EDA Technology

# Chapter 7

# Deep Understanding of Verilog HDL

# Two Types of Assignment Statements in Process

- Blocking Assignment with Unspecified Time-delay

  *target variable name=drive expression;*

# Two Types of Assignment Statements in Process

- Blocking Assignment with Specified Time-delay

*[delay] target variable name=drive expression;*

*target variable name= [delay] drive expression;*

```
... //procedural statement
    Y1 = A^B;
#6 Y2 = A & B | C;
```

```
... /procedural statement
Y1 = A^D;
Y2 = #6 A & E | C;
```

```
assign  Q1 = A|B;
assign  Q1 = B&C;
assign  Q1 = ~C;
```

```
begin
Q1 = A|B;
Q1 = B&C;
Q1 = ~C ;    end
```

```
begin
Q1 <= A|B;
Q1 <= B&C;
Q1 <= ~C ;    end
```

```
assign  Q1 = A|B;
assign  Q1 = B&C;
assign  Q1 = ~C;
```

```
begin
Q1 = A|B;
Q1 = B&C;
Q1 = ~C ;    end
```

```
begin
Q1 <= A|B;
Q1 <= B&C;
Q1 <= ~C ;    end
```

```
begin
Y1 = #6 A^B;
Y2 = #4 A|B;
Y3 = #7 A&B;
end
```

```
begin
Y1 <= #6 A^B;
Y2 <= #4 A|B;
Y3 <= #7 A&B;
end
```

```
begin
Y1  = #5 A^B;
Y2 <= #3 A|B;
Y3 <= #2 A&B;
Y4  = #4 (~B);  end
```

```
module DDF3(CLK,D,Q);
input CLK,D;  output Q; reg a,b,Q;
always @(posedge CLK) begin
a <= D;
b <= a;
Q <= b;     end
endmodule
```

```
module DFF3(CLK,D,Q);
input CLK,D; output Q;reg a,b,Q;
always @(posedge CLK) begin
a = D;
b = a;
Q = b;       end
endmodule
```

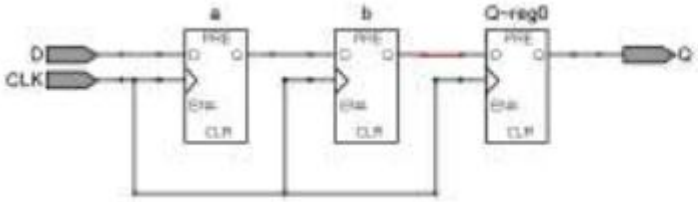


Figure: The RTL diagram of Example 7-11 after synthesis

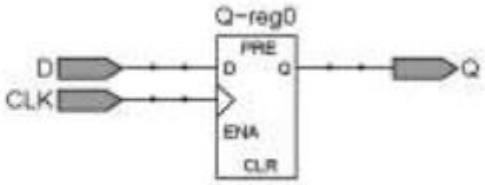


Figure: The RTL diagram of Example 7-12 after synthesis

# Further Discussion of Different Initialization Ways

```
module MUX41a(D,S,DOUT);
  output DOUT ;
  input [3:0] D; input [1:0] S;
  integer T;  reg DOUT;
  always @(D,S)
  begin   T <= 0;
    if (S[0]==1)   T<=T+1;
    if (S[1]==1)   T<=T+2;
    case (T)
      0 : DOUT = D[0] ;
      1 : DOUT = D[1] ;
      2 : DOUT = D[2] ;
      3 : DOUT = D[3] ;
    default : DOUT = D[0] ;
    endcase    end
endmodule
```

```
module MUX41a(D,S,DOUT);
  output DOUT ;
  input [3:0] D; input [1:0] S;
  integer T; reg DOUT;
  always @(D,S)
  begin   T = 0;
    if (S[0]==1)   T=T+1;
    if (S[1]==1)   T=T+2;
    case (T)
      0 : DOUT = D[0] ;
      1 : DOUT = D[1] ;
      2 : DOUT = D[2] ;
      3 : DOUT = D[3] ;
    default : DOUT = D[0] ;
    endcase    end
endmodule
```

