1 组合逻辑电路--基本门电路

1.1 基本门电路

1.1.1 结构化描述方式

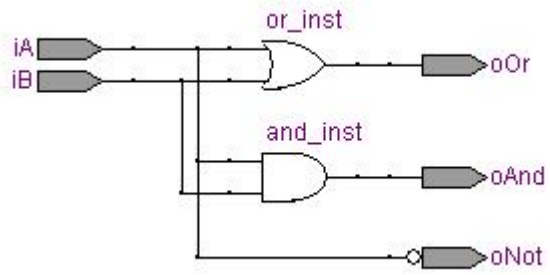
代码如下

View Code

```
1 module logics
2 (
3    input iA,
4    input iB,
5    output oAnd,
6    output oOr,
7    output oNot
8 );
9
10 and and_inst(oAnd,iA,iB);
11 or  or_inst(oOr,iA,iB);
12 not not_inst(oNot,iA);
13
14 endmodule
```

最底层的是门级原语 and or not

RTL 级视图



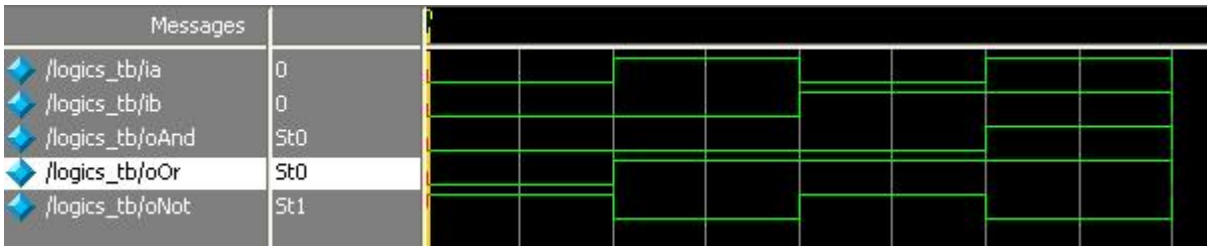
testbench 如下

View Code

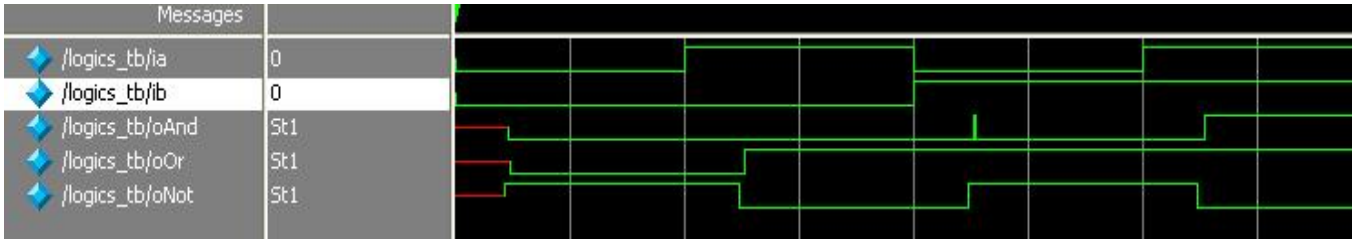
```
1 `timescale 1 ns/ 1 ns
2 module logics_tb();
3
4 reg ia;
5 reg ib;
6
```

```
 7 wire oAnd;
 8 wire oOr;
 9 wire oNot;
10
11 initial
12 begin
13     ia=0;
14     #40 ia=1;
15     #40 ia=0;
16     #40 ia=1;
17     #40 ia=0;
18 end
19
20 initial
21 begin
22     ib=0;
23     #40 ib=0;
24     #40 ib=1;
25     #40 ib=1;
26     #40 ib=0;
27 end
28
29 logics logics_inst
30 (
31     .iA(ia),
32     .iB(ib),
33     .oAnd(oAnd),
34     .oOr(oOr),
35     .oNot(oNot)
36 );
37
38 endmodule
```

RTL 级仿真图形如下



GATE 级仿真图如下



可见 RTL 级仿真是理想的，GATE 级仿真考虑了延迟和信号开始的不确定。

1.1.2 采用流描述方法

代码如下

View Code

```
module logics
(
    input iA,
    input iB,
    output oAnd,
    output oOr,
    output oNot
);

assign oAnd=iA&iB;
assign oOr=iA|iB;
assign oNot=~iA;

endmodule
```

RTL 级视图，仿真图形同上。

1.1.3 采用行为描述方式

代码如下

View Code

```
module logics
(
    input iA,
    input iB,
    output reg oAnd,
    output reg oOr,
    output reg oNot
);

always @(*)
begin
```

```
12     oAnd=iA&iB;
13     oOr=iA|iB;
14     oNot=~iA;
15 end
16
17 endmodule
```

always@()括号内的敏感信号填*，则综合器自动加上敏感信号。

由于 always 语句中左边信号都要是寄存器型，故输出信号定义为寄存器型。描述组合逻辑时，always 中使用阻塞赋值方式。

RTL 级视图及仿真图形同上。

2 组合逻辑电路--多路选择器与多路分解器

1.2 多路选择器

1.2.1 不带优先级的多路选择器

四路选择器如下

代码如下

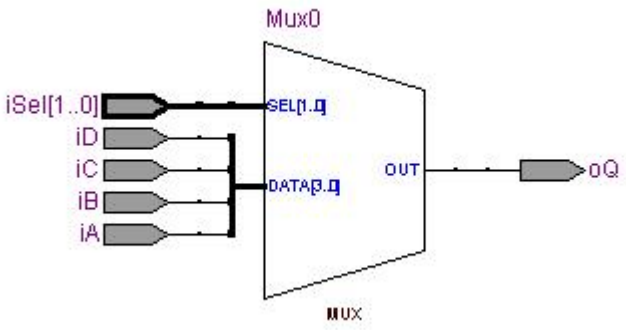
View Code

```
1 module multiplexer
2 (
3     input iA,
4     input iB,
5     input iC,
6     input iD,
7     input [1:0] iSel,
8     output reg oQ
9 );
10
11 always @(*)
12 begin
13     case(iSel)
14         2'b00 : oQ=iA;
15         2'b01 : oQ=iB;
16         2'b10 : oQ=iC;
17         2'b11 : oQ=iD;
18     endcase
19 end
```

```
20
21 endmodule
```

