

8. Design at the Register Transfer Level

8.2 Register Transfer Level(RTL) Notation

- Modules of digital devices : registers, decoders, multiplexers, arithmetic elements, control logic
- Register operations : shift, count, clear, load
- RTL (Register Transfer Level)
 - The set of registers in the system.
 - The operations that are performed on the data stored in the registers.
 - The control that supervises the sequence of operations in the system.

$R2 \leftarrow R1$

If (T3=1) then ($R2 \leftarrow R1$, $R1 \leftarrow R2$)

$R1 \leftarrow R1 + R2$ Add content of R2 to R1

$R3 \leftarrow R3 + 1$ Increment R3 by 1 (count up)

$R4 \leftarrow \text{shr } R4$ Shift right R4

$R5 \leftarrow 0$ Clear R5 to 0

8.3 Register Transfer Level in HDL


assign $S = A + B;$	Continuous assignment
always @ (A or B) $S = A + B;$	Procedural assignment (without a clock)
always @ (posedge clock) begin $RA = RA + RB;$ $RD = RA;$ end	Blocking procedural assignment
always @ (negedge clock) begin $RA <= RA + RB;$ $RD <= RA;$ end	Non-blocking procedural assignment

8.3 RTL in HDL – HDL Operators

Table 8-1
Verilog HDL Operators

Operator Type	Symbol	Operation Performed
Arithmetic	+	addition
	-	subtraction
	*	multiplication
	/	division
	%	modulus
Logic (bit-wise or reduction)	~	negation (complement)
	&	AND
		OR
Logical	^	Exclusive-OR (XOR)
	!	negation
	&& 	AND OR
Shift	>>	shift right
	<<	shift left
	{ , }	concatenation
Relational	>	greater than
	<	less than
	==	equality
	!=	inequality
	>=	greater than or equal
	<=	less than or equal

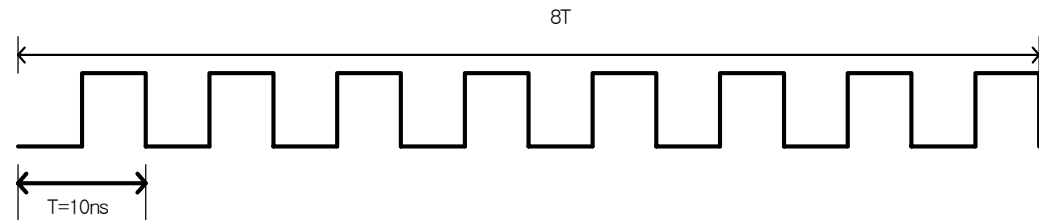
Table 8.2
Verilog Operator Precedence

+ - ! ~ & ~ & ~ ^ ~ ^ ^ ~ (unary)	Highest precedence
**	
*/%	
+- (binary)	
<< >> <<< >>>	
< <= > >=	
== != === !==	
& (binary)	
^ ^~ ~^ (binary)	
(binary)	
&&	
?: (conditional operator)	
{ } { }	Lowest precedence

8.3 RTL in HDL – Loop Statement

repeat

```
integer count;  
initial  
begin  
    count = 0;  
    while (count < 64)  
        count = count + 1;  
end
```



forever

```
initial  
begin  
    clock = 1'b0;  
    forever  
        # clock = ~ clock;  
end
```

while

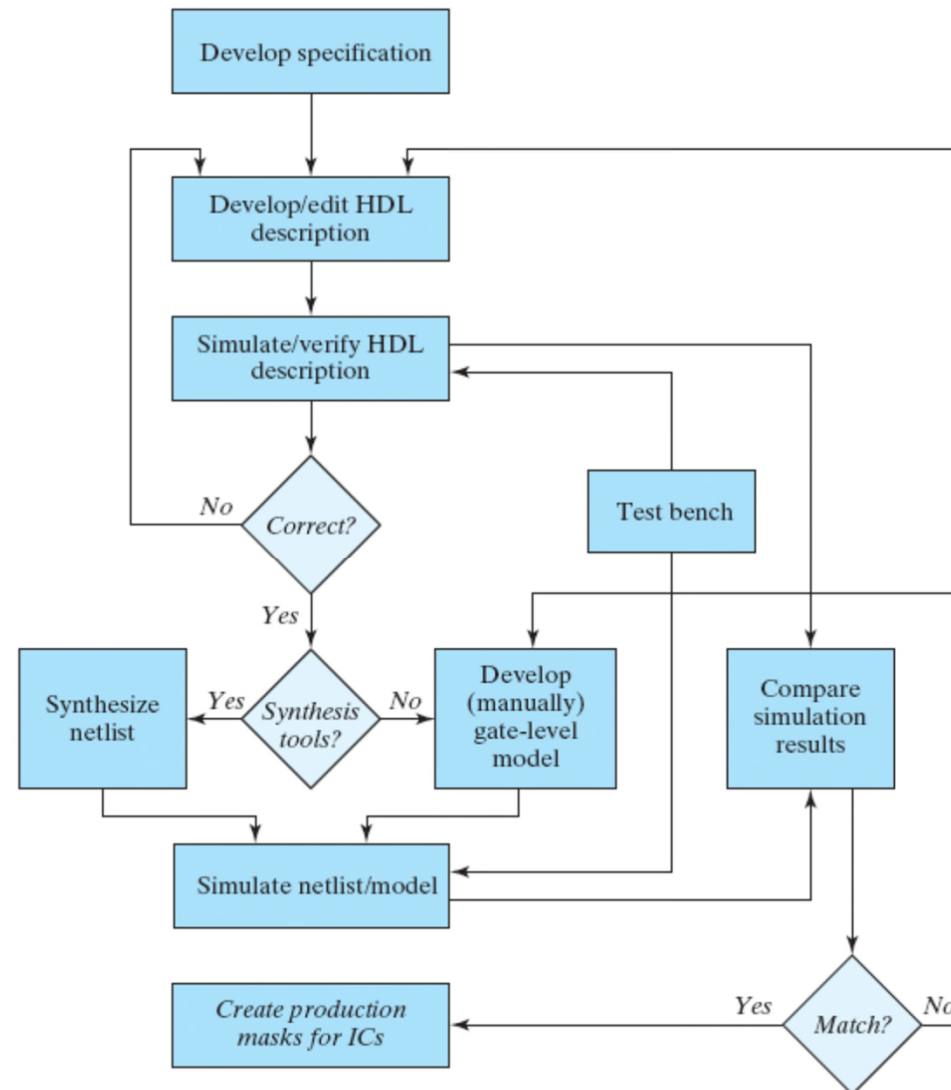
```
initial  
begin  
    clock = 1'b0;  
    repeat (16)  
        #5 clock = ~ clock;  
end
```

8.3 RTL in HDL – Loop Statement

- for

```
//HDL Example 8-1
//-----
//description of 2x4 decoder
//using for-loop statement
module decoder (IN, Y);
    input [1:0] IN;    //Two binary inputs
    output [3:0] Y;    //Four binary outputs
    reg [3:0] Y;
    integer I;        //control variable for loop
    always @ (IN)
        for (I = 0; I <= 3; I = I + 1)
            if (IN == I) Y[I] = 1;
            else Y[I] = 0;
endmodule
```

