Verilog 簡介-- for Lab 0

Reference:

M. Morris Mano and Michael D. Ciletti, *Digital Design*, 5th ed., 2013, Prentice Hall. (§3-9)

J.J. Shann

Design Flow of an Integrated Circuit (IC)

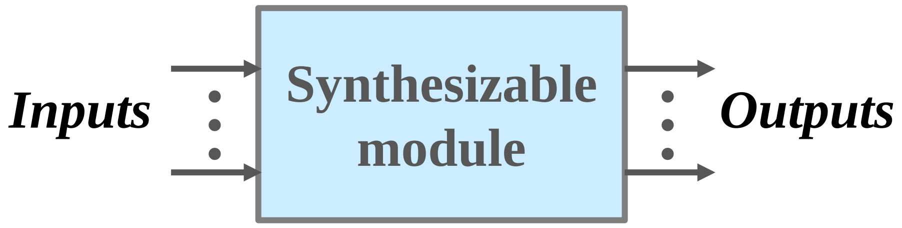
- Major steps in the design flow of an IC:
  - Design entry: **Verilog** **HDL: Hardware Description Language**
    - creates an HDL-based description of the functionality that is to be implemented in hardware
  - Function simulation or verification: **ModelSim**
    - displays the behavior of a digital system, e.g., simulation waveforms, through the use of a computer
  - Logic synthesis
    - derives a list of physical components and their interconnections, *netlist*, from the model of a digital system described in an HDL
  - Timing verification
    - confirms that the fabricated IC will operate at a specified speed
  - Fault simulation
    - compares the behavior of an ideal ckt w/ the behavior of a ckt that contains a process-induced flaw

Verilog Model

- Verilog model: *case sensitive*
  - is composed of text using keywords (100)
- Keywords:
  - are predefined *lowercase* identifiers that define the language constructs
  - E.g.s: **module**, **endmodule**, **input**, **output**, **wire**, **and**, **or**, **not**, …
- Comments:
  - // : single-line comment
  - /* …… */ : multiline comments
- File name: **.v**

Synthesizable Modules & Testbench

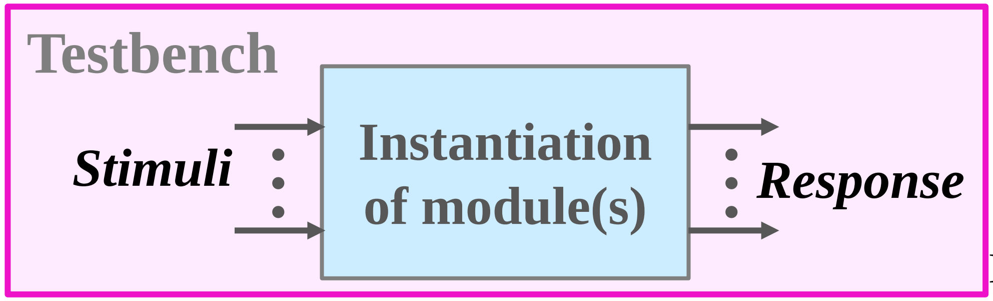
- HDL code may be divided into *synthesizable modules* and a *testbench*.
- Synthesizable modules:
  - describe the **hardware**



- Testbench:
  - contains code to apply inputs to a module, check whether the output results are correct, and print discrepancies b/t expected and actual outputs.

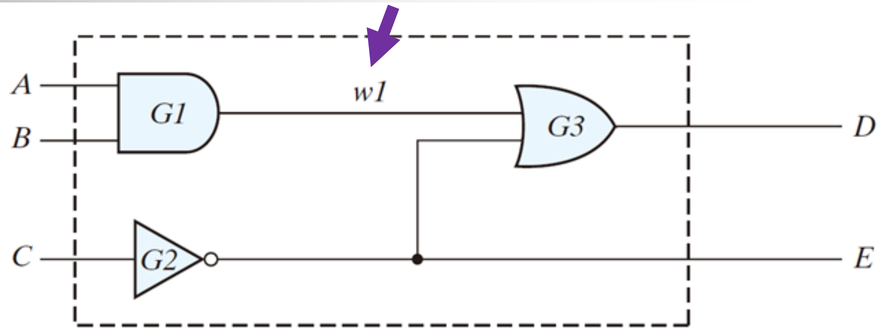
  * Testbench code is intended only for **simulation** and cannot be synthesized.

