A sequential circuit: ch5.5

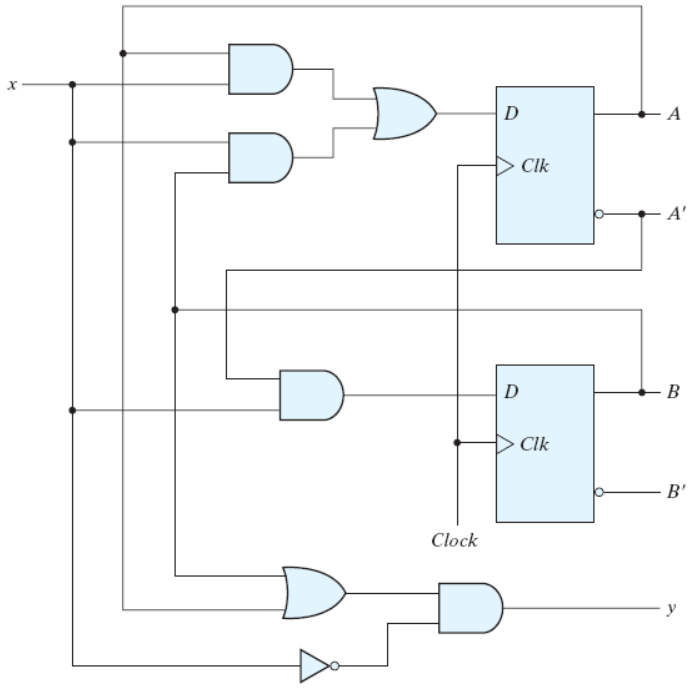


Figure 5.15

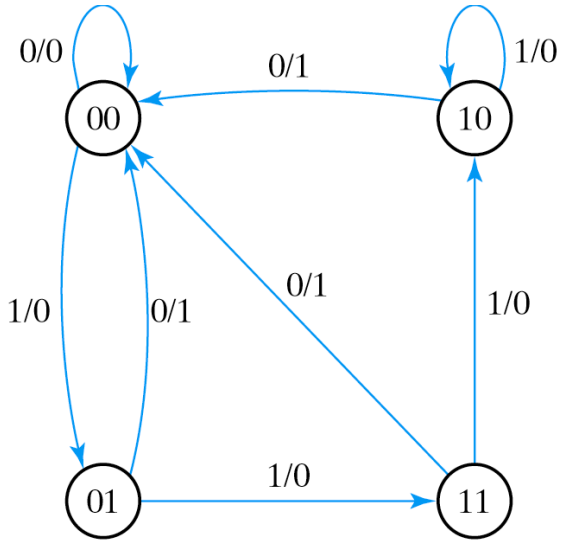
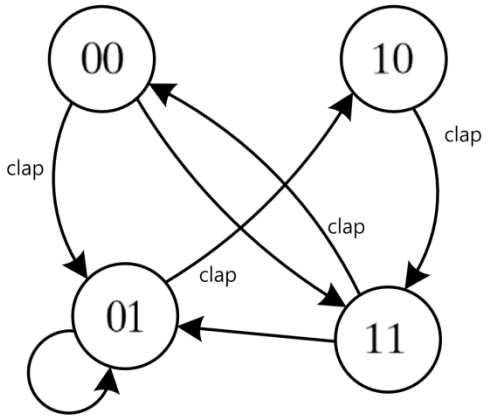


Fig. 5-16 State Diagram of the Circuit of Fig. 5-15

Another simple state machine



State diagram for control

Don't you think it's too simple?

Features to be implemented:

노래하는 시간? 최소 1분?
박수 몇 번에 동작을 바꿔?

Walk:

Sing:
1분이 경과?

Cry:
박수횟수 세기

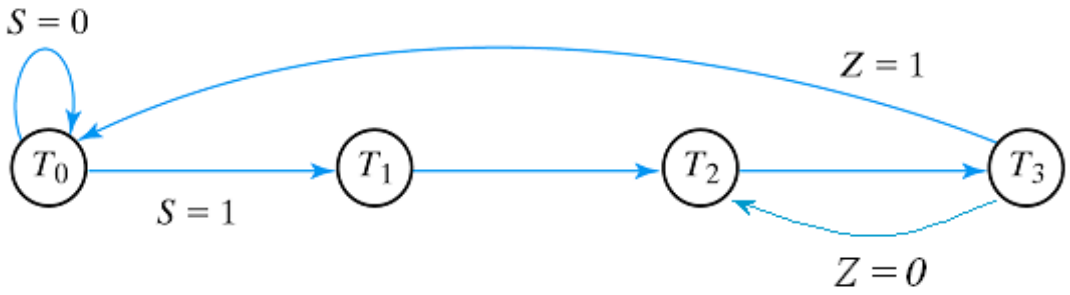
Hop:

Register transfer operations

How many flipflops?

How many states?

How about state diagram or state table?



(a) State diagram

T_0: Initial state

T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$

This is a real digital module…

T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

T_3: shift right CAQ, $C \leftarrow 0$

(b) Register transfer operations

Fig. 8-15 Control Specifications for Binary Multiplier

Chap. 8 Register Transfer Level

8-2 Register transfer level (RTL)

How to design a large digital system?

- no more state table technique (too complex)
- **modular design** approach (i.e. partition to modular subsystems)

How to define *module*

- control and data operation

Data transaction between modules?

- modules are defined by ***a set of registers*** and ***the operations*** that are performed on the data stored in registers
- modules are connected with *common data and control paths*

⇒ **information flow and data processing (*i.e. data transaction*) between modules** is ***register transfer*** operation.

레지스터 전송연산

Representation of digital system

- RTL representation

RTL specifications (register set, operation and control)

* A **register** means ***the group of FF's and associated gates*** that is capable to ***store data*** and ***perform operations***

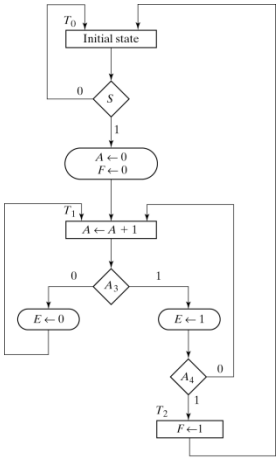
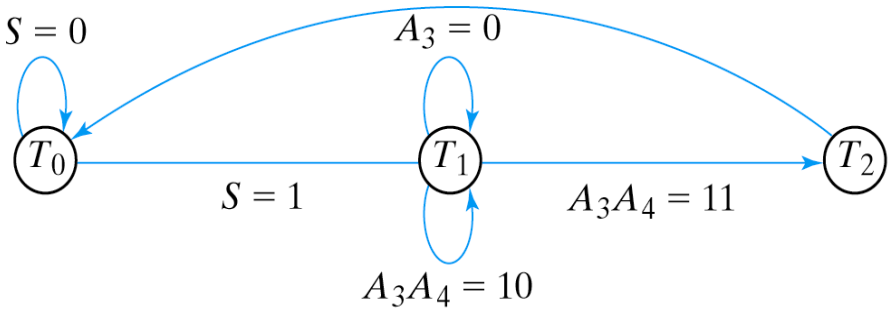


Fig. 8-9 ASM Chart for Design Example

Modules are connected with *common data and control paths*

Can be divided into two parts: control and data operation



(a) State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

Register transfer operations

Fig. 8-11 of Design Example

RTL description

Chap. 8 Register Transfer Level

8-2 Register transfer level (RTL)

How to design a large digital system?

