Digital circuit design notes

Vikas Dhiman for ECE275[1]

January 16, 2025

[1] The notes are from the following books [1, 3]. Not intended for distribution outside the class.

Contents

ToDo

1. 2. 3.

Chapter 1

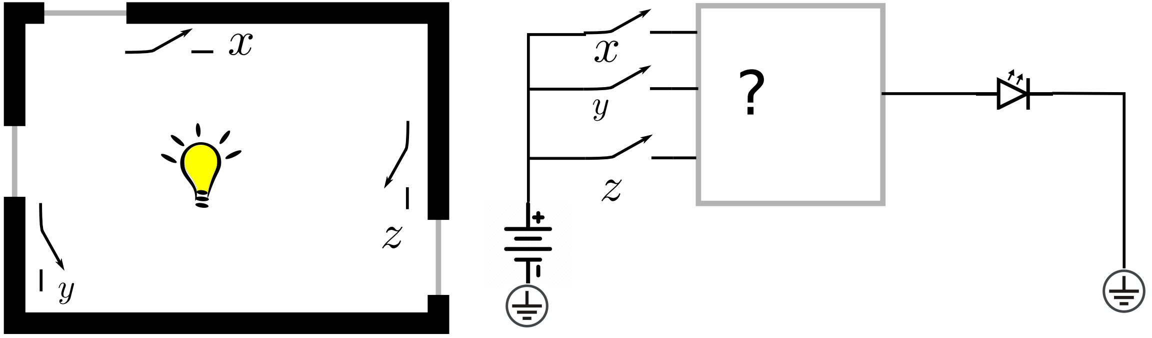
Boolean Algebra

1.1 Learning objectives

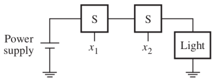
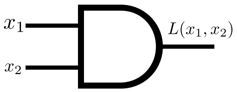
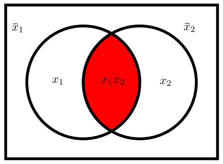
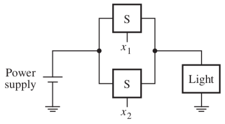
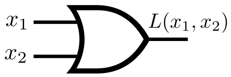
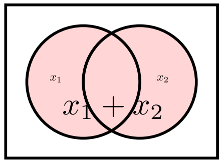
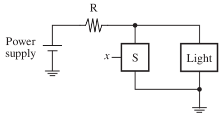
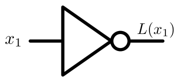
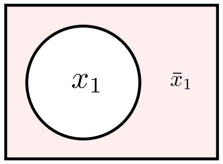
1. Representing digital circuits
2. Converting between different notations: Boolean expression, logic networks and switching circuits
3. Converting between different logic network specifications: truth table, minterm, maxterms, product of sums canonical form and sum of product canonical form.
4. Introduce truth tables as Behavioral Verilog

1.2 Motivating Problem

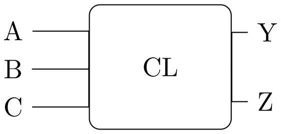
Example 1. *Assume that a large room has three doors and that a switch near each door controls a light in the room. It has to be possible to turn the light on or off by changing the state of any one of the switches.*



1.3 Basic Gates and notations summary

<table>
<tr><th>Name</th><th>C/Verilog</th><th>Boolean expr.</th><th>Truth Table</th><th>Switching circuit</th><th>(ANSI) symbol</th><th>Venn diagram</th></tr>
<tr><td>AND Gate</td><td><code>L = x1 & x2</code></td><td>$L = x_1 \cdot x_2 = x_1 x_2$</td><td>$\begin{array}{cc|c} x_1 & x_2 & x_1 \cdot x_2 \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{array}$</td><td>
</td><td>
</td><td>
</td></tr>
<tr><td>OR Gate</td><td><code>L = x1 | x2</code></td><td>$L = x_1 + x_2$</td><td>$\begin{array}{cc|c} x_1 & x_2 & x_1 + x_2 \\ \hline 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{array}$</td><td>
</td><td>
</td><td>
</td></tr>
<tr><td>NOT Gate</td><td><code>L = ∼ x1</code></td><td>$L = \bar{x}_1 = x_1'$</td><td>$\begin{array}{c|c} x_1 & \bar{x}_1 \\ \hline 0 & 1 \\ 1 & 0 \end{array}$</td><td>
</td><td>
</td><td>
</td></tr>
</table>

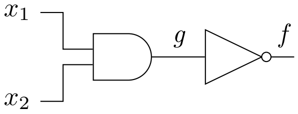
1.4 Digital circuits or networks



$$Y = F(A, B, C) \qquad Z = G(A, B, C)$$

1.5 Two input networks

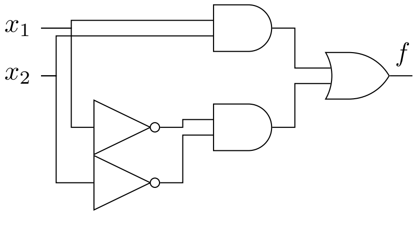
Example 2. *Convert the following (ANSI) network into a Boolean expression, a truth table and a Venn diagram.*



Example 3. *Convert the following Boolean expression into a (ANSI) network, a truth table and a Venn diagram:*

$$f = \overline{x_1 + x_2}$$

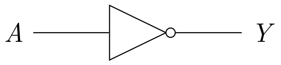
Problem 1. *Convert the following (ANSI) network into a Boolean expression, a truth table and a Venn diagram.*

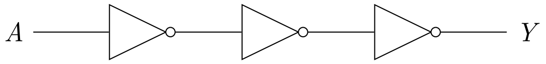


Example 4. *Convert the following Boolean expression into a network, a truth table and a Venn diagram:*

$$f = x_1\bar{x}_2 + \bar{x}_1 x_2$$

Problem 2. *Can two different circuits have the same truth table? Can two different truth tables have the same circuit? Consider the following two circuits for example*



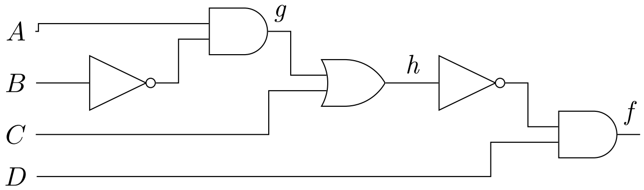


How about Venn digrams?

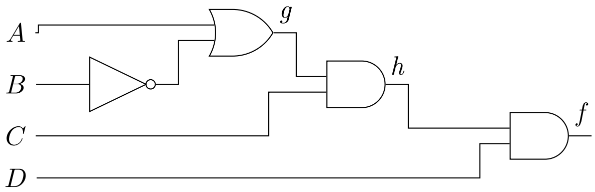
Remark 1. *Truth tables and Venn diagrams define* what *the combinational circuit should do. Truth tables define output for every input. Boolean expression and networks define* how *to achieve the desired input output relationship.*

1.6 Multi-input networks

Example 5. *Convert the following (ANSI) network into a Boolean expression and a truth table.*



Problem 3. *Convert the following (ANSI) network into a Boolean expression and a truth table.*



1.7 Minterms and Maxterms

1.7.1 Minterms

Minterm is a product involving all inputs (or complements) to a function. Every row of a truth table has a corresponding minterm. Minterm is true if and only if the corresponding row in the table is active.

Minterms defined as follows for each row of a two input truth table:

A	B	minterm	minterm name
0	0	$\bar{A}\bar{B}$	m_0
0	1	$\bar{A}B$	m_1
1	0	$A\bar{B}$	m_2
1	1	AB	m_3

Consider a two input circuit whose output Y is given by the truth table:

A	B	Y	minterm	minterm name
0	0	0	$\bar{A}\bar{B}$	m_0
0	1	1	$\bar{A}B$	m_1
1	0	0	$A\bar{B}$	m_2
1	1	1	AB	m_3

then $Y = \bar{A}B + AB = m_1 + m_3 = \sum(1,3)$.
This also gives the *sum of products canonical form.*

Example 6. *What is the minterm m_{13} for a 4-input circuit with inputs x, y, z, w (ordered from MSB to LSB).*

Problem 4. *What is the minterm m_{23} for a 5-input circuit with inputs a, b, c, d, e (ordered from MSB to LSB).*

Example 7. *Convert the following 4-input truth table into sum of minterms and sum of products canonical form.*

minterm name	*A*	*B*	*C*	*D*	*f*
m_0	*0*	*0*	*0*	*0*	*0*
m_1	*0*	*0*	*0*	*1*	*1*
m_2	*0*	*0*	*1*	*0*	*0*
m_3	*0*	*0*	*1*	*1*	*0*
m_4	*0*	*1*	*0*	*0*	*0*
m_5	*0*	*1*	*0*	*1*	*1*
m_6	*0*	*1*	*1*	*0*	*0*
m_7	*0*	*1*	*1*	*1*	*0*
m_8	*1*	*0*	*0*	*0*	*0*
m_9	*1*	*0*	*0*	*1*	*0*
m_{10}	*1*	*0*	*1*	*0*	*0*
m_{11}	*1*	*0*	*1*	*1*	*0*
m_{12}	*1*	*1*	*0*	*0*	*0*
m_{13}	*1*	*1*	*0*	*1*	*1*
m_{14}	*1*	*1*	*1*	*0*	*0*
m_{15}	*1*	*1*	*1*	*1*	*0*

Problem 5. *Convert the following 4-input truth table into sum of minterms and sum of products canonical form.*

minterm name	*A*	*B*	*C*	*D*	*f*
m_0	*0*	*0*	*0*	*0*	*0*
m_1	*0*	*0*	*0*	*1*	*0*
m_2	*0*	*0*	*1*	*0*	*0*
m_3	*0*	*0*	*1*	*1*	*1*
m_4	*0*	*1*	*0*	*0*	*0*
m_5	*0*	*1*	*0*	*1*	*0*
m_6	*0*	*1*	*1*	*0*	*0*
m_7	*0*	*1*	*1*	*1*	*1*
m_8	*1*	*0*	*0*	*0*	*0*
m_9	*1*	*0*	*0*	*1*	*0*
m_{10}	*1*	*0*	*1*	*0*	*0*
m_{11}	*1*	*0*	*1*	*1*	*1*
m_{12}	*1*	*1*	*0*	*0*	*0*
m_{13}	*1*	*1*	*0*	*1*	*1*
m_{14}	*1*	*1*	*1*	*0*	*1*
m_{15}	*1*	*1*	*1*	*1*	*0*

1.7.2 Maxterms

Maxterm is a sum involving all inputs (or complements) to a function. Every row of a truth table has a corresponding maxterm. Minterm is false if and only if the corresponding row in the table is active.

Maxterms are defined as follows for each row of a two input truth table:

A	B	maxterm	maxterm name
0	0	$A+B$	M_0
0	1	$A+\bar{B}$	M_1
1	0	$\bar{A}+B$	M_2
1	1	$\bar{A}+\bar{B}$	M_3

Consider a two input circuit whose output Y is given by the truth table:

A	B	Y	maxterm	maxterm name
0	0	0	$A+B$	M_0
0	1	1	$A+\bar{B}$	M_1
1	0	0	$\bar{A}+B$	M_2
1	1	1	$\bar{A}+\bar{B}$	M_3

then $Y = (A+B)(\bar{A}+B) = M_0 M_2$.

Writing a functional specification in terms of minterms is also called product of sums canonical form.

Example 8. *Write the maxterm M_{11} for 4-input Boolean function with the ordered inputs A, B, C, D.*

Example 9. *Convert the following 4-input truth table into product of maxterms and product of sums canonical form.*

maxterm name	*A*	*B*	*C*	*D*	*f*
M_0	*0*	*0*	*0*	*0*	*0*
M_1	*0*	*0*	*0*	*1*	*0*
M_2	*0*	*0*	*1*	*0*	*0*
M_3	*0*	*0*	*1*	*1*	*1*
M_4	*0*	*1*	*0*	*0*	*0*
M_5	*0*	*1*	*0*	*1*	*0*
M_6	*0*	*1*	*1*	*0*	*0*
M_7	*0*	*1*	*1*	*1*	*1*
M_8	*1*	*0*	*0*	*0*	*0*
M_9	*1*	*0*	*0*	*1*	*0*
M_{10}	*1*	*0*	*1*	*0*	*0*
M_{11}	*1*	*0*	*1*	*1*	*1*
M_{12}	*1*	*1*	*0*	*0*	*0*
M_{13}	*1*	*1*	*0*	*1*	*1*
M_{14}	*1*	*1*	*1*	*0*	*1*
M_{15}	*1*	*1*	*1*	*1*	*0*

Problem 6. *Convert the following 4-input truth table into product of maxterms and products of sums canonical form.*

maxterm name	A	B	C	D	f
M_0	0	0	0	0	0
M_1	0	0	0	1	1
M_2	0	0	1	0	1
M_3	0	0	1	1	1
M_4	0	1	0	0	1
M_5	0	1	0	1	0
M_6	0	1	1	0	1
M_7	0	1	1	1	1
M_8	1	0	0	0	0
M_9	1	0	0	1	1
M_{10}	1	0	1	0	1
M_{11}	1	0	1	1	1
M_{12}	1	1	0	0	0
M_{13}	1	1	0	1	1
M_{14}	1	1	1	0	1
M_{15}	1	1	1	1	0

Example 10. *Write the 3-input truth table for the function* $f = m_2 + m_3 + m_7$.

Problem 7. *Write the 3-input truth table for the function* $f = M_4 M_5 M_7$.

Problem 8. *Write the truth table for the function* $f = \bar{A}B\bar{C} + AB\bar{C}$.

1.8 Karnaugh maps

1.8.1 Two input K-maps

A B	0	1
0	m_0	m_2
1	m_1	m_3

1.8.2 Three input K-maps

AB C	00	01	11	10
0	m_0	m_2	m_6	m_4
1	m_1	m_3	m_7	m_5

1.8.3 Four input K-maps

AB CD	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

1.8.4 Five input K-maps

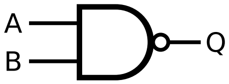
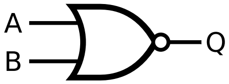
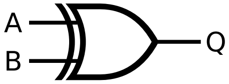
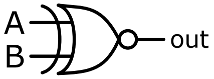
A = 0

BC DE	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

A = 1

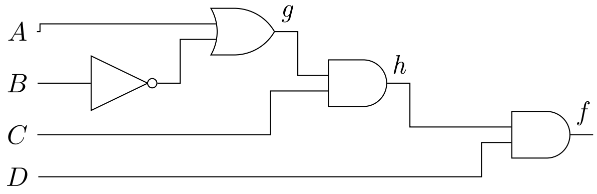
BC DE	00	01	11	10
00	m_{16}	m_{20}	m_{28}	m_{24}
01	m_{17}	m_{21}	m_{29}	m_{25}
11	m_{19}	m_{23}	m_{31}	m_{27}
10	m_{18}	m_{22}	m_{30}	m_{26}

1.9 More Gates and notations summary

Name	C/Verilog	Boolean expr.	Truth Table	(ANSI) symbol	K-map
NAND Gate	`Q = ~(x1 & x2)`	$Q = \overline{x_1 \cdot x_2} = \overline{x_1 x_2}$	x_1 x_2 | $\overline{x_1 \cdot x_2}$ 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0		B\A 0 1 0 | 1 | 1 1 | 1 | 0
NOR Gate	`Q = ~(x1 | x2)`	$Q = \overline{x_1 + x_2}$	x_1 x_2 | $\overline{x_1 + x_2}$ 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0		B\A 0 1 0 | 0 | 0 1 | 0 | 1
XOR Gate	`Q = x1 ^ x2`	$Q = x_1 \oplus x_2$	x_1 x_2 | $x_1 \oplus x_2$ 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0		B\A 0 1 0 | 0 | 1 1 | 1 | 0
XNOR Gate	`Q = ~(x1 ^ x2)`	$Q = \overline{x_1 \oplus x_2}$	x_1 x_2 | $\overline{x_1 \oplus x_2}$ 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1		B\A 0 1 0 | 1 | 0 1 | 0 | 1

Example 11. *Convert the following Boolean expression into a K-map.* $f = \overline{A\bar{B} + CD}$

Problem 9. *Convert the following logic circuit into a K-map.*



1.10 Boolean Algebra

1.10.1 Axioms of Boolean algebra

1. $0 \cdot 0 = 0$

2. $1 + 1 = 1$

3. $1 \cdot 1 = 1$
4. $0 + 0 = 0$
5. $0 \cdot 1 = 1 \cdot 0 = 0$
6. $\bar{0} = 1$
7. $\bar{1} = 0$
8. $x = 0$ if $x \neq 1$
9. $x = 1$ if $x \neq 0$

1.10.2 Single variable theorems (Prove by drawing K-maps)

1. $x \cdot 0 = 0$
2. $x + 1 = 1$
3. $x \cdot 1 = x$
4. $x + 0 = x$
5. $x \cdot x = x$
6. $x + x = x$
7. $x \cdot \bar{x} = 0$

8. $x + \bar{x} = 1$

9. $\bar{\bar{x}} = x$

Remark 2 (Duality)**.** *Swap + with · and 0 with 1 to get another theorem*

1.10.3 Two and three variable properties (Prove by K-maps)

1. Commutative: $x \cdot y = y \cdot x$, $x + y = y + x$

2. Associative: $x \cdot (y \cdot z) = (x \cdot y) \cdot z$, $x + (y + z) = (x + y) + z$

3. Distributive: $x \cdot (y + z) = x \cdot y + x \cdot z$, $x + y \cdot z = (x + y) \cdot (y + z)$

4. Absorption: $x + x \cdot y = x$, $x \cdot (x + y) = x$

5. Combining: $x \cdot y + x \cdot \bar{y}$, $(x + y) \cdot (x + \bar{y}) = x$

6. DeMorgan's theorem: $\overline{x \cdot y} = \bar{x} + \bar{y}$, $\overline{x + y} = \bar{x} \cdot \bar{y}$.

7. Concensus:

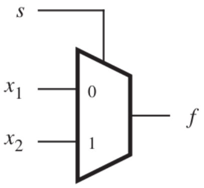
   (a) $x + \bar{x} \cdot y = x + y$

   (b) $x \cdot (\bar{x} + y) = x \cdot y$

   (c) $x \cdot y + y \cdot z + \bar{x} \cdot z = x \cdot y + \bar{x} \cdot z$

   (d) $(x + y) \cdot (y + z) \cdot (\bar{x} + z) = (x + y) \cdot (\bar{x} + z)$

Example 12 (Multiplexer)**.** *Multiplexer is a circuit used to select one of the input lines x_1 and x_2 based only select input s. When $s = 0$, x_1 is selected, x_2 is selected otherwise. Find a boolean expression and a circuit for multiplexer*



s	$f(s, x_1, x_2)$
0	x_1
1	x_2

Example 13. *Simplify $f = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}\bar{C}$ using boolean algebra.*

Example 14. *Simplify $f = \bar{A}\bar{A}\bar{C} + \bar{A}\bar{B}C$ using K-maps.*

Chapter 2

Verilog and Schematics

Contents

2.1 Lab Overview

There are two parts in this lab. For the first part, the goal of will be to install the Intel Quartus software, and then utilize it to program the provided Altera DE0 development FPGAs with an example Verilog program.

The installation can be somewhat finicky, so make sure to perform all the steps and get the correct software version.

While there is a Linux version of Quartus, it has not been tested for this class, so use that at your own risk. There is no native Mac OS support.

For the second part, The goal is to utilize the ModelSim simulator to simulate your Verilog code without the use of the FPGA. This is an incredibly important part of FPGA design, as it can be very difficult to diagnose issues with your code when running on the FPGA. The simulator allows you to see specific states of logic with very tight timing constraints on inputs and outputs.

2.2 Installing Quartus

Important Note

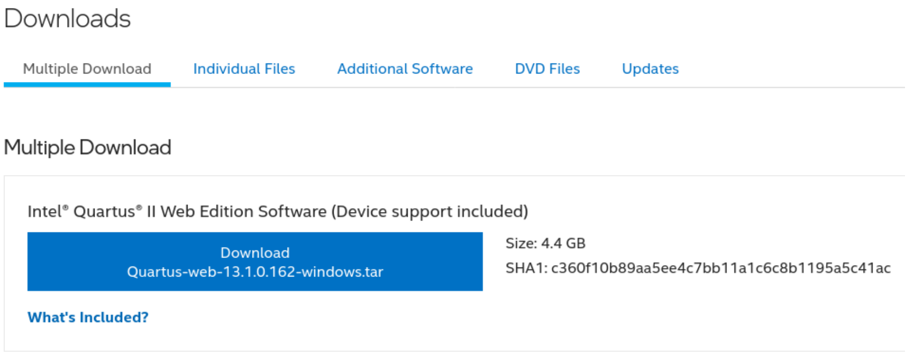
If you have **Intel-based Mac**, then you can install Parallels VM and Windows with the university license. Make sure you learn about USB Passthrough.
If you have **M2 Mac (arm64)**, then you are in trouble. The best solution so far is to use UTM which is pretty slow. Install Windows 10 on top of UTM. Calvin Benider (ECE274-F2022) noted that Windows 11 did not work in UTM. Make sure you learn about USB passthrough. He also warns the user about corrupting the virtual machine, if you do not shut it down properly. He wrote a short guide on Quartus for Mac.
Once you have windows, proceed as normal.
One route that is unexplored is dual booting Ubuntu arm64 on M2 Mac and using QEMU or Box64 on Ubuntu to run Quartus x86 for Linux.

2.2.1 Download and install

We will be utilizing the Quartus II 13.1 software for this class. This version has been tested to be compatible with DE0 FPGA boards. You can download it from here: `https://www.intel.com/content/www/us/en/software-kit/666221/intel-quartus-ii-web-edition-design-software-version-13-1.html`

You will need to select the combined tab to download the "multiple download" software, so you will not have to download device support for the FPGA separately. Make sure you are downloading the 13.1 Web Edition.

Extracting and installing Once the download is finished, extract the tar file using WinRaR or 7-zip. With 7-zip, you can right-click→7-Zip→"Extract to Quartus-web-13.1.0.162-windows
". You should see a file `setup.bat` and a folder `components`. Double-click on `setup.bat` to install the Quartus II software. When asked about components to install, make sure all the devices,



Downloads

Multiple Download | Individual Files | Additional Software | DVD Files | Updates

Multiple Download

Intel® Quartus® II Web Edition Software (Device support included)

Download
Quartus-web-13.1.0.162-windows.tar

Size: 4.4 GB
SHA1: c360f10b89aa5ee4c7bb11a1c6c8b1195a5c41ac

What's Included?

Figure 2.1: Select the Combined Installation and Quartus Version to Download

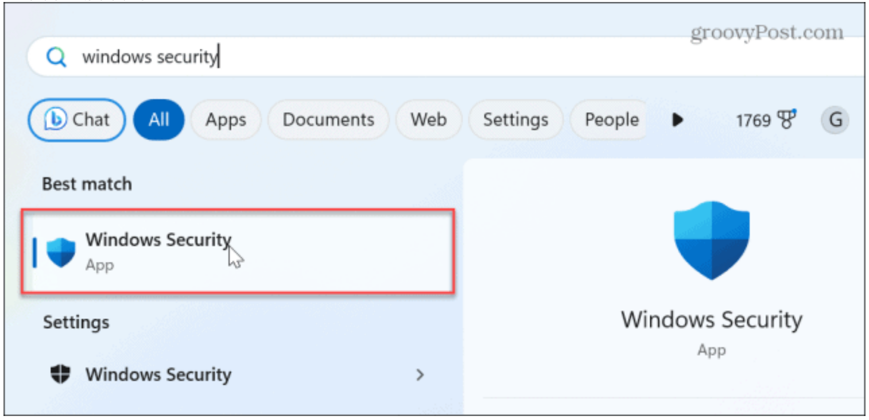
especially `Cyclone III` is selected. Take note of your installation path, you will need it for the next step. For me, it is `C:\altera\13.1`.

2.2.2 Setup the USB Blaster Driver

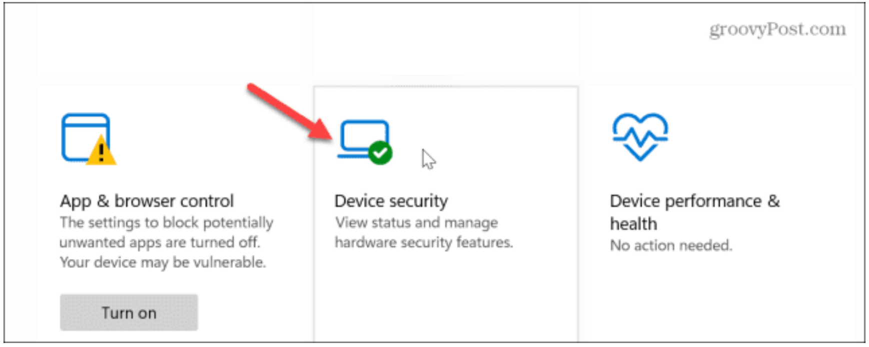
Windows 11 users: Disable the core isolation memory integrity

Please disable core isolation memory integrity as it causes problems with Quartus. We follow instructions from here.

