YORK
UNIVERSITÉ
UNIVERSITY

EECS 3201: Digital Logic Design Lecture 4

Ihab Amer, PhD, SMIEEE, P.Eng.

What is a HDL?

- A high-level computer language that can describe digital systems in textual form
- Two applications of HDL processing:
  - Logic Simulation
  - Logic Synthesis

HDL Applications

- Logic Simulation
  - A simulator translates the HDL description to a readable output such as *timing diagram*
  - It predicts how the hardware will work before it is actually fabricated
  - Functional errors can be corrected before actual fabrication
  - Stimulus that tests the design is called *test-bench (also written in HDL)*
- Logic Synthesis
  - Deriving the *gate-level netlist* from the HDL
  - Typically accompanied with optimization, and automated with computer software
  - Restrictions on coding style for RTL model
  - The outcome (netlist) is tool dependent

IEEE-Supported HDL's

VHDL

- VHSIC HDL
- Based on Ada
- Department of defense (DARPA) mandated language
- Generally, considered more difficult to learn

Verilog

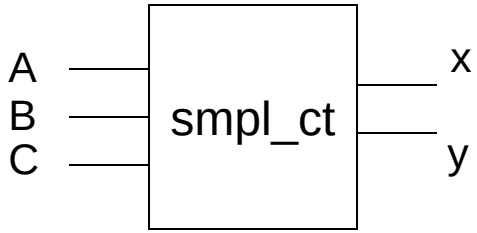
- Verify Logic
- Based on C
- Started as a Gateway Design proprietary language then later bought by Cadence
- Generally, considered easier to learn

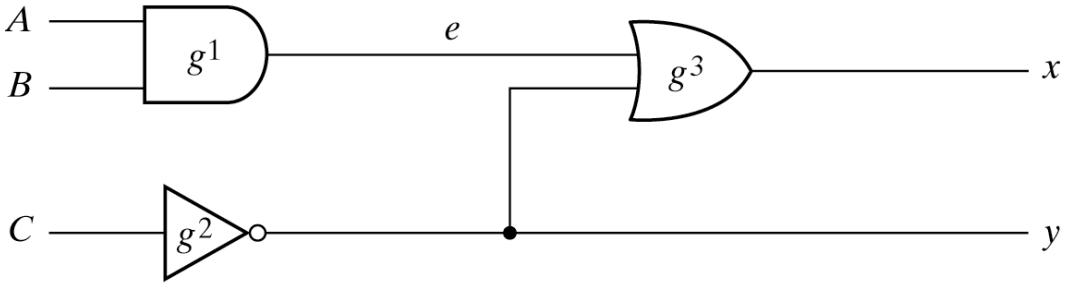
Example (Simple Circuit)

Module name *Ports names* *Punctuation*

```
module smpl_ct (A, B, C, x, y);

    input A, B, C;
    output x, y;
    wire e;

    and g1(e, A, B);
    not g2(y, C);
    or g3(x, e, y);

endmodule
```

Ports modes *Internal connection* *Gate output* *Primitive gates* *Gate inputs* *Optional gate-name*



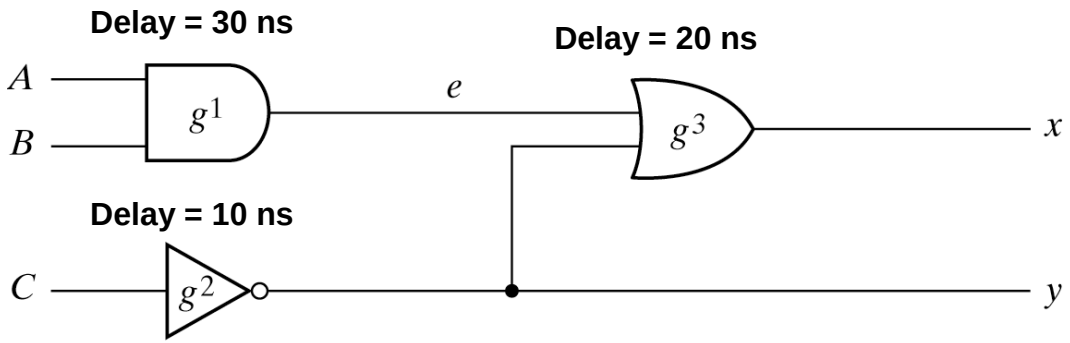


Gate Delays

```
module ct_with_delay (A, B, C, x, y);
        input A, B, C;
        output x, y;
        wire e;
        and #(30) g1(e, A, B);
        or #(20) g3(x, e, y);
        not #(10) g2(y, C);
endmodule
```

