تصميم الدارات الإلكترونية بالحاسوب

Computer Design of Electronic Circuits

المحاضرة الثانية

السنة الرابعة قسم التحكم والأتمتة
العام الدراسي 2023-2024

إعداد: د. علا جزماتي

محتويات المقرر

1. مدخل إلى أهمية تطوير أدوات التصميم باستخدام الحاسب (Introduction to The Need of Developing CAD Tools)
2. تصنيف عام لأنواع أدوات التصميم (General Classification of CAD Tools Used in Electronic Systems Design)
3. مدخل إلى اللغات المستخدمة في التصميم (Introduction to Design Languages VHDL, Verilog, Verilog System, ..)
4. مدخل إلى مراحل بناء النظم الرقمية (Introduction to Digital Systems Synthesis)
5. مرحلة البناء منخفض المستوى (Low Level Synthesis)
6. تصميم الدارات المتكاملة للنظم عالية التكامل (Layout Design for VLSI Systems)
7. تطبيقات تصميمية (Design Applications)
8. اتجاهات التطور الحديثة (Trends and New Directions)

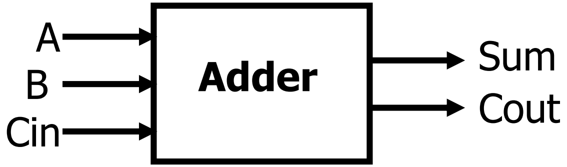
تذكرة

أولا : لغة Verilog

كيف نقوم بتوصيف أي نظام رقمي باستخدام Verilog ؟

- أنماط المعطيات (Datatypes)
- العمليات الأساسية (Basic Operators)
- التوابع (Functions-called module in Verilog)

Full adder



```
module adder (  input a, input b, input cin, output sum,  output cout );

    assign sum = a ^ b ^ cin;
    assign cout = (a & b)  |  (a & cin)  |  (b & cin);

endmodule
```

Data Levels

0	**represents a logic zero, or a false condition**
1	represents a logic one, or a true condition
x	represents an unknown logic value (can be zero or one)
z	represents a high-impedance state

Data Types

- Filling datatypes
  - single bits → 0, 1
  - multiple bits → [Width]'[Radix][Value] → 2'b01
    - 2 bit wide
    - binary
    - 01 filled

Verilog also supports:
'b01 → automatic width matching
'b0 → sets all bits to zero

Width of number has to match wire or reg width!

