Digital Circuit Design

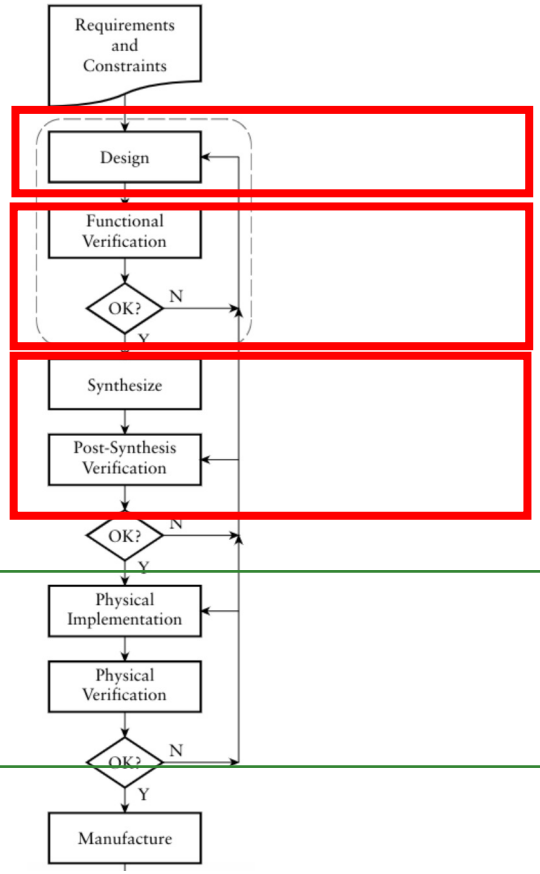
Li Bai

FPGA and ASIC

- FPGA: Field programmable gate array
  - Use computer languages that describe digital circuits, not programs to be executed on CPU
  - hard description language (VHDL and Verilog - IEEE 1364, Verilog 95, 2001, 2005 - systemverilog)
  - Xilinx and Altera (Intel) 87% of market share
  - Vivado and Quartus (IDE to synthesis Verilog code on FPGA), concurrent program

- ASIC: application specific integrated circuit - Arduino, raspberry pi, maybe your iPhone/android
  - Software: program load from memory and executed on CPU in sequential

Understand Designs (pg. 27)



Design under testing (DUT)

Unit under testing (UUT)

1. Testbench (verification)
2. Timing diagram (verification)

Create an account in EDA Playground

Wrapper implementation

Download sof, bit file on FPGA

Vivado --→ logic sim

Quartus --→ model sim

EDA playground

https://edaplayground.com/

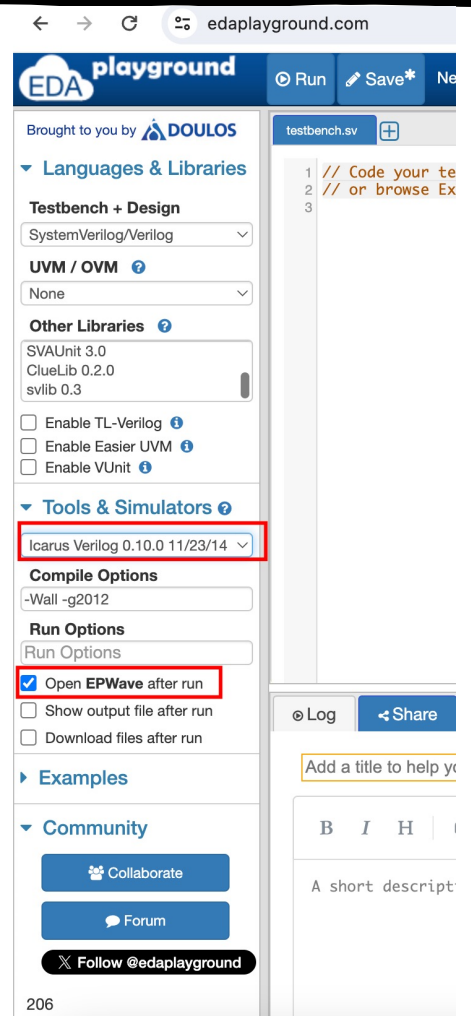
Verilog Syntax

- wire (verilog)
  - wire var
- Reg (verilog)
  - reg var
- Logic (system verilog)
  - logic var
- Bus (system verilog, need to specify number of bits)
  - wire [6:0] var
  - reg [6:0] var
  - logic [6:0] var