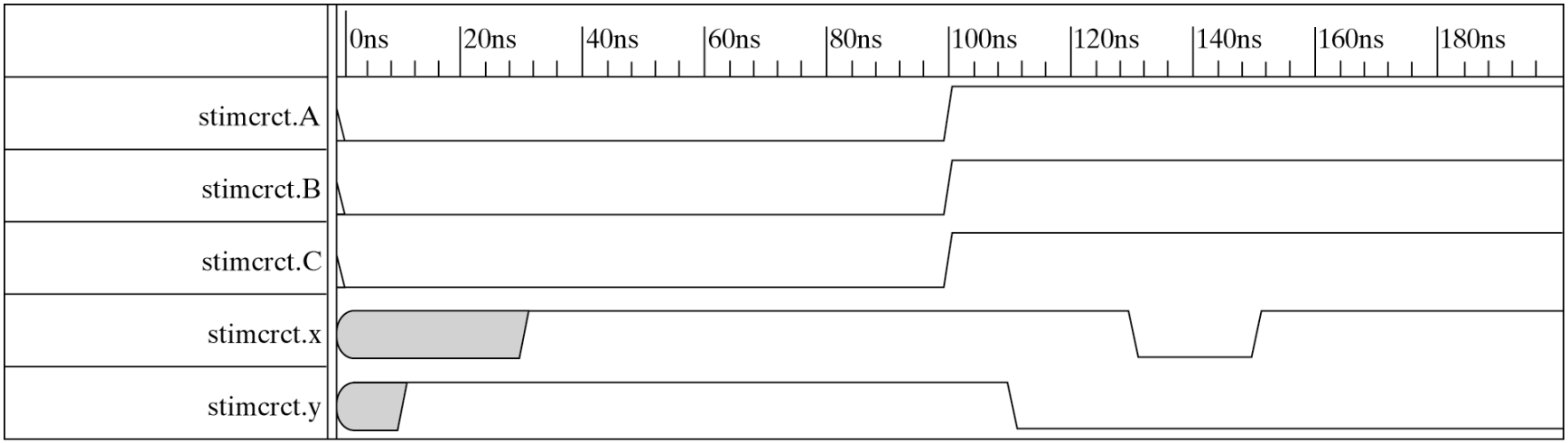
Gate delay in (ns)

Time (ns)	Input A B C	Output y e x
–	0 0 0	1 0 1
–	1 1 1	1 0 1
10	1 1 1	0 0 1
20	1 1 1	0 0 1
30	1 1 1	0 1 0
40	1 1 1	0 1 0
50	1 1 1	0 1 1



Simulation Output

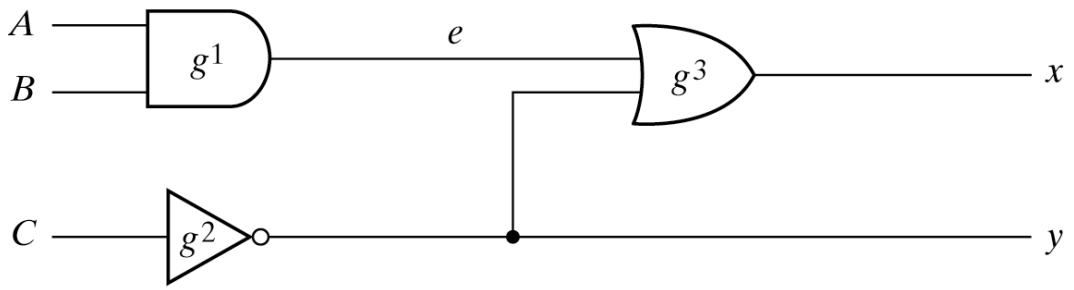
Timing Diagram



Boolean Expression

```
module ct_bln (A, B, C, x, y);

        input A, B, C;
        output x, y;

        assign x = (A & B) | ~C;

endmodule
```