- We can also use:
  - 'b → binary
  - 'h → hexadecimal
  - 'd → decimal

Data Types

Format : <size>’<base><value>

Example : 8’d16

8’h10

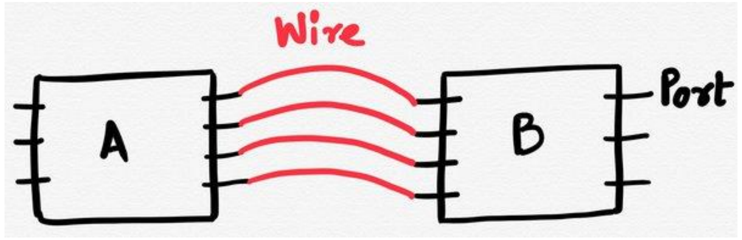
8’b00010000

Data Types

- reg
  - source of signal
- wire
  - acts as a connection transports a signal

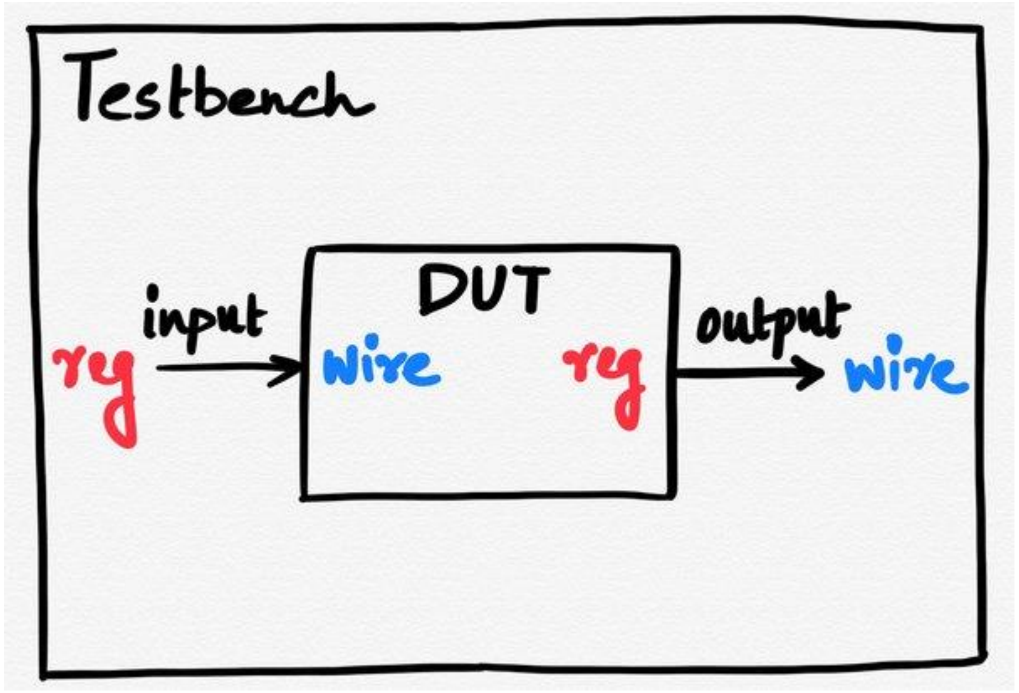
Every signal needs one source!
External inputs can also be sources.

Multiple drivers are not allowed!



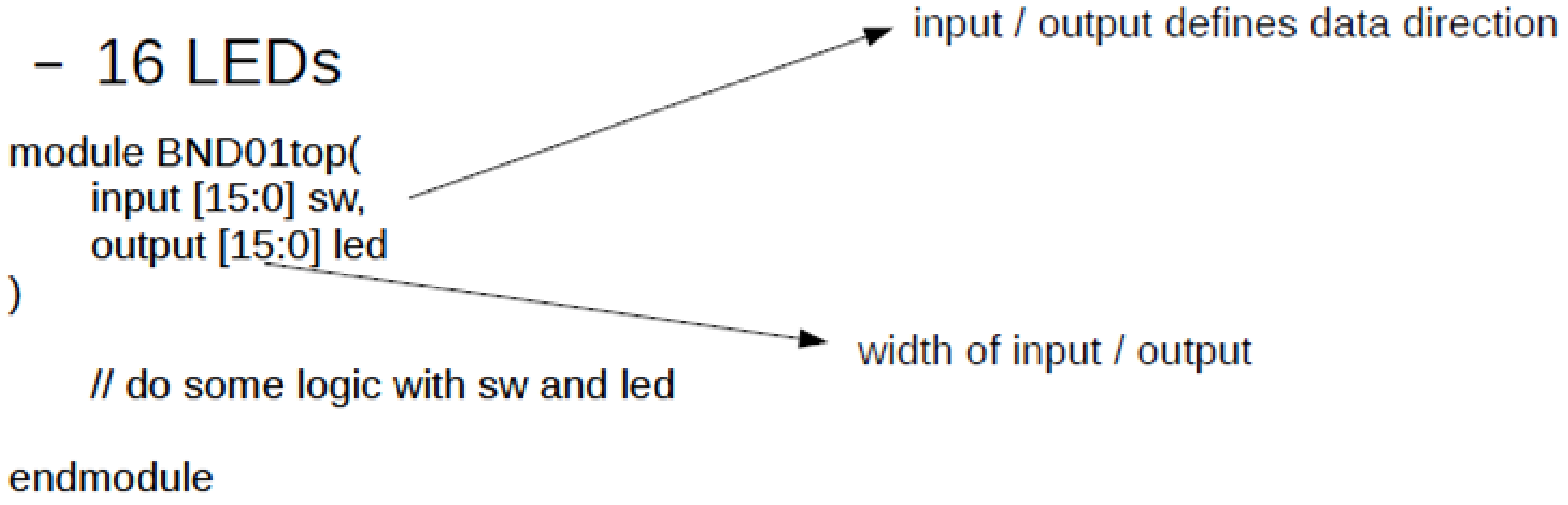
Data Types

مثال ملف المحاكاة



Inputs and Outputs

- First module using:
  - 16 slide switches
  - 16 LEDs



```
module BND01top(
      input [15:0] sw,
      output [15:0] led
)

      // do some logic with sw and led

endmodule
```

Basic Operators

- Basic logical operators:
  - OR bitwise 'b01 | 'b10 = 'b11
  - AND bitwise 'b01 & 'b11 = 'b01
  - NOT bitwise ~ 'b01 = 'b10
  - XOR bitwise 'b01 ^ 'b11 = 'b10
- Basic comparisons
  - equality logical ==
  - inequality logical !=
  - AND logical &&
  - OR logical ||

There are a lot more operators:
>, >=, < etc.
I will introduce them, when they are needed.

