



DIGITAL DESIGN

LAB2 BUILD DIGITAL DESIGN AND TESTBENCH IN VERILOG

2022 SUMMER TERM

TOPIC

- Verilog (1)
 - keyword
 - comments
 - time
 - wire vs reg
 - statement vs block statement (initial)
 - digital design vs testbench
 - verilog testbench
 - repeat, forever, for, while
 - \$display, \$write, \$monitor

KEYWORDS IN VERILOG

- Keyword is a special identifier reserved in the language for defining the language structure. Keyword all lowercase
 - **module, endmodule**
 - **input, output**
 - **wire, tri, reg**
 - **assign**
 - **initial, always**
 - **begin, end**
 - *Primitive gates* : **and, nand, or, nor, xor, xnor, not, buf**

MODULE(1)

```
module Simple_Circuit (  
    input [15:0] sw,  
    output [15:0] led  
);  
    assign led=sw;  
endmodule
```

```
module Simple_Circuit (sw, led);  
    input [15:0] sw;  
    output [15:0] led;  
  
    assign led=sw;  
endmodule
```

It is declared by the keyword **module** and must always be terminated by the keyword **endmodule** . Each statement must be terminated with a semicolon, but there is no semicolon after **endmodule**.

MODULE(2)

- The keyword **module** is followed by a name and a list of ports.
- The name (*Simple_Circuit* in this example) is an **identifier**. Identifiers are names given to modules, variables (e.g., `sw signal`), and other elements of the language so that they can be referenced in the design.
- In general, we choose meaningful names for modules. Identifiers are composed of alphanumeric characters , the underscore (`_`) and the dollar(`$`), and are **case sensitive**. Identifiers must start with an alphabetic character or an underscore, but they cannot start with a number or a dollar. The keywords **input** and **output** specify which of the ports are inputs and which are outputs.

COMMENTS

- We use comments to enhance the readability of the program.
- Single-line comments
 - Started with “//”
 - Verilog will ignore the contents from this mark to the end of the line.
- Multiline comments
 - Begin with “/*”, terminate with “*/”
 - Verilog will ignore the contents between the two marks.

GATE DELAY(1)

- In Verilog, the propagation delay of a gate is specified in terms of *time units* and by the symbol `#`. The numbers associated with time delays in Verilog are dimensionless.
- The association of a time unit with physical time is made with the ``timescale` compiler directive. (Compiler directives start with the (```) back quote, or grave accent, symbol.) Such a ``` directive is specified before the declaration of a module and applies to all numerical values of time in the code that follows.

GATE DELAY(2)

- An example of a timescale directive is

```
`timescale 1ns / 1ps
```

- The first number specifies the unit of measurement for time delays. The second number specifies the precision for which the delays are rounded off, in this case to 0.001 ns. If no timescale is specified, a simulator may display dimensionless values or default to a certain time unit, usually 1 ns ($=10^{-9}$ s). Our examples will use only the default time unit.

LEGAL BASE FORMAT

`<bit width>'<base format><number>`

- 's'/S: signed
- 'd'/D: Decimal
- 'b'/B: Binary
- 'o'/O: Octonary
- 'h'/H: Hexadecimal

```
sw_sim = 16'sd0;
```

```
sw_sim = 16'd0;
```

```
sw_sim = 16'b0000_0000_0000_0000;
```

```
sw_sim = 16'o00_0000;
```

```
sw_sim = 16'h0000;
```

TEST BENCH(1)

- An HDL description that provides the stimulus to a design is called a **test bench**. It's a *virtual platform for simulating input and output verification in real environments* .
- Within the **test bench**, the **inputs** to the circuit are declared with keyword **reg** and the **outputs** are declared with the keyword **wire**. The module *Simple_Circuit* is instantiated with the instance name *uut*.
- Every instantiation of a module must include a unique instance name.

Note that using a test bench is similar to testing actual hardware by attaching signal generators to the inputs of a circuit and attaching probes (wires) to the outputs of the circuit.

TEST BENCH(2)

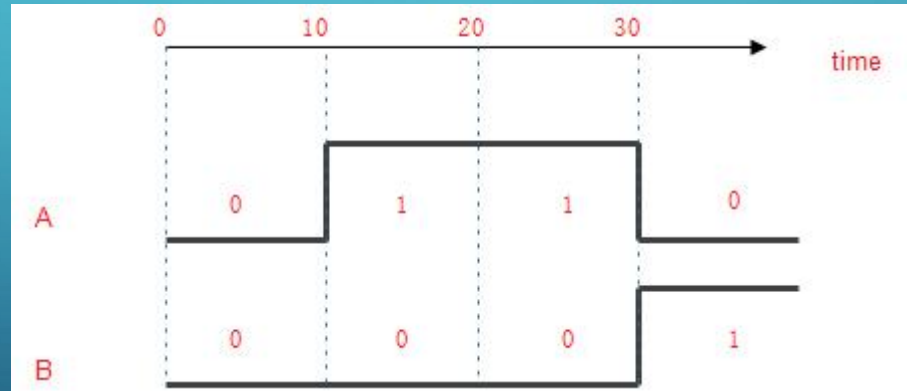
```
module Simple_Circuit _Tb( );  
reg [15:0] sw_sim=16'h0000;    //sw_sim is used to connect to the input of the tested module  
wire [15:0] led_sim;          // led_sim is used to connect to the output of the tested module  
Simple_Circuit uut(.sw(sw_sim), .led(led_sim));    //instantiate the unit, do the connection  
endmodule
```