case 语句如果没有全部包含所有的情况，则一般要用 default 语句将信号默认赋值为 0，否则会出现锁存器，使得仿真结果不一致。

RTL 级视图如下：



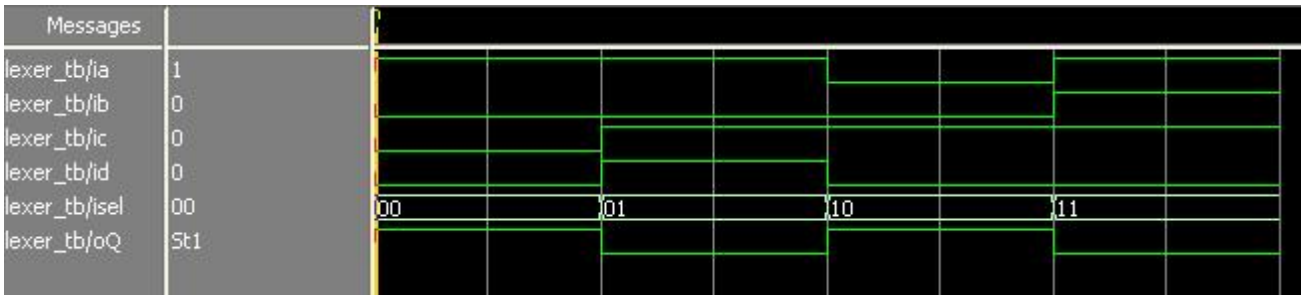
testbench 如下

```
View Code
 1 `timescale 1 ns/ 1 ns
 2 module logics_tb();
 3
 4 reg ia;
 5 reg ib;
 6
 7 wire oAnd;
 8 wire oOr;
 9 wire oNot;
10
11 initial
12 begin
13     ia=0;
14     #40 ia=1;
15     #40 ia=0;
16     #40 ia=1;
17     #40 ia=0;
18 end
19
20 initial
21 begin
22     ib=0;
23     #40 ib=0;
24     #40 ib=1;
```

```
25      #40 ib=1;
26      #40 ib=0;
27  end
28
29  logics logics_inst
30  (
31      .iA(ia),
32      .iB(ib),
33      .oAnd(oAnd),
34      .oOr(oOr),
35      .oNot(oNot)
36  );
37
38  endmodule
```

initial 语句和 always 语句左边的信号类型都要为 reg。

RTL 级仿真图形如下



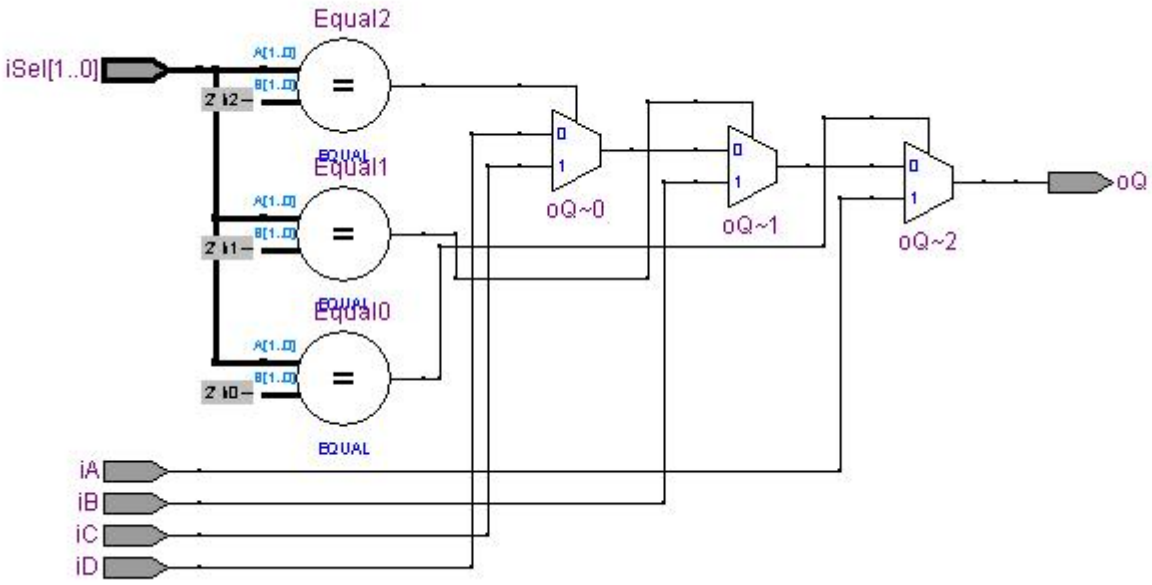
1.2.2 带优先级的多路选择器

代码如下

```
View Code
 1  module multiplexer
 2  (
 3      input iA,
 4      input iB,
 5      input iC,
 6      input iD,
 7      input [1:0] iSel,
 8      output reg oQ
 9  );
10
11  always @(*)
12      if(iSel==2'b00)
13          oQ=iA;
14      else if(iSel==2'b01)
15          oQ=iB;
```

```
16     else if(iSel==2'b10)
17         oQ=iC;
18     else
19         oQ=iD;
20
21 endmodule
```

RTL 级视图如下



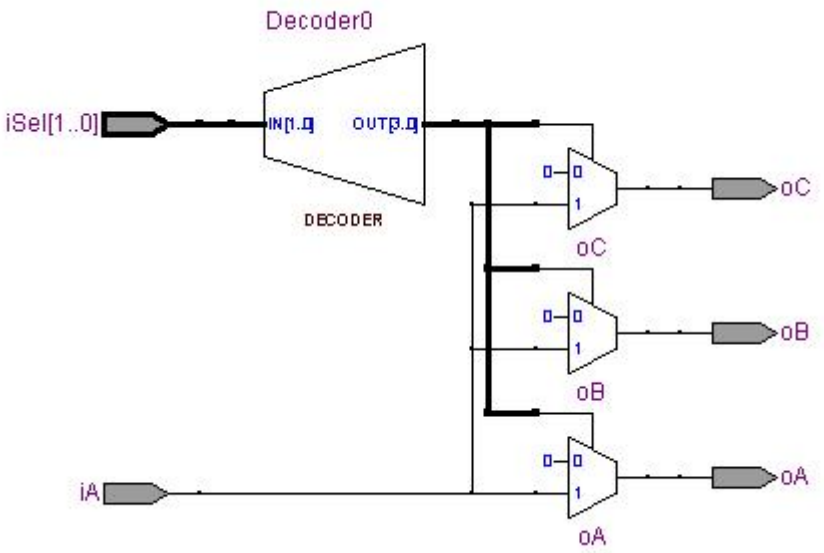
1.2.3 多路分解器

一路输入，三路输出分解器代码如下

```
View Code
 1 module de_multiplexer
 2 (
 3     input iA,
 4     input [1:0] iSel,
 5     output reg oA,
 6     output reg oB,
 7     output reg oC
 8 );
 9
10 always @(*)
11 begin
12     oA=0;oB=0;oC=0;
13     case(iSel)
14         2'b00 : oA=iA;
```

```
15        2'b01 : oB=iA;
16        2'b10 : oC=iA;
17        default :;
18    endcase
19 end
20
21 endmodule
```