Basic Operators

- Howto assing values to outputs
  - combinatorical logic:

```
assign led[0] = sw[0] & sw[1];      sw0 and sw1
assign led[0] = sw[0] | ~sw[1];     sw0 or not sw1
```

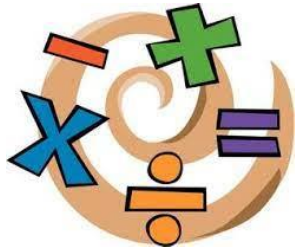
Basic Operators

Arithmetic operators: These include the following that operate on two operands

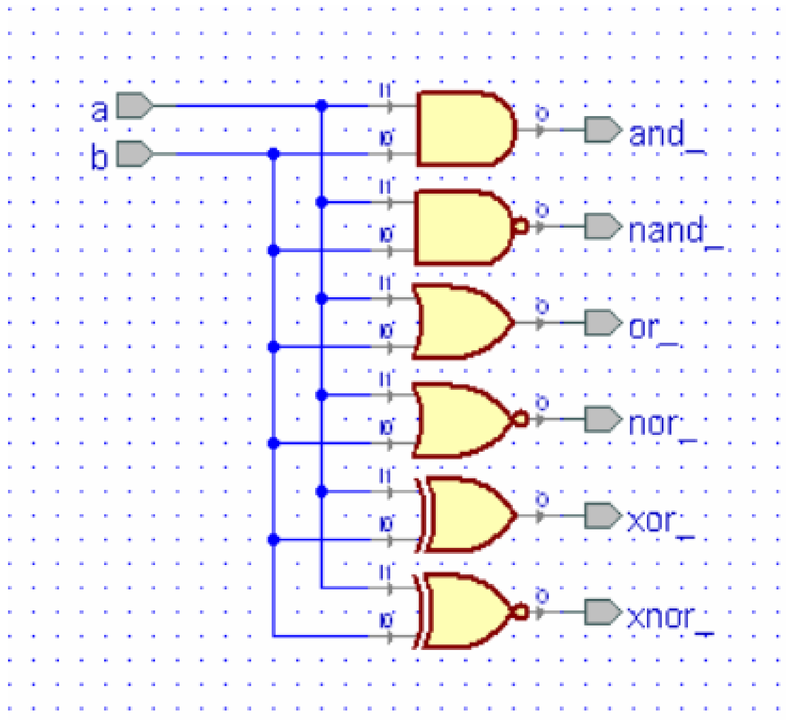
Operator	Add	Subtract	Multiply	Divide	Modulus
Symbol	+	−	*	/	%

Operator	greater than	less than	greater than or equal to	less than or equal to
Symbol	>	<	>=	<=

Operator	equal	not equal	equal (case)	not equal (case)
Symbol	==	!=	===	!===



// Example 1

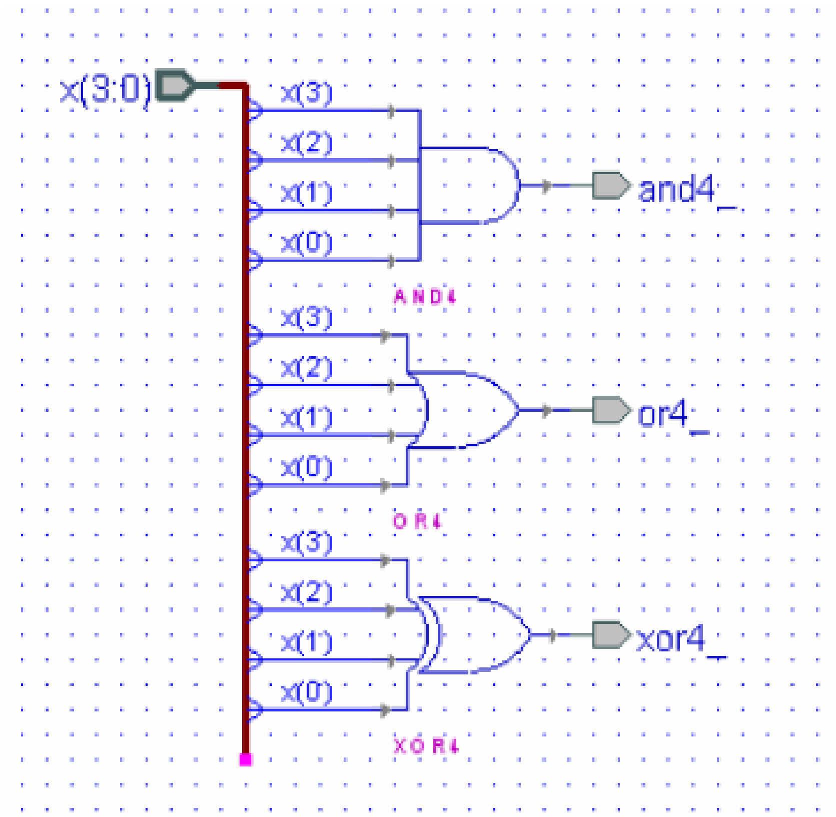


```
module gates2 (input a, input b,
output and_, output nand_, output nor_, output or_,
output xnor_, output xor_ ) ;

assign and_ = b & a;
assign nand_ = ~(b & a);
assign or_ = b | a;
assign nor_ = ~(b | a);
assign xor_ = b ^ a;
assign xnor_ = ~(b ^ a);

endmodule
```

