Lógica Digital (1001351)

Projeto de Circuitos Aritméticos com Verilog

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: 10 de abril de 2019

figure3.18.v

```
1  module fulladd (Cin, x, y, s, Cout);
2    input Cin, x, y;
3    output s, Cout;
4
5    xor (s, x, y, Cin);
6    and (z1, x, y);
7    and (z2, x, Cin);
8    and (z3, y, Cin);
9    or (Cout, z1, z2, z3);
10 endmodule
```

figure3.19.v

```
1  module fulladd (Cin, x, y, s, Cout);
2    input Cin, x, y;
3    output s, Cout;
4
5    xor (s, x, y, Cin);
6    and (z1, x, y),
7        (z2, x, Cin),
8        (z3, y, Cin);
9    or  (Cout, z1, z2, z3);
10 endmodule
```

figure3.20.v

```
1 module fulladd (Cin, x, y, s, Cout);
2   input Cin, x, y;
3   output s, Cout;
4
5   assign s = x ^ y ^ Cin;
6   assign Cout = (x & y) | (x & Cin) | (y & Cin);
7 endmodule
```

figure3.21.v

```
1 module fulladd (Cin, x, y, s, Cout);
2   input Cin, x, y;
3   output s, Cout;
4
5   assign s = x ^ y ^ Cin,
6          Cout = (x & y) | (x & Cin) | (y & Cin);
7 endmodule
```

figure3.22.v

```
1  module adder4 (carryin, x3, x2, x1, x0, y3, y2, y1, y0,
2                 s3, s2, s1, s0, carryout);
3    input carryin, x3, x2, x1, x0, y3, y2, y1, y0;
4    output s3, s2, s1, s0, carryout;
5
6    fulladd stage0 (carryin, x0, y0, s0, c1);
7    fulladd stage1 (c1, x1, y1, s1, c2);
8    fulladd stage2 (c2, x2, y2, s2, c3);
9    fulladd stage3 (c3, x3, y3, s3, carryout);
10 endmodule
11
12 module fulladd (Cin, x, y, s, Cout);
13   input Cin, x, y;
14   output s, Cout;
15
16   assign s = x ^ y ^ Cin;
17   assign Cout = (x & y) | (x & Cin) | (y & Cin);
18 endmodule
```

figure3.23.v

```
1  module adder4 (carryin, X, Y, S, carryout);
2    input carryin;
3    input [3:0] X, Y;
4    output [3:0] S;
5    output carryout;
6    wire [3:1] C;
7
8    fulladd stage0 (carryin, X[0], Y[0], S[0], C[1]);
9    fulladd stage1 (C[1], X[1], Y[1], S[1], C[2]);
10   fulladd stage2 (C[2], X[2], Y[2], S[2], C[3]);
11   fulladd stage3 (C[3], X[3], Y[3], S[3], carryout);
12 endmodule
```

figure3.24.v

```
1  module addern (carryin, X, Y, S, carryout);
2    parameter n=32;
3    input carryin;
4    input [n-1:0] X, Y;
5    output reg [n-1:0] S;
6    output reg carryout;
7    reg [n:0] C;
8    integer k;
9
10   always @(X, Y, carryin)
11   begin
12     C[0] = carryin;
13     for (k = 0; k < n; k = k+1)
14     begin
15       S[k] = X[k] ^ Y[k] ^ C[k];
16       C[k+1] = (X[k] & Y[k]) | (X[k] & C[k]) |
17                (Y[k] & C[k]);
18       end
19     carryout = C[n];
20   end
21 endmodule
```

figure3.25.v

```
1  module ripple_g (carryin, X, Y, S, carryout);
2    parameter n = 4;
3    input carryin;
4    input [n-1:0] X, Y;
5    output [n-1:0] S;
6    output carryout;
7    wire [n:0] C;
8
9    genvar i;
10   assign C[0] = carryin;
11   assign carryout = C[n];
12   generate
13     for (i = 0; i <= n-1; i = i+1)
14     begin:addbit
15       fulladd stage (C[i], X[i], Y[i], S[i], C[i+1]);
16     end
17   endgenerate
18 endmodule
```

figure3.26.v

```
1 module addern (carryin, X, Y, S);
2   parameter n = 32;
3   input carryin;
4   input [n-1:0] X, Y;
5   output reg [n-1:0] S;
6
7   always @(X, Y, carryin)
8     S = X + Y + carryin;
9 endmodule
```