1. Press the Windows key, type Windows Security, and select the top result under the Best match section.



2. Select the Device security option from the main Security at a glance screen.

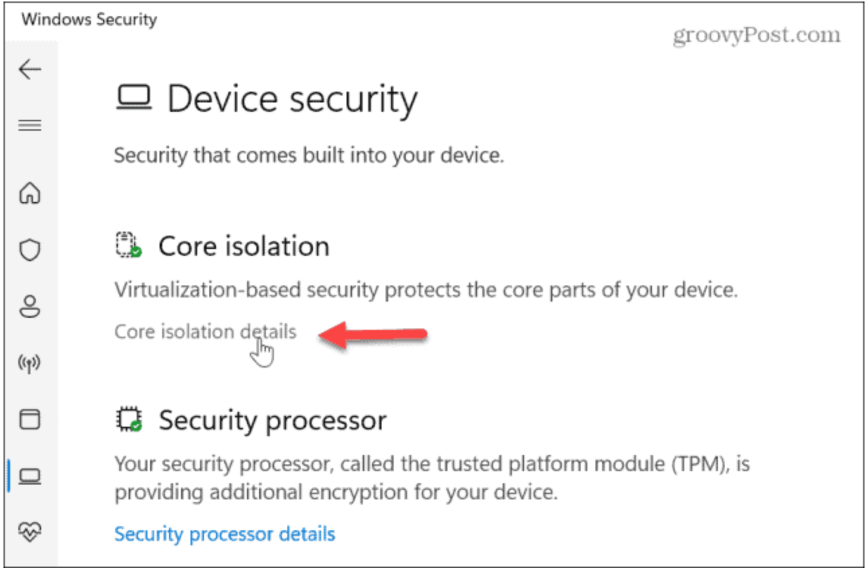


The settings to block potentially unwanted apps are turned off. Your device may be vulnerable.

View status and manage hardware security features.

No action needed.

3. Click the Core isolation details link under the Core isolation section.



Windows Security

Device security

Security that comes built into your device.

Core isolation

Virtualization-based security protects the core parts of your device.

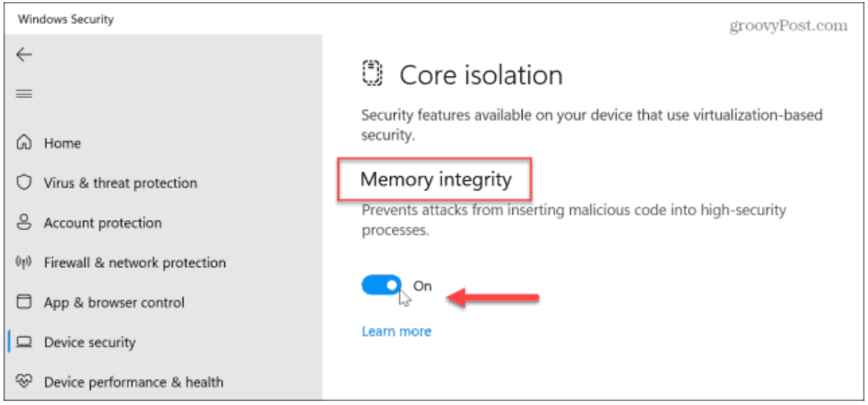
Core isolation details

Security processor

Your security processor, called the trusted platform module (TPM), is providing additional encryption for your device.

Security processor details

4. Enable or disable the Memory integrity feature by toggling its switch on or off.



Windows Security

Core isolation

Security features available on your device that use virtualization-based security.

Memory integrity

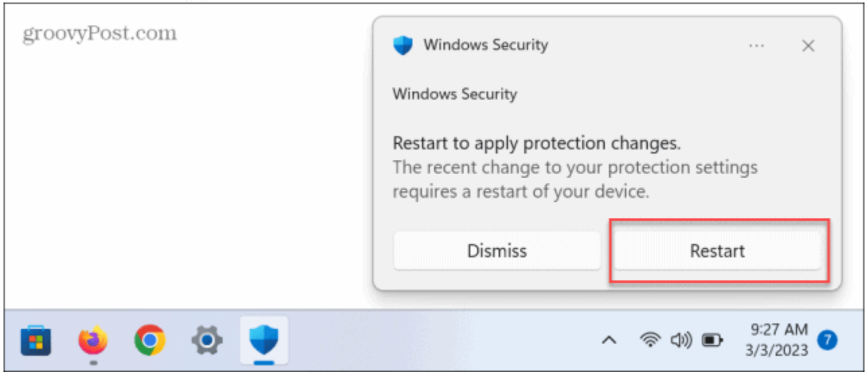
Prevents attacks from inserting malicious code into high-security processes.

On

Learn more

groovyPost.com
Home
Virus & threat protection
Account protection
Firewall & network protection
App & browser control
Device security
Device performance & health

5. If you turn memory integrity off, you will need to restart your Windows system, and a message to restart will appear in the taskbar's notification area.



Windows Security

Windows Security

Restart to apply protection changes.
The recent change to your protection settings requires a restart of your device.

Setup the USB Blaster Driver

Instructions from `http://www.terasic.com.tw/wiki/Altera_USB_Blaster_Driver_Installation_Instructions`

For Windows 11 and a generic way of USB Blaster Driver setup on Windows

1. Search for Device Manager and open it.
2. Under “Universal Serial Bus controllers” or “Other Devices” you’ll find Altera USB. Right click on Altera USB Blaster and Click Update Driver >Browse my Computer for Drivers.
3. Click on Browse and go to `C:\altera\13.1\quartus\drivers` and click ‘OK’ and ‘INSTALL’.

For Windows 10

1. Use the provided USB cable to connect the DE0 board to your computer
2. Open your control panel, and then go to devices and printers
3. Under Unspecified, USB Blaster should be listed. Right click on this and then select Properties
4. Select the Hardware tab and select Properties
5. A new window should pop up with the General tab already selected. Select Change Settings.
6. Again a new window should pop up with the General tab already selected. Select Update.
7. Select Browse my computer for driver software.
8. Find <Path to Quartus II installation>\quartus\drivers\ Check notes below for additional path information
9. Select OK. Make sure the proper path was selected then select Next.
10. If the Windows security window pops up check the Always trust software from “Altera Corporation”box and select Install.

- Note 1: Your Altera file is located at the location you selected when you first installed Quartus. The location listed in this document is the default location) `C:\altera\13.1`.
- Note 2: Stop at the drivers folder, i.e., do NOT go deeper by opening a folder within the drivers folder)

2.2.3 Creating a New Project with the Quartus Software

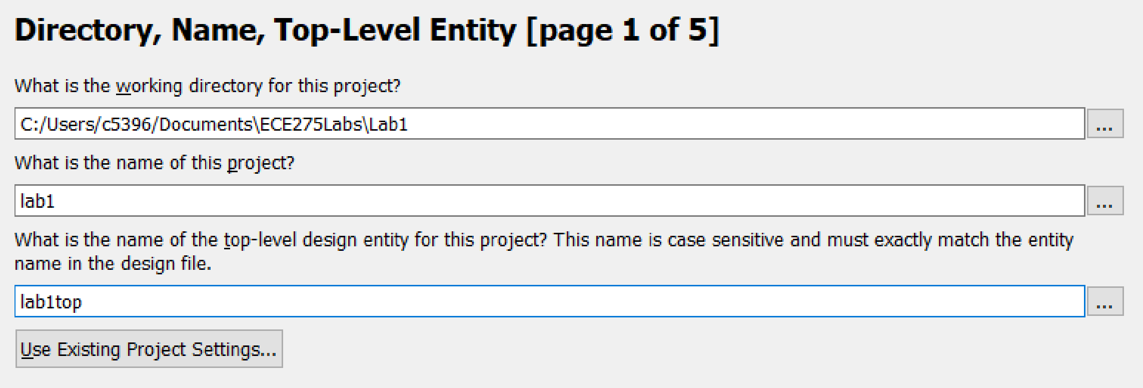
The Quartus software should be in your start menu programs list in a folder named Altera 13.1 Web Edition. Run the Quartus II 13.1 software in this folder.

After the software is started, you will need to create a new project. You can do this by selecting ”Create a New Project” in the initial pop up window, or by going to the file menu at the top left, going to new, and then selecting the new project option.

In the project setup wizard, you will need to set up the project name hierarchy, and select the correct FPGA model.

The first window should ask for the project working directory, project name, and top-level name.

Try to use underscores instead of spaces in the project name and path, and avoid use of any special characters. The project save path will be a folder that contains multiple files that relate to the project, so it is a good idea to create a unique folder in a larger folder structure for each lab section. For example if a lab had multiple parts with unique code, the path could be <Path to Your Documents Folder>\ECE275Labs\Lab1\Lab1Part1 for the first part, and then <Path to Your Documents Folder>\ECE275Labs\Lab1\Lab1Part2 for the second part. The next box asks for the name of the project. Give it a short relevant name. The last box asks for the top level, generally an easy name to use is the project name with top added at the end (ex. Project: lab2part1, Top Level: lab2part1top). This makes it easy to keep track of your top level module name (you will use the top level name similar to how main is used for the main function in C).



Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

C:/Users/c5396/Documents\ECE275Labs\Lab1

What is the name of this project?

lab1

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

lab1top

Figure 2.2: Set up the project naming hierarchy

The next step will ask you to add files, you can just hit next here.

Next you will need to select the correct FPGA model. You can use the family dropdown to select Cyclone III. Then, in the available devices window, find the model name:

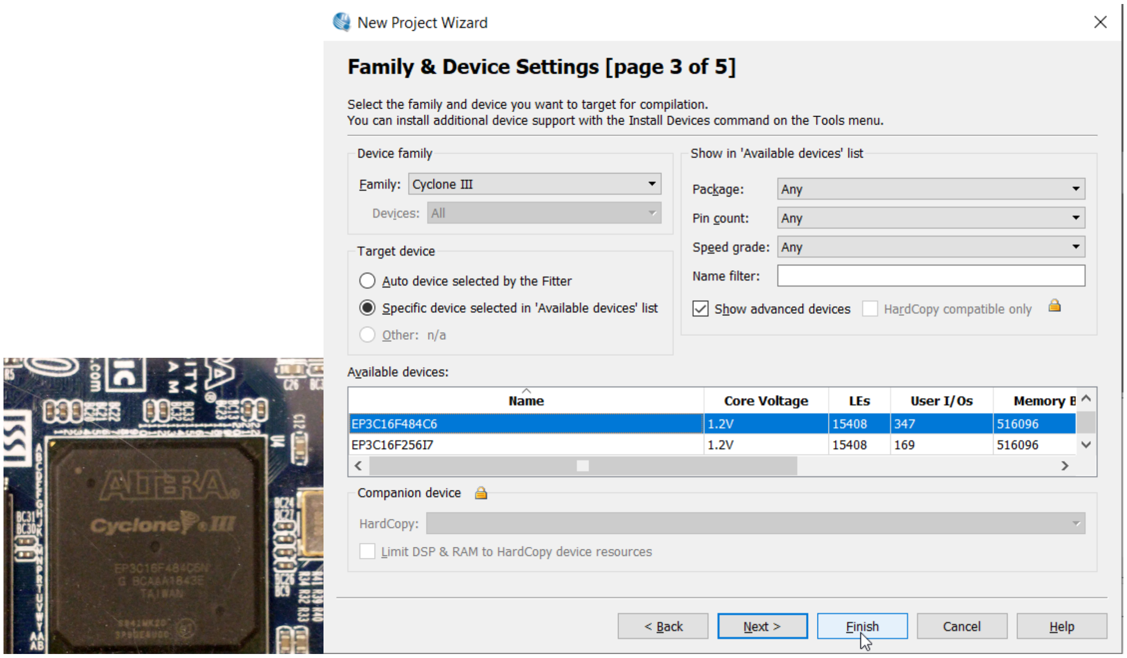
```
FPGA Model Name:  EP3C16F484C6
```

This information is available on the chip of the FPGA board. After selecting the correct model name, you can hit Finish on the New Project Wizard.

2.2.4 Create a New Verilog File and Copy the Code

Verilog is known as a Hardware Description Language (HDL). In future labs, you will be writing your own code in Verilog, but for today's lab it is being provided for you.

To create a new Verilog file, go to the file menu in the top left, go to new, and then in the Window that pops up, select Verilog HDL File
This will create a new blank file, where you can copy and paste the following code. You will need to change the module name at the top to the name you chose for your top level in the wizard.



New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

Device family

Family: Cyclone III

Devices: All

Target device

○ Auto device selected by the Fitter

● Specific device selected in 'Available devices' list

○ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Speed grade: Any

Name filter:

☑ Show advanced devices □ HardCopy compatible only

Available devices:

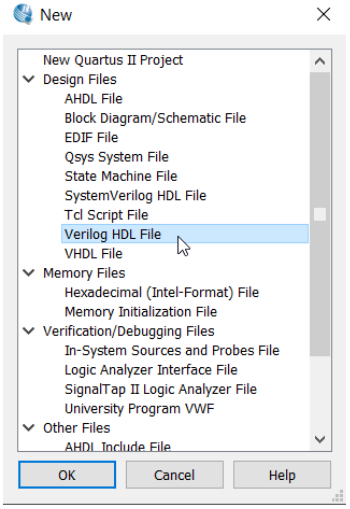
Name	Core Voltage	LEs	User I/Os	Memory B
EP3C16F484C6	1.2V	15408	347	516096
EP3C16F256I7	1.2V	15408	169	516096

Companion device

HardCopy:

□ Limit DSP & RAM to HardCopy device resources

Figure 2.3: Select the correct FPGA model, EP3C16F484C6



New

New Quartus II Project
Design Files
- AHDL File
- Block Diagram/Schematic File
- EDIF File
- Qsys System File
- State Machine File
- SystemVerilog HDL File
- Tcl Script File
- Verilog HDL File
- VHDL File

Memory Files
- Hexadecimal (Intel-Format) File
- Memory Initialization File

Verification/Debugging Files
- In-System Sources and Probes File
- Logic Analyzer Interface File
- SignalTap II Logic Analyzer File
- University Program VWF

Other Files
- AHDL Include File

Figure 2.4: Create a new Verilog HDL File

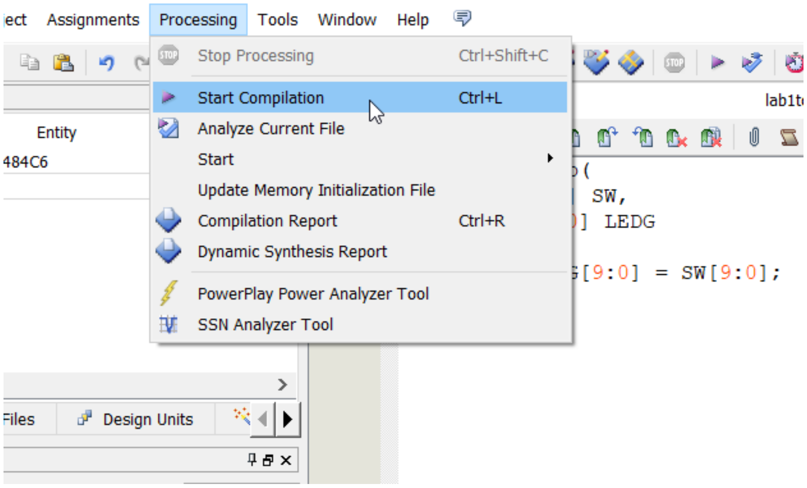
```
// module in verilog is like a function in C
module lab1top(
    // input specifies the input wires to a circuit
    input [9:0] SW,
    // output specifices the output wires from a circuit
    // In C language these will be return  variable
    output [9:0] LEDG
);
    // assign keyword creates continuous assignment.
    // It connects the two variables with a wire.
    // Whenever SW is updated, then LEDG is updated (instantly).
    assign LEDG[9:0] = SW[9:0];
endmodule // Verilog
```

After pasting the code, hit control+s to save the file. You can leave the file name as your top-level name.

2.2.5 Run Compilation

The next step is to run compilation. You will have to run this before downloading to the FPGA anytime you make any code updates, or pin assignment changes. The compilation creates the files that are downloaded to the FPGA from your project files.

You can run compilation by hitting the purple play icon in the center top of the toolbar, or by going to the processing menu at the top, and selecting Start Compilation.

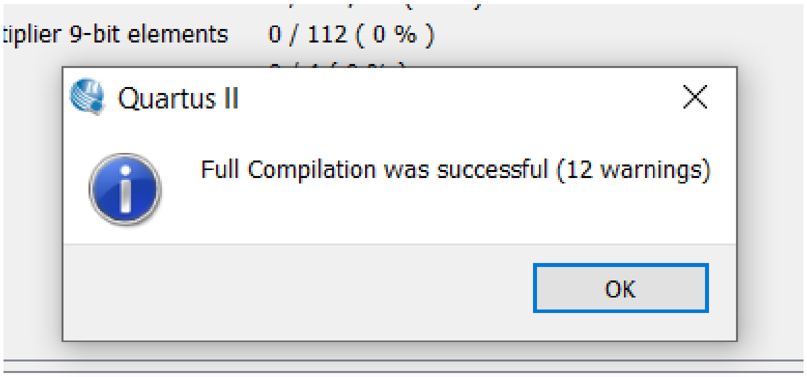


```
 (
  SW,
 ] LEDG

 [9:0] = SW[9:0];
```

Figure 2.5: Start Project Compilation

When the compilation finished, a box should pop up indicating there were 12 warnings. This is ok, as long as you do not have any errors. Errors are higher level than warnings, and would indicate you did not copy something correctly with the code, or you forgot to change the module name to your top-level name.



```
tiplier 9-bit elements    0 / 112 ( 0 % )
```

Full Compilation was successful (12 warnings)

Figure 2.6: Compilation was successful, even with warnings

2.2.6 Set Pin Assignments

The last step before downloading to the FPGA is to set the pin assignments. This essentially references pins on the FPGA to their real world device connection. The pins are laid out in a grid on the FPGA, so are referenced with names such as AA12. This step will associate the switches and LED's that are utilized in the Verilog program to pins they are physically connected to on the FPGA. Detailed information on the pins can be found in the user manual of Altera DE0 User Manual `https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=56&No=364&PartNo=4`

The easiest way to set pin assignments is to use their unique names as the variable names in the verilog code, and then use pre-made qsf files to create the references. The .qsf file for the board in this class will provided to you. For this lab, you only need a few lines from the qsf file, and those are provided below:

```
set_location_assignment PIN_B1 -to LEDG[9]
set_location_assignment PIN_B2 -to LEDG[8]
set_location_assignment PIN_C2 -to LEDG[7]
set_location_assignment PIN_C1 -to LEDG[6]
set_location_assignment PIN_E1 -to LEDG[5]
set_location_assignment PIN_F2 -to LEDG[4]
set_location_assignment PIN_H1 -to LEDG[3]
set_location_assignment PIN_J3 -to LEDG[2]
set_location_assignment PIN_J2 -to LEDG[1]
set_location_assignment PIN_J1 -to LEDG[0]
set_location_assignment PIN_D2 -to SW[9]
set_location_assignment PIN_E4 -to SW[8]
set_location_assignment PIN_E3 -to SW[7]
set_location_assignment PIN_H7 -to SW[6]
set_location_assignment PIN_J7 -to SW[5]
set_location_assignment PIN_G5 -to SW[4]
set_location_assignment PIN_G4 -to SW[3]
set_location_assignment PIN_H6 -to SW[2]
set_location_assignment PIN_H5 -to SW[1]
set_location_assignment PIN_J6 -to SW[0]
```

You will need to copy and paste these lines into the Quartus Tcl Console for your project, and then hit enter to run them. The Quartus console can be found below your verilog file in the Quartus window. If it is not showing, you may need to go to the view file menu, go to Utility Windows, and check if the Tcl Console is activated.

You can check if the pin assignments were correctly run by opening the pin planner. This can be found in the assignments menu at the top. Your pins should match the pin layout shown in the picture.

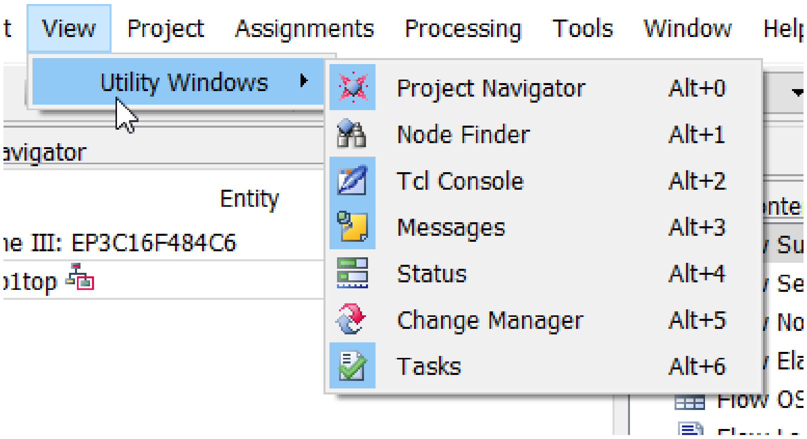
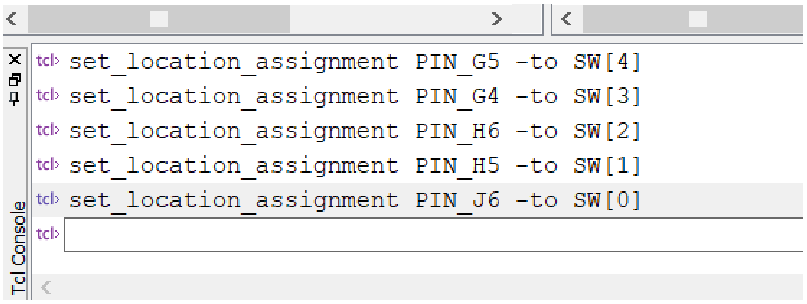


Figure 2.7: Check if the Tcl Console is Activated



```
tcl> set_location_assignment PIN_G5 -to SW[4]
tcl> set_location_assignment PIN_G4 -to SW[3]
tcl> set_location_assignment PIN_H6 -to SW[2]
tcl> set_location_assignment PIN_H5 -to SW[1]
tcl> set_location_assignment PIN_J6 -to SW[0]
tcl>
```

Figure 2.8: Tcl Console After Running Pin Assignments

2.2.7 Rerun Compilation and Program the FPGA

Make sure to first rerun the compilation using the purple play icon or start compilation in the processing menu. After that is finished, go to the tools menu, and select the programmer option.
In the programmer window, the is an option for hardware setup towards the top left. Click on it to open the hardware setup window. Here you will have to select the USB Blaster option. If that is not available, make sure you properly completed the step of installing the USB-Blaster driver, and that the FPGA is on and connected to your computer through USB. Leave the JTAG programming mode selected in the main programmer window.
At this point, you should be able to program your FPGA by selecting the output files .sof file in the programmer window, and then hit Start to program the FPGA. There is a confusing switch on the FPGA labeled RUN and PROG next to the 7-Segment LEDs. This must be in the RUN position to program the board when using JTAG. If the programmer says programming failed, check to see if that is in the RUN position.
If your programmer window does not show any file in the middle to select, you will have to add your compiled project file. You can do this by selecting Add File, and then in the window the pops up double click on the output files folder, and then double clicking on your project's .sof file inside.