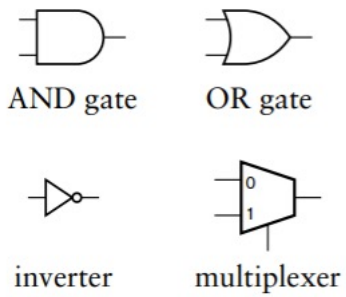
Create a EDA playground account

- Use your temple account as a gmail account to login
- You can share the code with short url link
- You have the eda files stored.
- You can use it for simple verification testing

https://edaplayground.com/x/hHX_



Gate symbols



A	B	AND	OR	XOR	Inverter (A)
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

$$A \text{ and } \mathrm{B} = \mathrm{A} \cdot \mathrm{B} \qquad A \text{ or } \mathrm{B} = \mathrm{A} + \mathrm{B}$$

Verilog and Boolean algebra

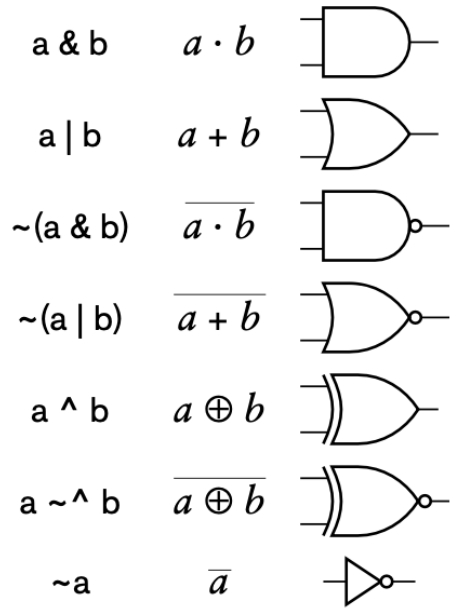


FIGURE 2.12 Verilog operators and their corresponding Boolean operations and gates.

EXERCISE 2.16 For each of the following Boolean equations, write a Verilog model for a circuit that implements the equation.

a) $m = a \cdot b + b \cdot c + a \cdot c$

b) $s = \overline{(x + y)} \cdot (x + \overline{z})$

c) $y = (a \oplus b) \cdot (a + c)$

The Axioms of Boolean algebra

- Commutative laws:

$$x + y = y + x \tag{2.1}$$

$$x \cdot y = y \cdot x \tag{2.2}$$

- Associative laws:

$$(x + y) + z = x + (y + z) \tag{2.3}$$

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{2.4}$$

- Distributive laws:

$$x + (y \cdot z) = (x + y) \cdot (x + z) \tag{2.5}$$

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z) \tag{2.6}$$

- Identity laws:

$$x + 0 = x \tag{2.7}$$

$$x \cdot 1 = x \tag{2.8}$$

- Complement laws:

$$x + \overline{x} = 1 \tag{2.9}$$

$$x \cdot \overline{x} = 0 \tag{2.10}$$

Theorem of Boolean Algebra

- Idempotence laws:

$$x + x = x \tag{2.11}$$

$$x \cdot x = x \tag{2.12}$$

- Further identity laws:

$$x + 1 = 1 \tag{2.13}$$

$$x \cdot 0 = 0 \tag{2.14}$$

- Absorption laws:

$$x + (x \cdot y) = x \tag{2.15}$$

$$x \cdot (x + y) = x \tag{2.16}$$

- DeMorgan laws:

$$\overline{(x + y)} = \overline{x} \cdot \overline{y} \tag{2.17}$$

$$\overline{(x \cdot y)} = \overline{x} + \overline{y} \tag{2.18}$$

Example

EXERCISE 2.5 Derive a truth table for the Boolean function implemented by the circuit in Figure 2.24.

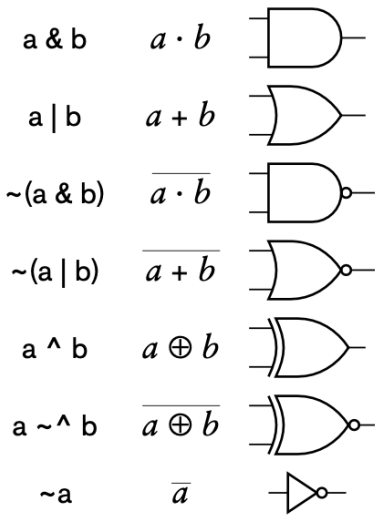
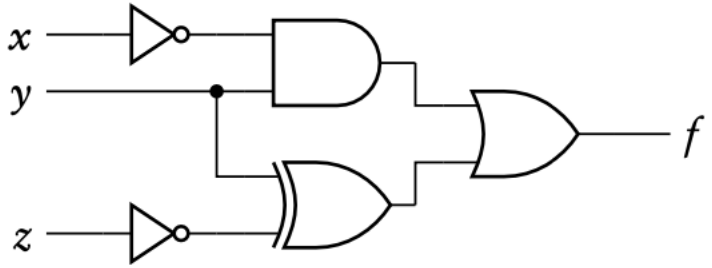


