DIGITAL DESIGN

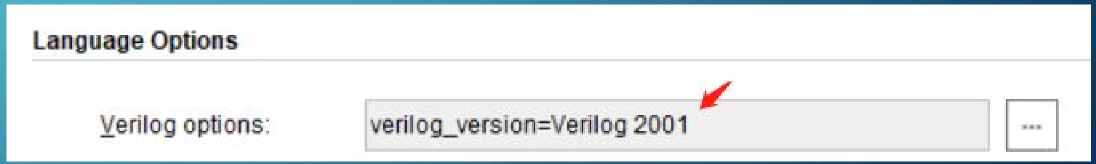
LAB2 BUILD DIGITAL DESIGN AND TESTBENCH IN VERILOG

TOPIC

- verilog (1)
  - keywords
  - module, endmodule, input, output
  - time
  - wire vs reg
  - statement vs block statement (initial)
  - digital design vs testbench

VERILOG - A TYPE OF HDL

- What is HDL(Hardware Description Language)?
  - HDL is a formal notation intended for use in all phases of the creation of electronic systems. Because it is both machine readable and human readable, it supports the development, verification, synthesis, and testing of hardware designs; the communication of hardware design data; and the maintenance, modification, and procurement of hardware.
- What version of verilog do we use?
  - https://ieeexplore.ieee.org/document/954909



- Any other chooices?
  - VHDL; SystemC; C; Python

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 (sw, led );
    input [23:0] sw;
    output [23:0] led;

    assign led=sw;
endmodule
```

```
module Simple_Circuit (
    input [23:0] sw,
    output [23:0] 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.

LEGAL BASE FORMAT

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

- ‘d’/D: Decimal
- ‘b’/B: Binary
- ‘o’/O: Octonary
- ‘h’/H: Hexadecimal

```
sw_sim = 24'd0;

sw_sim = 24'b0000_0000_0000_0000_0000_0000;

sw_sim = 24'o0000_0000;

sw_sim = 24'h00_0000;
```

8’d0	8’d15	8’d17
8’b0	8’b0000_1111	8’b0001_0001
8’o0	8’o17	8’o21
8’h0	8’h0f	8’h11

24'b0, 24'b0, 24'o0, 24'h0 are all right here.

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 [23:0] sw_sim=24'h00_0000; //sw_sim is used to connect to the input of the tested module

wire [23:0] led_sim; // led_sime 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

// TBD...

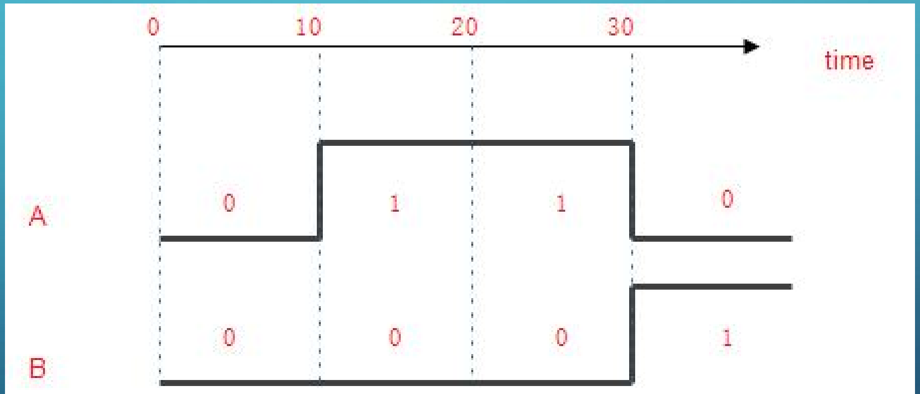
endmodule

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.