8.3 RTL in HDL – Logic Synthesis



8.4 ASM (Algorithmic State Machines)

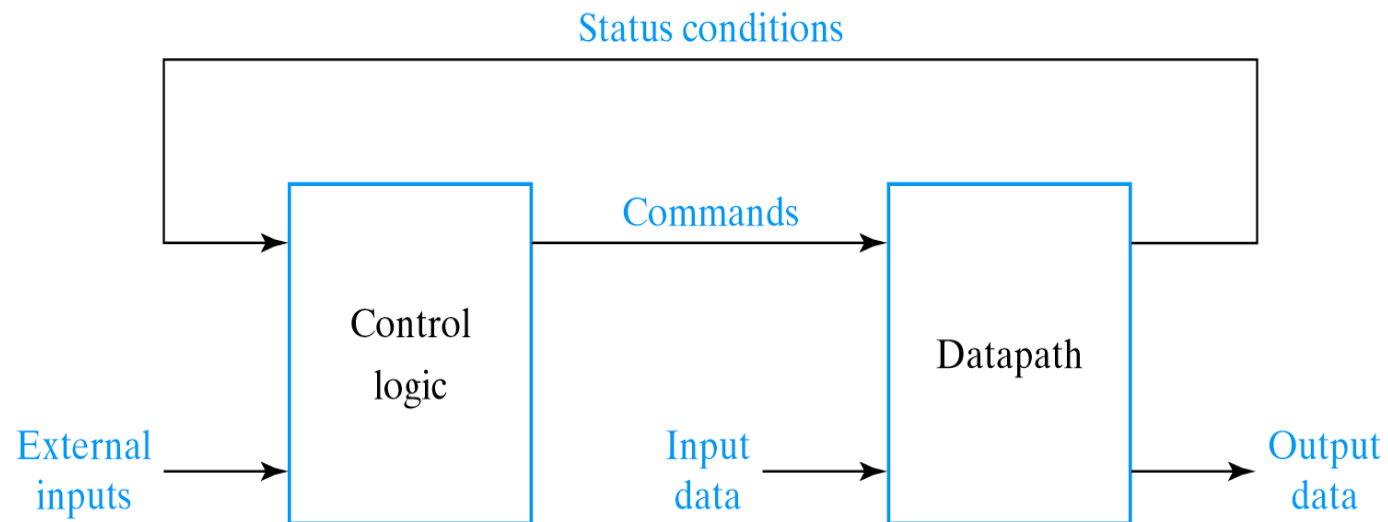


Fig. 8-2 Control and Datapath Interaction

8.4 ASM - Chart

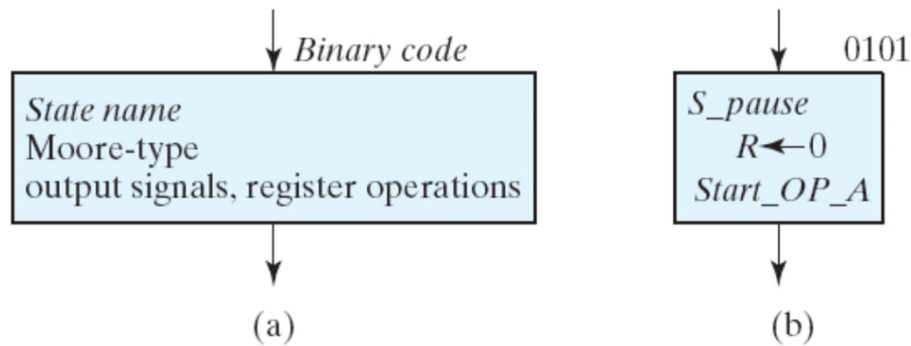


Fig. 8-3 State Box

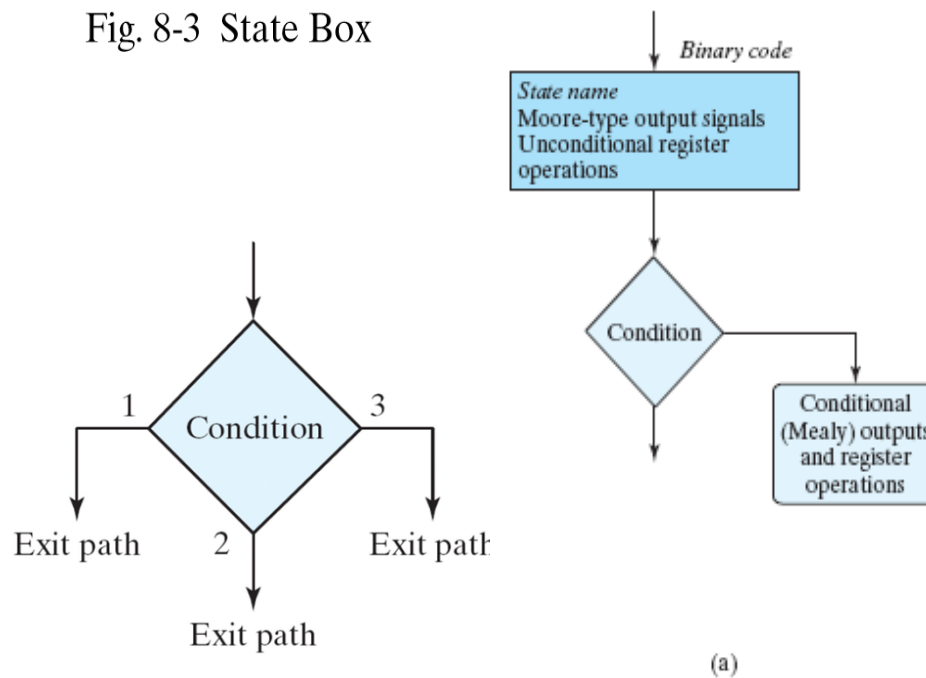


Fig. 8-4 Decision Box

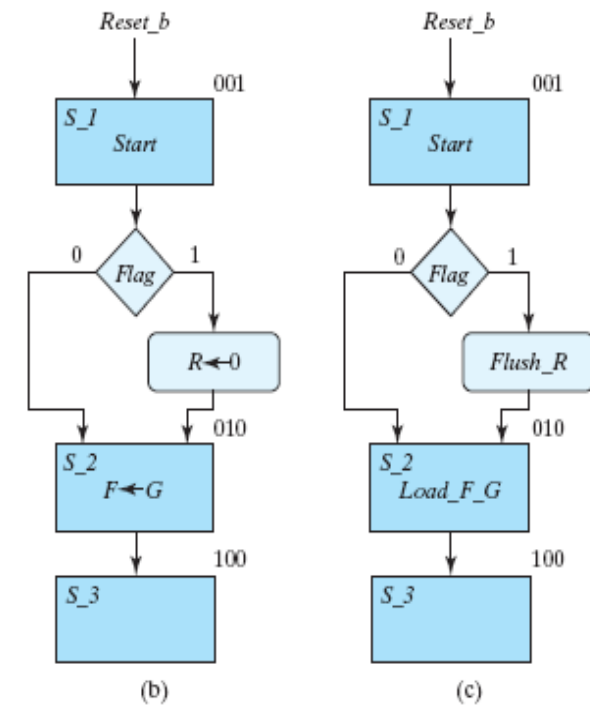


Fig. 8-5 Conditional Box

8.4 ASM – Block & Timing Considerations

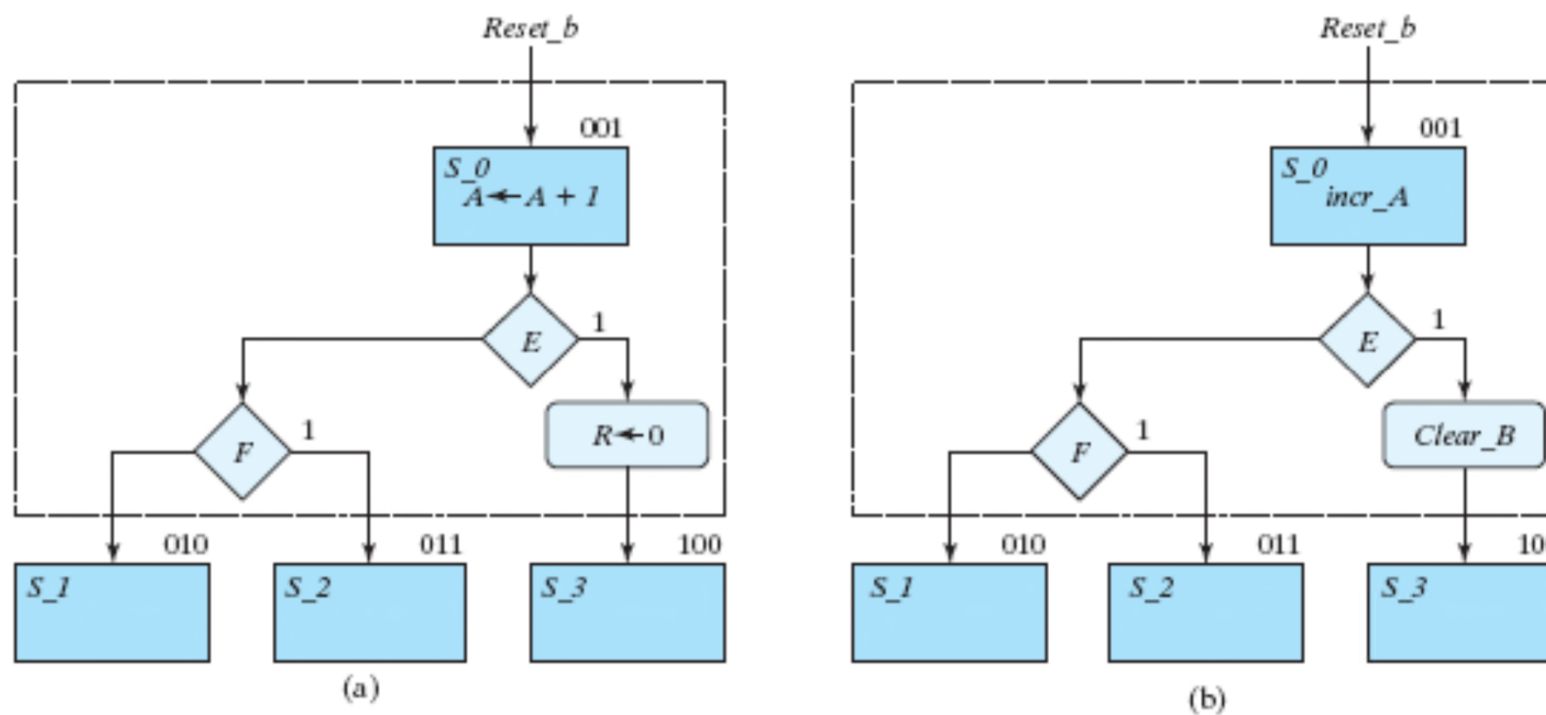


Fig. 8-6 ASM Block

8.4 ASM – Block & Timing Considerations

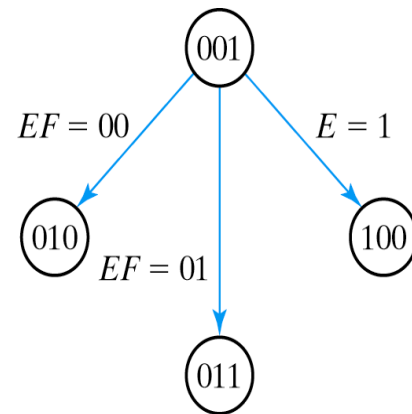


Fig. 8-7 State Diagram Equivalent to the ASM Chart of Fig. 8-6

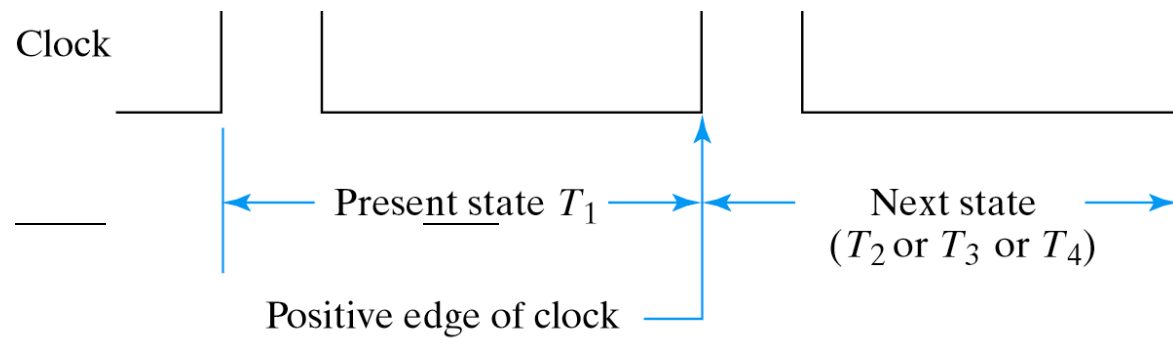
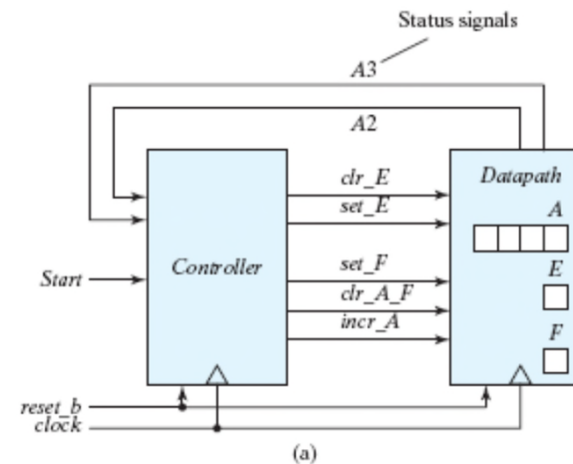


Fig. 8-8 Transition Between States

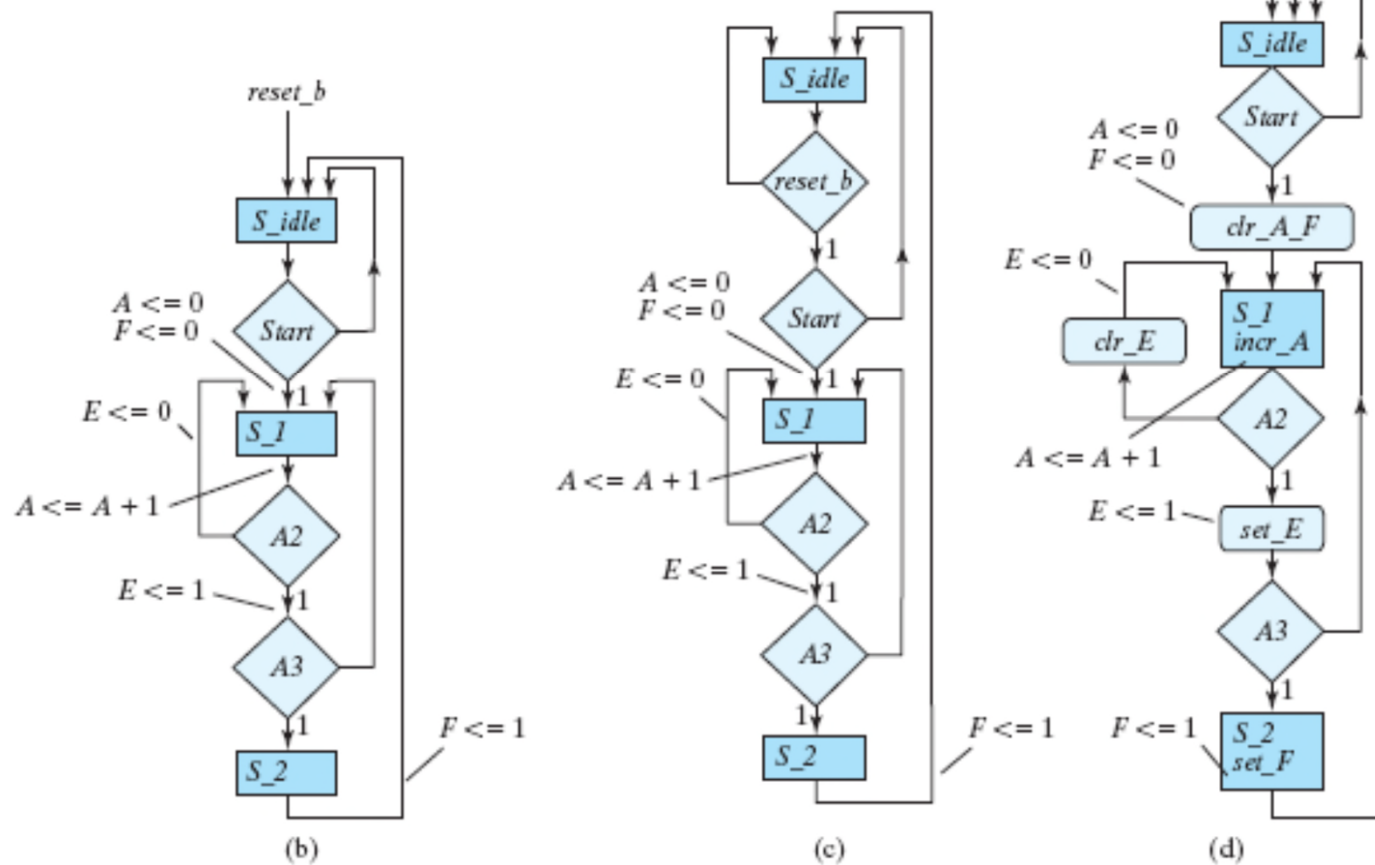
8.5 Design Example – ASMD Chart

- E, F : flip-flop
 - A : 4-bit binary counter (A_4, A_3, A_2, A_1)
-
- If $A_3 = 0$, E is cleared to 0 and the count continues.
 - If $A_3 = 1$, E is set to 1; then if $A_4 = 0$, the count continues, but if $A_4 = 1$, F is set to 1 on the next clock pulse and the system stops counting.
 - Then if $S = 0$, the system remains in the initial state, but if $S = 1$, the operation cycle repeats.