// Example 2

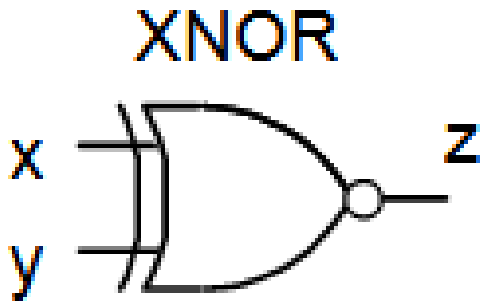


```
// 4-input gates
module gates4b (
input wire [3:0] x ,
output and4_ ,
output or4_ ,
output xor4_);
assign and4_  =  &x;
assign or4_  =  |x;
assign xor4_  =  ^x;
endmodule
```

// Example 3

توصيف كاشف المساواة

Equality Detector



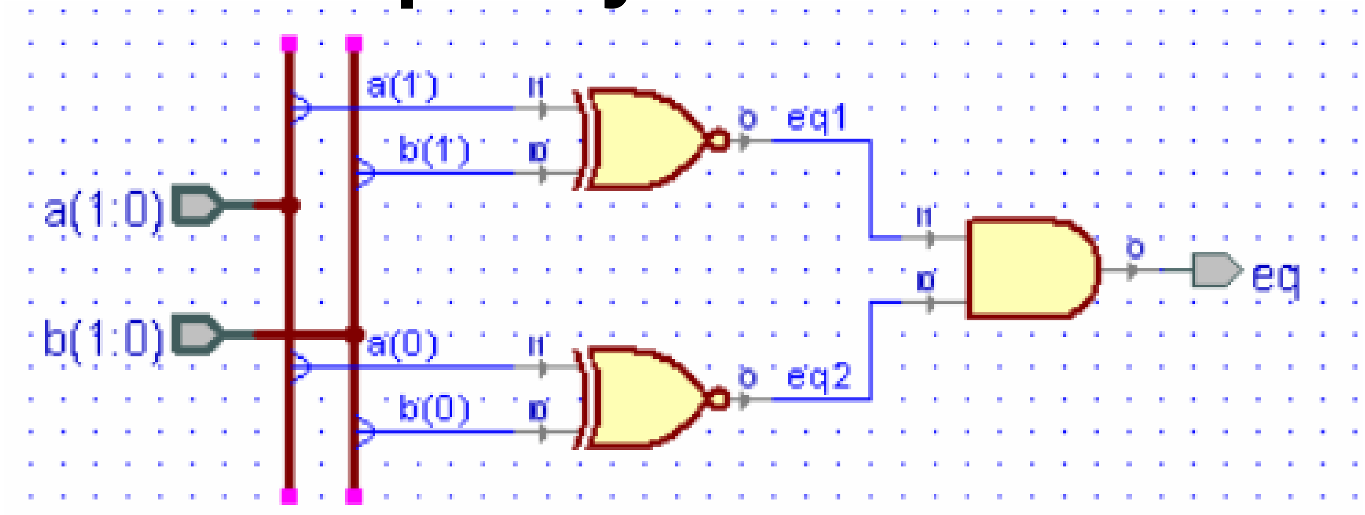
x	y	z
0	0	1
0	1	0
1	0	0
1	1	1

The XNOR gate is a 1-bit equality detector

// Example 4

توصيف كاشف المساواة

Equality Detector

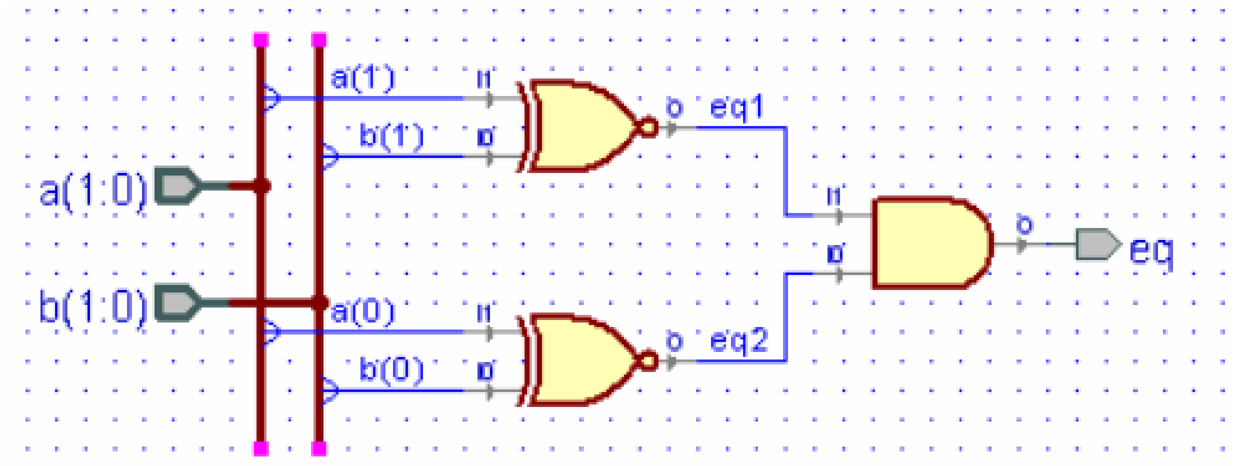


Block diagram of a 2-bit equality detector

