

Laborator 5

Fundamentele Calculatoarelor

Obiective:

- Funcții logice și operatori booleani în Verilog (Porti logice)
- Înțelegerea și implementarea unor arhitecturi în Verilog
- Înțelegerea conceptului de "Decision Table Test"

Funcții logice și operatori booleani în Verilog (Porti logice)

Funcții logice fundamentale:

• AND: $a \& b$ 


• OR: $a | b$ 

• XOR: $a \wedge b$ 

• NOT: $\sim a$ 

• NAND: $\sim(a \& b)$ 

• NOR: $\sim(a | b)$ 

• XNOR: $\sim(a \wedge b)$ 

Funcția logică	Operator bitwise	Operator logic	Operator de reducere
AND	$a \& b$	$a \&\& b$	$\& \text{ vector}$
OR	$a b$	$a b$	$ \text{ vector}$
XOR	$a \wedge b$	-	$\wedge \text{ vector}$
NOT	$\sim a$	$!a$	$\sim \text{ vector}$
NAND	$\sim(a \& b)$	-	$\sim\& \text{ vector}$
NOR	$\sim(a b)$	-	$\sim \text{ vector}$
XNOR	$\sim(a \wedge b)$	-	$\sim\wedge \text{ vector}$

!?

În tabelul de mai sus observăm operatorii din Verilog asociați funcțiilor logice elementare și moduri de folosire ale lor.

Operatorii bitwise realizează funcția logică între cei doi operanzi bit cu bit.

Operatorii logici realizează funcția logică între cei doi operanzi la nivel de expresie și se utilizează de regulă în condițiile instrucțiunilor de tipul if.

Operatorii de reducere acționează asupra unui vector și realizează operația logică între toți biții acelui vector.

Deoarece operatorul \sim este un operator unar, acesta se poate aplica unui singur operand, deci nu se poate scrie pentru funcția NAND a $\sim\& b$, ci se va scrie $\sim(a \& b)$.

Tabelul de adevar al functiei AND :

Input: $a \& b = a * b$

a	b	AND
0	0	0
0	1	0
1	0	0
1	1	1

Input: $a \& b \& c = a * b * c$

a	b	c	AND
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Tabelul de adevar al functiei OR :

Input: $a | b = a + b$

a	b	Or
0	0	0
0	1	1
1	0	1
1	1	1

Input: $a | b | c = a + b + c$

a	b	c	OR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Tabelul de adevar al functiei XOR :

Input: $a \wedge b$

a	b	XOr
0	0	0
0	1	1
1	0	1
1	1	0

Input: $a \wedge b \wedge c$

a	b	c	XOR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Tabelul de adevar al functiei NOT :

Input: $\sim a$

a	NOT
0	1
1	0

Tabelul de adevar al functiei NAND :

Input: $\sim(a \& b) = \sim(a * b) = (\sim a) + (\sim b)$

a	b	NAND
0	0	1
0	1	1
1	0	1
1	1	0

Input: $\sim(a \& b \& c) = (\sim a) + (\sim b) + (\sim c)$

a	b	c	NAND
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Tabelul de adevar al functiei NOR :

Input: $\sim(a | b) = \sim(a + b) = (\sim a) * (\sim b)$

a	b	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Input: $\sim(a | b | c) = (\sim a) * (\sim b) * (\sim c)$

a	b	c	NOR
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Tabelul de adevar al functiei XNOR :

Input: $\sim(a \wedge b)$

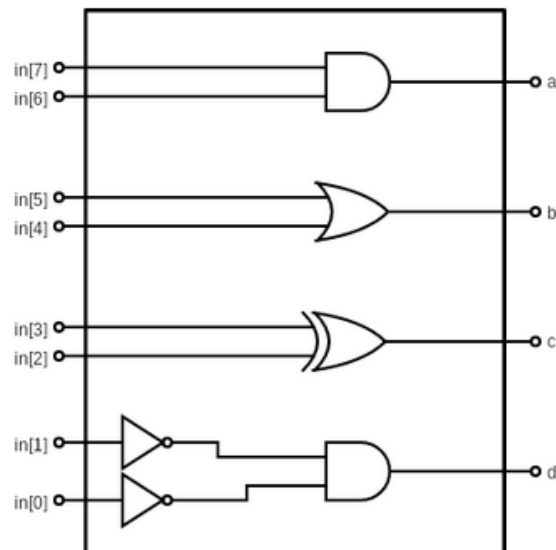
a	b	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Input: $\sim(a \wedge b \wedge c)$

a	b	c	XNOR
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Intelegerea si implementarea unor arhitecturi in Verilog