Keyword *AND* *OR* *NOT*

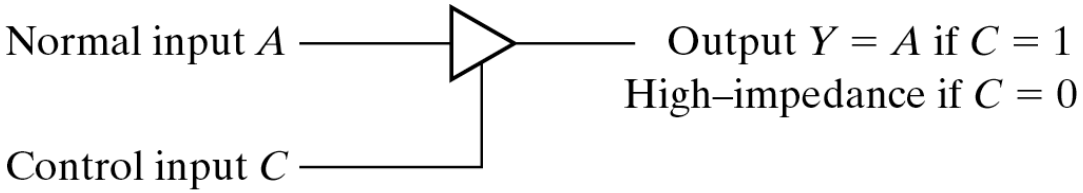


Verilog HDL Operators

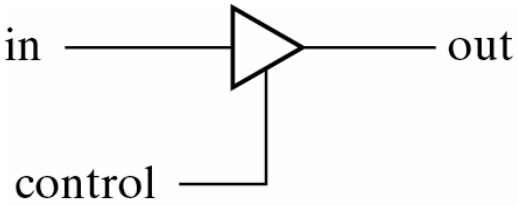
- Refer to table 4-10 of Mano textbook for a list of Verilog HDL Operators

Three-State Gates

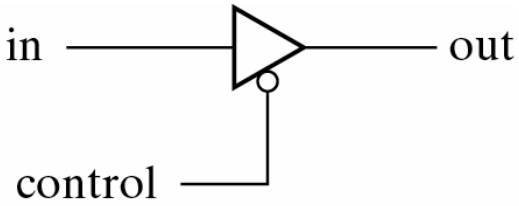
Tri-state buffer



if1 vs. if0 tri-state buffers

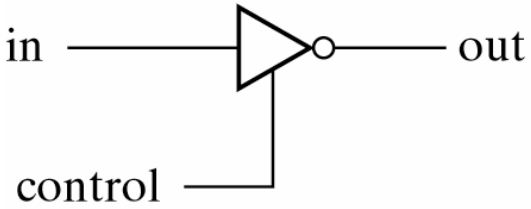


bufifl

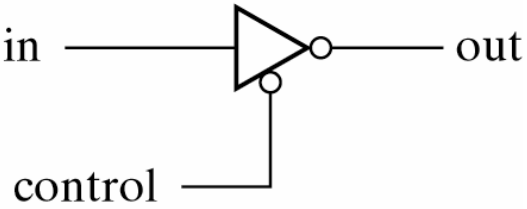


bufif0

if1 vs. if0 tri-state inverters



notifl



notif0

Four-Valued Logic

- Verilog Logic Values
  - The underlying data representation allows for any bit to have one of four values:
  - 0, 1, z (high impedance), and x (unknown)

No Question!

- A possible output from tri-state gates

- It is a real electric effect

- Not a real value

- Maybe 0, 1, z, or in the state of change

- Simulator cannot determine the value, and perhaps you should worry!

