block diagram for the circuit in the following area. Have your TA to check you design.

7.4.2 Enter the designed circuit onto Logisim and simulate it to ensure that it works as prescribed. Have your TA to check your work.

7.4.3 Construct the designed circuit using IC chips and test it to ensure that it works as prescribed. Have your TA to check your work.

7.5 Questions

7.5.1 Assume that the output of the arithmetic circuit, you designed in this lab, could exceed the value of 9. Design a module that converts the binary output (R) to BCD. Insert this module before the BCD decoder/driver. Carry out any other required modifications and draw the new block diagram.

7.5.2 What modifications should be done to the arithmetic circuit you designed in this lab to be able to handle 8-bit inputs (both A and B)

Week (14)

Tutorial (5): Sequential logic, flip flops, FSM analysis and design

Reminder: Please, remember to print and bring the tutorial with you

Week (15)
Experiment 8: Latches, Flip Flops and Counters

8.1 Objectives

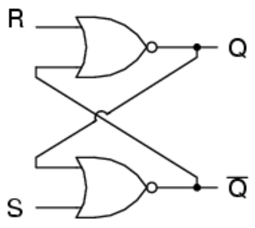
- To learn the design and operation of different types of latches and flip flops
- To convert one type of latches/flip flops to another
- To understand the operation of asynchronous ripple binary counters
- To learn how to design an n-bit counter using J-K flips flops

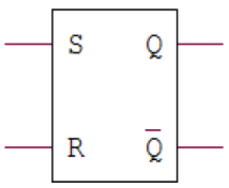
8.2 Background Information

Sequential circuits are logic circuits with feedback. Latches and Flip-flops are the most elementary sequential circuits. Latches and flip flops always have two outputs: Q and its complementary value Q'. The output of sequential circuits depends not only on the present value of the inputs but also on the state of the circuit. The state of the circuit is indicated by the content of its memory elements (i.e., its flip flops). Sequential logic circuits often require a timing generator (a clock) for their operation. The clock synchronizes the operation of the circuit and indicates when the output of the sequential circuit should be changed. In the literature, there are four types of latches as well as flip flops. These are the S-R, D, J-K, and T latches/flip flops. In this lab, you will experience some of these lathes/flip flops. Furthermore, you will use the J-K flip flop to build a larger sequential circuit: a 4-bit ripple counter.

8.2.1 <u>S-R latch</u>

An S-R latch consists of two cross-coupled NOR gates, as shown below. The truth table and the block diagram of the latch are also shown. An S-R latch can also be design using cross-coupled NAND gates. The S-R latch could be envisioned as a basic 1-bit memory cell. The user of an S-R latch could store binary 1 (which is known as SET), store binary 0 (which is known as RESET), or keep the stored value as it is (which is known as HOLD). The following truth table shows the values of S and R to hold, set, or reset the latch. The outputs of the latch are the stored bit (Q) and its complement (Q'). Accordingly, having S and R high simultaneously is not allowed. In the truth table, Q means the value (also known as the state) of the latch before changing S and R; whereas, Q+ means he value (or the state) of the latch after changing S and R.

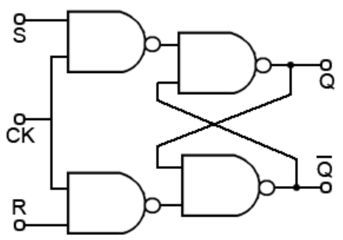




S	R	Q+	Q'+	Action
0	0	Q	Q'	Hold the stored value
0	1	0	1	Reset "store binary 0"
1	0	1	0	Set "store binary 1"
1	1	0	0	Not allowed

8.2.2 <u>Gated S-R latch</u>

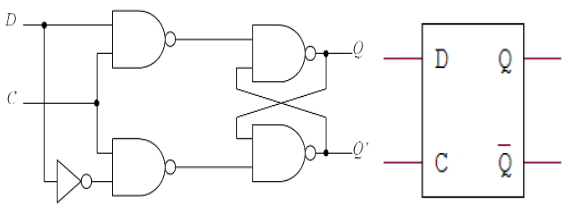
A gated S-R latch (also called a latch with enable or control) has an additional gate input. The gate (or the enable) input works as an enable for the operation of the latch. It allows the designer to change the state of the latch at specific time. This, in turn, allows for the synchronization between modules in large digital systems. The following figure shows a NAND-based gated S-R latch with its truth table. In this gated latch, S and R inputs are effective only when the enable (CK or EN) is high. When the enable goes low, irrespective of S and R, the state of the latch is hold and cannot change until the enable goes high again.



CK (or EN)	S	R	Q+	Q'+	Action
0	X	X	Q	Q'	Hold the stored value
1	0	0	Q	Q'	Hold the stored value
1	0	1	0	1	Reset "store binary 0"
1	1	0	1	0	Set "store binary 1"
1	1	1	1	1	Not allowed

8.2.3 Conversion of an S-R latch to D latch

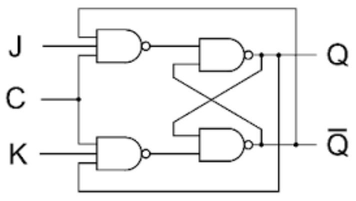
A D latch combines the S and R inputs of an S-R latch into one input by adding an inverter. When the enable is high, the output (Q) follows the D input, and when the enable goes low, the state is latched. The following figure shows the schematic of the D latch with its block diagram and truth table.



C (or EN)	D	Q+	Q'+	Action
0	X	Q	Q'	Hold the stored value
1	0	0	1	Reset "store binary 0"
1	1	1	0	Set "store binary 1"

8.2.4 Conversion of an S-R latch to J-K latch

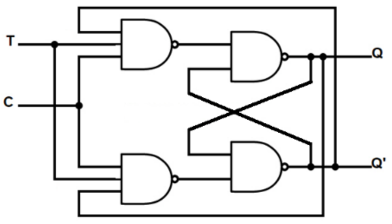
The J-K latch overcome the problem of the "not-allowed" state in the S-R latch. Beside the enable, it has two inputs J and K, which are corresponding to S and R, respectively. When both J and K are active simultaneously, the J-K latch will toggle (i.e., complement) its previous state of Q. The following figure shows the schematic of the J-K latch with its truth table.



