EDA Technology

Chapter 3

Behavioral Statements

Behavioral Statements

- Procedural Statement
  - *always*
  - *initial*
- Block Statemen
  - *begin ... end*
- Conditional Statement
  - *case*
  - *if*

Behavioral Statements

- Looping Statement
  - *for*
  - *while*
  - *repeat*
  - *forever*

Behavioral Statements

- Statement of Procedural Assignment
  - =
  - <=
- Procedural continuous assignments
  - *assign*
- subroutine Statements
  - *task*
  - *Function*

Procedural Statement

always Statement

```
always @ (sensitive signal and sensitive signal list or expression)
      Various behavioral statements including block statements
```

sensitive signal list

```
always @(a or b or c or d or s1 or s0).  // Verilog -1995
always @(a, b, c, d, s1, s0).                 / / Verilog-2001
always @(*)
always @*
```

Procedural Statement

- The Application of *always* Statement in D flip-flop Design

Example:

```
module DFF1(CLK,D,Q);
    output Q ;
    input  CLK, D ;
    reg Q;
  always @(posedge CLK )
    Q <= D;
Endmodule
```

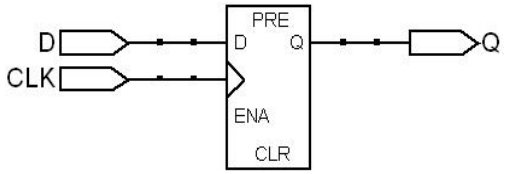


Figure: Edge-triggered D flip-flops

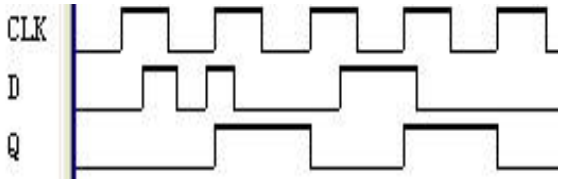


Figure: D Trigger Sequence

Procedural Statement

- The Application of Multi-Process and Asynchronous Sequential Circuit Design

Example:

```
module AMOD(D,A,CLK,Q);
  output Q; input A,D,CLK; reg Q,Q1;
  always @(posedge CLK) //process 1
    Q1 <= ~(A | Q);
  always @(posedge Q1 ) //process 2
    Q <=  D;
Endmodule
```

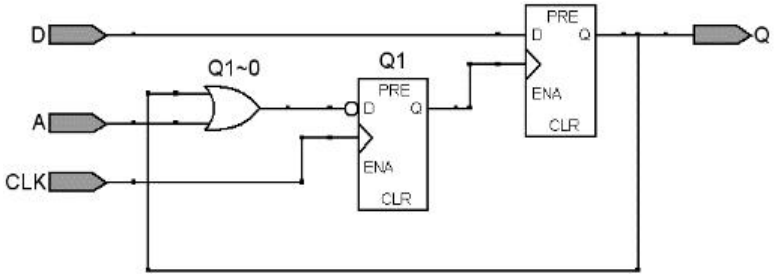


Figure: Sequential circuit of Example

Procedural Statement

Verilog Expression of Simple Addition Counter

```
module CNT4(CLK,Q);
  output [3:0] Q;  input  CLK;
  reg [3:0] Q1 ;
  always @(posedge CLK)
    Q1 = Q1+1 ;  // Attention to
the assignment symbol
  assign Q = Q1;
endmodule
```

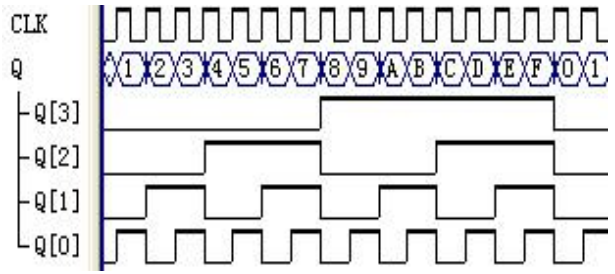
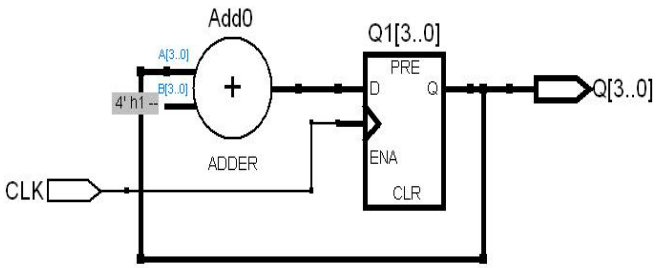


