Verilog

Harris & Harris, Digital Design and Computer Architecture

Ch. 4 - Hardware Description Languages

Introduction

- Hardware description language (HDL):
  - specifies logic function only
  - Computer-aided design (CAD) tool produces or *synthesizes* the optimized gates
- Most commercial designs built using HDLs
- Two leading HDLs:
  - **SystemVerilog**
    - developed in 1984 by Gateway Design Automation
    - IEEE standard (1364) in 1995
    - Extended in 2005 (IEEE STD 1800-2009)
  - **VHDL 2008**
    - Developed in 1981 by the Department of Defense
    - IEEE standard (1076) in 1987
    - Updated in 2008 (IEEE STD 1076-2008)

HDL to Gates

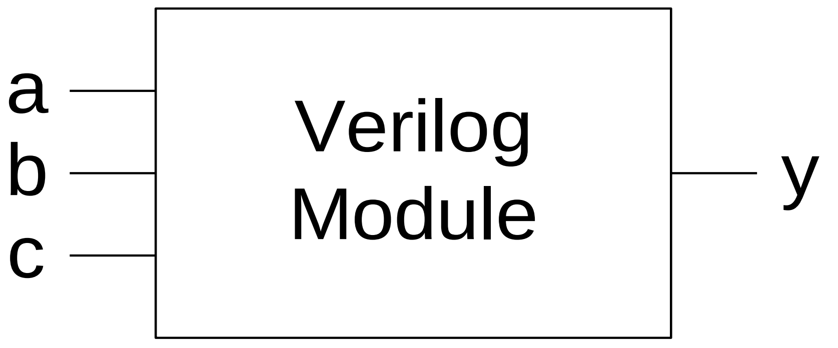
- **Synthesis**
  - Transforms HDL code into a ***netlist*** describing the hardware (i.e., a list of gates and the wires connecting them)
  - The logic synthesizer might perform **optimizations to reduce the amount of hardware required**
  - The netlist may be a text file, or it may be drawn as a schematic to help visualize the circuit
- **Simulation**
  - **Inputs** applied to circuit and **Outputs** checked for correctness
  - Millions of dollars saved by debugging in simulation instead of hardware

HDL to Gates

IMPORTANT: When using an HDL, think of the **hardware** the HDL should produce

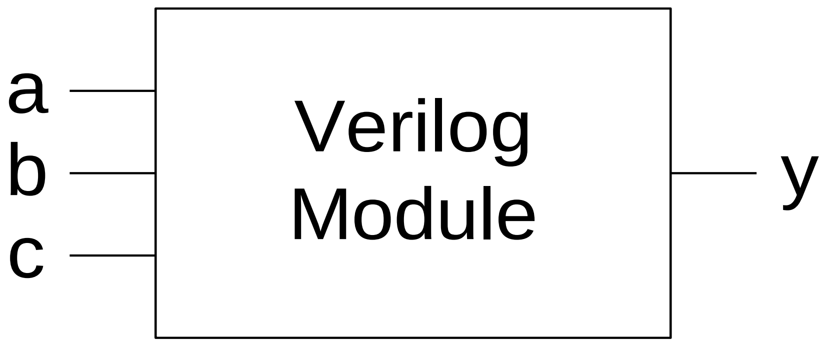
- Think of your system in terms of **blocks of combinational logic**, **registers**, and **finite state machines**
- Sketch these blocks on paper and show how they are connected before you start writing code
- The best way to learn an HDL is by example
- HDLs have specific ways of describing various classes of logic

SystemVerilog Modules



- A block of hardware with inputs and outputs is called a module
- An AND gate, a multiplexer, and a priority circuit are all examples of hardware modules

SystemVerilog Modules



Two types of Modules:

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

Behavioral SystemVerilog

SystemVerilog:

```
module example(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

Behavioral SystemVerilog

SystemVerilog:

```
module example(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

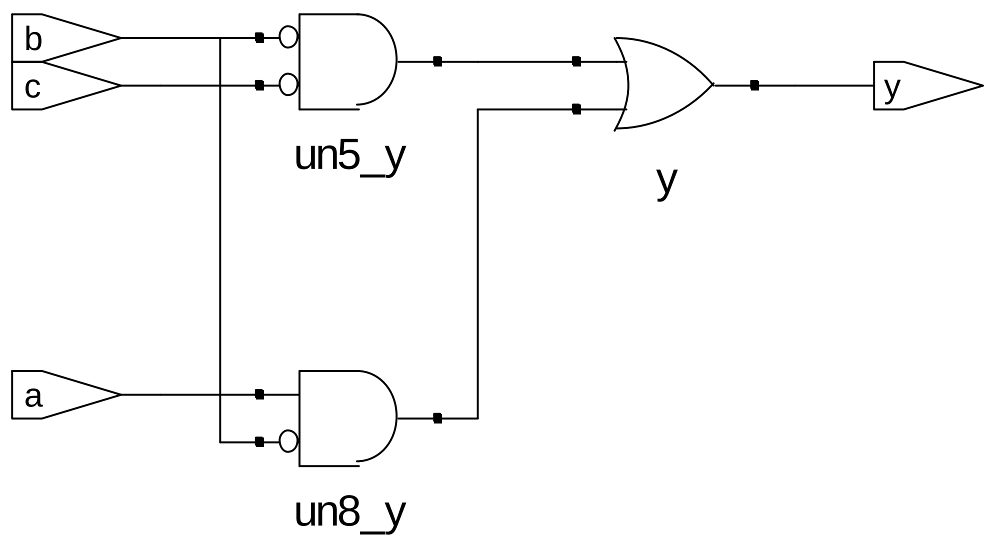
- `module/endmodule`: required to begin/end module
- `example`: name of the module
- Operators:
  - `~`: NOT
  - `&`: AND
  - `|`: OR

HDL Synthesis

SystemVerilog:

```
module example(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

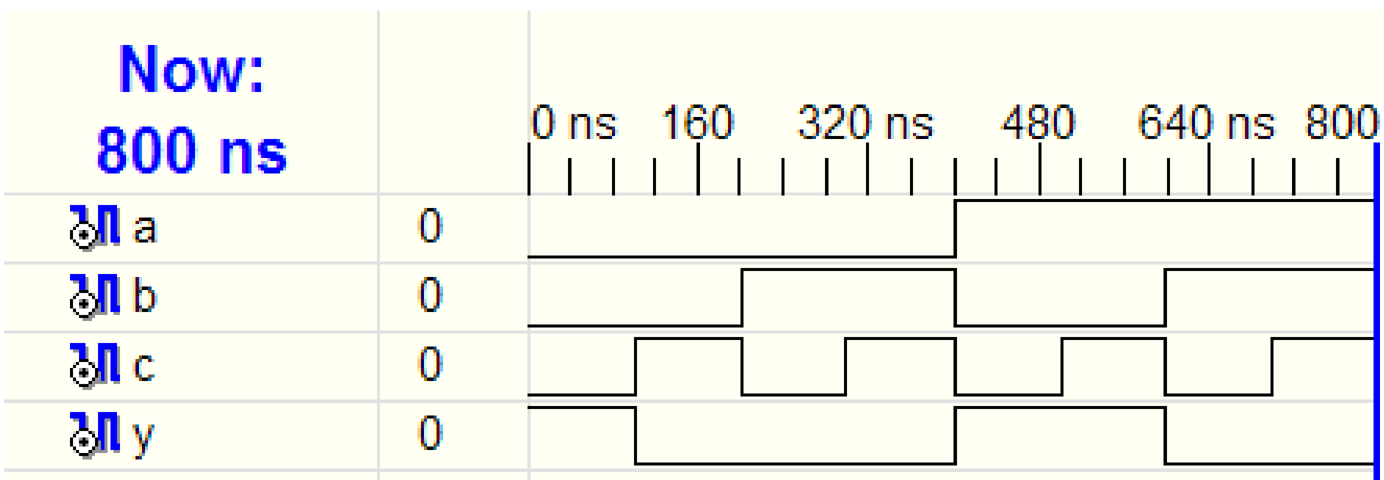
Synthesis:



HDL Simulation

SystemVerilog:

```
module example(input  logic a, b, c,
                output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```



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 */`

Structural Modeling - Hierarchy

```
module and3(input  logic a, b, c,
            output logic y);
  assign y = a & b & c;
endmodule
---------
module inv(input  logic a,
           output logic y);
  assign y = ~a;
endmodule
---------
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                   // internal signal

  and3 andgate(a, b, c, n1);  // instance of and3
  inv  inverter(n1, y);       // instance of inv
endmodule
```

Structural Modeling - Hierarchy

```
module and3(input  logic a, b, c,
            output logic y);
  assign y = a & b & c;
endmodule
---------
module inv(input  logic a,
           output logic y);
  assign y = ~a;
endmodule
---------
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                   // internal signal

  and3 andgate(a, b, c, n1);  // instance of and3
  inv  inverter(n1, y);       // instance of inv
endmodule
```

internal signal are neither inputs nor outputs, but are used only internal to the module (similar to local variables in programming languages)

Bitwise Operators

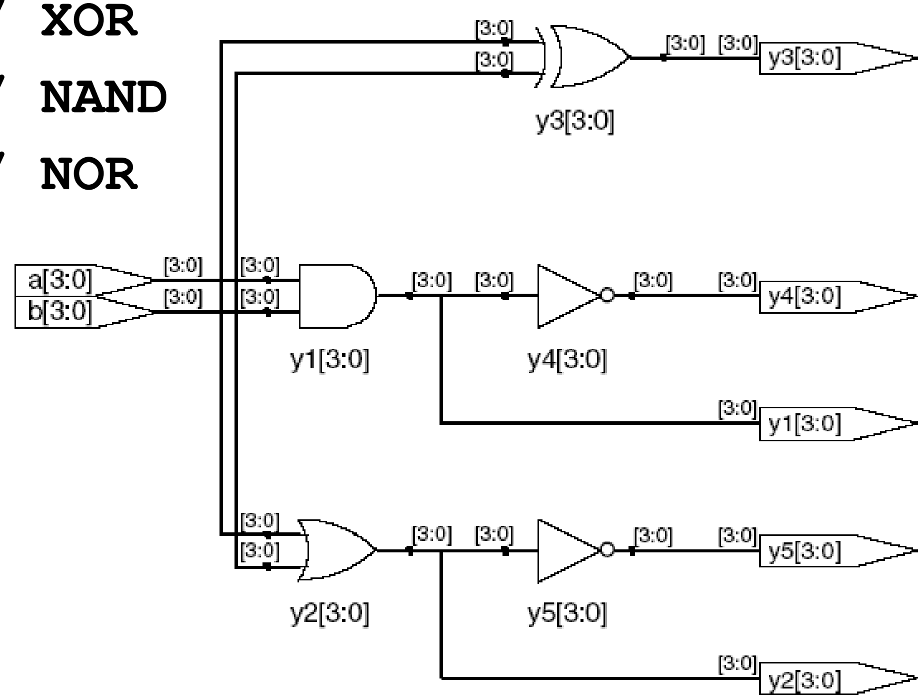
```
module gates(input  logic [3:0] a, b,
                    output logic [3:0] y1, y2, y3, y4, y5);
   /* Five different two-input logic
      gates acting on 4 bit busses */
   assign y1 = a & b;    // AND
   assign y2 = a | b;    // OR
   assign y3 = a ^ b;    // XOR
   assign y4 = ~(a & b); // NAND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

Bitwise operators act on single-bit signals or on multi-bit busses

a[3:0] rappresenta un bus a 4 bit denominati dal più significativo al meno significativo **a[3] a[2] a[1] a[0]**
Si può denominare il bus **a[4:1]** oppure **a[0:3]** e usare gli indici di conseguenza

Bitwise Operators

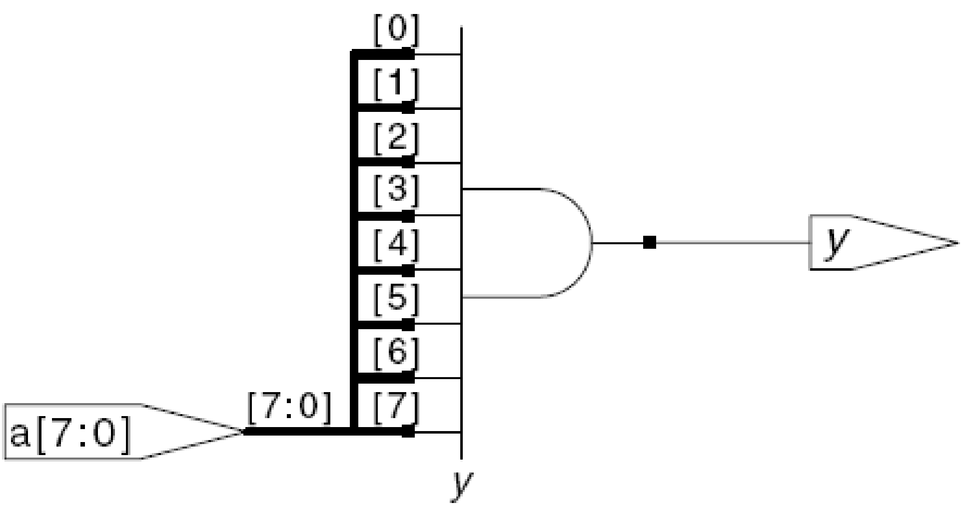
```
module gates(input  logic [3:0]  a, b,
             output logic [3:0] y1, y2, y3, y4, y5);
   /* Five different two-input logic
      gates acting on 4 bit busses */
   assign y1 = a & b;    // AND
   assign y2 = a | b;    // OR
   assign y3 = a ^ b;    // XOR
   assign y4 = ~(a & b); // NAND
 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