8.5 Design Example – ASMD Chart

Note: A3 denotes A[3],
 A2 denotes A[2],
 \leq denotes nonblocking assignment
 reset_b denotes active-low reset condition

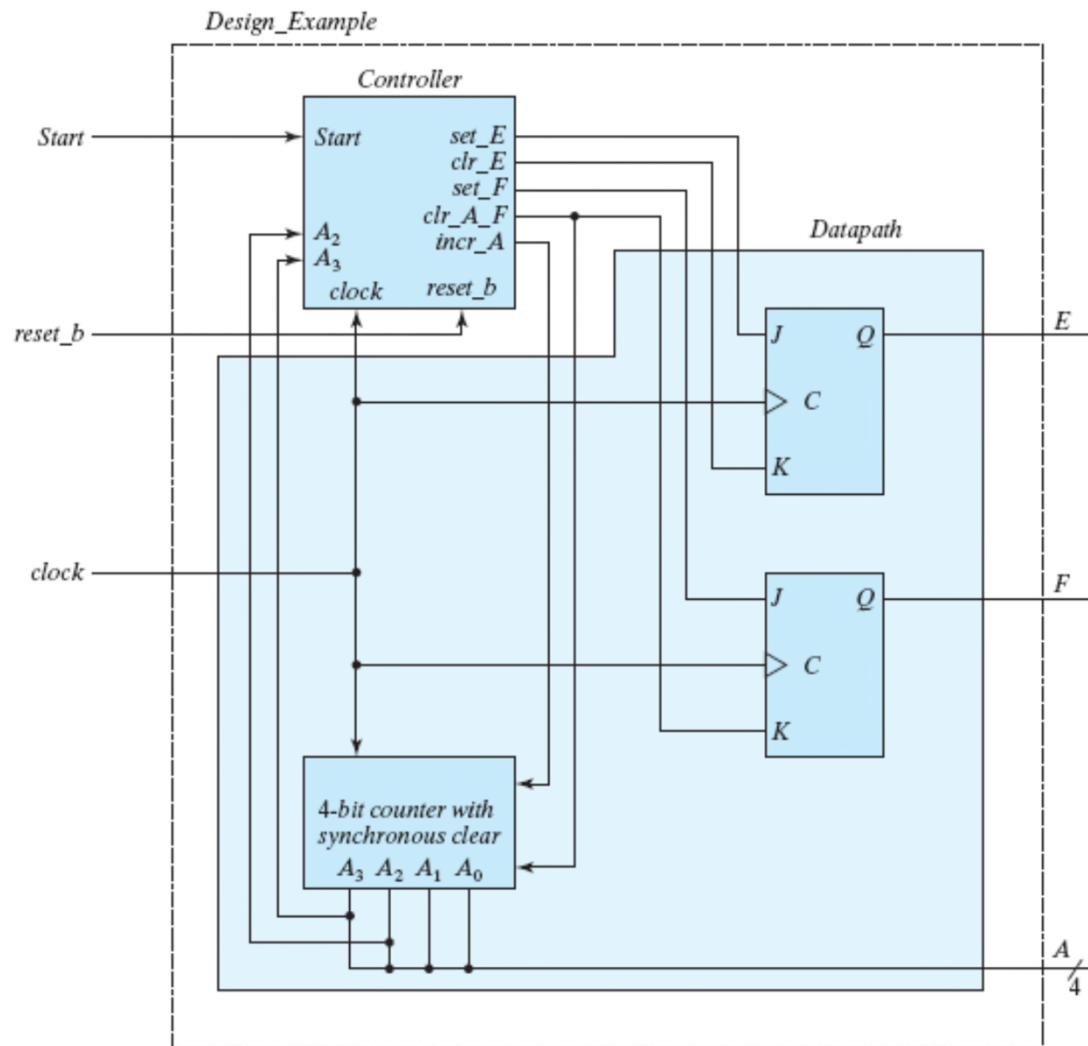


8.5 Design Example – Timing Sequence

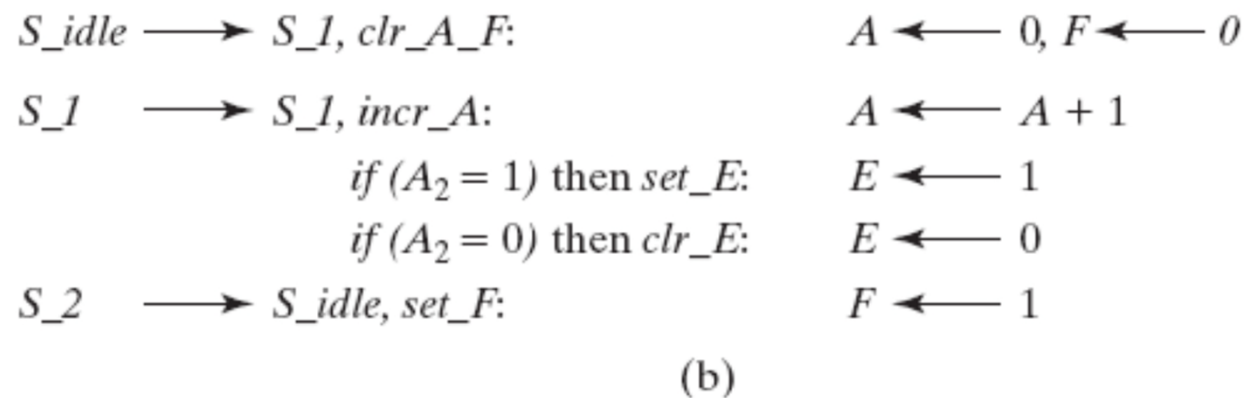
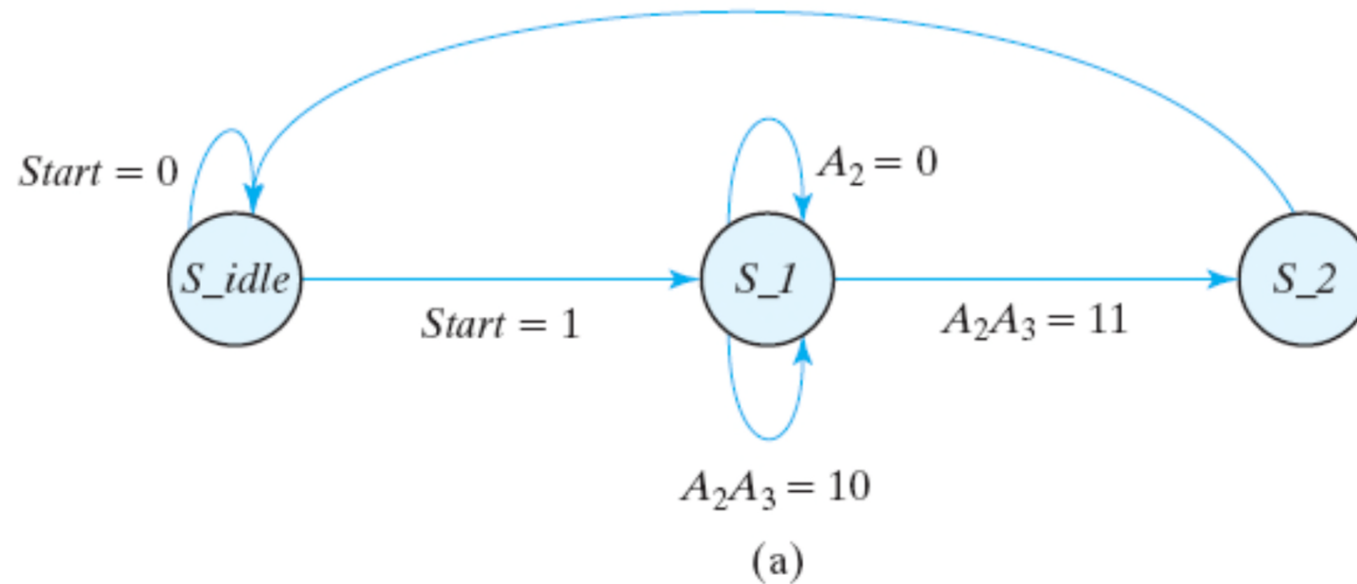
Table 8.3
Sequence of Operations for Design Example

Counter				Flip-Flops		Conditions	State
A_3	A_2	A_1	A_0	E	F		
0	0	0	0	1	0	$A_2 = 0, A_3 = 0$	S_1
0	0	0	1	0	0		
0	0	1	0	0	0		
0	0	1	1	0	0		
0	1	0	0	0	0	$A_2 = 1, A_3 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_2 = 0, A_3 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_2 = 1, A_3 = 1$	S_2
1	1	0	1	1	0		
1	1	0	1	1	1		S_{idle}

8.5 Design Example – Datapath Design



8.5 Design Example – Register Transfer Representation & State Table

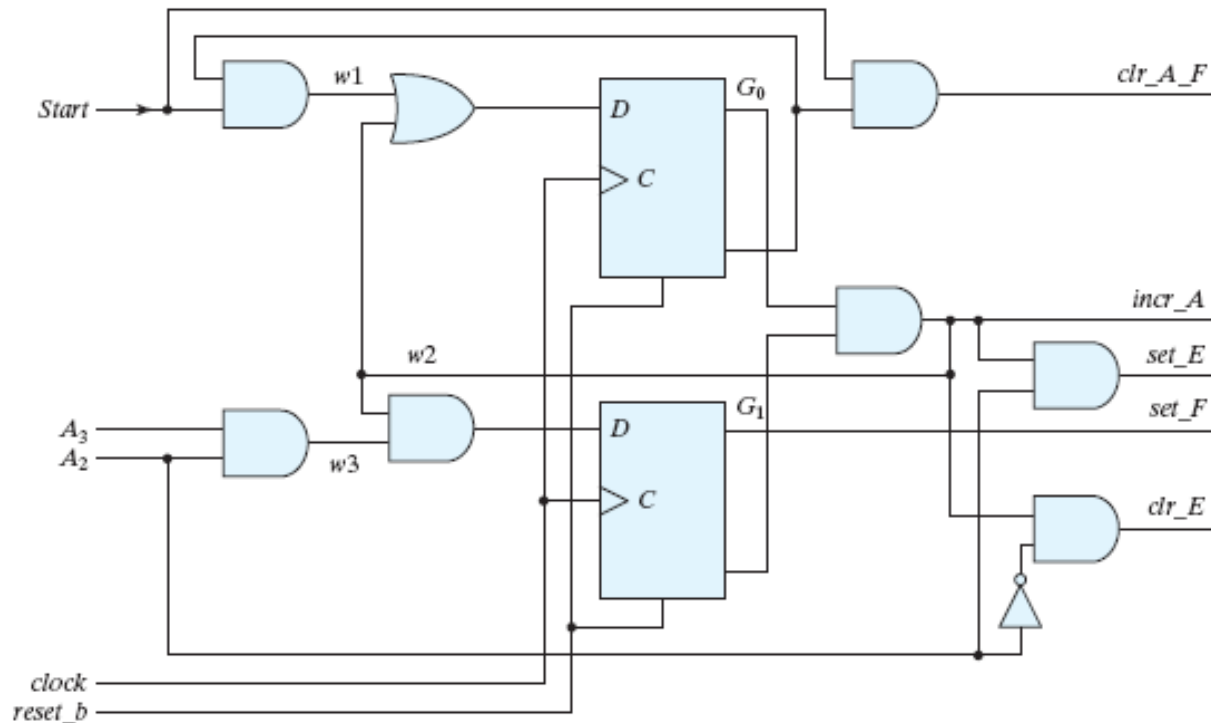


8.5 Design Example – Register Transfer Representation & State Table

Table 8.4
State Table for the Controller of Fig. 8.10

Present-State Symbol	Present State		Inputs			Next State		Outputs				
	G ₁	G ₀	Start	A ₂	A ₃	G ₁	G ₀	set_E	clr_E	set_F	clr_A_F	incr_A
<i>S_idle</i>	0	0	0	X	X	0	0	0	0	0	0	0
<i>S_idle</i>	0	0	1	X	X	0	1	0	0	0	1	0
<i>S_1</i>	0	1	X	0	X	0	1	0	1	0	0	1
<i>S_1</i>	0	1	X	1	0	0	1	1	0	0	0	1
<i>S_1</i>	0	1	X	1	1	1	1	1	0	0	0	1
<i>S_2</i>	1	1	X	X	X	0	0	0	0	1	0	0

8.5 Design Example – Control Logic



$$D_{G1} = S_1 A_2 A_3$$

$$D_{G0} = Start \ S_idle + S_1$$

$$set_E = S_1 A_2$$

$$clr_E = S_1 A_2'$$

$$set_F = S_2$$

$$clr_a_F = Start \ S_idle$$

$$incr_A = S_1$$

8.6 HDL Description of Design Example

- The structure description : gates, flip-flops, standard circuits
 - The RTL description : H/W configuration among the registers
 - The algorithmic-based behavioral description : the function of the design in a procedural algorithmic form similar to a programming language
-
- The RTL description
 1. The inputs, outputs, and registers in the design.
 2. The control sequence.
 3. The register transfer operations and outputs.