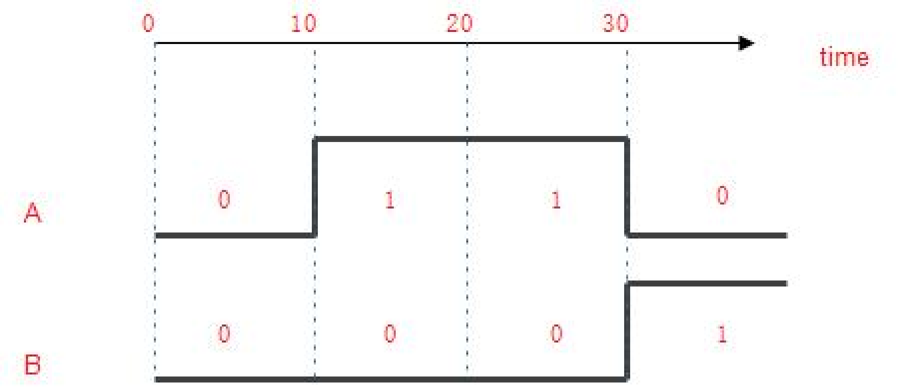
INITIAL(2)

- 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 = 1'b0; B = 1'b0;
        #10 A = 1'b1;
        #20 A = 1'b0; B = 1'b1;
        end
```

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.

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.

LAB2 TASK: TWO-BIT ADDITION(1)

TASK1: Create a project named as Lab2_Addition, design circuit to get the addition of two two-bit unsigned numbers, build a test bench to verify the function of your design, finally programe the the FPGA chip with bitstream file to test your design.

Prompt: there should be two inputs(we need input two operands through dial switch)