RTL 级视图如下



testbench 代码如下

```
View Code
 1 `timescale 1ns/1ns
 2 module de_multiplexer_tb;
 3 reg ia;
 4 reg [1:0] isel;
 5 wire oa;
 6 wire ob;
 7 wire oc;
 8
 9 initial
10 begin
11     ia=0;
12     forever
13         #10 ia=~ia;
14 end
15
16 initial
17 begin
```

```
18     isel=2'b00;
19     #20 isel=2'b01;
20     #20 isel=2'b10;
21     #20 isel=2'b11;
22     #20 $stop;
23 end
24
25 de_multiplexer de_multiplexer_inst
26 (
27     .iA(ia),
28     .iSel(isel),
29     .oA(oa),
30     .oB(ob),
31     .oC(oc)
32 );
33
34 endmodule
```

RTL 级仿真波形如下

1 组合逻辑电路--编码器和译码器

1.3.1 编码器

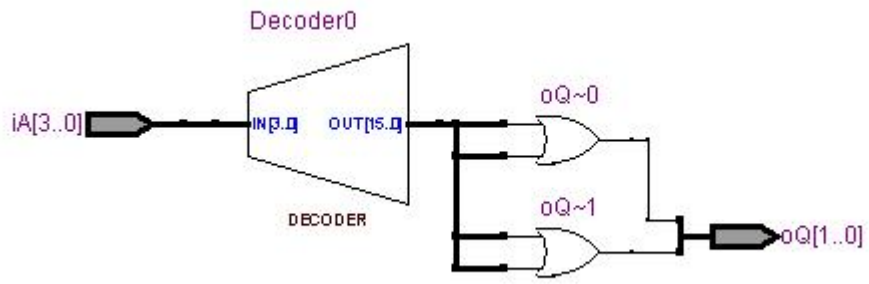
4 输入 2 输出编码器代码如下

View Code

```
1 module encoder
2 (
3     input [3:0] iA,
4     output reg [1:0] oQ
5 );
6
7 always @(*)
8 begin
9     oQ=2'b00;
10    case(iA)
```

```
        4'b0001:oQ=2'b00;
        4'b0010:oQ=2'b01;
        4'b0100:oQ=2'b10;
        4'b1000:oQ=2'b11;
        default :;
    endcase
end

endmodule
```

RTL 级视图如下



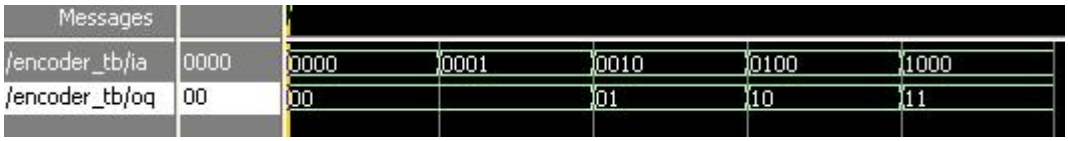
testbench 代码如下

View Code

```
`timescale 1ns/1ns
module encoder_tb;

reg [3:0] ia;
wire [1:0] oq;

initial
begin
    ia=4'b0000;
    #20 ia=4'b0001;
    #20 ia=4'b0010;
    #20 ia=4'b0100;
    #20 ia=4'b1000;
    #20 $stop;
end

encoder encoder_inst
(
    .iA(ia),
    .oQ(oq)
);
```

```
22
23 endmodule
```

RTL 级仿真波形如下



1.3.2 优先编码器

真值表如下，参考艾米电子

i3	i2	i1	i0	o1	O0
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

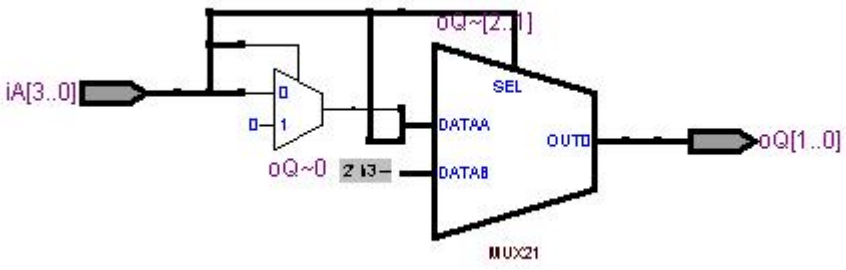
艾米电子

代码如下

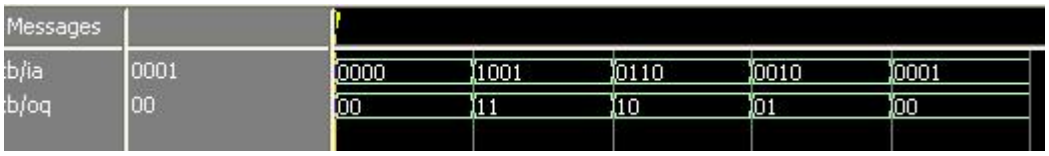
View Code

```
1 module encoder
2 (
3     input [3:0] iA,
4     output reg [1:0] oQ
5 );
6
7 always @(*)
8 begin
9     oQ=2'b00;
10    if(iA[3])oQ=2'b11;
11    else if(iA[2])oQ=2'b10;
12    else if(iA[1])oQ=2'b01;
13    else oQ=2'b00;
14 end
15
16 endmodule
```

RTL 级视图



RTL 级仿真波形



1 组合逻辑电路--算术运算电路

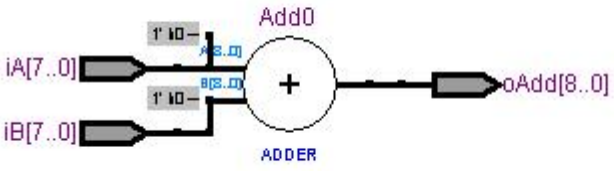
1.4.1 +、-、*、/、%电路

（1）加法电路：每 1 位大约消耗 1 个 LE，示例代码如下

View Code

```
1 module arithmetic
2 (
3     input [7:0] iA,
4     input [7:0] iB,
5     output [8:0] oAdd
6 );
7
8 assign oAdd=iA+iB;
9
10 endmodule
```