- no more state table technique (too complex)
- **modular design** approach (i.e. partition to modular subsystems) *Modules*

Data transaction between modules?

- *Modules* are defined by ***a set of registers*** and ***the operations*** that are performed on the data stored in registers
- modules are *connected with* ***common data and control paths***

⇒ **information flow and data processing (*i.e. data transaction*) between modules** is ***register transfer*** operation.

레지스터 전송연산

In general,
Data processing circuits
called ***datapath***

Representation of digital system

- RTL representation

RTL specifications (register set, operation and control)

* A **register** means ***the group of FF's and associated gates*** that is capable to ***store data*** and ***perform operations***

Data components:
registers,
adders,
shift registers,
comparators, …

Register and vector

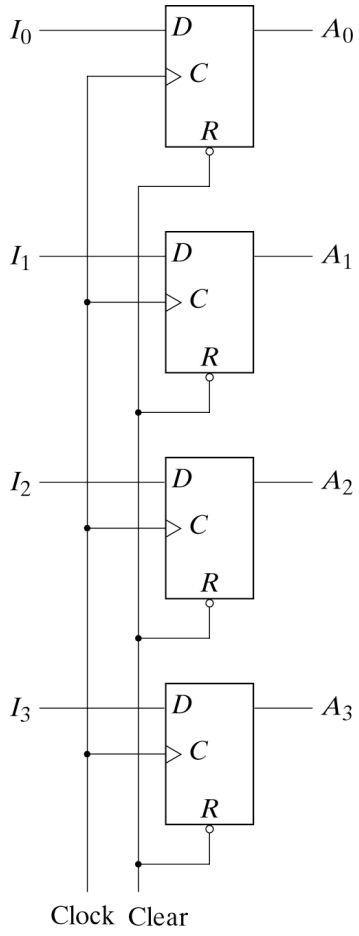
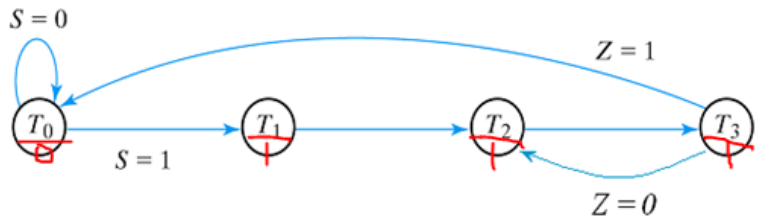


Fig. 6-1 4-Bit Register

```
module Reg_4bits(I,A,clk,nClr);  // define i/o
  input clk,nClr;
  input [3:0] I;
  output [3:0] A;
  reg [3:0] A;
        //output reg [3:0] A;

 always @(posedge clk)
    if (~nClr) A <= 4'b0000;    //Initialize
    else A <= I;

endmodule
```

```
module Reg_4bits(I,Q,clk,nClr);  // define i/o
  input clk,nClr;
  input [3:0] I;
  output [3:0] Q;
  reg [3:0] Q;
        //output reg [3:0] Q;

 always @(posedge clk)
    if (~nClr) Q <= 4'b0000;    //Initialize
    else Q <= I;

endmodule
```

FSM –Sequential behavior

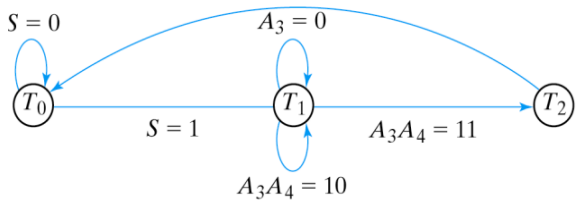


(a) State diagram

Two procedures design

```
module Example_FSM (S,clk,nClr,Z,X);  // define i/o
  input S,clk,nClr,Z;
  output reg X;

//Specify system registers
  reg [1:0] pstate, nstate;   //control register

//Encode the states
  parameter T0 = 2'b00, T1 = 2'b01,
            T2 = 2'b11 , T3 = 2'b10;

//State update procedure
  always @(posedge clk, negedge nClr)
    if (~nClr) pstate <= T0;    //Initial state
    else pstate <= nstate;    //Clocked operations

//Combinational Logic procedure
  always @ (S, Z, pstate) begin
     X <= 1;
     case (pstate)
       T0: begin  X<=0;
            if(S) nstate <= T1; else nstate <= T0; end
       T1: nstate <= T2;
       T2: nstate <= T0;
       T3: if(Z) nstate <= T0; else nstate<=T2;
     endcase
  end
endmodule
```

RTL Description



(a) State diagram for control

T_0: if ($S = 1$) then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if ($A_3 = 1$) then E $\leftarrow 1$

if ($A_3 = 0$) then E $\leftarrow 0$

T_2: $F \leftarrow 1$

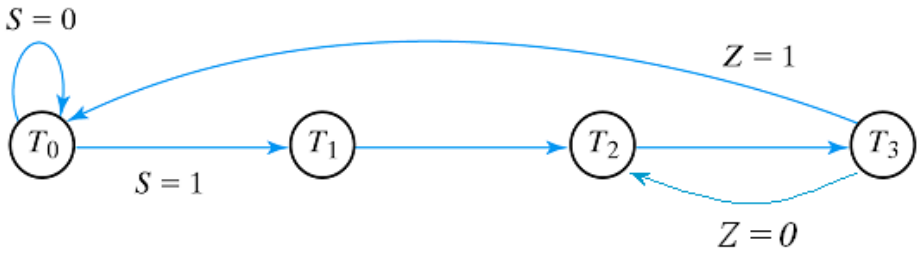
(a) Register transfer operations

Fig. 8-11 Register Transfer Level Description of Design Example

```
module Example_RTL (S,CLK,Clr,E,F,A);
  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 -- control sequence
//ASM - See state diagram of 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; else nstate = T0;
       T1: if(A[3] & A[4]) nstate = T2; else nstate = T1;
       T2: nstate = T0;
     endcase
```

```
//Register transfer operations, See list of operations Fig.8-11(b)
  always @(posedge CLK)  // nonblocking stmt for synch
    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
```

In general,
Data processing circuits
called ***datapath***



(a) State diagram

T_0: Initial state

T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$

T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

T_3: shift right $CAQ, C \leftarrow 0$

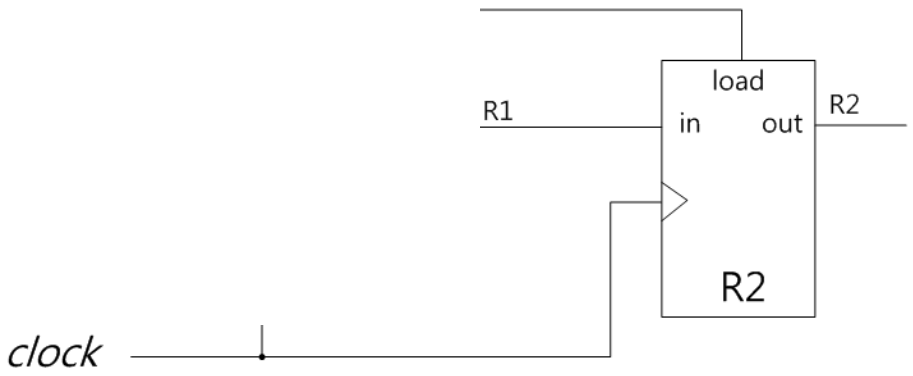
(b) Register transfer operations

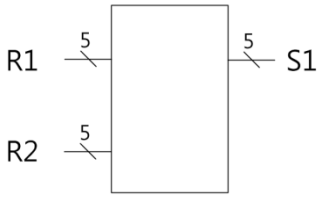
Fig. 8-15 Control Specifications for Binary Multiplier

Module can be divided into 2 parts: *control and data operation*

Modules are connected with *common data and control paths*

Register Operation





R1 register의 output을
R2 register의 input port에 연결

Information transfer

- designation by a replacement operator. R2 ← R1 (***connection***)
- *when transferred?* (***transfer of contents***)

* transfer statement implies the **circuit connection**

* **conditional statement** (*control* signal) and *clock* controls the transfer time.