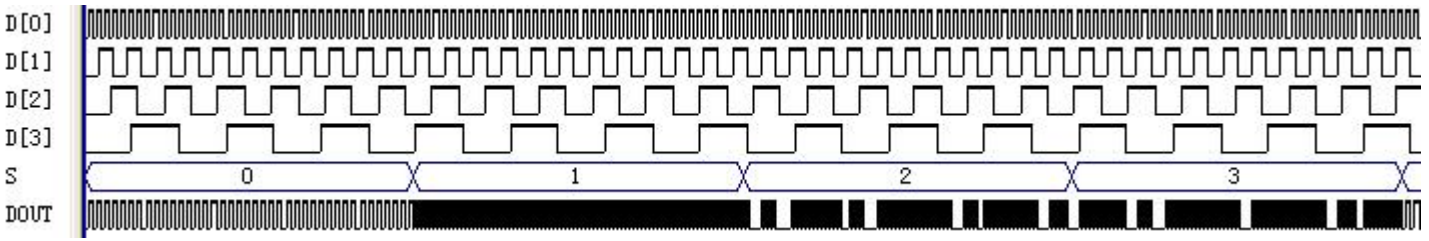


Figure: The operational timing of Example 7-13

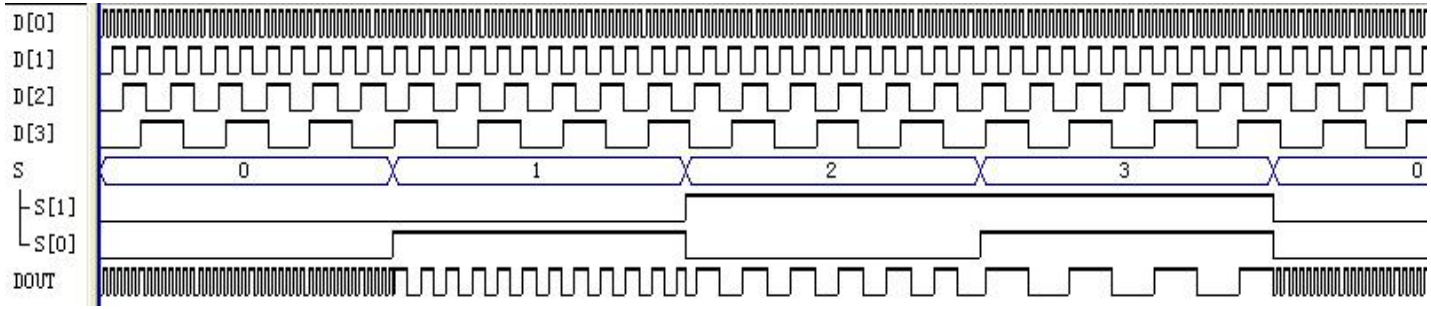


Figure: The operational timing of Example 7-14

# Discussion of Procedural Statement

```
module mux2_1
  (CLK, D, Q, RST);
   output Q;
   input CLK,D,RST;
   reg Q;
always @(D, CLK, RST)
   if(CLK)  Q <= D;
      else  Q <= RST;
endmodule
```

```
module COMP(A,B,Q);
input[3:0] A,B;
output Q;  reg Q;
always @(A,B)
  begin
   if(A>B) Q=1'b1;
    else
   if(A<B)  Q=1'b0;
end   endmodule
```

```
module COMP(A,B,Q);
input[3:0] A,B;
output Q;   reg Q;
always @(A,B)
  begin
      if(A>B) Q=1'b1;
else  if(A<B) Q=1'b0;
      else     Q=1'bz;
end     endmodule
```

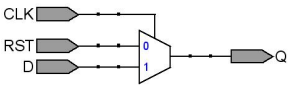


Figure:  The RTL diagram of example 7-15

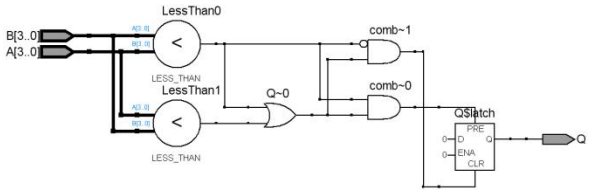


Figure:  The RTL diagram of Example 7-16, and the output port is added by latch.

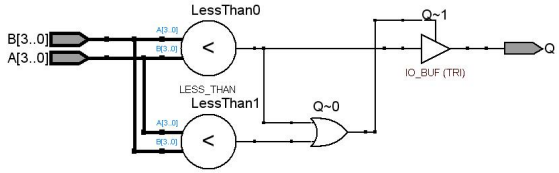


Figure 7-7  The circuit of Example 7-17, which is a pure combinational circuit.

# Design of Three-state and Bidirectional Port

```
module tri4B (ENA,DIN,DOUT);
   input ENA;
input [3:0] DIN ;
   output [3:0] DOUT ;
   reg [3:0] DOUT;
   always @(DIN,ENA)
     if (ENA)   DOUT <= DIN ;
          else    DOUT <= 4'HZ;
endmodule
```

