PUE 3141: ADVANCED DIGITAL SYSTEMS DESIGN

LECTURE 2

13/07/2022

BY DR. MUTUGI KIRUKI

Topics:

i) Combinational & Sequential Logic Designs with Verilog HDL

References:

i) Mano, M.M., & Ciletti, M.D. (2007) *Digital Design* (4th Edition). Pearson.

Introduction to Verilog HDL

Hardware Description Language (HDL): Computer-based language that describes the hardware of digital systems in a textual form. It describes hardware structures and the behavior of logic circuits. It can be used to represent:

- Logic diagrams
- Boolean expressions
- Truth tables
- Complex abstractions of the behavior of a digital system

HDLs are used in various major steps in digital system design flow such as:

- Design entry
- Logic Synthesis
- Logic Simulation
- Timing verification & fault simulation

Design entry:

Creates an HDL-based description of the functionality that is to be implemented in hardware. This can be in various forms such as: Boolean logic equations, truth tables, a netlist of interconnected gates, or an abstract behavioral model.

Logic simulation:

Simulation of a circuit predicts how the hardware will behave before it is actually fabricated. This allows the detection of functional errors in a design without having to physically create and operate the circuit. Errors detected in the simulation can be corrected by modifying the appropriate HDL statements. ***Test bench*** is the stimulus (i.e., the logic values of the inputs to a circuit) that tests the functionality of the design.

Logic synthesis:

This is the process of deriving a list of physical components and their interconnections (netlist) from the model of a digital system described in an HDL. This netlist can then be used to fabricate an IC or to lay out a PCB with the hardware counterparts of the gates in the list. Logic synthesis is similar to compiling a program in a conventional high-level language. However, instead of producing an object code, logic synthesis produces a database describing the elements and structure of a circuit. The database specifies how to fabricate a physical IC that implements in silicon the functionality described by statements made in an HDL.

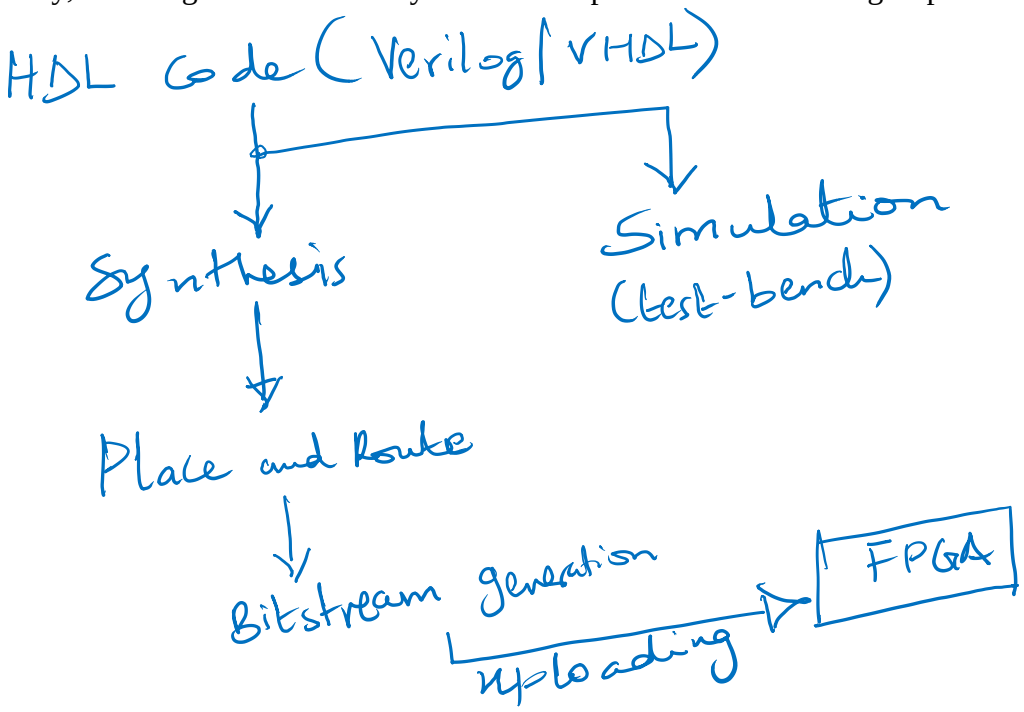
Timing verification:

Each logic gate in a circuit has a *propagation delay*, hence a signal transition at the input of a circuit cannot immediately cause a change in the logic value of the output of a circuit. Propagation delays ultimately limit the speed at which a circuit can operate. Timing verification checks each signal path to verify that it is not compromised by propagation delay.