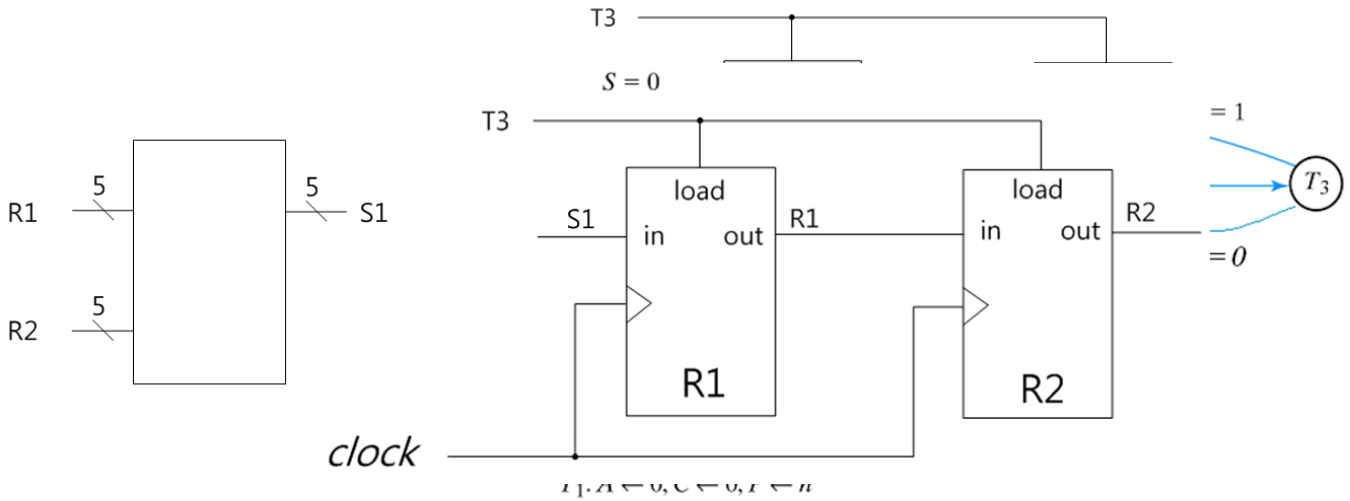
* It is assumed that all transfers occur during a clock edge transition

ex1) *If (T3=1)* then (*R2 ← R1, R1 ← R2*)

Register Operation

Register operation

- performed in parallel during one clock cycle.
- result may replace the previous information or may be transferred to another registers



T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

T_3: shift right CAQ, $C \leftarrow 0$

(b) Register transfer operations

```
1  module Binary_Counter_4_Par_Load (
2
3    output reg [3:0]  A_count,    // Data output
4    output    C_out,  // Output carry
5    input [3:0]   Data_in,    // Data input
6    input     Count,  // Active high to count
7              Load,   // Active high to load
8              CLK,    // Positive edge sensitive
9              Clear   // Active low
10     );
11
12 assign C_out = Count & (~Load) & (A_count == 4'b1111);
13
14 //always @ (posedge CLK, negedge Clear, posedge Load)
15 always @ (posedge CLK)
16   if (~Clear)       A_count <= 4'b0000;
17   else if (Load)    A_count <= Data_in;
18   else if (Count)   A_count <= A_count + 1'b1;
19   else              A_count <= A_count; // redundant statement
20 endmodule
```

8-3 Register transfer level in HDL

RTL description in Verilog HDL

* Remember that a **register** means ***the group of FF's and associated gates***

- combinational circuit ← continuous assignment or procedural assignment statements
- register transfer ← procedural assignment statements

symbol for the transfer : "=" or "<="

- synchronization by ***always*** statement with an event control of posedge or negedge

Examples

1. **assign** S=A+B; // continuous assignment --- combination circuit (S can't be reg)
2. **always** @ (A or B) // procedural assignment w/o a clock --- combination circuit (S: reg)
   S=A+B;
3. **always** @ (**posedge** clock) // **blocking** procedural assignment --- sequential circuit
   begin
   RA = RA + RB; RD = RA; // executed sequentially
   end
4. **always** @ (**negedge** clock) //**nonblocking** procedure --- sequential circuit
   begin
   RA <= RA+RB; RD <= RA;// concurrent operation @clock edge
   end

Use nonblocking procedure to ensure synchronous operations

Logic synthesis

Logic synthesis : automatic transformation of HDL codes
into optimized logic circuits (a netlist of gates)

