Introduction to Digital Logic Design

Combinational Logic

Week 3

Outline

- Introduction to Digital Design
- Combinational Logic Design
- Sequential Logic Design
- Introduction to Verilog HDL

Introduction to Digital Design

Abstraction

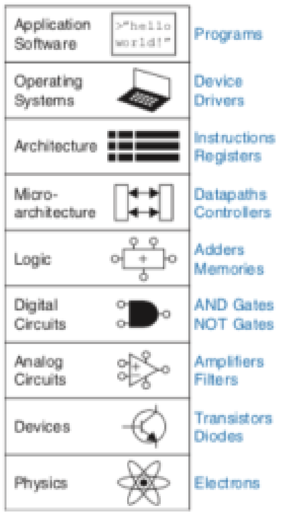
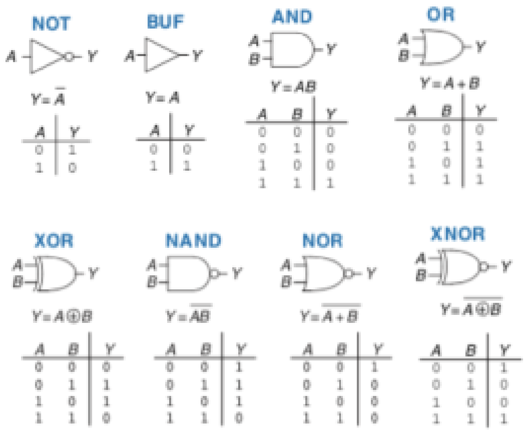


Figure 1.1 **Levels of abstraction for an electronic computing system**

Digital Circuits:
restrict the voltages to discrete ranges to build more complex structures, such as, adders, multipliers memories…

Logic Gates are digital systems that perform operations on these binary variables.



NOT

A	Y
0	1
1	0

BUF

A	Y
0	0
1	1

AND

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Table 1.3 **Range of *N*-bit numbers**

System	Range
Unsigned	$[0, 2^N - 1]$
Sign/Magnitude	$[-2^{N-1} + 1, 2^{N-1} - 1]$
Two's Complement	$[-2^{N-1}, 2^{N-1} - 1]$

The relationship between the inputs and the output can be described with a truth table or a Boolean equation.

Beneath the Abstraction

Supply Voltage:

V_{DD} has dropped from 5 V to 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V

Logic Levels:

Mapping of a continuous variable onto a discrete binary variable

Noise Margin:

The amount of noise that could be added to a worst-case output such that the signal can still be interpreted as a valid input.

choose $V_{OL} < V_{IL}$ and $V_{OH} > V_{IH}$.

Low and High Noise Margins

$$NM_L = V_{IL} - V_{OL}$$

$$NM_H = V_{OH} - V_{IH}$$

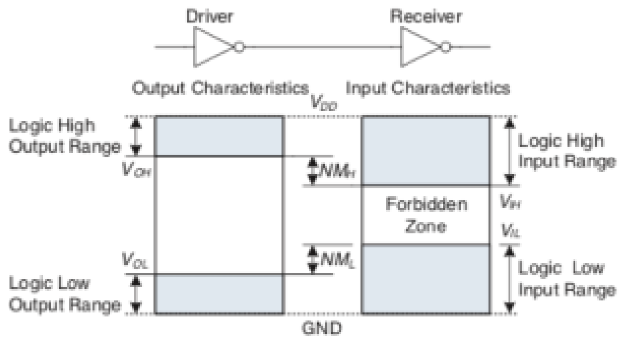
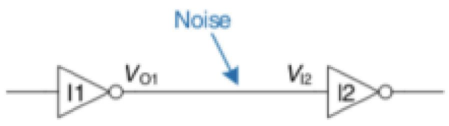


Figure 1.23 Logic levels and noise margins

Example

Consider the inverter circuit:



Both inverters have the following characteristics:

V_{DD}=5V, V_{IL}=1.35V, V_{IH} =3.15V, V_{OL} =0.33V, and V_{OH} =3.84V.

Can the circuit tolerate 1 V of noise between VO1 and VI2?

Example (Solution)

The inverter noise margins are:

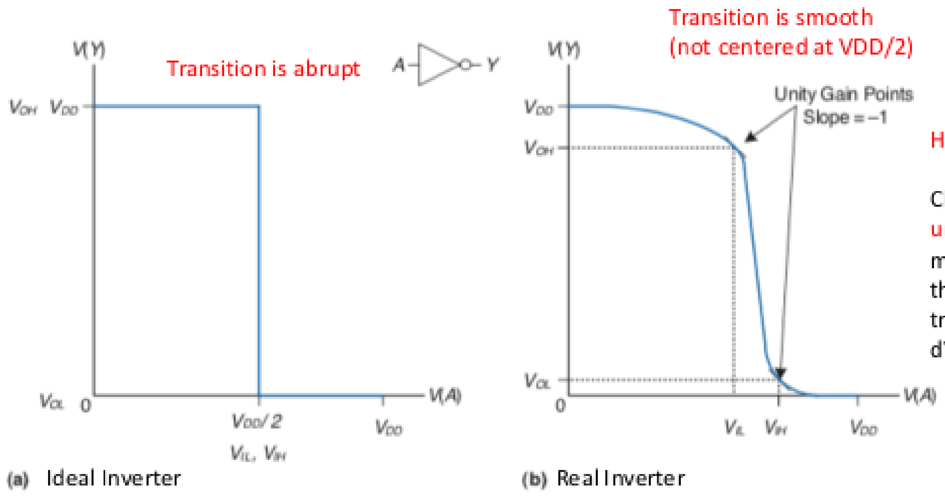
NM_L = VIL − VOL = (1.35 V − 0.33 V) = 1.02 V

NM_H = VOH − VIH = (3.84 V − 3.15 V) = 0.69 V.

The circuit can tolerate 1 V of noise when the output is LOW but not when the output is HIGH.

DC Transfer Characteristics

Relationship Between Input and Output Voltages



How to define the logic levels?

Choosing logic levels at the unity gain points usually maximizes the noise margins, that is where the slope of the transfer characteristic dV(Y)/dV(A) = −1

Figure 1.25 DC transfer characteristics and logic levels

Static Discipline

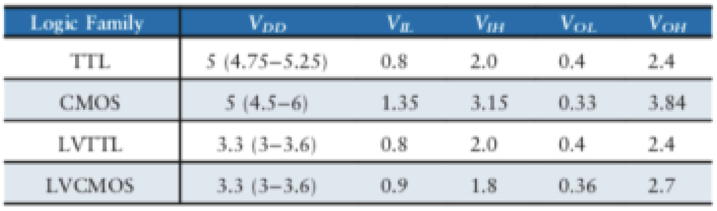
To avoid inputs falling into the forbidden zone, digital logic gates are designed to conform to the static discipline, that is, given logically valid inputs, every circuit element will produce logically valid outputs.

Tradeoff

Simplicity and robustness at the expense of freedom of using arbitrary analog circuit elements.

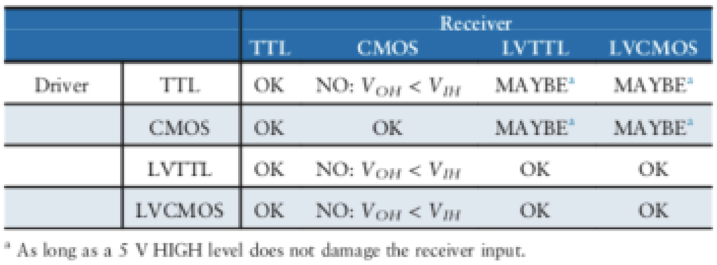
All gates that communicate must have compatible logic levels.