RTL 级视图如下

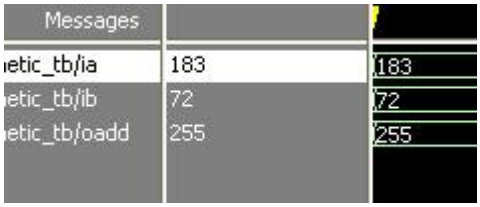


testbench 如下

View Code

```
1 `timescale 1ns/1ns
2 module arithmetic_tb;
3
4 reg [7:0] ia=8'b1011_0111;
5 reg [7:0] ib=8'b0100_1000;
6 wire [8:0] oadd;
7
8 initial #100 $stop;
9
10 arithmetic arithmetic_inst
11 (
12     .iA(ia),
13     .iB(ib),
14     .oAdd(oadd)
15 );
16
17 endmodule
```

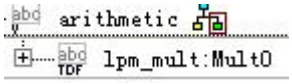
仿真波形如下



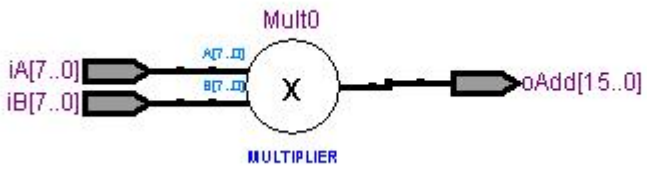
（2）乘法电路

代码将加法电路代码中语句 oAdd=iA+iB;改为　oMul=iA*iB；即可。oMul 位宽为[15:0]

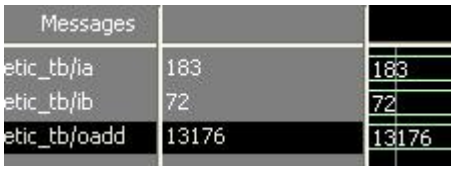
代码综合后乘法单元直接调用的 CycloneII 嵌入式乘法器 lpm_mult



RTL 级视图



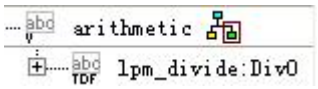
仿真波形图



（3）除法电路

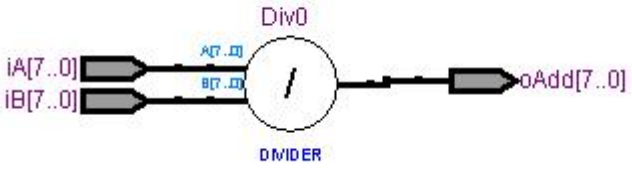
代码将加法电路代码中语句 oAdd=iA+iB;改为　oDiv=iA/iB；即可。oDiv 位宽为[7:0]

代码综合后除法单元直接调用的 lpm_divide 实现的

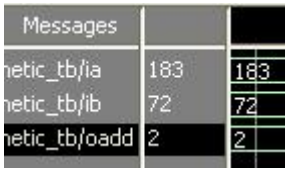


取余运算也很消耗资源，每位运算大约消耗 10LE.

RTL 级视图如下



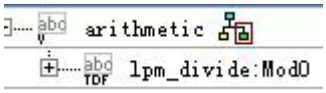
RTL 级波形仿真



(4)取余运算

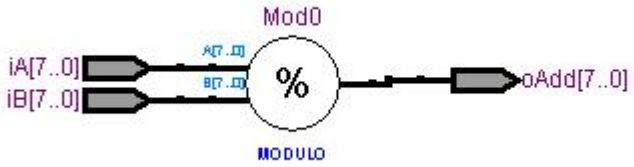
代码将加法电路代码中语句 oAdd=iA+iB;改为　oMod=iA%iB；即可。oMod 位宽为[7:0]

代码综合后取余运算直接调用的 lpm_divide 实现的

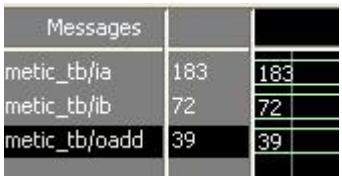


除法运算很消耗资源，每位运算大约消耗 10LE.