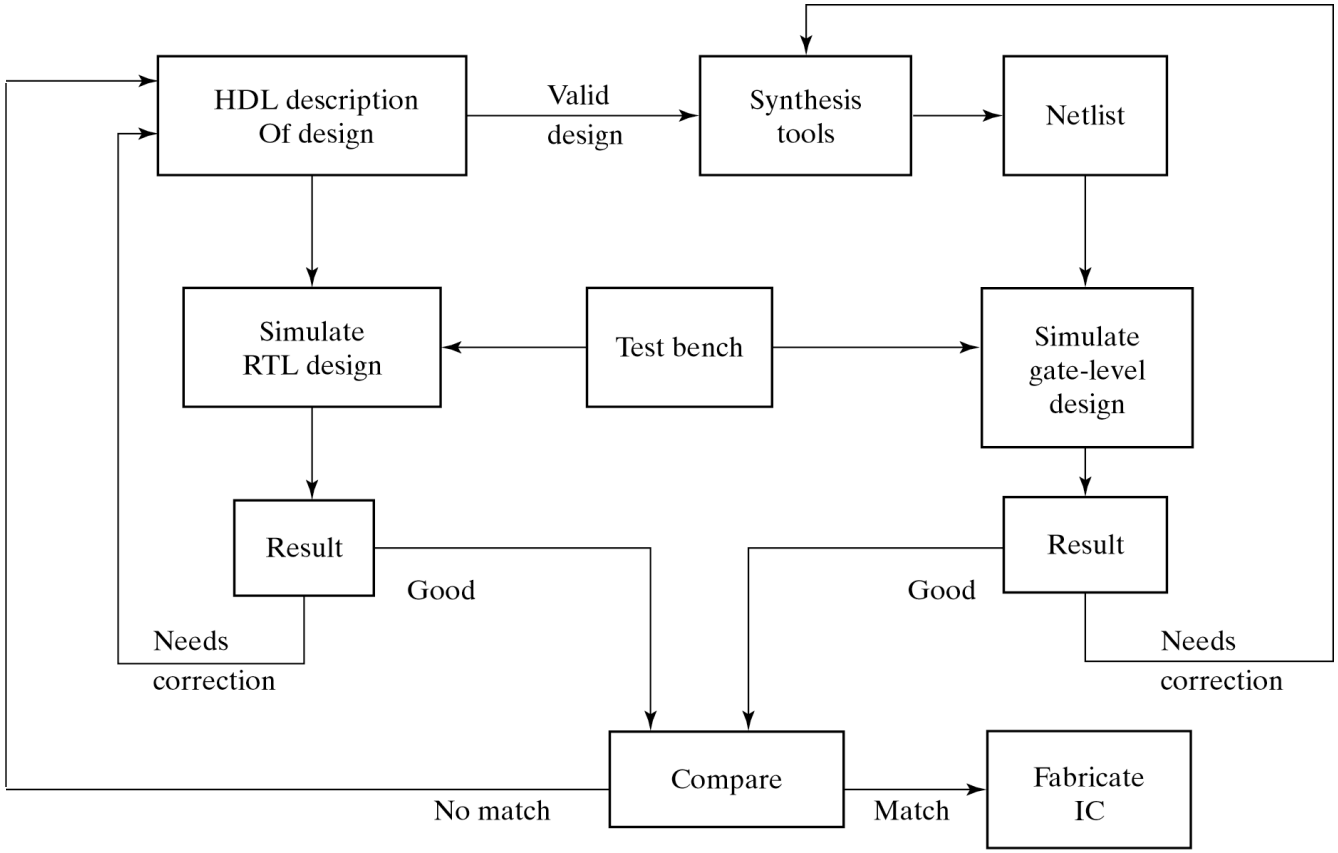


Fig. 8-1 Process of HDL Simulation and Synthesis

8-4 Algorithmic State Machines (ASMs)

Binary information stored in a digital system can be classified as either ***data*** *or* ***control information***

data : information that is manipulated to perform data processing tasks

data operations by adder, decoder, multiplexer, counter, shift register, …

control : command signals that supervise the data operations

=> Logic design

1. design of ***data processing circuits*** (datapath)
2. design of ***control circuits*** for the control of action sequence

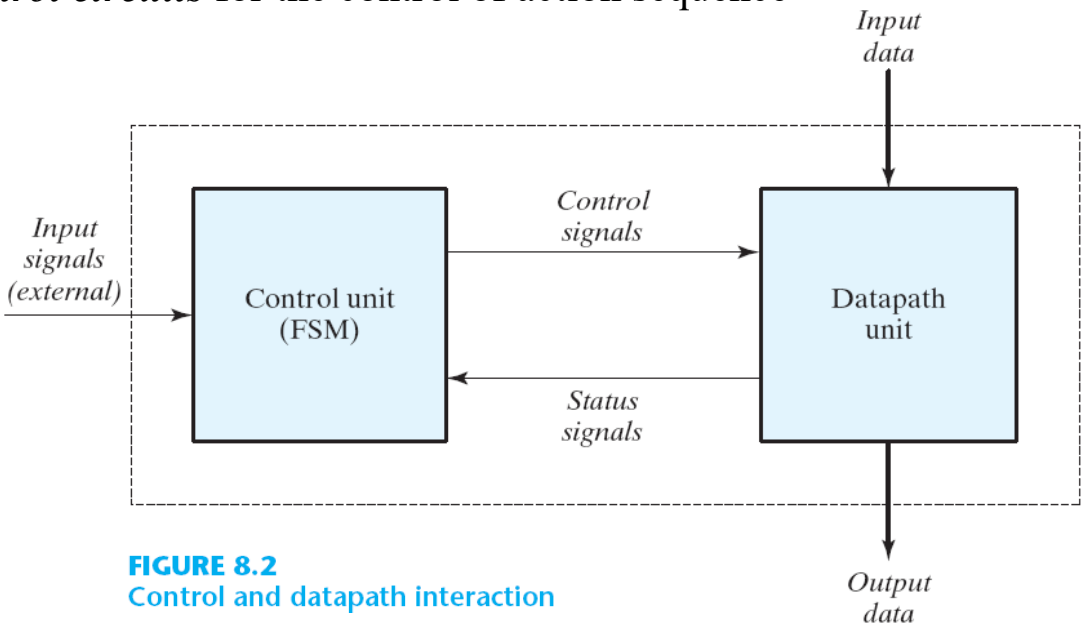


FIGURE 8.2
Control and datapath interaction

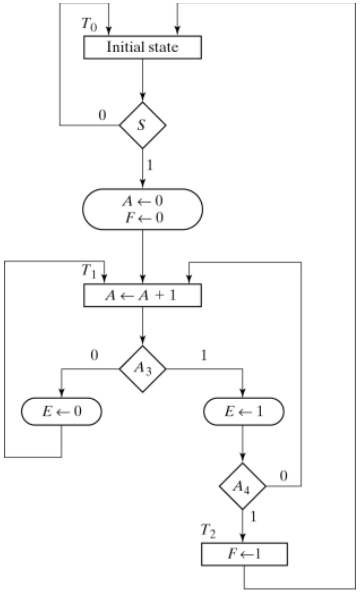
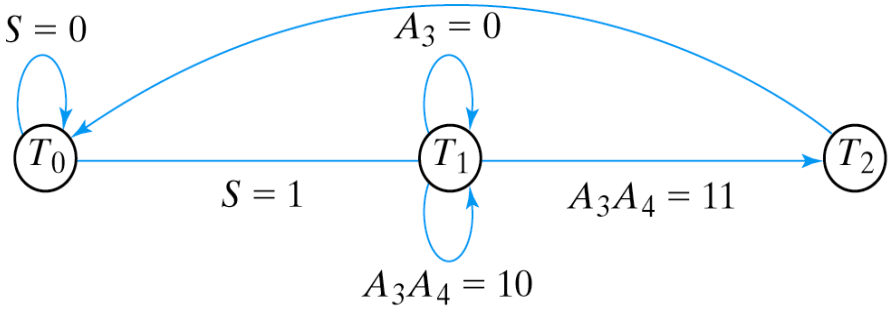


Fig. 8-9 ASM Chart for Design Example

ASM chart



(a) State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

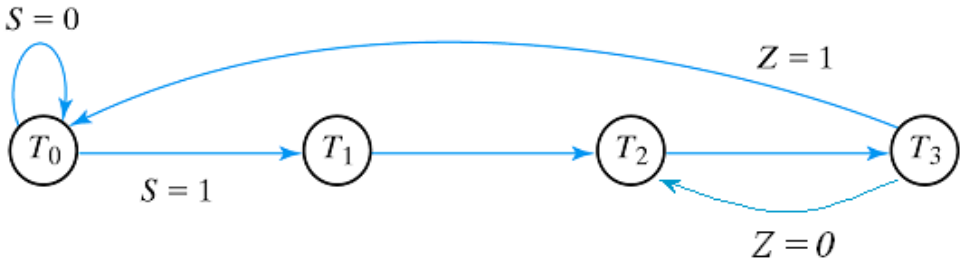
if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

(a) Register transfer operations

RTL description



(a) State diagram

Control Unit

T_0: Initial state

T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$

T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

Datapath Unit

T_3: shift right CAQ, $C \leftarrow 0$

(b) Register transfer operations

Fig. 8-15 Control Specifications for Binary Multiplier

Digital system operation

clock edge

at any given time

the control state initiates a prescribed data operation

the state goes to the next state depending on status condition and external inputs

Role of the control logic

provide a time sequence of *signals for* ***initiating the prescribed datapath operation***

and also determine ***the next state*** of the control circuit

How to specify the control sequence and data-processing tasks?

Hardware algorithm ← described by ***algorithmic state machine*** (ASM)

* algorithm? *procedural steps that specify how to obtain a solution to a problem*
* state machine? *another name for a sequential circuit*

ASM ?