C (or EN)	J	K	Q+	Q'+	Action
0	X	X	Q	Q'	Hold the stored value
1	0	0	Q	Q'	Hold the stored value
1	0	1	0	1	Reset "store binary 0"
1	1	0	1	0	Set "store binary 1"
1	1	1	Q'	Q	Toggle the stored value

8.2.5 Conversion of a J-K latch to T latch

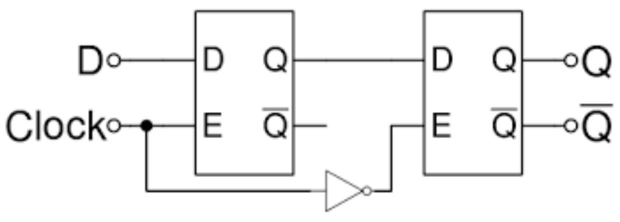
The J-K latch could be converted into a T latch by connecting J and K inputs together. The T latch either holds its current value or toggles it. It is very useful in building different types of counters. The following figure shows the schematic of the T latch with its truth table.

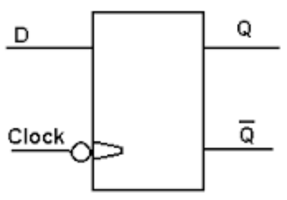


C (or EN)	T	Q+	Q'+	Action
0	X	Q	Q'	Hold the stored value
1	0	Q	Q'	Hold the stored value
1	1	Q'	Q	Toggle the stored value

8.2.6 Master Slave Flip Flops

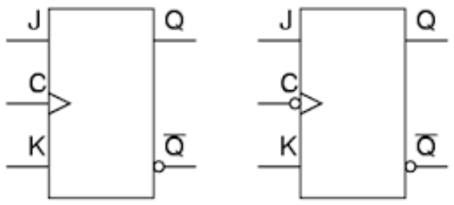
Latches are said to be level-sensitive. As long as the enable input is active, the output could continuously change. Therefore, if a designer wants the output to change at a certain moment, the enable should be activated for a very short period of time, corresponding to this moment. The generation of this narrow pulse is not easy. On the other hand, flip flops are edge-sensitive. Their output changes only at the edge of the clock. Otherwise, the flip flop holds its current value till an edge transition occurs at the clock again. The flip flop could be designed to be rising edge sensitive or falling edge sensitive. Using flip flops, the designer could decide the moments at which the states would change by only adjusting the clock frequency (i.e., when the edges occur). There is no need to have a sophisticated pulse generator, as in the case of latches. The following figure shows an example circuit of the D flip flop with its block diagram and truth table. In the block diagram, notice the small triangle of the clock input. This indicates that the module is an edge-sensitive flip flop rather than a level-sensitive latch. The small circle beside the triangle indicates that the flip flop is negative-edge sensitive. i.e., the output would change only on the falling edge of the clock, as shown in the truth table.





Clock	D	Q+	Q'+	Action
No edge	X	Q	Q'	Hold the stored value
↓	0	0	1	Reset "store binary 0"
↓	1	0	1	Set "store binary 1"

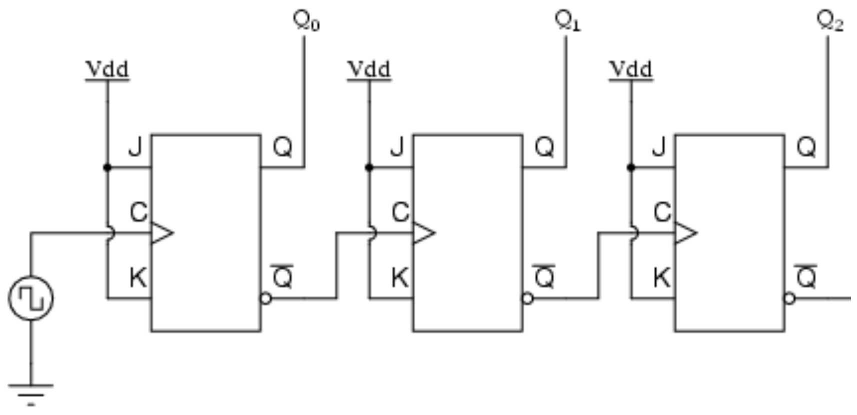
Similar to the D flip flop, there are S-R, J-K, and T flip flops. The difference between these flip flops and their corresponding latches is that they are edge-sensitive rather than level-sensitive. The following figure shows an example block diagram of the J-K flip flop. The small triangles indicate that these are flip flops rather than latches. Moreover, the small bubble beside the triangle indicates that this flip flop is falling (negative) edge sensitive, rather than a rising (positive) edge sensitive.



Clock (C)	J	K	Q+	Q'+	Action
No edge	X	X	Q	Q'	Hold the stored value
↓ or ↑	0	0	Q	Q'	Hold the stored value
↓ or ↑	0	1	0	1	Reset "store binary 0"
↓ or ↑	1	0	1	0	Set "store binary 1"
↓ or ↑	1	1	Q'	Q	Toggle the stored value

8.2.7 <u>Asynchronous Ripple Counters</u>

Counters are sequential circuits that increment "or decrement" their outputs every clock pulse. Flip flops are the building block of counters. In an asynchronous ripple counter, T or J-K flip flops are usually used. The inputs of flip flops (J, K, or T) are connected to logic 1 in order to toggle with every clock pulse. The first (i.e., the lowest significant) flip flop is triggered from the external clock. Thereafter, each flip flop is triggered by the Q or the Q' outputs of its previous flip flop. Based on this connection, as well as whether the flip flop is rising-edge or falling-edge sensitive, the counter could count up or down. The following figure shows an example of 3-bit up counter using J-K flip flops. For the counter to count down, either the connection is done from Q instead of Q', or falling-edge sensitive flip flops could be used.