Gates are grouped into four major logic families



Logic Family	V_{DD}	V_{IL}	V_{IH}	V_{OL}	V_{OH}
TTL	5 (4.75–5.25)	0.8	2.0	0.4	2.4
CMOS	5 (4.5–6)	1.35	3.15	0.33	3.84
LVTTL	3.3 (3–3.6)	0.8	2.0	0.4	2.4
LVCMOS	3.3 (3–3.6)	0.9	1.8	0.36	2.7

Table 1.5 Compatibility of logic families



		Receiver			
		TTL	CMOS	LVTTL	LVCMOS
Driver	TTL	OK	NO: $V_{OH} < V_{IH}$	MAYBE[a]	MAYBE[a]
	CMOS	OK	OK	MAYBE[a]	MAYBE[a]
	LVTTL	OK	NO: $V_{OH} < V_{IH}$	OK	OK
	LVCMOS	OK	NO: $V_{OH} < V_{IH}$	OK	OK

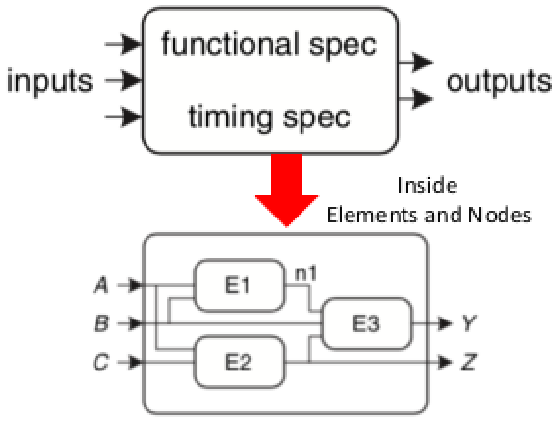
[a] As long as a 5 V HIGH level does not damage the receiver input.

Combinational Logic Design

Digital Circuit

A network that processes discrete-valued variables with:

- one or more input terminals
- one or more output terminals
- a functional specification describing the relationship between inputs and outputs
- a timing specification describing the delay between inputs changing and outputs responding.

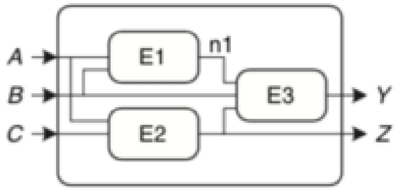


An element is a circuit with inputs, outputs, and a specification.

A node is a wire, whose voltage conveys a discrete-valued variable and are classified as input, output, or internal.

How Many Elements and Nodes?

- 3 elements E1, E2, E3
- 6 Nodes:
  - 3 input nodes: A, B, C
  - 2 output nodes: Y, Z
  - 1 internal: n1



Digital Circuits Classification

Combinational

Memoryless: Outputs depend only on the current values of the inputs, i.e. logic gates.

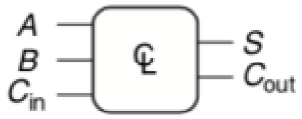
- Function Specification
- Timing Specification

Sequential

Has memory: Outputs depend on both current and previous values of the inputs or input sequence, i.e. registers.

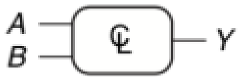
Examples

Full Adder



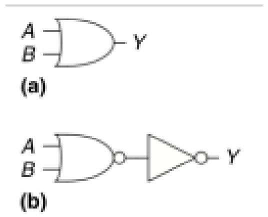
$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Design Constraints
Area
Speed
Power and
Design time



$$Y = F(A, B) = A + B$$

Multiple implementations

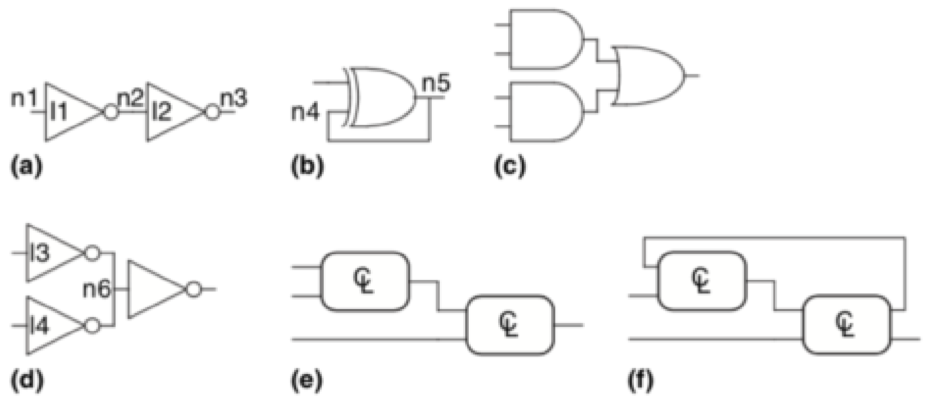


A circuit is combinational if

- Every circuit element is itself combinational.
- Every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element.
- The circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once.

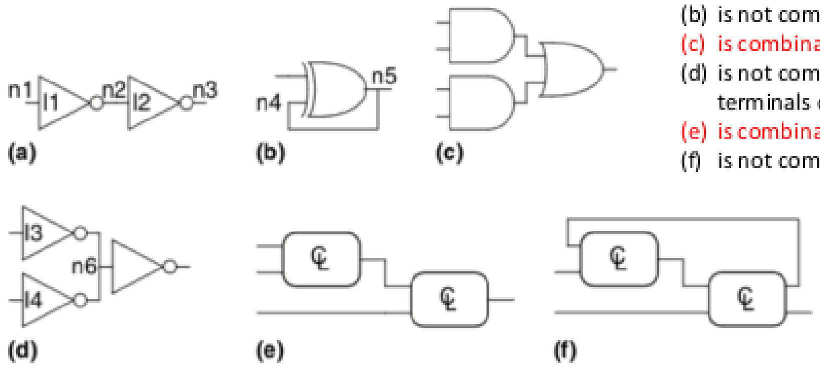
Example

Which of the following is combinational?



Example

Which of the following is combinational?



(a) is combinational.
(b) is not combinational, there is a cyclic path
(c) is combinational.
(d) is not combinational, node n6 connects to the output terminals of both I3 and I4.
(e) is combinational
(f) is not combinational, there is a cyclic path

Function Specification

Sum-of-Products Form

Product-of-Sums Form

Truth Table

Derive

A Boolean Equation

Implement

Logic Gates

Analyze

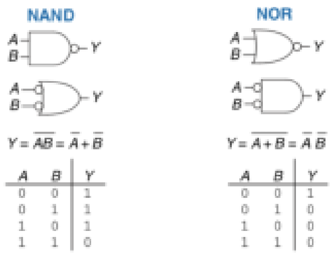
Speed

Simplify

Axioms and Theorems Tables 2.1, 2.2, 2.3, 2.4 2.5

Boolean algebra

Karnaugh maps



NAND: $Y = \overline{AB} = \overline{A} + \overline{B}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

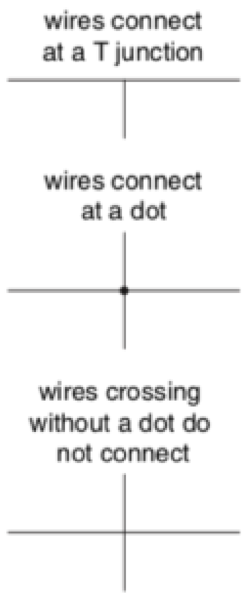
NOR: $Y = \overline{A+B} = \overline{A}\,\overline{B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 2.19 De Morgan equivalent gates

Schematics Guidelines

- Inputs are on the left (or top) side of a schematic.
- Outputs are on the right (or bottom) side of a schematic.
- Whenever possible, gates should flow from left to right.
- Straight wires are better to use than wires with multiple corners
- Wires always connect at a T junction.
- A dot where wires cross indicates a connection between the wires.
- Wires crossing without a dot make no connection.