```
module Lab2_Addition(add_in1, add_in2, add_out);
input [1:0] add_in1, add_in2;
output [?:0] add_out;
……
endmodule
```

LAB2 TWO-BIT ADDITION(2)

- TASK2: build a design to implement the **addition of two-bit signed numbers** and test.
- Attention: Only when both operands are signed numbers can both operands be considered as signed numbers, otherwise both signed and unsigned numbers will be calculated as unsigned numbers.
- Adding keyword signed indicates a signed number. For example:

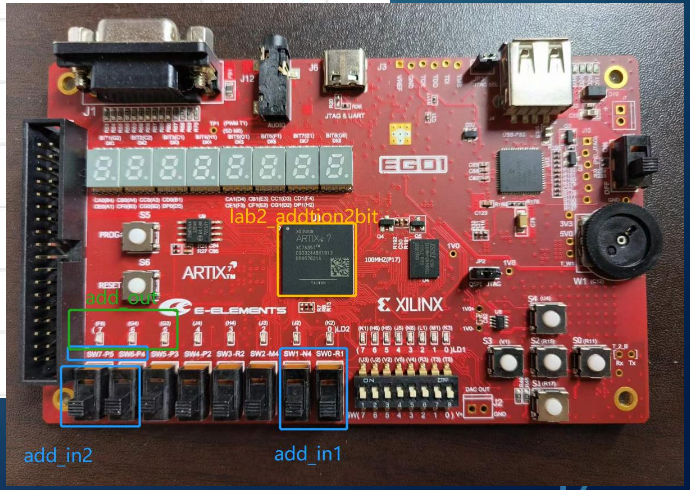
```
input signed[1:0] add_in1;
```

CONNECTION BETWEEN PACKAGE_PIN AND PORTS (1)

- Open **RTL analysis / Synthesized design**, connect the I/O ports to the package pin and set their I/O Std.

Tcl Console | Messages | Log | Reports | Design Runs | Package Pins | I/O Ports ×

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vre
add_in1[1]	IN		N4	✓	34	LVCMOS33*	3.300	
add_in1[0]	IN		R1	✓	34	LVCMOS33*	3.300	
add_in2 (2)	IN			✓	34	LVCMOS33*	3.300	
add_in2[1]	IN		P5	✓	34	LVCMOS33*		
add_in2[0]	IN		P4	✓	34	LVCMOS33*		
add_out (3)	OUT			✓	35	LVCMOS33*		
add_out[2]	OUT		F6	✓	35	LVCMOS33*		
add_out[1]	OUT		G4	✓	35	LVCMOS33*		
add_out[0]	OUT		G3	✓	35	LVCMOS33*		

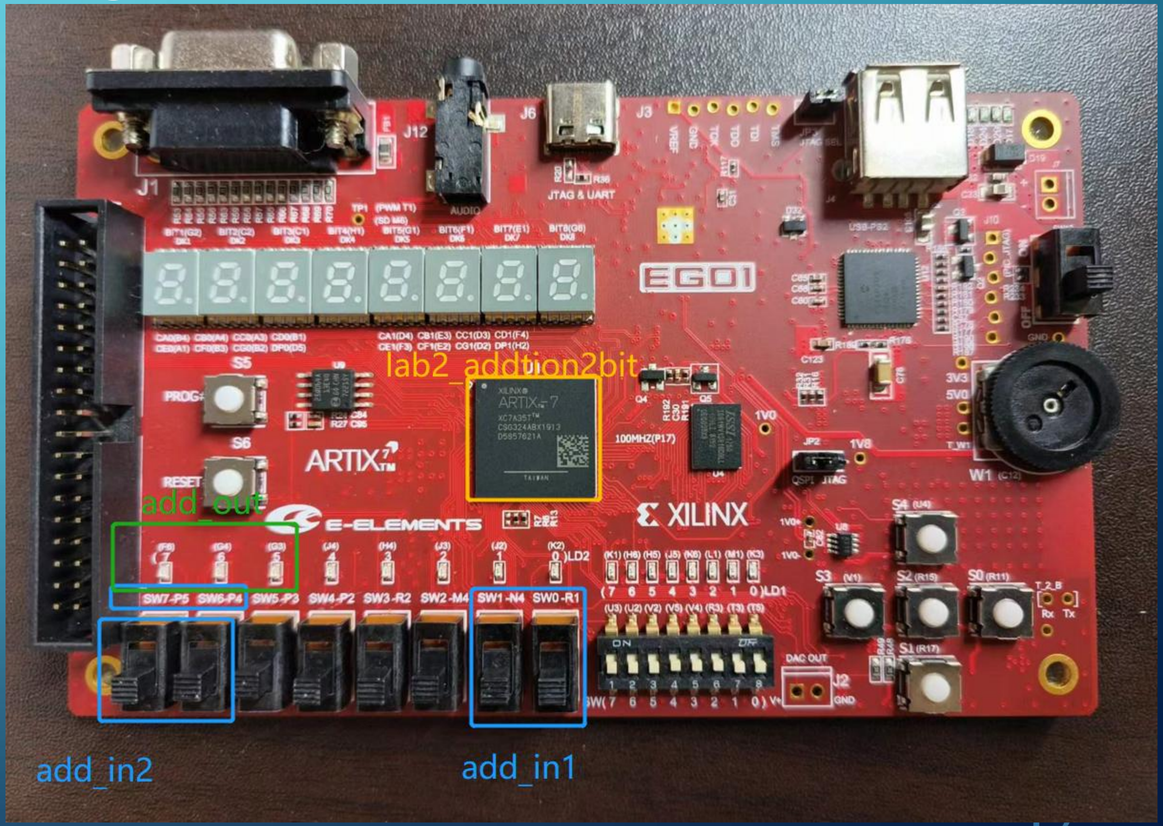


CONNECTION BETWEEN PACKAGE_PIN AND PORTS （2）

- Edit the constraints file could also make the connection between package_pins of FPGA chip and ports of the designed circuit .

lab2_addition2bit.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports {add_in1[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {add_in1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {add_in2[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {add_in2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {add_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {add_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {add_out[0]}]
set_property PACKAGE_PIN F6 [get_ports {add_out[2]}]
set_property PACKAGE_PIN G4 [get_ports {add_out[1]}]
set_property PACKAGE_PIN G3 [get_ports {add_out[0]}]
set_property PACKAGE_PIN P5 [get_ports {add_in2[1]}]
set_property PACKAGE_PIN P4 [get_ports {add_in2[0]}]
set_property PACKAGE_PIN R1 [get_ports {add_in1[0]}]
set_property PACKAGE_PIN N4 [get_ports {add_in1[1]}]
```



TEST ON THE BOARD(1)

