Lógica Digital (1001351)

Outros Circuitos Combinacionais

Prof. Edilson Kato
kato@ufscar.br

Prof. Maurício Figueiredo
mauricio@ufscar.br

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Roberto Inoue
rsinoue@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 18 de abril de 2019

Comparadores

Comparadores

Deslocadores

Comparadores

Deslocadores

ULA

Comparadores

Deslocadores

ULA

Verilog

Comparador

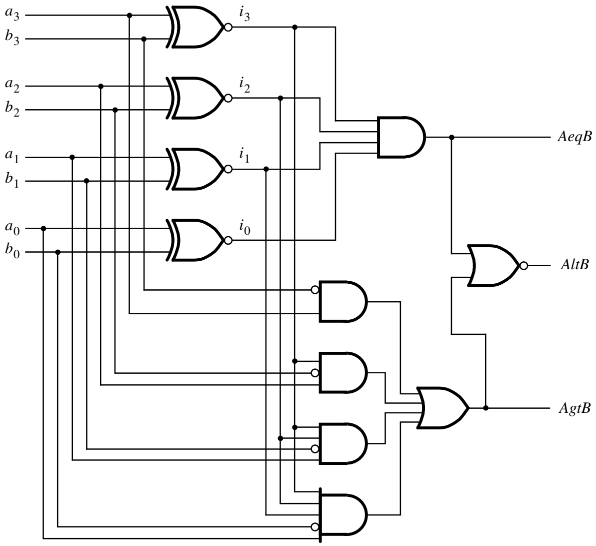


Figure 4.22 A four-bit comparator circuit.

figure4.40.v

```verilog
module compare (A, B, AeqB, AgtB, AltB);
  input [3:0] A, B;
  output reg AeqB, AgtB, AltB;

  always @(A, B)
  begin
    AeqB = 0;
    AgtB = 0;
    AltB = 0;
    if(A == B)
      AeqB = 1;
    else if (A > B)
      AgtB = 1;
    else
      AltB = 1;
  end

endmodule
```

Comparador por subtração

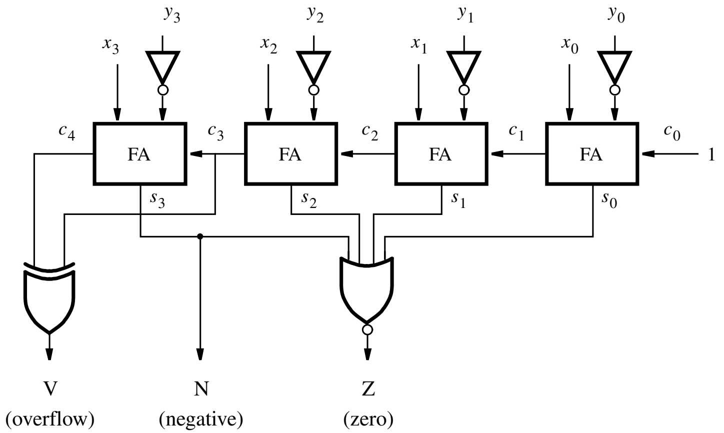


Figure 3.45 A comparator circuit.

$X < Y \iff N \oplus V$ $\quad$ $X = Y \iff Z$ $\quad$ $X \leq Y \iff (N \oplus V) + Z$

figure3.46.v

```
module comparator (X, Y, V, N, Z);
  input [3:0] X, Y;
  output V, N, Z;
  wire [3:0] S;
  wire [4:1] C;

  fulladd stage0 (1'b1, X[0], ~Y[0], S[0], C[1]);
  fulladd stage1 (C[1], X[1], ~Y[1], S[1], C[2]);
  fulladd stage2 (C[2], X[2], ~Y[2], S[2], C[3]);
  fulladd stage3 (C[3], X[3], ~Y[3], S[3], C[4]);
  assign V  = C[4] ^ C[3];
  assign N = S[3];
  assign Z = !S;

endmodule

module fulladd (Cin, x, y, s, Cout);
  input Cin, x, y;
  output s, Cout;

  assign s = x ^ y ^ Cin,
       Cout = (x & y) | (x & Cin) | (y & Cin);

endmodule
```

figure3.47.v

```
module comparator (X, Y, V, N, Z);
  parameter n = 32;
  input [n-1:0] X, Y;
  output reg V, N, Z;
  reg [n-1:0] S;
  reg [n:0] C;
  integer k;

  always @(X, Y)
  begin
    C[0] = 1'b1;
    for (k = 0; k < n;  k = k + 1)
    begin
      S[k] = X[k] ^ ~Y[k] ^ C[k];
      C[k+1] = (X[k] & ~Y[k]) | (X[k] & C[k]) | (~Y[k] & C[k]);
    end
    V  = C[n] ^ C[n-1];
    N = S[n-1];
    Z = !S;
  end

endmodule
```