// Example 4

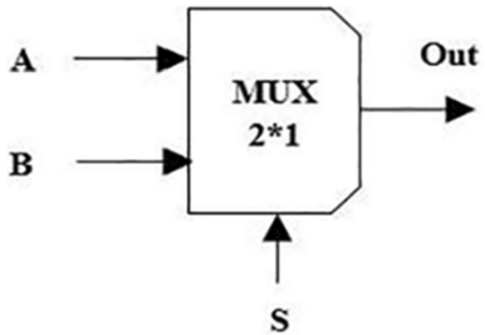
توصيف كاشف المساواة

```
module eqdet2 (
input [1:0] a,
input  [1:0] b,
output eq
) ;
wire eq1;
wire eq2;
assign eq1 = ~(b[1] ^ a[1]);
assign eq2 = ~(b[0] ^ a[0]);
assign eq = eq2 & eq1;
endmodule
```

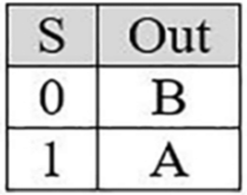


/ / Example 5

توصيف ناخب

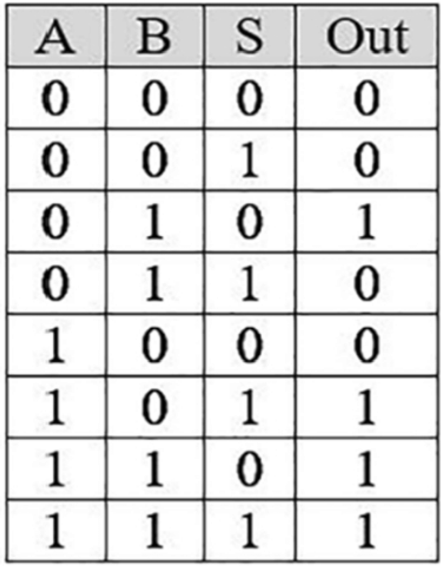


(a)



S	Out
0	B
1	A

(b)



A	B	S	Out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

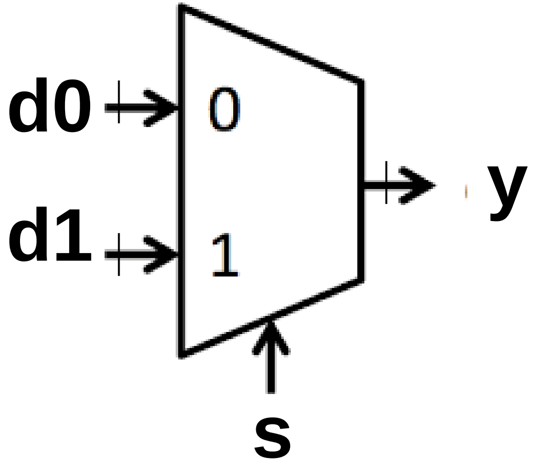
(c)

