** San Francisco Bay University**

**EE461L - Verilog HDL Lab**

**Week#1 Introduction to Verilog and**

**Combinational & Sequential Logic Design**

1. **Lab Outlines:**
2. Example “Hello World” & Simulation
3. Sequential Logic in RTL
4. Combinational Logic in RTL
5. **Lab Procedures:**

1. **Example “Hello World” & Simulation**

//-----------------------------------------------------

// This is my first Verilog Program

// Design Name : hello\_world

// File Name : hello\_world.v

// Function : This program will print 'hello world'

//-----------------------------------------------------

module helloWorld;

initial begin

$display ("Hello World!!!");

#10 $finish;

end

endmodule // End of Module helloWorld

-Run this program by the online compiler

You will see “Hello World!!!” on the monitor

-Why does hardware design need Verilog?

Traditional design: pen & paper

example: 1-bit full adder design

Step 1: Truth table buildup

Step 2: Get logic function & simplify it by K-map or other methods.

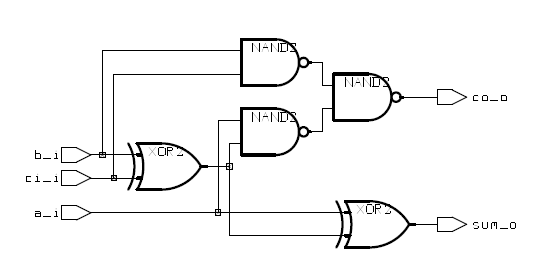
Step 3: Create gate logic schematic.

Verilog code:

{co, sum} = a + b + cin;

Logic Synthesiser(software)

Verilog code ==== > gate logic schematic



1. **HDL Level and examples:**

* **Behavior:** describe function only. We can't create logic schematic by synthesizer through it. Pure software likes high level computer language. Usually, it is used for design idea verification.

begin: count1s

reg [7:0] tempreg;

count = 0;

tempreg = rega;

while (tempreg)begin

if (tempreg[0]) count = count + 1;

tempreg = tempreg >> 1;

end

end

* **RTL (low level):** logic schematic can be created by synthesizer.
* **Gate level:** Final product of verilog design. It is equivalent to logic schematic.

module gates();

wire out0;

wire out1;

wire out2;

reg in1,in2,in3,in4;

not U1(out0,in1);

and U2(out1,in1,in2,in3,in4);

xor U3(out2,in1,in2,in3);

endmodule

1. **Combinational Logic & Synthesis in RTL**

* Design Example: 1-bit full adder

1. “continuous assignment” in oneBitFA1.v

module oneBitFA1(

input wire a\_i,

input wire b\_i,

input wire ci\_i,

output wire sum\_o,

output wire co\_o

);

assign {co\_o,sum\_o} = a\_i + b\_i + ci\_i;

endmodule

“always block” in oneBitFA2.v

module oneBitFA2(

input wire a\_i,

input wire b\_i,

input wire ci\_i,

output reg sum\_o,

output reg co\_o

);

always @(a\_i, b\_i, ci\_i)begin  
 {co\_o, sum\_o} = a\_i + b\_i + ci\_i;

end

endmodule

1. **Sequential Logic in RTL**

Synchronous DFF:

> vi DFFSynch .v

module DFFSynch(

d\_i,

rst\_i,

clk\_i,

q\_o

);

input d\_i,rst\_i,clk\_i;

output q\_o;

reg q\_o;

always @(posedge clk\_i)begin

if(rst\_i) q\_o <= 0;

else q\_o <= d\_i;

end

endmodule

Asynchronous DFF:

> vi DFFAsynch .v

module DFFAsynch(

d\_i,

rst\_i,

clk\_i,

q\_o

);

input d\_i,rst\_i,clk\_i;

output q\_o;

reg q\_o;

always @(posedge clk\_i or posedge rst\_i)begin

if(rst\_i) q\_o <= 0;