Reduction Operators

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

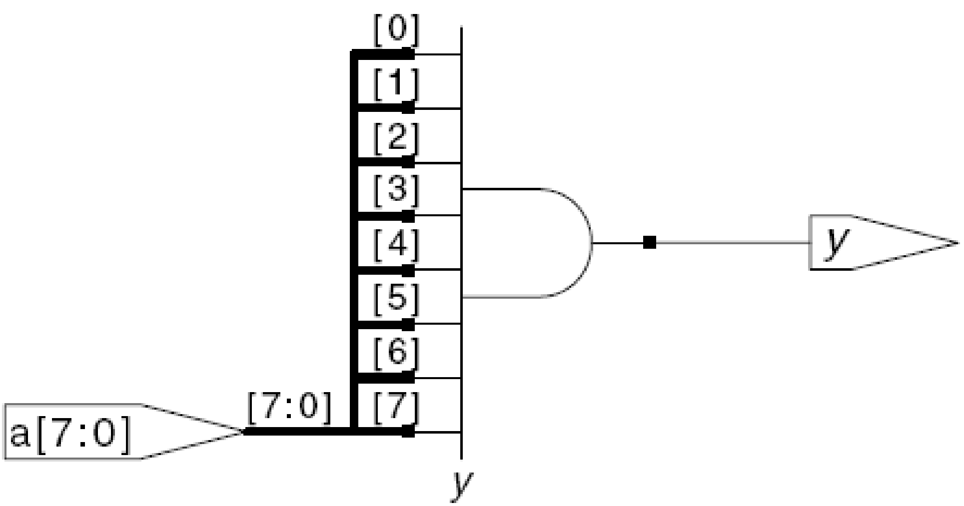
```
module and8(input  logic [7:0] a,
            output logic       y);
   assign y = &a;
   // &a is much easier to write than
   // assign y = a[7] & a[6] & a[5] & a[4] &
   //            a[3] & a[2] & a[1] & a[0];
endmodule
```