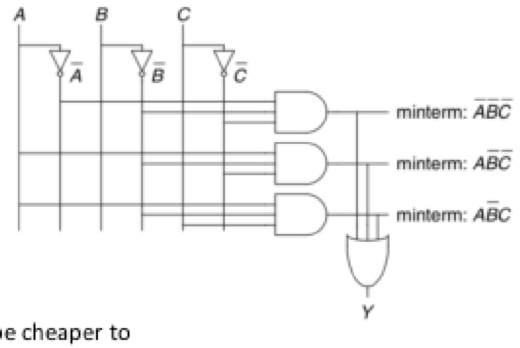

wires connect at a T junction
wires connect at a dot
wires crossing without a dot do not connect

Programmable Logic Array (PLA)

Sum-of-Products Form

The inverters, AND gates, and OR gates are arrayed in a systematic fashion.

$$Y = \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C} + A\overline{B}C$$

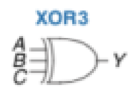

A
B
C
$\overline{A}$
$\overline{B}$
$\overline{C}$
minterm: $\overline{A}\overline{B}\overline{C}$
minterm: $A\overline{B}\overline{C}$
minterm: $A\overline{B}C$
Y

Note: Depending on the implementation technology, it may be cheaper to use the fewest gates or to use certain types of gates in preference to others. For example, NANDs and NORs are preferred over ANDs and ORs in CMOS implementations.

Multilevel Combinational Logic

Multilevel combinational circuits may use less hardware than their two-level counterparts.

Example- 3 input XOR



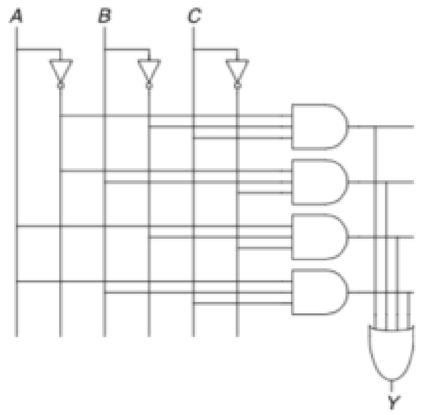
$Y = A \oplus B \oplus C$

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

TRUE output when an odd number of inputs are TRUE

$$Y = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$

How about an 8 input??

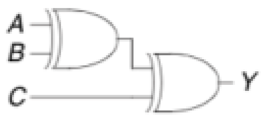


Hardware Reduction

Alternatively...

$$A \oplus B \oplus C = (A \oplus B) \oplus C$$

The three-input XOR can be built out of a cascade of two-input XORs



Selecting the best multilevel implementation of a specific logic function tradeoffs:

fewest gates

Fastest

shortest design time

least cost

least power consumption.

Bubble Pushing

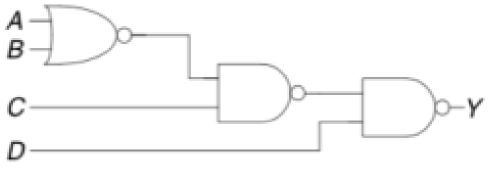
- CMOS circuits prefer NANDs and NORs over ANDs and ORs
- However, function is not immediately clear by inspection.
- Bubble pushing is a helpful way to redraw these circuits so that the bubbles cancel out and the function can be more easily determined

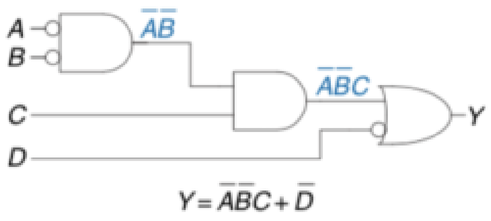
Guidelines

- Begin at the output of the circuit and work toward the inputs.
- Push any bubbles on the final output back toward the inputs so that you can read an equation in terms of the output
- Working backward, draw each gate in a form so that bubbles cancel.

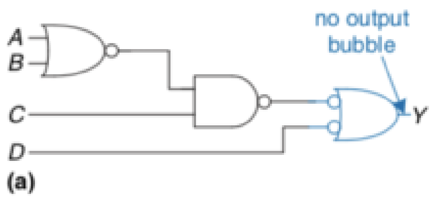
(If the current gate has an input bubble, draw the preceding gate with an output bubble. If the current gate does not have an input bubble, draw the preceding gate without an output bubble.)

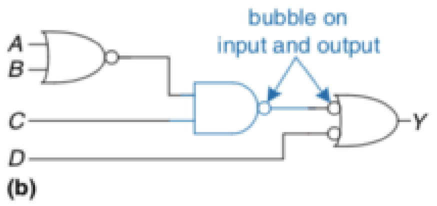
Example

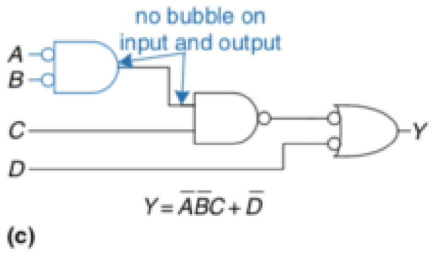




$Y = \overline{A}\,\overline{B}C + \overline{D}$







$Y = \overline{A}\,\overline{B}C + \overline{D}$

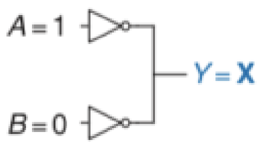
Illegal Value X

It is being driven to both 0 and 1 at the same time, a.k.a. contention

Voltage on such a node is often, but not always, in the forbidden zone

It can cause large amounts of power to flow between the fighting gates, resulting in the circuit getting hot and possibly damaged.

Symbol X indicates that the circuit node has an unknown or illegal value.



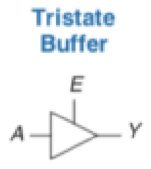
Floating Value Z

Node is Floating, high impedance, or high Z

a node is being driven neither HIGH nor LOW

Floating node might be 0, might be 1, or might be at some voltage in between, depending on the history of the system

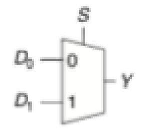
One common way to produce a floating node is to forget to connect a voltage to a circuit input

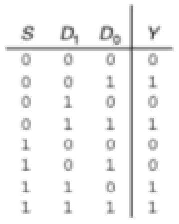


E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Combinational Building Blocks
(Abstraction)

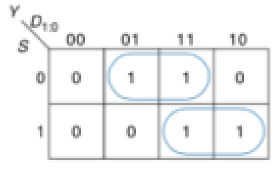
Multiplexers choose an output from among several possible inputs based on the value of a select (control) signal





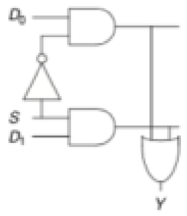
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Figure 2.54 **2:1 multiplexer symbol and truth table**

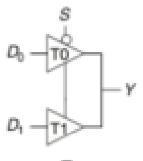


$S \backslash D_{1:0}$	00	01	11	10
0	0	1	1	0
1	0	0	1	1

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



Multiplexer using two-level logic

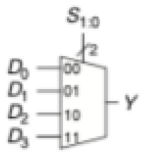


$Y = D_0\bar{S} + D_1 S$

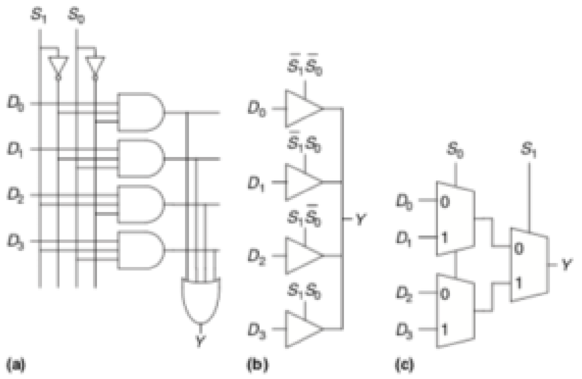
Using tristate buffers

Wider Multiplexer

4:1 multiplexer



N:1 multiplexer needs $\log_2 N$ select lines



(a) two-level logic (b) tristates (c) hierarchical

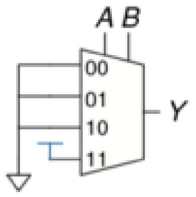
MUX Logic - Lookup Tables (LUTs)

Note: Multiplexers can be used as lookup tables to perform logic functions.

