Introducción a SystemVerilog

Arquitectura de Computadoras 2021

HDL to Gates

Simulation

- Inputs applied to circuit
- Outputs checked for correctness
- Millions of dollars saved by debugging in simulation instead of hardware

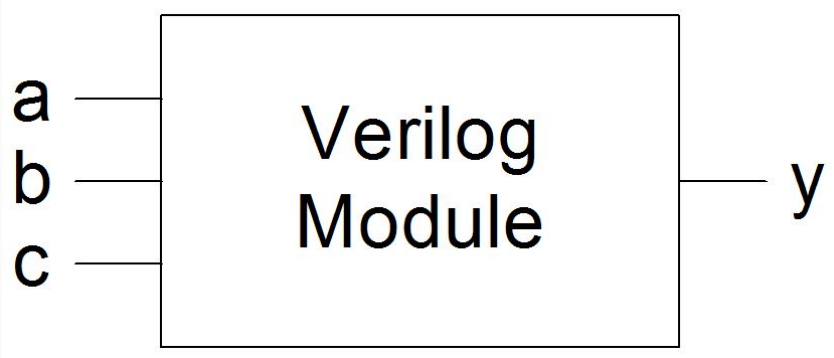
Synthesis

- Transforms HDL code into a netlist describing the hardware (i.e., a list of gates and the wires connecting them)

SystemVerilog Modules

Two types of Modules:

- Structural: describe how it is built from simpler modules
- Behavioral: describe what a module does



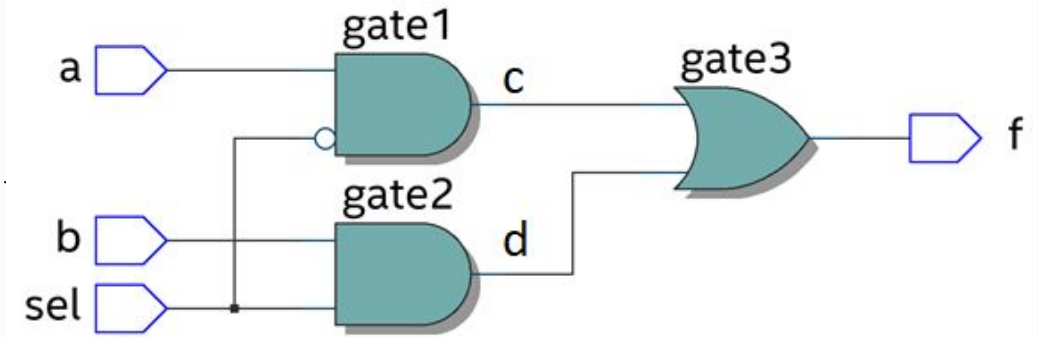
SystemVerilog Modules - Structural type

```
// multiplexor
module smux (input  logic a, b, sel,
             output logic f);
    logic c, d, not_sel;
    not gate0(not_sel, sel);
    and gate1(c, a, not_sel);
    and gate2(d, b, sel);
    or gate3(f, c, d);
endmodule
```

* Built-in gates

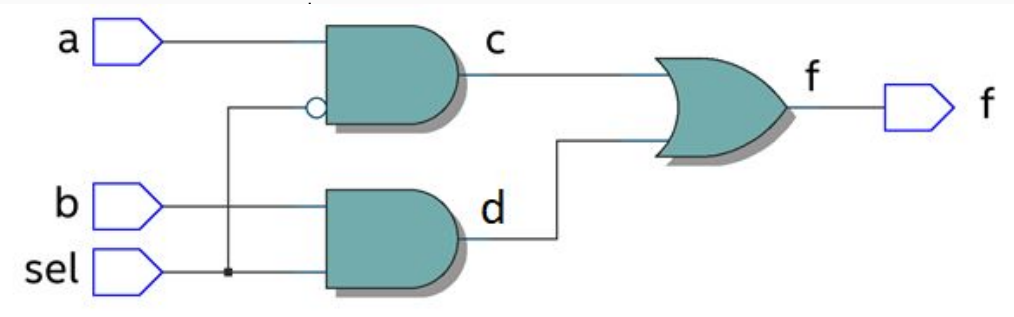
Port order is:

output, input(s)



SystemVerilog Modules - Behavioral type

```
// multiplexor
module bmux (input  logic a, b, sel,
                    output logic f);
    logic c, d;
    assign c = a & (~sel);
    assign d = b & sel;
    assign f = c | d;
// or alternatively
//assign f = sel ? b : a;
endmodule
```



Behavioral SystemVerilog

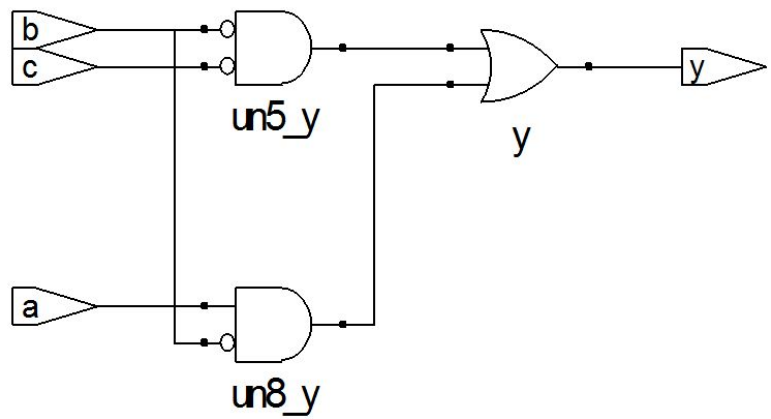
Behavioral modeling: describing a module in terms of the relationships between inputs and outputs.