RTL 级视图如下



RTL 级波形仿真



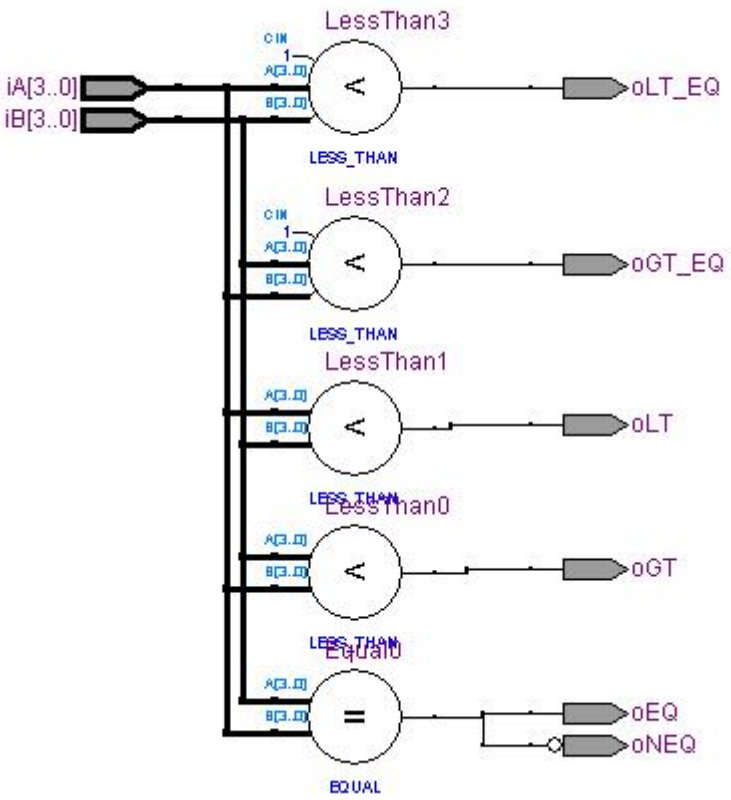
1.4.2 数据比较器

代码如下，风格非常类似 C 语言

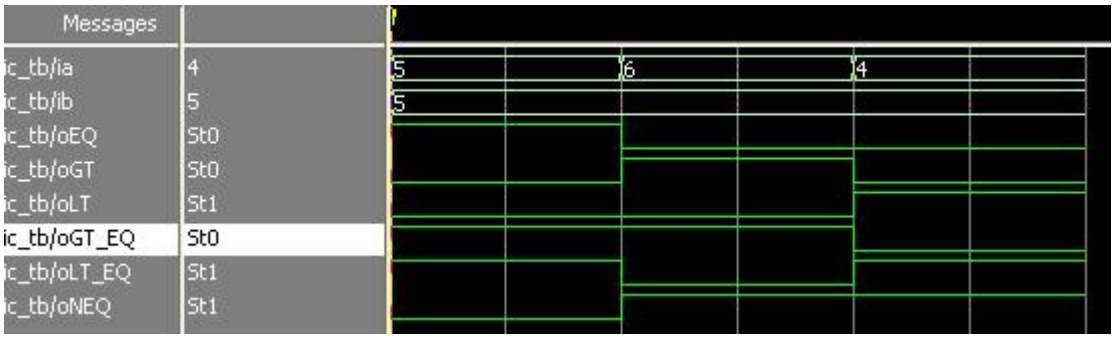
View Code

```
module arithmetic
(
    input [3:0] iA,
    input [3:0] iB,
    output oEQ,
    output oGT,
    output oLT,
    output oGT_EQ,
    output oLT_EQ,
    output oNEQ
);

assign    oEQ=(iA==iB),
        oNEQ=(iA!=iB),
        oGT=(iA>iB),
        oLT=(iA<iB),
        oGT_EQ=(iA>=iB),
        oLT_EQ=(iA<=iB);
endmodule
```

RTL 级视图如下



RTL 级仿真波形如下



1.4.3 移位电路

1.4.3.1 逻辑移位电路

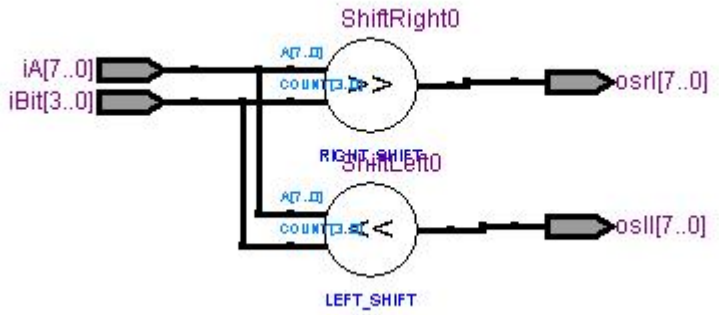
代码如下

View Code

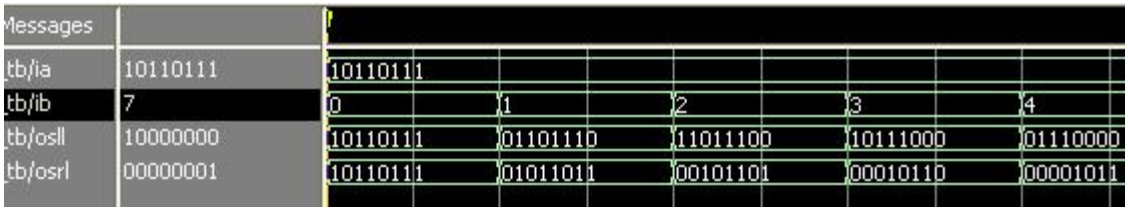
```
module arithmetic
(
    input [7:0] iA,
    input [3:0] iBit,
    output [7:0] osll,
    output [7:0] osrl
);
```

```
 8
 9 assign    osll=iA<<iBit,
10         osrl=iA>>iBit;
11 endmodule
```

RTL 级视图如下



RTL 级仿真波形



1.4.3.2 算术移位

算术移位运算符为<<<,>>>;

有符号数算术移位和逻辑移位，两者左移时一样，右移时逻辑移位最高位填充的是 0，算术移位最高位填充的是符号位。

2 时序逻辑电路--触发器与锁存器

2.1.1 同步复位 D 触发器

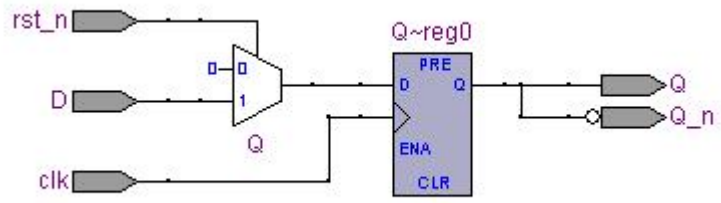
复位信号在所需时钟边沿才有效，复位操作需要同步于时钟故称作同步复位。

代码如下

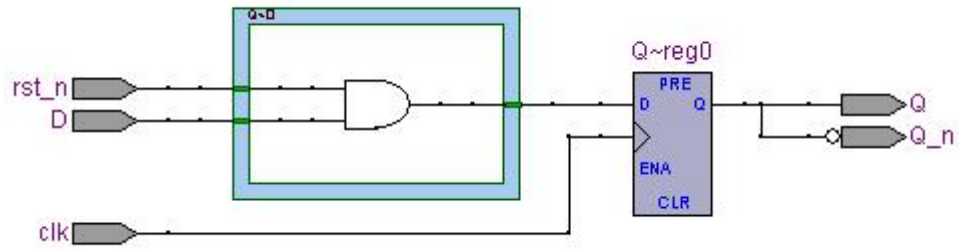
```
View Code
 1 module d_ff
 2 (
 3     input clk,
 4     input rst_n,
 5     input D,
```

```
    output reg Q,
    output Q_n
);

always @(posedge clk)
    if(!rst_n)Q<=1'b0;
    else Q<=D;

assign Q_n=~Q;

endmodule
```