Figure: 4-bit Add Counter Operation Timing




```
module CNT4 (CLK,Q);
    output [3:0] Q;  input  CLK;
    reg [3:0] Q ;
  always @(posedge CLK)
      Q <= Q+1 ;  // Attention to
the assignment symbol
endmodule
```

Figure: 4-bit addition counter RTL circuit diagram

Procedural Statement

- Verilog Expression of Simple Addition Counter

Example:

```
module CNT4 (CLK,Q);
    output [3:0] Q;  input  CLK;
    reg [3:0] Q ;
  always @(posedge CLK)
      Q <= Q+1'b1 ;   // no warning
endmodule
```

Procedural Statement

- *initial* Statement

```
initial
  begin  statement1;  statement2; ... end
```

The following statement is synthesizable, at least it will affect the synthesis results:

```
initial  $readmemh("RAM78_DAT.dat", mem );
```

Procedural Statement

initial Statement

```
'timescale 1ns/100ps // Declare simulation time unit 1ns, accuracy 100ps
module test;                 // Define testbench test module named test
reg A, B, C;
initial      // Defining the structure of the initial procedure statement
  begin
   A=0;B=1;C=0 // Define the initial values of A, B, and C at time 0 in the process
  #50  A=1;B=0;  // After 50ns delay, the input values of A and B are 1,0 at the simulation
time of 50ns.
  #50  A=0;C=1;  // After 50ns delay, the input values of A and C are respectively 0,
1 at 100ns.
  #50  B=1;   // After 50ns delay, the input value of B is 1 at time 150ns.
  #50  B=0;C=0  // After 50ns delay, the input values of B and C are all 0 at time 200ns.
  #50  $finish // After 50ns delay, the end
  end

  'timescale simulation time unit / simulation accuracy
```

Procedural Statement

- *`timescale*

```
'timescale simulation time unit / simulation accuracy
```

```
// Declare simulation time unit 1ns,
// accuracy 100ps
'timescale 1ns/100ps
```

Block Statement

- *Begin … end*

```
Begin [: block_name]
    Statement1; Statement2; ...;
    Statementn;
End
```

Conditional Statement

case Conditional Statement

```
Case (expression)
    Value1 : begin statement1; statement2; ...; statementn; end
    Value2 : begin statementn+1; statementn+2; ...statementn+m; end
       ...
    Default : begin statementn+m+1; ...; end
Endcase
```

Conditional Statement

case Conditional Statement

Example:

```
case({s1,s0})
  2'b00 : y <= a;
  2'b01 : y <= b;
Endcase
```

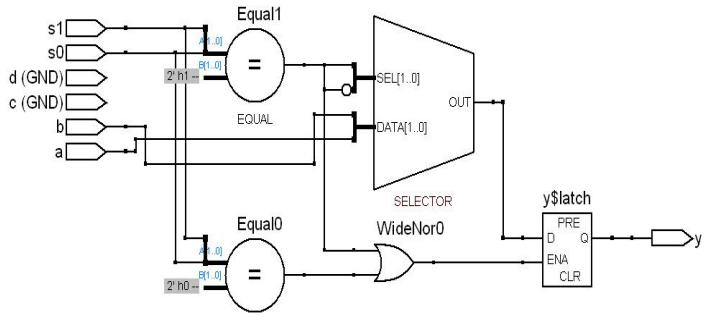


Figure: RTL diagram of Example

if Conditional Statement

General Expression of if Statement

```
(1)if (conditional expression) begin StatementBlock; end //if statement type 1

(2)if (conditional expression) begin StatementBlock1; end //if statement type 2
   else begin StatementBlock2; end

(3)if (conditional expression 1) begin StatementBlock1; end //if statement type 3
   else if (conditional expression 2) begin StatementBlock2; end
   else if (conditional expression n) begin StatementBlockn; end
   else begin StatementBlockn+1; end
```

if Conditional Statement

Combinational Circuit Design Based on if Statement

```
if (S)  Y = A;  else  Y = B;
```

```
module MUX41a (A,B,C,D,S1,S0,Y);
    input A,B,C,D,S1,S0;     output Y;
    reg [1:0] SEL ;    reg Y;
    always @(A,B,C,D,SEL)
    begin                    // Block statement start
          SEL = {S1,S0}; // S1,S0 as a 2-element vector variable SEL[1:0]
          if (SEL==0)  Y = A;    // When SEL==0, that is (SEL==0)=1, Y=A
    else  if (SEL==1)  Y = B;    // When (SEL==1) is true, then Y=B
    else  if (SEL==2)  Y = C;    // When (SEL==2) is true, then Y=C
    else               Y = D;    // When SEL==3, ie SEL==2'b11, Y=D
    end                          // Block statement ends
endmodule
```