- `module/endmodule`: required to begin/end module.
- Module begins with the name (`example`) and a listing of the inputs/outputs.
- `Logic` signals such as the inputs and outputs are Boolean variables (0 or 1). They may also have floating and undefined values.
- The `assign` statement describes combinational logic.

```
// Boolean function y = a'b'c'+ ab'c+ ab'c
module example(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

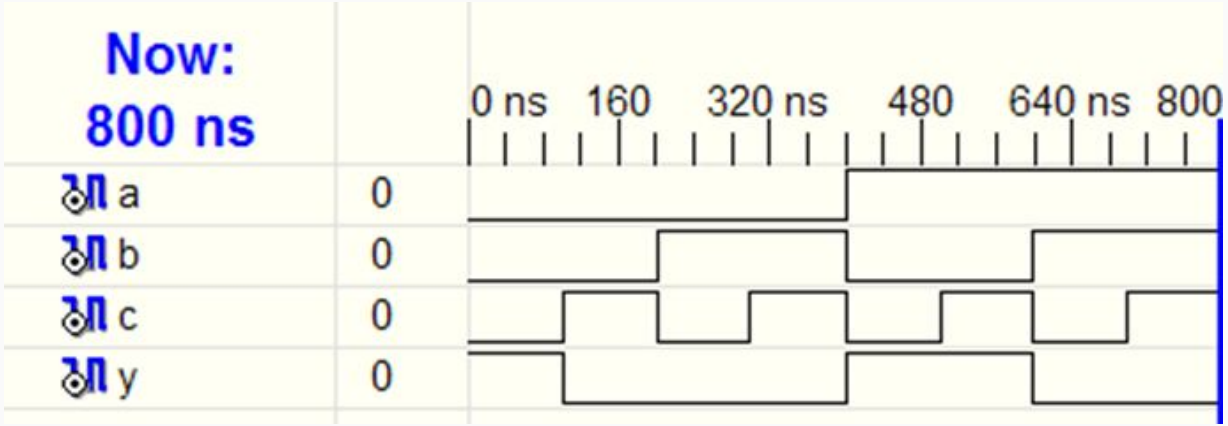
HDL Synthesis

```
// Boolean function y = a'b'c'+ ab'c+ ab'c
module example(input  logic a, b, c,
                      output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```



HDL Simulation

```
// Boolean function y = a'b'c'+ ab'c+ ab'c
module example(input  logic a, b, c,
                      output logic y);
  assign y = ~a&~b&~c | a&~b&~c | a&~b&c;
endmodule
```



a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

SystemVerilog Syntax

- **Case sensitive**
  Example: reset and Reset are not the same signal.

- **No names that start with numbers**
  Example: 2mux is an invalid name.

- **Whitespace ignored**

- **Comments:**

```
// single line comment
/* multiline
   comment */
```

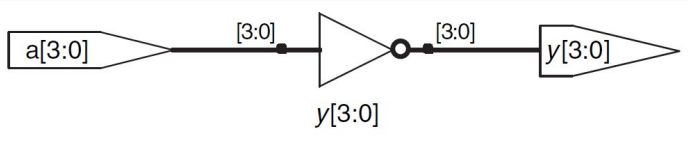
SystemVerilog Operators and precedence

	Op	Meaning
Highest	~	NOT
	*, /, %	MUL, DIV, MOD
	+, -	PLUS, MINUS
	<<, >>	Logical Left/Right Shift
	<<<, >>>	Arithmetic Left/Right Shift
	<, <=, >, >=	Relative Comparison
	==, !=	Equality Comparison
	&	AND
	^	XOR
	|	OR
Lowest	?:	Conditional

Bitwise Operators

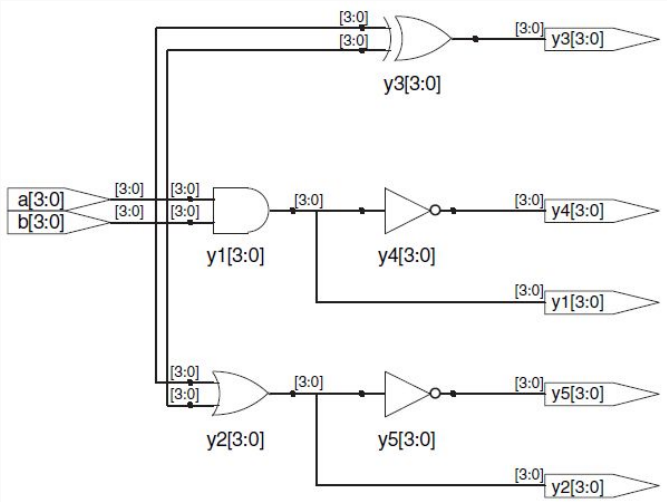
- Bitwise operators act on single-bit signals or on multi-bit busses.
- a[3:0] represents a 4-bit bus. The bits, from most significant to least significant, are a[3], a[2], a[1] and a[0]. The least significant bit has the smallest bit number.

```
module inv  (input logic [3:0] a,
             output logic [3:0] y);
   assign y = ~a;
endmodule
```