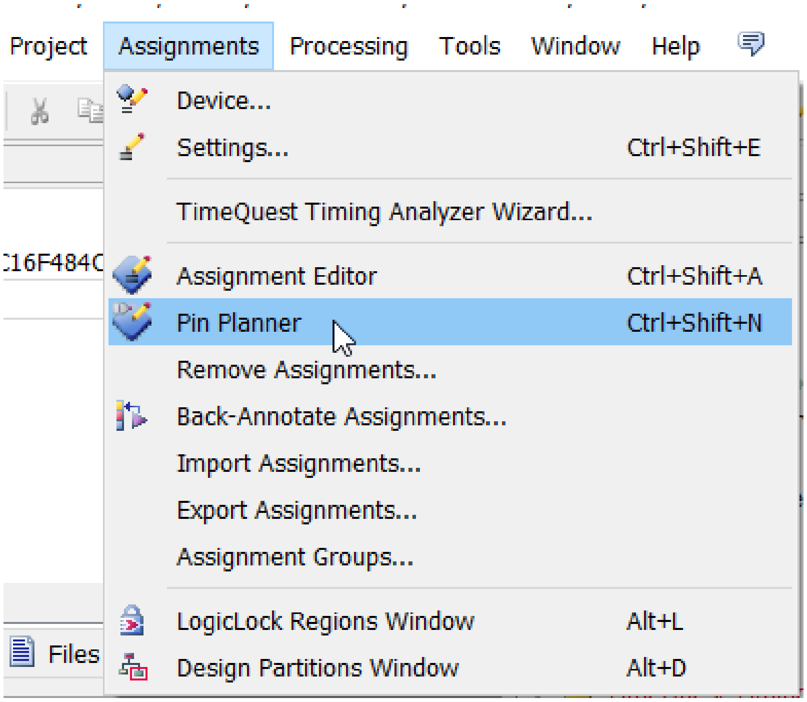
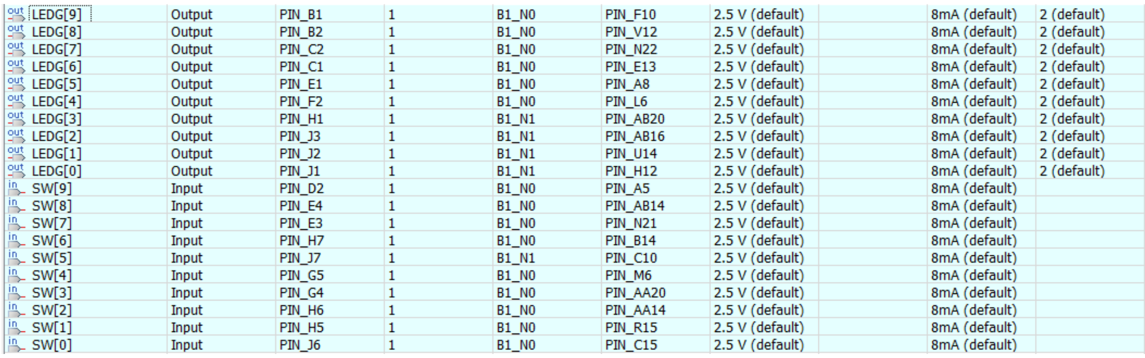


Figure 2.9: Open the Pin Planner



LEDG[9]	Output	PIN_B1	1	B1_N0	PIN_F10	2.5 V (default)		8mA (default)	2 (default)
LEDG[8]	Output	PIN_B2	1	B1_N0	PIN_V12	2.5 V (default)		8mA (default)	2 (default)
LEDG[7]	Output	PIN_C2	1	B1_N0	PIN_N22	2.5 V (default)		8mA (default)	2 (default)
LEDG[6]	Output	PIN_C1	1	B1_N0	PIN_E13	2.5 V (default)		8mA (default)	2 (default)
LEDG[5]	Output	PIN_E1	1	B1_N0	PIN_A8	2.5 V (default)		8mA (default)	2 (default)
LEDG[4]	Output	PIN_F2	1	B1_N0	PIN_L6	2.5 V (default)		8mA (default)	2 (default)
LEDG[3]	Output	PIN_H1	1	B1_N1	PIN_AB20	2.5 V (default)		8mA (default)	2 (default)
LEDG[2]	Output	PIN_J3	1	B1_N1	PIN_AB16	2.5 V (default)		8mA (default)	2 (default)
LEDG[1]	Output	PIN_J2	1	B1_N1	PIN_U14	2.5 V (default)		8mA (default)	2 (default)
LEDG[0]	Output	PIN_J1	1	B1_N1	PIN_H12	2.5 V (default)		8mA (default)	2 (default)
SW[9]	Input	PIN_D2	1	B1_N0	PIN_A5	2.5 V (default)		8mA (default)	
SW[8]	Input	PIN_E4	1	B1_N0	PIN_AB14	2.5 V (default)		8mA (default)	
SW[7]	Input	PIN_E3	1	B1_N0	PIN_N21	2.5 V (default)		8mA (default)	
SW[6]	Input	PIN_H7	1	B1_N0	PIN_B14	2.5 V (default)		8mA (default)	
SW[5]	Input	PIN_J7	1	B1_N1	PIN_C10	2.5 V (default)		8mA (default)	
SW[4]	Input	PIN_G5	1	B1_N0	PIN_M6	2.5 V (default)		8mA (default)	
SW[3]	Input	PIN_G4	1	B1_N0	PIN_AA20	2.5 V (default)		8mA (default)	
SW[2]	Input	PIN_H6	1	B1_N0	PIN_AA14	2.5 V (default)		8mA (default)	
SW[1]	Input	PIN_H5	1	B1_N0	PIN_R15	2.5 V (default)		8mA (default)	
SW[0]	Input	PIN_J6	1	B1_N0	PIN_C15	2.5 V (default)		8mA (default)	

Figure 2.10: Pin layout in Quartus Pin Planner

2.2.8 Checking of this part of the Lab

After programming the FPGA, you should see that if you change the position of any of switches on the FPGA (SW0-SW9) it should illuminate or turn off the corresponding green LED. Please review the Verilog code so you can understand why this is accomplished. After this is working correctly, the lab instructor or TA will ask you the three questions in the questions section at the end of the lab. The answers to these questions are in the lab writeup, be prepared to answer them.

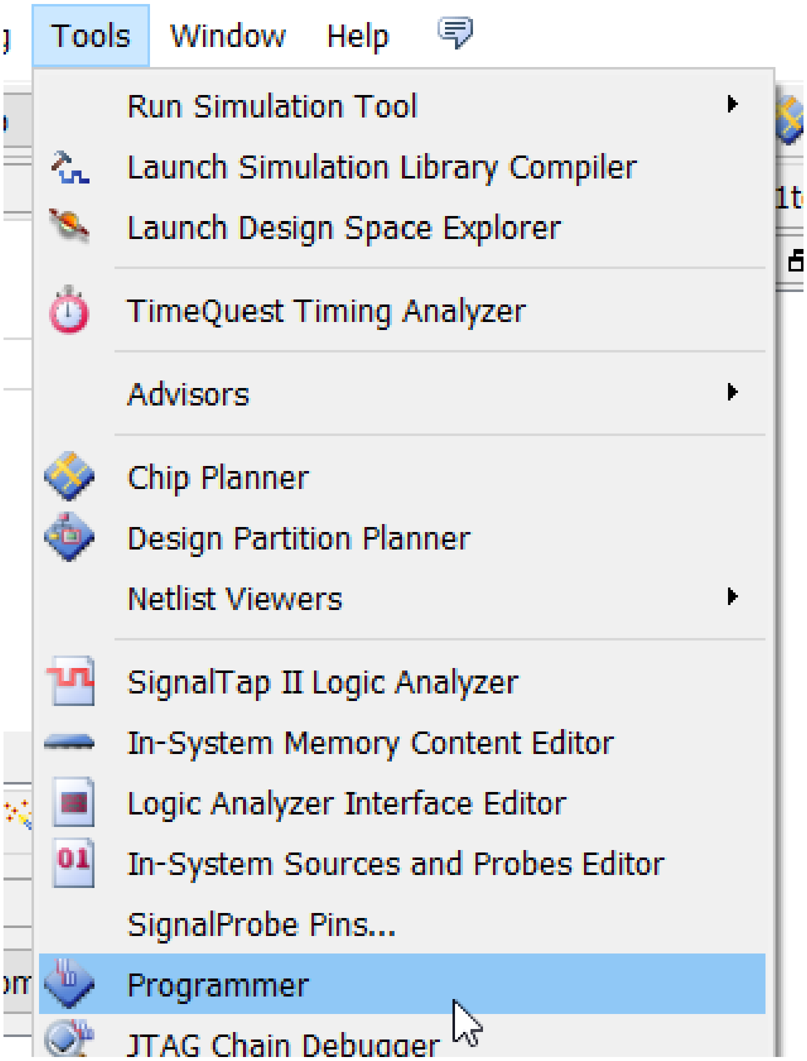
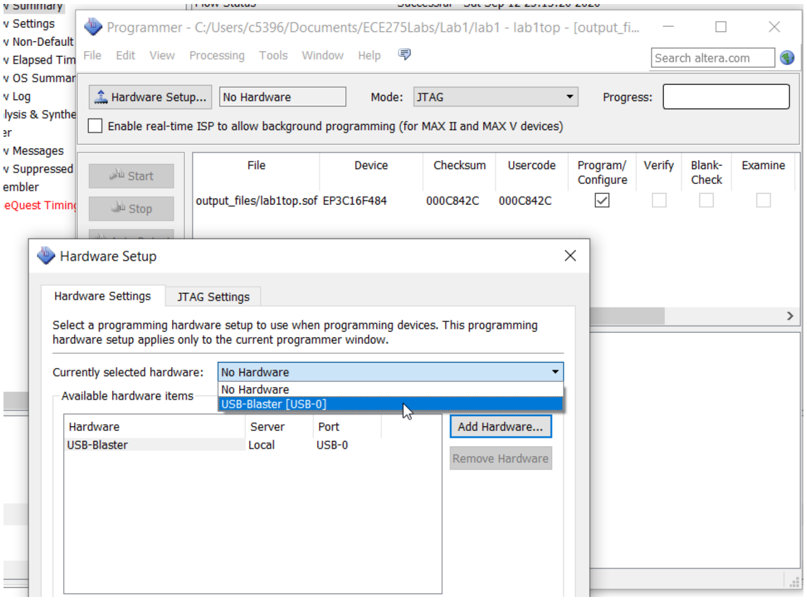

Tools
Window
Help
Run Simulation Tool
Launch Simulation Library Compiler
Launch Design Space Explorer
TimeQuest Timing Analyzer
Advisors
Chip Planner
Design Partition Planner
Netlist Viewers
SignalTap II Logic Analyzer
In-System Memory Content Editor
Logic Analyzer Interface Editor
In-System Sources and Probes Editor
SignalProbe Pins...
Programmer
JTAG Chain Debugger

Figure 2.11: Open the Quartus Programmer

2.3 Part 2: Analyzing a Verilog Program Utilizing the ModelSim Software

2.3.1 Create a Testbench

A testbench is essentially a Verilog module that lets you simulate a Verilog code without an actual digital circuit board. Normally you would create the testbench as a separate Verilog file, but in this case, feel free to keep it in the same file as your top-level for easy troubleshooting. Specifically, add the following verilog code right below endmodule to the same file `lab1.v`. Be careful about copying code from the pdf file. PDF will introduce special characters in your code which will cause errors. It is better to type in the code, or copy from the left half of the overleaf interface. If you are interested in learning more about test benches outside this lab or are trying to troubleshoot this lab, you can read more about Modelsim test benches in this Intel document: `https://community.intel.com/t5/Intel-Quartus-Prime-Software/ModelSim-testbench-tutorial/m-p/1295002/thread-id/69901` and ModelSim tutorial `https://www.microsemi.com/document-portal/doc_view/131618-modelsim-tutorial`



Programmer - C:/Users/c5396/Documents/ECE275Labs/Lab1/lab1 - lab1top - [output_fi...

No Hardware Mode: JTAG Progress:

□ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
output_files/lab1top.sof	EP3C16F484	000C842C	000C842C	☑	□	□	□

Hardware Setup

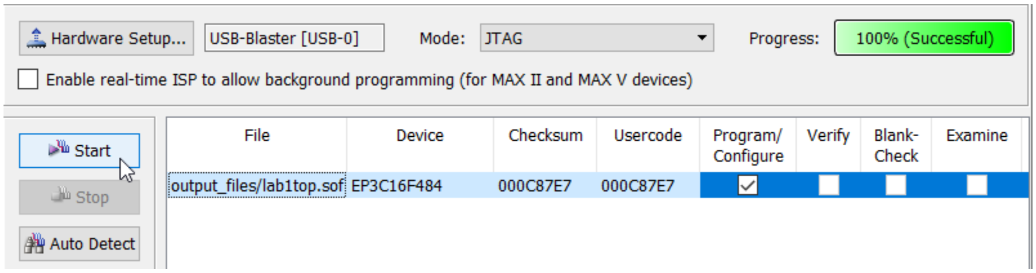
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No Hardware
No Hardware
USB-Blaster [USB-0]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-0

Figure 2.12: Select the USB-Blaster



USB-Blaster [USB-0] Mode: JTAG Progress: 100% (Successful)

□ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
output_files/lab1top.sof	EP3C16F484	000C87E7	000C87E7	☑	□	□	□

Figure 2.13: Program the FPGA

```
// Timescale sets the duration of one clock with precision
// Here 1ns is the duration of a single timestep and 1ps is the precision
`timescale 1ns / 1ps
module testbench ( );
    // Intialize a clock variable to keep track of time
    reg Clk;
    // Intialize a variable for switches using 10 bit binary notation.
    reg [9:0] SW;
    // Create wire for output LEDG
    wire [9:0] LEDG;

    // Connect the switch and LED to simulated module lab1top
    lab1top lt1 (SW, LEDG);

    // Intial block is executed when the circuit starts
    initial begin
        Clk <= 0; SW <= 0;
        // After a delay of 5 timesteps set SW to the following value using 10 bit binary
        #5 SW <= 10'b00_0001_0001;
    // After another delay of 5 timesteps set SW to the following value 10 bit binary notati
    #5 SW <= 10'b01_0011_0001;
  end

  // Always block is like an infinite while loop
  always begin
   // Every one 1 timestep invert the clock signal
   #1 Clk <= ~Clk;
  end
endmodule
```

The first line,”‘timescale 1ns / 1ps” sets the time deltas for changing input values at 1ns, and simulation precision at 1ps. Setting a lower simulation precision can give higher accuracy but increase simulation time. The inputs for simulation are then defined as registers, and outputs as wires. Next, the modules to be simulated are instantiated. In this case, that is the ‘lab1top‘ module. The next section contains the initial, begin, and end lines. The code inside sets the initial values for the inputs and then the times for them to be modified based on the timescale value. In the code shown in the example, the Clk, SW inputs are all 0. 5ns later, the SW bits are modified to a 0000010001. 5ns after that, the SW bits are modified to a new value 0100110001. This means that when the simulation is run, the LEDG output should show the same value as SW bits.

2.3.2 Set Path to ModelSim

To utilize your testbench, you will need to tell Quartus what simulator to use and its location. In Quartus, go to the tools menu, and then towards the bottom, select options. In the window that appears in the top left under general, select the option for EDA Tool Options. Here you will find ModelSim-Altera towards the bottom. Click the three-dot icon to the right of the ModelSim-Altera, and then browse to your ModelSim installation directory. You will need to go deeper into the directory for the path it is looking for. In my case, that was `C:\altera\13.1\modelsim_ase\win32aloem\`. You would have to modify this path if you selected a different installation directory for your ModelSim installation.

Important Note

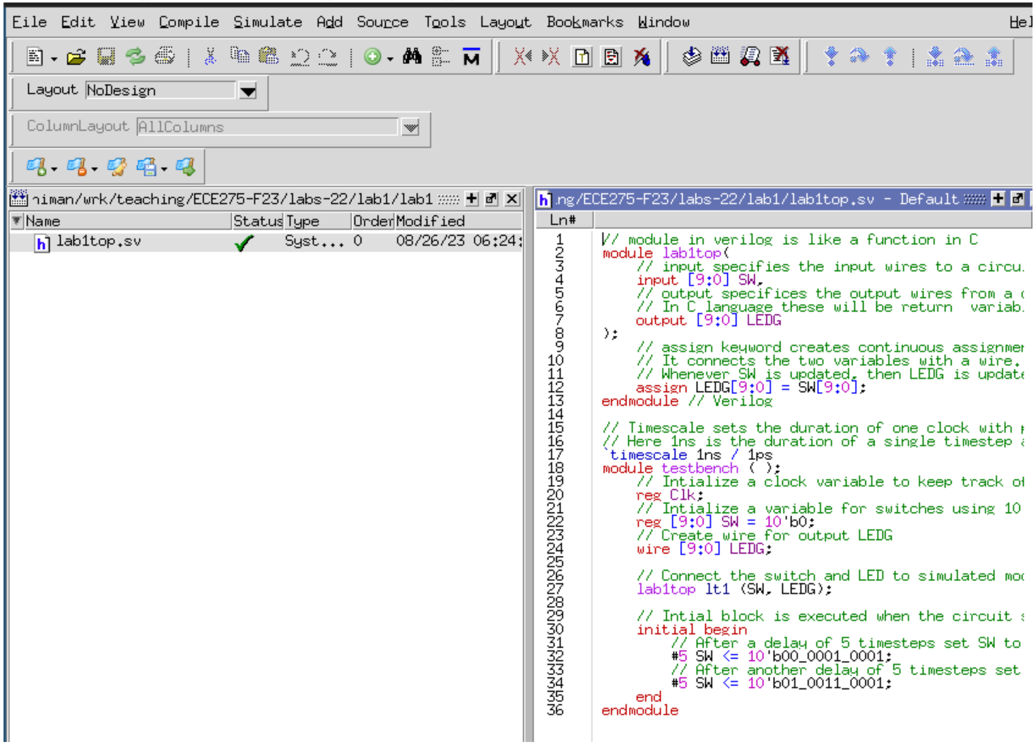
If you miss the backslash at the end of win32aloem, you will get errors and Quartus will fail to launch ModelSim-Altera.

2.3.3 Run Simulation

After you have selected your Modelsim installation, you can move on to the actual simulation. In Quartus, go into the tools menu, go to the Run Simulation Tool section, and choose the RTL Simulation option. If you get an error, you likely have not installed ModelSim or set your path correctly to your ModelSim installation.

2.3.4 Interact with the Simulation

The last step should have opened a separate window, ModelSim Altera.


File Edit View Compile Simulate Add Source Tools Layout Bookmarks Window Hel
Layout NoDesign
ColumnLayout AllColumns
ıiman/wrk/teaching/ECE275-F23/labs-22/lab1/lab1
.ng/ECE275-F23/labs-22/lab1/lab1top.sv - Default
Ln#

Name	Status	Type	Order	Modified
lab1top.sv	✓	Syst...	0	08/26/23 06:24:

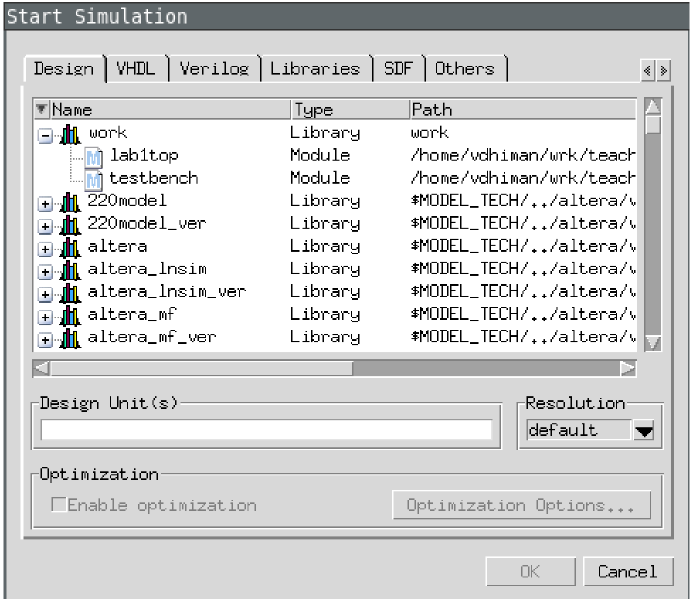
```
// module in verilog is like a function in C
module lab1top(
    // input specifies the input wires to a circu.
    input [9:0] SW,
    // output specifices the output wires from a (
    // In C language these will be return  variab.
    output [9:0] LEDG
);
    // assign keyword creates continuous assignmer
    // It connects the two variables with a wire.
    // Whenever SW is updated, then LEDG is update
    assign LEDG[9:0] = SW[9:0];
endmodule // Verilog

// Timescale sets the duration of one clock with
// Here 1ns is the duration of a single timestep
`timescale 1ns / 1ps
module testbench ( );
    // Intialize a clock variable to keep track of
    reg Clk;
    // Intialize a variable for switches using 10
    reg [9:0] SW = 10'b0;
    // Create wire for output LEDG
    wire [9:0] LEDG;

    // Connect the switch and LED to simulated mod
    lab1top lt1 (SW, LEDG);

    // Intial block is executed when the circuit
    initial begin
        // After a delay of 5 timesteps set SW to
        #5 SW <= 10'b00_0001_0001;
        // After another delay of 5 timesteps set
        #5 SW <= 10'b01_0011_0001;
    end
endmodule
```

By default, none of the waves you want to view will be selected, and your testbench will likely not be selected. To select your testbench, you must expand the work option under the libraries on the left.



Start Simulation

Name	Type	Path
work	Library	work
lab1top	Module	/home/vdhiman/wrk/teach
testbench	Module	/home/vdhiman/wrk/teach
220model	Library	$MODEL_TECH/../altera/v
220model_ver	Library	$MODEL_TECH/../altera/v
altera	Library	$MODEL_TECH/../altera/v
altera_lnsim	Library	$MODEL_TECH/../altera/v
altera_lnsim_ver	Library	$MODEL_TECH/../altera/v
altera_mf	Library	$MODEL_TECH/../altera/v
altera_mf_ver	Library	$MODEL_TECH/../altera/v

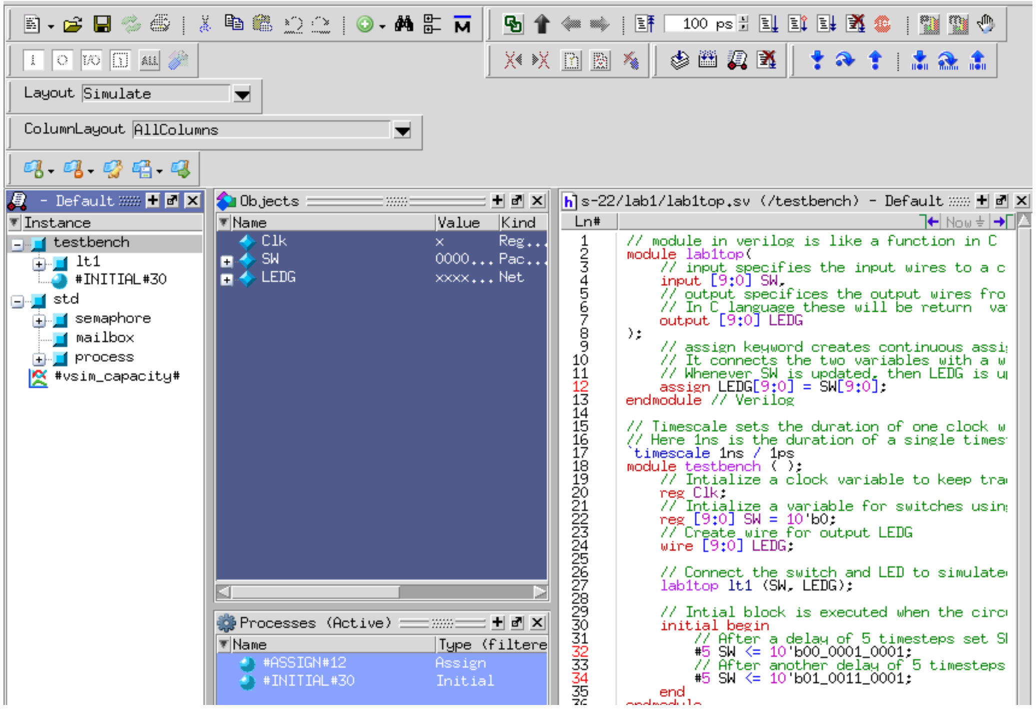
Design Unit(s)

Resolution: default

Optimization

□ Enable optimization

As long as you have run compilation on your top level, the testbench should show under the work section. Double-click on it to set it as active. Now under "Objects" , you should see the inputs and outputs from the testbench. Click on any one of Clk, SW and LEDG. Use Ctrl+A to select all, right click, and choose the option to "Add Wave".



Name	Value	Kind
Clk	x	Reg...
SW	0000...	Pac...
LEDG	xxxx...	Net

Processes (Active)

Name	Type (filtere
#ASSIGN#12	Assign
#INITIAL#30	Initial

s-22/lab1/lab1top.sv (/testbench) - Default

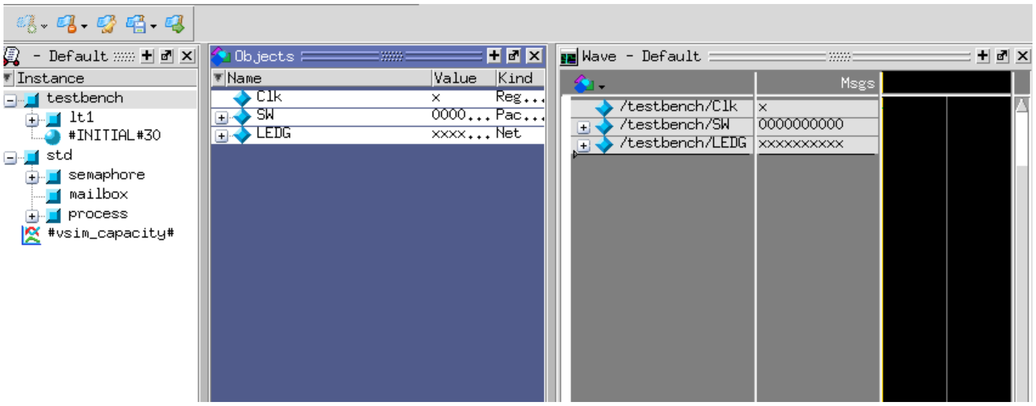
```
// module in verilog is like a function in C
module lab1top(
    // input specifies the input wires to a c
    input [9:0] SW,
    // output specifies the output wires fro
    // In C language these will be return  va
    output [9:0] LEDG
);
    // assign keyword creates continuous assi
    // It connects the two variables with a w
    // Whenever SW is updated, then LEDG is u
    assign LEDG[9:0] = SW[9:0];
endmodule // Verilog

// Timescale sets the duration of one clock w
// Here 1ns is the duration of a single times
`timescale 1ns / 1ps
module testbench ( );
    // Intialize a clock variable to keep tra
    reg Clk;
    // Intialize a variable for switches usin
    reg [9:0] SW = 10'b0;
    // Create wire for output LEDG
    wire [9:0] LEDG;

    // Connect the switch and LED to simulate
    lab1top lt1 (SW, LEDG);

    // Intial block is executed when the circ
    initial begin
        // After a delay of 5 timesteps set S
        #5 SW <= 10'b00_0001_0001;
        // After another delay of 5 timesteps
        #5 SW <= 10'b01_0011_0001;
    end
endmodule
```

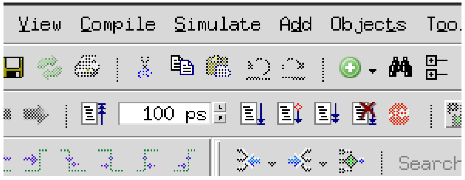
You should now see a wave graph on the right, but it will not contain data until you actually run the simulation.