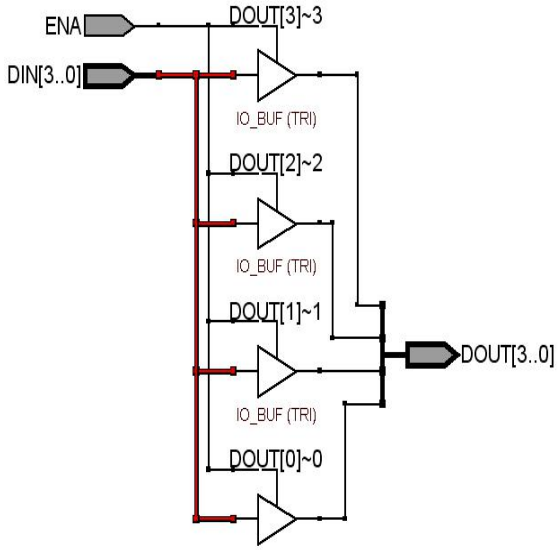


Figure: 4-bit three-state control gate circuit

```
module bi4b(TRI_PORT,DOUT,DIN,ENA,CTRL);
   inout TRI_PORT;   input DIN,ENA,CTRL;
output DOUT ;
   assign TRI_PORT = ENA ? DIN : 1'bz;
   assign DOUT = TRI_PORT | CTRL;
 endmodule
```

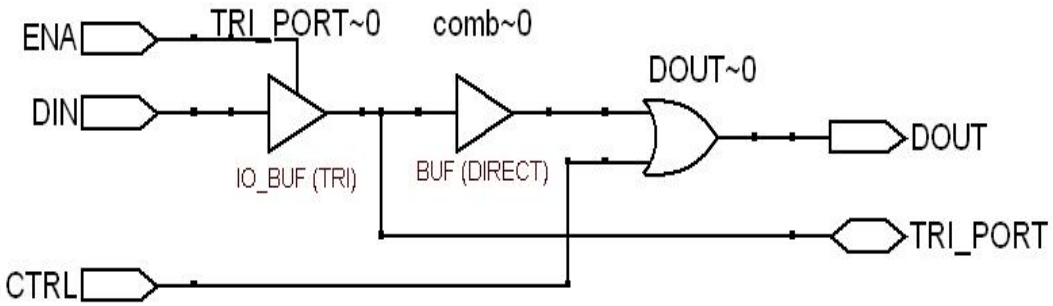


Figure: The RTL diagram of circuit design of 1-bit bidirectional port in Example 7-19

```
module BI4B(CTRL,DIN,Q,DOUT);
  input CTRL;     input[3:0] DIN;
  inout[3:0]Q; output[3:0] DOUT;
  reg [3:0] DOUT,Q ;
  always @(Q,DIN,CTRL)
    if (!CTRL)  DOUT<=Q ;
      else
  begin Q<=DIN; DOUT<=4'HZ;  end
    endmodule
```

```
module BI4B(CTRL,DIN,Q,DOUT);
  input CTRL;     input[3:0] DIN;
  inout[3:0] Q; output[3:0] DOUT;
  reg [3:0] DOUT,Q ;
always @(Q,DIN,CTRL)
if (!CTRL)  begin DOUT<=Q;
       Q<=4'HZ; end  else
begin Q<=DIN;  DOUT<=4'HZ;  end
endmodule
```