Deslocamento (*shift*)

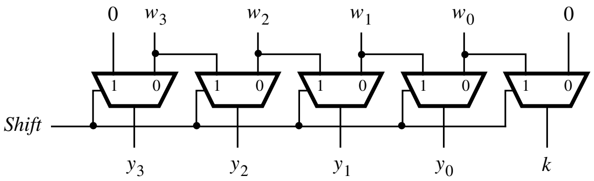


Figure 4.50 A shifter circuit.

figure4.53.v

```verilog
module shifter (W, Shift, Y , k);
  input [3:0] W;
  input Shift;
  output reg [3:0] Y;
  output reg k;

  always @(W, Shift)
  begin
    if (Shift)
    begin
      Y[3] = 0;
      Y[2:0] = W[3:1];
      k = W[0];
    end
    else
    begin
      Y = W;
      k = 0;
    end
  end

endmodule
```

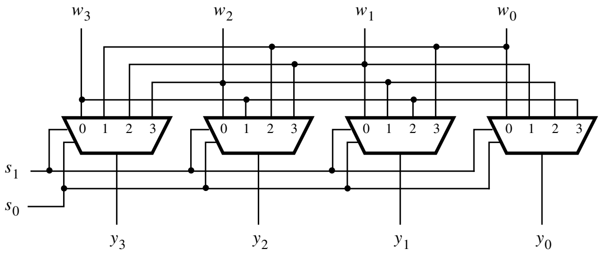
figure4.54.v

```verilog
module shifter (W, Shift, Y , k);
  input [3:0] W;
  input Shift;
  output reg [3:0] Y;
  output reg k;

  always @(W, Shift)
  begin
    if (Shift)
    begin
      Y = W >> 1;
      k = W[0];
    end
    else
    begin
      Y = W;
      k = 0;
    end
  end

endmodule
```

Deslocamento (*rotate*)

s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3

(a) Truth table



(b) Circuit

Figure 4.51 A barrel shifter circuit.

figure4.55.v

```
module barrel (W, S, Y);
  input [3:0] W;
  input [1:0] S;
  output [3:0] Y;
  wire [3:0] T;

  assign {T, Y} = {W, W} >> S;

endmodule
```

ULA

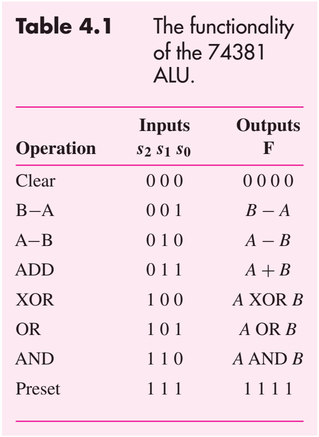


Table 4.1 The functionality of the 74381 ALU.

	Inputs	Outputs
Operation	$s_2\ s_1\ s_0$	F
Clear	0 0 0	0 0 0 0
B–A	0 0 1	$B - A$
A–B	0 1 0	$A - B$
ADD	0 1 1	$A + B$
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1

ULA

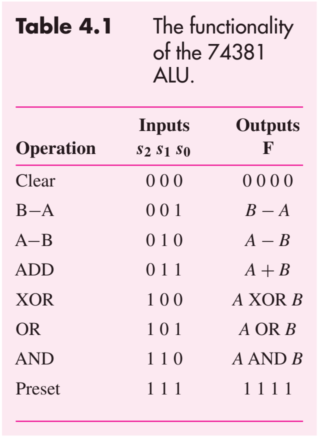


Table 4.1 The functionality of the 74381 ALU.