Conditional Operators in Expressions

- Like C/C++/Java Conditional Operator

```
module mux2(d0, d1, s, y);
  input  [3:0] d0, d1;
  input        s;
  output [3:0] y;

  assign y = s ? d1 : d0;
  // output d1 when s=1, else d0
 endmodule
```



- “if” statements not allowed in assign

Combinational Modeling with `always`

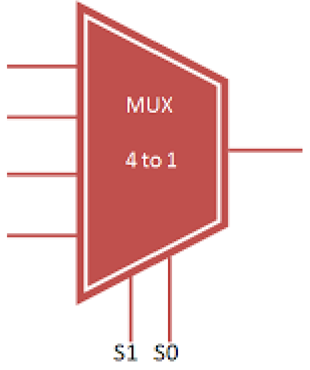
- **Example: 4-input mux behavioral model**

```
module mux4(d0, d1, d2, d3, s, y);
    input          d0, d1, d2, d3;
    input   [1:0]  s;
    output         y;
    reg            y;  // declare y as a variable
```

Sensitivity List (activates on change)

```
    always @(d0 or d1 or d2 or d3 or s)
        case (s)
            2'd0 : y = d0;
            2'd1 : y = d1;
            2'd2 : y = d2;
            2'd3 : y = d3;
            default : y = 1'bx;
        endcase
endmodule
```

Blocking assignments (immediate update)



Combinational Modeling with `always`

- **Useful shorthand avoids common simulation errors**

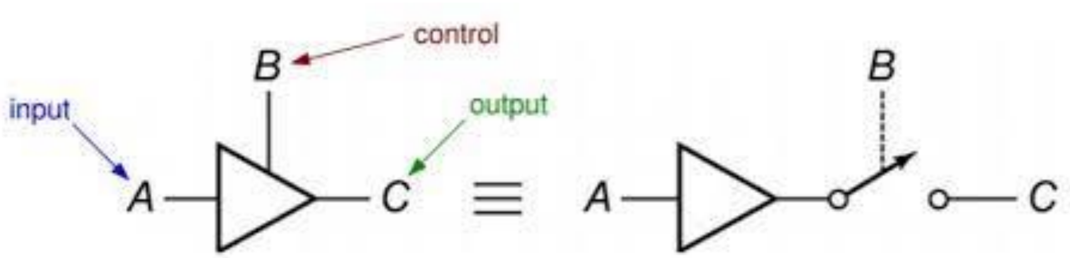
```
module mux4(d0, d1, d2, d3, s, y);
     input          d0, d1, d2, d3;
     input   [1:0]  s;
     output         y;
     reg            y;

     always @*
         case (s)
             2'd0 : y = d0;
             2'd1 : y = d1;
             2'd2 : y = d2;
             2'd3 : y = d3;
             default : y = 1'bx;
        endcase
endmodule
```

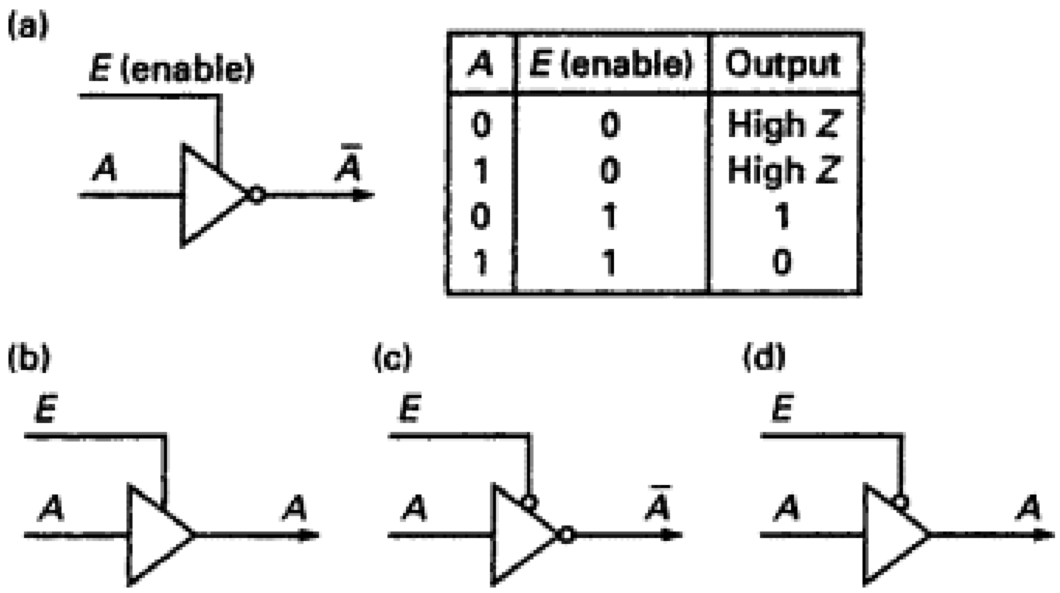
Activates on any input change
`d0, d1, d2, d3`