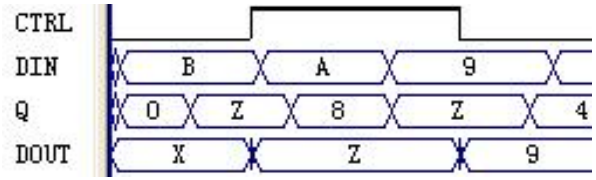


Figure: The simulation waveform of Example 7-20

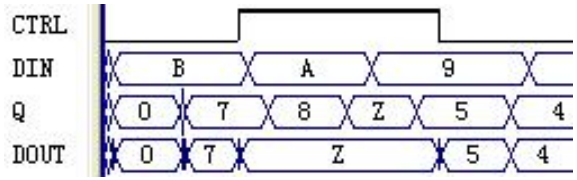


Figure: The simulation waveform of Example 7-21

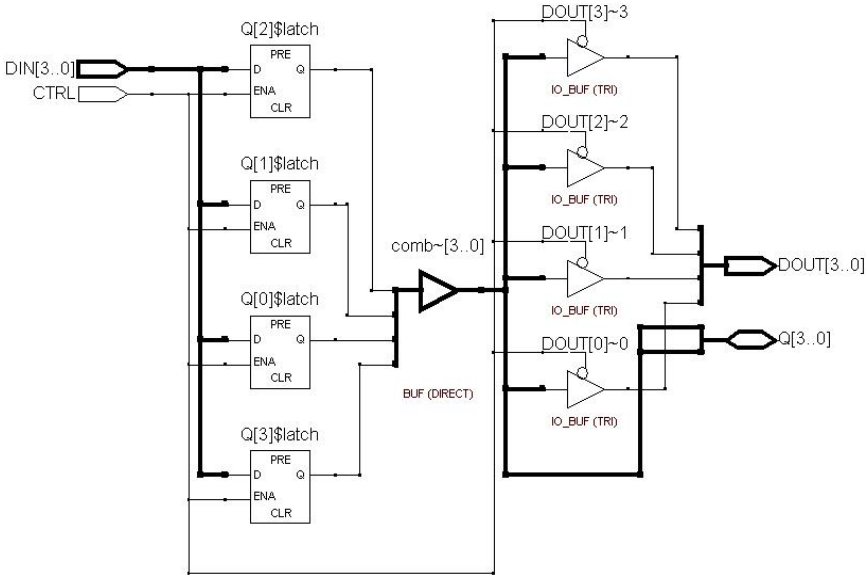


Figure: The RTL diagram of Example 7-20

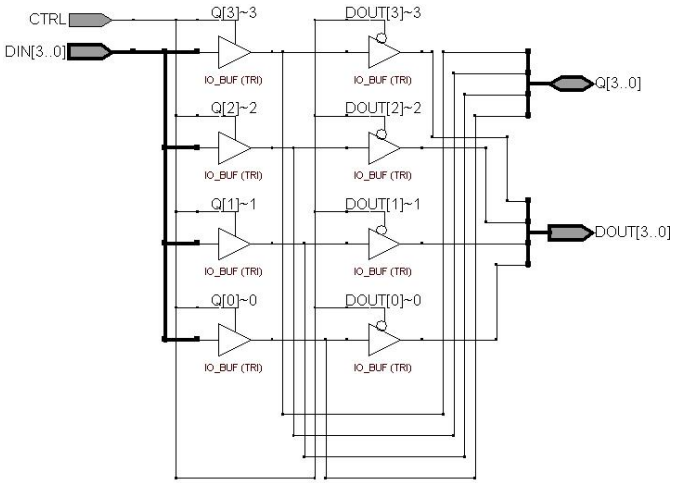


Figure: The RTL diagram of Example 7-21

# Design of Three-state Bus Control Circuit

```
module triBUS4(
  IN3,IN2,IN1,IN0,ENA,DOUT);
  input[3:0] IN3,IN2,IN1,IN0 ;
  input[1:0]  ENA;
  output[3:0] DOUT;
  reg[3:0] DOUT;
  always @(ENA,IN3,IN2,IN1,IN0)
    begin
    if (ENA==0)  DOUT=IN3 ;
        else  DOUT=4'HZ;
    if (ENA==1)  DOUT=IN2 ;
        else  DOUT=4'HZ;
    if (ENA==2)  DOUT=IN1 ;
        else  DOUT=4'HZ;
    if (ENA==3)  DOUT=IN0 ;
        else  DOUT=4'HZ;
    end
endmodule
```

```
module triBUS4(
    IN3,IN2,IN1,IN0,ENA,DOUT);
  input[3:0] IN3,IN2,IN1,IN0 ;
  input[1:0]  ENA;
  output[3:0] DOUT; reg[3:0]DOUT;
  always @(ENA, IN0)
  if (ENA==2'b00)  DOUT=IN0;
      else  DOUT=4'hz;
  always @(ENA, IN1)
  if (ENA==2'b01)  DOUT=IN1;
      else  DOUT=4'hz;
  always @(ENA, IN2)
  if (ENA==2'b10)  DOUT=IN2;
      else  DOUT=4'hz;
  always @(ENA, IN3)
  if (ENA==2'b11)  DOUT=IN3;
      else  DOUT=4'hz;
endmodule
```