8.6 HDL Description of Design Example - RTL Description

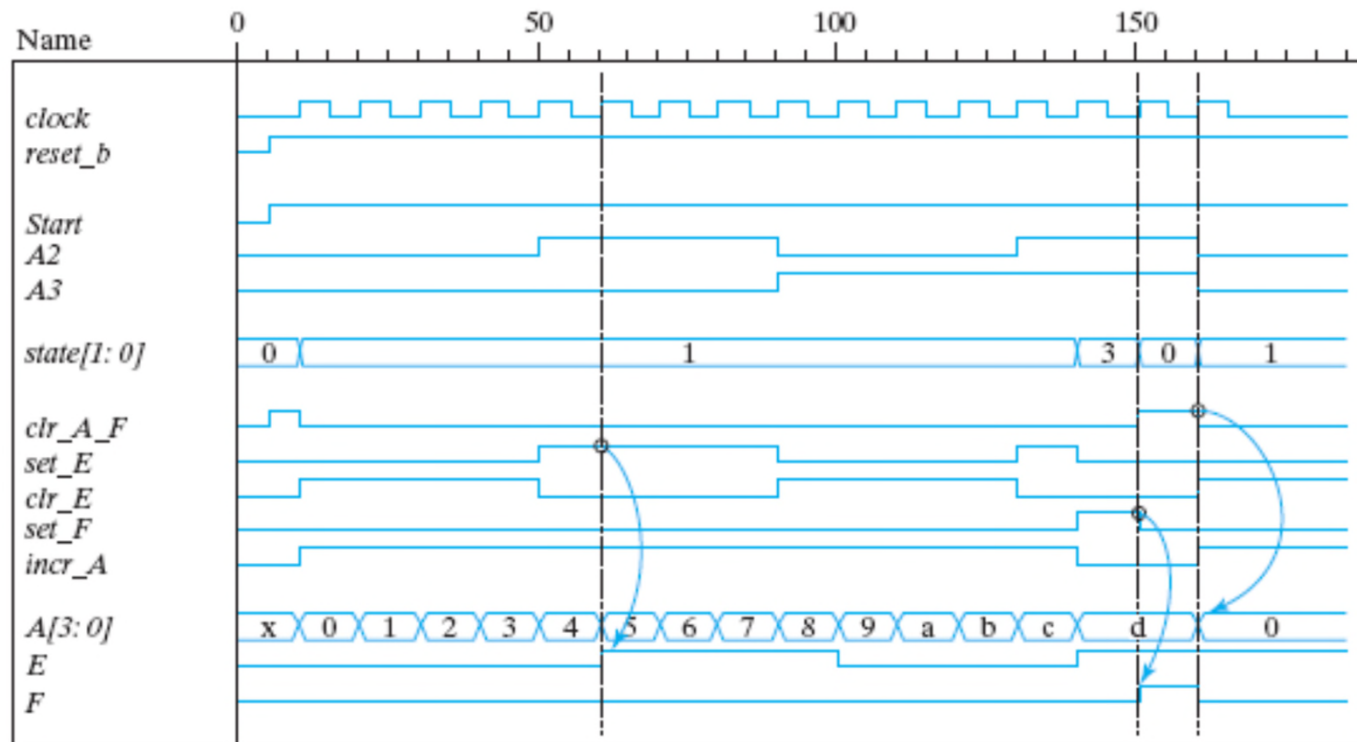
```
module Example_RTL (S,CLK,Clr,E,F,A);
//Specify inputs and outputs
//See block diagram Fig. 8-10
  input S,CLK,Clr;
  output E,F;
  output [4:1] A;
//Specify system registers
  reg [4:1] A;           //A register
  reg E, F;             //E and F flip-flops
  reg [1:0] pstate, nstate; //control register
//Encode the states
  parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b11;
//State transition for control logic
//See state diagram Fig. 8-11(a)
  always @(posedge CLK or negedge Clr)
    if (~Clr) pstate = T0; //Initial state
    else pstate <= nstate; //Clocked operations
  always @ (S or A or pstate)
    case (pstate)
      T0: if(S) nstate = T1;
      T1: if(A[3] & A[4]) nstate = T2;
      T2: nstate = T0;
      default: nstate = T0;
    endcase
```

```
//Register transfer operations
//See list of operations Fig.8-11(b)
  always @(posedge CLK)
    case (pstate)
      T0: if(S)
        begin
          A <= 4'b0000;
          F <= 1'b0;
        end
      T1:
        begin
          A <= A + 1'b1;
          if (A[3]) E <= 1'b1;
          else E <= 1'b0;
        end
      T2: F <= 1'b1;
    endcase
endmodule
```

8.6 HDL Description of Design Example - Testing the Design Description

```
//HDL Example 8-3
//-----
//Test bench for design example
module test_design_example;
  reg S, CLK, Clr;
  wire [4:1] A;
  wire E, F;
  //Instantiate design example
  Example_RTL dsexp (S,CLK,Clr,E,F,A);
  initial
  begin
    Clr = 0;
    S = 0;
    CLK = 0;
    #5 Clr = 1; S = 1;
    repeat (32)
    begin
      #5 CLK = ~ CLK;
    end
  end
  initial
  $monitor("A = %b E = %b F = %b time = %0d", A,E,F,$time);
endmodule
```

8.6 HDL Description of Design Example - Testing the Design Description



8.6 HDL Description of Design Example - Structure Description

● Structure Description

1. The control block.
2. The E and F flip-flops and associated gates.
3. The counter with synchronous clear.

```
//HDL Example 8-4
//-----
//Structural description of design example
//See block diagram Fig. 8-10
module Example_Structure (S,CLK,Clr,E,F,A);
    input S,CLK,Clr;
    output E,F;
    output [4:1] A;
    //Instantiate control circuit
    control ctl (S,A[3],A[4],CLK,Clr,T2,T1,Clear);
    //Instantiate E and F flip-flops
    E_F EF (T1,T2,Clear,CLK,A[3],E,F);
    //Instantiate counter
    counter ctr (T1,Clear,CLK,A);
endmodule
```

```
//Control circuit (Fig. 8-12)
module control
(Start,A3,A4,CLK,Clr,T2,T1,Clear);
    input Start,A3,A4,CLK,Clr;
    output T2,T1,Clear;
    wire G1,G0,DG1,DG0;
    //Combinational circuit
    assign DG1 = A3 & A4 & T1,
           DG0 = (Start & ~G0) | T1,
           T2 = G1,
           T1 = G0 & ~G1,
           Clear = Start & ~G0;
    //Instantiate D flip-flop
    DFF G1F (G1,DG1,CLK,Clr),
    G0F (G0,DG0,CLK,Clr);
endmodule
```

8.6 HDL Description of Design Example - Structure Description

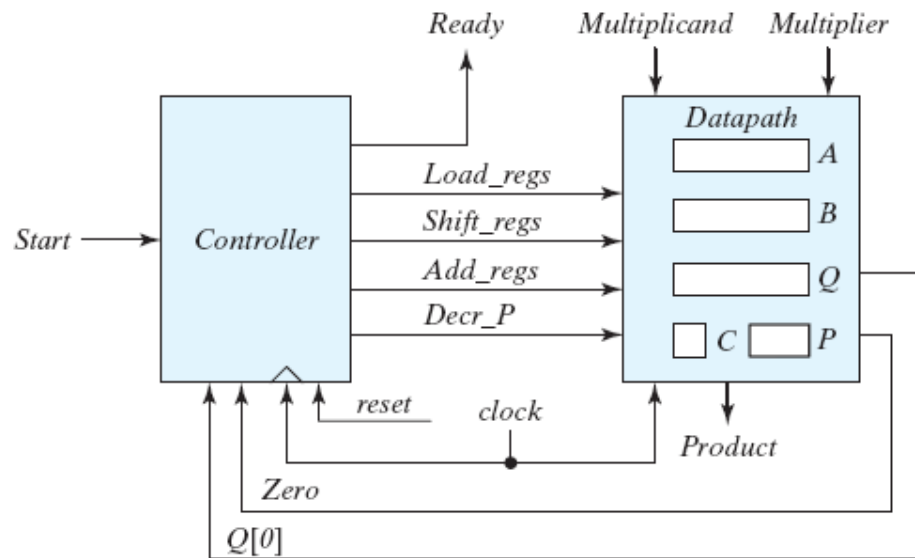
```
//D flip-flop
module DFF (Q,D,CLK,Clr);
  input D,CLK,Clr;
  output Q;
  reg Q;
  always @ (posedge CLK or negedge Clr)
    if (~Clr) Q = 1'b0;
    else Q = D;
endmodule
```

```
//E and F flipflops
module E_F (T1,T2,Clear,CLK,A3,E,F);
  input T1,T2,Clear,CLK,A3;
  output E,F;
  wire E,F,JE,KE,JF,KF;
  //Combinational circuit
  assign JE = T1 & A3,
         KE = T1 & ~A3,
         JF = T2,
         KF = Clear;
  //Instantiate JK flipflop
  JKFF EF (E,JE,KE,CLK),
  FF (F,JF,KF,CLK);
endmodule
```

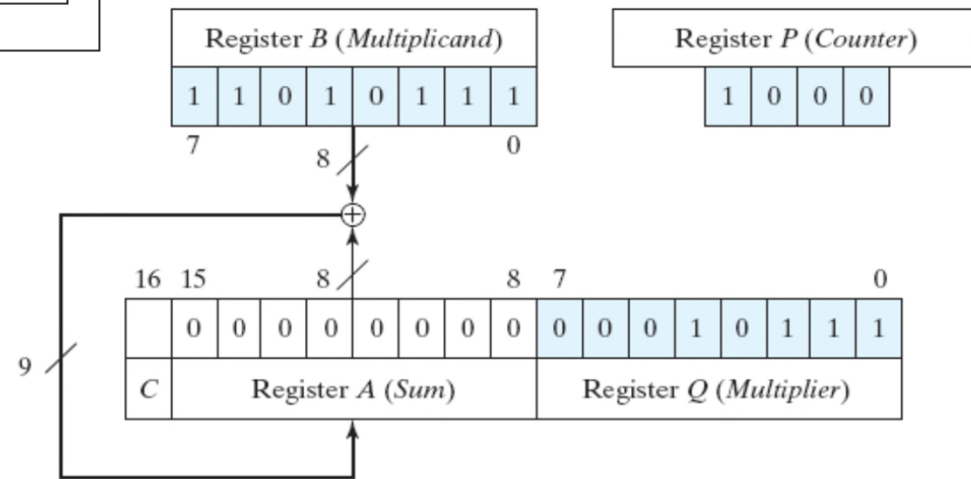
```
//JK flip-flop
module JKFF (Q,J,K,CLK);
  input J,K,CLK;
  output Q;
  reg Q;
  always @ (posedge CLK)
    case ({J,K})
      2'b00: Q = Q;
      2'b01: Q = 1'b0;
      2'b10: Q = 1'b1;
      2'b11: Q = ~Q;
    endcase
endmodule
```

```
//counter with synchronous clear
module counter (Count,Clear,CLK,A);
  input Count,Clear,CLK;
  output [4:1] A;
  reg [4:1] A;
  always @ (posedge CLK)
    if (Clear) A <= 4'b0000;
    else if (Count) A <= A + 1'b1;
    else A <= A;
endmodule
```