if Conditional Statement

- Combinational Circuit Design Based on if Statement

```
module CODER83 (DIN,DOUT);
  output[0:2]DOUT;
  input[0:7]DIN;
  reg [0:2] DOUT;
  always @(DIN)
    casez (DIN)
    8'b????????0 : DOUT<=3'b000;
    8'b??????01 : DOUT<=3'b100;
    8'b?????011 : DOUT<=3'b010;
    8'b????0111 : DOUT<=3'b110;
    8'b???01111 : DOUT<=3'b001;
    8'b??011111 : DOUT<=3'b101;
    8'b?0111111 : DOUT<=3'b011;
    8'b01111111 : DOUT<=3'b111;
    default : DOUT<=3'b000;
endcase
endmodule
```

```
module CODER83 (DIN,DOUT);
  output[0:2]DOUT;
  input[0:7]DIN;
  reg [0:2] DOUT;
  always @(DIN)
      if (DIN[7]==0)  DOUT=3'b000;
else if (DIN[6]==0)  DOUT=3'b100;
else if (DIN[5]==0)  DOUT=3'b010;
else if (DIN[4]==0)  DOUT=3'b110;
else if (DIN[3]==0)  DOUT=3'b001;
else if (DIN[2]==0)  DOUT=3'b101;
else if (DIN[1]==0)  DOUT=3'b011;
else                 DOUT=3'b111;
endmodule
```

if Conditional Statement

- Combinational Circuit Design Based on if Statement

```
(DIN[7]==1) & (DIN[6]==1) & (DIN[5]==1) & (DIN[4]==1)
& (DIN[3]==1) & (DIN[2]==1) & (DIN[1]==1) &
(DIN[0]==0)
//This coincides with the last row of Table 3-1
```

if Conditional Statement

Combinational Circuit Design Based on if Statement

Figure: Timing simulation waveforms

Table: 8-line-3 line priority encoder truth table

input								output		
din0	din1	din2	din3	din4	din5	din6	din7	output0	output1	output2
x	x	x	x	x	x	x	0	0	0	0
x	x	x	x	x	x	0	1	1	0	0
x	x	x	x	x	0	1	1	0	1	0
x	x	x	x	0	1	1	1	1	1	0
x	x	x	0	1	1	1	1	0	0	1
x	x	0	1	1	1	1	1	1	0	1
x	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	1	1

if Conditional Statement

Sequential Circuit Design Based on if Statement

Example:

```
module LATCH1(CLK,D,Q);
       output Q ; input CLK,D;
       reg Q;
     always @(D or CLK)
       if(CLK)  Q <= D;
endmodule
```

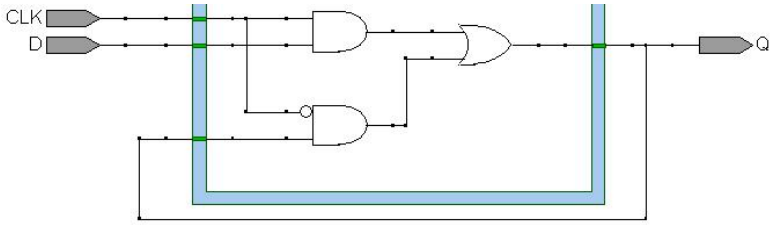


Figure: Latch Module Internal Logic Circuit

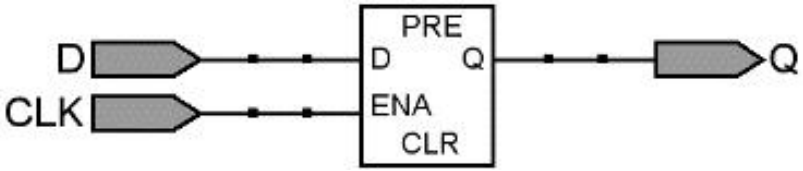


Figure: Latch Module

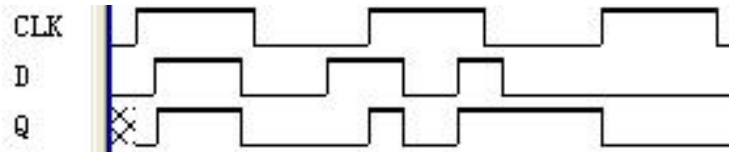


Figure: Timing waveform of the latch in Example

if Conditional Statement

- Design of DFF with Asynchronous Reset and Clock Enable

Example:

```
module DFF2(CLK,D,Q,RST,EN);
  output Q;
input CLK,D,RST,EN;
  reg Q;
always @(posedge CLK or negedge RST)
  begin
    if (!RST)  Q <= 0;
  else if (EN)     Q <= D;
  end
endmodule
```