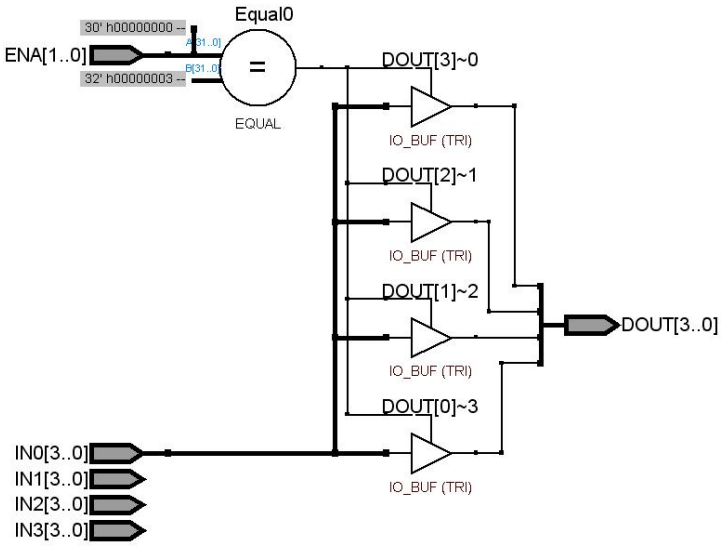


Figure: The RTL diagram of Example 7-22

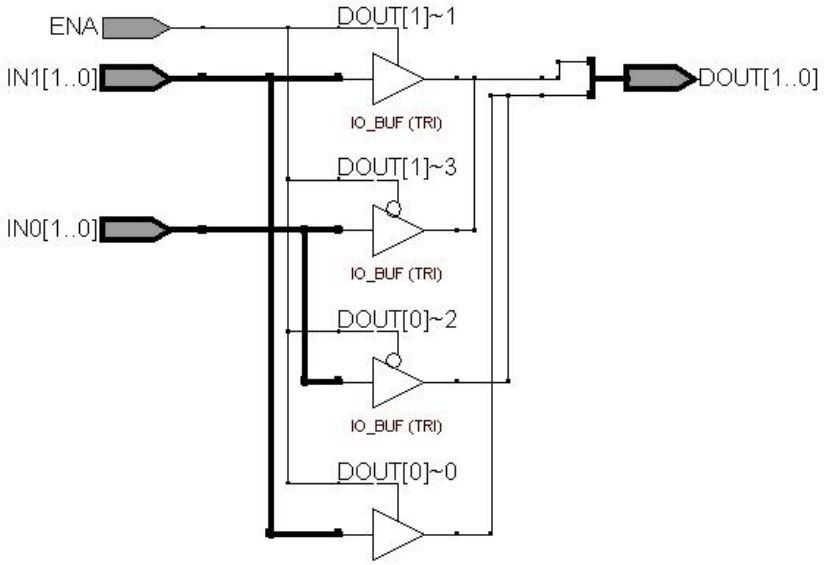


Figure: The 2-bit simplified RTL diagram of Example 7-23

# Resource Sharing

```
module multmux (A0, A1, B, S, R);
  input[3:0] A0, A1, B;  input S;
  output[7:0] R;   reg[7:0] R;
  always @(A0 or A1 or B or S)
    begin
    if (S==1'b0)  R <= A0*B;
      else  R <= A1*B ;   end
endmodule
```

```
module multmux (A0, A1, B, S,R);
  input[3:0] A0, A1, B;  input S;
  output[7:0] R;   wire [7:0] R;
   reg [3:0] TEMP;
  always @(A0 or A1 or B or S)
  begin  if (S==1'b0)  TEMP<=A0;
   else  TEMP <= A1;   end
  assign R = TEMP*B;
endmodule
```

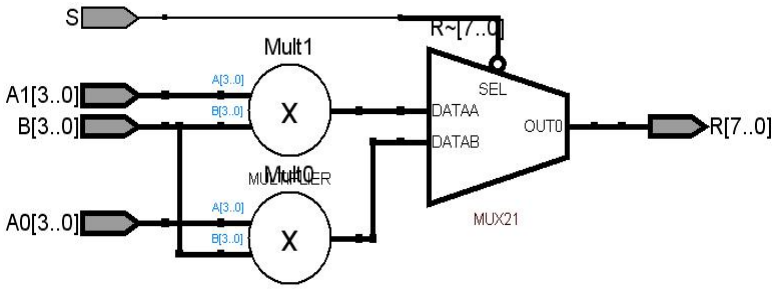


Figure: The RTL structure of first multiplication and next selection design method

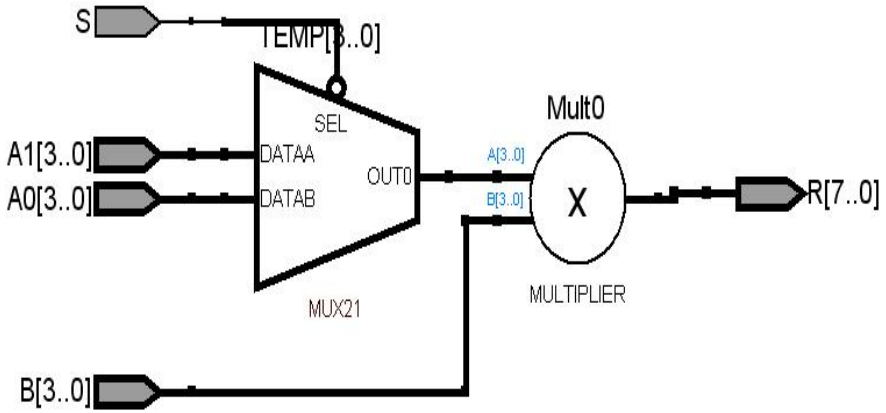


Figure: The RTL structure of first selection and next multiplication design method

# Logic Optimization

In the practical design, we always meet the case that two numbers multiply with each other and one of them is a constant.

```
module mult1 (clk, ma, mc);
  input clk;  input[11:0] ma;
  output[23:0] mc;
  reg[23:0] mc; reg[11:0] ta,tb;
  always @(posedge clk)
  begin  ta<=ma; mc<=ta * tb;
    tb <= 12'b100110111001; end
endmodule
```

```
module mult2  (clk, ma, mc);
  input clk;  input[11:0] ma;
  output[23:0] mc;
  reg[23:0] mc;   reg[11:0] ta;
  parameter tb=12'b100110111001;
  always @(posedge clk)
  begin  ta<=ma ; mc<=ta * tb; end
endmodule
```

If it is adapted to the EPF10K20，the example of 9-3 totally consumes LC of 167 and the example of 9-4 consumes LC of 93.