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
```

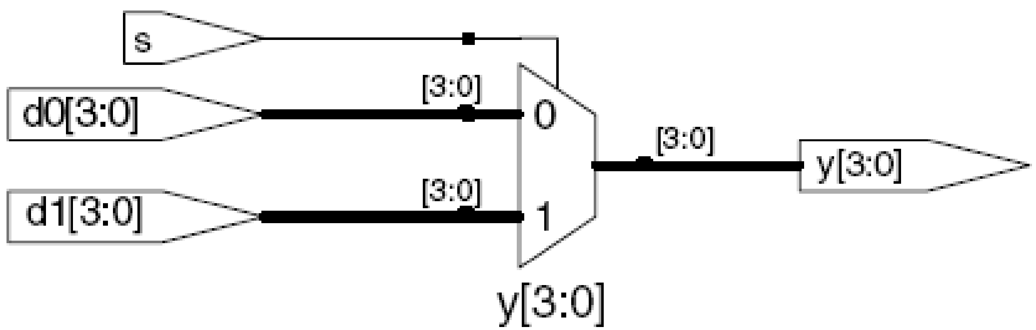


- Eight-input AND gate with inputs a7, a6, . . . , a0
- Reduction operators exist for OR, XOR, NAND, NOR, and XNOR gates
- Note that a multiple-input XOR performs parity: TRUE if an odd number of inputs are TRUE

Conditional Assignment

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

Select the output from among alternatives based on an input called the condition

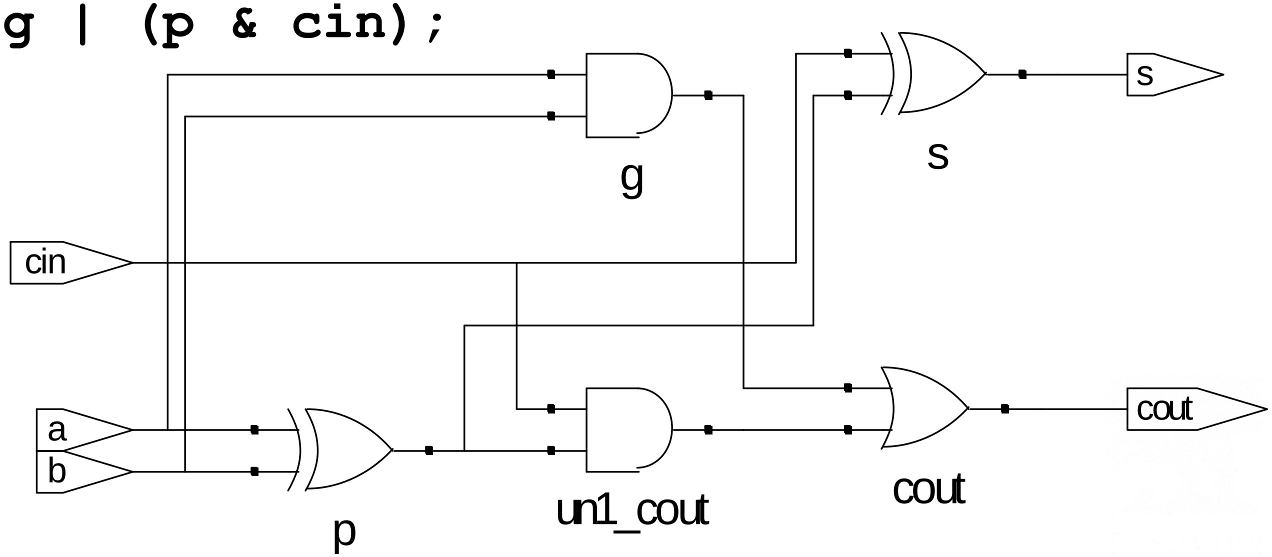


? : is also called a *ternary operator* because it operates on 3 inputs: s, d1, and d0

? : chooses, based on a first expression, between a second and third expression

Internal Variables