INITIAL(1)

- The **initial** keyword is used with a set of statements that begin executing when the simulation is initialized; The **initial** statements are commonly used to describe waveforms in a **test bench**.
 - The set of statements to be executed is called a *block statement* and consists of several statements enclosed by the keywords **begin** and **end**.
 - The action specified by the statements begins when the simulation is launched, and the statements are executed in sequence, left to right, from top to bottom, by a simulator in order to provide the input to the circuit.

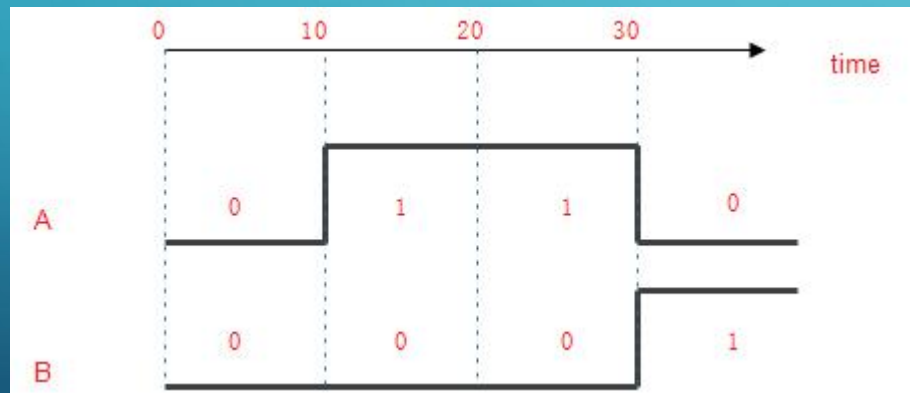
INITIAL(2)

- **The initial statement executes only once**, starting from simulation time 0, and may continue with any operations that are delayed by a given number of time units, as specified by the symbol #. For example, consider the initial block



INITIAL(3)

- The block is enclosed between the keywords `begin` and `end` . At time 0, A and B are set to 0. Ten time units later, A is changed to 1. Twenty time units after that (at $t = 30$), A is changed to 0 and B to 1.



initial

begin

A = 0; B = 0;

#10 A = 1;

#20 A = 0; B = 1;

end

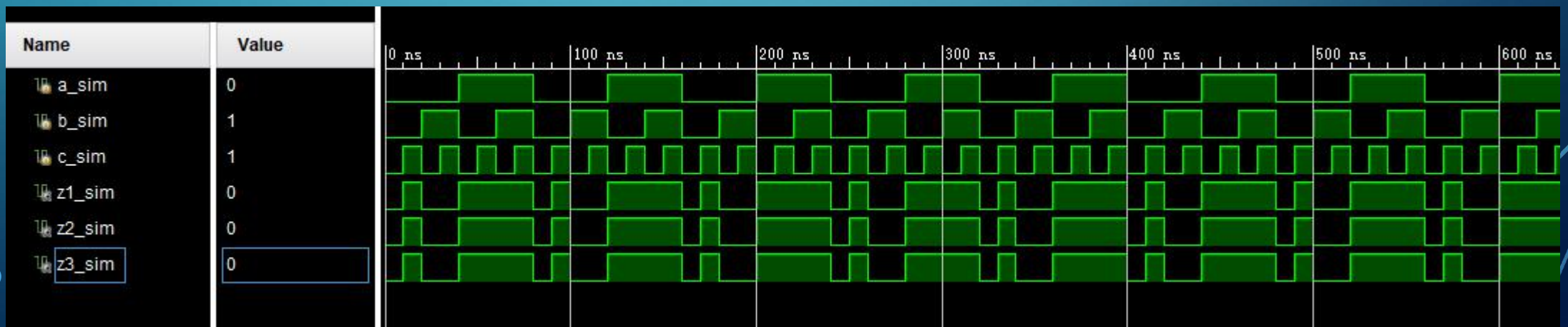
WIRE VS REG

- **wire** represents the physical connection between two hardware units, which can be a wire or a group of wires. The output of the logic gate needs to be declared as wire type, The assignment object of assign must be wire type.
- **reg** represents register on hardware , would be integrated into trigger or latch. a reg can hold data until a subsequent assignment statement changes it. both `initial` and `always` block can only do the assignment to the reg.