Conventional flowchart : step-by-step

sequence of procedural steps and decision paths for an algorithm
without concern for their timing relationship

ASM : concurrency

Concurrent?
statements are considered in parallel
the ordering of statements does not matter

flowchart + timing relationship between the states of sequential controller
and the events that occur while going to one state to the next.

i.e. **flowchart considering the constraint of digital hardware**

ASM chart

Basic elements

state box

decision box : describe the effect of an input

conditional box : - unique to ASM chart

- input path must come from decision box
- The register operations or outputs listed inside box are generated during a given state if the condition is true

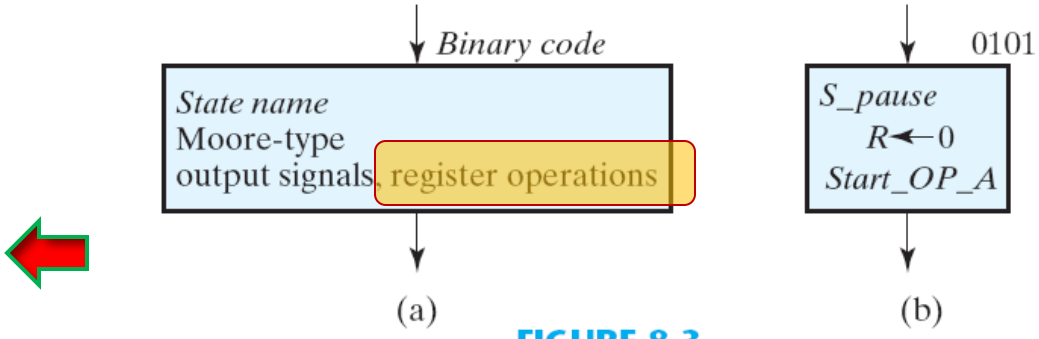


FIGURE 8.3
ASM chart state box

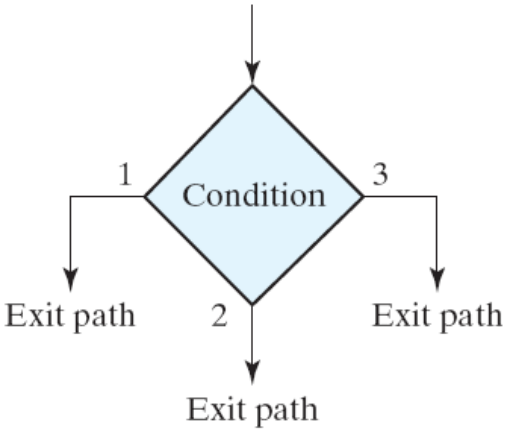


FIGURE 8.4
ASM chart decision box

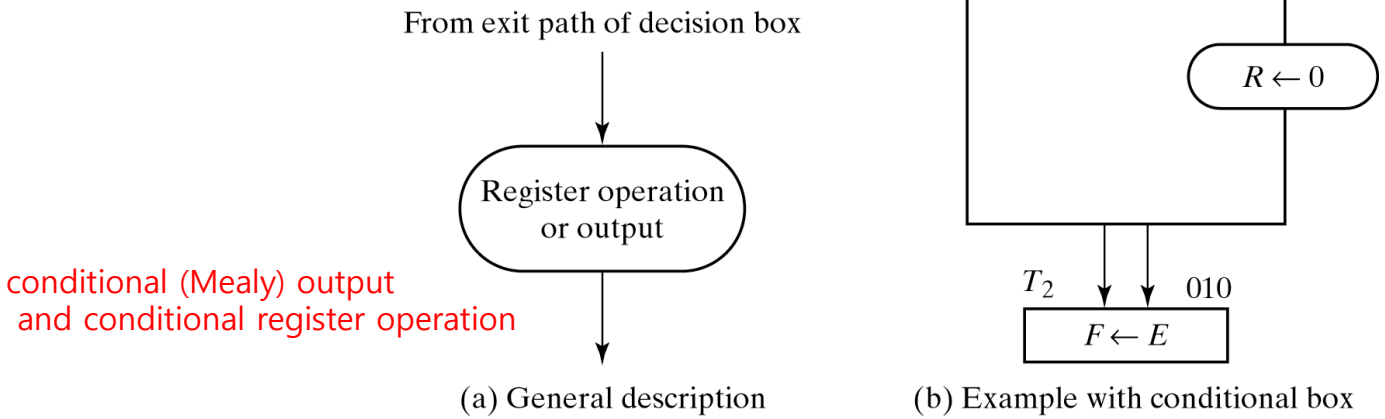


Fig. 8-5 Conditional Box

ASM block

- a structure consisting of one state box,

  all the decision and conditional boxes connected to its exit paths

- each ASM block describes ***the state of the system*** *during one clock pulse interval*

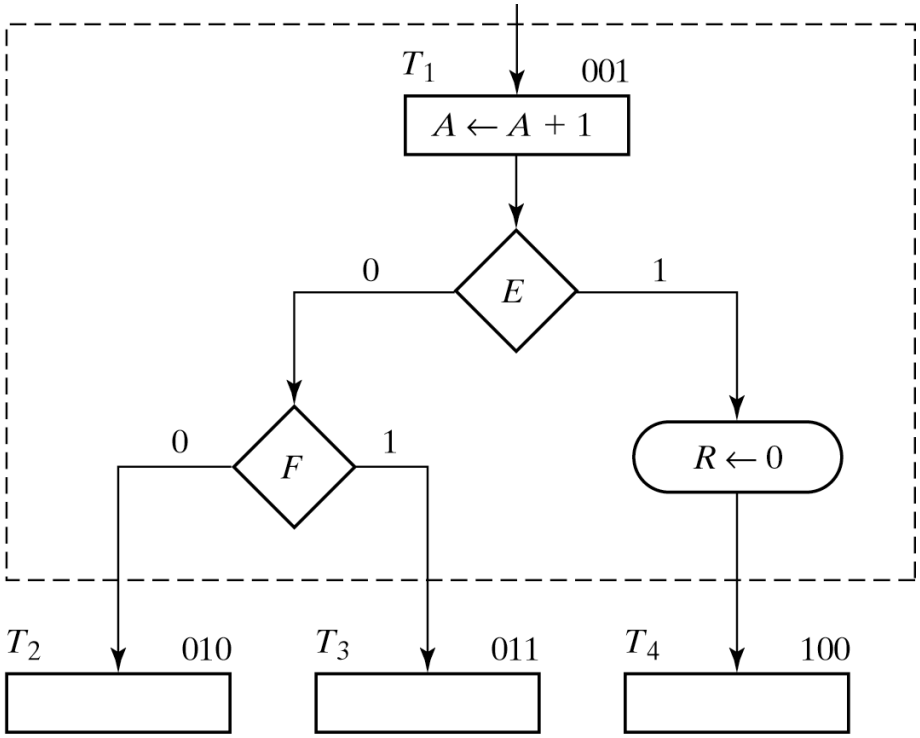


Fig. 8-6 ASM Block

Timing sequence for the example

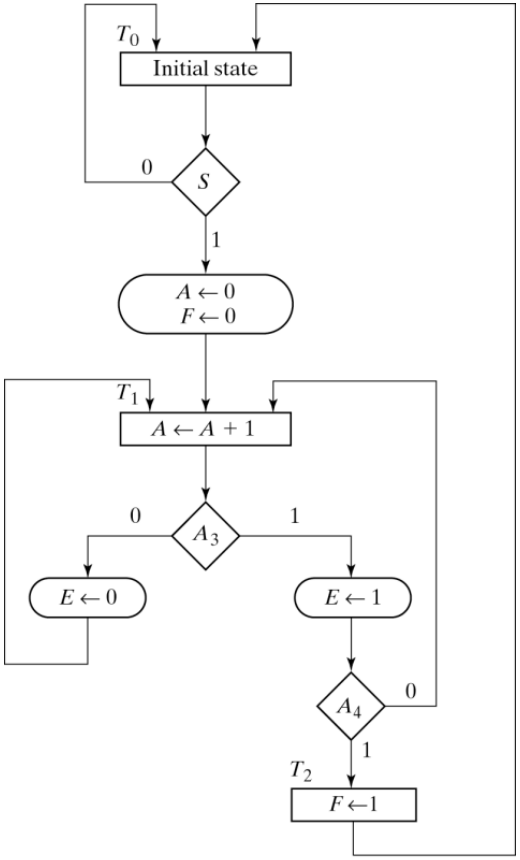


Fig. 8-9 ASM Chart for Design Example

Table 8-2
Sequence of Operations for Design Example

Counter				Flip-Flops			
A_4	A_3	A_2	A_1	E	F	Conditions	State
0	0	0	0	1	0	$A_3 = 0, A_4 = 0$	T_1
							T_2
				1	1		T_0