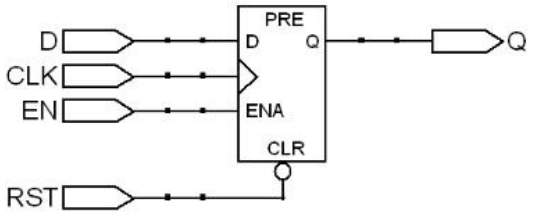


Figure: D Trigger with Enable and Reset Control

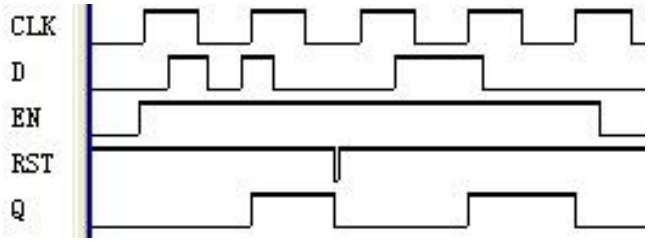


Figure: Sequence of Example

if Conditional Statement

Design of DFF with Synchronous Reset

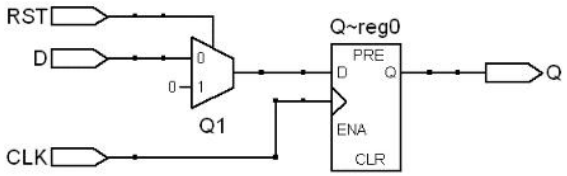


Figure: D flip-flop with Synchronous Clear Control

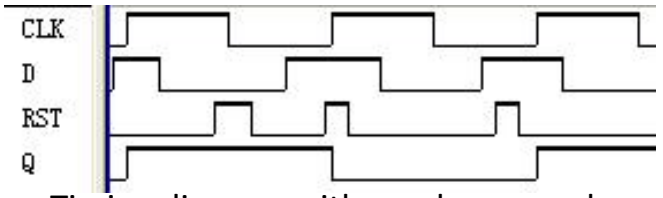


Figure: Timing diagram with synchronous clear control D flip-flop

Example:

```
module DFF3(CLK,D,Q,RST);
     output Q ;
     input CLK,D,RST ;
     reg Q;
     always @(posedge CLK )
          if (RST==1)  Q = 0;
    else if (RST==0)  Q = D;
            else        Q = Q;
endmodule
```

if Conditional Statement

- Design of DFF with Synchronous Reset

```
module DFF1(CLK,D,Q,RST);
    output Q;  input CLK,D,RST;
    reg Q, Q1; //Note the Q1 signal definition
    always @(RST,D) // Pure combination process
     if (RST==1)  Q1=0;
           else  Q1=D;
    always @(posedge CLK )
         Q <= Q1;
endmodule
```

if Conditional Statement

Design of Latches with Clear

Example:

```
module LATCH2 (CLK,D,Q,RST);
   output Q ; input CLK,D,RST;
   assign Q = (!RST)? 0:(CLK ? D:Q);
endmodule
```