TESTBENCH USING ALWAYS

```
module lab2_summer2021_sim1();  
    reg a_sim, b_sim, c_sim;  
    wire z1_sim, z2_sim, z3_sim;  
  
    lab2_summer2021 uut1(  
        .a(a_sim),  
        .b(b_sim),  
        .c(c_sim),  
        .z1(z1_sim),  
        .z2(z2_sim),  
        .z3(z3_sim)  
    );
```

```
    initial begin  
        {a_sim, b_sim, c_sim} = 3'b000;  
        #1000 $finish;  
    end  
  
    always #10 {a_sim, b_sim, c_sim} = {a_sim, b_sim, c_sim} + 3'b001;  
endmodule
```



TESTBENCH USING LOOP

```
lab2_summer2021 uut2(sa, sb, sc, sz1, sz2, sz3);

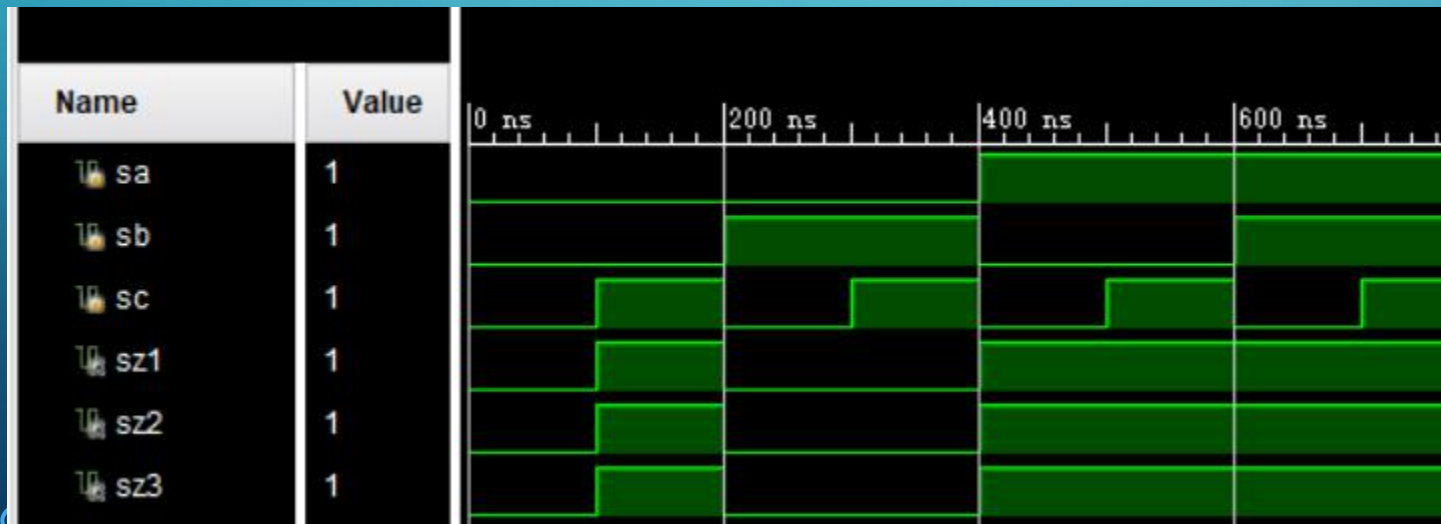
initial begin
    {sa, sb, sc} = 3'b000;
    repeat(7)
    begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #1000 $finish(1);
end
```

```
lab2_summer2021 uut3(sa, sb, sc, sz1, sz2, sz3);

initial begin
    {sa, sb, sc} = 3'b000;
    for(integer i=0; i<7; i=i+1)
    begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #1000 $finish(1);
end
```

```
lab2_summer2021 uut4(sa, sb, sc, sz1, sz2, sz3);

initial begin
    {sa, sb, sc} = 3'b000;
    while({sa, sb, sc} < 3'b111)
    begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #1000 $finish(1);
end
```



```
lab2_summer2021 uut5(sa, sb, sc, sz1, sz2, sz3);

initial begin
    {sa, sb, sc} = 3'b000;
    forever
    begin
        #100 {sa, sb, sc} = {sa, sb, sc} + 1;
    end
    #1000 $finish(1);
end
```