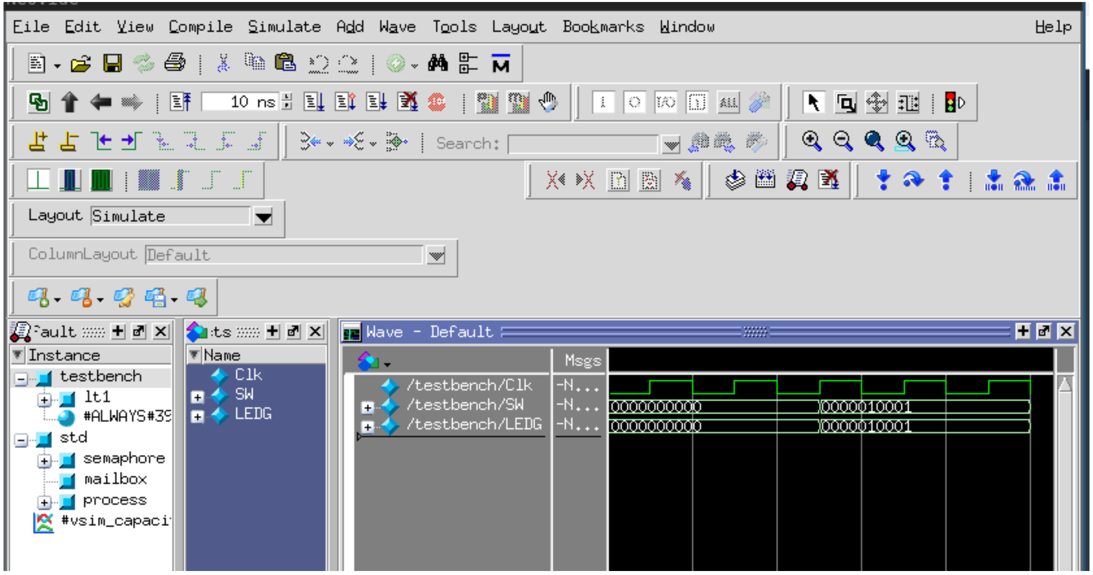
Name	Value	Kind
Clk	x	Reg...
SW	0000...	Pac...
LEDG	xxxx...	Net

```
/testbench/Clk    x
/testbench/SW     0000000000
/testbench/LEDG   xxxxxxxxxx
```

The option to run-time steps of the simulation is at the top of the ModelSim program, next to a box that says “100 ps”. This is the time step value the ModelSim simulation will simulate each time you hit the run option (”Simulate→Run→Run 100”)



This is significantly shorter than the amount of time before the first input will change state (100ps vs. 10ns). Increase this value to “100 ns” and use the run button next to the time (or ”Simulate→Run→Run 100”). To see all the data easily, right-click on the graph and choose the zoom full option. With this, you should see the full waveforms that show al the waveforms as shown in image below. Use the ”Simulate→Restart..” to restart the simulation from 0ns.



```
/testbench/Clk    -N...
/testbench/SW     -N...   0000000000   0000010001
/testbench/LEDG   -N...   0000000000   0000010001
```

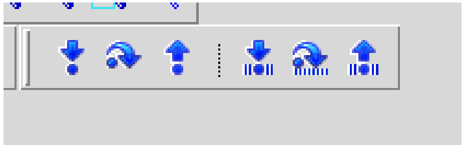
2.4 Questions

2.4.1 Part 2: Q1: Modifiying Clk and SW values

Modify your testbench file to use different time values for the Clk, SW. Show your TA how this affects your waveform output in the simulation, and explain to them the changes you made.

2.4.2 Part 2: Q2: Debugging

Recall from your C-language class what is a debugger and a breakpoint? Use ModelSim Tutorial to find a way to add breakpoint to stop at the line 12 of the module lab1top. Use ”Simulate→Run→Continue” to keep stopping at the breakpoint, every time SW changes. Look at the value of LEDG in the Objects pane. What do you observe? What does each button on the ”Step toolbar” (shown below) do?



2.4.3 Part 1: Question 1

When do you have to run or rerun compilation for a Quartus project?

2.4.4 Part 1: Question 2

After copying commands from a .qsf file to assign FPGA pins to the correct real world devices, where would you navigate in Quartus to check if the pins were assigned properly?

2.4.5 Part 1: Question 3

What position should the RUN/PROG switch be in on the DE0 boards when programming with JTAG?

2.4.6 Part 1: Question 4

What hardware description language will we be using for labs in this course?

2.4.7 Part 1: Question 5

Why should you create separate folders for each Quartus project, instead of saving all of the projects in the same folder?

2.5 Contributing to this document

Please submit errors, corrections, edits and improvements to this document to shihab.ahamad@maine.edu

2.6 Acknowledgement to the contributors

Thanks to all those contributed to this document.

1. Calvin Benider (ECE275-F2022) added tips for M2 Mac laptops.
2. Madi Martz (ECE275-F2023) helped with the `win32aloem` gotcha.

Contents

2.7 Lab Overview

The goal of this lab will be to learn the basics of the FPGA boards, and of SystemVerilog by creating multiplexers.

You will need to:

1. Create a multiplexer using the GUI schematic functionality in Quartus
2. Create a multiplexer with SystemVerilog only using the = ! ∼ | & operators
3. Compare the gates synthesized from your SystemVerilog code by Quartus to the schematic design
4. Extend the multiplexer to accept a vector of inputs

2.8 Part 1: Graphical Section

2.8.1 Create a New Project and Schematic

Create a new Quartus project in the same method as the previous lab, with a name specific to this section of the lab. Something along the lines of lab2part1. Make sure to select the correct FPGA version. Refer to the Lab 1 document for the steps if you do not remember. In this project DO NOT create a SystemVerilog file. We will instead be creating a Block Diagram/Schematic File as shown in Figure 2.14.

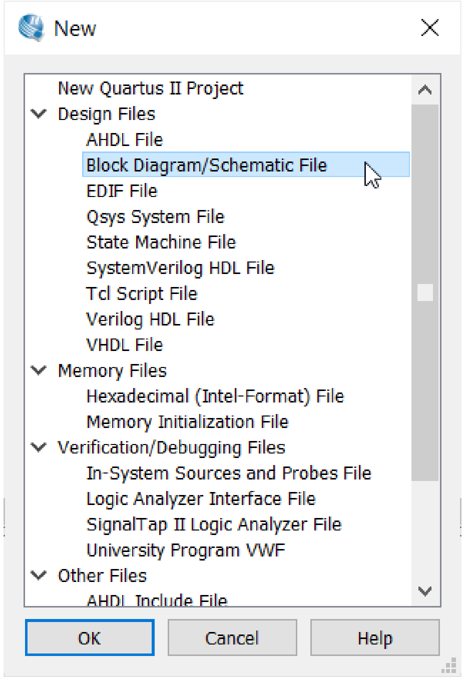


Figure 2.14: Create a new Block Diagram/Schematic File

2.8.2 Multiplexer Overview

A multiplexer is in essence a digital selector. One of the inputs allows you to select which of the other inputs would be moved to the output. A simple multiplexer (as we are building in this section) would have 3 input bits. In our case we will label them as s, x, and y. s would be our selector bit, where if it is a 0, then the value of x is present on the output (m=x). If it is a 1, then the value of y is present on the output (m=y). The symbol for a multiplexer is shown in Figure 2.15 and represents the multiplexer you will be building today, with an output m.

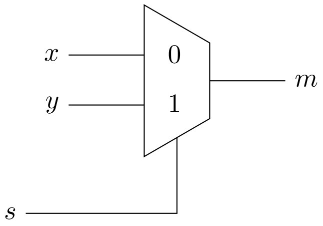


Figure 2.15: Simple Multiplexer

This can also be represented by the digital logic circuit shown in Figure 2.16

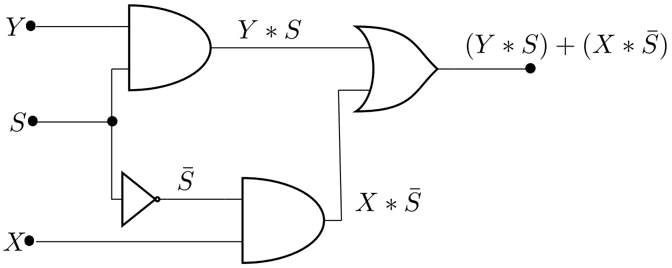


Figure 2.16: Simple Multiplexer Logic Diagram

2.8.3 Create the Multiplexer

The schematic file you created in the previous step should be open. Go ahead and save it right away. You will need to save it as name of your top level, so something like lab2part1top.bdf. .bdf files are the Block Diagram/Schematic files, while .v represents SystemVerilog files.

Next, use the pin tool at the top of the schematic (shown in Figure 2.17) to create 3 input pins and one output pin. You will need to use the arrow to the right of the box to change the selection to output to create the output pin. These represent pins x, y, s, and m for the multiplexer. Generally it is a good idea to create your inputs on the left and outputs on the right so logic flows intuitively from left to right.

Next, create the logic elements you will need to build the multiplexer from Figure 2.16, such as the 2 AND gates, 1 OR gate, and 1 NOT. These symbols are under the symbol tool to the left of the pin tool. Make sure to select the logic folder in the symbol tool to find these as shown in Figure 2.18.

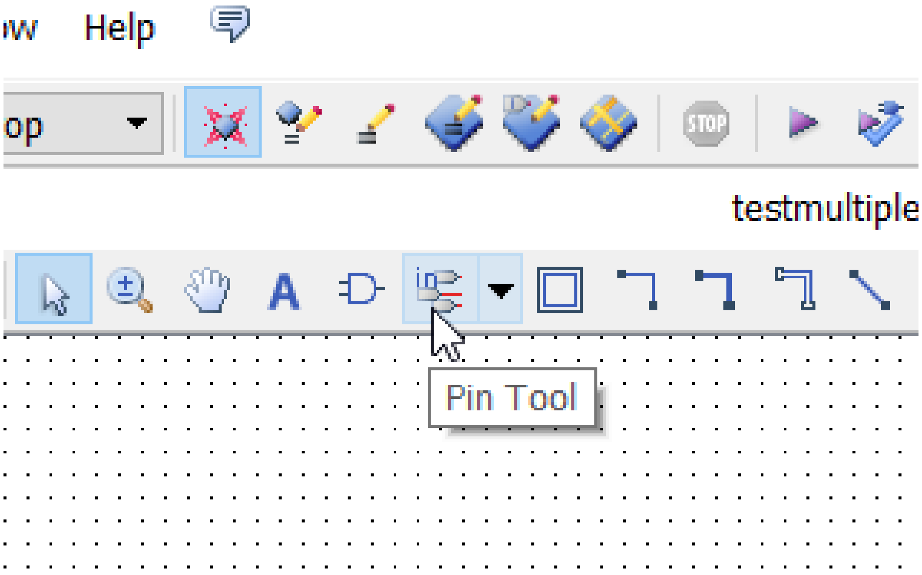


Figure 2.17: Quartus Schematic Pin Tool

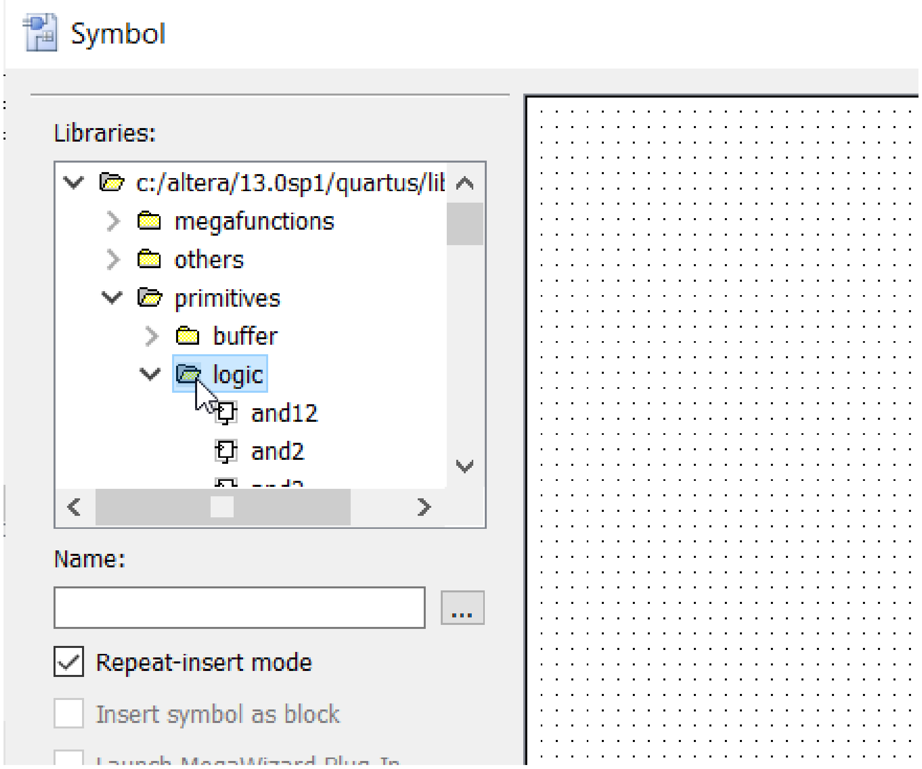


Figure 2.18: Select the Logic Elements from the Symbol Library

Next, attach wires to complete the logic. To start a wire you just need to click and drag from any of the pins on any of the inputs/output/gates.

2.8.4 Label Pins and Set Pin Assignments

You should now have a graphical logic diagram that represents a multiplexer. The last step is to label the pins and then assign them to real-world devices (switches for the inputs, and LED for the output). The best idea for the pin labeling is to use the pin names from the .qsf files, so you can copy and paste pin assignments from there. Open the qsf to verify these names, but the switches should be labeled as SW[0], SW[1], etc. and the green LEDs would be LEDG[0], LEDG[1], etc. For this lab use SW[0] as s, SW[1] as x, and SW[2] as y. LEDG[0] will represent the output m. Make sure you understand this portion, as these pin addresses will not be explicitly given to you in the future. You can find all the addresses of the default pins in the Altera DE0 User Manual from Lab1. If you do not understand these names and how they relate to the .qsf files, ask the instructor or a TA.

After you label the pins select the relevant lines from the .qsf file and paste them into the TCL console in Quartus to assign the pins.

2.8.5 Run Compilation and Program the FPGA

At this point you should be able to run compilation and then program to the board as you did in Lab 1. If you run into errors during compilation please try to read through the error messages and diagnose the error yourself before asking a TA.

Create a truth table of x, y, s, and m from manipulating the switches and have it checked off by a TA to complete this section of the lab. Take clear screenshots of the schematic diagram you drew.

2.9 Part 2: SystemVerilog

2.9.1 Basics of SystemVerilog

In Lab 1, we used Verilog. From this Lab onwards, we will use SystemVerilog, which is, loosely speaking, a more modern version of Verilog.

From this lab onwards, when creating a new file, you will choose SystemVerilog HDL (Hardware Description Language) instead of Verilog HDL (See Fig 2.19).

The first section was meant to help you see the connection between what you program in SystemVerilog, and the digital logic representation. You will now be creating the same multiplexer you created graphically, but now by utilizing SystemVerilog. Below is the code from last week's lab. We will take a closer look at it today, so you can modify it to create your multiplexer.

```
module lab1top(
    input [9:0] SW,
    output [9:0] LEDG
);
    assign LEDG[9:0] = SW[9:0];
endmodule
```

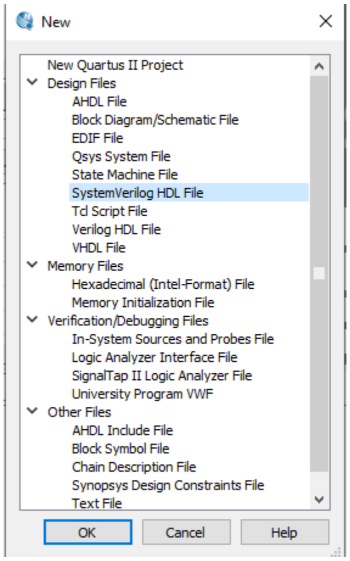


Figure 2.19: Creating a new SystemVerilog HDL file

The first line ”module lab1top(” is similar, but not identical to a C function. You will learn more about the differences later, but for now, just see it as the module that has your top-level name will be the module that runs on your FPGA. My top level was lab1top, so make sure you modify that to match your top level.

The next part inside the parenthesis, ”input [9:0] SW, output [9:0] LEDG); signifies the variables to be used in the module. You can just name the variables in the parenthesis without giving a size or type, as long as you specify those in the module. With this being a simple module, I just designated the switches as inputs, and the LEDs as outputs right in the parenthesis. The [9:0] part of the variable declaration indicates to create the variables SW[0], SW[1], SW[2],..., SW[9]. These can be utilized similarly to C arrays where you can assign to a range SW[5:3] or to a single value SW[3]. You can probably guess that the input keyword signifies those are either inputs from other digital logic, or real-world input devices such as switches. Outputs signify being output to other digital logic or tied to real-world outputs such as LEDs. Other options exist, such as wire or register, that would represent a wire between logic, or a register to hold a value.

The next part ”assign LEDG[9:0] = SW[9:0];” is taking the values of the switches and assigning them to the LEDs. In this case, LEDG[0] is controlled by SW[0], LEDG[1] is tied to SW[1], and so on. The assign statement creates connections between variables similar to how you drew the wires in part 1. The example only utilizes simple equals, but SystemVerilog supports many math operators such as +,-,*,/, etc. We will be utilizing the more basic boolean logic operators & (AND), | (OR), ! (NOT). Use parenthesis to specify the order of operations.

Take clear screenshots of the code you wrote to include in the report.

2.9.2 Writing Your Own SystemVerilog

Start the section off by creating a new project, with a blank SystemVerilog file. You will be modifying the SystemVerilog code in the previous section to create the multiplexer from Figure 2.16. The first thing would be to change the inputs/outputs to match our case. The lab would still work if you left them as they are, but it is bad practice to create more variables than you need. We are only using SW[0], SW[1], SW[2], and LEDG[0].

Important Note

SystemVerilog will not let you create a variable as a single-bit vector. So while input [1:0] LEDG would create a two-bit wide vector for LEDG (LEDG[0] and LEDG[1]), input [0] LEDG would give you an error. You would instead just do input LEDG. The issue with this is the variable name would no longer match what is in the .qsf file for that pin assignment. This leaves two options. You can manually assign the LEDG variable in the pin planner, or you can modify the line from the .qsf file for LEDG[0] to just LEDG.

Now change the assign statement to match the logic you need for the multiplexer. You can see this at the far right of Figure 2.16. The equation in SystemVerilog would be: ”assign m = (! s & x) | (s & y);” Replace the assign statement in the example SystemVerilog from Lab 1 with this assign statement and then change the variable names (s,x,y,m) to match the proper input/output addresses. For example, s is SW[0]. After you replace the variables properly, run compilation and then program FPGA. Check that the results for this section match your truth table from the first part.

2.9.3 Compare the Synthesized Logic Gates

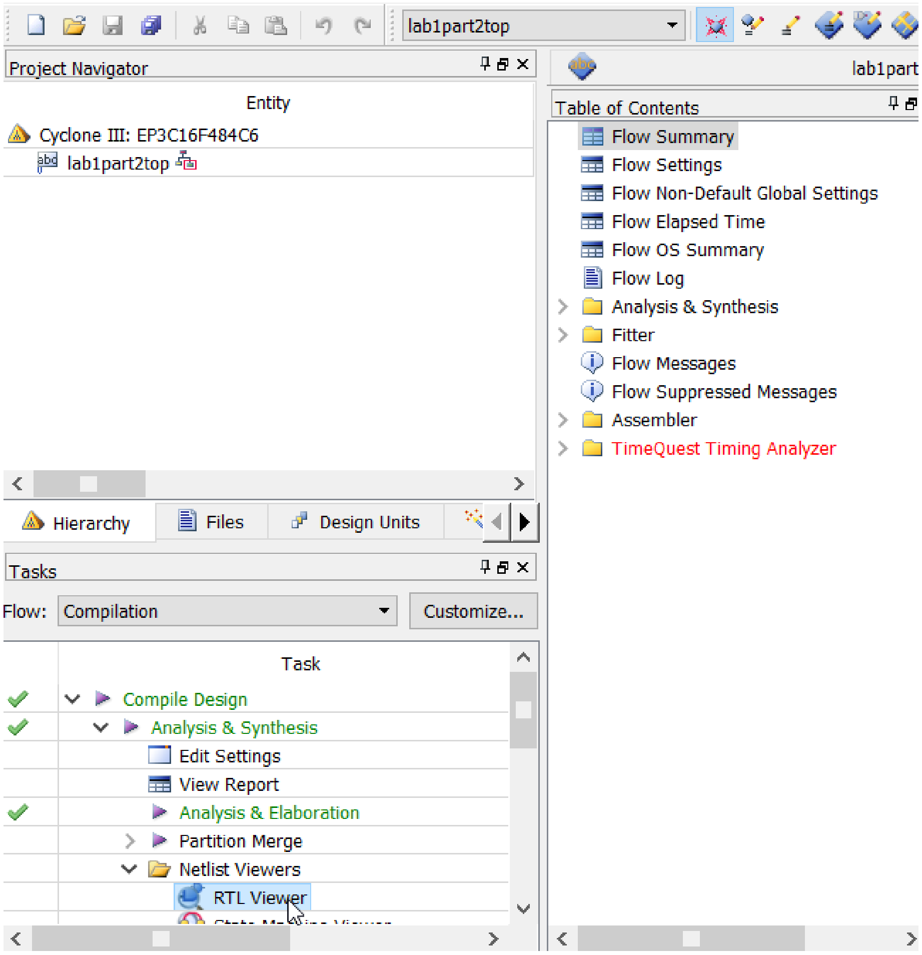
The last step for this section is to compare the synthesized logic gates from your SystemVerilog code to how you would design the multiplexer with logic gates. After you do the compilation, a compilation list should appear on the left. Navigate to the RTL Viewer Option and double-click. This is shown in Figure 2.20, and the created logic is shown in Figure 2.21

2.10 Part 3: Extend the multiplexer

For this section, create another new project and blank the SystemVerilog file. It would be a good idea to copy and paste your code from the last section, as you can modify it to make this section easier.

In the previous part you created a simple multiplexer with a single bit switch, and 2 single bit inputs. For this portion you will need to create the inputs as 4 bit wide vectors, as well as the output being 4 bits wide. This will show you how to manipulate vectors in SystemVerilog, as it is the most useful basic tool to reduce code complexity. You have already seen how to create input and output vectors from the code from the last section (SW[3:0] would be a 4 bit wide vector).

The easiest way to accomplish this section is to utilize multiple assign statements and assign each output LED using a similar equation to the assigned line from the previous section. You will just need to update each assign statement to look at the correct location in the vector. So if in the last section the assign line was ”assign m = (!s & x) | (s & y);”, and you replaced s, x, and y with the corresponding SW[X], m with the corresponding LEDG[X] value, in this section if you create multiple lines of the form ”assign m = (!s & x) | (s & y);” you can update each line with different SW and LEDG values. With this method, you would need 4 assign lines.



Flow: Compilation

Figure 2.20: Select the RTL Viewer in the Compilation Window

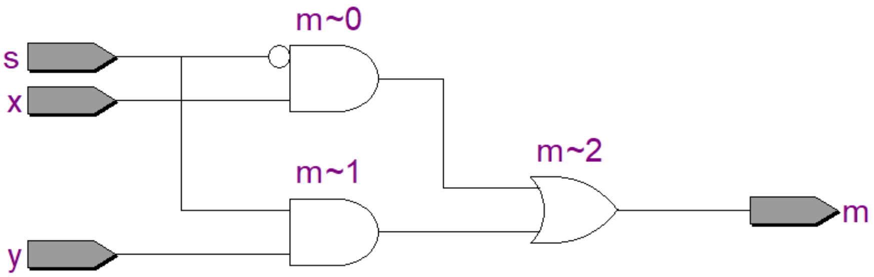


Figure 2.21: Digital Logic Created by the SystemVerilog Code

There are other ways to accomplish the same result to complete this lab, and you are welcome to try a different method if you would like. You just must not use a prebuilt multiplexer module, you must use SystemVerilog code, and in your SystemVerilog, you cannot use if statements, or math outside of & (AND), | (OR),! (logical NOT)/∼ (bitwise NOT). Most other methods would also likely require you to create a temporary storage variable. You can do this by adding ”wire [1:0]

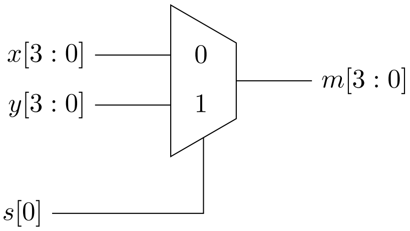


Figure 2.22: Extended Multiplexer

VARNAME" to the module variable declaration section. You would replace the [1:0] with the size of the vector you need. You can then use it as a storage variable, but it acts in a similar manner to a wire in-between logic gates.

2.11 Questions

Answer these questions for a TA at the end of class to complete the lab.

2.11.1 Question 1

Why is it helpful to use SW[0], SW[1], etc as the variable names instead of using names like s, x, y?

2.11.2 Question 2

What would the SystemVerilog line "assign LEDG[7:4]=SW[3:0]" accomplish?