Example
4:1 multiplexer implementation of two-input AND function

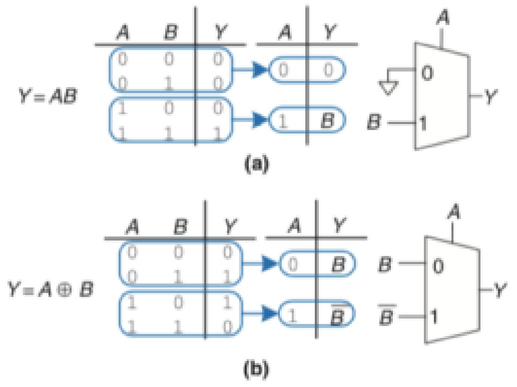
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$



- The inputs, A and B, serve as select lines.
- The multiplexer data inputs are connected to 0 or 1 according to the corresponding row of the truth table.
- In general, a 2^N input multiplexer can be programmed to perform any N-input logic function by applying 0's and 1's to the appropriate data inputs.
- By changing the data inputs, the multiplexer can be reprogrammed to perform a different function.

Using only a 2^{N-1}-input multiplexer we can perform any N-input logic function cutting the multiplexer's size in half.



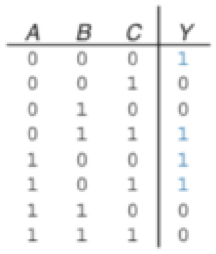
Example

Implement the function

$$Y = A\overline{B} + \overline{B}\,\overline{C} + \overline{A}BC$$

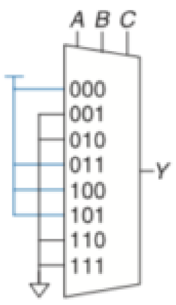
a. With an 8:1 multiplexer
b. With a 4:1 multiplexer and an inverter

Example

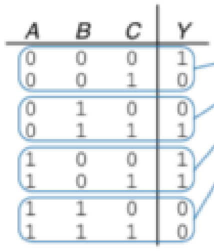


A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC$
(a)

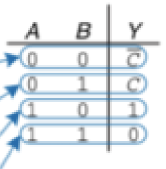


(b)



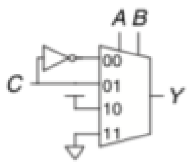
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

(a)



A	B	Y
0	0	$\overline{C}$
0	1	C
1	0	1
1	1	0

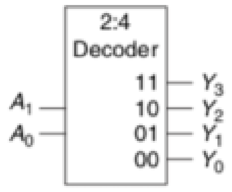
(b)



(c)

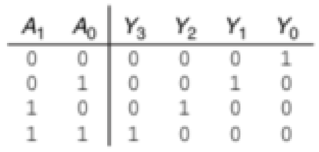
Decoders

A decoder has N inputs and 2N outputs. It asserts exactly one of its outputs depending on the input combination.

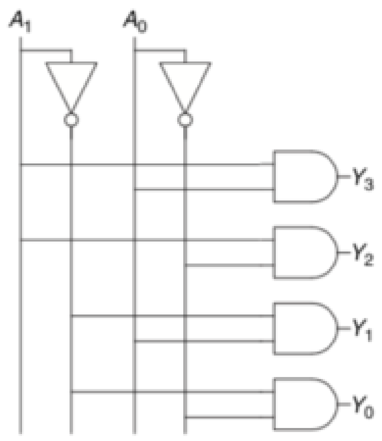


Exercise

Implement a 2:4 decoder with AND, OR, and NOT gates.



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

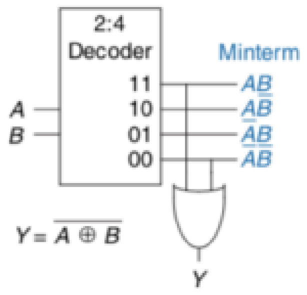


- An $N:2^N$ decoder can be constructed from 2^N N-input AND gates that accept the various combinations of true or complementary inputs.
- Each output in a decoder represents a single minterm. i.e. Y3 represents the minterm A1A0

Decoder Logic

- Decoders can be combined with OR gates to build logic functions

Example- XNOR



- When using decoders to build logic, it is easiest to express functions as a truth table or in canonical sum-of-products form.
- In general, an N-input function with M 1's in the truth table can be built with an N:2^N decoder and an M-input OR gate attached to all of the minterms containing 1's in the truth table.
- This is used to build Read Only Memories (ROMs)

Timing

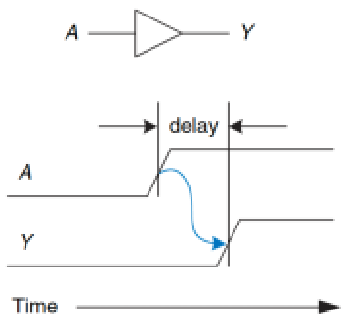
- Propagation and Contamination Delay
- Glitches

Delay

An output takes time to change in response to an input change.

- measured from the 50% point of the input signal, A , to the 50% point of the output signal, Y

Consider the timing diagram showing the transient response of a buffer



Propagation and Contamination

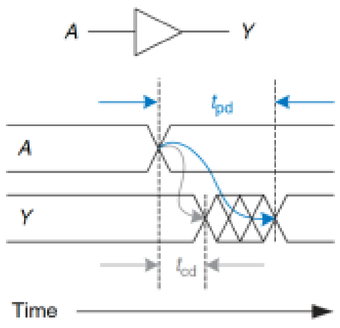
Combinational logic is characterized by:

Propagation delay (t_{pd})

The maximum time from when an input changes until the output or outputs reach their final value.

Contamination delay (t_{cd})

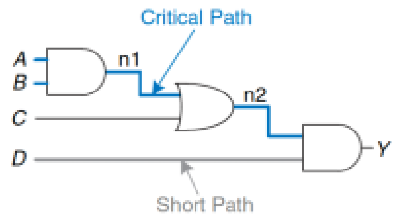
The minimum time from when an input changes until any output starts to change its value.

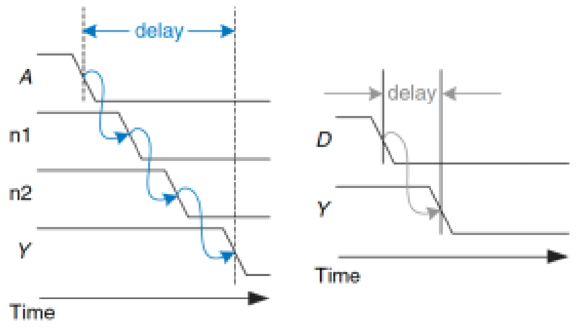


Delay Determination

Propagation and contamination delays are determined by the path a signal takes from input to output.

- The propagation delay of a combinational circuit is the sum of the propagation delays through each element on the critical path.
- The contamination delay is the sum of the contamination delays through each element on the short path.





$$t_{pd} = 2t_{pd\_\text{AND}} + t_{pd\_\text{OR}}$$

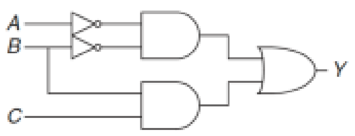
$$t_{cd} = t_{cd\_\text{AND}}$$

Glitches

It is possible that a single input transition can cause multiple output transitions.