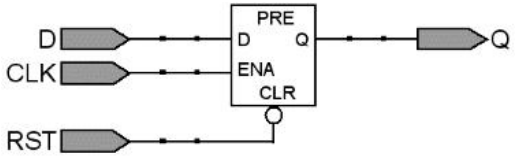


Figure: A latch with asynchronous clear

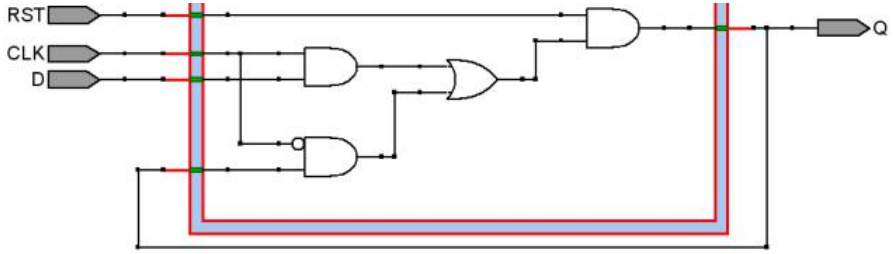


Figure: Logic Circuit Diagram with Asynchronously Cleared Latches

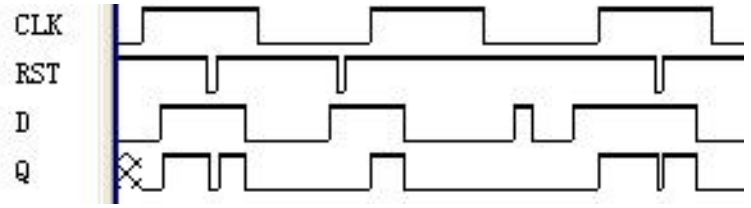


Figure: Simulated Waveforms with Asynchronously Cleared Latches

if Conditional Statement

Design of Latches with Clear

```
module LATCH3 (CLK,D,Q,RST);
    output Q ;
    input CLK,D,RST;
   reg Q;
    always @(D or CLK or RST)
            if(!RST)   Q<=0;
       else
            if(CLK)    Q<=D;
endmodule
```

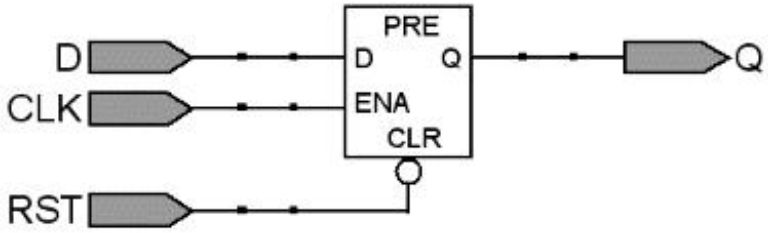


Figure: A latch with asynchronous clear

if Conditional Statement

Characteristics and Rules of Clock Procedural Statement

Mixed signals are not allowed in the sensitive signal table. Once the sensitive signal contains the edge sensitive signal of posedge or negedge (i.e. a single edge), all other ordinary variables cannot be placed in the sensitive signal table. The so-called mixed signal representation is not allowed, as the following form is wrong:

```
always @(posedge CLK or RST )
Or
always @(posedge CLK or negedge RST or A)
```

if Conditional Statement

- Characteristics and Rules of Clock Procedural Statement

```
always @ (posedge CLK or negedge RST )
    begin if (!RST) ...

always @ (posedge CLK or negedge RST )
    begin if (RST== 0) ...

always @ (posedge CLK or negedge RST )
    begin if (!RST==1) ...
```

if Conditional Statement

- Characteristics and Rules of Clock Procedural Statement

Example:

```
module DFF5(CLK,D,Q,RST,DIN,OUT);
  output Q,OUT;
  input CLK,D,RST,DIN; reg Q,OUT;
  always @(posedge CLK )
    begin   OUT = !DIN;
      if (RST==1)  Q=0;
        else if(RST==0)  Q=D;
    end
endmodule
```

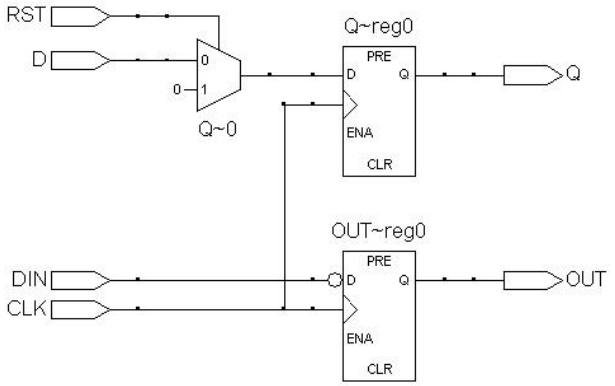


Figure: RTL diagram of the Example

if Conditional Statement

- Characteristics and Rules of Clock Procedural Statement

Example:

```
module DFF5(CLK,D,Q,RST,DIN,OUT);
  output Q,OUT;
  input CLK,D,RST,DIN; reg Q,OUT;
  always @(posedge CLK )
    begin   OUT = !DIN;
      if (RST==1)  Q=0;
        else if(RST==0)  Q=D;
    end
endmodule
```