# Serialization

- `yout = a0b0 + a1b1 + a2b2 + a3b3`

**[ Example ]**

```
module pmultadd  (clk, a0, a1, a2, a3, b0, b1, b2, b3, yout);
   input clk;   input[7:0] a0, a1, a2, a3, b0, b1, b2, b3;
   output[15:0] yout;    reg[15:0] yout;
   always @(posedge clk)   begin
      yout <= ((a0 * b0)+(a1 * b1))+((a2 * b2)+(a3 * b3)) ;   end
endmodule
```

It totally consumes 4 8-bit multiplier and some adder. If it is adapted to EP3C5, 460 LC will be consumed.

**The serialization is to (1) split up the parallel execution logic which consumes large resource and complete within one clock period; (2) extract the same logic module; (3) multiplex the logic module in time and finish the same functionality by using several clock periods. The drawback of this method is the lowering of the operation speed.**

**[ Example ]**

```
module smultadd (clk, start, a0, a1, a2, a3, b0, b1, b2, b3, yout);
    input clk, start; input[7:0] a0, a1, a2, a3, b0, b1, b2, b3;
    output[15:0] yout;   reg[15:0] yout, ytmp;   reg[2:0] cnt;
    wire[7:0] tmpa, tmpb; wire[15:0] tmp;
assign tmpa=(cnt==0)? a0:(cnt==1)? a1:(cnt==2)? a2:(cnt==3)? a3:a0;
assign tmpb=(cnt==0)? b0:(cnt==1)? b1:(cnt==2)? b2:(cnt==3)? b3:b0;
assign tmp = tmpa * tmpb ;
always @(posedge clk) begin
    if (start==1'b1) begin cnt<=3'b000 ; ytmp<={16{1'b0}} ; end
    else if (cnt<4)   begin cnt<=cnt+1 ; ytmp<=ytmp+tmp ; end
    else if (cnt==4)   begin yout<=ytmp ;  end   end
endmodule
```

It only needs one 8-bit multiplier and one 16-bit adder. Moreover, 3-bit binary counter and tow large selector are needed. If the same synthesis/adaption configuration is used, 186 LC will be consumed.

# Speed optimization

**The design of pipeline**

The pipeline technology is one of the most commonly used technologies in the speed optimization. It can significantly enhance the upper limit of the operational speed of the circuit design.

In fact, adding the pipeline into the design will not reduce the delay of the design. Sometimes, it will a little bit increase the delay of the inserted register. However, it will enhance the overall operation speed.