8.7 Binary Multiplier

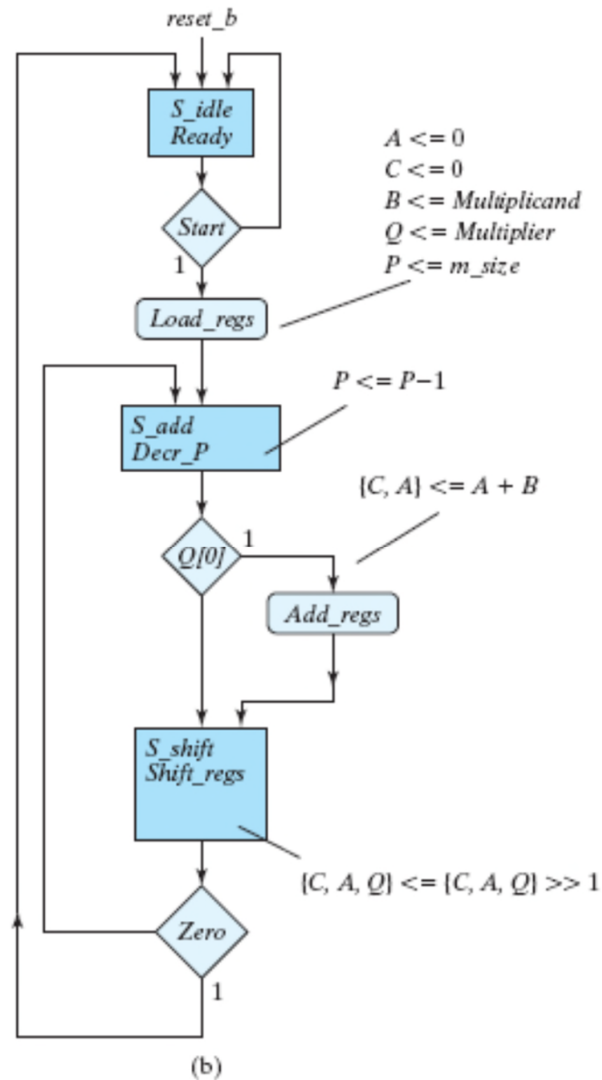
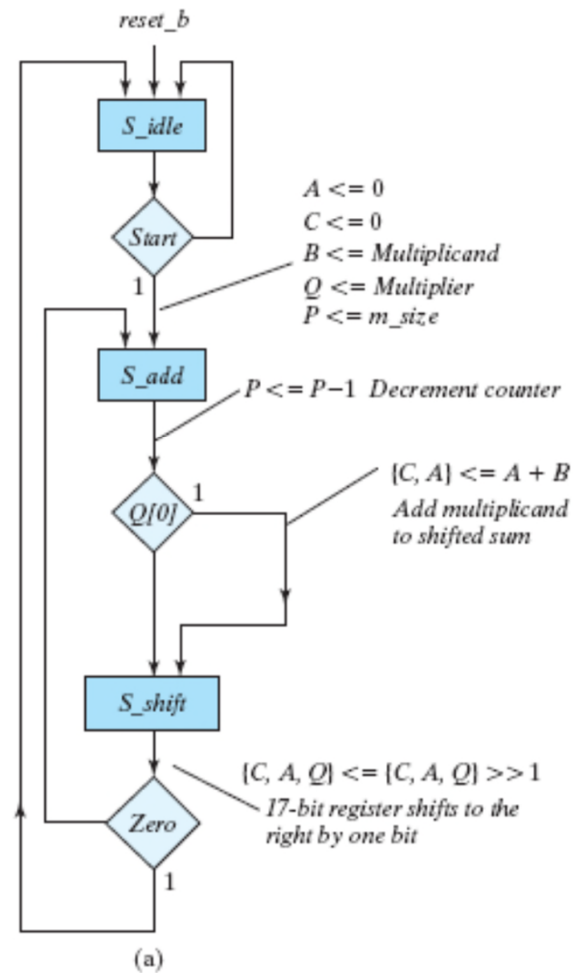


(a)



(b)

8.7 Binary Multiplier – ASMD Chart



8.7 Binary Multiplier – ASMD Chart

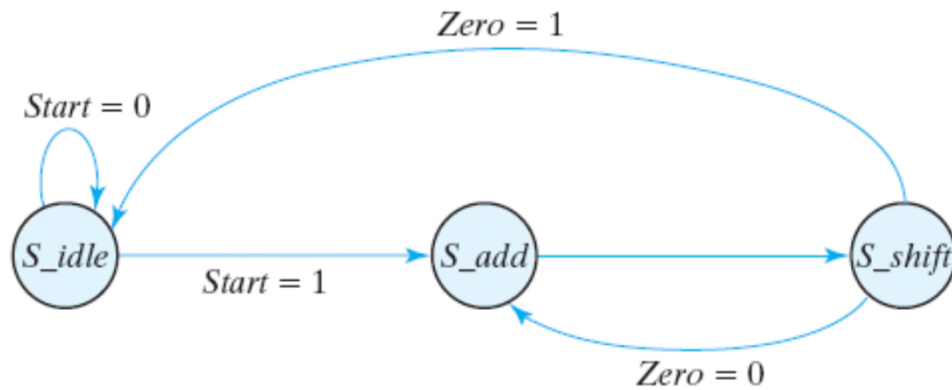
Table 8.5

Numerical Example For Binary Multiplier

Multiplicand $B = 10111_2 = 17_H = 23_{10}$ Multiplier $Q = 10011_2 = 13_H = 19_{10}$

	C	A	Q	P
Multiplier in Q	0	00000	10011	101
$Q_0 = 1$; add B		<u>10111</u>		
First partial product	0	10111		100
Shift right CAQ	0	01011	11001	
$Q_0 = 1$; add B		<u>10111</u>		
Second partial product	1	00010		011
Shift right CAQ	0	10001	01100	
$Q_0 = 0$; shift right CAQ	0	01000	10110	010
$Q_0 = 0$; shift right CAQ	0	00100	01011	001
$Q_0 = 1$; add B		<u>10111</u>		
Fifth partial product	0	11011		
Shift right CAQ	0	01101	10101	000
Final product in $AQ = 0110110101_2 = 1b5_H$				

8.8 Control Logic



(a)

Table 8.6
State Assignment for Control

State	Binary	Gray Code	One-Hot
<i>S_idle</i>	00	00	001
<i>S_add</i>	01	01	010
<i>S_shift</i>	10	11	100

State Transition		Register Operations
<u>From</u>	<u>To</u>	
<i>S_idle</i>		Initial state
<i>S_idle</i>	<i>S_add</i>	$A \leq 0, C \leq 0, P \leq dp_width$
<i>S_add</i>	<i>S_shift</i>	$P \leq P - 1$ if ($Q[0]$) then ($A \leq A + B, C \leq C_{out}$)
<i>S_shift</i>		shift right {CAQ}, $C \leq 0$

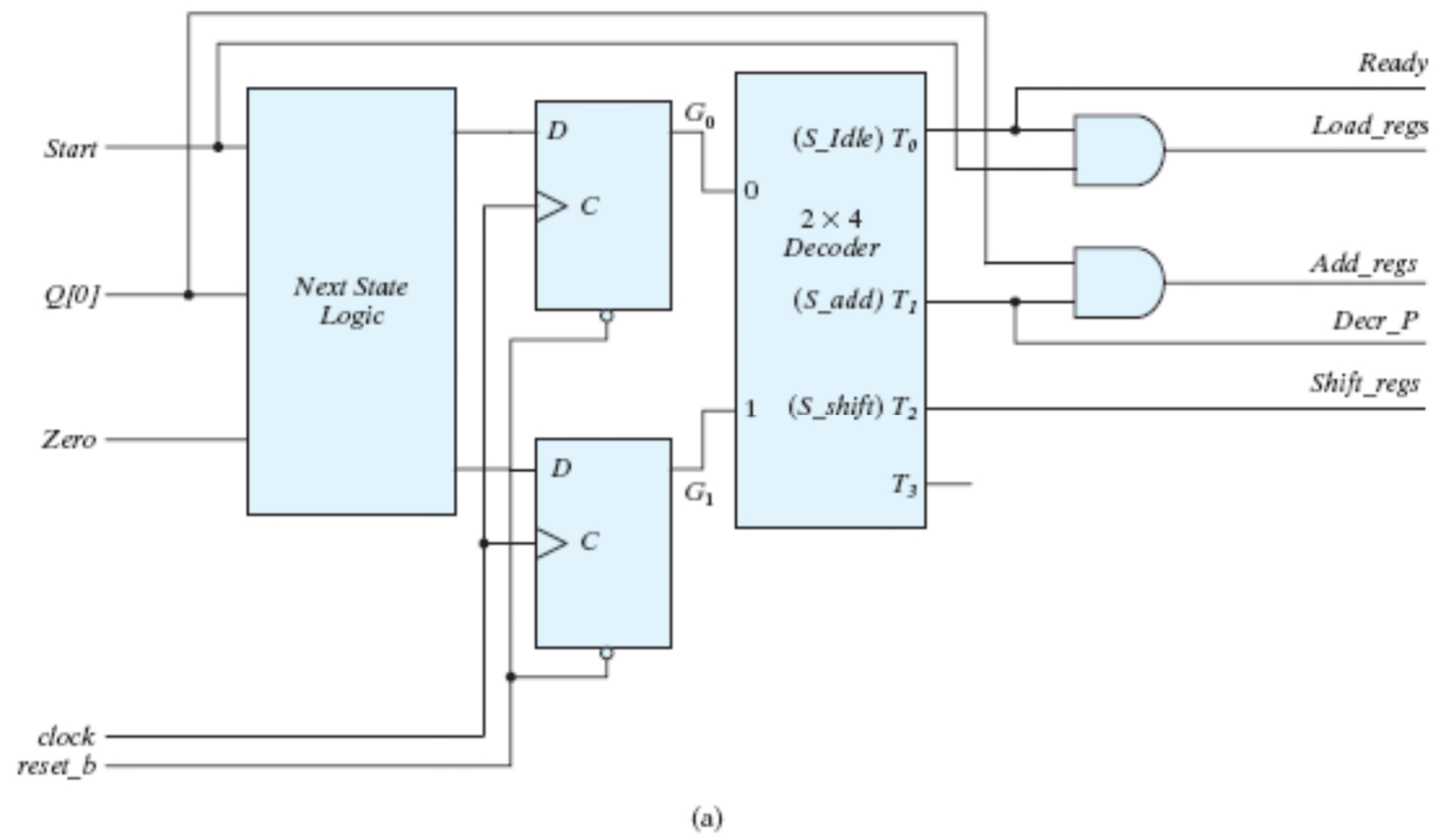
(b)