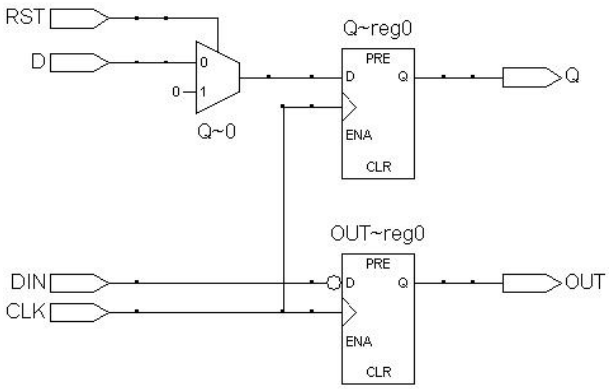


Figure: RTL diagram of the Example

if Conditional Statement

⌘ Practical Addition Counter Design

Example:

```
module CNT10 (CLK,RST,EN,LOAD,COUT,DOUT,DATA);
    input CLK,EN,RST,LOAD ;  // Clock, Clock Enable, Reset, Data Load Control Signals
    input [3:0] DATA ;          //4-bit parallel loading data
    output [3:0] DOUT ;         //4-bit count output
    output COUT ;               // Counting carry output
    reg [3:0] Q1 ;     reg COUT ;
    assign DOUT = Q1; // Output the internal register count result to DOUT
    always @(posedge CLK or negedge RST)    // Timing process
      begin
      if (!RST)  Q1 <= 0;//Asynchronously clear the internal register unit when RST=0
      else  if (EN)  begin // Synchronization enable EN=1, allows loading or counting
      if (!LOAD)   Q1<=DATA; //When LOAD=0, data is loaded into internal registers
      else if (Q1<9)  Q1 <= Q1+1;//Accumulation allowed when Q1 is less than 9
      else Q1 <= 4'b0000; end // Otherwise clear and return to initial value after one
clock
      end
    always @(Q1)              // Combination process
      if (Q1==4'h9)  COUT = 1'b1;  else   COUT = 1'b0;
endmodule
```