* No input/output port.



HDL Example 3.1: Gate-Level Model

- Synthesizable Module: Gate-Level Model (p.127)
  - Combinational logic modeled w/ primitives

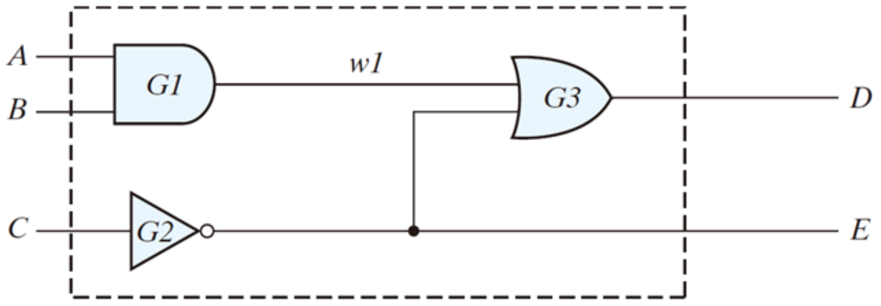


.v

```
module Simple_Circuit (A, B, C, D, E);
    output   D, E;
    input    A, B, C;
    wire     w1;

    and      G1(w1, A, B);   //Optional gate instance name
    not      G2(E, C);
    or       G3(D, w1, E);
endmodule
```

The output of a *primitive gate* is always listed first, followed by the inputs.

* Concurrence



```
module Simple_Circuit(A, B, C, D, E);
    output    D, E;
    input     A, B, C;
    wire      w1;

    and        G1(w1, A, B);
    not        G2(E, C);
    or         G3(D, w1, E);
endmodule
```

- Keywords in this example:
  - module: start the declaration (description) of a module
    - is followed by a *name* and a *list of ports*
    - The port list is the interface b/t the module and its environment.
    - The inputs and outputs of a module may be listed in any order.
  - endmodule: complete the declaration of a module
  - input, output: specify which of the ports are inputs/outputs
  - wire: declare *internal connection*
  - and, or, not: primitive gates
    - The output of a primitive gate is always listed first, followed by the inputs.

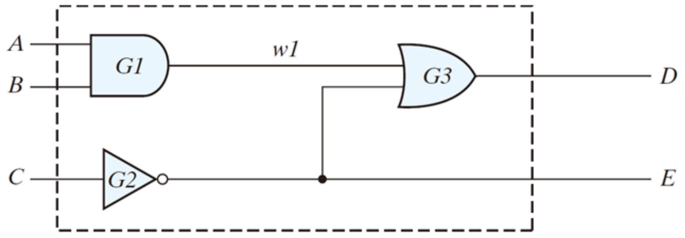
```
module Simple_Circuit(A, B, C, D, E);
    output   D, E;
    input    A, B, C;
    wire     w1;

    and       G1(w1, A, B);
    not       G2(E, C);
    or        G3(D, w1, E);
endmodule
```

- Identifiers:
  - are names given to modules, variables, and other elements of the language for reference in the design
  - are composed of alphanumeric characters and the underscore "_ ", but can not start w/ a number.
  - are *case sensitive*
  - * Choose meaningful names for modules.
- Each statement must be terminated w/ a " ; ", but not after endmodule.