2.11.3 Question 3

In the following code section, which assign line will be executed first when running on the FPGA?

```
module lab1top(
    input [9:0] SW,
    output [9:0] LEDG
);
    assign LEDG[0] = SW[9];
    assign LEDG[1] = SW[8];
    assign LEDG[2] = SW[7];
endmodule
```

2.11.4 Question 4

What advantages can you envision for utilizing SystemVerilog vs the schematic method for designing digital logic for the FPGA?

2.11.5 Question 5

Use the IEEE Standard (See section 11.4.7 and 11.4.8) as a reference for this question: `https://vikasdhiman.info/ECE275-F23-Sequential-Logic/lab_pdfs/1800-2017.pdf`

What would be the difference between the following SystemVerilog assign statements:

assign LEDG[9:0]=!SW[9:0];
assign LEDG[9:0]=∼SW[9:0];

2.12 Report Writing Guidelines

For each part of Lab2, organize your content as follows:

1. **Part Title**: Clearly mention the title or main objective of the part.
2. **Description**: Begin with a brief 1-2 line description explaining the purpose or main task of that part.
3. **Truth Table**: If applicable for the part, present the truth table. Ensure it is clearly labeled and easy to read.
4. **Screenshots/Picture**:
   - **Schematic Diagram**: Include clear screenshots of the schematic diagram you drew for that part1.
   - **Code**: Include clear screenshots of the code you wrote for that part2 and part3.
   - **FPGA Kit Output**: Include pictures showing the output of the FPGA kit for each part.
   - **RTL View**: Attach clear screenshots of the RTL view related to the part2 and part3.
5. **Description for Screenshots**: Below each screenshot or picture, add a brief description or note to explain what it represents.
6. And ofcourse answer all the questions mentioned in the document

2.13 NOTE

After completing each part make sure to take necessary screenshots/pictures and get checked-off with TA.

2.14 Logic minimization

2.15 Logic minimization

A general optimization criteria for multi-level logic are to Minimize some combination of:

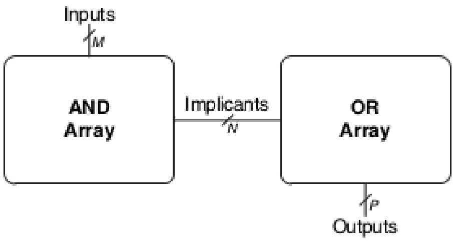
1. Area occupied by the logic gates and interconnect;
2. the Critical Path Delay of the longest path through the logic;

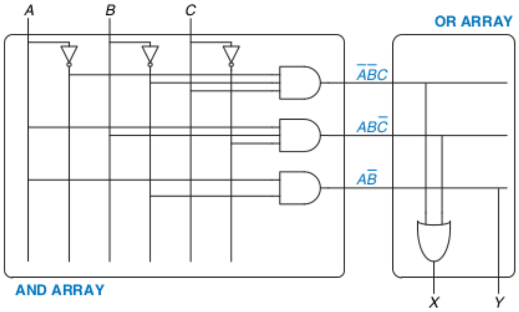
3. the Degree of Testability of the circuit, measured in terms of the percentage of faults covered by a specified set of test vectors, for an appropriate fault model (Eg., single stuck faults, multiple stuck faults, etc.);

4. Power consumed by the logic gates.

In this course, we will start with two-level multi-input circuits and a criteria based on the number of gates/transistors/diodes.

2.16 Programmable Logic Arrays





2.17 Two-level circuits

The cost that we are going to consider in this class depend upon:

1. Number of gates.

2. Number of input to the gates.

More gates need more transistors, more area on the chip. More-inputs the gate need more transistors within each gate. Number of gate inputs can be considered secondary criterion to the number of gates.

Example 15. *Find the cost of the following Boolean expression* $X = \bar{A}\bar{B}C + AB\bar{C} + A\bar{B}$.

Problem 10. *Find the cost of the following Boolean expression* $X = A\bar{B}C + \bar{A}B\bar{C} + \bar{B}C$.

2.18 Terminology for K-maps

Running Example: $f = \sum m(0, 1, 2, 3, 7) = \bar{x}_1 + x_1x_2x_3$.

Literal A single variable or its complement. Example: $\bar{x}, x_1, x_2, x_3$

Implicant A product term which is true for a function. All minterms are implicants. Example: $x_1x_2x_3$, $\bar{x}_1$, $m_0 = \bar{x}_1\bar{x}_2\bar{x}_3$, $\bar{x}_1x_3$, $\bar{x}_1\bar{x}_3$.

Prime Implicant An implicant that cannot be combined into fewer literals. Example: $\bar{x}_1, x_2x_3$.

Essential Prime Implicant An implicant that cannot be combined into fewer literals. Example: x_2x_3.

Cover : List of Prime Implicants that account for all $f = 1$.

Cost : Number of gates (excluding not gate on literals) and number of inputs to each gate.

Example 16. *Find minimum cost expression for the function* $f(x_1, x_2, x_3) = \prod M(4, 5, 6)$

Problem 11. *Find minimum cost expression for the function* $f(x_1, x_2, x_3) = \prod M(2, 5, 6)$

2.18.1 Incompletely specified functions or Don't cares

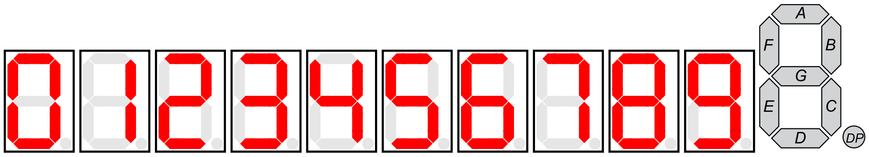


Figure 2.23: 7 Segment Representations of Each Integer

BCD Value				LED Segment
D_3	D_2	D_1	D_0	E
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	d
1	0	1	1	d
1	1	0	0	d
1	1	0	1	d
1	1	1	0	d
1	1	1	1	d

Example 17. *Find minimum cost expression for the function*

$$f(x_1, \ldots, x_4) = \sum m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)$$

Problem 12. *Find minimum cost expression for the function*

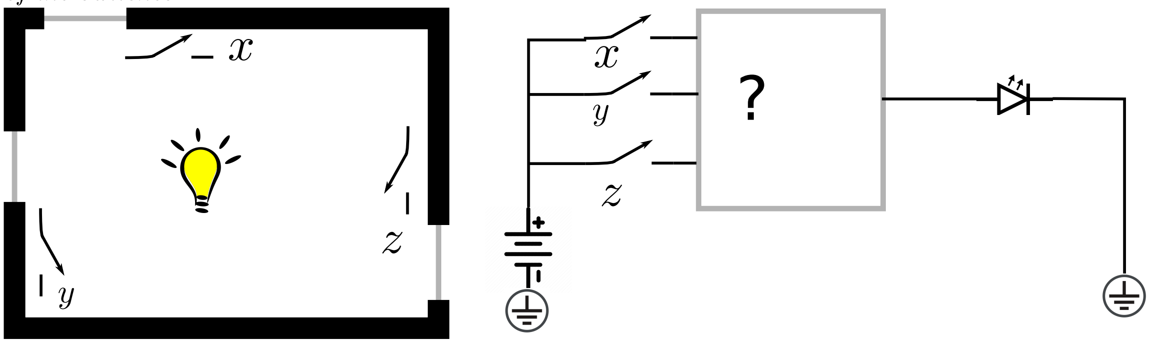
$$f(x_1, \ldots, x_4) = \sum m(0, 2, 4, 6, 7, 8, 9, 13) + D(1, 12, 15)$$

2.19 A few more Boolean problems

Example 18. *Simplify the following Boolean expression:*

$$f = x_1\bar{x}_3\bar{x}_4 + x_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3$$

Example 19. *Assume that a large room has three doors and that a switch near each door controls a light in the room. It has to be possible to turn the light on or off by changing the state of any one of the switches.*



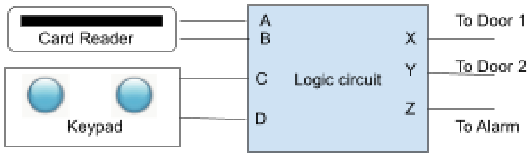
Problem 13. *A simple security system for two doors consists of a card reader and a keypad.*

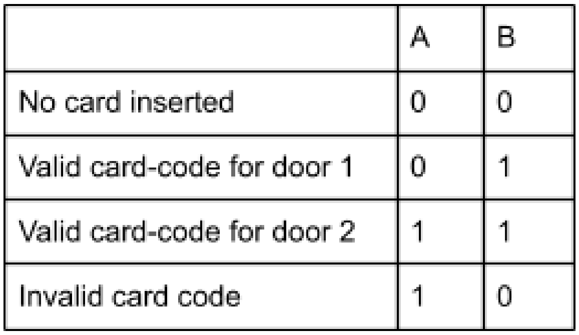
A person may open a particular door if he or she has a card containing the corresponding code and enters an authorized keypad code for that card. Note that card-code and keypad-code are different. The outputs from the card reader are given in the table below.

To unlock a door, a person must hold down the proper keys on the keypad and, then, insert the card in the reader. The authorized keypad code for door 1 is 10, and the authorized keypad code for *door 2 is 11. If the card has an invalid code or if the wrong keypad code is entered, the alarm will ring when the card is inserted. If the correct keypad code is entered, the corresponding door will be unlocked when the card is inserted.*

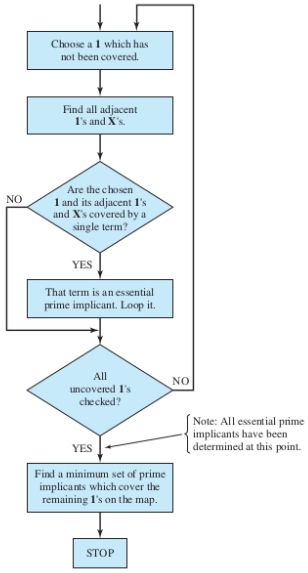
Design the logic circuit for this simple security system. Your circuit's inputs will consist of a card code AB, and a keypad code CD. The circuit will have three outputs XYZ (if X is 1, door 1 will be opened; if Y is 1, door 2 will be opened; if Z 1, the alarm will sound).

Find the minimal cost two-level circuit using K-maps for X, Y, Z. Provide the minimal cost. (It can be either of SOP/POS forms)





	A	B
No card inserted	0	0
Valid card-code for door 1	0	1
Valid card-code for door 2	1	1
Invalid card code	1	0


Choose a 1 which has not been covered.
Find all adjacent 1's and X's.
Are the chosen 1 and its adjacent 1's and X's covered by a single term?
NO
YES
That term is an essential prime implicant. Loop it.
All uncovered 1's checked?
NO
YES
Note: All essential prime implicants have been determined at this point.
Find a minimum set of prime implicants which cover the remaining 1's on the map.
STOP

Example 20.

$CD \backslash AB$	*00*	*01*	*11*	*10*
00	*d*	*1*		*1*
01		*1*	*1*	*1*
11		*d1*	*d*	
10		*1*		*1*

Row	x_1	x_2	x_3	f
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Table 2.1: Truth table for a 3-way light switch

Problem 14. *Find the minimum SOP (sum of products) and POS (product of sum) expression for the function* $f(a,b,c,d) = \prod M(5,7,13,14,15) \cdot \prod D(1,2,3,9)$

Problem 15. *Read Chapter 2 up to Section 2.7 of Harris and Harris textbook. Write a statement saying that you have read and understood the chapter. [5 marks]*

Problem 16. *If the Sum of Products (SOP) form for* $\bar{f} = AB\bar{C} + \bar{A}\bar{B}$*, then give the Product of Sums (POS) form for* f*. [10 marks]*

Problem 17. *Use DeMorgan's Theorem to find* f *if* $\bar{f} = (A + \bar{B}C)D + EF$*. [10 marks]*

Problem 18. *For the function* $f = AB\bar{C} + BD$*,*

1. *Write the Truth table. [10 marks]*
2. *Write* f *in Sum of Products form. [10 marks]*
3. *Write* f *in canonical minterm form. [10 marks]*
4. *Write* f *as Product of Sums. [10 marks]*
5. *Write* f *in canonical maxterm form. [10 marks]*

Problem 19. *Implement the function in Table 2.1 using only NAND gates. [10 marks]*

Problem 20. *Implement the function in Table 2.1 using only NOR gates. [10 marks]*

Problem 21. *Find the minimum-cost Sum of Products (SOP) and Product of Sums (POS) forms for the function* $f(x_1, x_2, x_3) = m(1,3,4,5)$*. Chose the minimum-cost expression by comparing Product of Sums (POS) and Sum of Products (SOP) forms. [10 marks]*

Problem 22. *Find the minimum-cost Sum of Products (SOP) and Product of Sums (POS) forms for the function* $f(x_1, x_2, x_3) = \sum m(1,5,7) + D(2,4)$*. [10 marks]*

Problem 23. *Find the minimum-cost Sum of Products (SOP) and Product of Sums (POS) forms for the function* $f(x_1, x_2, x_3, x_4) = \prod M(1,2,4,5,7,8,9,10,12,14,15)$*. Chose the minimum-cost expression by comparing Product of Sums (POS) and Sum of Products (SOP) forms. [10 marks]*

Problem 24. *Find the minimum-cost Sum of Products (SOP) and Product of Sums (POS) forms for the function* $f(x_1, x_2, x_3, x_4) = \sum m(2,8,9,12,15) + D(1,3,6,7)$*. Chose the minimum-cost expression by comparing Product of Sums (POS) and Sum of Products (SOP) forms. [10 marks]*

Problem 25. *Derive a minimum-cost realization of the four-variable function that is equal to 1 if exactly two or exactly three of its variables are equal to 1; otherwise it is equal to 0. [10 marks]*

Problem 26. *Find the minimum-cost Sum of Products (SOP) and Product of Sums (POS) forms for the function* $f(x_1, \ldots, x_5) = \sum m(1,3,4,6,8,9,11,13,14,16,19,20,21,22,24,25) + D(5,7,12,15,17,23)$*. Chose the minimum-cost expression by comparing Product of Sums (POS) and Sum of Products (SOP) forms. [10 marks]*

Chapter 3

Number System

3.1 Place value number system

- What is a place value number system?

- What are some examples?

- What are some non-examples?

- What is a radix (or base)?

- How to convert between different radix in place value system?

- What are some commonly used number systems in computer engineering?

3.1.1 Decimal number system

3.1.2 Binary numbers

3.1.3 Conversion between different radix

Problem 27. *Convert the following binary numbers to decimal:* $(11110)_2$*,* $(100111)_2$*.*

Conversion from decimal to binary The value is in decimal because we find it easy to do calculations in decimal numbers. Decimal values can be converted back to Binary representation by *repeated division* by 2 while noting down the remainder. Allow me to use / sign to denote both quotient and remainder after division. Let's convert $(22)_{10}$ back to binary:

$$\begin{aligned}
22/2 &= (11, 0) && \text{11 is the quotient and 0 is the remainder}\\
11/2 &= (5, 1) && \text{5 is the quotient and 1 is the remainder}\\
5/2 &= (2, 1)\\
2/2 &= (1, 0)\\
1/2 &= (0, 1)
\end{aligned}$$

Read the remainders from bottom to top and right them as left to right, to form the resultant binary number $(22)_{10} = (10110)_2$.

Problem 28. *Find the binary representation for decimal numbers:* 123 *and* 89. *Show your work.*

3.2 Hexadecimal numbers

Numbers with base 16 are called Hexadecimal numbers. From 0 to 9 the symbols are same as decimal numbers. From 10 to 15, Hexadecimal numbers use A to F.

$$A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$$

. Example, $(10AD)_{16} = 1 \times 16^3 + 10 \times 16^1 + 13 = 4096 + 160 + 13 = 4269$.

3.3 Octal numbers

Numbers with base 8 are called octal numbers. Example, $(354)_8 = 3 \times 8^2 + 5 \times 8 + 4 = 192 + 40 + 4 = 236$.

3.4 Hexadecimal/octal to binary and vice-versa

Normally, if you have to convert between a number of base r_1 to a number of base r_2, we will have to convert it via decimal numbers. Convert from base r_1 to decimal and then from decimal to r_2.

Since Hexadecimal base 16 is an exact power of 2 ($16 = 2^4$). Conversion between Hexadecimal to binary is easy. You can group 4 binary digits from right to left and convert each group of 4 binary digits to a single Hexadecimal digit and back. Example, $(10110)_2 = (0001\_0110)_2 = (16)_{16}$. To convert back. Take example, $(10AD)_{16} = (0001\_0000\_1010\_1101)_2 = (1\_0000\_1010\_1101)_2$.

Problem 29. *Find the binary and decimal values of the following Hexadecimal numbers* $(A25F)_{16}$*,* $(F0F0)_{16}$.

Similarly octal to binary can proceed by grouping 3-binary digits at a time. Example, $(354)_8 = (011\_101\_100)_2$.

Problem 30. *Find the binary and decimal values of the following Octal numbers* $(3751)_8$ *and* $(722)_8$.

3.5 Signed binary numbers

Signed numbers include both negative and positive numbers. There three common signed number representations

1. Sign magnitude representation
2. One's complement
3. Two's complement

3.5.1 Sign-magnitude representation

The Most significant (left most) *bit* (binary digit) represents sign ($0 = +$ and $1 = -$), the rest represent the magnitude. Example, a 5-bit number $(11010)_2$ in signed magnitude representation has the value of $(-1010)_2 = -10$. Note that $+10$ has to be represented by a leading 0 at the most significant bit (MSB) $+10 = (01010)_2$. Hence, the number of bits have to be specified.

Problem 31.

- *Write down all possible 4-digit binary numbers and corresponding decimal values if they are in signed magnitude format? What is the minimum and maximum value?*
- *What is the minimum and maximum value of n-digit signed binary number in sign-magnitude format?*

3.5.2 One's complement negation

You can convert a positive number (say +10) to negative number by applying a negative sign in front of it (-(+10) = -10). It is more evident from taking negative of a negative number (-(-10) = +10). In case of sign-magnitude representation, the "negative operator" flips the sign bit. The next two signed number representations (1's complement and 2's complement) are designed around specific negative operator definitions.

Negate $13_{10} = 01101_2$ using 5-bit one's complement.

Negate -13_{10} using 5-bit one's complement.

3.5.3 One's complement binary numbers

In one's complement representation, the negative operation is obtained by flipping all the bits of the binary number. Example, a 5-bit one's complement of $+10 = (01010)_2$ is $(10101)_2 = -10$. Note that flipping bits is equivalent to subtracting the number from $(11111)_2$, hence the name. You can also confirm that double negative operator yields back the same number.

Problem 32.

- *Write down all possible 4-digit binary numbers and corresponding decimal values if they are in sign magnitude format? What is the minimum and maximum value?*

- *What is the minimum and maximum value of n-digit signed binary number in one's complement?*

Problem 33. *Determine the decimal values of the following 1's complement 6-digit binary numbers :*

1. *01101110*

2. *10101101*

Problem 34. *Convert the decimal numbers -17 and +23 into the 6-digit one's complement binary numbers and try adding them. What adjustments will you need to make to get the right result's (23-17=6) in binary representation.*

3.5.4 Two's complement negation

In two's complement representation, the n-digit negative number is obtained by subtracting the positive number from 2^n. Example, two's complement of 5-digit binary number $+10 = (01010)_2$ is $2^5 - 10 = 22 = (11000)_2$. An easier algorithm to get two's complement goes via one's complement. Note that $(11111)_2 = 2^5 - 1$. We can get two's complement by adding 1 to one's complement. To get two's complement:

1. Flip all the bits. (Same as taking one's complement).

2. Add 1 to the number.

Negate $13_{10} = 01101_2$ using 5-bit two's complement.

Negate -13_{10} using 5-bit two's complement.

How to convert one's complement number representation into sign-magnitude numbers?

1. Check if the number is positive or negative. Even for one's complement representation, or two's complement representation, if the MSB (Most-significant bit) is 1, then the number is negative, otherwise positive.

2. If positive: For positive numbers, two's complement, one's complement and sign magnitude are the same. No conversion between different representation is needed. 2.b If negative: For negative numbers. Flip the bits of 1's complement. Once you flip the 1's complement bits of a negative number, you get the corresponding positive number.

3. We still want to represent the original negative number. So we set the MSB of sign-magnitude representation to 1. Since the range (min and max) for both n-bit 1's complement and sign-magnitude are the same (between $-(2^{n-1}-1)$ and $2^{n-1}-1$), you can always represent 8-bit 1's complement numbers with needing to extend the 8-bit number to 9-bits.

Example: Convert 8-bit one's complement 10101010 to 8-bit sign-magnitude Let number n = 10101010

1. Is the number +ve or -ve: It is negative because it starts with 1.

2. The number is not positive.

3. Take the 1's complement of the negative number to get the positive part. i.e. Flip the bits:
-n = 01010101 or n = -(01010101)

4. We got the positive part of the number, but we want to represent the original negative number, so we set the MSB bit one. Hence, the equivalent sign-magnitude representation is:
n = 11010101

3.5.5 Two's complement representation

Problem 35. *Determine the decimal values of the following 2's complement 6-digit numbers :*

1. *01011110*

2. *10010111*

Problem 36. *Convert the decimal numbers -17 and +23 into the 6-digit two's complement binary numbers and try adding them. What adjustments will you need to make to get the right result's (23-17=6) in binary representation.*

Problem 37. *Convert the decimal numbers 73, 23, -17, and -163 into signed 8-bit numbers in the following representations:*

1. *Sign and magnitude*
2. *1's complement*
3. *2's complement*

3.5.6 Arithmetic overflow

Problem 38. *Consider addition of 4-digit two's complement binary numbers*

1. $1010_2 + 1101_2$
2. $1011_2 + 1100_2$

In which of the two case overflow happens? Can you come up with a rule to "easily" detect overflow?

Rules for detecting arithmetic overflow:

1. Adding numbers of different signs never produces an overflow.

2. Adding numbers of the same sign may produce an overflow

   (a) Wrong approach: Adding two negative 2's complement numbers always produces an additional carry-over 1, but that in itself isn't an overflow. An example, the range of 4-bit 2's complement numbers is between -8 to +7. Adding -3 to -4 in 2's complement is 1101 + 1100 produces an additional carry over 1. You can ignore the additional carry-over 1 to get the correct answer 1001 = -7 which is within range -8 to 7.

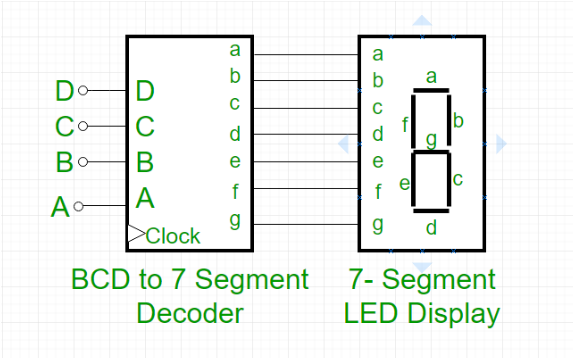
   (b) Approach 1: The easiest way for now to detect overflow is if adding two -ve numbers results in a +ve number, or adding +ve numbers results in a -ve number.

   (c) Approach 2: You can also do a range test in decimal based range test. The range of n-bit 2's complement numbers is between -2^{n-1} and $2^{n-1}-1$. For 5-bit 2's complement numbers, it is between -16 and 15. For 6-bit 2's complement numbers, it is between -32 and 31.

   (d) Approach 3: You can also check the carry-overs of the most significant two bits. If they match, i.e. 0 and 0, or 1 and 1, then there is no overflow. If they do not match, i.e. 0 and 1 or 1 and 0, then there is an overflow.

3.6 Binary coded decimal

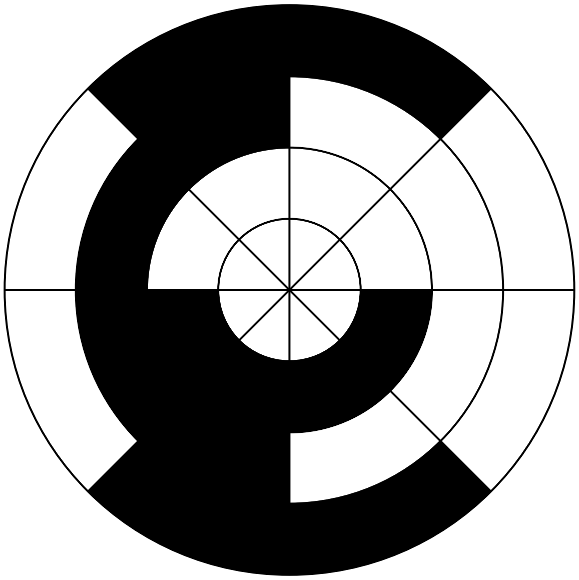
In Binary coded decimal (BCD), each decimal digit is represented by 4 bits. For example, 1047 = $(0001\_0000\_0100\_0111)_{\text{BCD}}$. It is useful in input-output applications where the number has to be either displayed as decimal or received as decimal.



Problem 39. *Convert 11, 23, 35, 57 and 103897 to BCD?*

3.7 Gray code

A sequence of binary numbers where only one bit changes when the number increases by 1. It is helpful in applications like wheel encoders



Problem 40. *Write all possible 3-bit binary numbers in gray-code*

Always show your work/process. Correct final answer is worth less than the correct process. Submit digitally via brightspace.