Truth Tables for Primitive Gates

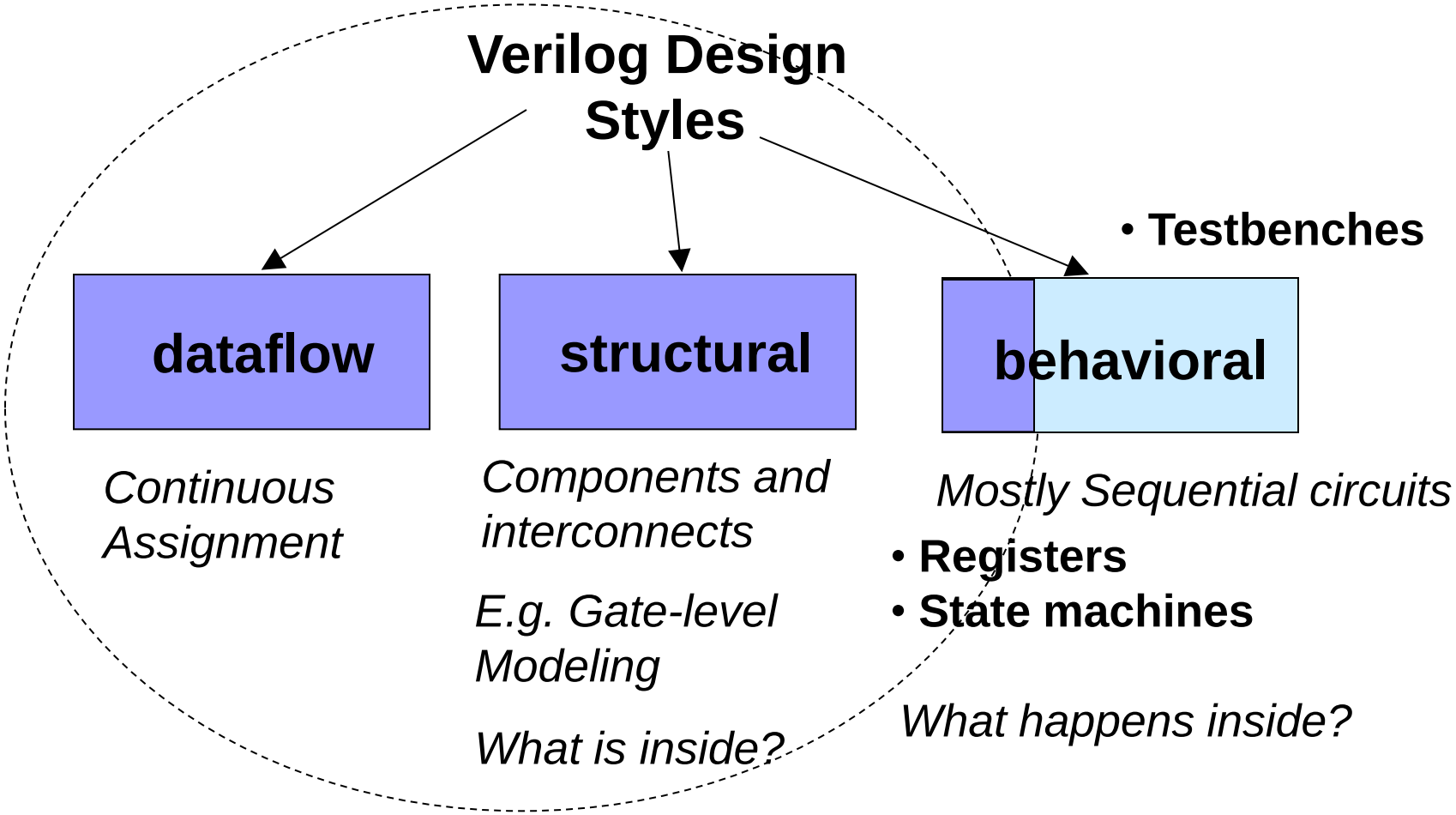
and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