USING DISPLAY ,WRITE, MONITOR TO DISPLAY THE VALUE

- \$display is a system task, not Synthesizable
- Please try \$write and \$monitor separately

```
module lab2_summer2021_sim2();
    reg sa, sb, sc;
    wire sz1, sz2, sz3;

    lab2_summer2021 uut2(sa, sb, sc, sz1, sz2, sz3);

    initial begin
        {sa, sb, sc} = 3'b000;
        repeat(7)
            begin
                #100 {sa, sb, sc} = {sa, sb, sc} + 1;
                $display($time, " {sa, sb, sc}: %d", {sa, sb, sc});
            end
        #1000 $finish(1);
    end
endmodule
```

```
# run 1000ns
100 {sa, sb, sc}: 1
200 {sa, sb, sc}: 2
300 {sa, sb, sc}: 3
400 {sa, sb, sc}: 4
500 {sa, sb, sc}: 5
600 {sa, sb, sc}: 6
700 {sa, sb, sc}: 7

$finish called at time : 700 ns ;
```

PRACTICE 1

Create a project named as Lab2_Addition, design the source code to get the addition of two two-bit unsigned numbers, build a test bench to verify the function of your design, finally program the the FPGA chip with bitstream file to test your design.

Tips: there should be two inputs(we need input two operands through dial switch) and at least one output to show the result.

```
module Lab2_Addition(addend, augend, addend_led, augend_led, sum_led);  
input [1:0] addend, augend;  
output [1:0] addend_led, augend_led;  
output [?:0] sum_led;  
.....  
endmodule
```

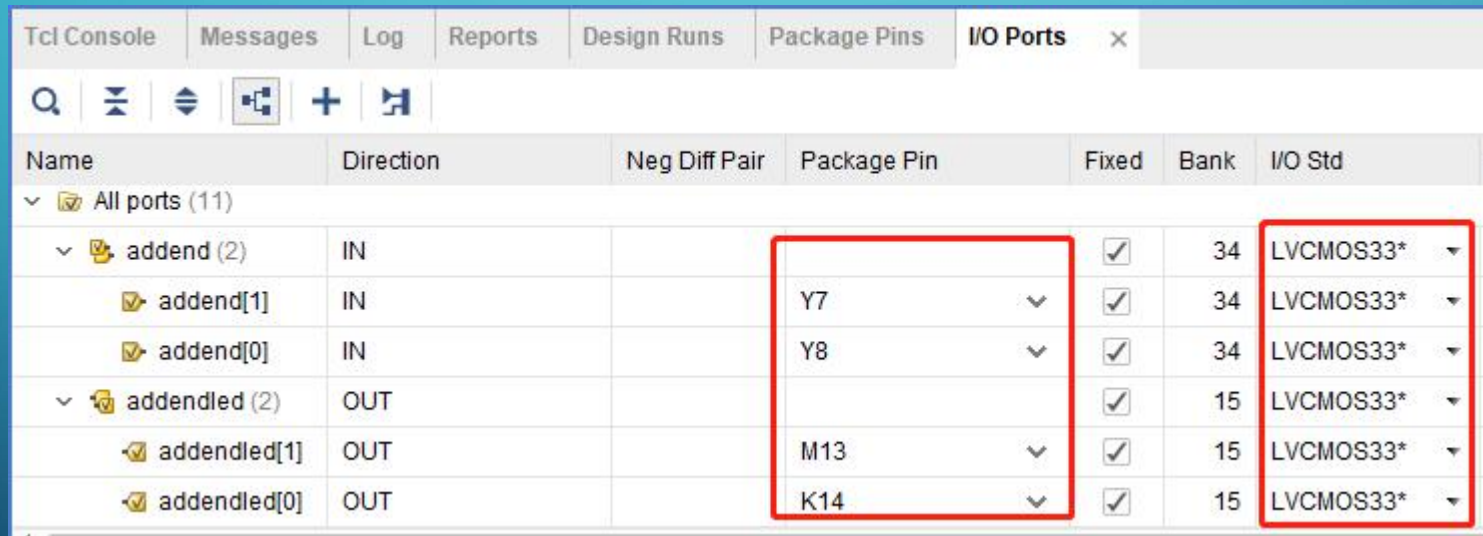
TIPS: WAVEFORM RADIX

- Right click the corresponding signal to change the radix



TIPS: SCONNECTION

- Open **Synthesized design**, connect the I/O ports to the package pin



Tcl Console	Messages	Log	Reports	Design Runs	Package Pins	I/O Ports	x
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	
▼ All ports (11)							
▼ addend (2)	IN			✓	34	LVCMOS33*	▼
addend[1]	IN		Y7 ▼	✓	34	LVCMOS33*	▼
addend[0]	IN		Y8 ▼	✓	34	LVCMOS33*	▼
▼ addendled (2)	OUT			✓	15	LVCMOS33*	▼
addendled[1]	OUT		M13 ▼	✓	15	LVCMOS33*	▼
addendled[0]	OUT		K14 ▼	✓	15	LVCMOS33*	▼