8.8 Control Logic – Sequence Register and Decoder

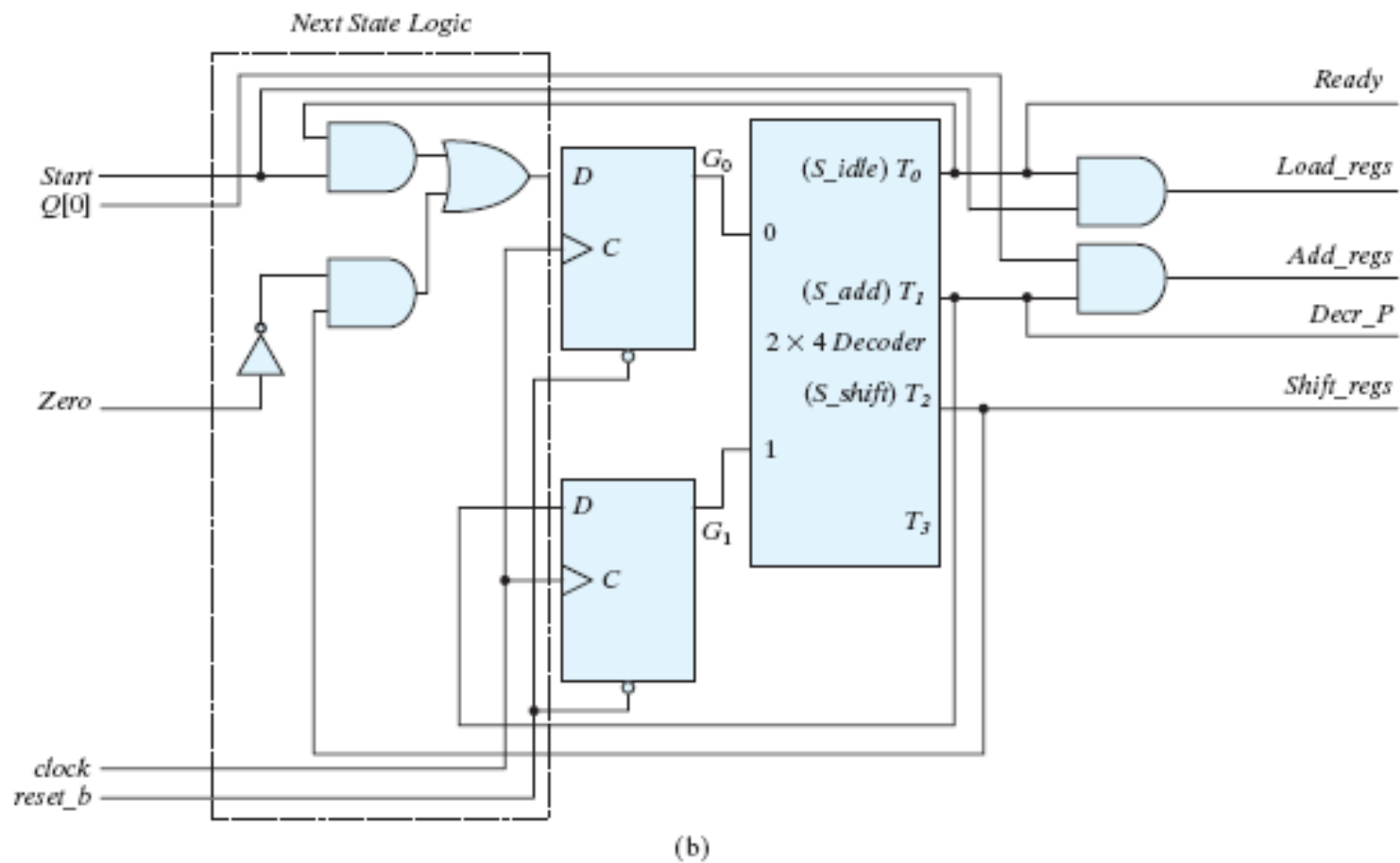
Table 8.7
State Table for Control Circuit

Present-State Symbol	Present State		Inputs			Next State		Ready	Load_regs	Decr_P	Add_regs	Shift_regs
	G ₁	G ₀	Start	Q[0]	Zero	G ₁	G ₀					
<i>S_idle</i>	0	0	0	X	X	0	0	1	0	0	0	0
<i>S_idle</i>	0	0	1	X	X	0	1	1	1	0	0	0
<i>S_add</i>	0	1	X	0	X	1	0	0	0	1	0	0
<i>S_add</i>	0	1	X	1	X	1	0	0	0	1	1	0
<i>S_shift</i>	1	0	X	X	0	0	1	0	0	0	0	1
<i>S_shift</i>	1	0	X	X	1	0	0	0	0	0	0	1

8.8 Control Logic – Sequence Register and Decoder



8.8 Control Logic – Sequence Register and Decoder



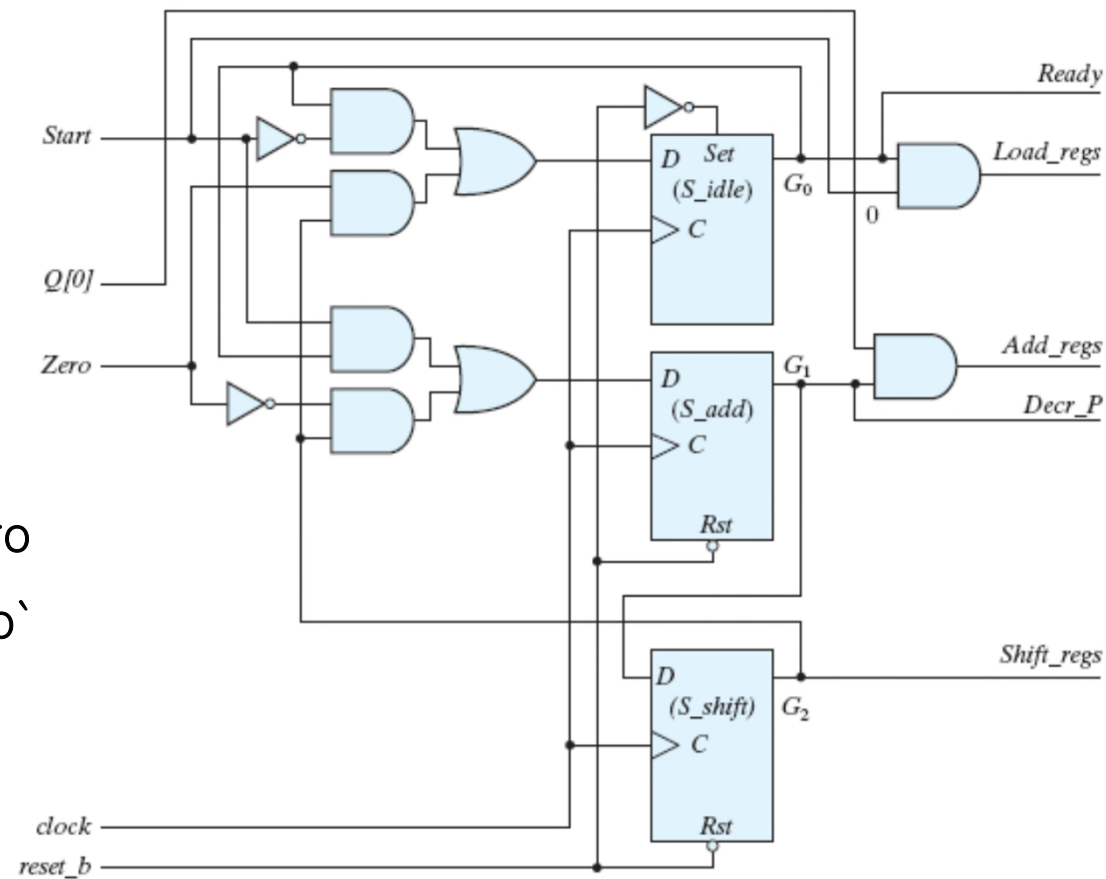
8.8 Control Logic – Sequence Register and Decoder

8.8 Control Logic – One Flip-Flop per State

$$D_{G0} = G_0 \text{Start}' + G_2 \text{Zero}$$

$$D_{G1} = G_0 \text{Start} + G_2 \text{Zero}'$$

$$D_{G2} = G_1$$



8.9 HDL Description of Binary Multiplier

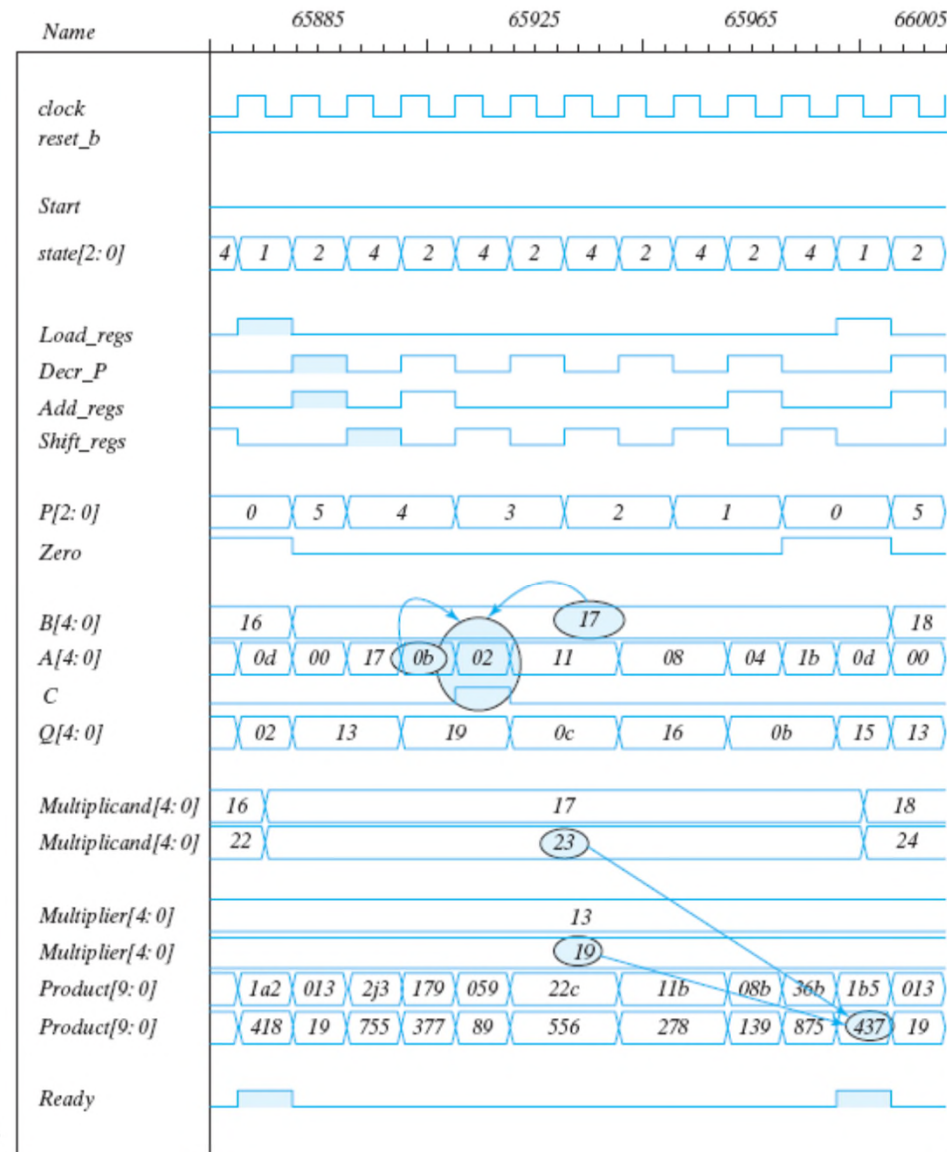
```
//HDL Example 8-5
//RTL description of binary multiplier
//Block diagram Fig.8-13 and ASM chart Fig.8-14
//n = 5 to compare with Table 8-4
module mltp(S,CLK,Clr,Binput,Qinput,C,A,Q,P);
    input S,CLK,Clr;
    input [4:0] Binput,Qinput;      //Data inputs
    output C;
    output [4:0] A,Q;
    output [2:0] P;
//System registers
    reg C;
    reg [4:0] A,Q,B;
    reg [2:0] P;
    reg [1:0] pstate, nstate;      //control register
    parameter T0=2'b00, T1=2'b01, T2=2'b10, T3=2'b11;
//Combinational circuit
    wire Z;
    assign Z = ~|P;                //Check for zero
```

```
//State transition for control
//See state diagram Fig. 8-15(a)
always @(negedge CLK or negedge Clr)
    if (~Clr) pstate = T0;
    else pstate <= nstate;
always @(S or Z or pstate)
    case (pstate)
        T0: if (S) nstate = T1;
        T1: nstate = T2;
        T2: nstate = T3;
        T3: if (Z) nstate = T0;
            else nstate = T2;
    endcase
```

8.9 HDL Description of Binary Multiplier

```
//Register transfer operations
//See register operation Fig.8-15(b)
always @(negedge CLK)
  case (pstate)
    T0: B <= Binput;           //Input multiplicand
    T1: begin
      A <= 5'b00000;
      C <= 1'b0;
      P <= 3'b101;           //Initialize counter to n=5
      Q <= Qinput;           //Input multiplier
    end
    T2: begin
      P <= P - 3'b001;       //Decrement counter
      if (Q[0])
        {C,A} <= A + B;      //Add multiplicand
      end
    T3: begin
      C <= 1'b0;             //Clear C
      A <= {C,A[4:1]};       //Shift right A
      Q <= {A[0],Q[4:1]};    //Shift right Q
    end
  endcase
endmodule
```