Test Bench of HDL Examples 3.1

* The test bench has no input or output port ⇐ it does not interact w/ its environment.

```
module Simple_Circuit(A, B, C, D, E);
    output   D, E;
    input    A, B, C;
    wire     w1;

    and       G1(w1, A, B);
    not       G2(E, C);
    or        G3(D, w1, E);
endmodule
```



```
module t_Simple_Circuit;
    wire D, E;
    reg  A, B, C;

    //instantiate device under test
    Simple_Circuit M1 (A, B, C, D, E);

    //apply inputs one at a time
    initial
        begin
            A = 1'b0; B = 1'b0; C = 1'b0;
            #100 A = 1'b1; B = 1'b1; C = 1'b1;
        end

    initial #200 $finish;
endmodule
```

The inputs to the ckt are declared w/ **reg** & the outputs are declared w/ **wire**.

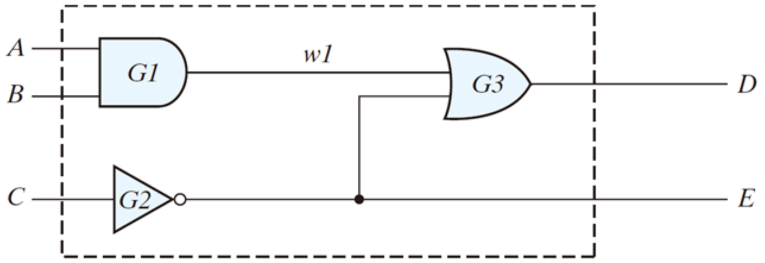
an instantiation of the model to be verified

a signal generator
* The statements are executed ***in sequence***.

.V

```
module t_Simple_Circuit;
    wire D, E;
    reg  A, B, C;

    //instantiate device under test
    Simple_Circuit_prop_delay M1 (A, B, C, D, E);

    //apply inputs one at a time
    initial
        begin
            A = 1'b0; B = 1'b0; C = 1'b0;
            #100 A = 1'b1; B = 1'b1; C = 1'b1;
        end

     initial #200 $finish;
endmodule
```

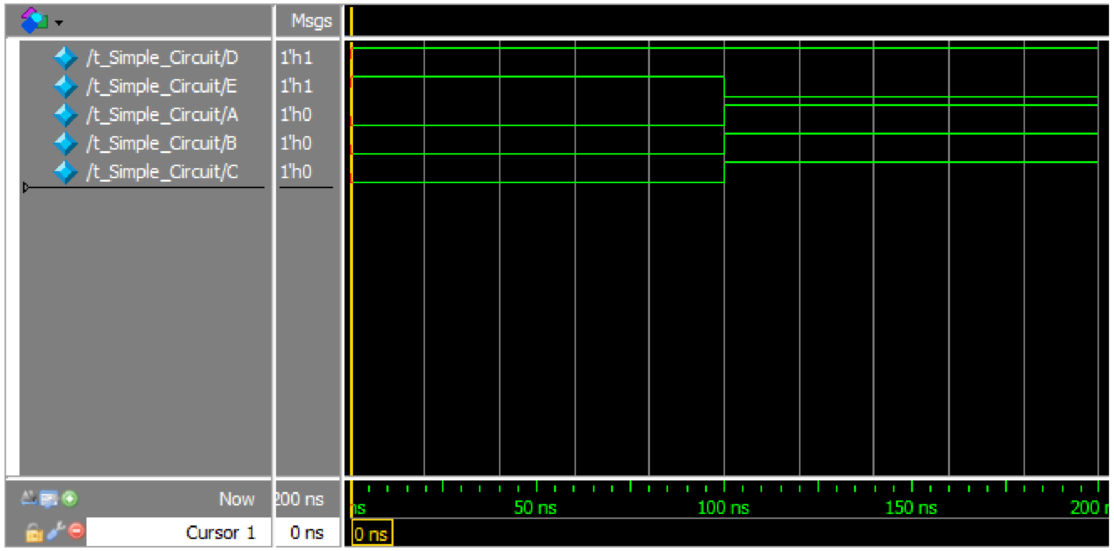
- initial statement:
  - executes the statements in its body at the *start of simulation*
  - * should be used only in ***testbenches*** for simulation
- begin, end
- $finish: terminate the simulation
- reg: declares signals in initial statements
  - Variables of type reg retain their value until they are assigned a new value by an assignment statement



```
module Simple_Circuit(A, B, C, D, E);
    output   D, E;
    input    A, B, C;
    wire     w1;

    and      G1(w1, A, B);
    not      G2(E, C);
    or       G3(D, w1, E);
endmodule
```