FIGURE 2.12 Verilog operators and their corresponding Boolean operations and gates.



Sum of Product

a	f	b	c	f
0	0	0	0	1
0	0	1	0	
0	1	0	0	
0	1	1	1	
1	0	0	0	1
1	0	1	0	
1	1	0	0	
1	1	1	0	

$\bar{a} \cdot \bar{b} \cdot \bar{c}$

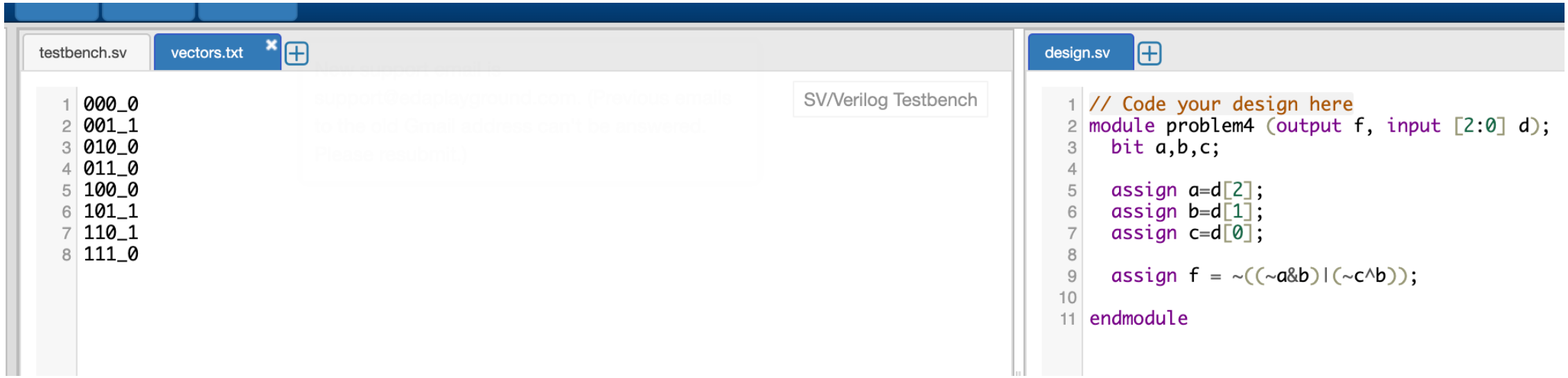
$\bar{a} \cdot b \cdot c$

$a \cdot \bar{b} \cdot \bar{c}$

$$f = \bar{a} \cdot \bar{b} \cdot \bar{c} + \bar{a} \cdot b \cdot c + a \cdot \bar{b} \cdot \bar{c}$$