Bitwise Operators

```
module gates(input  logic [3:0]  a, b,
                   output logic [3:0] y1, y2, y3, y4,
y5);
   /* Five different two-input logic
    gates acting on 4 bit busses */
   assign y1 = a & b; // AND
   assign y2 = a | b; // OR
   assign y3 = a ^ b; // XOR
   assign y4 = ~(a & b);  // NAND
   assign y5 = ~(a | b);  // NOR
endmodule
```

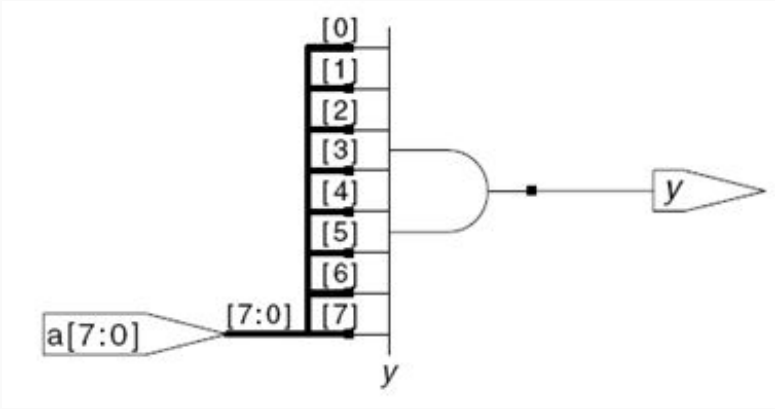


- Anytime the inputs on the right side of the = in a continuous assignment statement change, the output on the left side is recomputed. Thus, continuous assignment statements describe combinational logic.

Reduction Operators

- Reduction operators imply a multiple-input gate acting on a single bus.

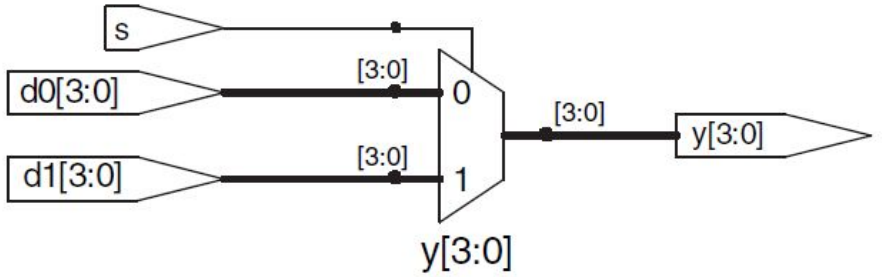
```
module and8(input  logic [7:0] a,
            output logic   y);
   assign y = &a;
// &a is much easier to write than
// assign y = a[7]&a[6]&a[5]&a[4]&
//            a[3]&a[2]&a[1]&a[0];
endmodule
```



Conditional Assignment

- Conditional assignments select the output from among alternatives based on an input called the condition.
- The conditional operator **? :** is called a ternary operator, because it takes three inputs

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);
   assign y = s ? d1 : d0;
endmodule
```



Nested conditional operators

```
// 4:1 multiplexer
module mux4 (input logic [3:0] d0, d1, d2, d3,
             input logic [1:0] s,
             output logic [3:0] y);

   assign y = s[1] ? (s[0] ? d3 : d2)
                   : (s[0] ? d1 : d0);
endmodule

/* If s[1] is 1, then the multiplexer chooses the first
expression, (s[0] ? d3 : d2). This expression in turn
chooses either d3 or d2 based on s[0] (y = d3 if s[0] is
1 and d2 if s[0] is 0). If s[1] is 0, then the
multiplexer similarly chooses the second expression,
which gives either d1 or d0 based on s[0]. */
```

Numbers

Format: `N'Bvalue`

- **N** = number of bits, **B** = base.
- SystemVerilog supports **'b** for binary, **'o** for octal, **'d** for decimal and **'h** for hexadecimal.
- **N'B** is optional but recommended (default is decimal).
- **'0** and **'1**: filling a bus with all 0s and all 1s, respectively.

- **z**: indicate a floating value.
- **x**: indicate an invalid logic level.

Bibliografía

- S. Harris and D. Harris, “Digital Design and Computer Architecture - ARM Edition”. Elsevier, 2016.

- S. Sutherland, S. Davidmann, P. Flake and P. Moorby, “SystemVerilog for Design - A Guide to Using SystemVerilog for Hardware Design and Modeling”, Second Edition. Springer, 2006.