if Conditional Statement

- Practical Addition Counter Design

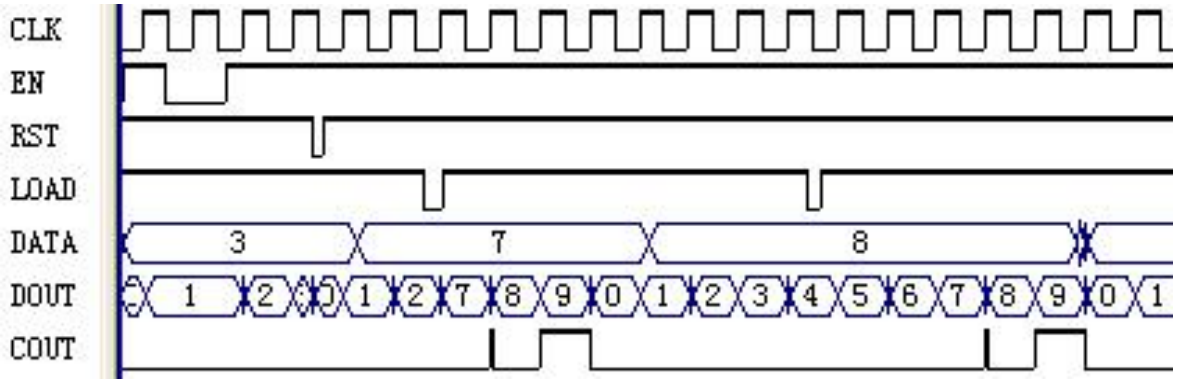


Figure: Simulation waveforms of the Example

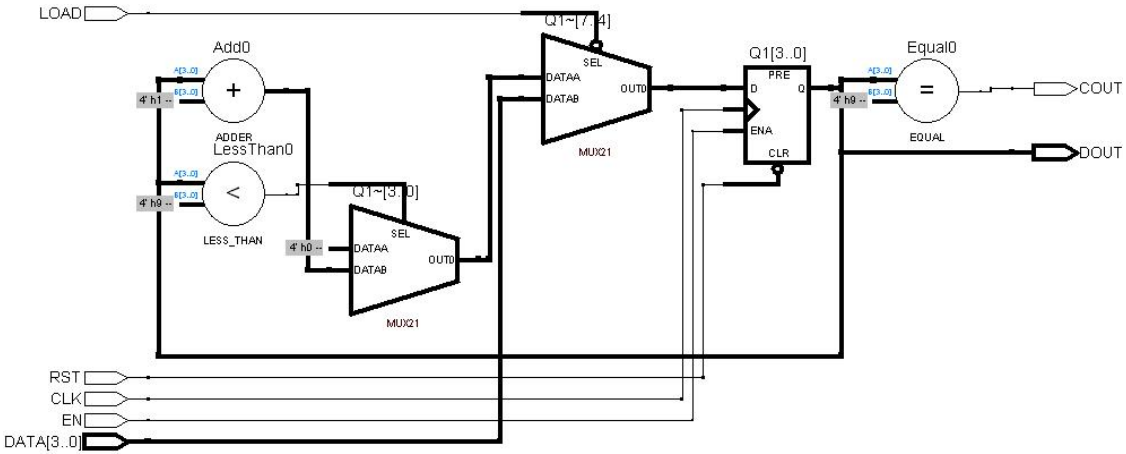


Figure: RTL circuit diagram synthesized and obtained by Quartus II for the Example

if Conditional Statement

- Shift Register Design with Synchronized Preset Function

Example:

```
module SHFT1(CLK,LOAD,DIN,QB);
   output QB;  input CLK,LOAD;
   input[7:0] DIN; reg[7:0]  REG8;
   always @(posedge CLK )
     if (LOAD)      REG8<=DIN ;
     else REG8[6:0]<=REG8[7:1];
   assign QB = REG8[0] ;
endmodule
```

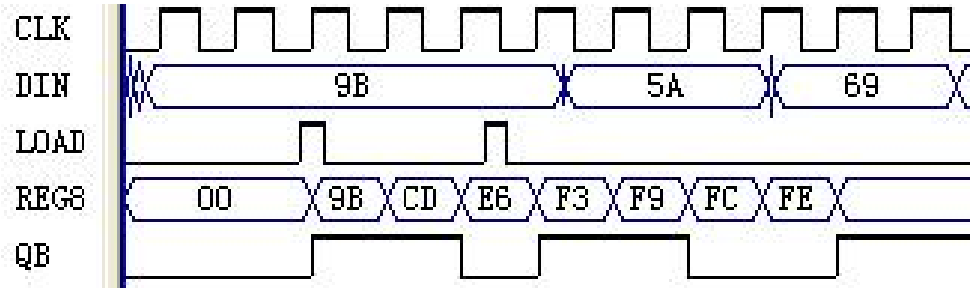


Figure: Working timing diagram of the Example

if Conditional Statement

- Conditional Instructions in if Statements

```
module andd(A,B,Q);
  output Q ;
  input A,B;
   reg Q;
  always @(A,B )
   if (A==0)
   if (B==0)  Q=0;
  else   Q=1;
endmodule
```

```
module andd(A,B,Q);
  output Q ;
  input A,B; reg Q;
  always @(A,B )
   if (A==0)
   begin
   if (B==0)  Q=0;
  else  Q=1; end
endmodule
```

if Conditional Statement

- Conditional Instructions in if Statements

```
module andd(A,B,Q);
  output Q; input A,B;
  reg Q;
  always @(A,B )
   if (A==0)
  begin if(B==0) Q=0;
  end
  else  Q=1;
endmodule
```

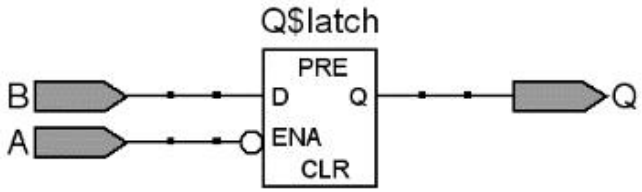


Figure 1

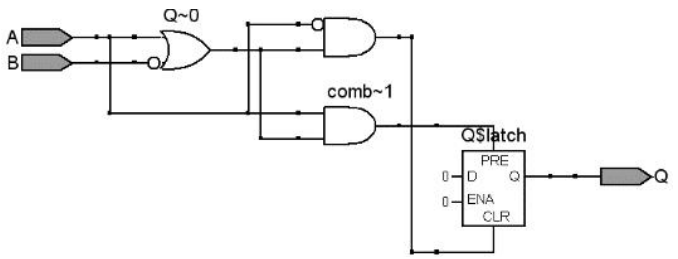


Figure 2

Statement of Procedural Assignment

- = or <=
- Blocking assignments.
  - x = a;
  - y = b;
- Non-blocking assignments.
  - y <= a;
  - y <= b;

Loop Statement

for Statement

```
for (initial value of loop setting expression; loop
condition control expression; loop control variable
increment expression)
   begin
loop body statement structure
end
```