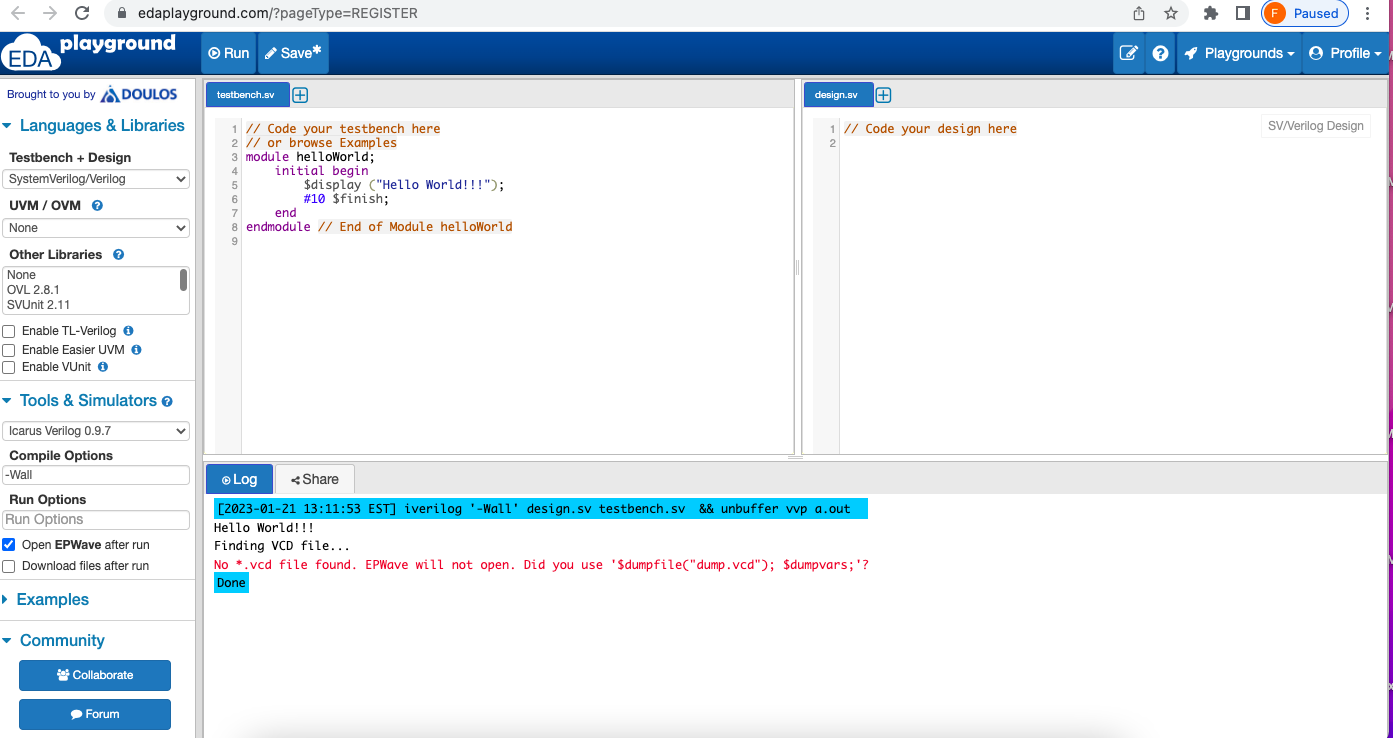
else q\_o <= d\_i;

end

endmodule

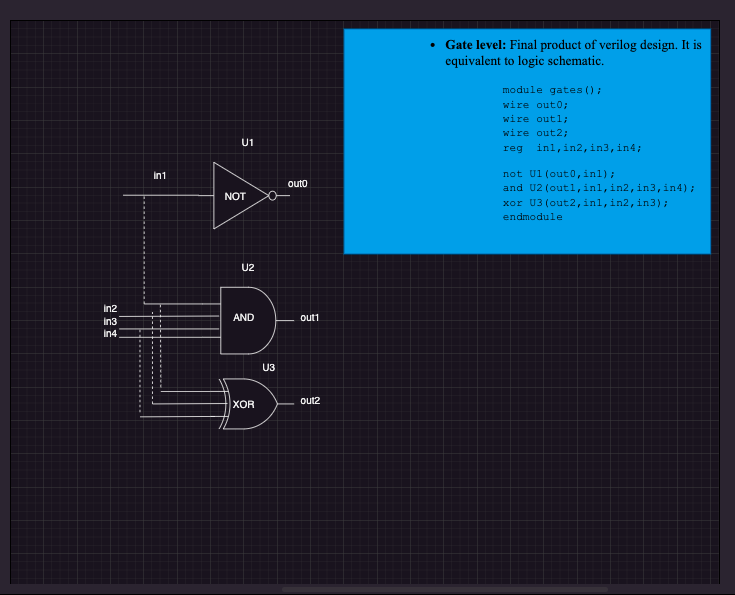
1. **Exercise: Combinational logic design**
   * + - 1. Directly run the module *helloWorld* in the online compiler

Output:



* + - * 1. Based the module *gates*, draw the digital circuit schematic.

Answer:



* + - * 1. Create the testbench first, and then simulate the design modules *oneBitFA1*, *oneBitFA2*, *DFFSynch*, and *DFFAsynch*.