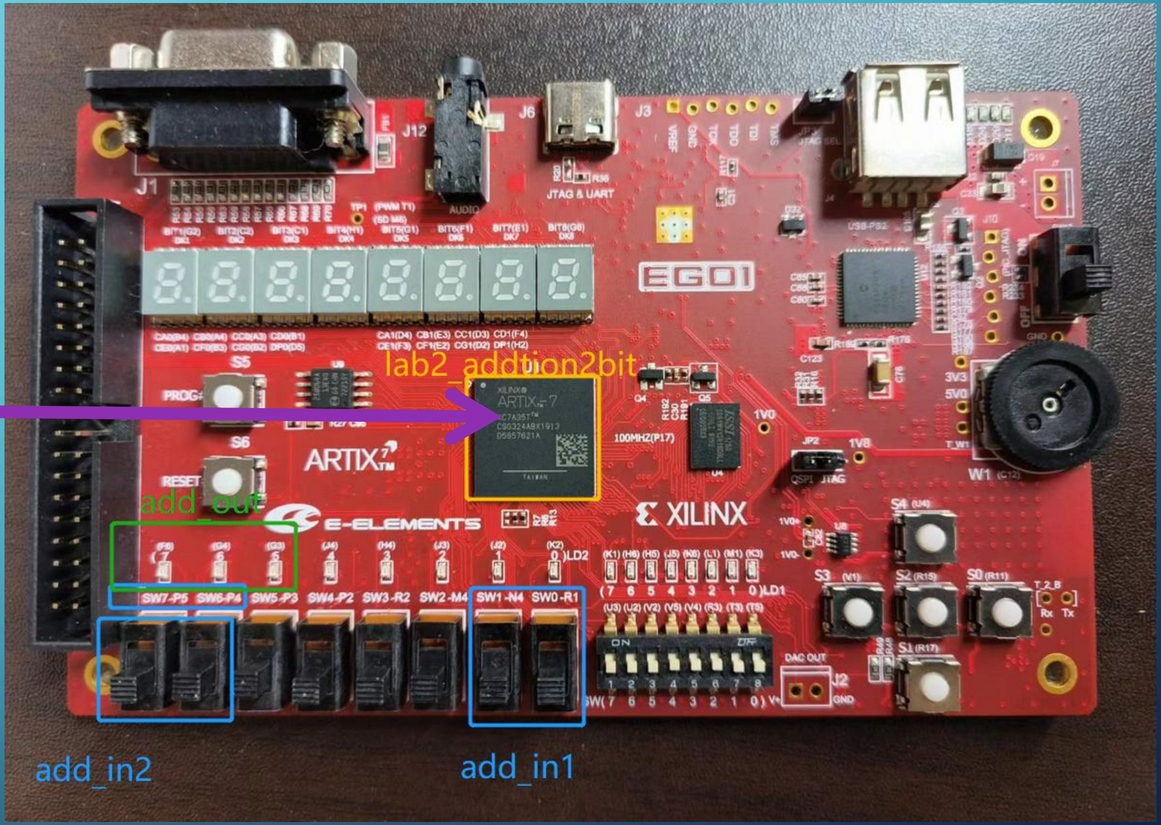
PC

vivado project

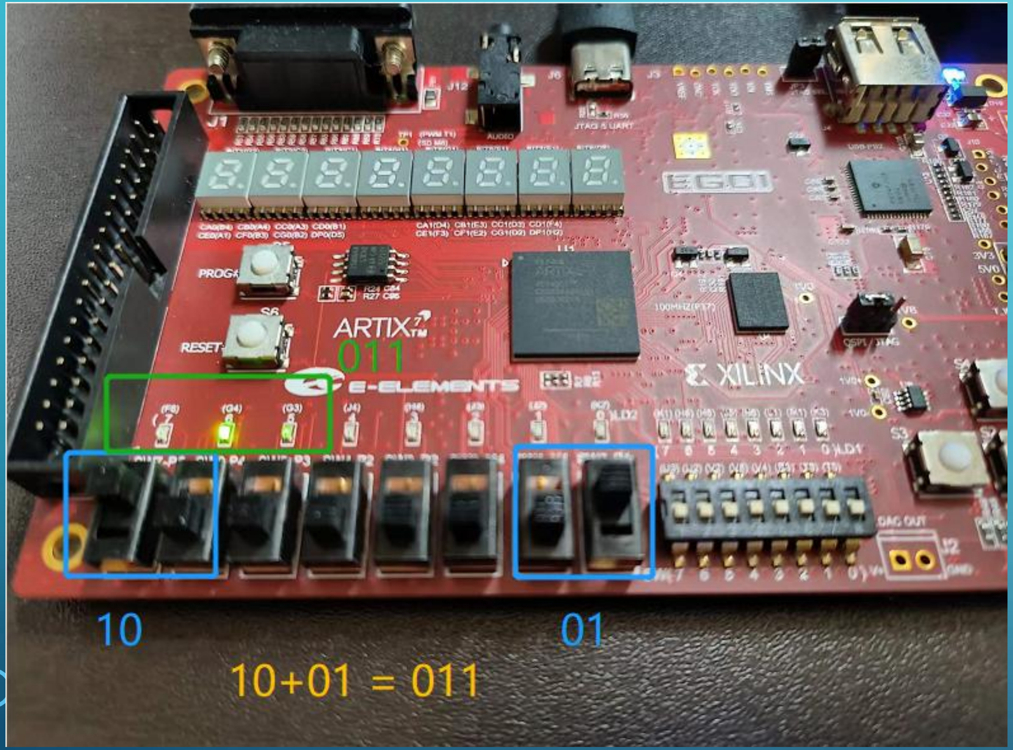
desgin sources file → bitsream file

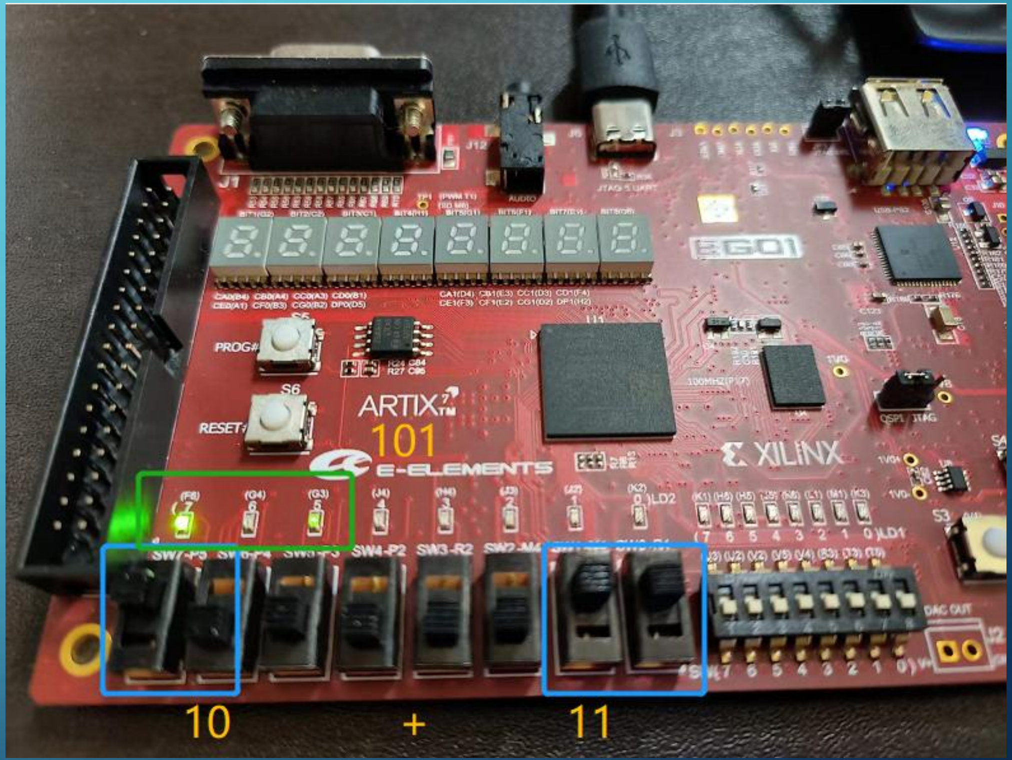
constraints file → bitsream file

Hardware Manager



TEST ON THE BOARD(2)





TEST BY SIMULATION

PC

vivado project

desgin sources file

simulation source file

simulator

```
module lab2_addition_sim();
reg [1:0] in1_sim, in2_sim;
wire [2:0] out_sim;
addition2bit ua1(.add_in1(in1_sim),.add_in2(in2_sim),.add_out(out_sim));
initial begin
    in1_sim = 2'b0; in2_sim = 2'b0;
    [illegible]
    #10 $finish();
end
endmodule
```