/ / Example 6

توصيف العازل
ثلاثي الحالة



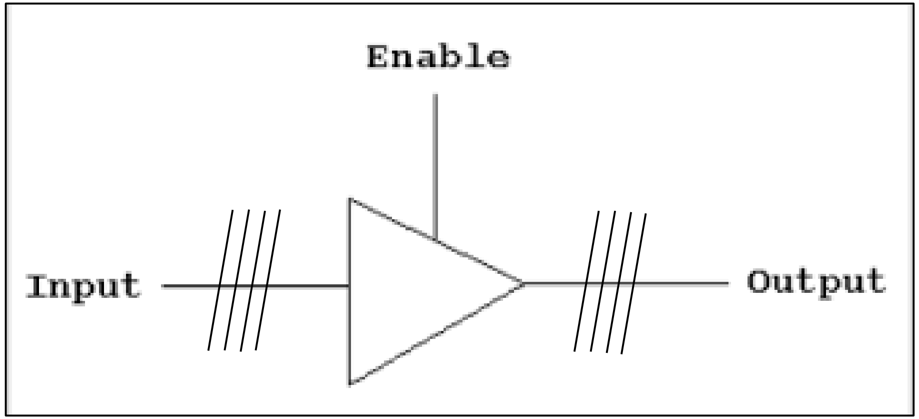
توصيف العازل ثلاثي الحالة



A	E (enable)	Output
0	0	High Z
1	0	High Z
0	1	1
1	1	0

```
// Tristate Buffer
module tristate_buffer(input A, input enable, output C);
Assign C = enable? A : 1'bz;
endmodule
```

توصيف العازل ثلاثي الحالة



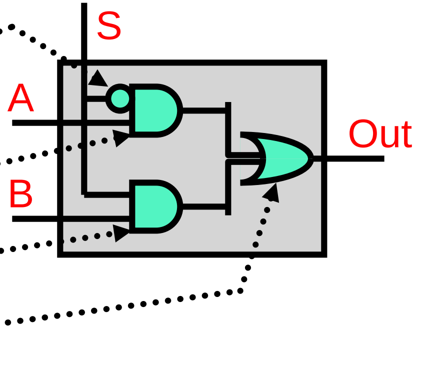
Reserved Words

always	and	assign	begin	buf	bufif0	bufif1	case
casex	casez	cmos	deassign	default	defparam	disable	edge
else	end	endcase	endfunction		endmodule		
endprimitive		endspecify		endtable	endtask	event	for
force	forever	fork	function	highz0	highz1	if	ifnone
initial	inout	input	integer	join	large	macromodule	
medium	module	nand	negedge	nmos	nor	not	
notif0	notif	or	output	parameter		pmos	
posedge	primitive		pull0	pull1	pulldown	pullup	rcmos
real	realtime	reg	release	repeat	rnmos	rpmos	rtran
rtranif0	rtranif1	scalared	small	specify	specparam		strong0
strong1	supply0	supply1	table	task	time	tran	tranif0
tranif1	tri	tri0	tri1	triand	trior	trireg	vectored
wait	wand	weak0	weak1	while	wire	wor	xnor
xor							

مقارنة بين طرق التوصيف

Structural

```
module mux2to1(
  input S, A, B,
  output Out );
  wire S_, aa, bb;
  not (S_, S);
  and (aa, A, S_);
  and (bb, B, S);
  or (Out, aa, bb);
endmodule
```



Behavioral

```
module mux2to1(
  input S, A, B,
  output Out );
  assign Out = (~S & A) | (S & B);
endmodule
```