ASM chart vs. state diagram

- similarity

ASM chart	state diagram
ASM block	state
decision box	input condition lines

=> sometimes convert the chart into a state diagram and then use sequential circuit procedure to design the control logic

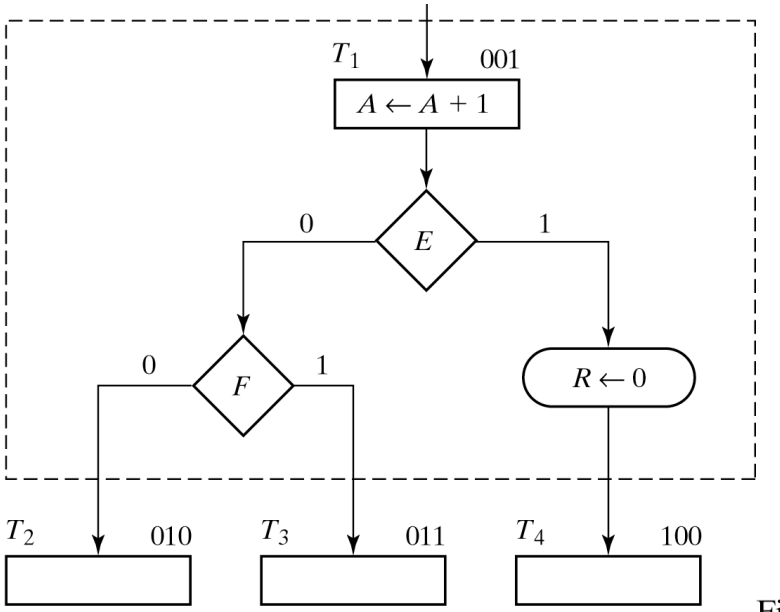


Fig. 8-6 ASM Block

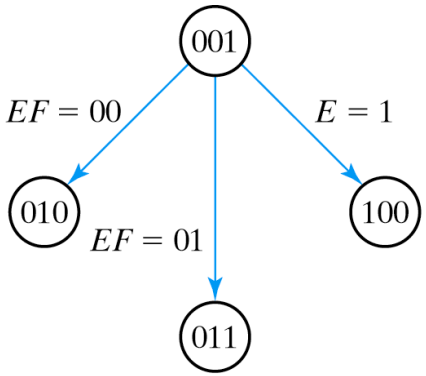


Fig. 8-7 State Diagram Equivalent to the ASM Chart of Fig. 8-6

Timing considerations

- assume that inputs, outputs, and states are all synchronized with the clock
- all the operations that are specified within one block must occur in synchronism

  during the same clock edge while the system moves to the next state

- In Fig. 8-6, at the end of T_1
  - Register A is incremented
  - If E=1, register R is cleared
  - Depending on the values of E and F, control moves to next state.

  i.e. ***data processing and state control occur at the same time.***

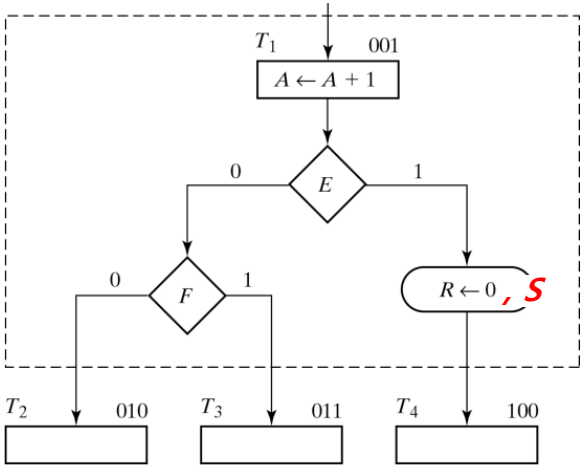


Fig. 8-6 ASM Block

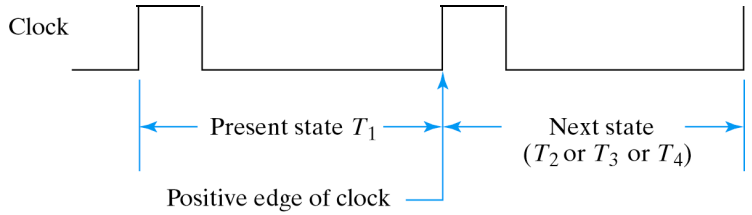


Fig. 8-8 Transition Between States

When inputs (for example E in Fig. 8-6) are not synchronized with the clock?

ASM block - revisited

- a structure consisting of one state box,

  all the decision and conditional boxes connected to its exit paths

- each ASM block describes ***the state of the system*** *during one clock pulse interval*

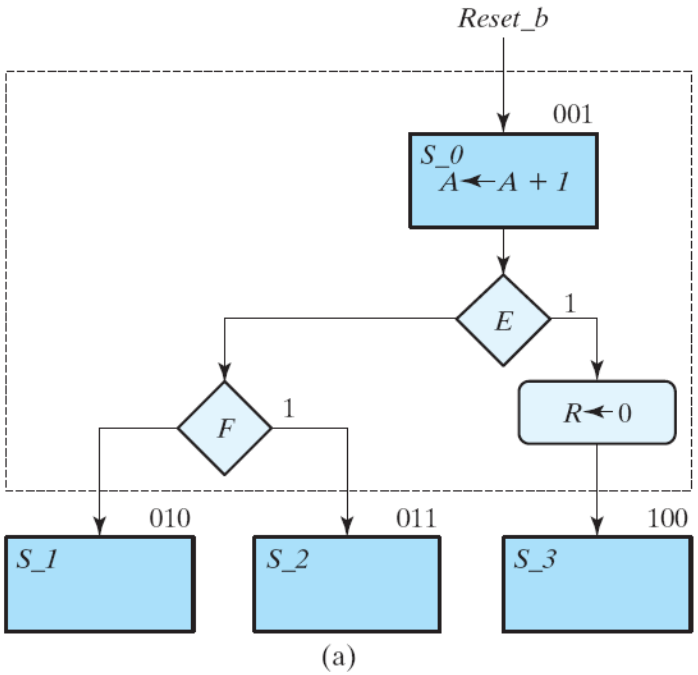
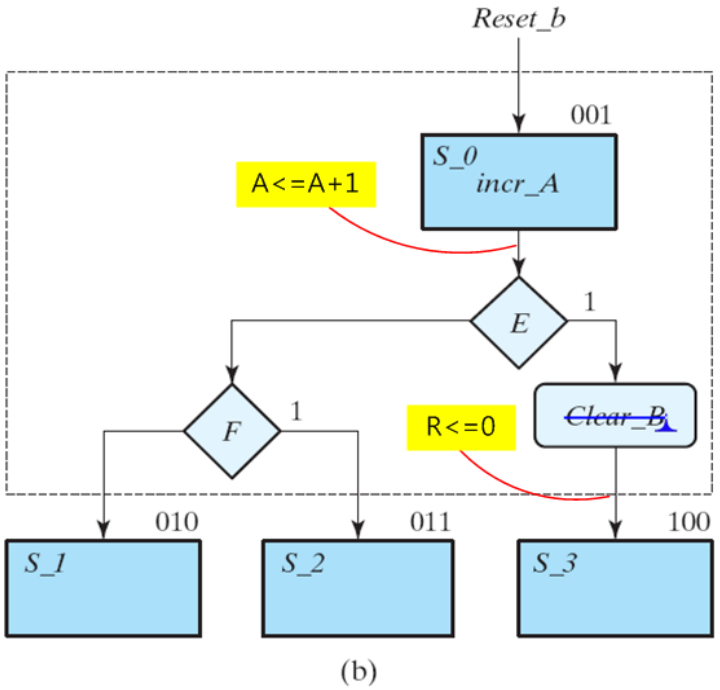