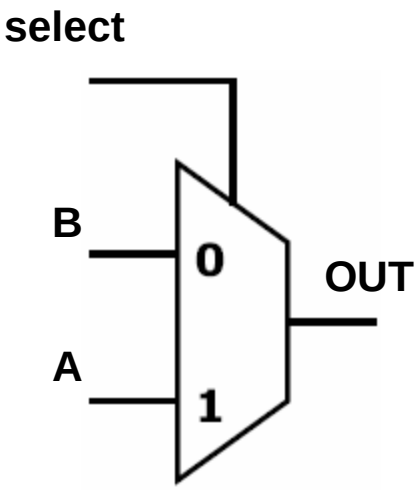
not	input	output
	0	1
	1	0
	x	x
	z	x

Verilog Design Styles



Subset most suitable for synthesis

Example – 2:1 MUX



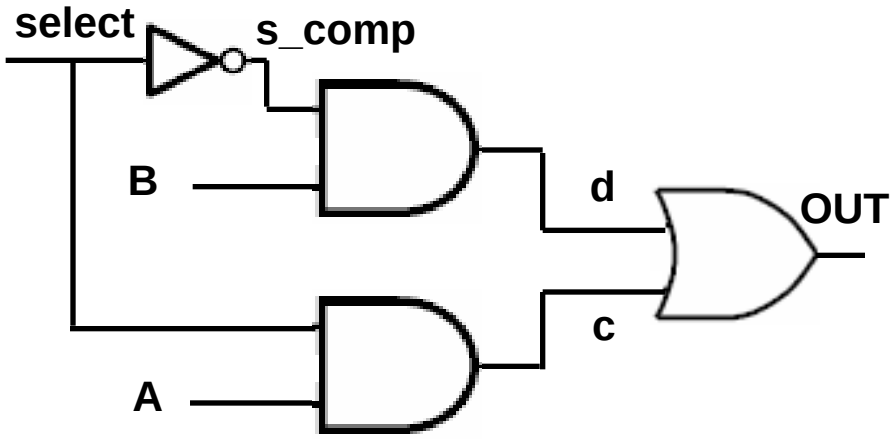
select	OUT
0	B
1	A

OUT = (A . select) + (B . select′)

Gate-Level Model

```
module mux2x1_gl (A, B, select, OUT);

        input A, B, select;
        output OUT;
        wire s_comp, c, d;

        not g1(s_comp, select);
        and g2(c,select,A);
        and g3(d,s_comp,B);
        or g4(OUT,c,d);

endmodule
```

Components and Interconnects



Dataflow Model

```
module mux2x1_df1 (A, B, select, OUT);

        input A, B, select;
        output OUT;

        assign    OUT = (A & select) | (B & ~select);

endmodule
```

Continuous Assignment

```
module mux2x1_df2 (A, B, select, OUT);

        input A, B, select;
        output OUT;

        assign    OUT = select ? A : B;

endmodule
```

Another Dataflow Model

Behavioral Model

Mostly used with sequential circuits

```
module mux2x1_bh (A, B, select, OUT);

    input A, B, select;
    output OUT;
    reg OUT;

    always @ (select or A or B)
        if (select == 1)  OUT = A;
        else OUT = B;

endmodule
```

Retains its value until a new value is assigned

Sensitivity List

Executes every time there is a change in any of the variables in the sensitivity list

Equality Symbol Can be written as: if (select) …

Procedural Assignment

Structural Design – Recap

- Structural design is the simplest to understand. This style is the closest to schematic capture and utilizes simple building blocks to compose logic functions
- Components are interconnected in a hierarchical manner
- Structural descriptions may connect simple gates (gate-level) or complex, abstract components
- Useful when expressing a design that is naturally composed of sub-blocks

Data-Flow Design – Recap

- Describes how data moves through the system and the various processing steps
- Data Flow uses series of continuous assignment statements
- Data Flow is most useful style when series of Boolean equations can represent a logic

Behavioral Design – Recap

- It accurately models what happens on the inputs and outputs of the black box (no matter what is inside and how it works)
- This style uses *always* statements in *Verilog*
- Procedural statements in an *always* block executes sequentially. However, the *always* block itself executes concurrently with other concurrent statements in the same module (instances, continuous assignments, and other *always* statements)
- Typically used for test-benches or high-level implementations to drive logic synthesis tools

Nets, Variables, Parameters, and Directives

- Net: Physical wire between modules
  - A *wire* is the most commonly-used net
- Variable: Stores a value during a Verilog program's execution, and needs not have physical significance in a circuit
  - A *reg* is the most commonly-used variable
- Parameter: A facility provided by Verilog for defining named constants within a module, to improve readability and maintainability
  - E.g. parameter ESC = 7′b0011011;
- Directive: To control the compilation process
  - ′include and ′define are the most commonly-used directives