Example

When A = 0, C = 1, and B transitions from 1 to 0



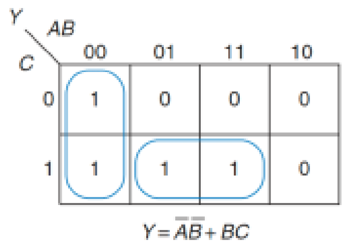
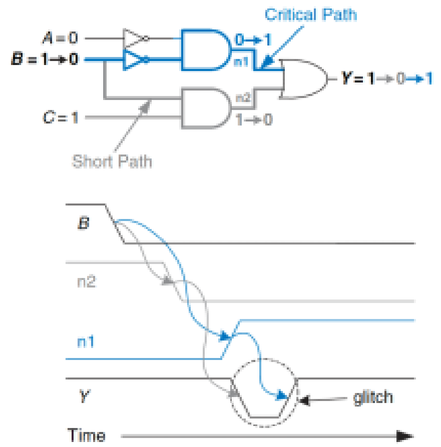


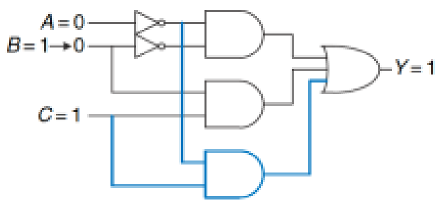
Figure 2.75 Circuit with a glitch

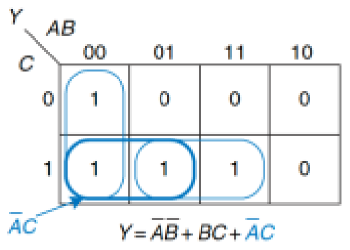
- As B transitions from 1 to 0, n2 falls before n1 can rise.
- Until n1 rises, the two inputs to the OR gate are 0, and the output Y drops to 0
- When n1 eventually rises, Y returns to 1



Y starts at 1 and ends at 1
but momentarily glitches to 0

Without a Glitch





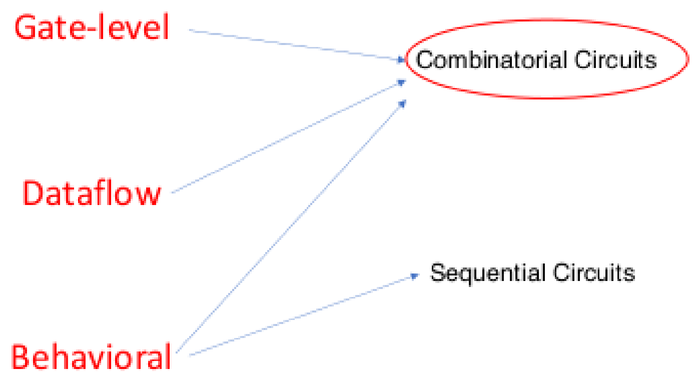
Introduction to Verilog HDL

Documentations and Designs

https://www.xilinx.com/support/university/boards-portfolio/xup-boards/DigilentZedBoard.html#designs

Verilog HDL

Three kinds of modeling styles



Objectives

- Create scalar and wide combinatorial circuits using gate-level, dataflow, and behavioral modeling
- Write models to read switches and output on LEDs
- Simulate and understand the design output
- Create hierarchical designs
- Synthesize, implement and generate bitstreams
- Download bitstreams into the board and verify functionality

Gate-Level Modeling

Built-in primitive gates modeling are supported

Multiple-input (one output)

and | **nand** | **or** | **nor** | **xor** | **xnor** [instance name] (out, in1, ..., inN);

Multiple-output (one input)

buf | **not** [instance name] (out1, out2, ..., out2, input);

Tristate (one input, one control signal, one output)

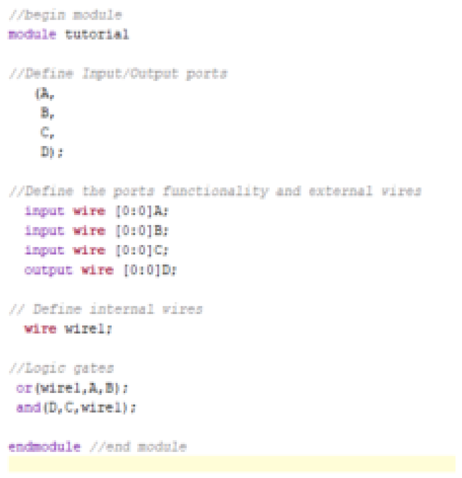
bufif0 | **bufif1** | **notif0** | **notif1** [instance name] (outputA, inputB, controlC);

Pull gates (single output, no input)

pullup | **pulldown** [instance name] (output A);

// [] is optional and | is selection

Writing a Module



```
//begin module
module tutorial

//Define Input/Output ports
    (A,
     B,
     C,
     D);

//Define the ports functionality and external wires
  input wire [0:0]A;
  input wire [0:0]B;
  input wire [0:0]C;
  output wire [0:0]D;

// Define internal wires
  wire wire1;

//Logic gates
 or(wire1,A,B);
 and(D,C,wire1);

endmodule //end module
```

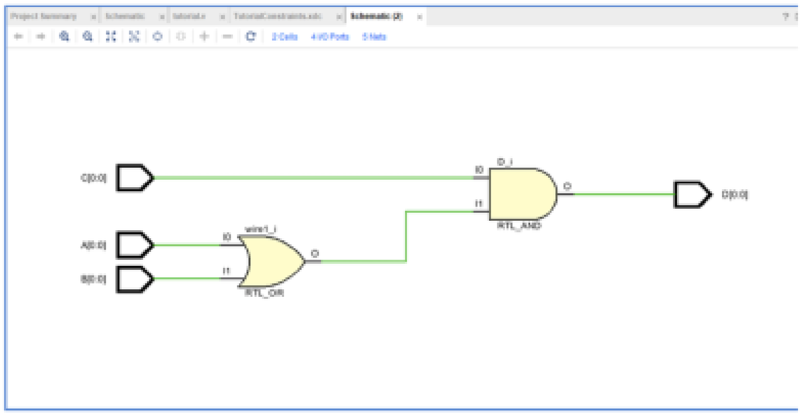
Step 1- Creating Verilog Code

- Write Verilog source file describing the combinational function you implemented last week.

- Perform RTL analysis on the source file.
- I/O Planning
- Simulate the Design Using a Testbench
- Run Simulations
- Synthesize the Design
- Implement the Design
- Timing Simulations
- Generate Bitstream and Verify Functionality

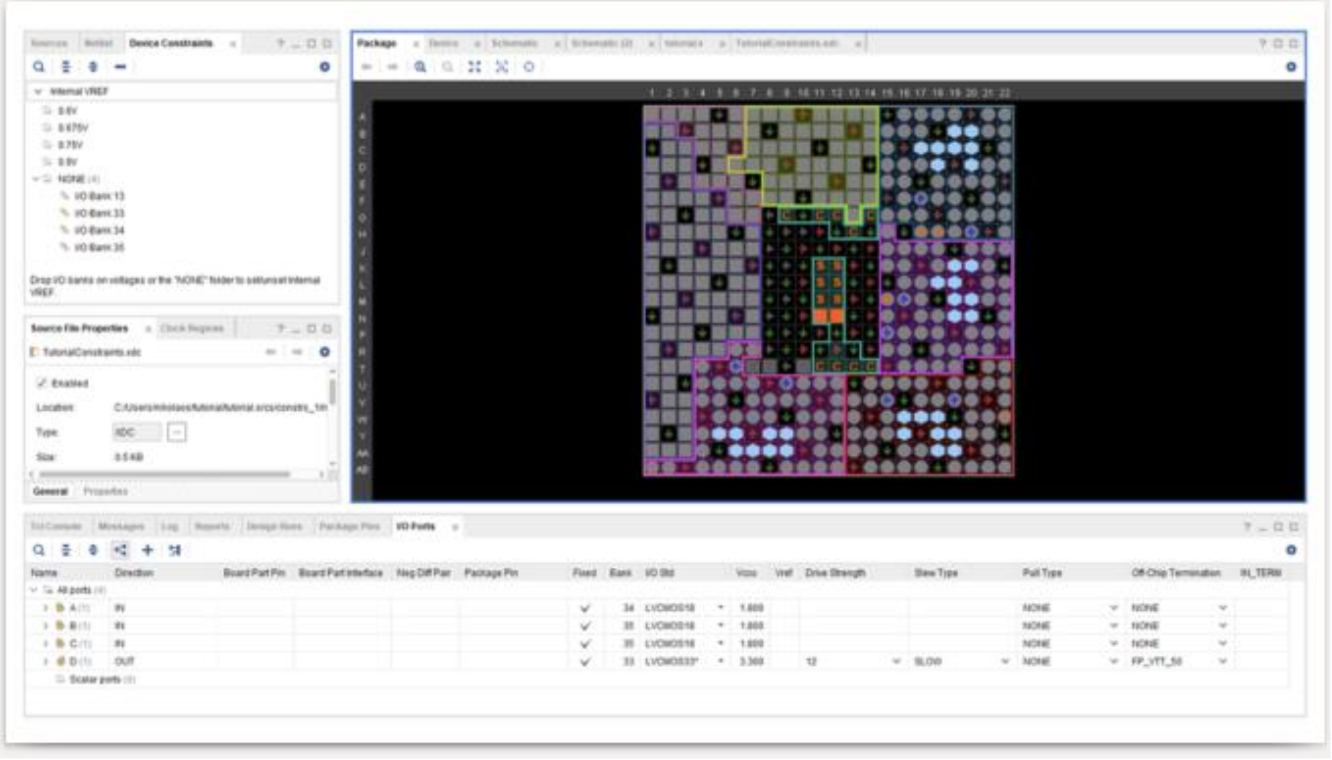
Perform RTL analysis on the source file.