8.3 Equipment Required

8.3.1 Software simulation: The Logisim software package is installed on every PC in the lab.

8.3.2 Hardware implementation: The following equipment are needed to perform all the procedures:

- IC Training Kit
- Jumper wire kit
- 2x 7473 DUAL J-K FLIP FLOPS WITH CLEAR

- 1x 7400 QUAD NAND GATES
- 3x Toggle Switches
- 3x Carbon-film Resistors (470Ω)
- 3x LEDs

8.4 Prelab

It is recommended that you read Section 8.2 of this lab in order to review latches, flip flops and counters.

8.5 Software Simulation

8.5.1 S-R Latch

8.5.1.1 Start a new Logisim project

8.5.1.2 Construct the S-R latch according to the circuit in subsection 8.2.1.

8.5.1.3 Use the *poke tool* to simulate the latch. Change the values of S and R according to the following table and fill in the values of Q and Q'. In the action column, write whether the latch is set, reset, hold, or not allowed. Ensure that the operation is as prescribed in subsection 8.2.1.

S	R	Q+	Q'+	Action
1	0			
0	0			
0	1			
0	0			
1	0			
1	1			

8.5.2 Gated S-R Latch

8.5.2.1 Start a new Logisim project

8.5.2.2 Construct the gated S-R latch according to the circuit in subsection 8.2.2.

8.5.2.3 Use the *poke tool* to simulate the latch. Change the values of CK, S, and R according to the following table and fill in the values of Q and Q'. In the action column, write whether the latch is set, reset, hold, or not allowed. Ensure that the operation is as prescribed in subsection 8.2.2.

8.5.2.4 Save your work as it will be used later in this lab

CK (or EN)	S	R	Q+	Q'+	Action
1	1	0			
1	0	0			
1	0	1			
1	0	0			
1	1	1			
1	1	0			
0	0	0			
0	0	1			
0	1	0			
0	1	1			

8.5.3 Conversion of an S-R latch to D Latch

8.5.3.1 Convert the S-R latch you build in subsection 8.5.2 to a D latch according to the circuit in subsection 8.2.3

8.5.3.2 Use the *poke tool* to simulate the D latch. Change the values of C and D according to the following table and fill in the values of Q and Q'. In the action column, write whether the latch is set, reset, or hold. Ensure that the operation is as prescribed in subsection 8.2.3.

C (or EN)	D	Q+	Q'+	Action
1	1			
1	0			
1	1			
0	1			
0	0			
0	1			

8.5.3.3 Save you D latch as you will use it in the following subsection to build the D flip flop.

8.5.4 Master Slave D Flip flop

8.5.4.1 Start a new logisim project.

8.5.4.2 From the *project menu*, select *Load Library* and click on Logisim Library. Browse to your D latch and load it into the project.

8.5.4.3 Construct the master slave D flip flop according to the circuit in subsection 8.2.6.

8.5.4.4 Use the *poke tool* to simulate the flip flop. Change the values of the clock and D, according to the following table and fill in the values of Q and Q'. In the action column, write whether the flip flop is set, reset or hold. Ensure that the operation is as prescribed in subsection 8.2.6.

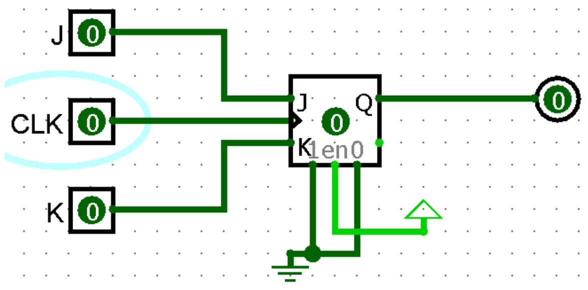
Clock	D	Q+	Q'+	Action
1	1			
1	0			
1	1			
0 "-ve edge"	1			
0	0			
0	1			
0	0			
1	0			
0 "-ve edge"	0			
0	1			

8.5.5 Master Slave J-K Flip flop

8.5.5.1 Start a new Logisim project

8.5.5.2 From the *Memory* folder in the *explorer pane*, insert a J-K Flip-Flop. Insert one *output pin* for Q and three *input pins* for J, K, and clock

8.5.5.3 From the *Wiring* folder in the *explorer pane*, insert *Power* and *Ground*. Connect the enable input of the flip flop (named en) to the *Power*. Connect the *Preset* (named 1) and the *Clear* (named 0) inputs of the flip flop to the ground.

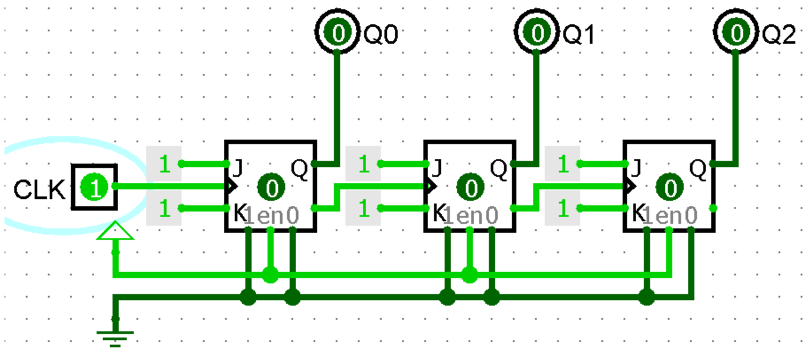


8.5.5.4 Use the *poke tool* to simulate the J-K flip flop. Ensure that the operation is as prescribed in subsection 8.2.6.

8.5.6 Asynchronous ripple counter

8.5.6.1 Start a new Logisim project

8.5.6.2 From the *Memory* folder in the *explorer pane*, insert 3 J-K Flip-Flops and use them to construct the 3-bit asynchronous counter circuit, as shown in subsection 8.2.7.