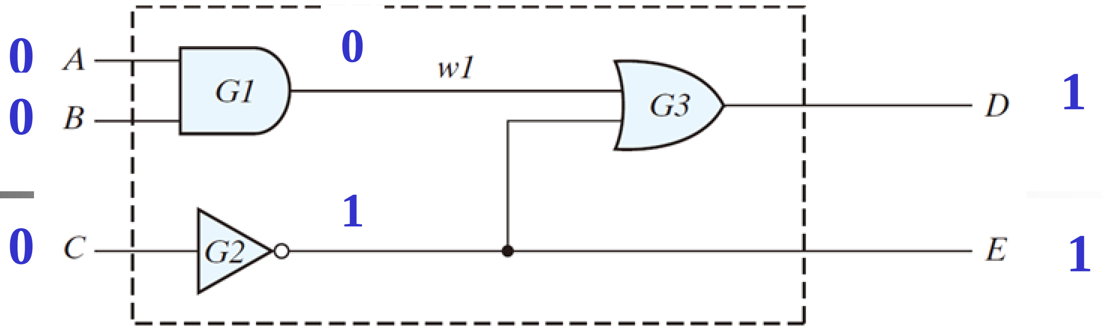
```
module t_Simple_Circuit;
    wire D, E;
    reg  A, B, C;

    //instantiate device under test
    Simple_Circuit_prop_delay M1 (A, B, C, D, E);

    //apply inputs one at a time
    initial
        begin
            A = 1'b0; B = 1'b0; C = 1'b0;
            #100 A = 1'b1; B = 1'b1; C = 1'b1;
        end

     initial #200 $finish;
endmodule
```

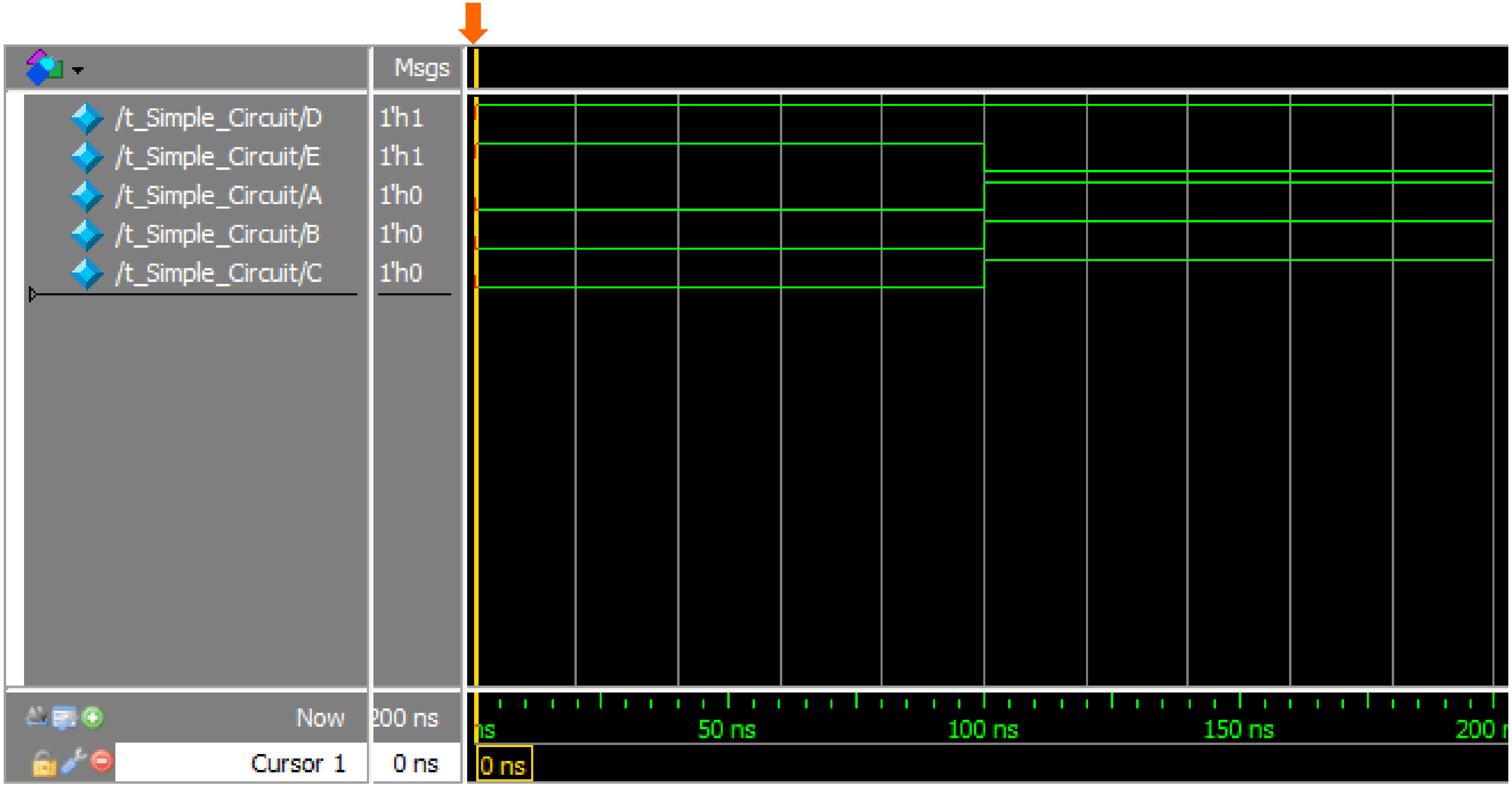
- Simulation waveforms:



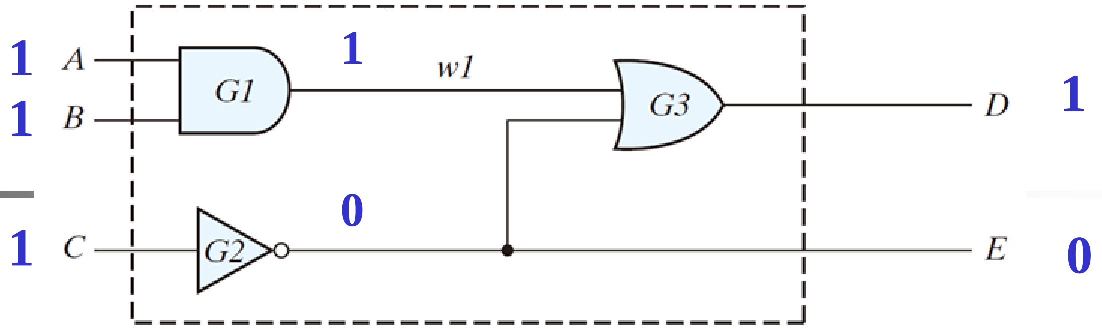
```
                      Msgs
/t_Simple_Circuit/D   1'h1
/t_Simple_Circuit/E   1'h1
/t_Simple_Circuit/A   1'h0
/t_Simple_Circuit/B   1'h0
/t_Simple_Circuit/C   1'h0

Now       200 ns
Cursor 1  0 ns
```