- Expand the *Open Elaborated Design* entry under the *RTL Analysis* tasks of the *Flow Navigator* pane and click on **Schematic**
- Click **OK** to run the analysis. The model (design) will be elaborated and a logic view of the design is displayed.



I/O Planning

- Click on the drop-down button, Layout, and select the *I/O Planning* layout
- Notice that the Package view is displayed in the Auxiliary View area, RTL Netlist tab is selected, and I/O ports tab is displayed in the Console View area. Also notice that design ports are listed in the I/O Ports tab and have multiple I/O standards.
- Move the mouse cursor over the Package view, highlighting different pins. Notice the pin site number is shown at the bottom of the Vivado GUI, along with the pin type (User IO, GND, VCCO...) and the I/O bank it belongs to.
- You can expand A, B, C, and D ports by clicking on the + box and observe that the used ports have assigned pins and use LVCMOS33 or LVCMOS18 I/O standard.



Drop I/O banks on voltages or the "NONE" folder to set/unset Internal VREF.

✓ Enabled

Location: C:/Users/nikolaos/tutorial/tutorial.srcs/constrs_1/n

Type: XDC

Size: 0.5 KB

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (4)															
A (1)	IN					✓	34	LVCMOS18	1.800				NONE	NONE	
B (1)	IN					✓	35	LVCMOS18	1.800				NONE	NONE	
C (1)	IN					✓	35	LVCMOS18	1.800				NONE	NONE	
D (1)	OUT					✓	33	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50	
Scalar ports (0)															

Pin Assignment

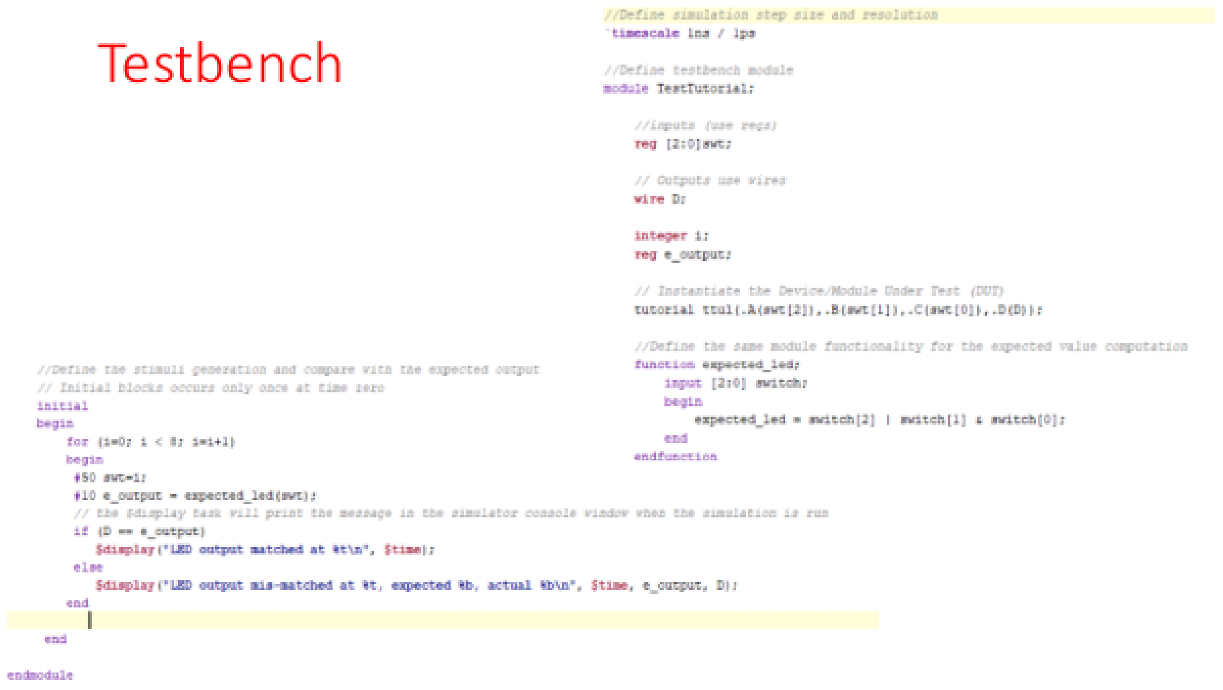
- The ports are already assigned to pins. If you would like to assign pins in this view, you would click under the *Package Pin* column across the desired port row to bring up a drop-down box. Type in the appropriate **Port Variable** to jump to the pins with that variable. Scroll-down until you see the correct port name, and then select it and hit the *Enter* key to assign the pin.
- You can also assign the pin by selecting its entry in the I/O ports tab, and dragging it to the Package view, and placing it at the desired location
- You can also assign the I/O standard by selecting its entry in the I/O Ports tab, selecting the Configure tab of the I/O Port Properties window, followed by clicking the drop-down button of the I/O standard field, selecting LVCMOS33.
- You can also assign the pin constraints and I/O standards using tcl commands, by typing the command in the Tcl Console tab to assign the R13 pin location and the LVCSMOS33 I/O like shown below and hitting the Enter key after the command.

```
set_property -dict { PACKAGE_PIN w22 IOSTANDARD LVCMOS33 } [get_ports D];
```

Simulate the Design Using a Testbench

- Click **Add Sources** under the *Project Manager* tasks of the *Flow Navigator* pane.
- Select the *Add or Create Simulation Sources* option and click **Next**.
- Select Create File name the file indicating that it is a test file then click OK then Finish.
- Select the *Sources* tab and expand the *Simulation Sources* group.
  The newly created test file is added under the *Simulation Sources* group.
- Verify that the **sim_1** directory is created at the same level as constrs_1 and sources_1 directories under the <name>.srcs directory, and that a copy of *<testFile>.v* is placed under **tutorial.srcs > sim_1 > sources**.
- Double-click on the **<testFile>.v** in the *Sources* pane to view its contents.