1. Sa se scrie un modul avand **o intrare pe 8 biti(notata in)** si **4 iesiri pe 1 bit(notate a, b, c, d)**. Iesirile vor fi egale cu: $a = in[7] \text{ AND } in[6]$; $b = in[5] \text{ OR } in[4]$; $c = in[3] \text{ XOR } in[2]$; $d = \text{NOT } in[1] \text{ AND NOT } in[0]$



```

module logic_gates(input [7:0]in, output a, b, c, d);
assign a = in[7] & in[6];
assign b = in[5] | in[4];
assign c = in[3] ^ in[2];
assign d = ~in[1] & ~in[0];
endmodule

module logic_gates_tb;

reg[7:0] in;
wire a, b, c, d;

logic_gates uut(.in(in), .a(a), .b(b), .c(c), .d(d));

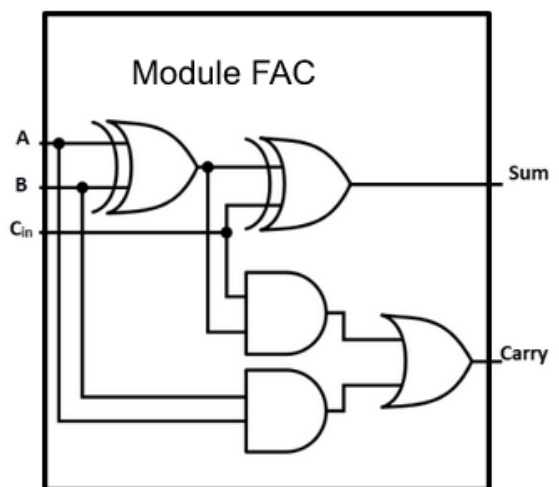
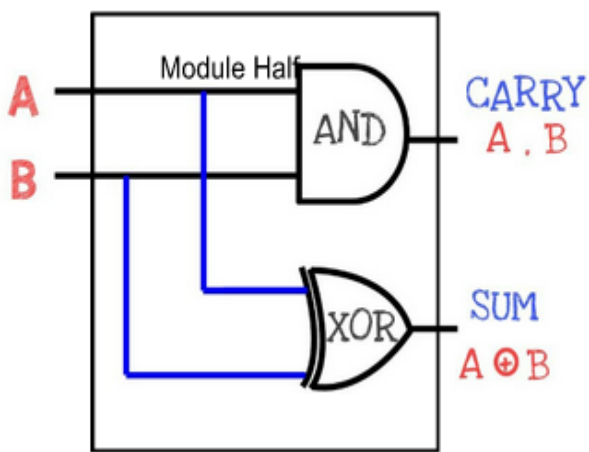
integer i;

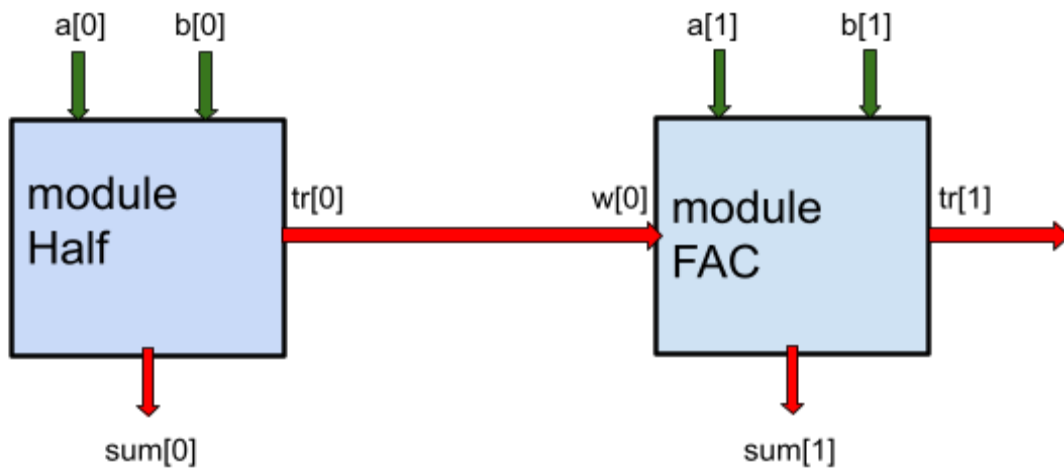
initial begin
    for(i = 0; i < 256; i = i + 1) begin
        in = i;
        #1;
        $display("in = %b, a = %b, b = %b, c = %b, d = %d", in, a, b, c, d);
    end
end
endmodule