Loop Statement

for Statement

```
module MULT4B(R,A,B);
   parameter S=4;
   output[2*S:1] R ;
   input[S:1] A,B ;
   reg[2*S:1] R ;
   integer i;
     always @(A or B)
     begin
     R = 0 ;
      for(i=1;  i<=S; i=i+1)
      if(B[i])
     R=R+(A<<(i-1)); //shift (i-1) bit
left
      end
 endmodule
```

```
module MULT4B (R,A,B);
   parameter S=4;
   output[2*S:1] R ;
   input[S:1] A,B ;
   reg[2*S:1] R,AT; reg[S:1] BT,CT;
   always @(A,B )   begin
    R=0; AT = {{S{1'B0}},A};
    BT = B;  CT = S;
    for(CT=S; CT>0; CT=CT-1)
      begin  if(BT[1]) R=R+AT;
        AT = AT<<1;   //shift 1-bit left
        BT = BT>>1;   // shift 1-bit
right
 end  end
 endmodule
```

Loop Statement

- *for* Statement

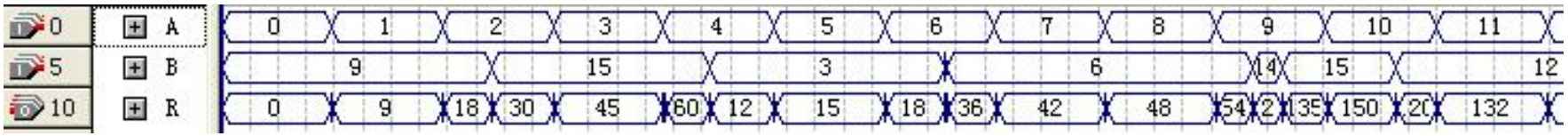


Figure: Timing simulation of a 4-bit multiplier

Loop Statement

while Statement

```
while (loop control conditional expression)
begin
loop body statement structure
end
```

Loop Statement

while Statement

```
module MULT4B(A,B,R);
  parameter S=4;
  input[S:1] A,B;
  output[2*S:1] R;
  reg[2*S:1] R,AT;
  reg[S:1] BT,CT;
  always@(A or B) begin
    R=0; AT={{S{1'b0}},A};
     BT=B; CT=S;
     while(CT>0) begin
      if(BT[1]) R=R+AT; else R=R;
begin  CT= CT-1; AT=AT<<1; BT=BT>>1;
end    end    end
endmodule
```

```
module MULT4B(R,A,B);
parameter S=4;
output [2*S:1] R; input [S:1] A,B ;
  reg [2*S:1] TA,R;
reg [S:1] TB;
always @(A or B)  begin
  R = 0 ; TA = A ;  TB = B ;
  repeat(S) begin
  if(TB[1]) begin R=R+TA; end
    TA = TA<<1;  //left shift 1 bit
    TB = TB>>1;  //right shift 1 bit
   end
  end
endmodule
```

Loop Statement

- *repeat* Statement

```
repeat (loop number expression)
   begin
loop body statement structure
end
```

Loop Statement

- *forever* Statement

```
forever statement;
or
forever begin statement; end
```

task and Function Statements

task

```
task <task name>;
     Port and data type declaration statement
     Begin procedure statement; end
   Endtask


<task name> (Port 1, Port 2, ..., Port N);
```

task and Function Statements

task

Example

```
module TASKDEMO (S,D,C1,D1,C2,D2);   // Main program module and port definition
input S;   input[3:0] C1,D1,C2,D2;
output [3:0] D;                 // Unlimited number of port definitions
reg [3:0] out1,out2;
task CMP;                      // Task definition, task name CMP, port definition statement
                               //cannot appear in this line
input [3:0] A,B;
output [3:0] DOUT;             // Note the sequence of task port names
begin if (A>B) DOUT= A;        // Mission process statement describes a comparison circuit
else DOUT=B; end               // Other tasks or functions can be invoked in the task
                               //structure, or even itself
endtask                        // End of task definition
always @ (*) begin             // The main program process begins
CMP(C1,D1,out1);               // Invoke the task once. Task invoking statement can only
                               //appear in the process structure
CMP(C2,D2,out2);   end         // The second invoke to the task
assign D=S? out1:out2;
endmodule
```

task and Function Statements

function

```
Function <bitwidth range declaration>
function name;
Input port description, other types of
variable definitions;
Begin procedure statement; end
Endfunction
```

```
<function name> (input parameter 1, input parameter 2, ...)
```