**testbench for oneBitFA1**

module fullAdder\_tb;

reg a\_i, b\_i, cin; // input variables are always assigned reg datatype

wire cout, sum;

oneBitFA1 U1(

.a\_i(a\_i),

.b\_i(b\_i),

.ci\_i(cin),

.cout(cout),

.sum\_o(sum)

);

initial begin

$dumpfile("waveform.vcd"); // fuile for dumping the variables

$dumpvars(2, fullAdder\_tb)

a\_i = 1'b0; // initialize Inputs

b\_i = 1'b0;

cin = 1'b0;

#10 a\_i = 1'b0;

#10 b\_i = 1'b0;

#10 cin = 1'b1;

#10 a\_i = 1'b0;

#10 b\_i = 1'b1;

#10 cin = 1'b0;

#10 a\_i = 1'b0;

#10 b\_i = 1'b1;

#10 cin = 1'b1;

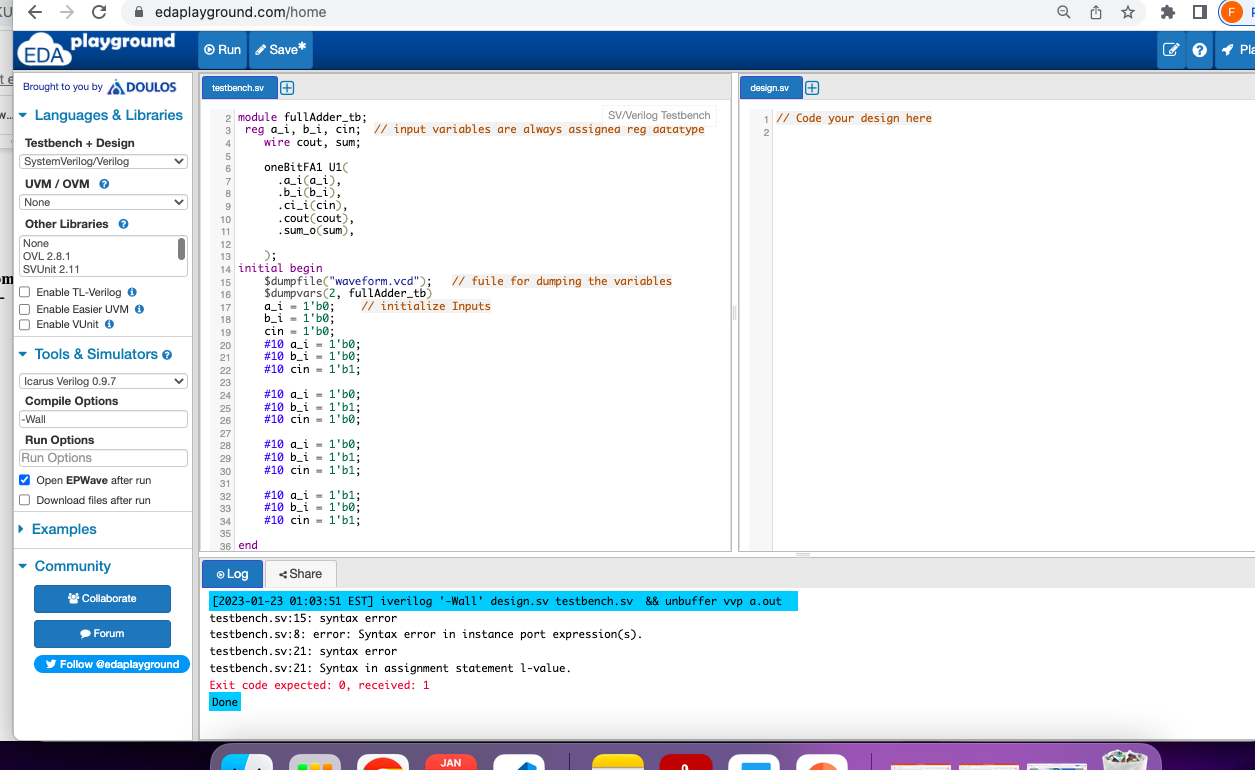
#10 a\_i = 1'b1;

#10 b\_i = 1'b0;

#10 cin = 1'b1;

end

endmodule



**testbench for oneBitFA2**

/timescale directive

`timescale 1ns / 1ps

module oneBitFA2;

//declare testbench variables

reg a\_input,

reg b\_input,

reg c\_input;

wire sum,

wire c\_output;

//instantiate the design module and connect to the testbench variables

full\_adder instantiation

(

.a\_i(a\_input),

.b\_i(b\_input),

.ci\_i(c\_input),

.sum\_o(sum),

.co\_o(c\_output));

initial

begin

$dumpfile("xyz.vcd");

$dumpvars;

//set stimulus to test the code

a\_input=0;

b\_input=0;

c\_input=0;

#100 $finish;

end

//provide the toggling input (just like truth table input)

//this acts as the clock input

always #40 a\_input=~a\_input;

always #20 b\_input=~b\_input;

always #10 c\_input=~c\_input;

//display output if there’s a change in the input event

always @(a\_input or b\_input or c\_input)

$monitor("At TIME(in ns)=%t, a=%d b=%d c=%d sum = %d Carry = %d", $time, a\_input, b\_input, c\_input, sum, c\_output);

endmodule