# Speed Optimization

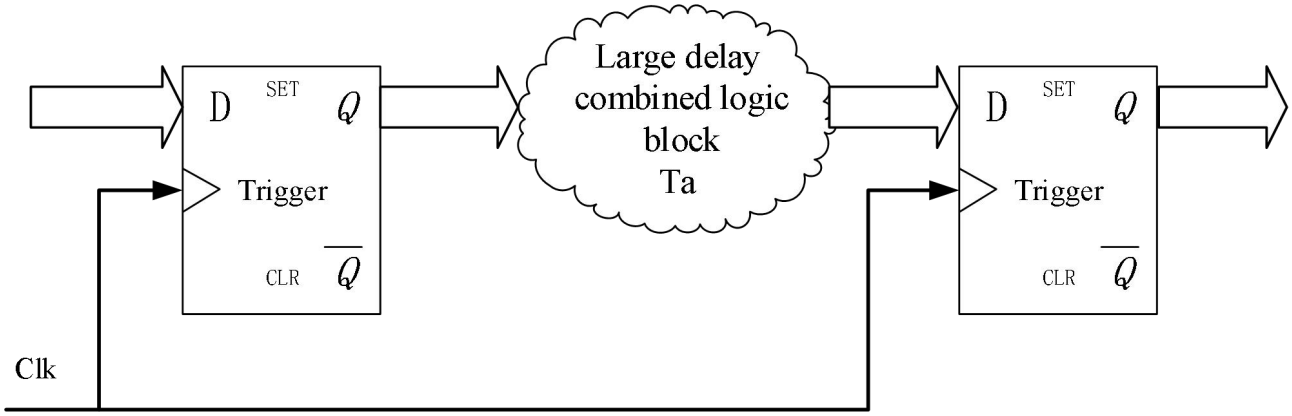


Figure: The unuse of pipeline

$$T_a \le T_{CLK} \Rightarrow \frac{1}{T_a} \ge \frac{1}{T_{CLK}} = f_{CLK}$$

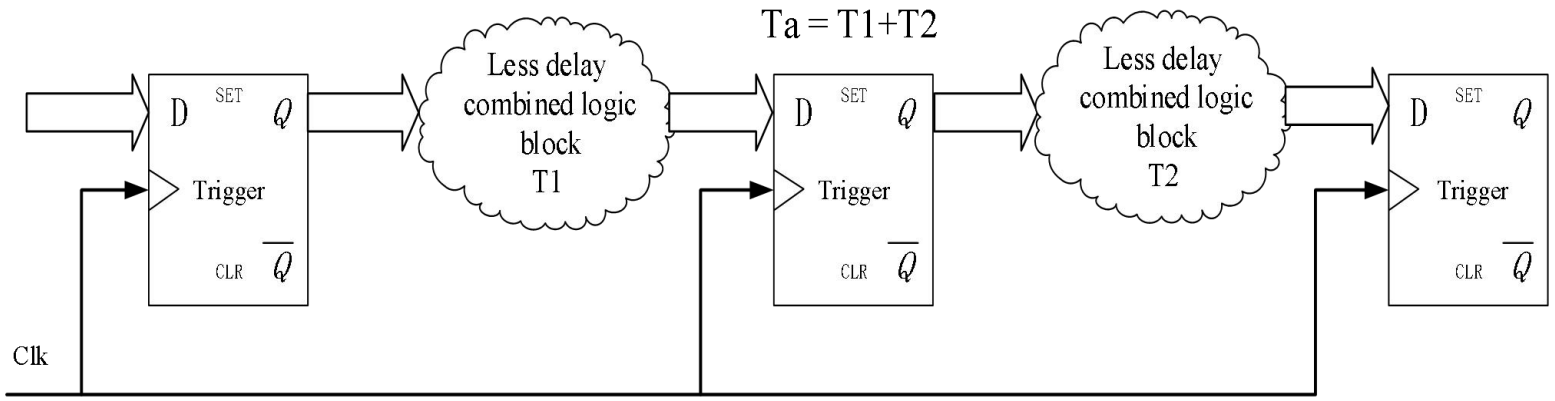


Figure: The use of pipeline structure

$$F_{max} \approx F_{max1} \approx F_{max2} \approx 1/T_1$$

$$T_1 \le T_{CLK} \Rightarrow \frac{1}{T_1} \ge \frac{1}{T_{CLK}} \Rightarrow \frac{2}{T_a} \ge \frac{1}{T_{CLK}} = f_{CLK}$$