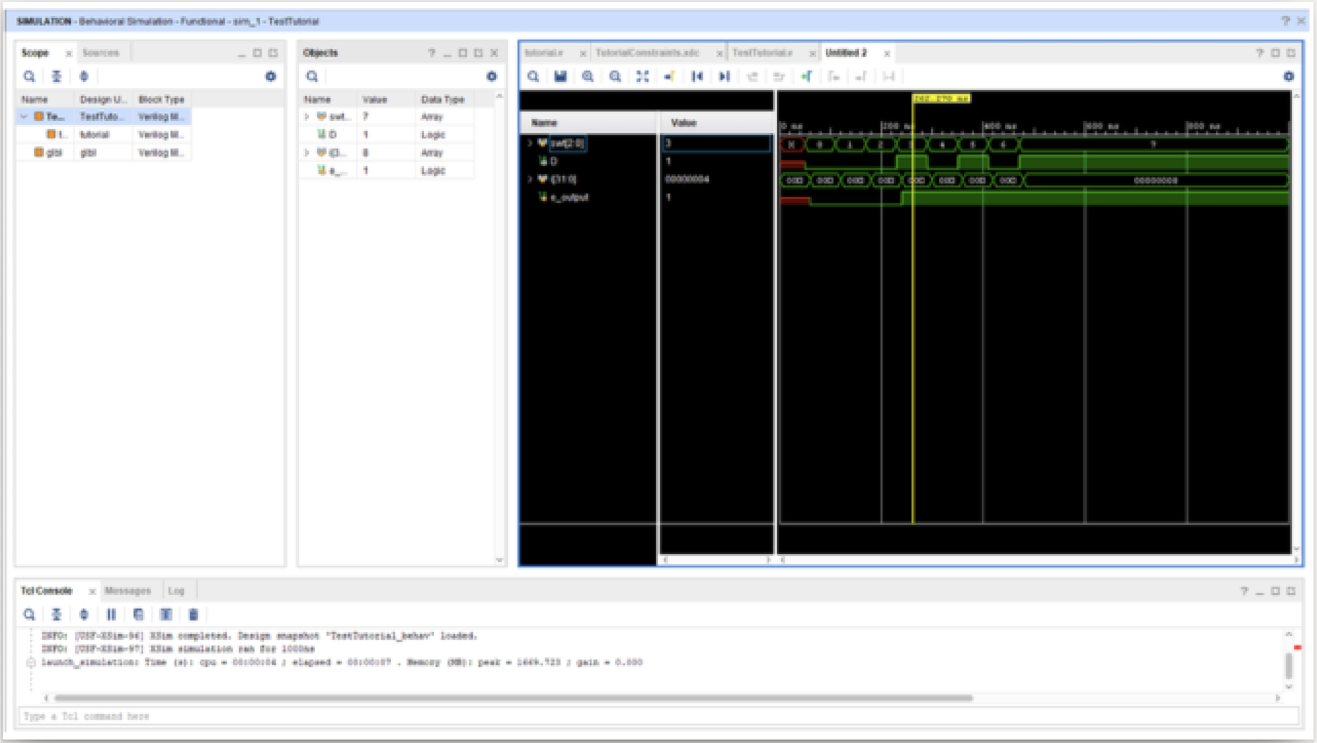
Testbench



```
//Define simulation step size and resolution
`timescale 1ns / 1ps

//Define testbench module
module TestTutorial;

    //inputs (use regs)
    reg [2:0]swt;

    // Outputs use wires
    wire D;

    integer i;
    reg e_output;

    // Instantiate the Device/Module Under Test (DUT)
    tutorial ttu1(.A(swt[2]),.B(swt[1]),.C(swt[0]),.D(D));

    //Define the same module functionality for the expected value computation
    function expected_led;
        input [2:0] switch;
        begin
            expected_led = switch[2] | switch[1] & switch[0];
        end
    endfunction

    //Define the stimuli generation and compare with the expected output
    // Initial blocks occurs only once at time zero
    initial
    begin
        for (i=0; i < 8; i=i+1)
        begin
         #50 swt=i;
         #10 e_output = expected_led(swt);
         // the $display task will print the message in the simulator console window when the simulation is run
         if (D == e_output)
            $display("LED output matched at %t\n", $time);
         else
            $display("LED output mis-matched at %t, expected %b, actual %b\n", $time, e_output, D);
        end

      end

endmodule
```

Run Simulations

- Select **Simulation Settings** under the *Project Manager* tasks of the *Flow Navigator* pane.
- A **Project Settings** form will appear showing the **Simulation** properties form.
- Select the **Simulation** tab, and set the **Simulation Run Time** value to 1000 ns and click **OK**.
- Click on **Run Simulation > Run Behavioral Simulation** under the *Flow Navigator* pane.
- You will see four main views: (i) *Scopes*, where the testbench hierarchy as well as glbl instances are displayed, (ii) *Objects*, where top-level signals are displayed, (iii) the waveform window, and (iv) *Tcl Console* where the simulation activities are displayed. Notice that since the testbench used is self-checking, the results are displayed as the simulation is run.
- Click on the *Zoom Fit* button located left of the waveform window to see the entire waveform.