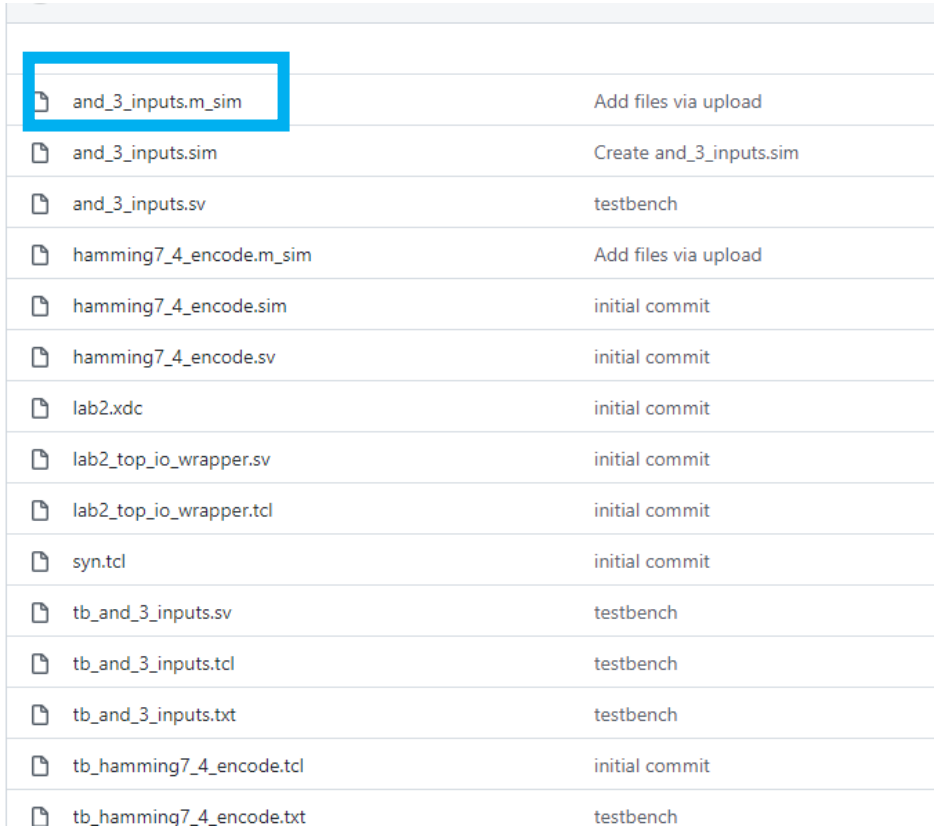
EDA Playground – test bench

- https://edaplayground.com/x/5gJq



Verilog file structure

- gates.m_sim
- gates.sim
- gates.sv
- lab1.xdc
- lab1_top_io_wrapper.sv
- lab1_top_io_wrapper.tcl
- tb_gates.sv
- tb_gates.tcl
- tb_gates.txt



and_3_inputs.m_sim	Add files via upload
and_3_inputs.sim	Create and_3_inputs.sim
and_3_inputs.sv	testbench
hamming7_4_encode.m_sim	Add files via upload
hamming7_4_encode.sim	initial commit
hamming7_4_encode.sv	initial commit
lab2.xdc	initial commit
lab2_top_io_wrapper.sv	initial commit
lab2_top_io_wrapper.tcl	initial commit
syn.tcl	initial commit
tb_and_3_inputs.sv	testbench
tb_and_3_inputs.tcl	testbench
tb_and_3_inputs.txt	testbench
tb_hamming7_4_encode.tcl	initial commit
tb_hamming7_4_encode.txt	testbench

Testbench

```
module tb_gates;

        // Inputs
        logic [1:0] d_in;

        // Outputs
        logic [3:0] d_out;

        // Instantiate the Unit Under Test (UUT)
        gates uut (
                .a0(d_in[0]), .b0(d_in[1]), .f0(d_out[0]), .a1(d_in[0]), .b1(d_in[1]), .f1(d_out[1]),
                .a2(d_in[0]), .b2(d_in[1]), .f2(d_out[2]), .a3(d_in[0]), .b3(d_in[1]), .f3(d_out[3])
        );
```

```
    `timescale 1ns / 1ps
    module gates(
            output logic f0, output logic f1, output logic f2, output logic f3,
            input logic a0, input logic b0, input logic a1, input logic b1,
            input logic a2, input logic b2, input logic a3, input logic b3
            );

            // Write code starting here ...

    endmodule
```

master — 2613_2021s / lab1 / tb_gates.txt

lbaitemple initial commit

1 contributor

12 lines (12 sloc) | 307 Bytes

```
1  //
2  //
3  // This file contains the test vectors for the gates code.
4  // The first two columns are the inputs: d_in[1:0] - which
5  // get mapped to the a & b inputs of each gate.
6  // The next four columns are the outputs of the gates.
7  // The operations are AND, OR, XOR and NAND.
8  //
9  00_1000
10 01_1110
11 10_1110
12 11_0011
```