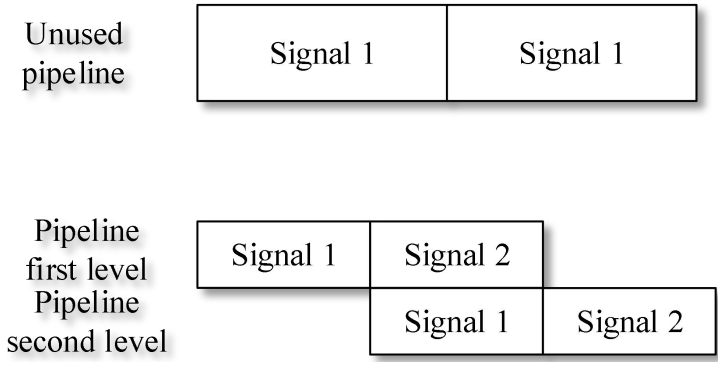


Figure: The working diagram of pipeline

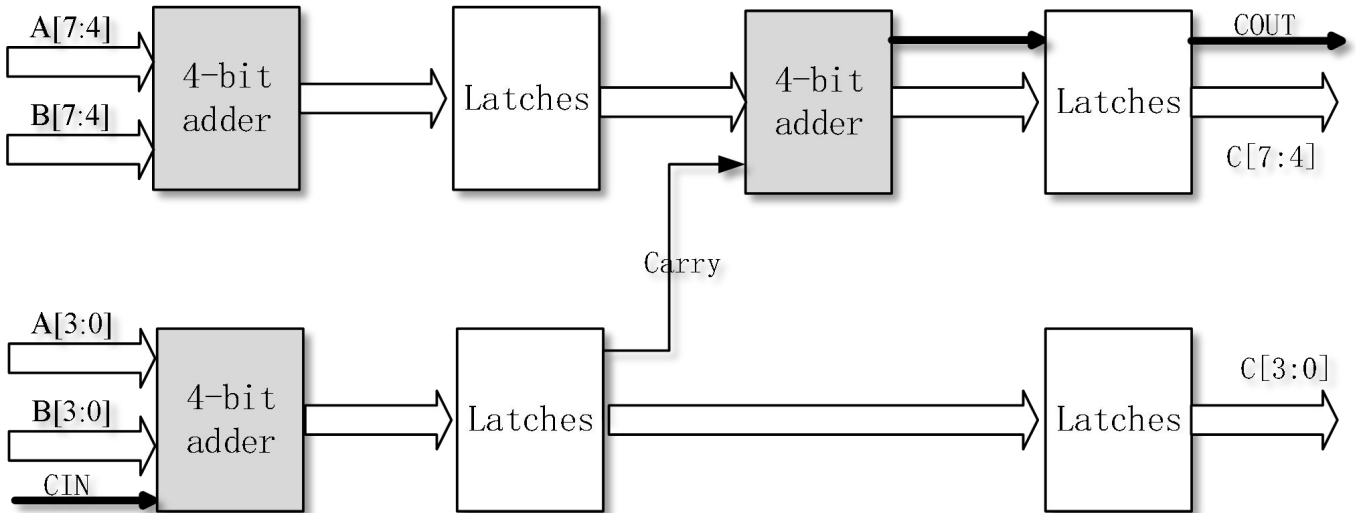


Figure: The working diagram of 8-bit adder pipeline

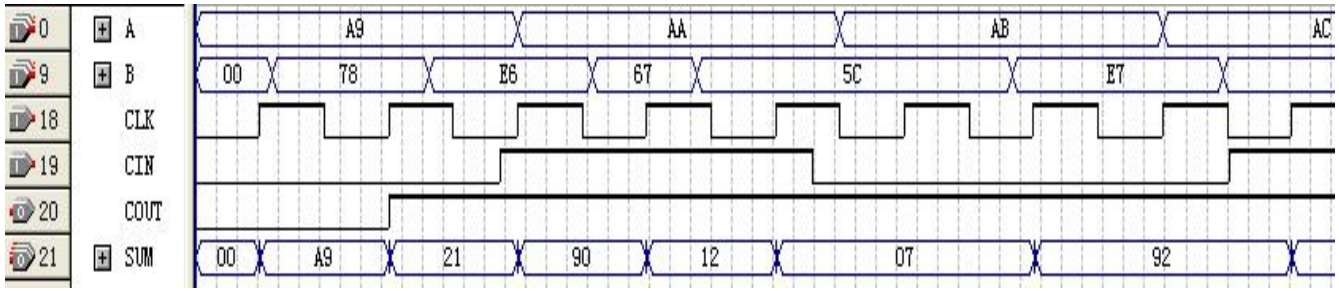


Figure: The timing simulation waveform of Example 7-30

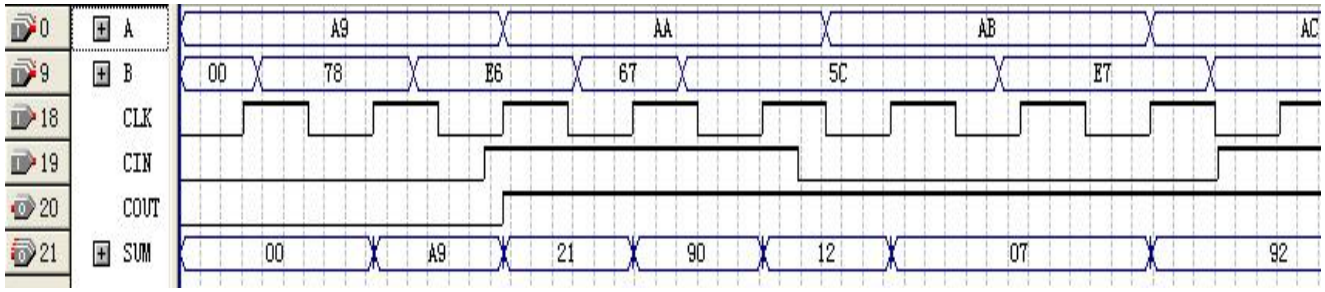


Figure: The timing simulation waveform of Example 7-31

[ **Example** ] Ordinary adder, and the synthesis result of EP3C10: LCs=10，REG=0，T=7.748ns

```
module ADDER8(CLK,SUM,A,B,COUT,CIN);
   input [7:0] A,B;  input CLK,CIN;  output COUT;  output [7:0] SUM;
   reg COUT;   reg [7:0] SUM;
    always @(posedge CLK)    {COUT,SUM[7:0]} <= A+B+CIN;
  endmodule
```

[ **Example** ] Pipelining adder, and the synthesis result of EP3C10: T=3.63ns，LCs=24，REG=22

```
module ADDER8(CLK,SUM,A,B,COUT,CIN);
   input [7:0] A,B;  input CLK,CIN;  output COUT;   output[7:0] SUM;
    reg TC,COUT;   reg[3:0] TS,TA,TB;    reg[7:0] SUM;
     always @(posedge CLK)  begin
        {TC,TS} <= A[3:0]+B[3:0]+CIN ;   SUM[3:0]<=TS;    end
     always @(posedge CLK)  begin
        TA <= A[7:4];   TB <= B[7:4];
       {COUT,SUM[7:4]} <= TA+TB+TC;     end
Endmodule
```