Operation	**Inputs** $s_2\ s_1\ s_0$	**Outputs** **F**
Clear	0 0 0	0 0 0 0
B–A	0 0 1	$B - A$
A–B	0 1 0	$A - B$
ADD	0 1 1	$A + B$
XOR	1 0 0	A XOR B
OR	1 0 1	A OR B
AND	1 1 0	A AND B
Preset	1 1 1	1 1 1 1

```verilog
// 74381 ALU
module alu (S, A, B, F);
  input [2:0] S;
  input [3:0] A, B;
  output reg [3:0] F;

  always @(S, A, B)
    case (S)
      0: F = 4'b0000;
      1: F = B - A;
      2: F = A - B;
      3: F = A + B;
      4: F = A ^ B;
      5: F = A | B;
      6: F = A & B;
      7: F = 4'b1111;
  endcase

endmodule
```

Operadores em Verilog (1/2)

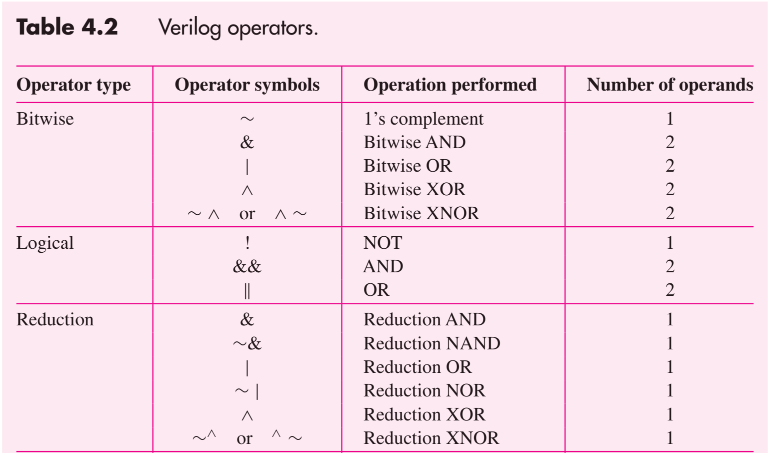


Table 4.2 Verilog operators.

Operator type	Operator symbols	Operation performed	Number of operands
Bitwise	~	1's complement	1
	&	Bitwise AND	2
	|	Bitwise OR	2
	^	Bitwise XOR	2
	~ ^ or ^ ~	Bitwise XNOR	2
Logical	!	NOT	1
	&&	AND	2
	||	OR	2
Reduction	&	Reduction AND	1
	~&	Reduction NAND	1
	|	Reduction OR	1
	~ |	Reduction NOR	1
	^	Reduction XOR	1
	~^ or ^ ~	Reduction XNOR	1

. . .

Operadores em Verilog (2/2)

. . .

Arithmetic	$+$	Addition	2
	$-$	Subtraction	2
	$-$	2's complement	1
	$*$	Multiplication	2
	/	Division	2
Relational	$>$	Greater than	2
	$<$	Less than	2
	$>=$	Greater than or equal to	2
	$<=$	Less than or equal to	2
Equality	$==$	Logical equality	2
	$!=$	Logical inequality	2
Shift	$>>$	Right shift	2
	$<<$	Left shift	2
Concatenation	{,}	Concatenation	Any number
Replication	{{}}	Replication	Any number
Conditional	?:	Conditional	3

Precedencia dos Operadores em Verilog

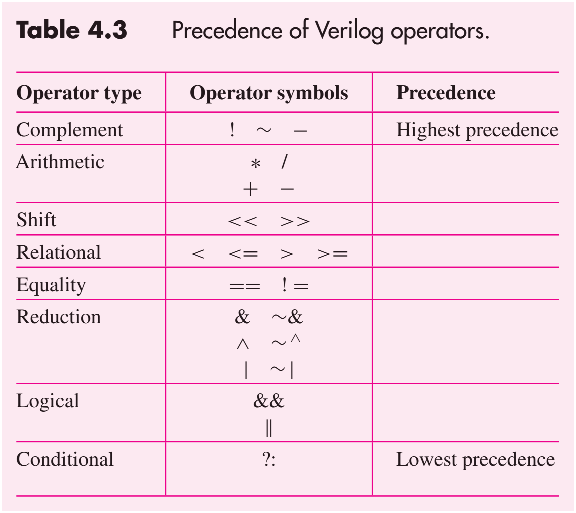
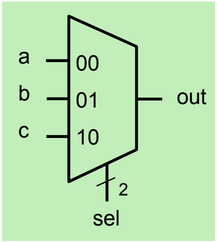


Table 4.3 Precedence of Verilog operators.