Problem 41. *Convert the each of the following numbers into binary, decimal, hexadecimal, octal numbers. Show your work. Just filling in the values is not enough.* (8 × 6 *marks)*

	Binary	*Decimal*	*Hexadecimal*	*Octal*
a)	1010_2			
b)	10_0110_2			
c)		329_{10}		
d)		741_{10}		
e)			$7D_{16}$	
f)			$EC3A_{16}$	
g)				351_8
h)				2563_8

Problem 42. *Convert the each of the following numbers into decimal, 8-bit sign-magnitude binary, 8-bit one's complement binary and 8-bit two's complement binary. Show your work.* (6 × 6 *marks)*

	Decimal	*Sign-magnitude*	*One's complement*	*Two's complement*
a)	-79_{10}			
b)	-110_{10}			
c)				0110_1110_2
d)				1011_1101_2
e)			0110_1101_2	
f)			1001_1010_2	

Problem 43. *Convert the decimal numbers to 6-bit two's complement binary and then add them. Check if the addition causes overflow* (3×6 *marks).*

1. $-16_{10} - 7_{10}$
2. $19_{10} - 5_{10}$
3. $-4_{10} - 29_{10}$

Problem 44.

1. *Convert* 299_{10} *to binary coded decimal (BCD). (2 marks)*
2. *Convert* $1001\_0111\_0101_{BCD}$ *to decimal. (2 marks)*
3. *Convert* 0110_1101_{BCD} *to binary. (4 marks)*

Chapter 4

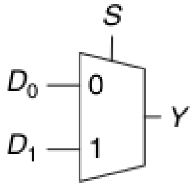
Adders, Muxes and Decoders

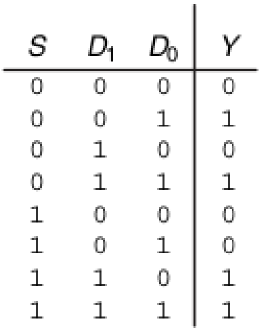
4.1 Objectives

1. Design combinational circuits using multiplexers and decoders

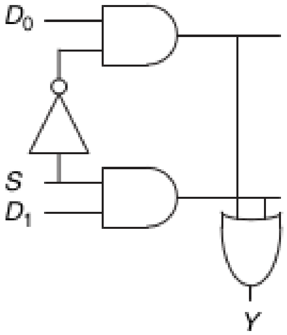
4.2 Design combinational circuit using multiplexers [1, Section 2.8.1]

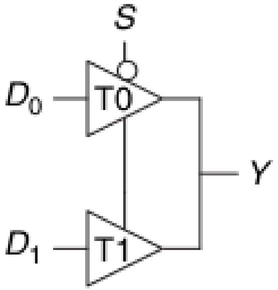
4.2.1 Review: 2to1 Multiplexer (MUX)





S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1





$Y = D_0\overline{S} + D_1 S$

4.2.2 Wider multiplexers

Draw the symbol for a 4:1 MUX, an 8:1 MUX and a 2^N : 1 MUX and write corresponding Boolean expressions.

Example 21. *Design a circuit for* $Y = A\bar{B} + \bar{B}\bar{C} + \bar{A}BC$ *using a 8:1 MUX.*

Remark 3. *A* $2^N : 1$ *MUX can be used to program any N-input logic function.*

Example 22. *Design a circuit for* $Y = A\bar{B} + \bar{B}\bar{C} + \bar{A}BC$ *using a 4:1 MUX and NOT gates only.*

Remark 4. *A* $2^{N-1} : 1$ *MUX can be used to program any N-input logic function, if we use literals on the input side.*

Example 23. *Design a circuit for* $Y = \bar{A}C + \bar{A}B + B\bar{D}$ *using a 8:1 MUX and NOT gates only. Also design using 4:1 MUX and other gates. fewest gates.*

4.3 Encoders and Decoders

Example 24. *Draw the symbol and the truth table for 2:4 decoder. Also write the logic expressions.*

Example 25. *Draw the symbol and the truth table for 3:8 decoder, 4:16 decoder and* $N : 2^N$ *decoder. Also write the logic expressions.*

Example 26. *Design a circuit for a XOR gate using a 2:4 decoder and an OR gate.*

Example 27. *Design a circuit for* $Y = A\bar{B} + \bar{B}\bar{C} + \bar{A}BC$ *using a 3:8 decoder and an OR gate.*

4.3.1 (Priority) Encoders

Example 28. *Draw symbol and truth table for a 4:2 priority encoder.*

Example 29. *Draw symbol and truth table for a 8:3 priority encoder.*

Chapter 5

Sequential Logic

4.Provide Verilog idioms for each of the sequential circuits

5.The build up of SR latch is interesting from here. `https://www.youtube.com/watch?v=BYN8Zmk6HJY`

6.JK latch and T ff are not that interesting

7.Metastability must be introduced

5.1 Objectives

1. Understand timing diagrams, gate delays and critical path
2. Design Hazard-free two level circuits
3. Building blocks of sequential circuits
4. Analyze a sequential circuit and derive a state-table and a state-graph
5. Derive a state graph or state table from a word description of the problem
6. Understanding the structure of an FPGA

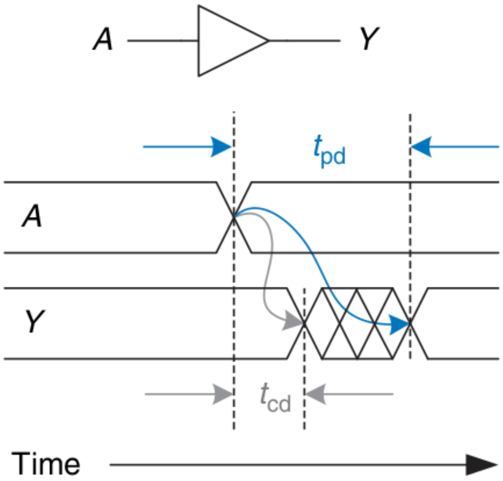
5.2 Why do we need sequential circuits?

Example 30. *Think about this problem: Design an occupancy counter that depends on a sensor S at the class door. The sensor is triggered every time a person passes through the door. The counter can be reset to zero with a reset button. Assume we only need up to two bit counter C_1C_0. Draw a truth table for this circuit. Do you have requisite knowledge for designing this circuit? Can this circuit be designed without a memory element?*

5.3 Timing diagrams and propagation delays

Example 31 (Timing diagram). *Draw a timing diagram for an ideal NAND gate.*

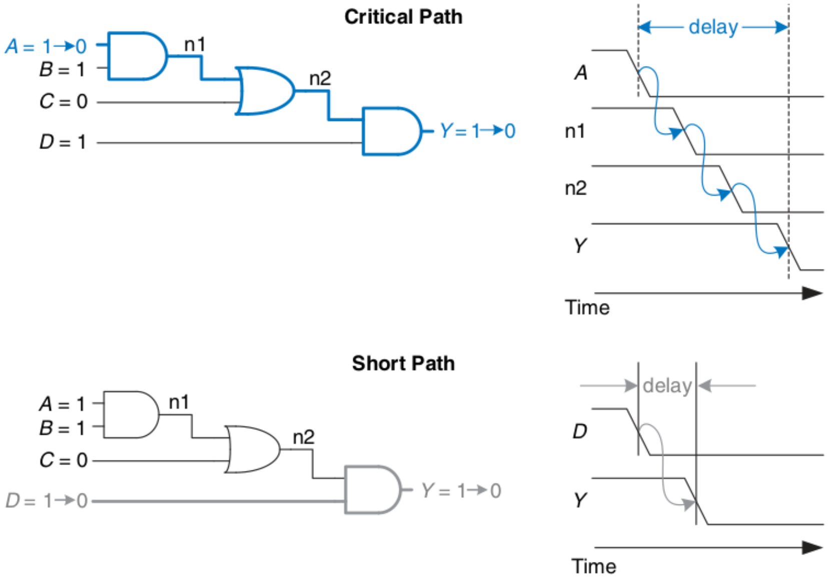
5.3.1 Delays



Definition 1 (Propagation delay (t_{pd})).

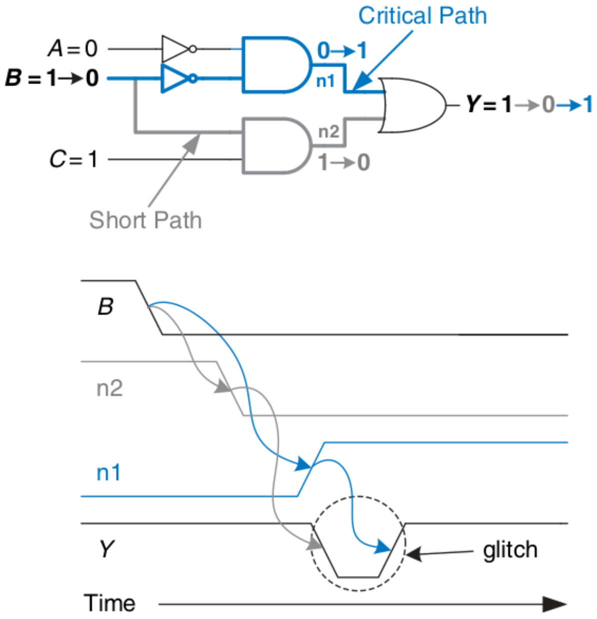
Definition 2 (Contamination delay (t_{cd})).

5.3.2 Paths



Example 32. *Find the propagation delay of the circuit above given that propagation delay of each gate is* 100*ps add contamination delay of* 60*ps.*

5.4 Glitches or Hazards



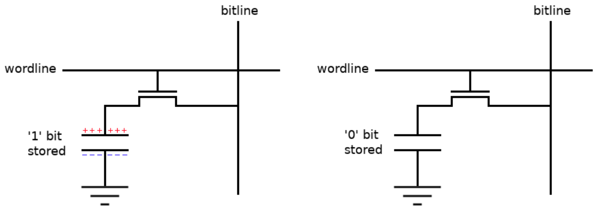
Definition 3 (Glitch or Hazard)**.**

Example 33. *Design a circuit that fixes the glitch in the above circuit (also known as glitch-free or hazard-free circuit).*

5.5 How to create memory element from circuits

Two types of memory

1. Volatile memory. For example, RAM, CPU registers.
2. Non-volatile memory. For example, SSD, Flash drives. (Not covered in this course)
    (a) Memories that require periodic refreshing. For example, DRAM: Dynamics Random Access memory (Not covered in this course)



[1]

(b) Memories that are always refreshing. For example, SRAM: Static Random Access memory [3, Appendix B.64]

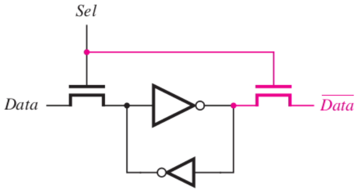
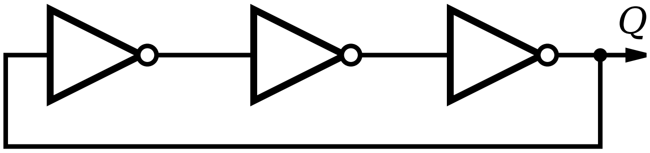


Figure B.64 An SRAM cell.

5.6 Latches and Flip-Flops [1, Sec 3.2]

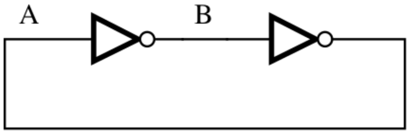
Example 34 (Ring oscillator). *[1, Sec 3.31] How many stable states does the following circuit have?*



Definition 4 (Astable circuits).

Example 35. *Analyze the timing diagram of the following circuit.*

[1] Image source: `allaboutcircuits.com/technical-articles/introduction-to-dram-dynamic-random-access-memory/`

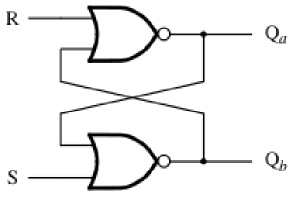


Definition 5 (Bistable circuits)**.**

Definition 6 (Characteristic or state table)**.** *Draw the characteristic or state table of the above circuit.*

5.6.1 SR (Set-Reset) latch [1, Sec 3.2.1]

Definition 7 (SR latch)**.** *The following circuit is called the SR latch.*

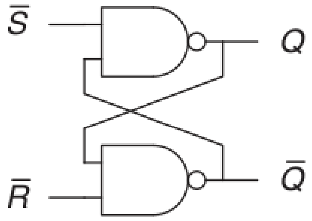


1. *How many stable states does this circuit have?*

2. *Draw its characteristic or state table.*

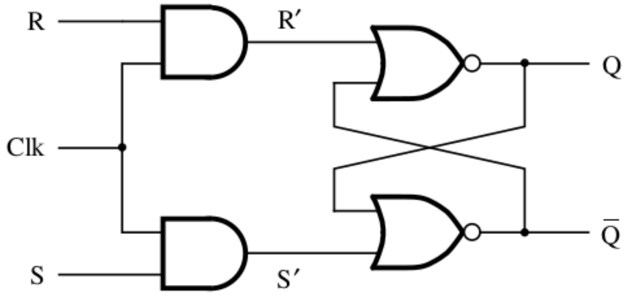
3. *Draw SR latch symbol*

Problem 45 (SR latch using NAND gates)**.** *Draw the characteristic or state table for the following circuit*



5.6.2 Gated SR latch [3, Sec 5.2]

Definition 8 (Gated SR latch)**.** *The following circuit is called the Gated SR latch.*

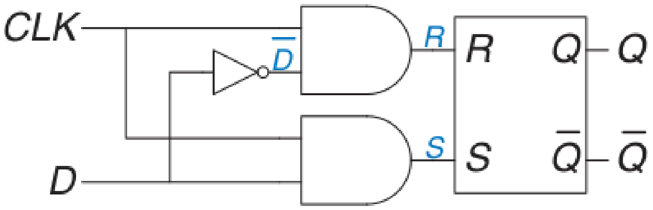


1. *Draw its characteristic table.*

2. *Draw the Gated SR latch symbol*

5.6.3 D (Data) latch [1, Sec 3.2.2]

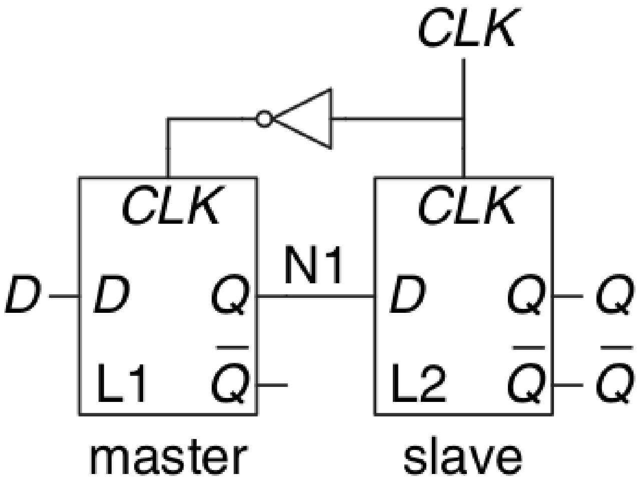
Definition 9 (D latch)**.** *The following circuit is called the D latch.*



1. *Draw its characteristic table.*
2. *Draw the D latch symbol*

5.6.4 D flip-flop [1, Sec 3.2.2]

Definition 10 (D flip-flop)**.** *The following circuit is called the D flip-flop.*



1. *Draw its timing diagram*
2. *Draw its characteristic table.*
3. *Draw the D flip-flop symbol*

Remark 5. *What is the difference between a latch and a flip-flop?*

Example 36. *Add a* RESET *signal to the D flip-flop that resets the state of flip-flop to 0.*

Example 37. *The toggle (T) flip-flop has one input, CLK, and one output, Q. On each rising edge of CLK, Q toggles to the complement of its previous value. Draw a schematic for a T flip-flop using a D flip-flop and an inverter.*

Problem 46. *A JK flip-flop receives a clock and two inputs, J and K. On the rising edge of the clock, it updates the output, Q. If J and K are both 0, Q retains its old value. If only J is 1, Q becomes 1. If only K is 1, Q becomes 0. If both J and K are 1, Q becomes the opposite of its present state.*

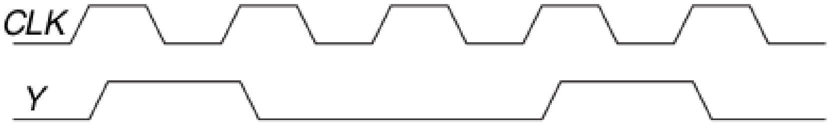
1. *Construct a JK flip-flop using a D flip-flop and some combinational logic.*
2. *Construct a D flip-flop using a JK flip-flop and some combinational logic.*
3. *Construct a T flip-flop (see Exercise 3.9) using a JK flip-flop.*

5.7 Finite State Machines [1, Sec 3.4]

[2]

Example 38. *Design an occupancy counter that depends on a sensor S at the class door. The sensor is triggered every time a person passes through the door. Assume that the counter starts at zero. Assume we only need up to two bit counter C_1C_0. Draw a state table for this circuit.*

Problem 47. *A divide-by-N counter has one output and no inputs. The output Y is HIGH for one clock cycle out of every N. In other words, the output divides the frequency of the clock by N. The waveform for a divide-by-3 counter is shown here:*



(a)

Sketch circuit designs for such a counter

Problem 48. *Design a 3-bit counter which counts in the sequence: 001, 011, 010, 110, 111, 100, (repeat) 001, ...*

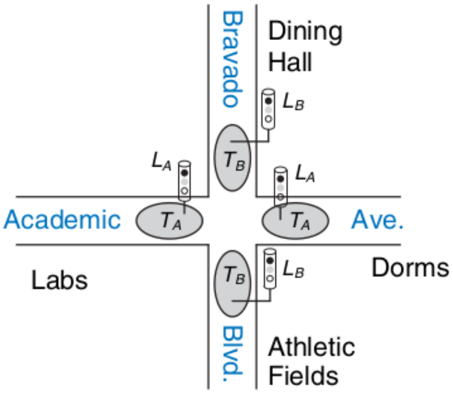
Example 39. *Design an odd-even counter for an single bit input. The output of this circuit should be 1 if the number of 1s to the input have been odd so far and 0 otherwise.*

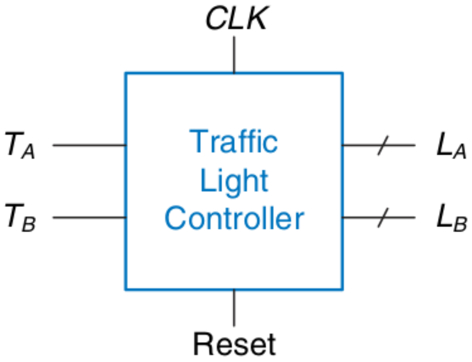
Example 40 (Sequence detectors)**.** *A sequential circuit has one input and one output. The output becomes 1 and remain 1 thereafter when at least two 0's and at least two 1's have occurred as inputs regardless of the order of*

Example 41. *Consider the problem of inventing a controller for a traffic light at a busy intersection on campus. There are two traffic sensors, T_A and T_B , on Academic Ave. and Bravado Blvd., respectively. Each sensor indicates TRUE if students are present and FALSE if the street is empty. There are two traffic lights, L_A and L_B, to control traffic. Each light receives digital inputs specifying whether it should be green, yellow, or red. When the system is reset, the lights are green on Academic Ave. and red on Bravado Blvd. As long as traffic is present on Academic Ave., the lights do not change. When there is no longer traffic on Academic Ave., the light on Academic Ave. becomes yellow for 5 seconds before it turns red and Bravado Blvd.'s light turns green. Similarly, the Bravado Blvd. light remains green as long as traffic is present on the boulevard, then turns*

[2] These notes will not fit on your note sheet.

yellow and eventually red.





1. *Draw a state transition diagram*
2. *Draw a state table*
3. *Assign binary encodings to each of the states*
4. *Redraw the state table with binary encodings. Design a minimal SOP boolean expression.*
5. *Assign binary encodings to each of the output and redraw the output table. Design a minimal SOP boolean expression for the outputs.*

Problem 49. *Design a circuit for a 2x2 pixel resolution pong game, where the ball can only occupy 4 possible pixels and a single paddle occupies another 2 pixels. The ball bounces of the paddle when the paddle is in the correct row. To keep it interesting, the ball takes a different path from the source path. Track the score with a single bit counter.*

5.8 Mealy Moore Sequence detector

5.9 Objectives

1. Analyse and design both Mealy and Moore sequential circuits with multiple inputs and multiple outputs
2. Convert between Mealy and Moore designs

5.10 Mealy vs Moore Finite State Machines

Definition 11 (Finite State Machines (FSM))**.** *[1, Sec 3.4]*

Definition 12 (Mealy FSM)**.** *[1, Sec 3.4.3]*

Definition 13 (Moore FSM)**.** *[1, Sec 3.4.3]*

Example 42. *A sequential circuit has one input (X) and one output (Z). The circuit examines groups of four consecutive inputs and produces an output Z=1 if the input sequence 0010 or 0001 occurs. The sequences can overlap. Draw both Mealy and Moore timing diagrams. Find the Mealy and Moore state graph.*

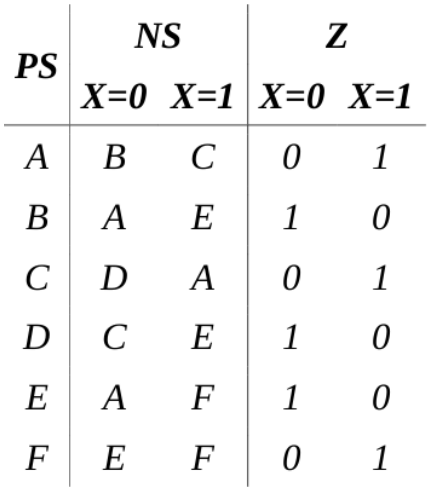
Problem 50. *A sequential circuit has one input (X) and one output (Z). The circuit examines groups of four consecutive inputs and produces an output Z=1 if the input sequence 0101 or 1001 occurs. The circuit resets after every four inputs. Draw both Mealy and Moore timing diagrams. Find the Mealy and Moore state graph.*

5.11 State reduction via Implication tables

8.Skip this section

To minimize the number of states, we will identify "equivalent states" and eliminate any redundancy found. Two states are equivalent if they have equivalent next states and the same output for each possible input condition. To find equivalent states we will create an "implication table" which looks at pairs of states and identifies which states have to be equivalent if this pair is to be equivalent. We can use a table to hold information about each pair.

To investigate all possible pairs, we could use a square table such as this to record information about pairs of states. But note every pair represented in the upper right "triangle" of the table is also listed in the lower left "triangle" of the table. Furthermore, the diagonal of the table will only present information about a state being equivalent to itself. Only the part of the table in bold is needed to investigate all possible pairs of states:



PS	NS		Z	
	X=0	X=1	X=0	X=1
A	B	C	0	1
B	A	E	1	0
C	D	A	0	1
D	C	E	1	0
E	A	F	1	0
F	E	F	0	1

Figure 5.1: Example state table

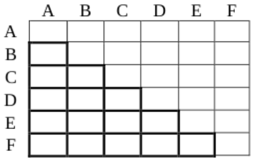


Figure 5.2: Implication table

5.12 State assignment

9.Skip this section

5.13 Finite State Machine Optimization

5.14 Objectives

10.highlight one-hot state vector 11.highlight the timing vs space trade-offs of mealy vs moore FSM in context of verilog 12.Consider describing the stages of Synthesis

1. Analyse and design both Mealy and Moore sequential circuits with multiple inputs and multiple outputs
2. Convert between Mealy and Moore designs
3. Perform a state assignment using the guideline method
4. Reduce the number of states in a state table using row reduction and implication tables

5.15 Mealy vs Moore Finite State Machines

Definition 14 (Finite State Machines (FSM))**.** *[1, Sec 3.4]*

Definition 15 (Mealy FSM)**.** *[1, Sec 3.4.3]*

Definition 16 (Moore FSM)**.** *[1, Sec 3.4.3]*

Example 43. *A sequential circuit has one input (X) and one output (Z). The circuit examines groups of four consecutive inputs and produces an output Z=1 if the input sequence 0010 or 0001 occurs. The sequences can overlap. Draw both Mealy and Moore timing diagrams. Find the Mealy and Moore state graph.*

Problem 51. *A sequential circuit has one input (X) and one output (Z). The circuit examines groups of four consecutive inputs and produces an output Z=1 if the input sequence 0101 or 1001 occurs. The circuit resets after every four inputs. Draw both Mealy and Moore timing diagrams. Find the Mealy and Moore state graph.*

5.16 Full procedure for designing sequential logic circuit

1. Convert the word problem to a state transition diagram. Let the states be $S_0, S_1, S_2, \ldots, S_n$.

2. Draw state transition table with named states. For example,

Present State	Next State		Outputs	
	X = 0	X = 1	X=0	X=1
S_0	S_1	S_2	0	0
S_1	S_2	S_0	0	0
⋮	⋮	⋮	⋮	⋮

3. State reduction step: Reduce the number of required states to a minimum. Eliminate unnecessary or duplicate states.

4. State assignment step: Assign each state a binary representation. For example,

State name	State assignments ($Q_2Q_1Q_0$)
S_0	000
S_1	001
⋮	⋮

5. Draw State assigned transition table. For example,

Inputs (X_1X_0)		Present State (Q_1Q_0)	Next State ($Q_1^+Q_0^+$)	Outputs (Z_1Z_0)	
0	0	00	01	0	0
0	0	01	10	0	0
⋮	⋮	⋮	⋮	⋮	⋮

(a) Use excitation tables to find truth tables for the combinational circuits. For example, the excitation table for J-K ff is

Q	Q^+	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

5.17 State assignment by guideline method [2, Section 8.2.5]

13.skip this section

5.17.1 State Maps

Example 44. *Draw a state map for a sequential assignment of the states*

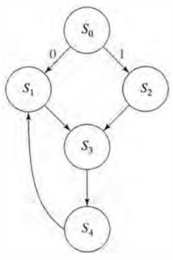
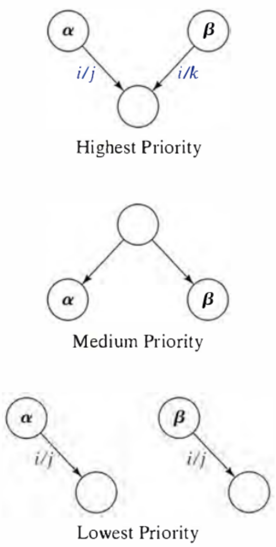


Figure 8.27 Five-state finite state machine.