Two standard HDLs supported by IEEE are:

- Very High Speed Integrated Circuit (VHSIC) HDL i.e., VHDL
- Verilog

Generally, the design flow in CAD systems is comprised of the following steps:



Verilog Syntax

- Verilog is case sensitive. All Verilog keywords are lower-case
- **Number syntax**: **<size>'<radix><value>;**
    - Unsized numbers are stored as 32-bit e.g., 1 is stored as 32-bit
    - Radix include binary, octal, decimal, hexadecimal
    - Radix and hex digits (a,b,c,d,e,f) are case insensitive
    - When <size> is smaller than <value>, then leftmost bits of <value> are truncated

    E.g.: 8'hBA, 7'b1001101, 'hFC

- **Data Types**:
  Verilog has two primary data types:
  - Nets: represent structural connections between components. E.g.: **wire**
  - Registers: represent variables used to store data.
    - ✓ Registers store the last value assigned to them until another assignment statement changes their value. They represent data storage constructs.
    - ✓ Register Data Types:

  reg – unsigned variable
  integer – signed variable (32 bits)
  time – unsigned integer (64 bits)
  real – Double precision floating point variable

- **Verilog Operators:**
  - Arithmetic Operators - Binary: +, -, *, /, %
  - Relational operators - <, >, <=, >=. The result is 0 if the relation is false and 1 if the relation is true
  - Logical operators - !: logic negation, &&: logical and, ||: logical or
  - Bit-wise operators - ~: negation, &: and, |: inclusive or, ^: exclusive or

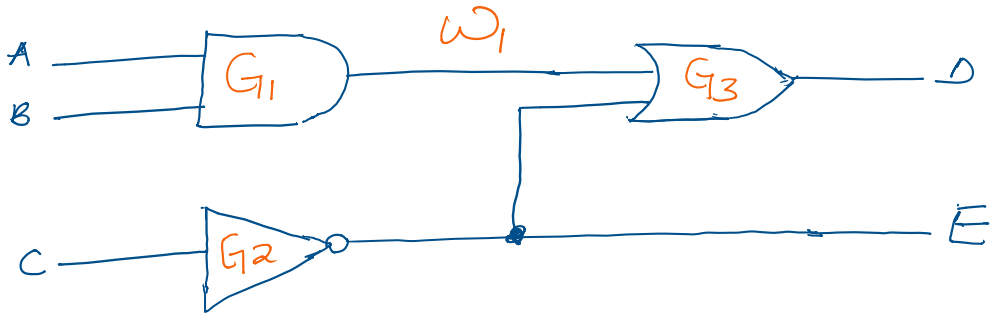
- **Modules:**
  A module is the fundamental descriptive unit in the Verilog language. It is declared by the keyword **module** and must always be terminated by the keyword **endmodule**.

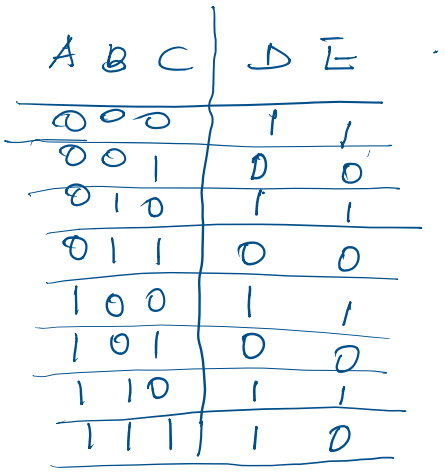
Combinational Logic

- Can be described by a schematic connection of gates, a set of Boolean equations, or by a truth table