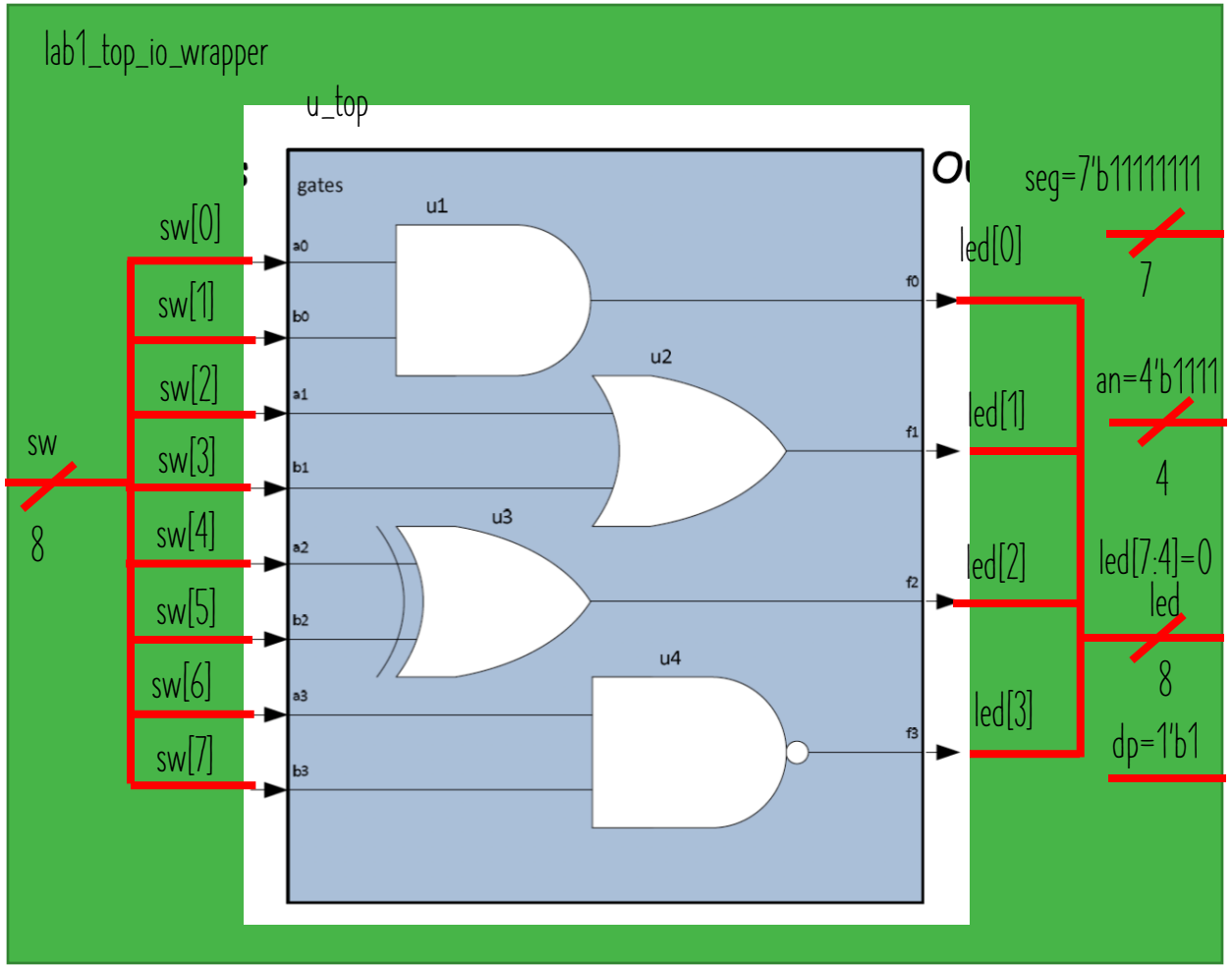
Instantiation

```
module submodule(input logic a, input logic b, output logic c)

....

endmodule
```

To instantiate submodule, you will need to

```
submodule uut (.a(w1), .b(w2), .c(w3));
```

Simple Verilog Syntax (uut and Instantiation)




```
`timescale 1ns / 1ps
module gates(
        output logic f0, output logic f1, output logic f2, output logic f3,
        input logic a0, input logic b0, input logic a1, input logic b1,
        input logic a2, input logic b2, input logic a3, input logic b3
        );

        // Write code starting here ...

endmodule
```

Can you draw the blockdiagram?

```
module tb_gates;

	// Inputs
	logic [1:0] d_in;

	// Outputs
	logic [3:0] d_out;

	// Instantiate the Unit Under Test (UUT)
	gates uut (
		.a0(d_in[0]), .b0(d_in[1]), .f0(d_out[0]), .a1(d_in[0]), .b1(d_in[1]), .f1(d_out[1]),
		.a2(d_in[0]), .b2(d_in[1]), .f2(d_out[2]), .a3(d_in[0]), .b3(d_in[1]), .f3(d_out[3])
	);
```

Example (in group)