FIGURE 8.6
ASM block



ASMD chart. For more, refer to Fig. 8.9

8.5 Design example – ASM approach

Design a digital system with

- 2 FF's (E and F)
- 1 4-bits binary counter A ($A_4A_3A_2A_1$)
- 1 start input, S

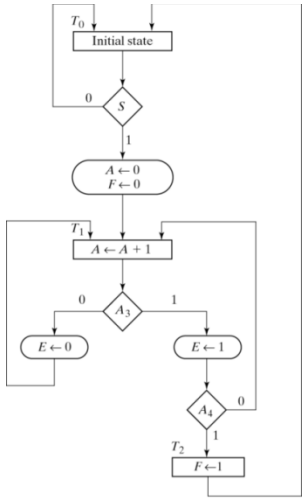
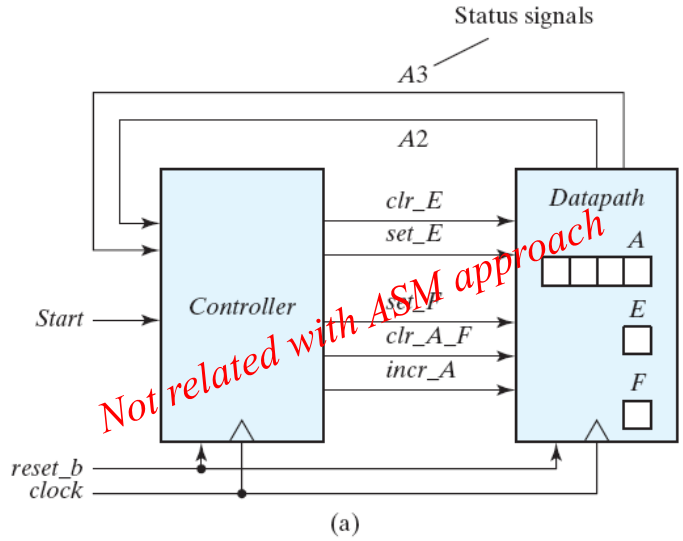
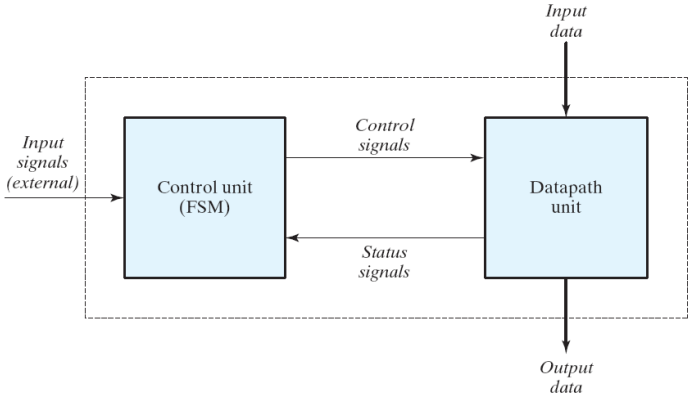


Fig. 8-9 ASM Chart for Design Example



Not related with ASM approach



Design example – cont'd

Design a digital system with

2 FF's (E and F)

1 4-bits binary counter A ($A_4A_3A_2A_1$)

1 start input, S

Operation

"S=1" initiate the system operation by clearing A and F, then

A counts up until the operation stop

A_3 and A_4 determine the operation sequence

If A_3=0, E ← 0 and **keeps counting**

else E ← 1 then

If A_4=0, **keeps counting**

else **F ← 1 on the next CP** and **stops counting**

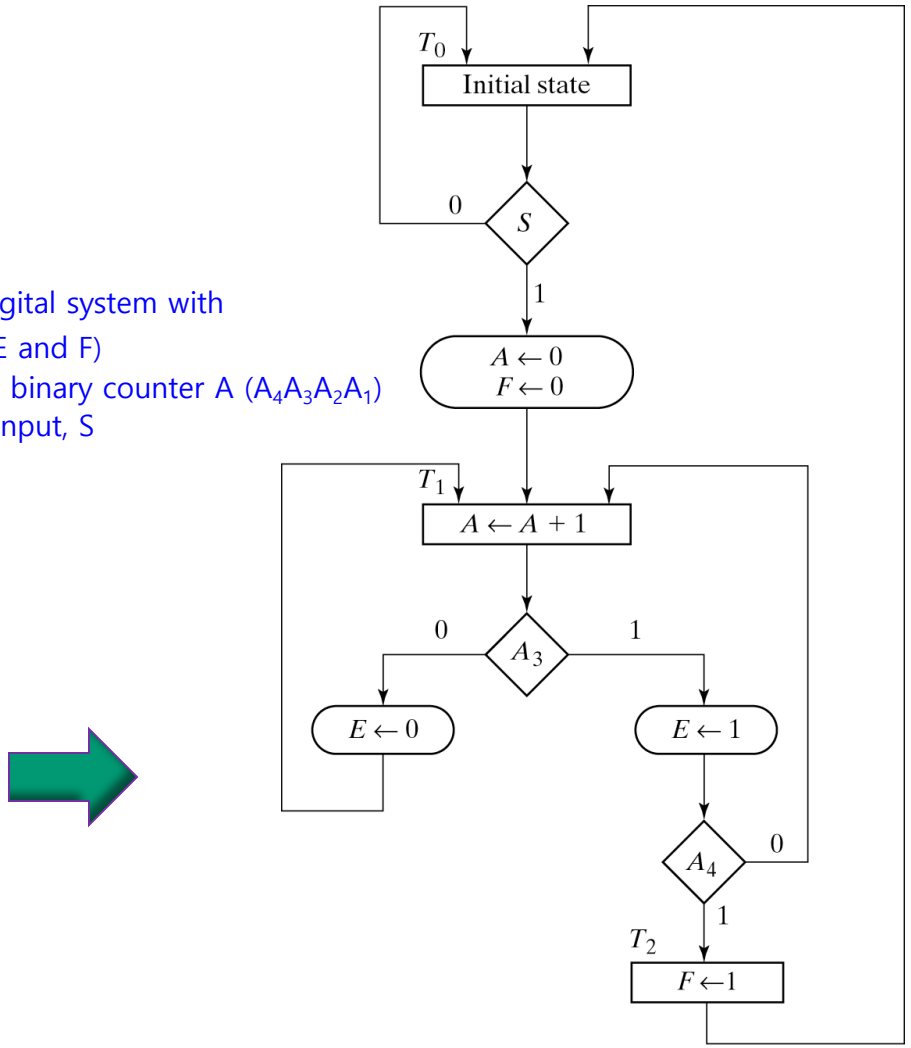
Then if S=0, the system remains in the **initial state**, else the operation cycle repeats by clearing A and F