5.17.2 Guideline method

Guideline method states that the following states should be adjacent in the state map according the following priorities:



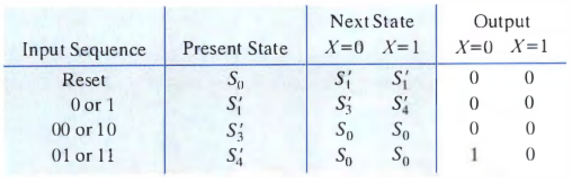
Highest Priority

Medium Priority

Lowest Priority

Figure 8.29 **Adjacent assignment priorities.**

Example 45. *A state transition table is given. Find optimal state assignment by using the guideline method.*



		Next State		Output	
Input Sequence	Present State	$X=0$	$X=1$	$X=0$	$X=1$
Reset	S_0	S_1'	S_1'	0	0
0 or 1	S_1'	S_3'	S_4'	0	0
00 or 10	S_3'	S_0	S_0	0	0
01 or 11	S_4'	S_0	S_0	1	0

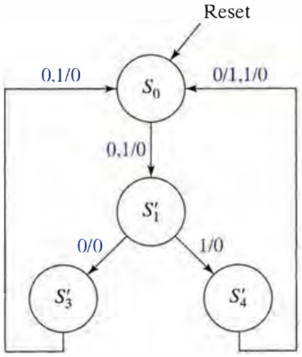
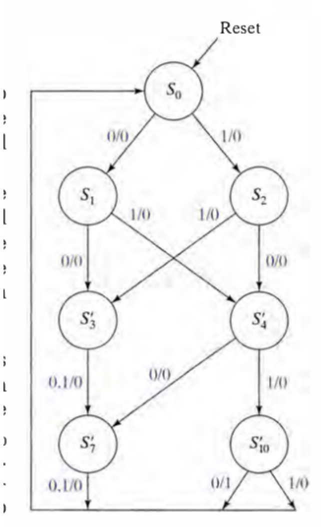


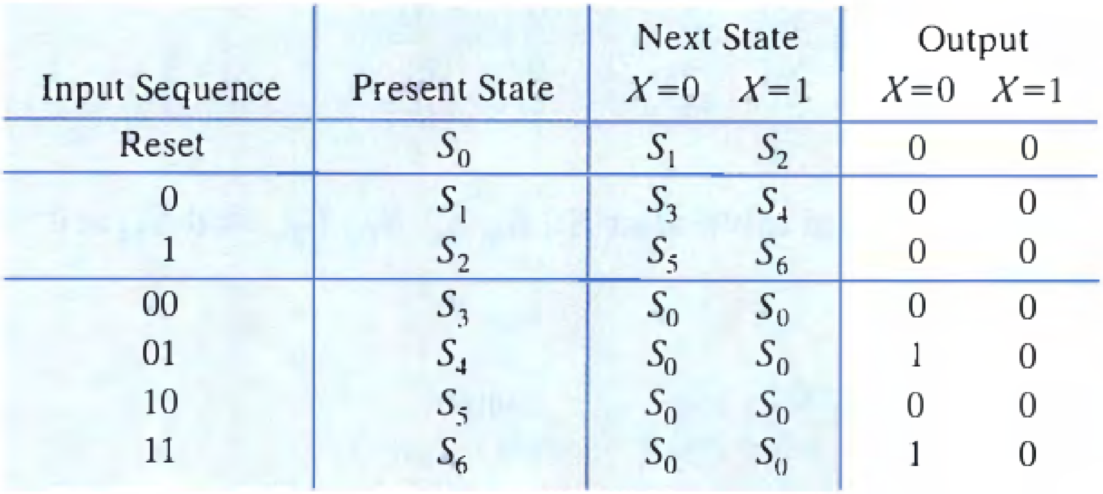
Figure 8.30 **Reduced state diagram for 3-bit sequence detector.**

Example 46. *Draw a Mealy FSM for detecting binary string 0110 or 1010. The machine returns to the reset state after each and every 4-bit sequence. Draw the state transition diagram on your own as practice problem. The state transition diagram is given here. Find optimal state assignment by using the guideline method.*



5.18 State reduction by implication chart

Example 47. *Design a Mealy FSM for detecting binary sequence 010 or 0110. The machine returns to reset state after each and every 3-bit sequence. For now the state transition table is given. Reduce the following state transition table*



Input Sequence	Present State	Next State $X=0$	Next State $X=1$	Output $X=0$	Output $X=1$
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_0	S_0	0	0
01	S_4	S_0	S_0	1	0
10	S_5	S_0	S_0	0	0
11	S_6	S_0	S_0	1	0

5.18.1 Implication chart Summary

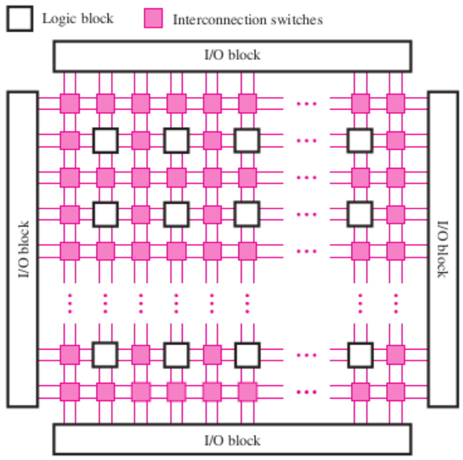
The algorithms for state reduction using the implication chart method consists of the following steps

1. Construct the implication chart, consisting of one square for each possible combination of states taken two at a time.
2. For each square labeled by states S_i and S_j, if the outputs of the states differ, mark the square with an X; the states are not equivalent. Otherwise, they may be equivalent. Within the square write implied pairs of equivalent next states for all input combinations.
3. Systematically advance through the squares of the implication chart. If the square labeled by states S_i, S_j contains an implied pair S_m, S_n and square S_m, S_n is marked with an X, then mark S_i, S_j with an X. Since S_m, S_n are not equivalent, neither are S_i, S_j.
4. Continue executing Step 3 until no new squares are marked with an X.
5. For each remaining unmarked square S_i, S_j, we can conclude that S_i, S_j are equivalent.

Chapter 6

More definitions

6.1 FPGA [3, Section B.6.5]



(a) General structure of an FPGA

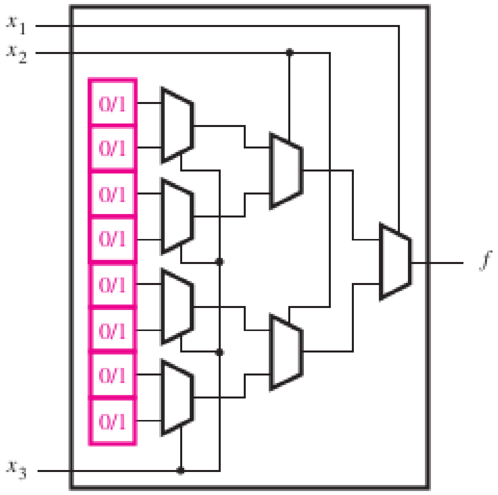


Figure B.37 A three-input LUT.

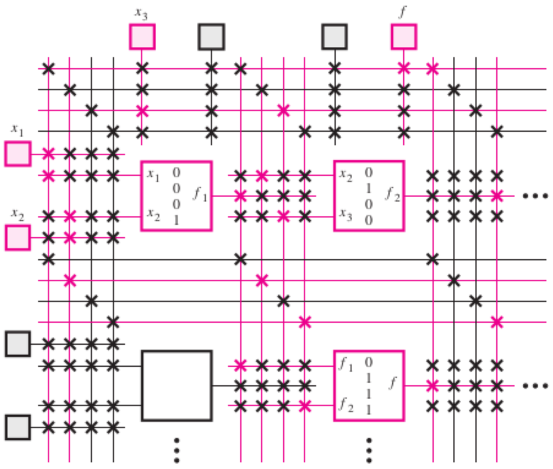


Figure B.39 A section of a programmed FPGA.

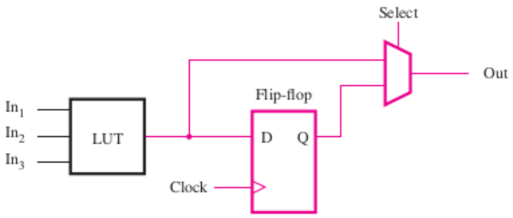


Figure B.38 Inclusion of a flip-flop in an FPGA logic element.

Definition 17 (Random Access Memory (RAM))**.** *Structure of a RAM is as follows:*

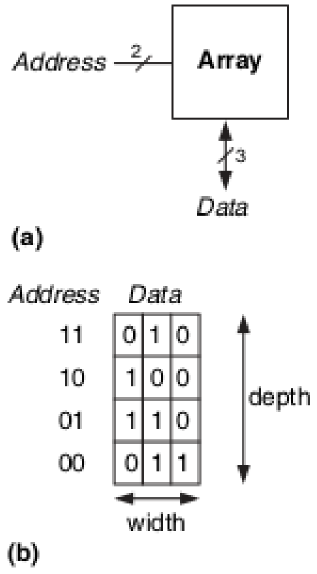


Figure 5.39 4 × 3 memory array: (a) symbol, (b) function

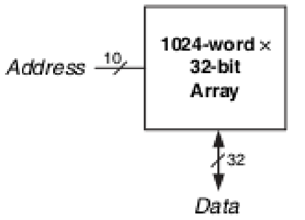


Figure 5.40 32 Kb array: depth = 2^{10} = 1024 words, width = 32 bits

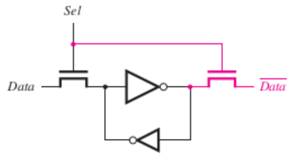


Figure B.64 An SRAM cell.

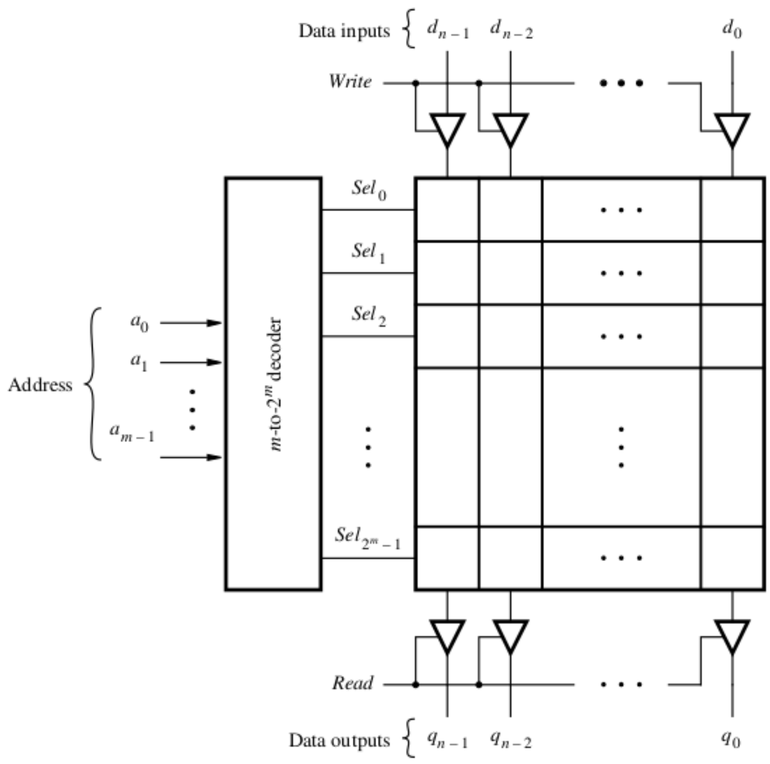


Figure B.66 A $2^m \times n$ SRAM block.

Definition 18 (Read Only Memory (ROM))**.** *Structure of a ROM is as follows:*

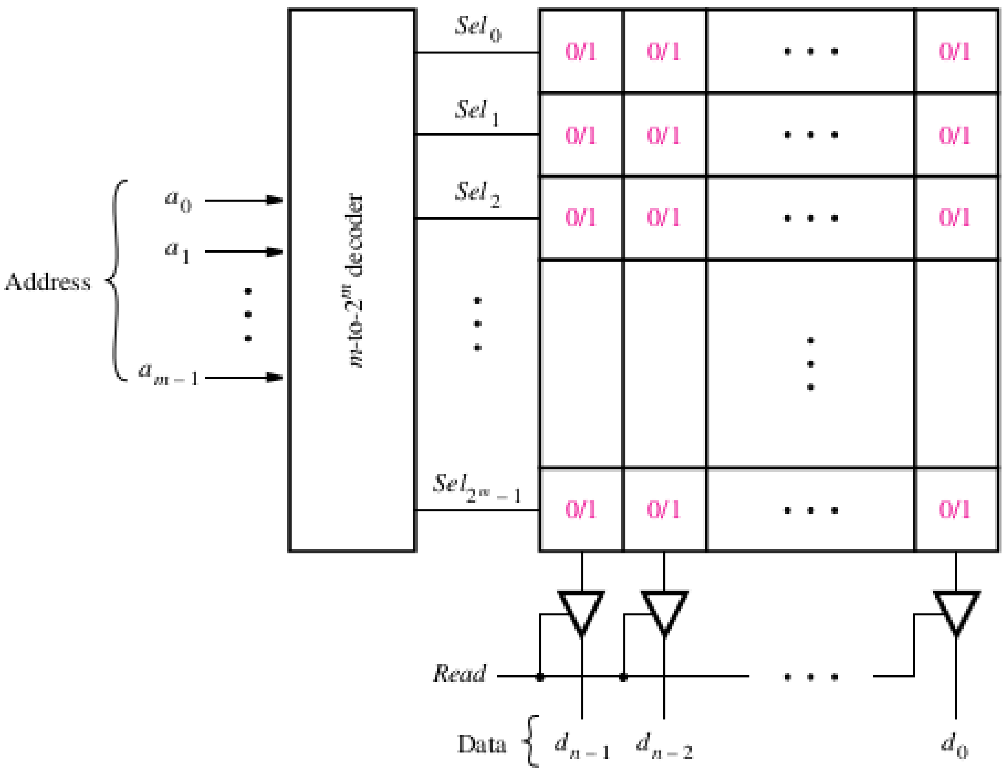
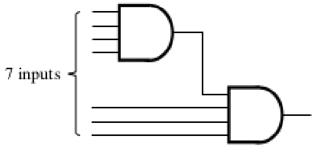


Figure B.72 A $2^m \times n$ read-only memory (ROM) block.

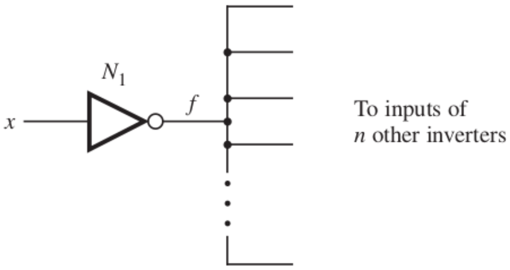
Definition 19 (Fan-in). *The fan-in of a logic gate is number of inputs to a logic gate. [3, Section B.8.9]*



Remark 6 (Fan-in). *The fan-in of a gate is limited by the propagation delay t_p. Higher the fan-in, higher the t_p. The output voltage thresholds like V_{OL} and V_{OH} also limit fan-in. Higher the fan-in, higher is V_{OL} (and lower is the V_{OH}).*

Example 48. *Implement an OR gate with fan-in of 7 using OR gates with fan-in of 3.*

Definition 20 (Fan-out). *The fan-out of a logic gate is the maximum number of other gates that can be connected to output of a gate. [3, Section B.8.9]*



Definition 21 (Programmable Logic Array (PLA))**.** *Structure of a PLA:*

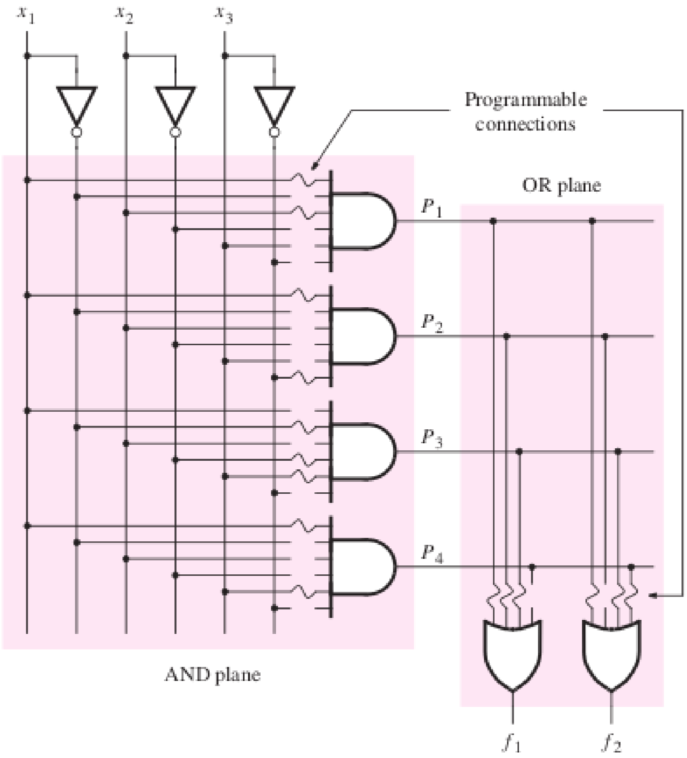


Figure B.26 Gate-level diagram of a PLA. *[3, Section B.6.1]*

Definition 22 (Programmable Array Logic (PAL))**.** *Structure of a PAL:*

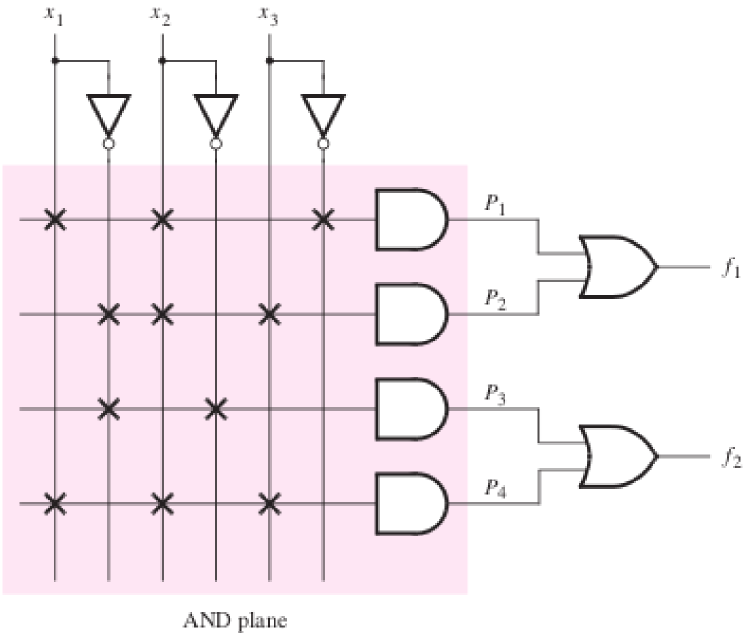
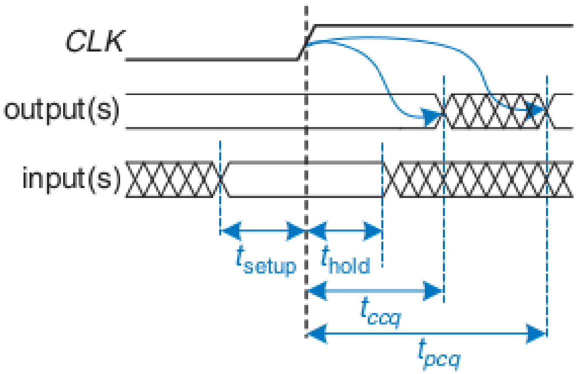


Figure B.28 An example of a PAL.

[3, Section B.6.2]

Example 49. *What is the difference between PLA and PAL?*

6.2 Timing parameters for sequential circuit [1, Section 3.5]



Definition 23 (Setup time t_{su} of a latch/flip-flop)**.** *Time for which input must be stable before the clock edge.*

Definition 24 (Hold time t_h of a latch/flip-flop)**.** *Time for which input must be stable after the clock edge.*

Definition 25 (Clock-to-Q contamination delay t_{ccq} of a latch/flip-flop)**.** *Time taken to influence (contaminate) the Q output after the clock edge.*

Definition 26 (Clock-to-Q propagation delay t_{ccq} of a latch/flip-flop)**.** *Time taken for Q output to stabalize after the clock edge.*

Chapter 7

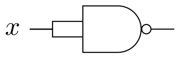
Quine McCluskey

7.1 Circuit design using NAND/NOR gates

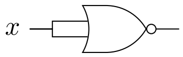
Example 50. *Implement the function $f(x_1, x_2, x_3) = \sum m(2, 3, 4, 6, 7)$ using (1) NAND gates only and (2) NOR gates only.*

Remark 7. *NAND-NAND logic is generated from SOP form. NOR-NOR logic is generated from POS form.*

Remark 8. *NOT gate can also be created from a NAND gate $\bar{x} = \overline{x \cdot x}$.*



Remark 9. *NOT gate can also be created from a NOR gate $\bar{x} = \overline{x + x}$.*



Problem 52. *Design the simplest circuit that implements the function $f(x_1, x_2, x_3) = \sum m(3, 4, 6, 7)$ using (1) NAND gates only (2) NOR gates only.*

7.2 PI Table reduction and Petrick's method

This is not in the text-book. For additional reading, please refer to the linked resources on the website.

Definition 27 (Implicant)**.** *Given a function f of n variables, a product term P is an implicant of f if and only if for every combination of values of the n variables for which $P = 1$, f is also equal to 1.*

Definition 28 (Prime Implicant)**.** *A prime implicant of a function f is an implicant which is no longer an implicant if any literal is removed from it.*

There are 4 main steps in the Quine-McCluskey algorithm/PI Table reduction and Petrick's method:

1. Generate Prime Implicants
2. Construct Prime Implicant Table. PIs as columns, and minterms as rows (don't cares are excluded).
3. Reduce Prime Implicant Table by repeating following steps until they it cannot be reduced further
   (a) Remove Essential Prime Implicants
   (b) Row Dominance: Remove *dominating* rows. (i.e. unnecessary minterms)
   (c) Column Dominance: Remove *dominated* columns. (i.e. remove unnecessary PIs)
4. Solve Prime Implicant Table by Petrick's method

7.2.1 Generate Prime Implicants

Example 51. *Generate prime implicants of the function $F(A, B, C, D) = \sum m(0, 2, 5, 6, 7, 8, 10, 12, 13, 14, 15)$ using Quine-McCluskey method*

Steps:

1. Start with writing minterms in binary format (include don't cares as minterms).

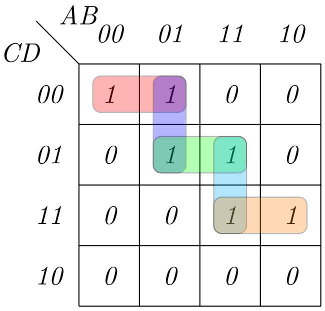
2. Create potential groups of minterms that can be combined (merged). The only minterms that can be combined differ only be single 1. Create a new list of combined minterms as n-1 literal implicants.
3. Check off the minterms that could be combined. Unchecked minterms are prime implicants (PIs).
4. Repeat the grouping process with n-1 literal implicants.

Problem 53. *Generate PIs for the function* $F(A,B,C,D) = \sum m(0,2,3,4,5,6,7,8,9,10,11,12,13)$.

7.2.2 Prime Implicants table and reduction

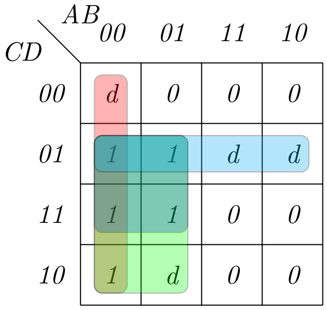
Example 52. *Reduce the prime implicants* $\{\bar{B}\bar{D}, C\bar{D}, BD, BC, A\bar{D}, AB\}$ *using prime implicants table.*

Example 53.



CD \ AB	00	01	11	10
00	*1*	*1*	*0*	*0*
01	*0*	*1*	*1*	*0*
11	*0*	*0*	*1*	*1*
10	*0*	*0*	*0*	*0*

Example 54.