8.9 HDL Description of Binary Multiplier



8.9 HDL Description of Binary Multiplier - Testing the Multiplier

● Testing the Multiplier

```
//HDL Example 8-6
//-----
//Testing binary multiplier
module test_mlt;
  //Inputs for multiplier
  reg S,CLK,Clr;
  reg [4:0] Binput,Qinput;
  //Data for display
  wire C;
  wire [4:0] A,Q;
  wire [2:0] P;
```

```
//Instantiate multiplier
mltp mp(S,CLK,Clr,Binput,Qinput,C,A,Q,P);
initial
begin
  S=0; CLK=0; Clr=0;
  #5 S=1; Clr=1;
  Binput = 5'b10111;
  Qinput = 5'b10011;
  #15 S = 0;
end
initial
begin
  repeat (26)
    #5 CLK = ~CLK;
  end
//Display computations and compare with Table 8-4
always @(negedge CLK)
  $strobe ("C=%b A=%b Q=%b P=%b time=%0d",C,A,Q,P,$time);
endmodule
```

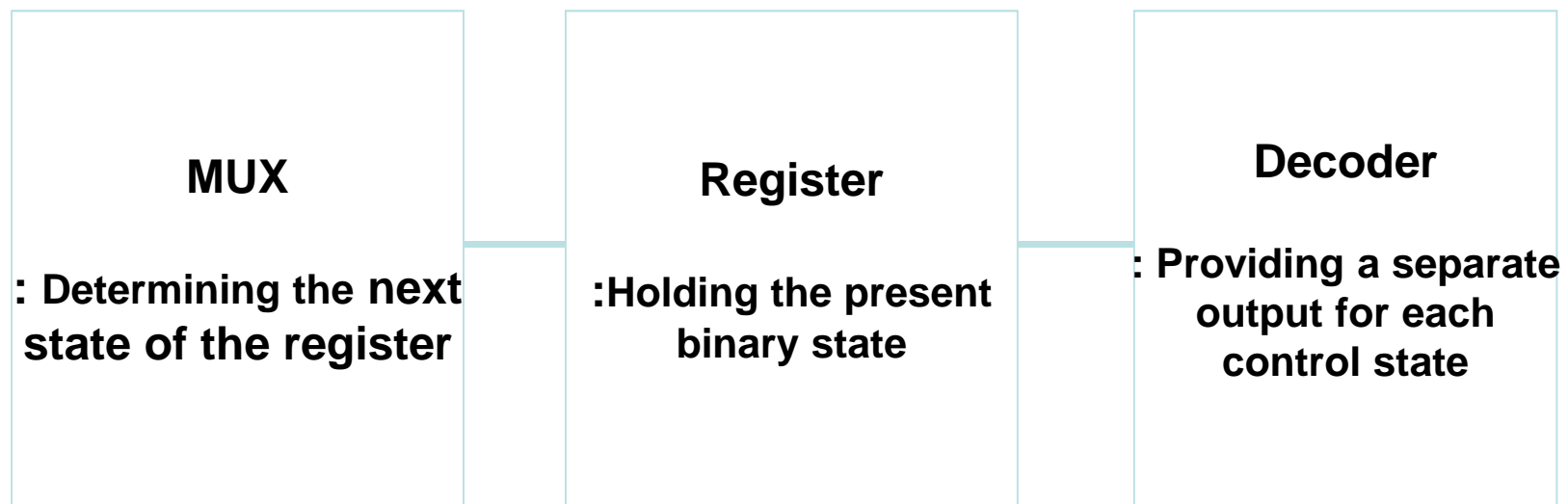
8.9 HDL Description of Binary Multiplier - Behavioral Description of Multiplier

● Behavioral Description of Multiplier

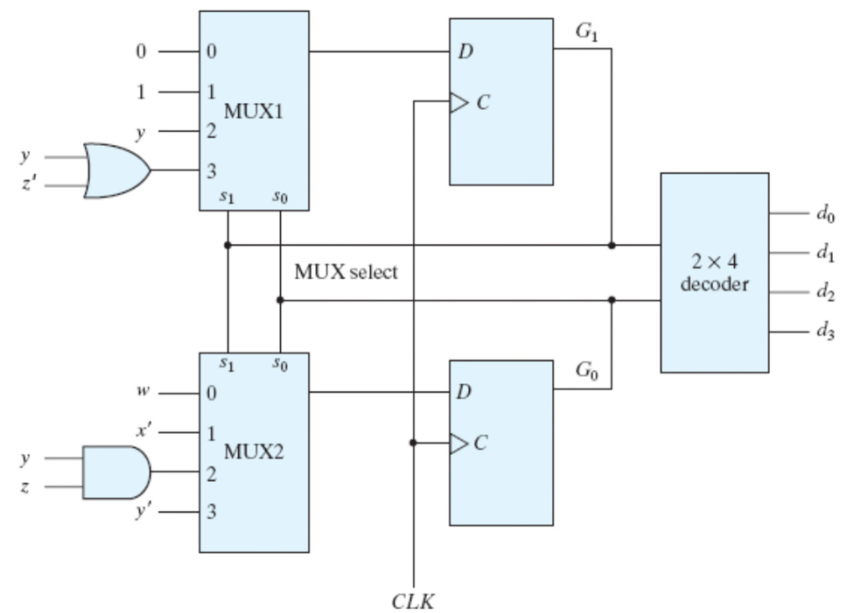
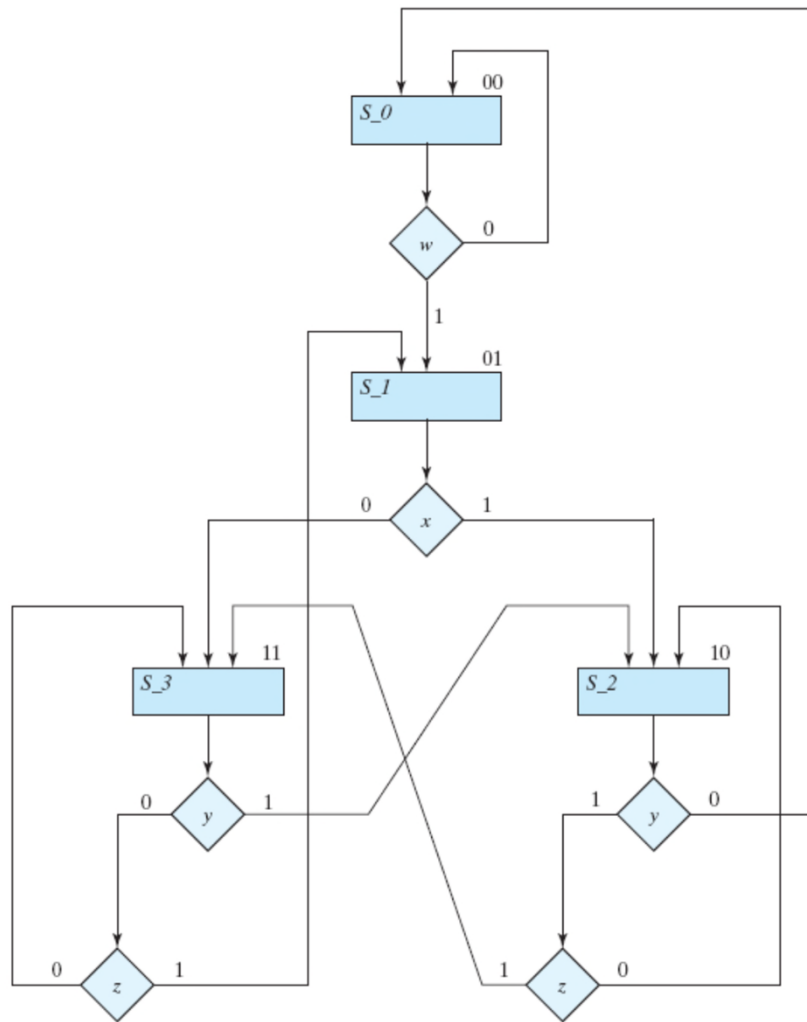
```
//HDL Example 8-7
//-----
//Behavioral description of multiplier (n = 8)
module Mult (A,B,Q);
    input [7:0] B,Q;
    output [15:0] A;
    reg [15:0] A;
    always @ (B or Q)
        A = B * Q;
endmodule
```

8.10 Design with Multiplexers

◆ Combination Circuits- control unit



8.10 Design with Multiplexers

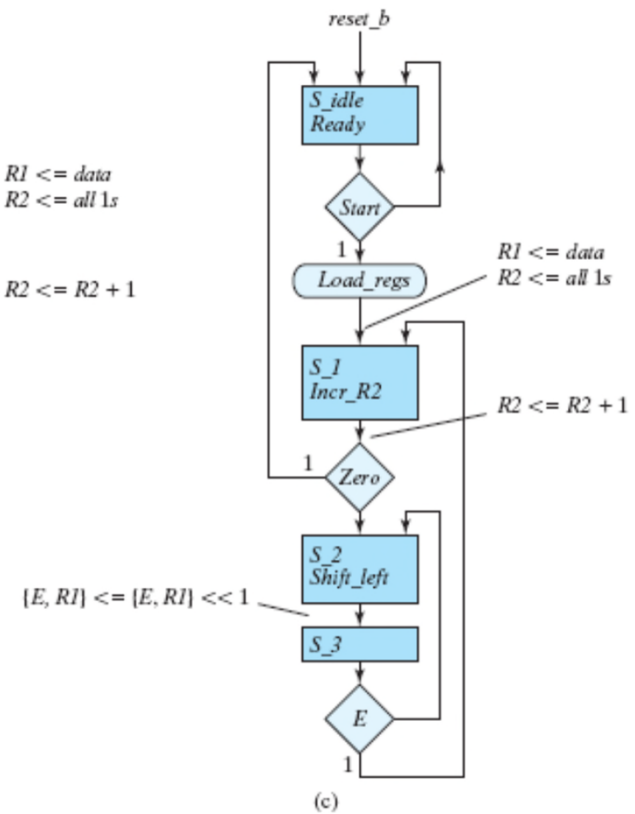
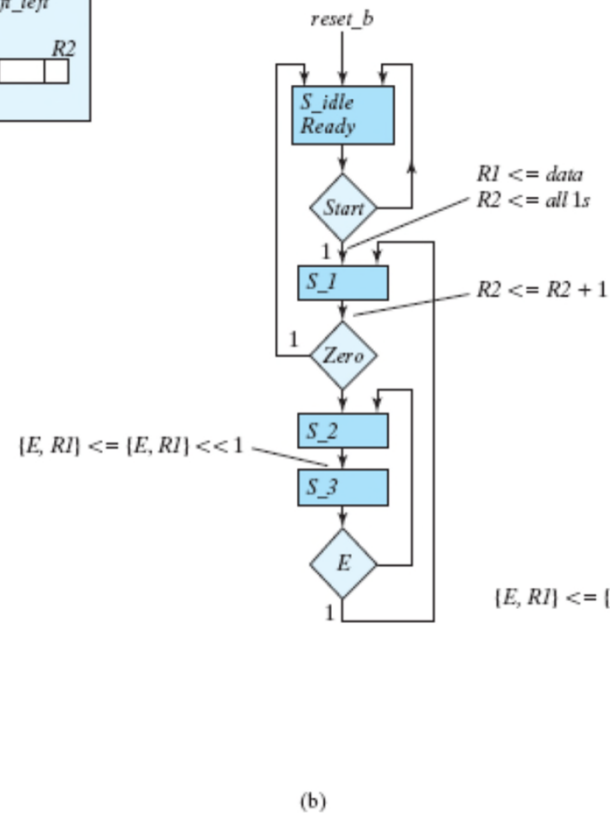
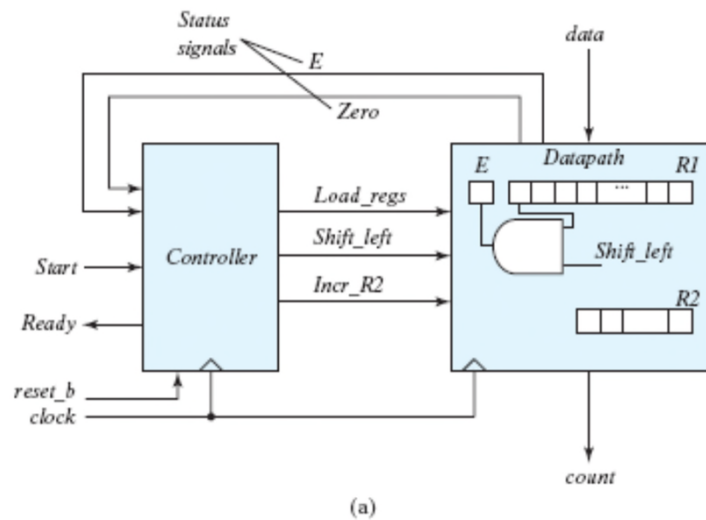