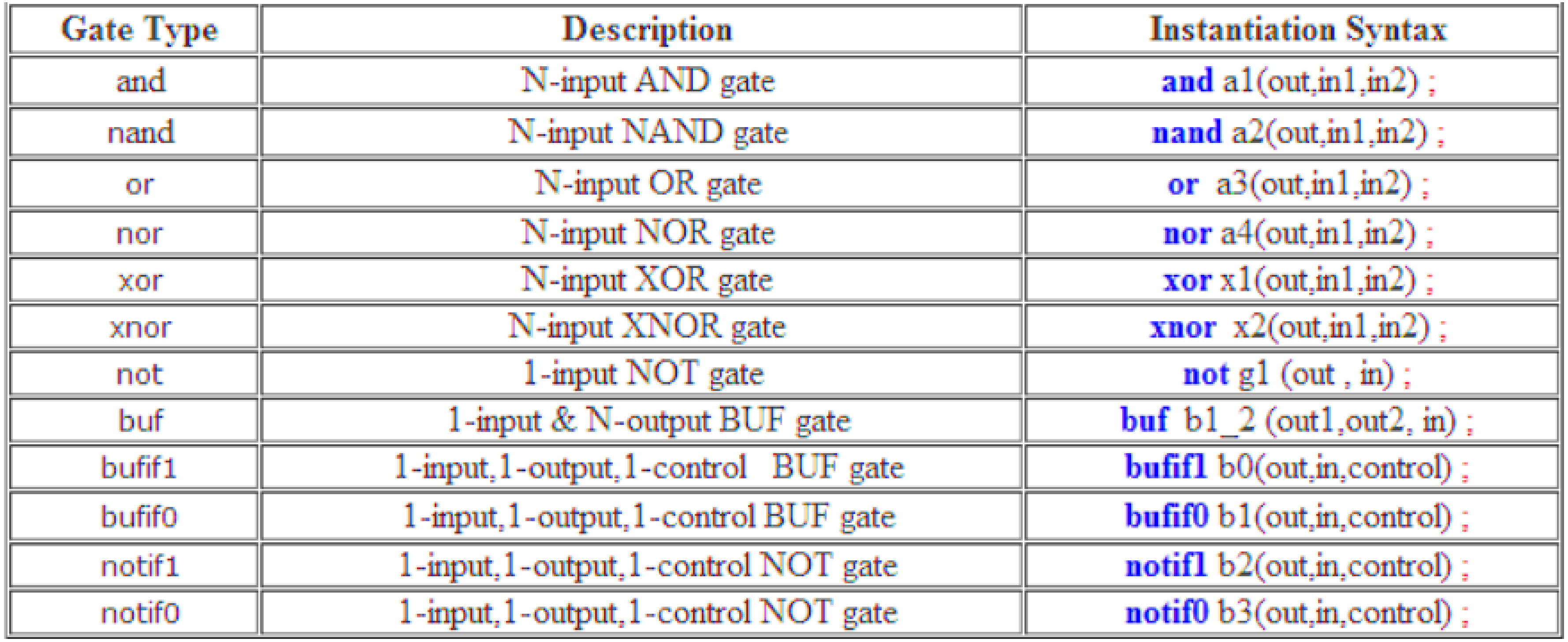
Data flow:

```
assign Out = S? B:A;
```

Or

```
assign Out = S == 1'b1? B:A;
```

العمليات المنطقية المتوفرة للتوصيف البنيوي:



Gate Type	Description	Instantiation Syntax
and	N-input AND gate	**and** a1(out,in1,in2) ;
nand	N-input NAND gate	**nand** a2(out,in1,in2) ;
or	N-input OR gate	**or** a3(out,in1,in2) ;
nor	N-input NOR gate	**nor** a4(out,in1,in2) ;
xor	N-input XOR gate	**xor** x1(out,in1,in2) ;
xnor	N-input XNOR gate	**xnor** x2(out,in1,in2) ;
not	1-input NOT gate	**not** g1 (out , in) ;
buf	1-input & N-output BUF gate	**buf** b1_2 (out1,out2, in) ;
bufif1	1-input,1-output,1-control BUF gate	**bufif1** b0(out,in,control) ;
bufif0	1-input,1-output,1-control BUF gate	**bufif0** b1(out,in,control) ;
notif1	1-input,1-output,1-control NOT gate	**notif1** b2(out,in,control) ;
notif0	1-input,1-output,1-control NOT gate	**notif0** b3(out,in,control) ;

الأسئلة والمناقشة