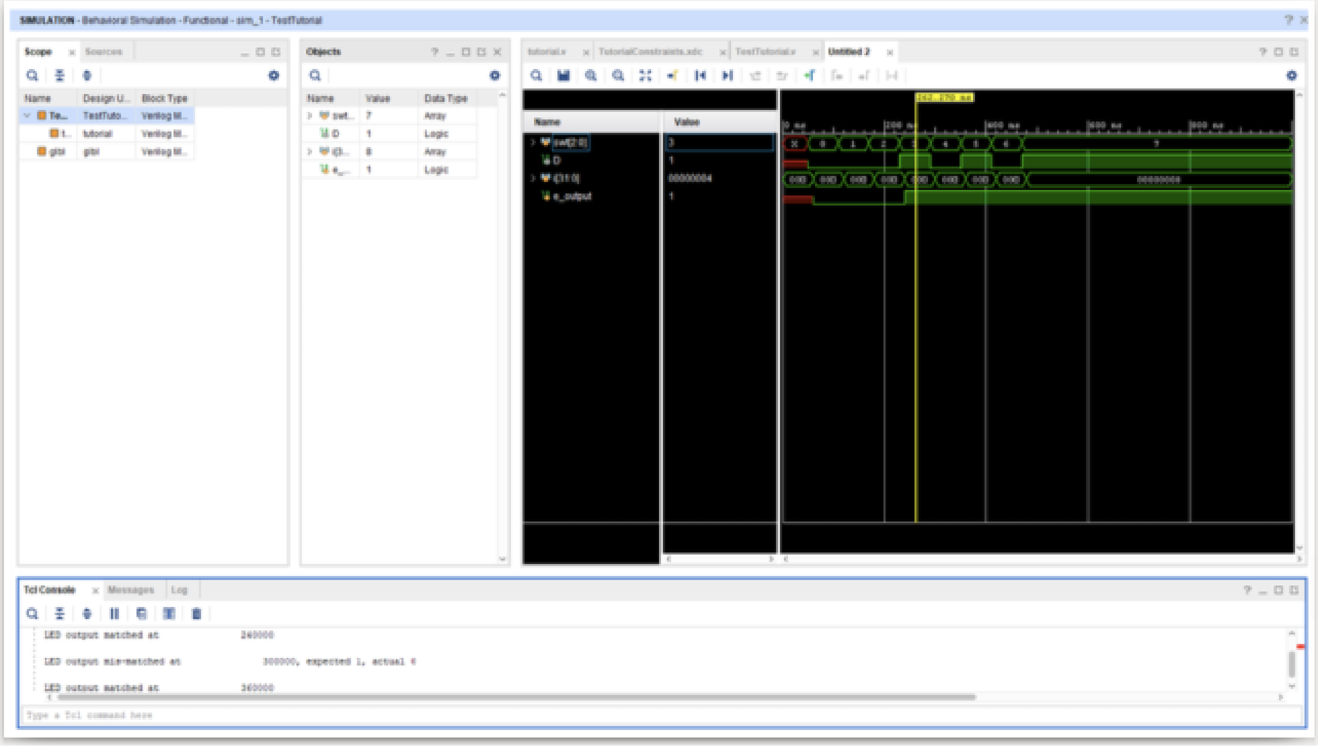


SIMULATION - Behavioral Simulation - Functional - sim_1 - TestTutorial

Name	Value	Data Type
swt...	7	Array
D	1	Logic
i[3]...	8	Array
e_...	1	Logic

Name	Value
swt[2:0]	3
D	1
i[31:0]	00000004
e_output	1

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'TestTutorial_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:07 . Memory (MB): peak = 1669.723 ; gain = 0.000
```

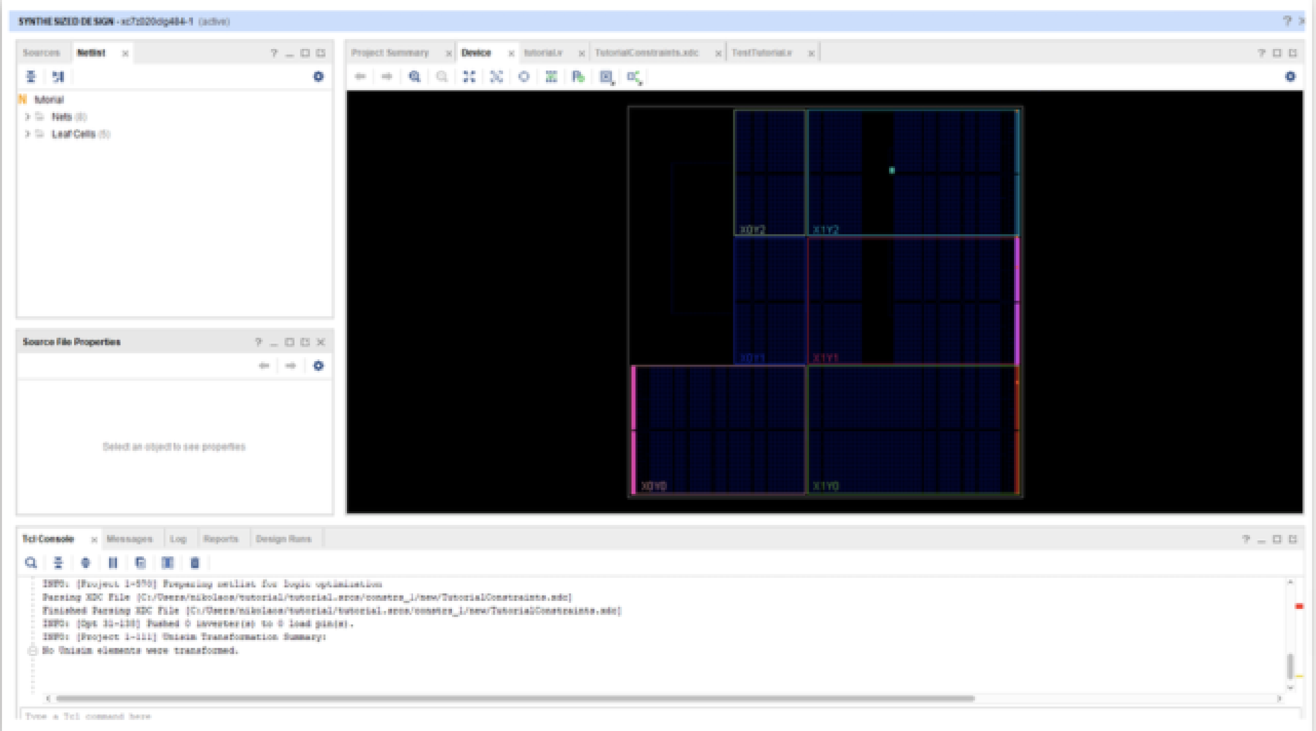


SIMULATION - Behavioral Simulation - Functional - sim_1 - TestTutorial

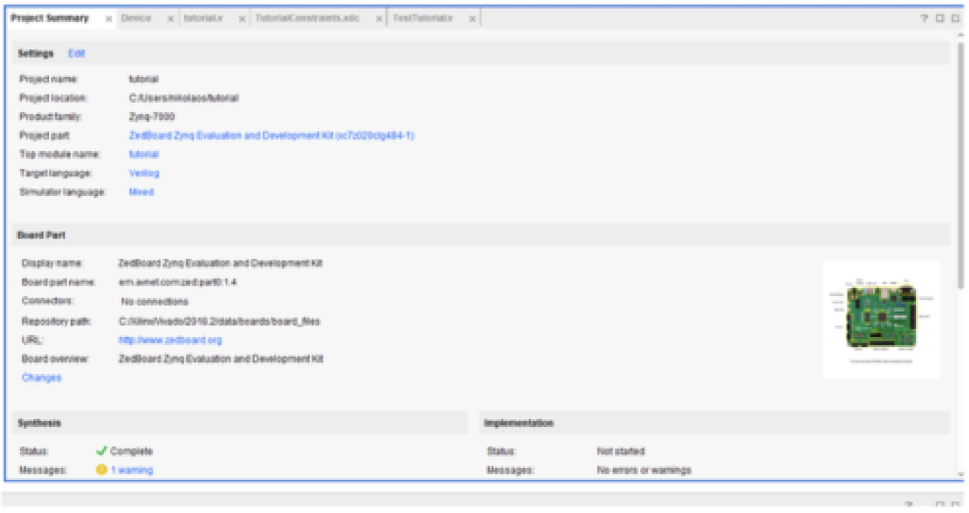
```
LED output matched at            240000

LED output mis-matched at             300000, expected 1, actual 0

LED output matched at            360000
```

Synthesize the Design

- Click on **Run Synthesis** under the *Synthesis* tasks of the *Flow Navigator* pane. When the process is completed a *Synthesis Completed* dialog box with three options will be displayed.
- Select the *Open Synthesized Design* option and click **OK** as we want to look at the synthesis output before progressing to the implementation stage. Click **Yes** to close the elaborated design if the dialog box is displayed.
- Select the **Project Summary** tab (Select default layout if the tab is not visible) and understand the various windows.
- Click on the **Table** tab in the **Project Summary** tab. Notice the estimated LUTs and IOs that are used.
- Click on **Schematic** under the *Open Synthesized Design* tasks of *Synthesis* tasks of the *Flow Navigator* pane to view the synthesized design in a schematic view.
- Notice that IBUF and OBUF are automatically instantiated (added) to the design as the input and output are buffered. The logical gates are implemented in LUTs. Notice how many LUTs and what LUTs are used for what gates and IOs



```
INFO: [Project 1-570] Preparing netlist for logic optimization
Parsing XDC File [C:/Users/nikolaos/tutorial/tutorial.srcs/constrs_1/new/TutorialConstraints.xdc]
Finished Parsing XDC File [C:/Users/nikolaos/tutorial/tutorial.srcs/constrs_1/new/TutorialConstraints.xdc]
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
No Unisim elements were transformed.
```



Settings Edit

Project name:	tutorial
Project location:	C:/Users/nikolaos/tutorial
Product family:	Zynq-7000
Project part:	ZedBoard Zynq Evaluation and Development Kit (xc7z020clg484-1)
Top module name:	tutorial
Target language:	Verilog
Simulator language:	Mixed

Board Part

Display name:	ZedBoard Zynq Evaluation and Development Kit
Board part name:	em.avnet.com:zed:part0:1.4
Connectors:	No connections
Repository path:	C:/Xilinx/Vivado/2018.2/data/boards/board_files
URL:	http://www.zedboard.org
Board overview:	ZedBoard Zynq Evaluation and Development Kit

Changes

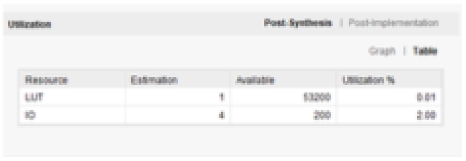
Synthesis

Status:	✓ Complete
Messages:	● 1 warning

Implementation

Status:	Not started
Messages:	No errors or warnings

DRC Violations
Run Implementation to see DRC results
Timing
Run Implementation to see timing results
Utilization
Post-Synthesis | Post-Implementation
Graph | Table
LUT
IO
0
25
50
75
100
Estimated Utilization (%)
Power
Run Implementation to see power results



Resource	Estimation	Available	Utilization %
LUT	1	53200	0.01
IO	4	200	2.00

Implement the Design

- Click on **Run Implementation** under the *Implementation* tasks of the *Flow Navigator* pane.
- Select **Open implemented design** and click **OK** as we want to look at the implemented design in a Device view tab.
- Click **Yes** to close the synthesized design. The implemented design will be opened.
- In the *Netlist* pane, select one of the nets (e.g. A_IBUF(1)) and notice that the net displayed in the X1Y1 clock region in the Device view tab (you may have to zoom in to see it).
- Close the implemented design view by selecting **File > Close Implemented Design**, and select the **Project Summary** tab (you may have to change to the Default Layout view) and observe the results.
- Select the *Report Utilization* entry under the *Synthesized Design* under the Flow Navigator. The report will be displayed in the auxiliary view pane showing resources utilization. Note that since the design is combinatorial no registers are used.

Timing Simulations

- Select **Run Simulation > Run Post-Implementation Timing Simulation** process under the *Simulation* tasks of the *Flow Navigator* pane.
- Click on the **Zoom Fit** button to see the waveform window.
- Right-click where the switch input is set to 0000000b and select **Markers > Add Marker**. Click on the **Add Marker** button and left-click at where the expected output changes. Does the time between the two match the delay set in the testbench?
- Close the simulator by selecting **File > Close Simulation** without saving any changes.

Generate Bitstream and Verify Functionality