8.10 Design with Multiplexers

Table 8.8
Multiplexer Input Conditions

Present State		Next State		Input Condition	Inputs	
G_1	G_0	G_1	G_0	s	MUX1	MUX2
0	0	0	0	w'		
0	0	0	1	w	0	w
0	1	1	0	x		
0	1	1	1	x'	1	x'
1	0	0	0	y'		
1	0	1	0	yz'		
1	0	1	1	yz	$yz' + yz = y$	yz
1	1	0	1	$y'z$		
1	1	1	0	y		
1	1	1	1	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

8.10 Design with Multiplexers

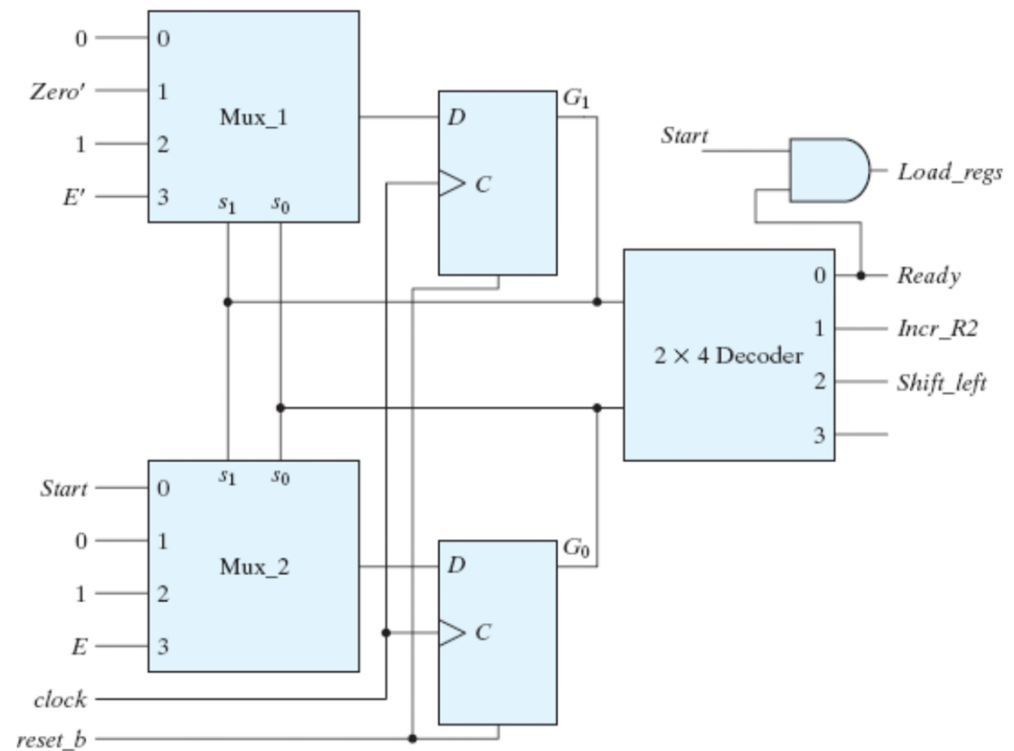


8.10 Design with Multiplexers – Design Example

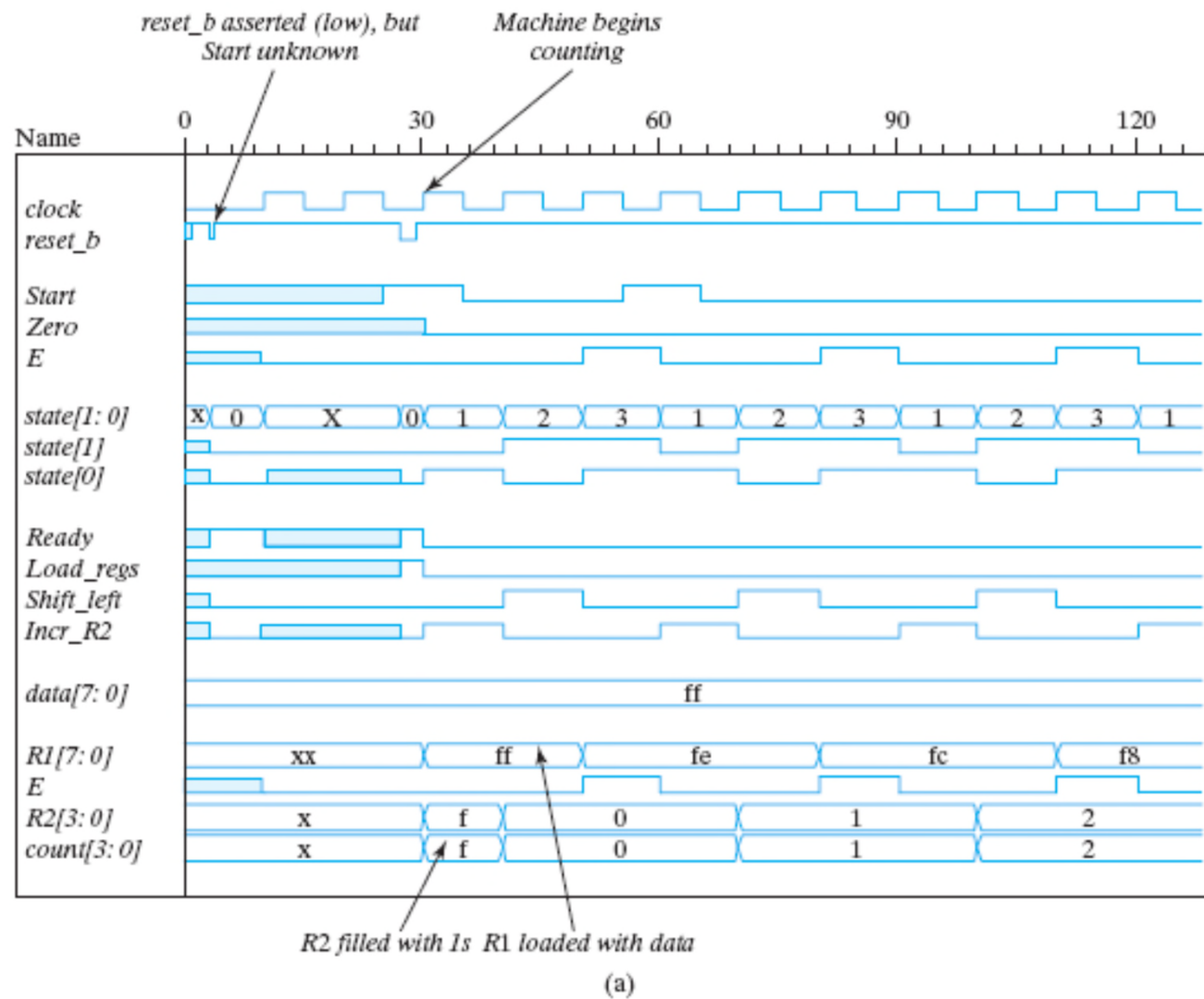
Table 8.9

Multiplexer Input Conditions for Design Example

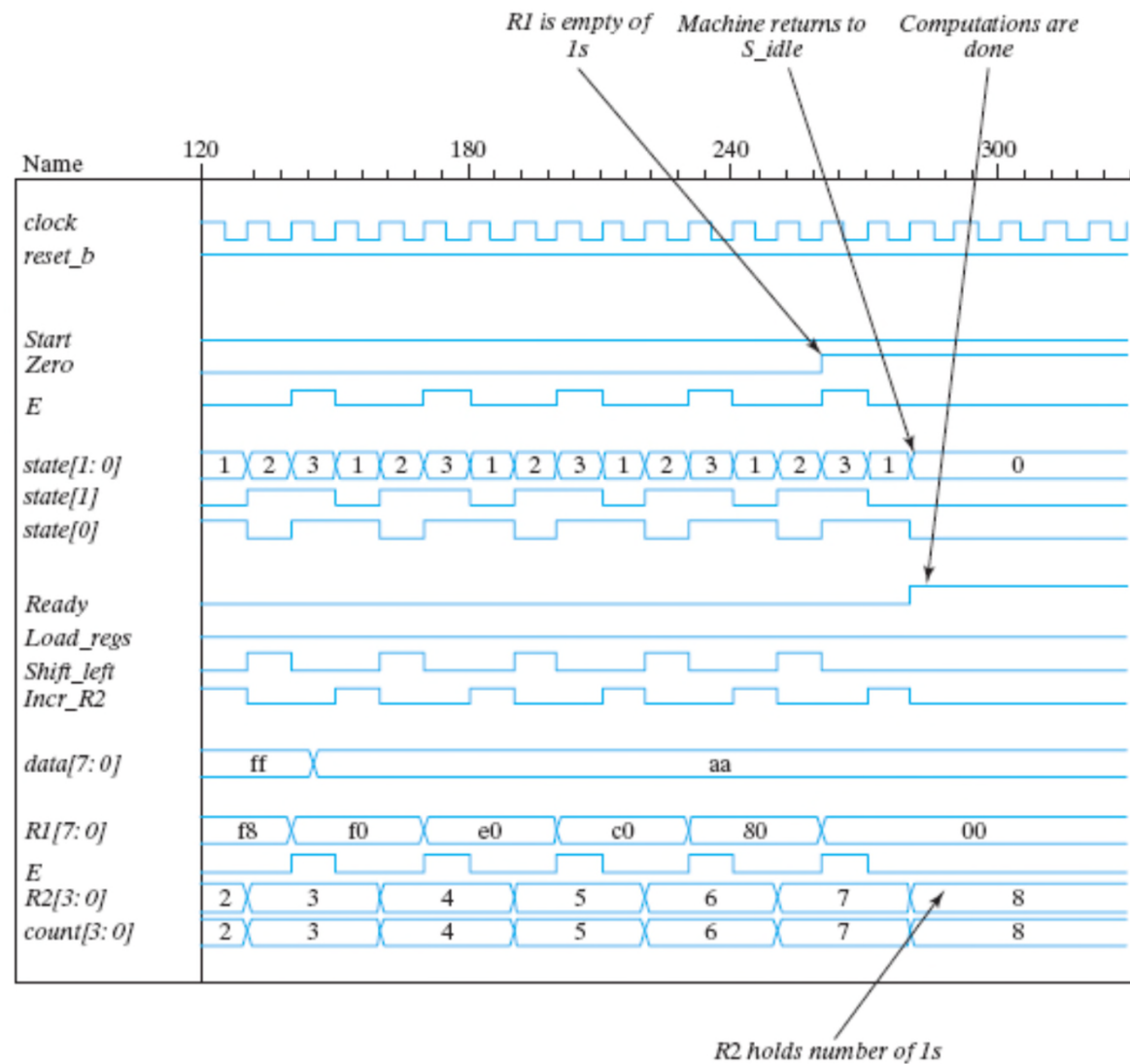
Present State		Next State		Input Conditions	Multiplexer Inputs	
G_1	G_0	G_1	G_0		MUX1	MUX2
0	0	0	0	$Start'$		
0	0	0	1	$Start$	0	$Start$
0	1	0	0	$Zero$		
0	1	1	0	$Zero'$	$Zero'$	0
1	0	1	1	None	1	1
1	1	1	0	E'		
1	1	0	1	E	E'	E



8.10 Design with Multiplexers – Design Example



8.10 Design with Multiplexers – Design Example



(b)