RTL 级视图



Technology Map Viewer 视图如下



2.1.2 异步复位 D 触发器

将上述语句改为 always @(posedge clk or negedge rst_n)即构成异步复位 D 触发器。只要复位信号有效，输出立即复位。

很多写法中都采取异步复位模式，后面就不在讨论。

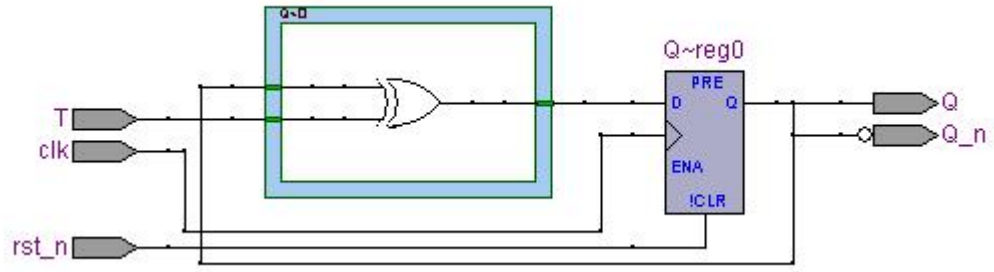
2.1.3 T 触发器

代码如下

```
View Code
1 module t_ff
2 (
```

```
    input clk,
    input rst_n,
    input T,
    output reg Q,
    output Q_n
);

always @ (posedge clk or negedge rst_n)
    if(!rst_n) Q<=1'b0;
    else if(T) Q<=~Q;

assign Q_n=~Q;

endmodule
```

Technology Map View 如下



testbench 如下

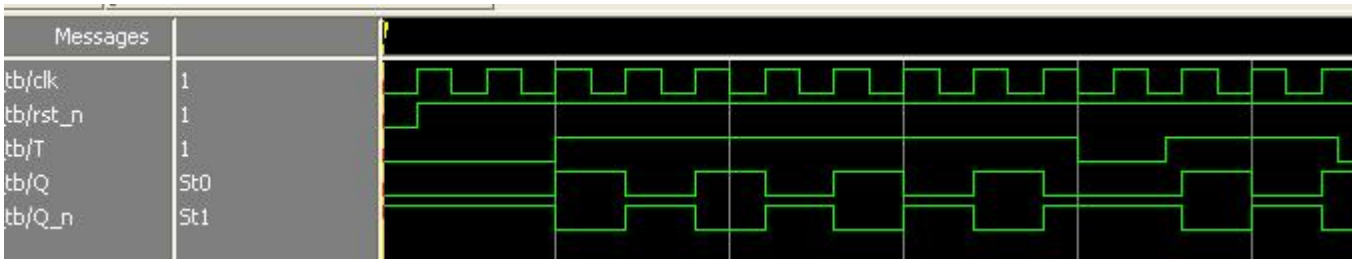
View Code

```
`timescale 1ns/1ns
module t_ff_tb;
reg clk,rst_n,T;
wire Q,Q_n;
//**************************************
//时钟与复位激励
parameter CLK_PERIOD=20,
          RESET_TIME=10;
initial clk=0;
initial forever #(CLK_PERIOD/2) clk=~clk;
initial rst_n=0;
initial forever #RESET_TIME rst_n=1;
//**************************************
initial T=0;
initial forever #25 T={$random}%2;
initial #200 $stop;
```

```
17
18 t_ff t_ff_inst
19 (
20     .clk(clk),
21     .rst_n(rst_n),
22     .Q(Q),
23     .Q_n(Q_n),
24     .T(T)
25 );
26
27 endmodule
```

{$random}%2 表示产生 0~2-1 之间的随机正整数。

RTL 级仿真波形



2.1.4 门控 D 触发器

当 clk 为高电平时输出 Q 才随着输入 D 变化，clk 为低电平时输出保持不变。门控 D 触发器表现的是组合逻辑电路的特点。

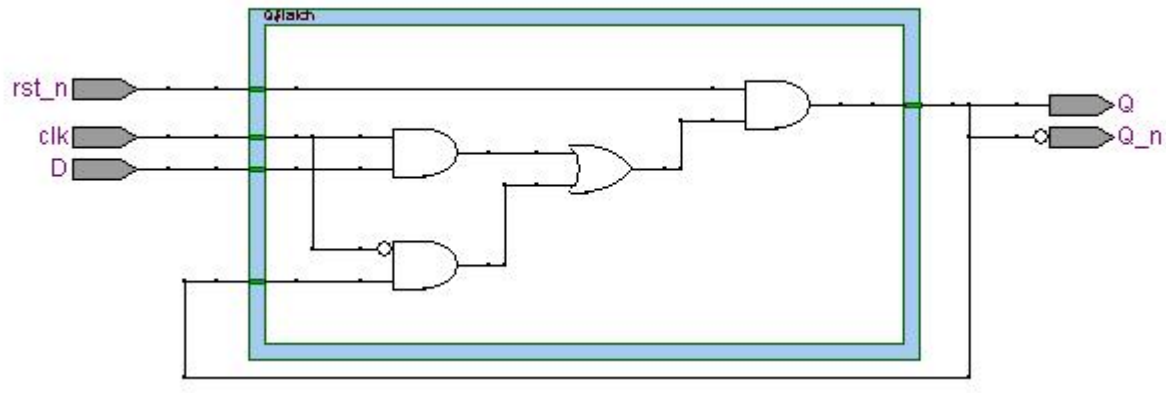
代码如下

View Code

```
1 module t_ff
2 (
3     input clk,
4     input rst_n,
5     input D,
6     output reg Q,
7     output Q_n
8 );
9
10 always @ (*)
11     if(!rst_n) Q<=1'b0;
12     else if(clk) Q<=D;
13
14 assign Q_n=~Q;
```

```
15
16 endmodule
```

Technology Map Viewer 如下



2.1.5 亚稳态

对于 D 触发器，clk 上升沿时输出 Q 等于输入 D，其余时间保持不变。clk 上升沿到来前 D 必须保持稳定的最短时间为触发器建立时间 t_su,clk 上升沿后 D 必须保持稳定的最短时间为触发器保持时间 t_h。如果设计使得 t_su,t_h 不满足要求，触发器就会出现不稳定状态，称作亚稳态。Q 随 D 变化而变化所需要的时间称作 clk 到 D 得传播延迟 t_cQ。

2 时序逻辑电路--寄存器

2.2.1 寄存器

寄存器是 D 触发器集合，不过其有位宽，将前述 D 触发器语句改为 input D;outpu Q;改为 input [7:0]D;outpu[7:0] Q;即构成一个 8b 的异步复位寄存器。