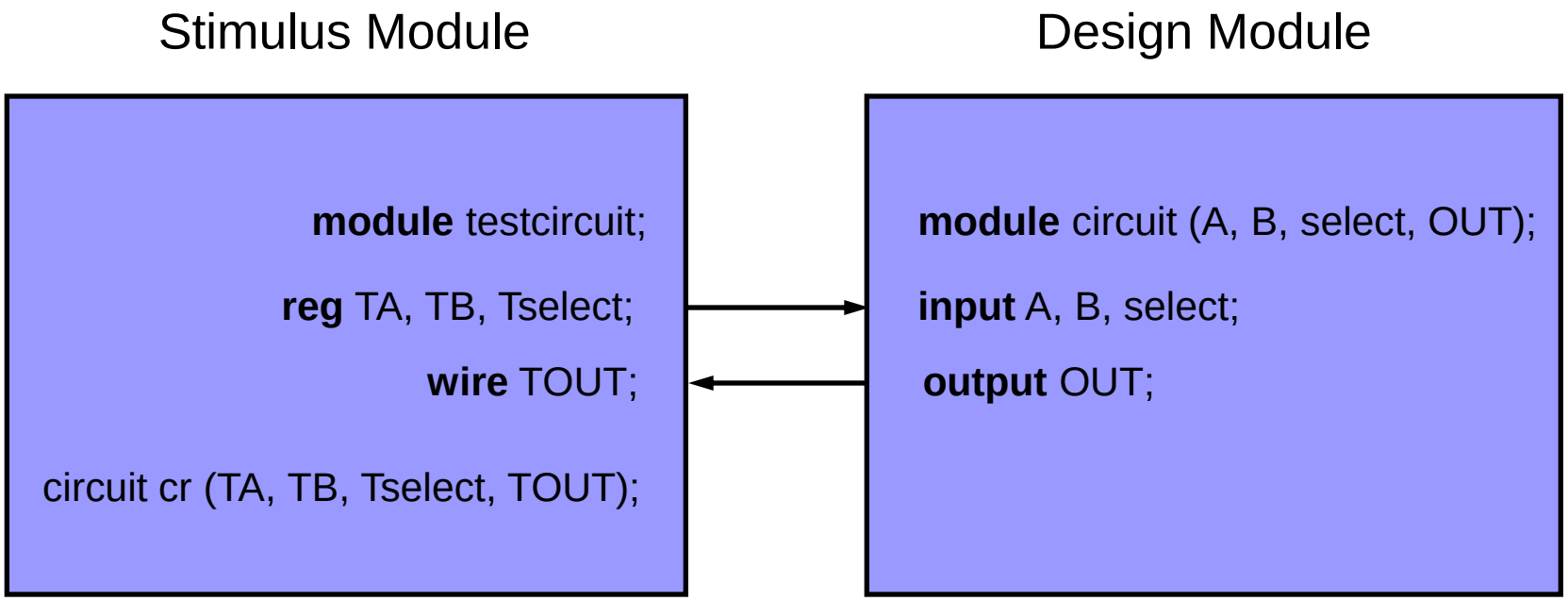
Logical Vs Bitwise Operators

- Examples of Ambiguities:
  - (2′b01 && 2′b10) Vs (2′b01 & 2′b10)
  - !(5) Vs ~(5)

Ok… Design is done… How should I test it?

- Same as what you would do to test a SW program:
  - Give it some inputs, and see if it does what you expect
  - After testing, do you guarantee that the program is bug free? NO!
  - But, to the extent possible, you have determined that the program does what you want it to do
- Same happens in HW design, you *simulate* the system's behavior with some input stimulus

Test Bench


Stimulus Module
Design Module
module testcircuit;
reg TA, TB, Tselect;
wire TOUT;
circuit cr (TA, TB, Tselect, TOUT);
module circuit (A, B, select, OUT);
input A, B, select;
output OUT;

I am sick of this MUX!!

```
module stimcrct;
        reg A, B, select;
        wire OUT;
        mux2x1_df2 mux (A, B, select, OUT);
        initial
            begin
                A = 1'b0; B = 1'b1; select = 1'b0;
                #100
                A = 1'b0; B = 1'b1; select = 1'b1;
                #100    $finish;
            end
endmodule
```

Instance of design module

Procedural Assignment – used with "reg"