figure3.27.v

```
1  module addern (carryin, X, Y, S, carryout, overflow);
2    parameter n = 32;
3    input carryin;
4    input [n-1:0] X, Y;
5    output reg [n-1:0] S;
6    output reg carryout, overflow;
7
8    always @(X, Y, carryin)
9    begin
10     S = X + Y + carryin;
11     carryout = (X[n-1] & Y[n-1]) | (X[n-1] & ~S[n-1]) |
12                (Y[n-1] & ~S[n-1]);
13     overflow = (X[n-1] & Y[n-1] & ~S[n-1]) |
14                (~X[n-1] & ~Y[n-1] & S[n-1]);
15   end
16 endmodule
```

figure3.28.v

```
1  module addern (carryin, X, Y, S, carryout, overflow);
2    parameter n = 32;
3    input carryin;
4    input [n-1:0] X, Y;
5    output reg [n-1:0] S;
6    output reg carryout, overflow;
7    reg [n:0] Sum;
8
9    always @(X, Y, carryin)
10   begin
11     Sum = {1'b0, X} + {1'b0, Y} + carryin;
12     S = Sum[n-1:0];
13     carryout = Sum[n];
14     overflow = (X[n-1] & Y[n-1] & ~S[n-1]) |
15                (~X[n-1] & ~Y[n-1] & S[n-1]);
16   end
17 endmodule
```

figure3.29.v

```
1  module addern (carryin, X, Y, S, carryout, overflow);
2    parameter n = 32;
3    input carryin;
4    input [n-1:0] X, Y;
5    output reg [n-1:0] S;
6    output reg carryout, overflow;
7
8    always @(X, Y, carryin)
9    begin
10     {carryout, S} = X + Y + carryin;
11     overflow = (X[n-1] & Y[n-1] & ~S[n-1]) |
12                (~X[n-1] & ~Y[n-1] & S[n-1]);
13   end
14 endmodule
```

figure3.30.v

```
1 module fulladd (Cin, x, y, s, Cout);
2   input Cin, x, y;
3   output reg s, Cout;
4
5   always @(x, y, Cin)
6       {Cout, s} = x + y + Cin;
7 endmodule
```

figure3.31.v

```
1  module adder_hier (A, B, C, D, S, T, overflow);
2    input [15:0] A, B;
3    input [7:0] C, D;
4    output [16:0] S;
5    output [8:0] T;
6    output overflow;
7    wire o1, o2; // used for the overflow signals
8
9    addern U1 (1'b0, A, B, S[15:0], S[16], o1);
10     defparam U1.n = 16;
11   addern U2 (1'b0, C, D, T[7:0], T[8], o2);
12     defparam U2.n = 8;
13   assign overflow = o1 | o2;
14 endmodule
```

figure3.32.v

```
1  module adder_hier (A, B, C, D, S, T, overflow);
2    input [15:0] A, B;
3    input [7:0] C, D;
4    output [16:0] S;
5    output [8:0] T;
6    output overflow;
7    wire o1, o2; // used for the overflow signals
8
9    addern #(16) U1 (1'b0, A, B, S[15:0], S[16], o1);
10   addern #(8) U2 (1'b0, C, D, T[7:0], T[8], o2);
11   assign overflow = o1 | o2;
12 endmodule
```

figure3.33.v

```
1  module adder_hier (A, B, C, D, S, T, overflow);
2    input [15:0] A, B;
3    input [7:0] C, D;
4    output [16:0] S;
5    output [8:0] T;
6    output overflow;
7    wire o1, o2; // used for the overflow signals
8
9    addern #(.n(16)) U1
10   (  .carryin (1'b0),
11     .X (A),
12     .Y (B),
13     .S (S[15:0]),
14     .carryout (S[16]),
15     .overflow (o1));
16   addern #(.n(8)) U2
17   (  .carryin (1'b0), .X (C), .Y (D), .S (T[7:0]),
18       .carryout (T[8]), .overflow (o2));
19   assign overflow = o1 | o2;
20 endmodule
```

Representação de Números em Verilog

Seguem o formato <#bits>'<base><valor>

- `12'b100010101001`
- `12'o4251`
- `12'h8A9`
- `12'd2217`

Representação de Números em Verilog

Seguem o formato <#bits>'<base><valor>

- `'b100010101001`
- `'o4251`
- `'h8A9`
- `2217`

Representação de Números em Verilog

Seguem o formato <#bits>'<base><valor>

- `'b1000_1010_1001`
- `'o4251`
- `'h8A9`
- `2217`

Concatenação e Extensão de Sinal

Supondo que A tem 8 bits e B tem 4 bits:

```
S = A + {4{B[3]}, B};
```

Bibliografia

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

Lógica Digital (1001351)

Projeto de Circuitos Aritméticos com Verilog

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: 10 de abril de 2019