2.2.2 寄存器文件

将寄存器以数组整合起来，并加上输入地址，输出地址，即构成可临时快速数据存储的寄存器文件。

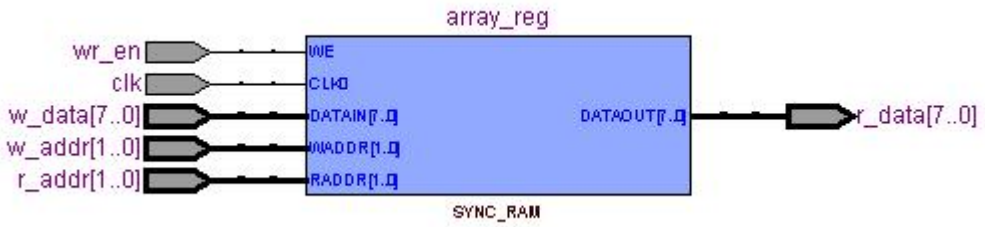
示例代码

```
View Code
1 module reg_file
2 #(
3     parameter B=8,
4     parameter W=2
```

```
5 )
6 (
7     input clk,
8     input wr_en,
9     input [W-1:0] w_addr,
10    input [W-1:0] r_addr,
11    input [B-1:0] w_data,
12    output [B-1:0] r_data
13 );
14
15 reg [B-1:0] array_reg[2**W-1:0];
16
17 always @ (posedge clk)
18    if(wr_en) array_reg[w_addr]<=w_data;
19
20 assign r_data=array_reg[r_addr];
21
22 endmodule
```

代码中直接利用 FPGA 中寄存器构成寄存器文件。其中 reg [B-1:0] array_reg[2**w-1:0]; 表示 reg[B-1:0]类型的寄存器组，个数为 2**W 个=4。

RTL 级视图



FPGA EP2C8Q 中每个 LE 含有一个异步复位、同步使能的 D 触发器，4 个 LUT。如果用寄存器存储数据将消耗很多 LE,建议用

器件内嵌的 RAM。

2 时序逻辑电路--移位寄存器

2.3.1 自动右移寄存器

信号由 s_in 进入 8 位寄存器最高位，寄存器最低位由 s_out 输出，代码如下

View Code

```
module free_run_shift
(
    input clk,
    input rst_n,
    input s_in,
    output s_out
);

reg [7:0] r_reg;//现态寄存器
wire [7:0] r_next;//次态寄存器

always @(posedge clk or negedge rst_n)//时钟上升沿将次态寄存器值赋给现态寄存器，更新数据
    if(!rst_n)
        r_reg<=0;
    else r_reg<=r_next;

assign r_next={s_in,r_reg[7:1]};
assign s_out=r_reg[0];

endmodule
```

2.3.2 通用移位寄存器

可以并入并出，串入并出，并入串出。ctrl=11 时，次态寄存器 r_next 得到输入值 Q。 ctrl=00 时，次态值等于现态值，不变。ctrl=01 时，次态值左移移位，最低位填充 D[0]。ctrl=11 时，次态值右移移位，最高位填充 D[N]。

View Code

```
module free_run_shift
#(parameter N=8)
(
    input clk,
    input rst_n,
    input [1:0] ctrl,
    input [N-1:0] D,
    output [N-1:0] Q
);

reg [N-1:0] r_reg;
reg [N-1:0] r_next;

always @(posedge clk or negedge rst_n)
    if(!rst_n)
```

```
16        r_reg<=0;
17    else r_reg<=r_next;
18
19 always @(*)
20    case(ctrl)
21        2'b00:r_next=r_reg;
22        2'b01:r_next={r_reg[N-2:0],Q[0]};
23        2'b10:r_next={Q[N-1],r_reg[N-1:1]};
24        2'b11:r_next=D;
25        default:;
26    endcase
27
28 assign Q=r_reg;
29
30 endmodule
```

2 时序逻辑电路--计数器

2.4.1 二进制计数器

代码如下

View Code

```
1 module counter
2 #(parameter N=8)
3 (
4    input clk,
5    input rst_n,
6    output [N-1:0] Q,
7    output Max_tick
8 );
9
10 reg [N-1:0] r_reg;
11 wire [N-1:0] r_next;
12
13 always @ (posedge clk or negedge rst_n)
14    if(!rst_n)
15        r_reg<=0;
16    else r_reg<=r_next;
17
18 assign r_next=r_reg+1'b1;
19 assign Q=r_reg;
20 assign Max_tick=(r_reg==2**N-1)?1'b1:1'b0;
```

```
21
22 endmodule
```

2.4.2 具有控制位的计数器

设计相应的控制线，根据设计的真值表编写代码，参考 COM 缺氧设计的真值表如下，需要说明的是 rst_n 异步清零主要用在系统的初始化，而 syn_ctrl 同步清零是在一般操作中使用。

syn_ctrl	load	en	up/down	Q次态	操作
1	x	x	x	0…0	同步清零
0	1	x	x	Q=D	并行载入
0	0	1	1	Q+1	增计数
0	0	1	0	Q-1	减计数
0	0	0	x	Q	保持

代码如下

View Code

```
1 module counter
2 #(parameter N=8)
3 (
4     input clk,
5     input rst_n,
6     input syn_ctrl,
7     input load,
8     input en,
9     input up,
10    output [N-1:0] Q,
11    output Max_tick,
12    output Min_tick
13 );
14
15 reg [N-1:0] r_reg;
16 reg [N-1:0] r_next;
17
18 always @ (posedge clk or negedge rst_n)
19    if(!rst_n)
20        r_reg<=0;
21    else r_reg<=r_next;
22
23 always @(*)
24    if(syn_ctrl)r_next=0;
25    else if(load) r_next=Q;
26    else if(en & up)r_next=r_reg+1'b1;
27    else if(en & ~up)r_next=r_reg-1'b1;
28    else r_next=r_reg;
```

```
29
30 assign Q=r_reg;
31 assign Max_tick=(r_reg==2**N-1)?1'b1:1'b0;
32 assign Min_tick=(r_reg==0)?1'b1:1'b0;
33
34 endmodule
```