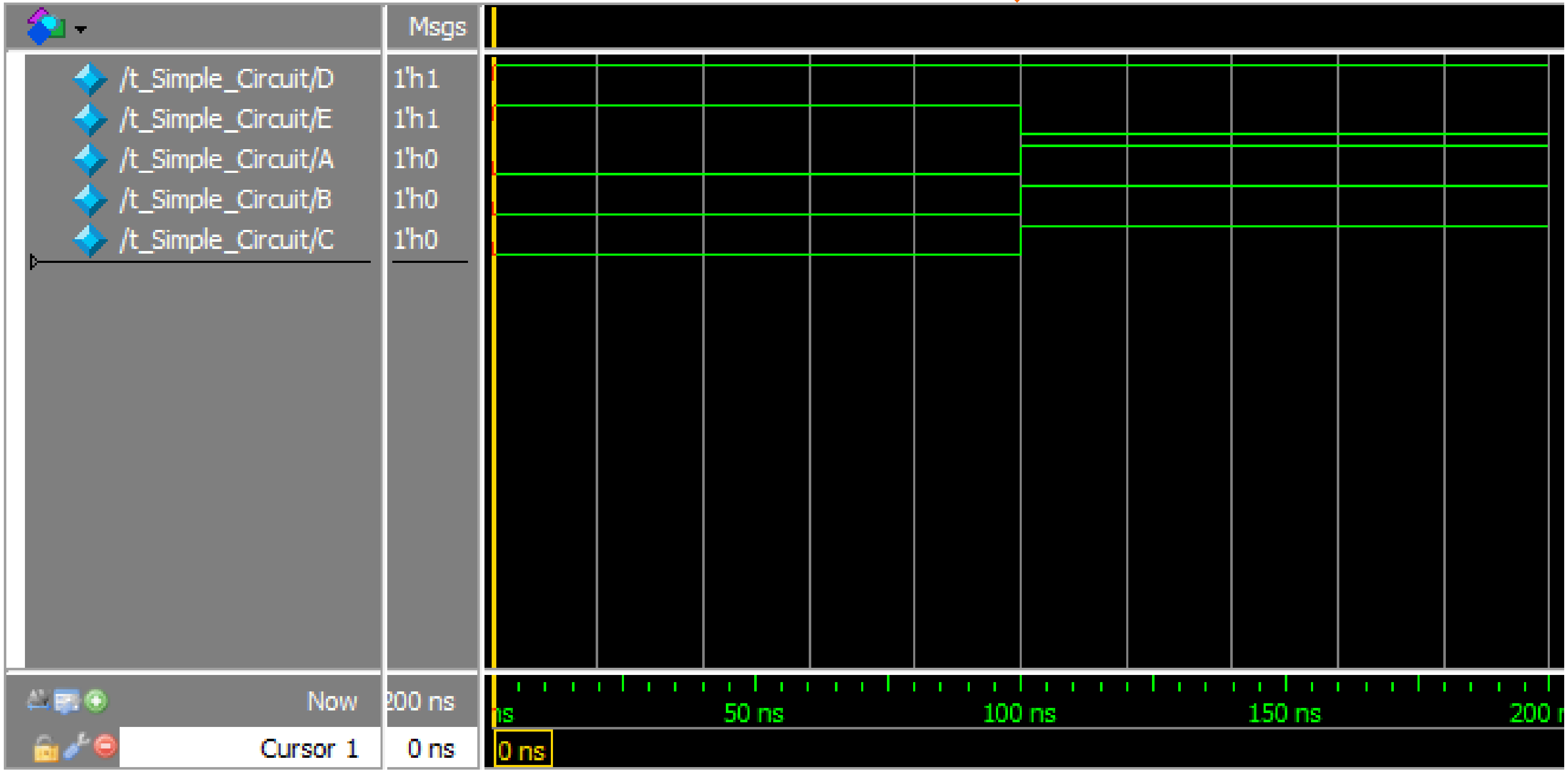
t = 0


0
A
0
B
0
C
G1
0
w1
G3
D
1
G2
1
E
1


Msgs
/t_Simple_Circuit/D
1'h1
/t_Simple_Circuit/E
1'h1
/t_Simple_Circuit/A
1'h0
/t_Simple_Circuit/B
1'h0
/t_Simple_Circuit/C
1'h0
Now
200 ns
ns
50 ns
100 ns
150 ns
200 n
Cursor 1
0 ns
0 ns

t = 100





```
/t_Simple_Circuit/D   1'h1
/t_Simple_Circuit/E   1'h1
/t_Simple_Circuit/A   1'h0
/t_Simple_Circuit/B   1'h0
/t_Simple_Circuit/C   1'h0
```