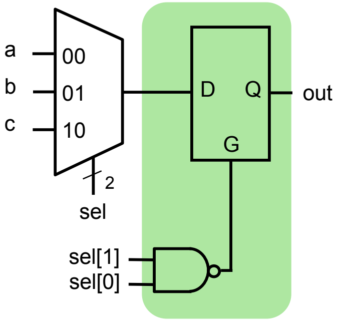
Operator type	**Operator symbols**	**Precedence**
Complement	! ~ −	Highest precedence
Arithmetic	* / + −	
Shift	<< >>	
Relational	< <= > >=	
Equality	== !=	
Reduction	& ~& ^ ~^ | ~|	
Logical	&& ||	
Conditional	?:	Lowest precedence

Perigo!: Especificação Incompleta



```
module mux3to1(a, b, c, sel, out);
  input [1:0] sel;
  input a, b, c;
  output out;
  reg out;

  always @(a or b or c or sel)
  begin
    case (sel)
      2'b00: out = a;
      2'b01: out = b;
      2'b10: out = c;
    endcase
  end
endmodule
```

Perigo!: Especificação Incompleta

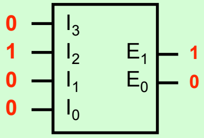


Evitando a Especificação Incompleta

```
always @(a or b or c or sel)
begin
  out = 1'bx
  case (sel)
    2'b00: out = a;
    2'b01: out = b;
    2'b10: out = c;
  endcase
end
```

```
always @(a or b or c or sel)
begin
  case (sel)
    2'b00: out = a;
    2'b01: out = b;
    2'b10: out = c;
    default: out = 1'bx
  endcase
end
```

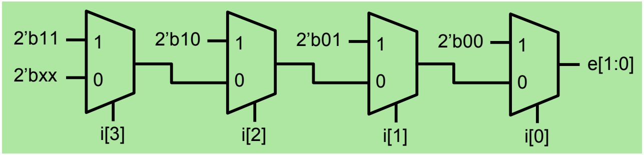
Perigo!: Prioridade Indesejada



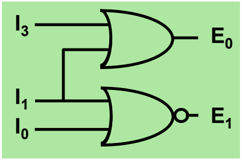
I_3 I_2 I_1 I_0	E_1 E_0
0 0 0 1	0 0
0 0 1 0	0 1
0 1 0 0	1 0
1 0 0 0	1 1
all others	X X

```
module binary_encoder(i, e);
  input [3:0] i;
  output [1:0] e;
  reg e;

  always @(i)
  begin
    if (i[0])
      e = 2'b00;
    else if (i[1])
      e = 2'b01;
    else if (i[2])
      e = 2'b10;
    else if (i[3])
      e = 2'b11;
    else
      e = 2'bxx;
  end
endmodule
```

Perigo!: Prioridade Indesejada



Evitando Prioridade Indesejada



```
module binary_encoder(i, e);
  input [3:0] i;
  output [1:0] e;
  reg e;

  always @(i)
  begin
    if (i == 4'b0001)
      e = 2'b00;
    else if (i == 4'b0010)
      e = 2'b01;
    else if (i == 4'b0100)
      e = 2'b10;
    else if (i == 4'b1000)
      e = 2'b11;
    else
      e = 2'bxx;
  end
endmodule
```

Bibliografia

- Brown, S. & Vranesic, Z. - Fundamentals of Digital Logic with Verilog Design, 3rd Ed., Mc Graw Hill, 2009

Lógica Digital (1001351)

Outros Circuitos Combinacionais

Prof. Edilson Kato
kato@ufscar.br

Prof. Maurício Figueiredo
mauricio@ufscar.br

Prof. Ricardo Menotti
menotti@ufscar.br

Prof. Roberto Inoue
rsinoue@ufscar.br

Departamento de Computação
Universidade Federal de São Carlos

Atualizado em: 18 de abril de 2019