Example 1_A:

- Using Icarus Verilog for HDL simulation.
- Logic described by logic diagram/schematic





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

module
inputs → A, B, C
outputs → D, E
wire → w1

endmodule

Top Module

```
module example_one (A, B, C, D, E);
//Declare inputs, outputs and internal connections
input A, B, C;
output D, E;
wire w1;

//Implementation of the circuit / schematic
and G1(w1, A, B); //Instantiate the and module
not G2(E, C);
or G3(D, w1, E);

endmodule
```

Test Bench

```
`timescale 1ns/1ns

`include "example_one.v"

module example_one_tb;
reg A, B, C; //inputs to simulate
wire D, E;   //outputs to simulate

//Instantiate the module that we are simulating
example_one  tb_one (A, B, C, D, E); //Retain the same
order

//simulate the circuit using the initial block
initial
    begin
    $dumpfile("example_one_tb.vcd");
    $dumpvars(0,example_one_tb);
    A = 1'b0; B = 1'b0; C = 1'b0;
    #30 A = 1'b1;
    #30 A = 1'b0; B = 1'b1; C = 1'b0;
    #60 A = 1'b1; B = 1'b1; C = 1'b1;
    #30
    $display ("Simulation ended");
    end

endmodule
```

Note:

In simulation, it's necessary to specify the propagation delay i.e. the amount of delay from the input to the output of its gates. In Verilog, the propagation delay of a gate is specified in terms of *time units* and is specified by the symbol #.

The association of a time unit with physical time is made with the **'timescale** compiler directive.

E.g. **'timescale 1ns/100ps** -> 1st value specifies the unit of measurement for time delays. The 2nd value specifies the precision for which delays are rounded off, in this case to 0.1 ns.

Top Module with Propagation Delays

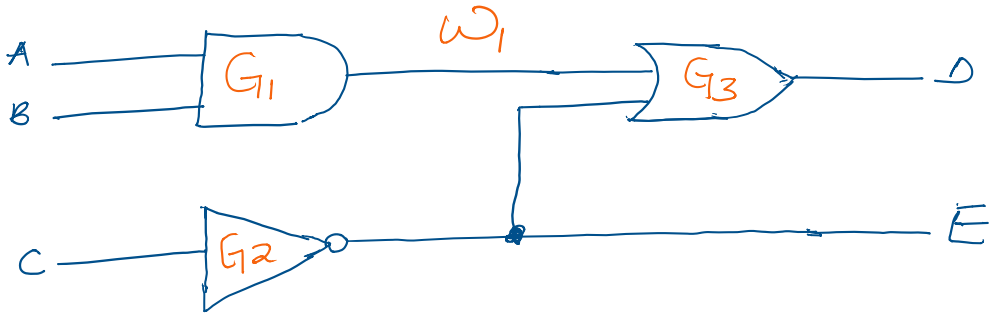
```
module example_one_prop_delay (A, B, C, D, E);
//Declare inputs, outputs and internal connections
input A, B, C;
output D, E;
wire w1;

//Implementation of the circuit / schematic
and #(20)G1(w1, A, B); //Instantiate the and module
not #(10)G2(E, C);
or #(30)G3(D, w1, E);

endmodule
```

Example 1_B:

- Logic circuit described by Boolean Equation



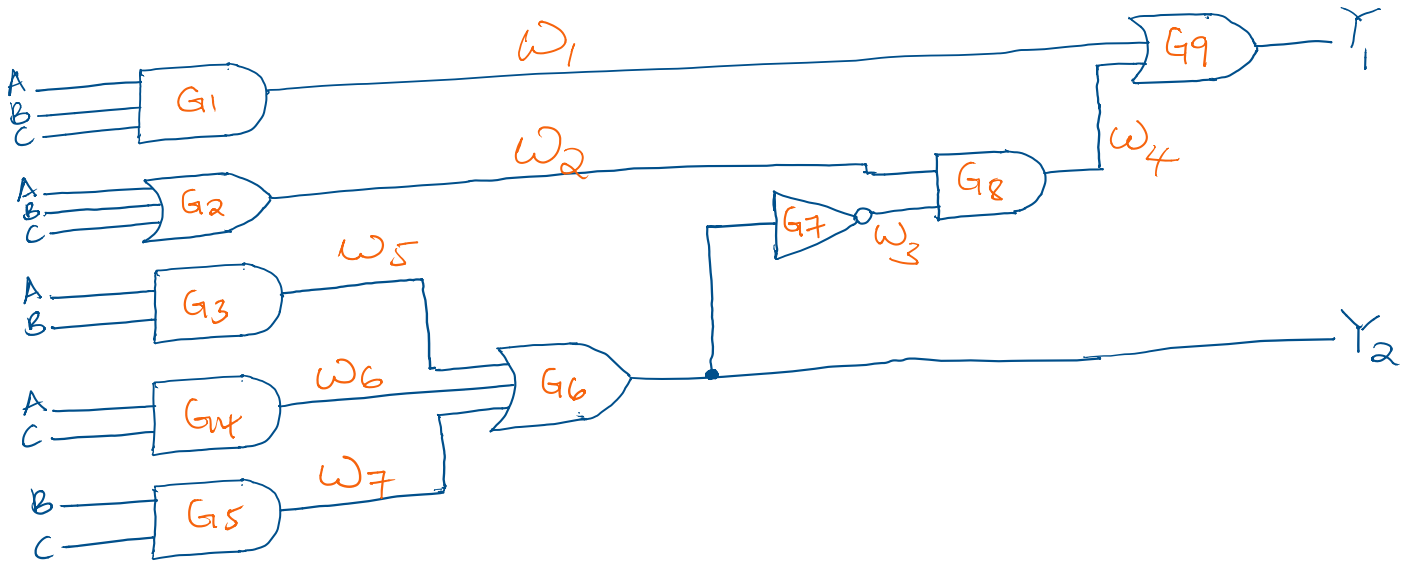
$$\mathbf{D = AB + \overline{C}}$$

Top Module

```
module example_one (A, B, C, D, E);
//Declare inputs, outputs and internal connections
input A, B, C;
output D, E;

//Implementation of the Boolean Equations
assign D = (A & B)|~ C;
assign E = ~ C;
endmodule
```

Example 2:



Boolean functions

$w_1 = ABC$

$w_2 = A + B + C$

$w_5 = AB$

$w_6 = AC$

$w_7 = BC$

$Y_2 = w_5 + w_6 + w_7 = AB + AC + BC$

$w_3 = \overline{Y_2}$

$w_4 = w_2 w_3 = \overline{Y_2}\, w_2$

$Y_1 = w_1 + w_4$

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

Top Module

```
module example_two (A, B, C, Y1, Y2);
//Declare inputs, outputs and internal connections
input A, B, C;
output Y1, Y2;
wire w1, w2, w3, w4, w5, w6, w7;

//Implementation of the circuit / schematic
and G1(w1, A, B, C); //Instantiate the and module
or G2(w2, A, B, C);
and G3(w5, A, B);
and G4(w6, A, C);
and G5(w7, B, C);
or G6(Y2, w5, w6, w7);
not G7(w3, Y2);
and G8(w4, w2, w3);
or G9(Y1, w1, w4);

endmodule
```

Test Bench

```
`timescale 1ns/1ns
`include "example_two.v"

module example_two_tb;
reg A, B, C; //inputs to simulate
wire Y1, Y2;   //outputs to simulate

//Instantiate the module that we are simulating
example_two  tb_two (A, B, C, Y1, Y2); //Retain the
same order

//simulate the circuit using the initial block
initial
    begin
    $dumpfile("example_two_tb.vcd");
    $dumpvars(0,example_two_tb);
```

```
    A = 1'b0; B = 1'b0; C = 1'b0;
    #30 B = 1'b1;
    #30 A = 1'b0; B = 1'b1; C = 1'b1;
    #30 A = 1'b1; B = 1'b1; C = 1'b0;
    #60 A = 1'b1; B = 1'b1; C = 1'b1;
    #30
    $display ("Simulation ended");
    end

endmodule
```