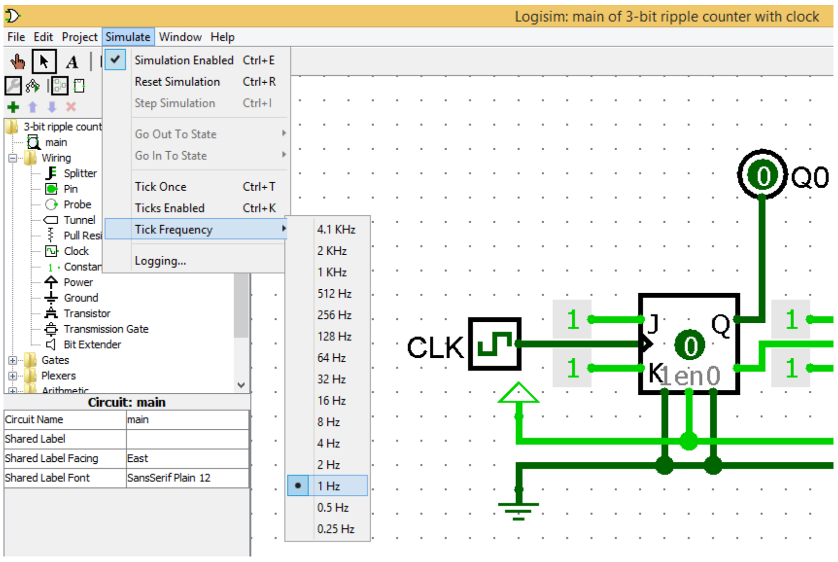
8.5.6.3 Use the *poke tool* to simulate the counter. Ensure that the count is incremented with every rising edge of the clock and fill in the following table. Notice that the order of the outputs in the above figure are the reverse to those in the table. For example, Q2 column is the output of the rightmost flip flop in the above figure.

Clock	Q2	Q1	Q0	Decimal value
	0	0	0	0
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				
↑ (0 → 1)				

8.5.6.4 Instead of using an input pin for the clock, let's investigate a simpler method.

8.5.6.5 Remove the *input pin* for the clock. Instead, insert *Clock* from the *Wiring* folder of the *explorer pane*.

8.5.6.6 From the *Simulate* menu, select *Tick Frequency* and click on 1Hz.



8.5.6.7 From the *Simulate* menu, click on *Ticks Enabled*. Notice that the counter is counting up every second.

8.6 Hardware Implementation

8.6.1 Gated S-R Latch

8.6.1.1 Construct the gated S-R latch, according to the circuit in subsection 8.2.2.

8.6.1.2 Change the values of CK, S, and R according to the following table and fill in the values of Q and Q'. In the action column, write whether the latch is set, reset, hold, or not allowed. Ensure that the operation is as prescribed in subsection 8.2.2.

CK	S	R	Q+	Q'+	Action
1	1	0			
1	0	0			
1	0	1			
1	0	0			
1	1	1			
1	1	0			
0	0	0			
0	0	1			
0	1	0			
0	1	1			

8.6.2 Asynchronous Ripple Counter

8.6.2.1 Construct the 3-bit asynchronous counter circuit, as shown in subsection 8.2.7. J, K, and CLR of all the flip flops should be connected to V_{cc}. Use a switch (or a push button) for the clock. Use LEDs for the three output Q_2 to Q_0.

8.6.2.2 Use the clock switch to trigger your counter. With every pulse, ensure that the counter output is incremented by 1. Have your TA to check your result.

8.7 Questions

8.7.1 Construct a J-K flip flop using a D flip flop. Draw your circuit and simulate it using logisim to ensure its proper operation. Snapshots from Logisim should be included in your report.

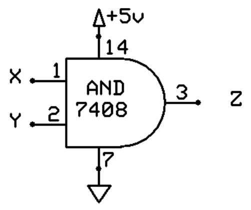
8.7.2 Design a 4-bit ripple down counter using T flip flops. Your counter should count down from 15 to 0 then underflow back to 15. Draw your circuit and simulate it using logisim to ensure its proper operation. Snapshots from Logisim should be included in your report.

8.7.3 Design a mod-5 up counter. The counter should count up from 0 to 4. Thereafter, the counter should go back to 0. Draw your circuit and simulate it using logisim to ensure its proper operation. Snapshots from Logisim should be included in your report.

Experiment 1 Worksheet: Introduction to Logic Gates

Student Name: ____ Student ID: ____________

Circuit



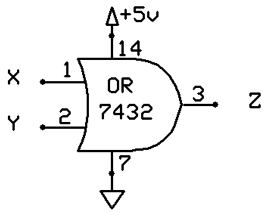
Truth Table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

Equation

Z = X.Y

Circuit



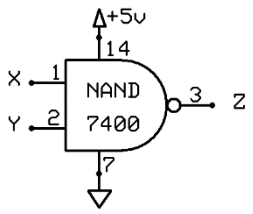
Truth Table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

Equation

Z = X+Y

Circuit



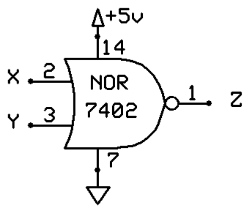
Truth Table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

Equation

$Z = \overline{X.Y}$

Circuit



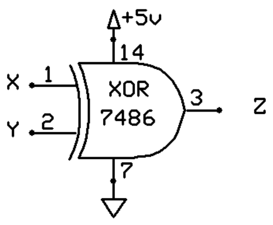
Truth Table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