$CD \backslash AB$	*00*	*01*	*11*	*10*
00	*d*	*0*	*0*	*0*
01	*1*	*1*	*d*	*d*
11	*1*	*1*	*0*	*0*
10	*1*	*d*	*0*	*0*

Example 55. *Reduce the following PI table*

	$\bar{A}\bar{D}$	$\bar{B}\bar{D}$	$\bar{C}\bar{D}$	$\bar{A}C$	$\bar{B}C$	$\bar{A}B$	$B\bar{C}$	$A\bar{B}$	$A\bar{C}$
0	*X*	*X*	*X*						
2	*X*	*X*		*X*	*X*				
3				*X*	*X*				
4	*X*		*X*			*X*	*X*		
5						*X*	*X*		
6	*X*			*X*		*X*			
7				*X*		*X*			
8		*X*	*X*					*X*	*X*
9								*X*	*X*
10		*X*			*X*			*X*	
11					*X*			*X*	
12			*X*		*X*		*X*		*X*
13							*X*		*X*

7.2.3 Petrick's method

Example 56. *Solve the Prime Implicant table using Petrick's method*

	$p_1 = \bar{A}C$	$p_2 = \bar{B}C$	$p_3 = \bar{A}B$	$p_4 = B\bar{C}$	$p_5 = A\bar{B}$	$p_6 = A\bar{C}$
3	X	X				
5			X	X		
7	X		X			
9					X	X
11		X			X	
13				X		X

Example 57. *Find the minimum SOP expression for the function* $F(A, B, C, D) = \sum m(2, 3, 7, 9, 11, 13) + \sum d(1, 10, 15)$ *using Quine-McCluskey method.*

Chapter 8

Analog details

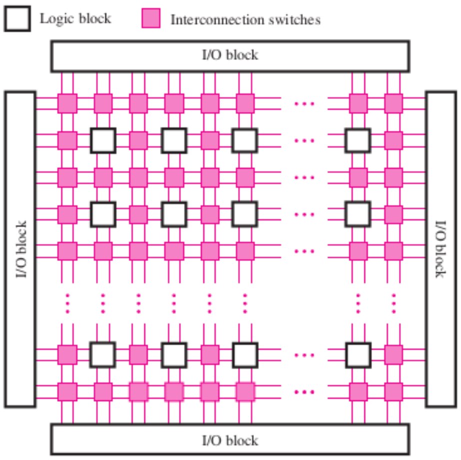
Some of the material is out of the textbook. Additional resources include Appendix B of Brown and Vranesic book, “Fundamentals of digital logic.”

8.1 Objectives

1. Describe how tri-state and open-collector outputs are different from totem-pole outputs

2. Compute noise margin of one device driving the same time

8.2 FPGA [3, Section B.6.5]



(a) General structure of an FPGA

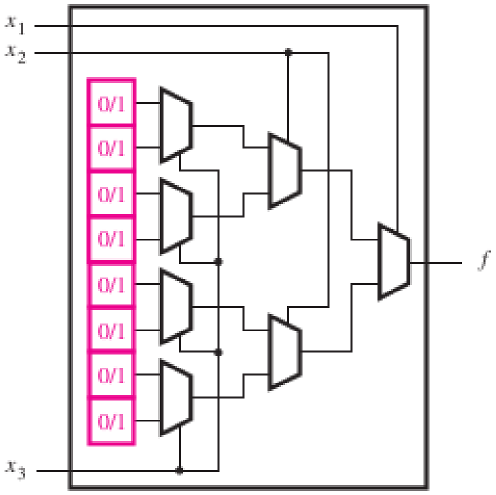


Figure B.37 A three-input LUT.

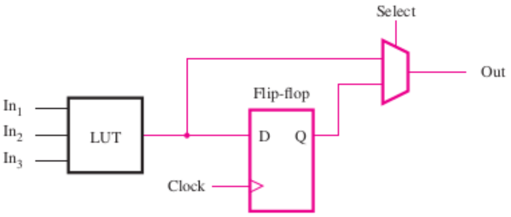


Figure B.38 Inclusion of a flip-flop in an FPGA logic element.

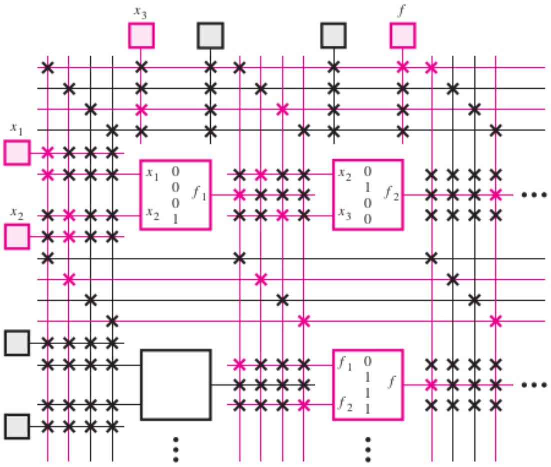
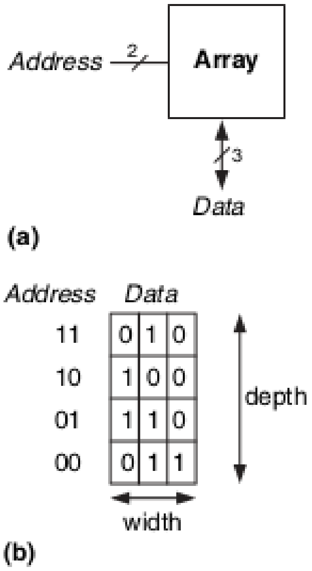


Figure B.39 A section of a programmed FPGA.

Definition 29 (Random Access Memory (RAM))**.** *Structure of a RAM is as follows:*



Address	Data		
11	0	1	0
10	1	0	0
01	1	1	0
00	0	1	1

Figure 5.39 4 × 3 memory array: (a) symbol, (b) function

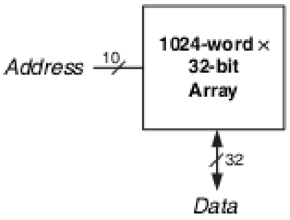


Figure 5.40 32 Kb array: depth = 2^{10} = 1024 words, width = 32 bits

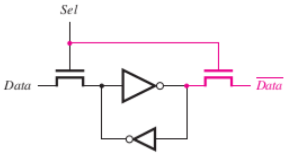


Figure B.64 An SRAM cell.

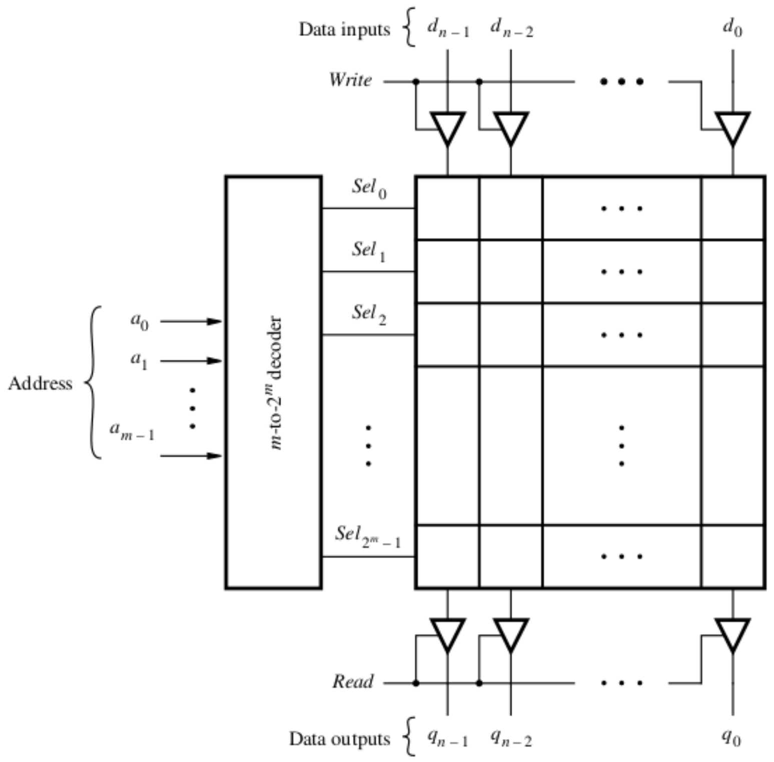


Figure B.66 A $2^m \times n$ SRAM block.

Definition 30 (Read Only Memory (ROM))**.** *Structure of a ROM is as follows:*

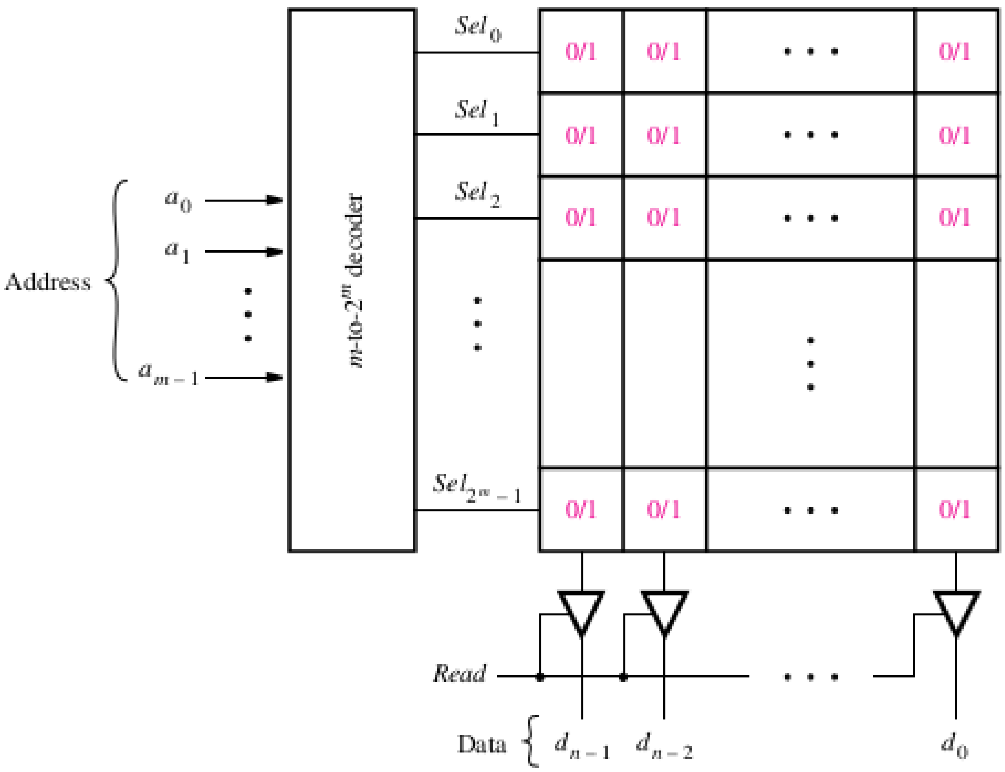
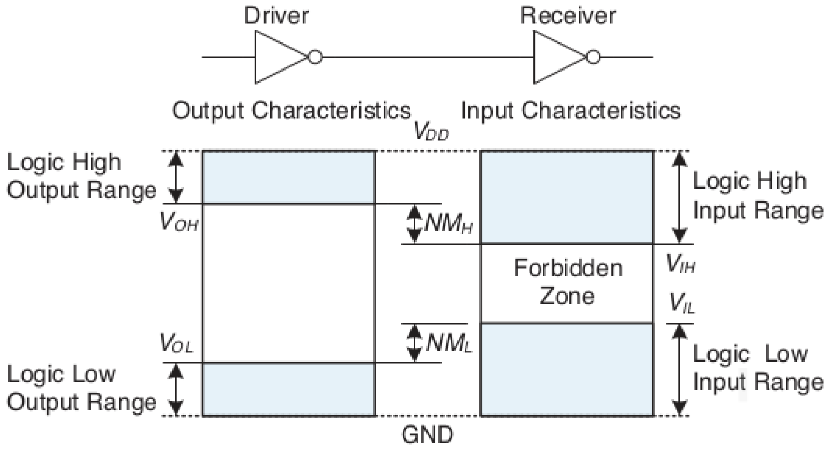


Figure B.72 A $2^m \times n$ read-only memory (ROM) block.

Example 58. *Draw a Multiplexer using sum of products form.*

8.3 Logic levels and Noise Margins



Definition 31 (Supply Voltage ($V_{DD}/V_{CC}/V_{SS}$)).

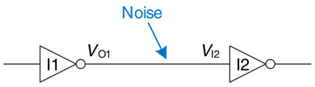
Definition 32 (Ground Voltage (V_{GND})).

Definition 33 (Input high (V_{IH}) and Input Low (V_{IL}) of a gate).

Definition 34 (Output high (V_{OH}) and Output low (V_{OL}) of gate).

Definition 35 (Positive logic and Negative logic).

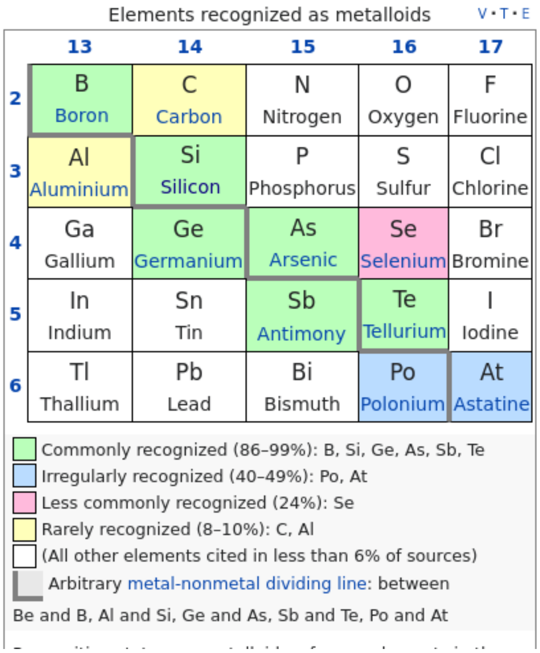
Definition 36 (Noise margins (NM_L and NM_H) of a channel).



Example 59.
If $V_{DD} = 5V$, $V_{IL} = 1.35V$, $V_{IH} = 3.15V$, $V_{OL} = 0.33V$ and $V_{OH} = 3.84V$ for both the "inverters", then what are the low and high noise margins? Can the circuit tolerate 1V of noise at the channel?

8.4 Semiconductors and Doping

Not in syllabus but good to know



Elements recognized as metalloids V • T • E

	13	14	15	16	17
2	B Boron	C Carbon	N Nitrogen	O Oxygen	F Fluorine
3	Al Aluminium	Si Silicon	P Phosphorus	S Sulfur	Cl Chlorine
4	Ga Gallium	Ge Germanium	As Arsenic	Se Selenium	Br Bromine
5	In Indium	Sn Tin	Sb Antimony	Te Tellurium	I Iodine
6	Tl Thallium	Pb Lead	Bi Bismuth	Po Polonium	At Astatine

🟩 Commonly recognized (86–99%): B, Si, Ge, As, Sb, Te

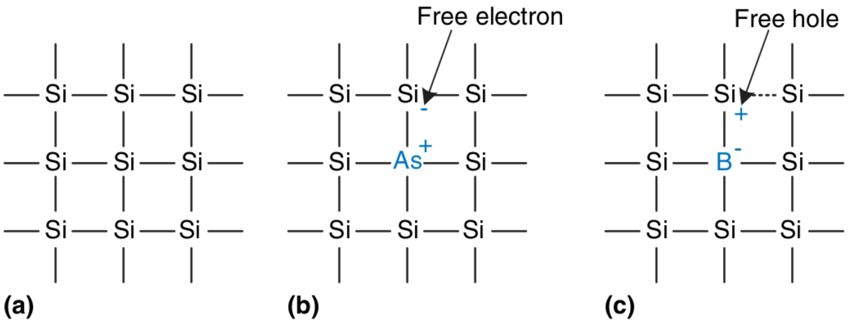
🟦 Irregularly recognized (40–49%): Po, At

🟪 Less commonly recognized (24%): Se

🟨 Rarely recognized (8–10%): C, Al

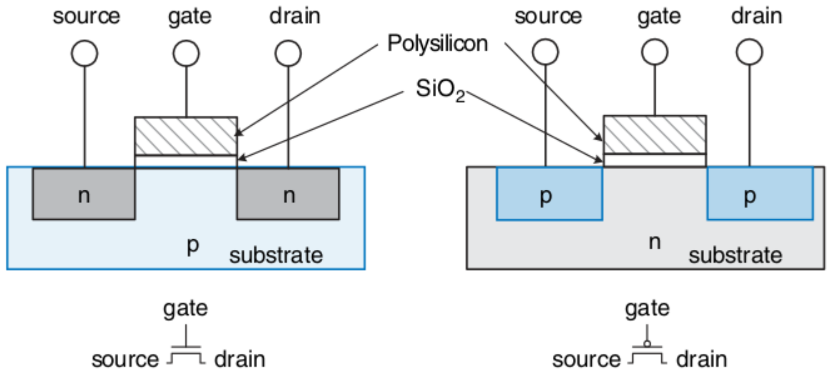
□ (All other elements cited in less than 6% of sources)

▙ Arbitrary metal-nonmetal dividing line: between Be and B, Al and Si, Ge and As, Sb and Te, Po and At

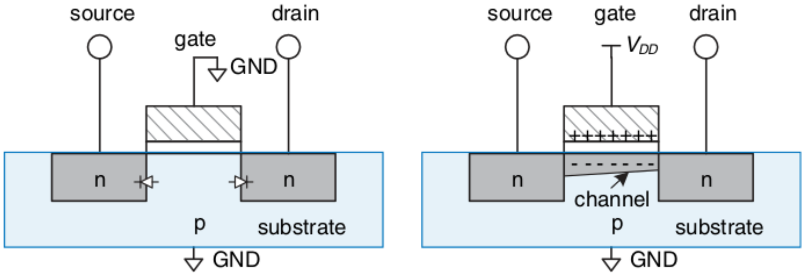


8.5 MOSFET: Metal Oxide Field Effect Transistors

Not in syllabus but good to know

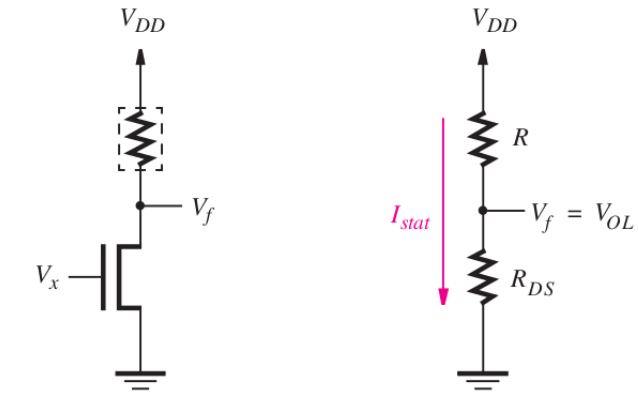


(a) nMOS **(b)** pMOS

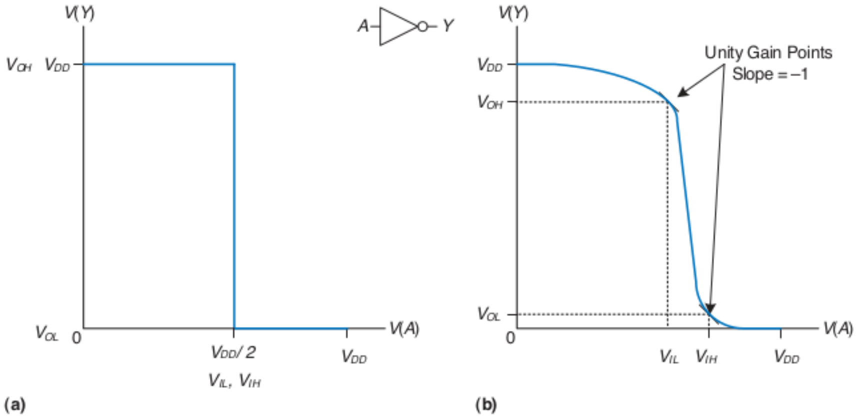


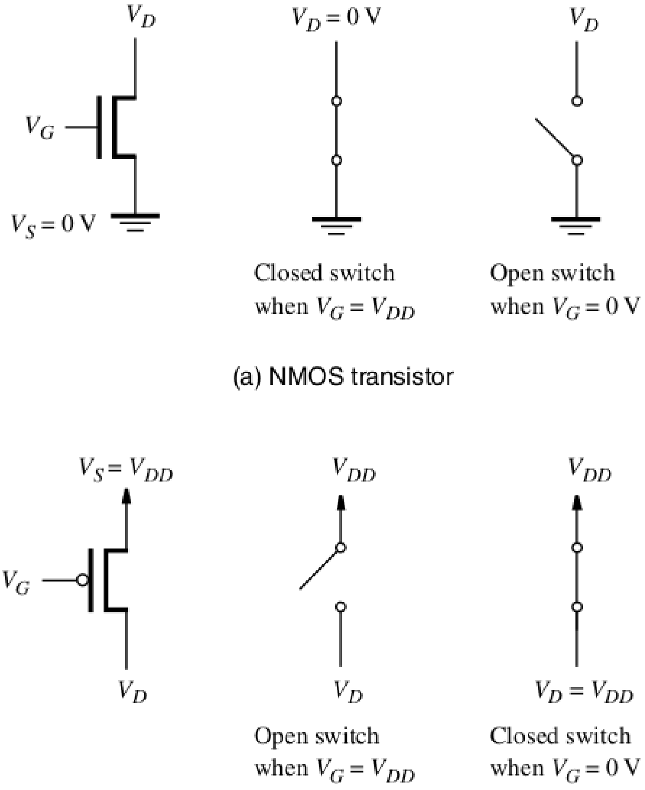
(a) **(b)**

8.6 DC Transfer characteristic



(a) NMOS NOT gate (b) $V_x = 5$ V





(a) NMOS transistor

(b) PMOS transistor

Example 60. *Draw a NOT gate using nMOS transistors.*

Example 61. *Draw a NOT gate using pMOS transistors.*

Remark 10. *nMOS transistors pass 0's well (output between 0 and $V_{DD} - V_t$). pMOS transistors pass 1's well (output between V_t and V_{DD}).*

Example 62. *Draw CMOS NOT Gate.*

Example 63. *Draw a two input CMOS NAND Gate*

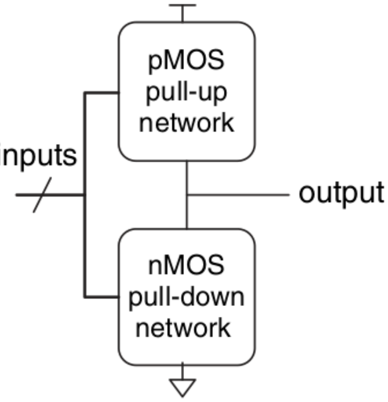
Definition 37 (Negative logic)**.**

Example 64. *Analyze the above circuit under negative logic.*

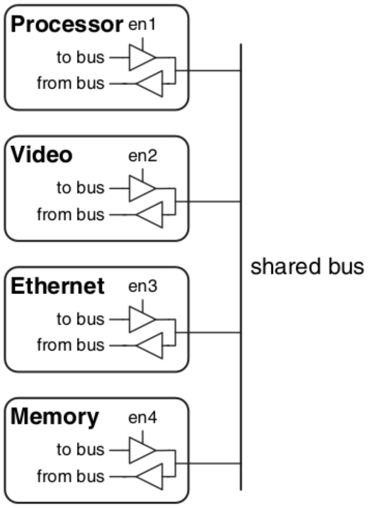
Example 65. *Draw a three input NAND using CMOS.*

Example 66. *Draw a three input NOR using CMOS.*

Example 67. *Draw a two input AND gate using CMOS.*



8.6.1 Gates with floating output



Definition 38 (Transmission gate)**.** *Draw a schematic of transmission gate and truth table for transmission gate. What is its commonly used symbol?*

Definition 39 (Tristate buffer)**.** *What is tristate buffer? Draw it's symbol and truth table? Where is it used?*

Example 68. *Draw a Multiplexer using transmission gates.*

Example 69. *Draw a Multiplexer using tristate buffers.*

Definition 40 (Totem-pole)**.** *Draw a Push-pull (or Totem-pole) output NAND gate using CMOS. Can you connect this gate to a shared bus?*

Definition 41 (Tristate)**.** *Draw a Tristate output NAND gate using CMOS with an output enable (OE) input. Can you connect this gate to a shared bus?*

Definition 42 (Open-collector)**.** *Draw a open-collector output NAND gate. Can you connect this gate to a shared bus?*

8.7 Verilog truth tables

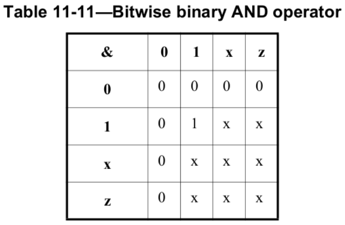


Table 11-11—Bitwise binary AND operator

&	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

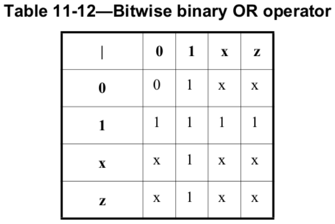


Table 11-12—Bitwise binary OR operator

|	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

Bibliography

[1] Sarah L Harris and David Harris. *Digital design and computer architecture.* Morgan Kaufmann, 2022.

[2] Randy Katz and Gaetano Barriello. *Contemporary Logic Design.* Prentice Hall, 2004.

[3] Brown Stephen and Vranesic Zvonko. *Fundamentals of digital Logic with Verilog design.* McGraw Hill, 2022.