```

2. Sa se scrie un modul care efectueaza suma a doua intrari pe 2 biți folosind:

- modulul Half suma a doua intrari pe un singur bit si returneaza atat rezultatul adunarii cat si bitul ramas pentru transport ("in minte")
- modul numit Full Adder Cell, care va efectua suma intrarilor si al bitului de transport returnat de modulul Half si va returna atat rezultatul adunarii cat si bitul ramas pentru transport (in minte) .





```

module half_adder_cell(input a, b, output sum_, carry);

assign sum_ = a ^ b;
assign carry = a & b;

endmodule

module half_adder_cell_tb;
reg a, b;
wire sum_, carry;

half_adder_cell dut (.a(a), .b(b), .sum_(sum_), .carry(carry));
integer i;

initial begin
    for(i = 0; i < 4; i = i + 1) begin
        {a, b} = i;
        #1;
        $display("a = %b, b = %b, sum_ = %b, carry = %b", a, b, sum_, carry);
    end
end

endmodule

module full_adder_cell(input a, b, cin, output sum_, carry);

assign sum_ = a ^ b ^ cin;
assign carry = (a & b) | (a & cin) | (b & cin);

endmodule

```

```

module full_adder_cell_tb;

reg a, b, cin;
wire sum_, carry;

full_adder_cell uut (.a(a), .b(b), .cin(cin), .sum_(sum_), .carry(carry));

integer i;

initial begin
    for(i = 0; i < 8; i = i + 1) begin
        {a, b, cin} = i;
        #1;
        $display("a = %b, b = %b, cin = %b, sum_ = %b, carry = %b", a, b, cin, sum_, carry);
    end
end
endmodule

module sum_2b(input[1:0] a, b, output[1:0] sum_, output carry);

wire w;

half_adder_cell hac(.a(a[0]), .b(b[0]), .sum_(sum_[0]), .carry(w));
full_adder_cell fac(.a(a[1]), .b(b[1]), .cin(w), .sum_(sum_[1]), .carry(carry));

endmodule

module sum_2b_tb;

reg[1:0] a, b;
wire[1:0] sum_;
wire carry;

sum_2b dut(.a(a), .b(b), .sum_(sum_), .carry(carry));

integer i;

initial begin
    for(i = 0; i < 16; i = i + 1) begin
        {a, b} = i;
        #1;
        $display("a = %d (%b), b = %d (%b), sum_ = %d (%b), carry = %b", a, a, b, b, sum_, sum_,
carry);
    end
end
endmodule

```

Întelegerea conceptului de “Decision Table Test”

Decision table testing este o tehnică de testare software utilizată pentru a testa comportamentul sistemului pentru diferite combinații de intrare. Aceasta este o abordare sistematică în care diferitele combinații de intrări și comportamentul lor corespunzător al sistemului (leșire) sunt capturate într-o formă tabelară. Acesta este motivul pentru care este, de asemenea, numit ca un tabel cauză-efect în cazul în care cauza și efectele sunt capturate pentru o mai bună acoperire a testelor.