Equation

$Z = \overline{X+Y}$

Circuit



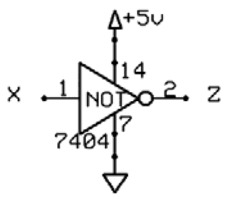
Truth Table

X	Y	Z
0	0	
0	1	
1	0	
1	1	

Equation

$Z = X \oplus Y$

Circuit



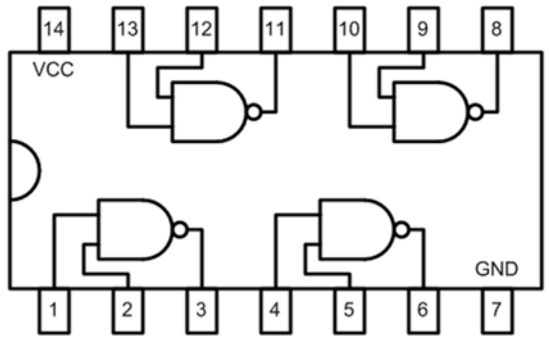
Truth Table

X	Z
0	
1	

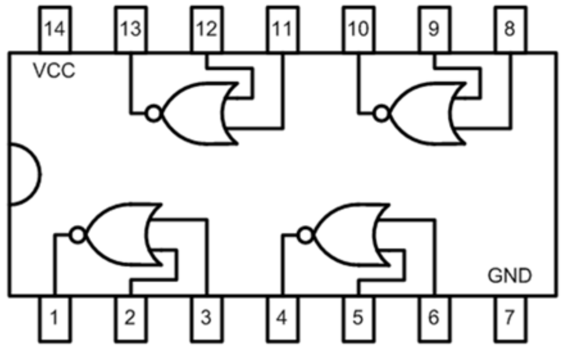
Equation

$Z = \overline{X}$

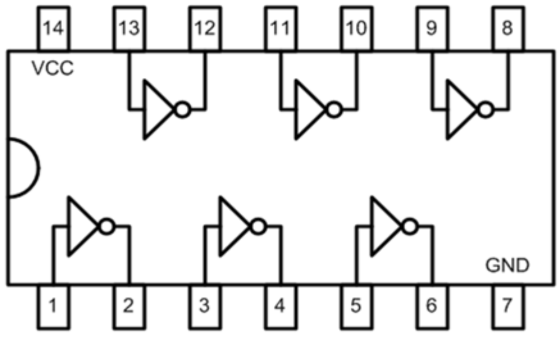
Data Sheets



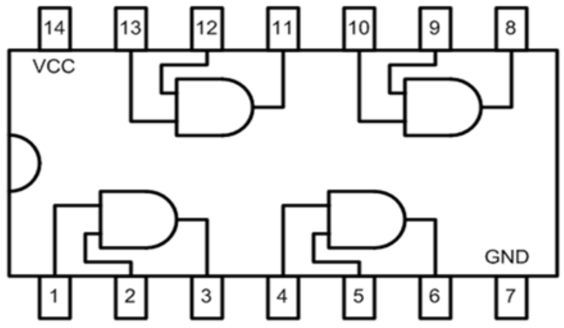
7400 Quad 2 Input NAND



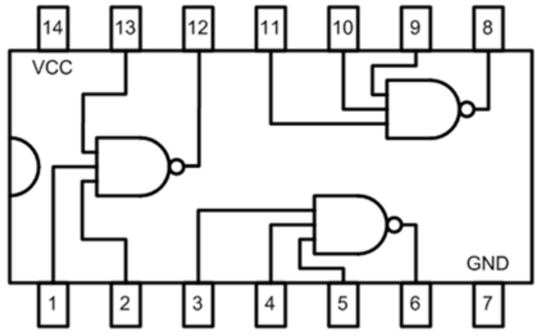
7402 Quad 2 Input NOR



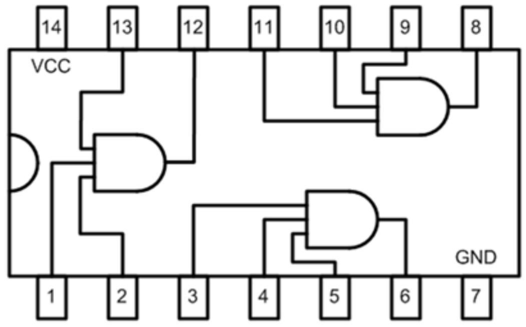
7404 Hex Inverter



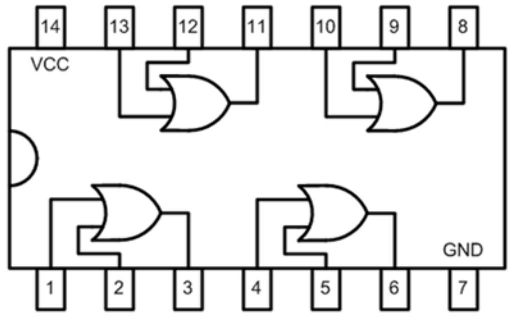
7408 Quad 2 Input AND



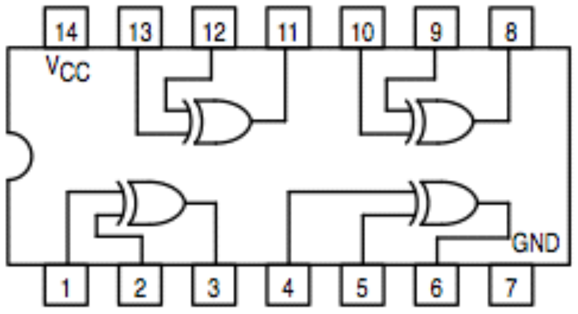
7410 Triple 3 Input NAND



7411 Triple 3 Input AND



7432 Quad 2 Input OR



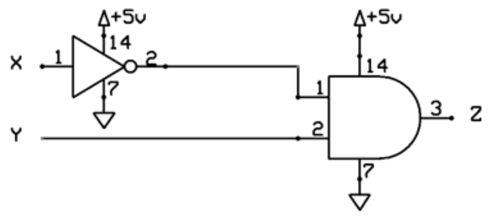
7486 Quad 2 Input XOR

Experiment 2 Worksheet: Implementation of Boolean Functions

Student Name: ____ Student ID: ____________

Part I:

(a) Circuit Diagram



(b) Truth Table

X	Y	X'	Z
0	0		
0	1		
1	0		
1	1		

(c) Equation

Z = X'.Y

Part II:

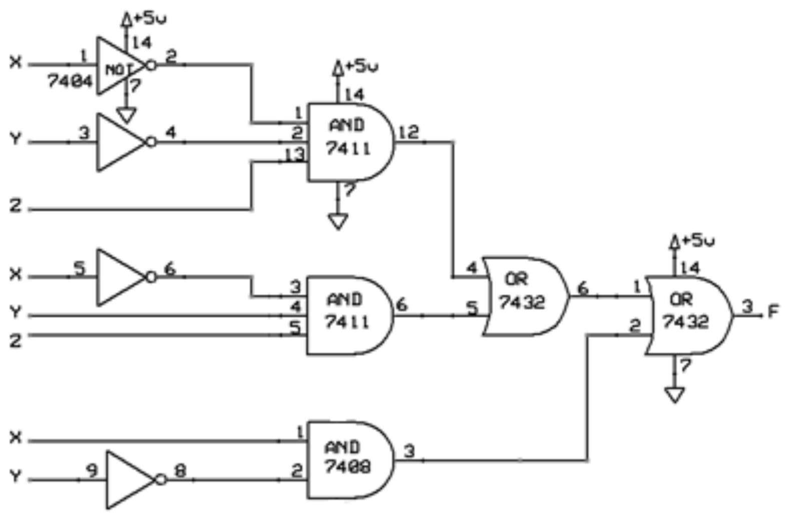
(a) Equation

F = X'Y'Z + X'YZ + XY'

(b) Truth Table

X	Y	Z	X'	Y'	Z'	X'Y'Z	X'YZ	XY'	F
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

(c) Circuit Diagram



Worksheet 2: Page **1** of **2**

Part III:

(a) Equation

$$F = A'B'C' + A'B'C + AB'C$$

(b) Truth Table

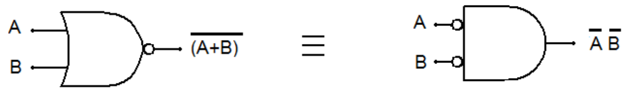
A	B	C							F
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

(c) Circuit Diagram

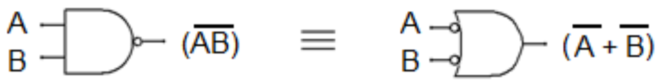
Experiment 3 Worksheet: NOR and NAND Implementation

Student Name: ____ Student ID: _____________

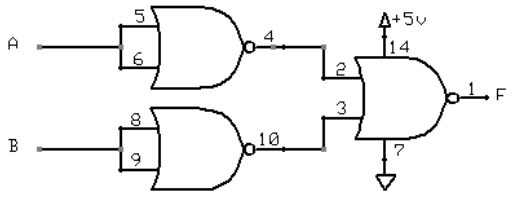
NOR Gate is Equivalent to Bubbled AND Gate



NAND Gate is Equivalent to Bubbled OR Gate

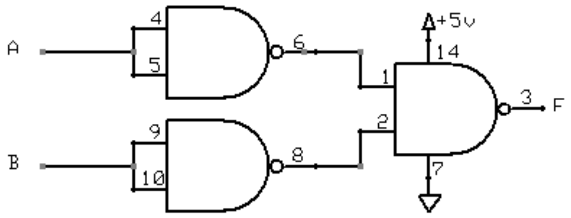


Part I: Implementation of AND gate using NOR gate.



A	B	F
0	0	
0	1	
1	0	
1	1	

Part II: Implementation of OR gate using NAND gate.



A	B	F
0	0	
0	1	
1	0	
1	1	

Part III: Implementation the following function using **NAND** gates.

$$F = X'Y'Z' + X'YZ + XY'Z + XYZ'$$

Draw circuit diagram and make truth table to verify the results using training board.

X	Y	Z	X'	Y'	Z'					
0	0	0								
0	0	1								
0	1	0								
0	1	1								
1	0	0								
1	0	1								
1	1	0								
1	1	1								

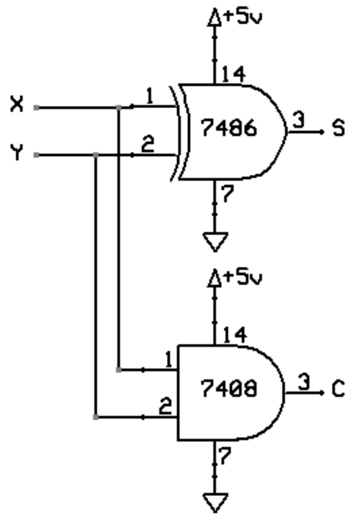
Experiment 4 Worksheet: Half Adder and Full Adder

Student Name: ____ Student ID: _____________

Half Adder:

C = X.Y

S = X ⊕ Y = (X'.Y) + (X.Y')

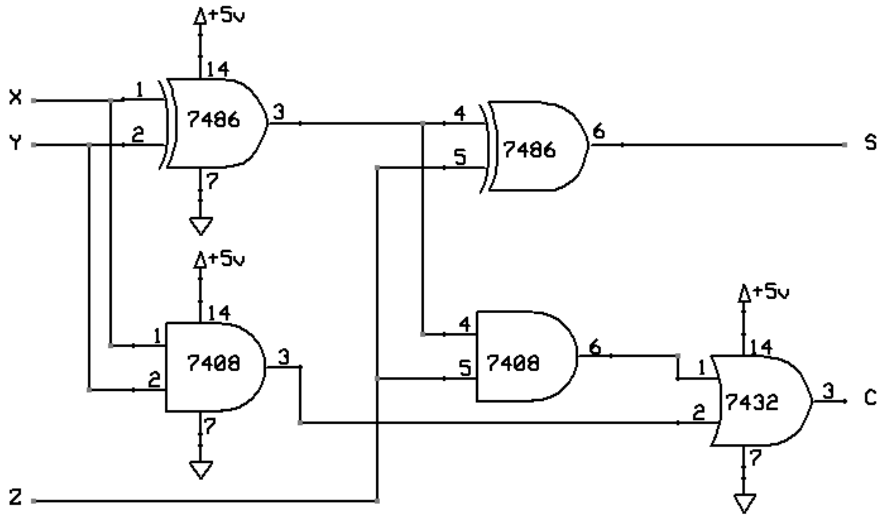


X	Y	C	S
0	0		
0	1		
1	0		
1	1		

Full Adder:

C =

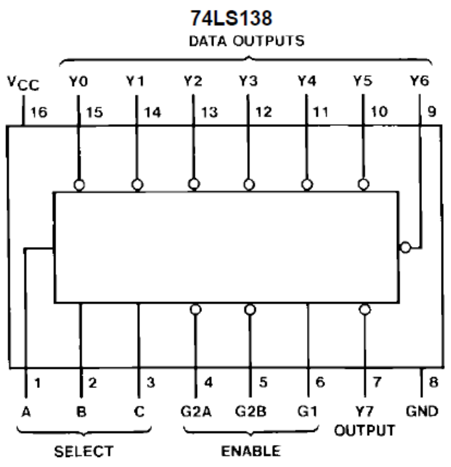
S =

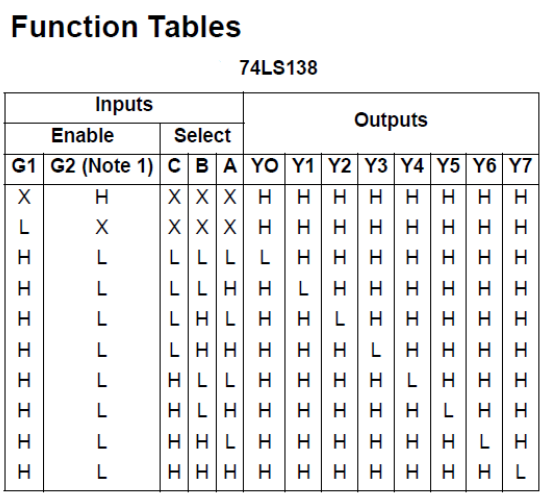


Z (Cin)	**X**	**Y**	**C**	**S**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Experiment 5 Worksheet: Decoders

Student Name: ____ Student ID: ___________





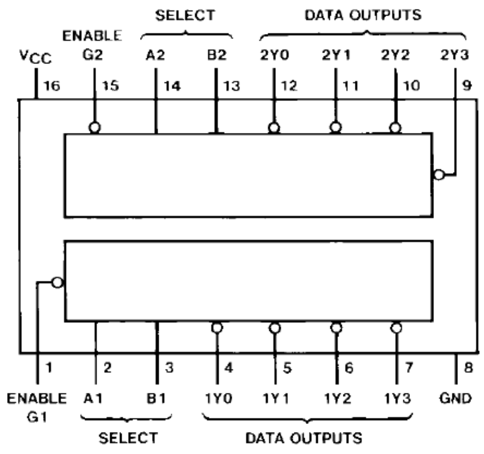
Function Tables

74LS138

Inputs					Outputs							
Enable		Select										
G1	G2 (Note 1)	C	B	A	YO	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

G1	**G2A**	**G2B**	**C**	**B**	**A**	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
L	X	X	X	X	X								
X	H	X	X	X	X								
X	X	H	X	X	X								
H	L	L	L	L	L								
H	L	L	L	L	H								
H	L	L	L	H	L								
H	L	L	L	H	H								
H	L	L	H	L	L								
H	L	L	H	L	H								
H	L	L	H	H	L								
H	L	L	H	H	H								

74LS139



74LS139

Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

H = HIGH Level
L = LOW Level
X = Don't Care

Note 1: G2 = G2A + G2B

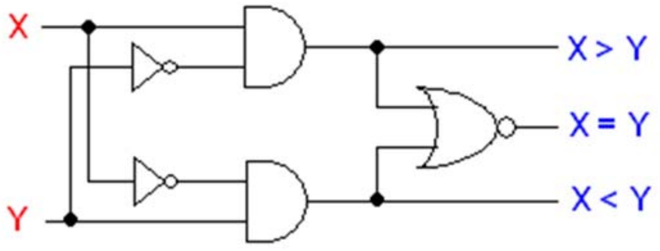
G1	**B1**	**A1**	$1Y_0$	$1Y_1$	$1Y_2$	$1Y_3$
H	X	X				
L	L	L				
L	L	H				
L	H	L				
L	H	H				

G2	**B2**	**A2**	$2Y_0$	$2Y_1$	$2Y_2$	$2Y_3$
H	X	X				
L	L	L				
L	L	H				
L	H	L				
L	H	H				

Experiment 6 Worksheet: Comparators and Multiplexers

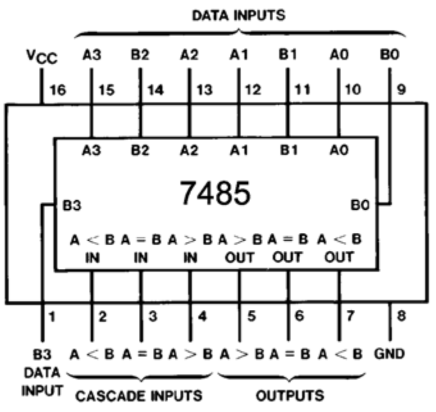
Student Name: ____ Student ID: _____________