将上述代码 N 值改为 3，编写 testbench 如下

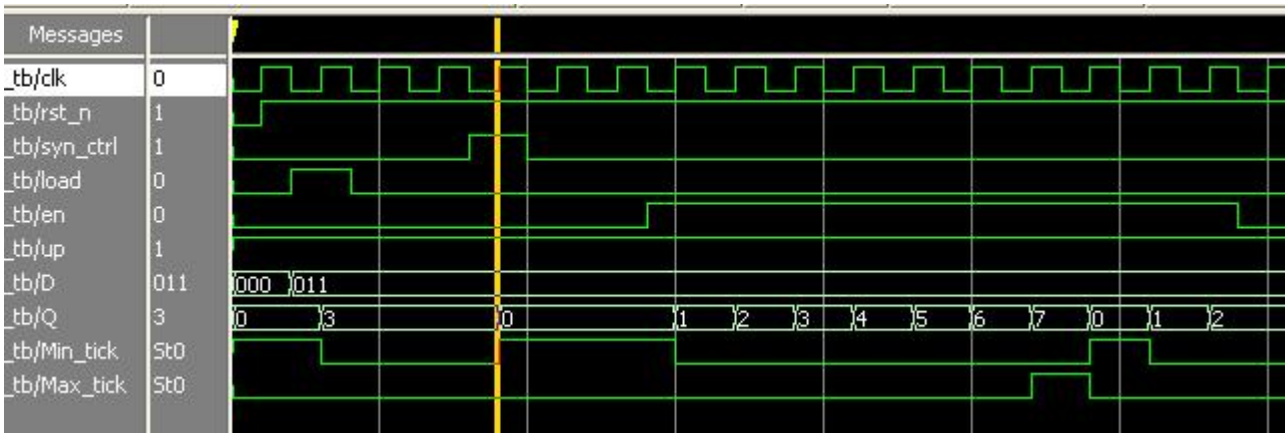
View Code

```
 1 `timescale 1ns/10ps
 2 module counter_tb;
 3 reg clk,rst_n,syn_ctrl,load,en,up;
 4 reg [2:0] D;
 5 wire [2:0] Q;
 6 wire Min_tick;
 7 wire Max_tick;
 8
 9 counter counter_inst
10 (
11     .clk(clk),
12     .rst_n(rst_n),
13     .syn_ctrl(syn_ctrl),
14     .load(load),
15     .up(up),
16     .en(en),
17     .D(D),
18     .Q(Q),
19     .Min_tick(Min_tick),
20     .Max_tick(Max_tick)
21 );
22
23 //*************************************
24 //时钟与复位激励
25 parameter CLK_PERIOD=20,
26           RESET_TIME=10;
27 initial clk=0;
28 initial forever #(CLK_PERIOD/2) clk=~clk;
29 initial rst_n=0;
30 initial forever #RESET_TIME rst_n=1;
31 //*************************************
32
33 initial
34 begin
35     //initial ctrl_signal input
```

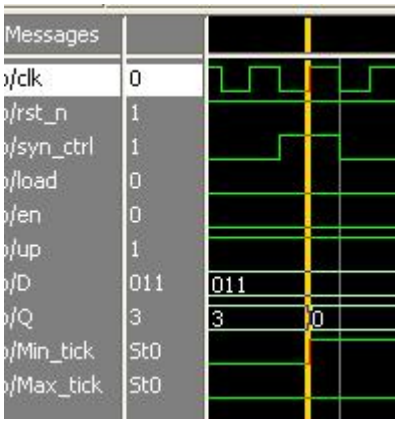
```
36    syn_ctrl=1'b0;
37    load=1'b0;
38    en=1'b0;
39    up=1'b1;
40    //initial data_signal input
41    D=3'b000;
42    //initial ctrl_signal input
43    @(posedge rst_n);//wait rst_n to deassert
44    @(negedge clk);//wait for one clk
45    //test load
46    load=1'b1;
47    D=3'b011;
48    @(negedge clk)
49    load=1'b0;
50    repeat(2) @(negedge clk);
51    //test syn_ctrl
52    syn_ctrl=1'b1;
53    @(negedge clk)
54    syn_ctrl=1'b0;
55    repeat(2) @(negedge clk);
56    //test up counter and pause
57    en=1'b1;
58    up=1'b1;
59    repeat(10) @(negedge clk);
60    en=1'b0;
61    repeat(2) @(negedge clk);
62    en=1'b1;
63    repeat(2) @(negedge clk);
64    //test down mod
65    up=1'b0;
66    repeat(10) @(negedge clk);
67    //test Max_tick and Min_tick
68    wait(Q==2);
69    @(negedge clk);
70    up=1'b1;
71    @(negedge clk);
72    wait(Min_tick);
73    @(negedge clk);
74    up=1'b0;
75    //delay
76    #(4*CLK_PERIOD);
77    en=1'b0;
78    #(4*CLK_PERIOD);
79    $stop;
```

```
80 end
81
82 endmodule
```

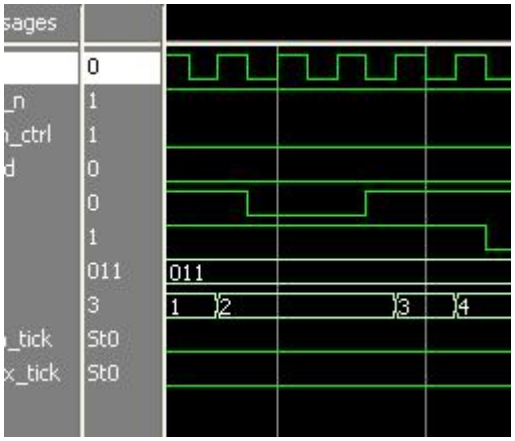
RTL 级仿真波形如下



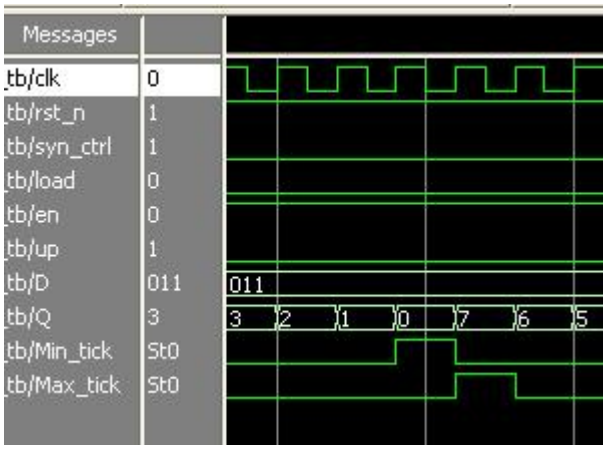
验证 syn_ctrl 波形如下



验证 load 信号波形如下



验证 Max_tick,Min_tick 波形如下



2.4.3 模 m 计数器

将上述二进制计数器判断语句 r_reg==2**N-1 改为 r_reg==M-1;即构成模 M 计数器。