```
module fulladder(input  logic a, b, cin,
                 output logic s, cout);
  logic p, g;   // internal nodes

  assign p = a ^ b;
  assign g = a & b;

  assign s = p ^ cin;
  assign cout = g | (p & cin);
endmodule
```



Precedence

Highest	~	NOT
	*, /, %	mult, div, mod
	+, -	add,sub
	<<, >>	shift
	<<<, >>>	arithmetic shift
	<, <=, >, >=	comparison
	==, !=	equal, not equal
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	ternary operator

Numbers

Numbers can be specified in binary, octal, decimal, or hexadecimal

Format: N'Bvalue

N = number of bits, **B** = base

N'B is optional but recommended (default is decimal)

Number	# Bits	Base	Decimal Equivalent	Stored
3'b101	3	binary	5	101
'b11	unsized	binary	3	00…0011
8'b11	8	binary	3	00000011
8'b1010_1011	8	binary	171	10101011
3'd6	3	decimal	6	110
6'o42	6	octal	34	100010
8'hAB	8	hexadecimal	171	10101011
42	Unsized	decimal	42	00…0101010

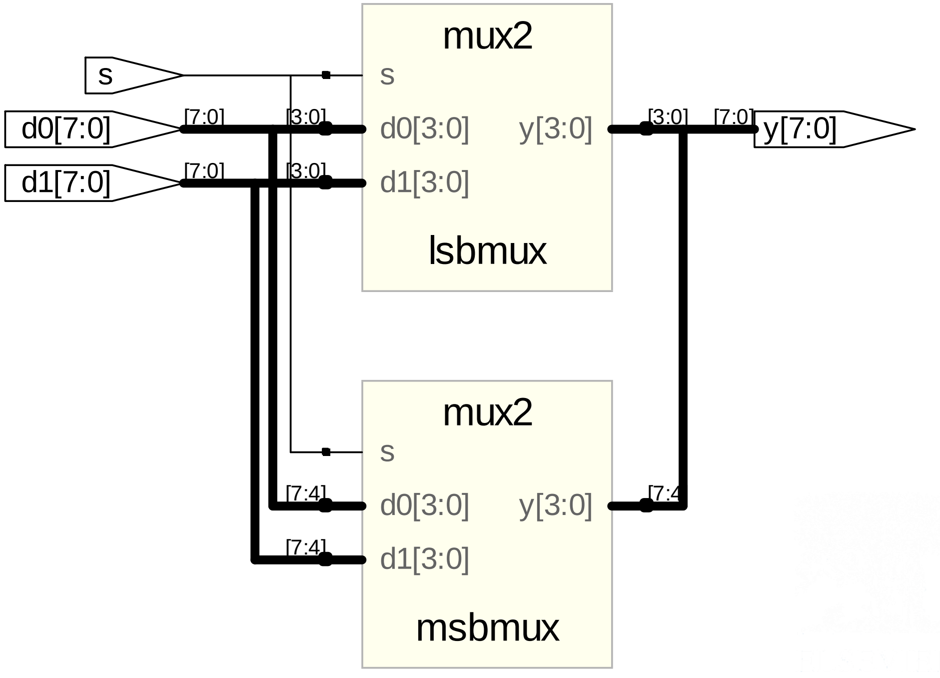
Bit Manipulations: Example 1

```
assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100_010};

// if y is a 12-bit signal, the above statement
   produces:
y = a[2] a[1] b[0] b[0] b[0] a[0] 1 0 0 0 1 0

// underscores (_) are used for formatting only to make
// it easier to read. SystemVerilog ignores them.
```

Bit Manipulations: Example 2

SystemVerilog:

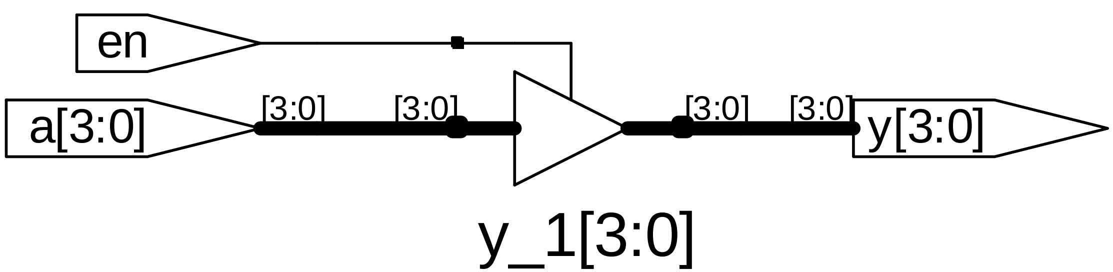
```
module mux2_8(input  logic [7:0] d0, d1,
              input  logic       s,
              output logic [7:0] y);

  mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
  mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule
```



Z: Floating Output

SystemVerilog:

```
module tristate(input  logic [3:0] a,
                input  logic       en,
                output tri   [3:0] y);
  assign y = en ? a : 4'bz;
endmodule
```



The tristate buffer has three possible output states: HIGH (1), LOW (0), and floating (Z). The tristate buffer has an input A, output Y, and enable E.

When the enable is:

- TRUE, the tristate buffer transfers the input value to the output
- FALSE, the output is allowed to float (Z)

SystemVerilog logic datatype

- SystemVerilog logic datatype can assume 0, 1, *z*, and *x* values.

- *x* indicates an **invalid logic level:** if a bus is simultaneously driven to 0 and 1 by two enabled tristate buffers (or other gates), the result is *x*, indicating contention
- At the start of simulation, state nodes such as flip-flop outputs are initialized to an unknown state (*x* in SystemVerilog)

- z indicates a **floating value** - particularly useful for describing a tristate buffer, whose output floats when the enable is 0: if the buffer is enabled, the output is the same as the input, if the buffer is disabled, the output is assigned a floating value z

TRUTH TABLES WITH UNDEFINED AND FLOATING INPUTS

SystemVerilog sometimes can determine the output of logic operation despite some inputs being unknown

&		A			
		0	1	z	x
B	0	0	0	0	0
	1	0	1	x	x
	z	0	x	x	x
	x	0	x	x	x

|		A			
		0	1	z	x
B	0	0	1	x	x
	1	1	1	1	1
	z	x	1	x	x
	x	x	1	x	x

Free tools

Indicazioni per sistemi Linux (funziona anche su Windows usando WSL)

Simulatore (icarus verilog):

sudo apt-get iverilog

Sintesi (yosys):

sudo apt-get yosys

Visualizzatore schematici

sudo npm install -g netlistsvg

Visualizzatore waveforms

sudo apt-get gtkwave

Free tools

In particolare usando WSL questa è la sequenza da eseguire per installare iverilog e tutto ciò che occorre per usarlo

```
# scaricare iverilog
git clone https://github.com/steveicarus/iverilog.git
cd iverilog
# per eseguire autoconf.sh bisogna installare autoconf e gperf
sudo apt-get install autoconf
sudo apt-get update
sudo apt-get install gperf
sh autoconf.sh
# per eseguire ./configure bisogna installare flex, gcc, bison, g++
sudo apt-get install flex
sudo apt-get install gcc
sudo apt-get install bison
sudo apt-get install g++
./configure
# per installare iverilog, gtkwave,yosys, netlistsvg
sudo apt-get install iverilog
sudo apt-get install gtkwave
sudo apt-get yosys
sudo apt-get install yosys
sudo apt-get install npm
sudo npm install -g netlistsvg
```

Free tools

```
annalisa@Annalisa-ThinkPad-T470:~$ cat AAA-how-to-install-verilog
# scaricare iverilog
git clone https://github.com/steveicarus/iverilog.git
cd iverilog
# per eseguire autoconf.sh bisogna installare autoconf e gperf
sudo apt-get install autoconf
sudo apt-get update
sudo apt-get install gperf
sh autoconf.sh
# per eseguire ./configure bisogna installare flex, gcc, bison, g++
sudo apt-get install flex
sudo apt-get install gcc
sudo apt-get install bison
sudo apt-get install g++
./configure
# per installare iverilog, gtkwave,yosys, netlistsvg
sudo apt-get install iverilog
sudo apt-get install gtkwave
sudo apt-get yosys
sudo apt-get install yosys
sudo apt-get install npm
sudo npm install -g netlistsvg

annalisa@Annalisa-ThinkPad-T470:~$
```