Проектирование стандартных элементов цифровых интегральных схем. Основы HDL VERILOG

Ильин Сергей,
vermut.42@gmail.com

Литература

«Выполнение междисциплинарного задания в цепочке дисциплин «Языки описания цифровых схем и систем», «Лингвистические средства САПР», «Автоматизация функционально-логического проектирования БИС»

Авторы: Попова Т.В., Гусев С.В., Ильин С.А (под ред. Поповой Т.В.)

Алфавит. Представление чисел.

- 0 – логический ноль;
- 1 – логическая единица;
- Z – высокий импеданс;
- X – неопределенность (0 или 1)

Алфавит. Представление чисел.

- Бинарное (b)
- Восьмеричное (o)
- Десятичное (d)
- Шестнадцатеричное (h)

Разрядность	‘	Система исчисления	Значение

Запись чисел

15’h7a50 – шестнадцатеричное пятнадцатиразрядное число

8’b01101100 – бинарное восьмиразрядное число

8’b0110_1100 – бинарное восьмиразрядное число

Запись чисел

Пример:

8’o377 – восьмеричное восьмиразрядное число

16’d7110 – десятичное шестнадцатиразрядное число

7110 – десятичное число

Операторы

& – логическое умножение (AND)

| – логическое сложение (OR)

/ – арифметическое деление

* – арифметическое умножение

+ – арифметическое сложение

~ – инверсия (NOT)

! – инверсия (NOT)

^ – сложение по модулю 2 (XOR)

Структура модуля

module <имя> (<список портов>);

input <список входов>;

output <список выходов>;

inout <список двунаправленных портов>;

< операторы>

< операторы>

< операторы>

endmodule

Структура модуля

```
module nand2 (x,a,b);
output x;
input a, b;

assign x=!(a&b);

endmodule
```

Переменные wire

Значение переменной меняется сразу же после изменения какого-либо аргумента.

Используется для описания комбинационной логики.

Переменные wire

Используется только в составе конструкции “assign”.

```
wire a, b, c;
assign c = !(a & b );
```

Переменные reg

Присвоение нового значения переменной происходит после выполнения указанных условий. Используется для описания как комбинационной логики так и последовательностной логики.

Переменные reg

Используется в составе конструкции “always”.

Структура конструкции

always @ (<список чувствительности>)

begin

операнд 1;

операнд 2;

……..

операнд n;

end

Переменные reg

Пример

```
wire a, b;
reg c;
always @ (a)
c= ~ ( b & a );
```

Переменные reg

```
module and ( x, a, b );
   input a, b;
   output x;
   reg x;
     always @(a or b)
      x = a & b ;
endmodule
```

Уровни абстракции

– поведенческий (behavioral);

– вентильный (gate);

– уровень регистровых передач (Register Transmit Level / RTL)

Behavioral

Уровень представляет систему в виде параллельных алгоритмов.

Каждый алгоритм является последовательным, и представляет собой набор инструкций, выполняющиеся одна за другой.

Behavioral

```
module DFF (D, C, R, Q, nQ);
   input D,C,R;
   output Q, nQ;
   reg Q;
   always @(posedge C or posedge R)
     if(R)
      Q=1'b0;
        else
          Q=D;
   assign nQ=~Q;
endmodule
```

RTL

RTL описание имеет синхросигнал и все события происходят в определенное время.

Сами элементы используемые в RTL описании, могут иметь behavioral описание

RTL

```
module D_REG (D, C, R, Q);
   input C, R;
   input [7:0] D;
   output [7:0] Q;
   reg [7:0] Q;
    always @(posedge C or posedge R)
     if (R)
      Q=8'h00;
        else
          Q=D;
endmodule
```

RTL

```
module SUM (Pi, A, B, S, Po);
   input A, B;
   input Pi;
   output S;
   output Po;
     assign S=A^B^Pi;
     assign Po=A&B|A&Pi|B&Pi;
endmodule
```