Executes only once at t = 0

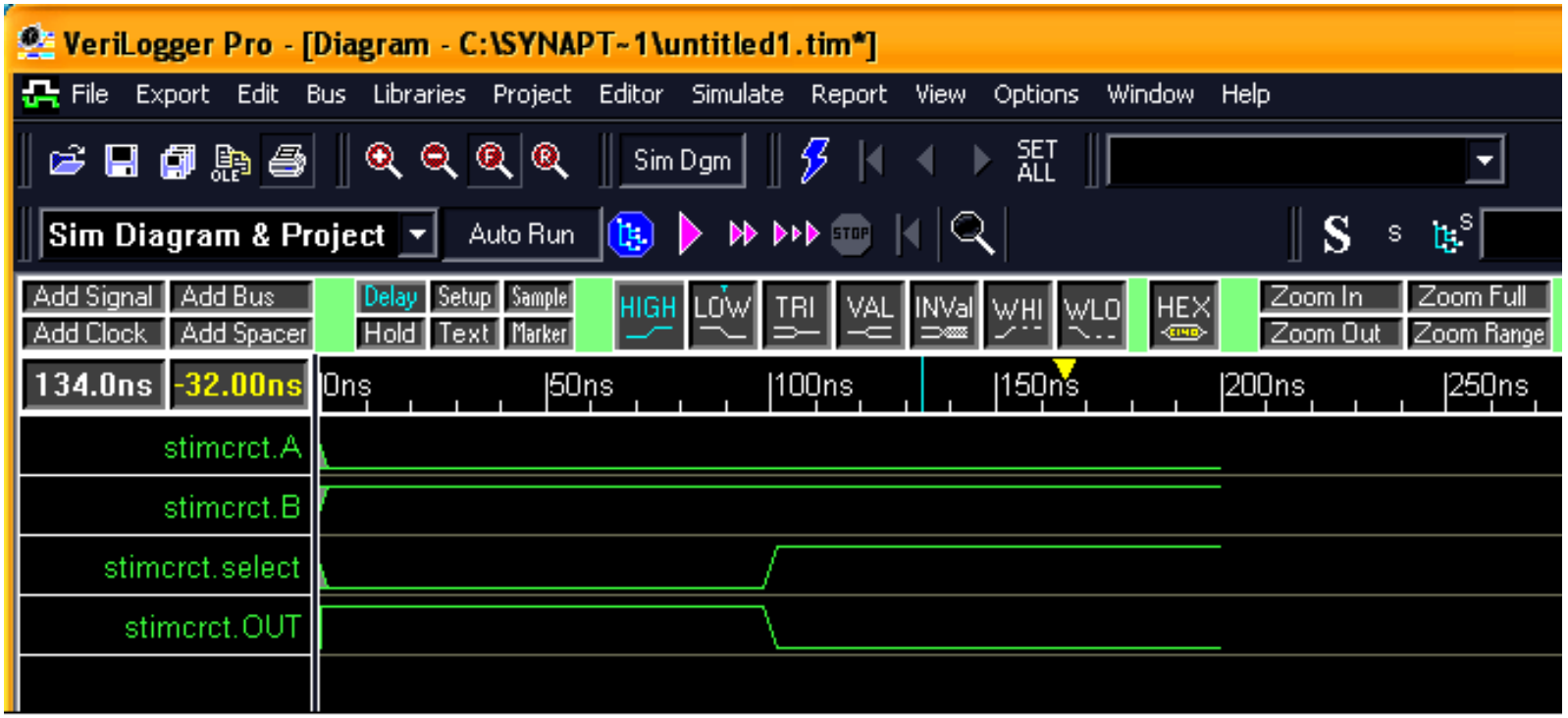
Terminates simulation

Stimulus Module

```
module mux2x1_df2 (A, B, select, OUT);
        input A, B, select;
        output OUT;
        assign   OUT = select ? A : B;
endmodule
```

Design Module

Simulation Output



Examples of Stimulus Generation

```
initial
    begin
                    A = 0; B = 0;
        #10         A = 1;
        #20         A = 0; B = 1;
    end
```

```
initial
    begin
                    D = 3'b000;
                    repeat (7)
        #10         D = D + 3'b001;
    end
```

3-bits Truth Table

References

- Lecture Notes of Dr. Sebastian Magierowski – Fall 2013
- Digital Design, 3rd Edition, M. Morris, Mano
- Digital Design, 4th Edition, John Wakerly
- cpk.auc.dk/education/SSU-2007/mm10/ssu_mm10.pdf
- www.ece.cmu.edu/~thomas/VSLIDES.pdf
- http://ece.gmu.edu/coursewebpages/ECE/ECE448/S10/