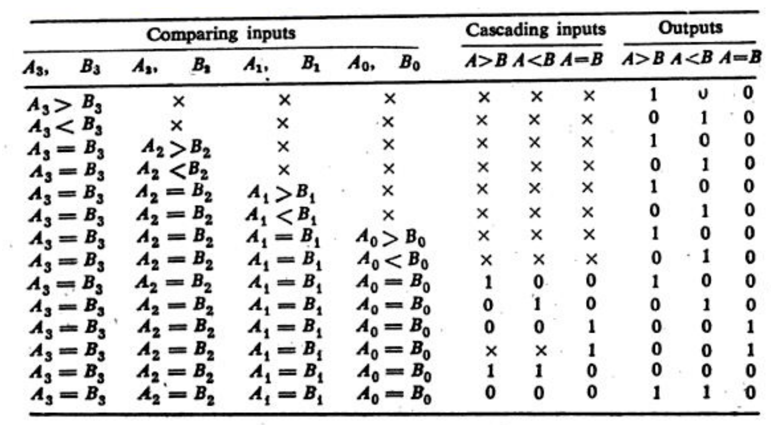
1 bit comparator



X	Y	E (X =Y)	L (X<Y)	G(X>Y)
L	L			
L	H			
H	L			
H	H			

4 bit comparator

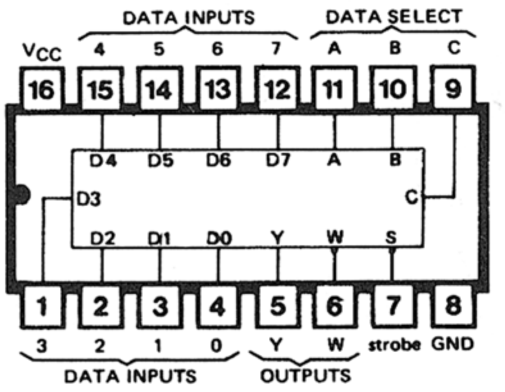


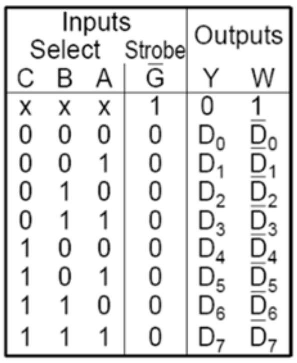


Comparing inputs				Cascading inputs			Outputs		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$A>B$	$A<B$	$A=B$	$A>B$	$A<B$	$A=B$
$A_3 > B_3$	×	×	×	×	×	×	1	0	0
$A_3 < B_3$	×	×	×	×	×	×	0	1	0
$A_3 = B_3$	$A_2 > B_2$	×	×	×	×	×	1	0	0
$A_3 = B_3$	$A_2 < B_2$	×	×	×	×	×	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	×	×	×	×	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	×	×	×	×	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	×	×	×	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	×	×	×	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	0	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0	0	1	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	×	×	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	1	0	0	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	0	1	1	0

A3	A2	A1	A0	B3	B2	B1	B0	A=B	A<B	A>B

8 to 1 line Multiplexer 74151





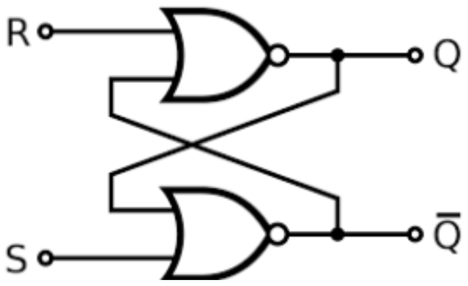
Inputs				Outputs	
Select			Strobe		
C	B	A	$\overline{G}$	Y	W
x	x	x	1	0	1
0	0	0	0	D_0	$\overline{D_0}$
0	0	1	0	D_1	$\overline{D_1}$
0	1	0	0	D_2	$\overline{D_2}$
0	1	1	0	D_3	$\overline{D_3}$
1	0	0	0	D_4	$\overline{D_4}$
1	0	1	0	D_5	$\overline{D_5}$
1	1	0	0	D_6	$\overline{D_6}$
1	1	1	0	D_7	$\overline{D_7}$

C	**B**	**A**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**Y**
L	L	L								▨	
L	L	H							▨		
L	H	L						▨			
L	H	H					▨				
H	L	L				▨					
H	L	H			▨						
H	H	L		▨							
H	H	H	▨								

Experiment 8 Worksheet: Latches and Flip Flops

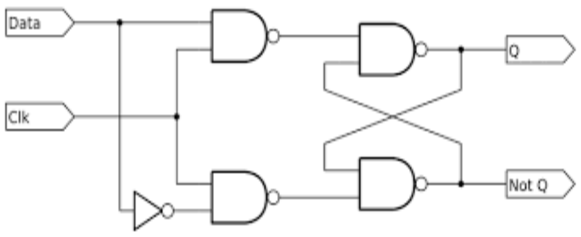
Student Name: ____ Student ID: _____________

S-R Latch



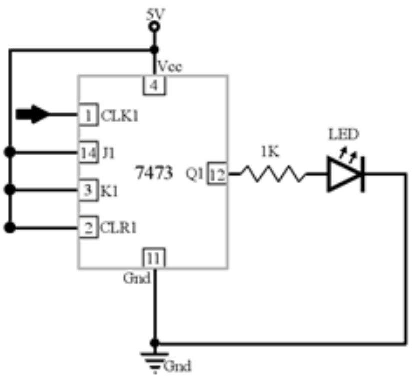
S	R	Q	Q'	Action
L	L	**Q**	**Q'**	Hold
L	H			Reset
H	L			Set
H	H	L	L	Not allowed

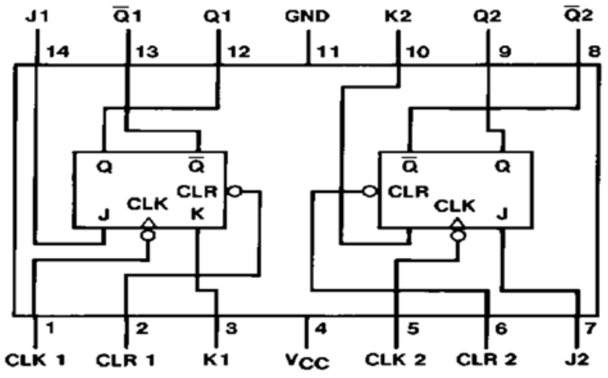
S-R Latch to D Latch



Clk	Data	Q	Q'	Action
L	X	**Q**	**Q'**	Hold
H	H	L	H	Reset
H	L	H	L	Set

J-K Flip Flop Toggle Mode





Inputs				Outputs	
CLR	CLK	J	K	J	K
L	X	X	X	L	H
H	↓	L	L	Q	Q'
H	↓	H	L	H	L
H	↓	L	H	L	H
H	↓	H	H	Toggle	
H	H	X	X	Q	Q'