RTL

```
`include "../SUM.v"
module SUMM (Pi, A, B, S, Po);
  input [7:0] A, B;
  input Pi;
  output [7:0] S;
  output Po;
  wire [6:0] p;
   SUM U0(.Pi(Pi),.A(A[0],.B(B[0]),.S(S[0]),  .Po(p[0]));
   SUM U1(.Pi(p[0]),.A(A[1]),.B(B[1]),.S(S[1]),.Po(p[1]));
   SUM U2(.Pi(p[1]),.A(A[2]),.B(B[2]),.S(S[2]),.Po(p[2]));
   SUM U3(.Pi(p[2]),.A(A[3]),.B(B[3]),.S(S[3]),.Po(p[3]));
   SUM U4(.Pi(p[3]),.A(A[4]),.B(B[4]),.S(S[4]),.Po(p[4]));
   SUM U5(.Pi(p[4]),.A(A[5]),.B(B[5]),.S(S[5]),.Po(p[5]));
   SUM U6(.Pi(p[5]),.A(A[6]),.B(B[6]),.S(S[6]),.Po(p[6]));
   SUM U7(.Pi(p[6]),.A(A[7]),.B(B[7]),.S(S[7]),.Po(Po));
endmodule
```

RTL

```
`include "../SUMM.v"
`include "../D_REG.v"
module COUNT (C,R,Q);
   input C,R;
   output [7:0] Q;
   wire [7:0] D;

D_REG REG_1 (.D(D), .C(C), .R(R),.Q(Q));
SUMM SUM_1 (.Pi(1'b1),.A(Q),.B(7'h00),.S(D));

endmodule
```

Gate-Level

Уровень, описание которого построено на основе логических примитивов, каждый из которых реализует свою логическую функцию

Gate-Level.
Встроенные примитивы

– buf – функция повторения сигнала
– not – инвертор
– and – функция логическое умножение
– or – функция логическое сложение
– xor – функция логическое сложение по модулю 2
– nand – функция логическое умножение с инверсией
– nor – функция логическое сложение с инверсией
– xnor – функция логическое сложение по модулю 2 с инверсией

Gate-Level.

```
module ELEM (A, B, F);
  input A,B;
  output F;
  wire n1, n2, n3;
   nand (n1, A, B);
   nor(n2, n1, A);
   xor(n3, n1, B);
   not(F, n2, n3);
endmodule
```

Gate-Level.

```
`include "../lib/DFFRX1.v"
module D_REG (D, C, R, Q);
  input C,R;
  input [7:0] D;
  output [7:0] Q;
    DFFRX1 reg_0 (.D(D[0]),.R(R),.C(C),.Q(Q[0]));
    DFFRX1 reg_1 (.D(D[1]),.R(R),.C(C),.Q(Q[1]));
    DFFRX1 reg_2 (.D(D[2]),.R(R),.C(C),.Q(Q[2]));
    DFFRX1 reg_3 (.D(D[3]),.R(R),.C(C),.Q(Q[3]));
    DFFRX1 reg_4 (.D(D[4]),.R(R),.C(C),.Q(Q[4]));
    DFFRX1 reg_5 (.D(D[5]),.R(R),.C(C),.Q(Q[5]));
    DFFRX1 reg_6 (.D(D[6]),.R(R),.C(C),.Q(Q[6]));
    DFFRX1 reg_7 (.D(D[7]),.R(R),.C(C),.Q(Q[7]));
endmodule
```

Testbench

```
`include "../prim/ELEM.v"
`timescale 1ns/1ps
module tb;
reg [1:0] I;
wire Q;
 ELEM u1 (I[0], I[1], Q);
initial
 begin
   I=0; $display("-------------");
   $display("| B | A | F |"); $display("-------------");
   while (I<3)
     begin
       #5 I=I+1;
       $display("| %b | %b | %b |",I[1],I[0],Q);
     end
   #5 $display("| %b | %b | %b |",I[1],I[0],Q);
   $display("-------------");
$finish;
end
endmodule
```