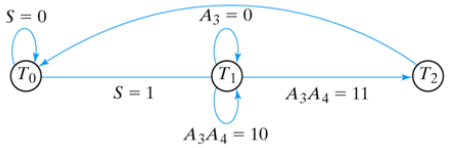
States

initial state	: T_0
counting	: T_1
F ← 1 and stop	: T_2

ASM chart for the example

Design a digital system with
- 2 FF's (E and F)
- 1 4-bits binary counter A ($A_4A_3A_2A_1$)
- 1 start input, S





State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

Register transfer operations

Operation

"S=1" initiate the operation by clearing A and F, then A counts up until the operation stop

A_3 and A_4 determine the operation sequence

If A_3=0, E ← 0 and **keeps counting**

else E ← 1 then

If A_4=0, **keeps counting**

else **F ← 1 on the next CP** and **stops counting**

Then if S=0, the system remains in the **initial state**, else the operation cycle repeats by clearing A and F

States

initial state	: T_0
Counting	: T_1
F ← 1 and stop	: T_2

Fig. 8-9 ASM Chart for Design Example

Timing sequence for the example

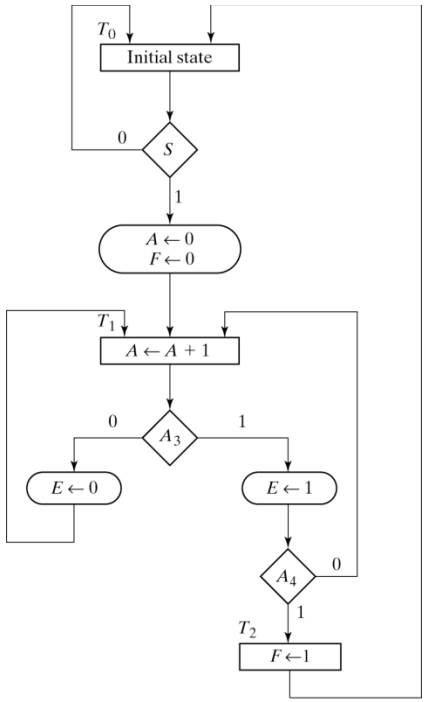


Fig. 8-9 ASM Chart for Design Example

Table 8-2
Sequence of Operations for Design Example

Counter				Flip-Flops			
A_4	A_3	A_2	A_1	E	F	Conditions	State
0	0	0	0	1	0	$A_3 = 0, A_4 = 0$	T_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	$A_3 = 1, A_4 = 0$	
0	1	0	1	1	0		
0	1	1	0	1	0		
0	1	1	1	1	0		
1	0	0	0	1	0	$A_3 = 0, A_4 = 1$	
1	0	0	1	0	0		
1	0	1	0	0	0		
1	0	1	1	0	0		
1	1	0	0	0	0	$A_3 = 1, A_4 = 1$	
1	1	0	1	1	0		T_2
1	1	0	1	1	1		T_0

Design of datapath and control logic

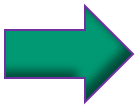
control logic controls the state transition (expressed in the ***decision box***)

datapath section covers ***state box*** and ***conditional boxes***

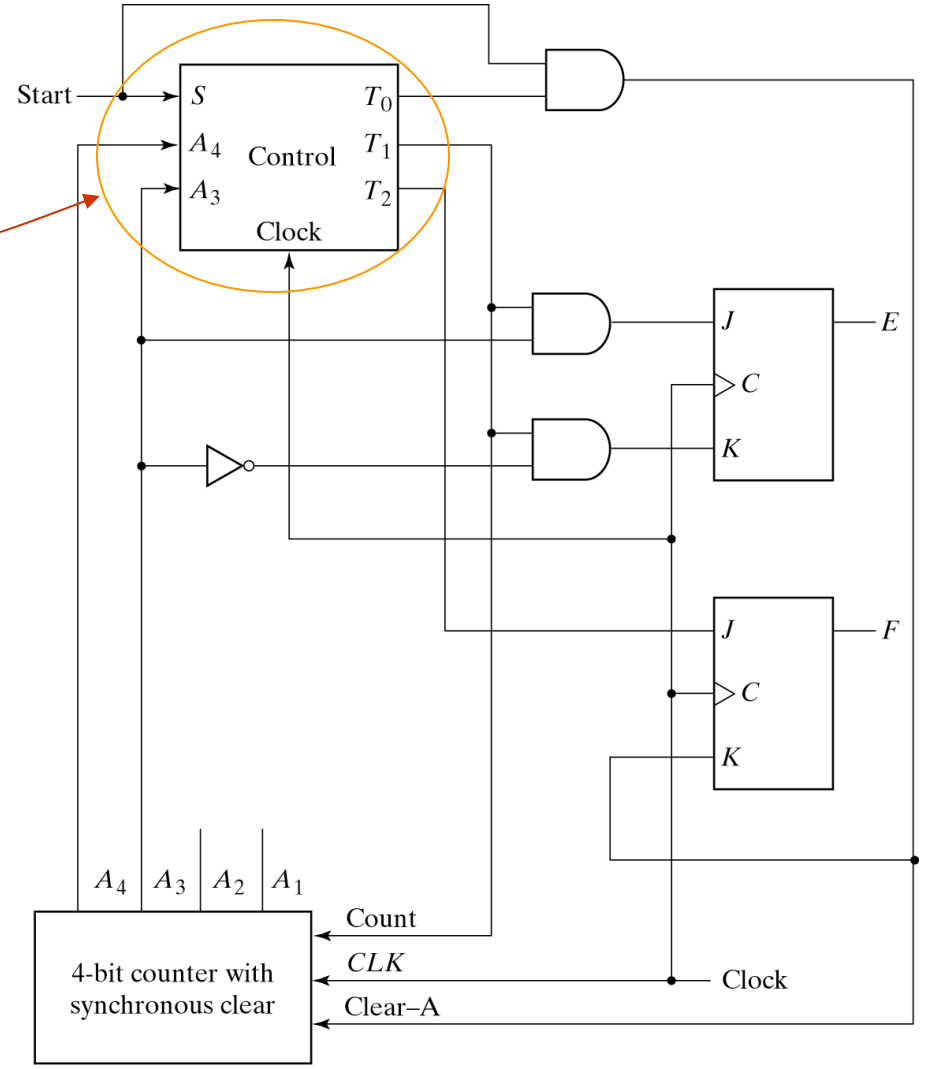
* The circuit in the *control part* generates the signals for all the operations specified in the present block *prior to the arrival of the next clock pulse*.

* *The next clock transition executes all the operations* in one ASM block, including the FF's in the controller that determine the next state

ASMD chart show this explicitly
(refer to p.45)



Datapath



control logic

Datapath?
2 FF's
one 4 bits counter
and gates

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

Fig. 8-10 Datapath for Design Example

Design implementation thru RTL

To represent a digital system in the RTL, specify
the registers in the system,
the operation performed,
and the *control sequence.*

ASM chart contains **the register operation and the control information**

It's sometimes convenient to separate

***the control logic* and *the register operation* for the datapath in the ASM chart.**

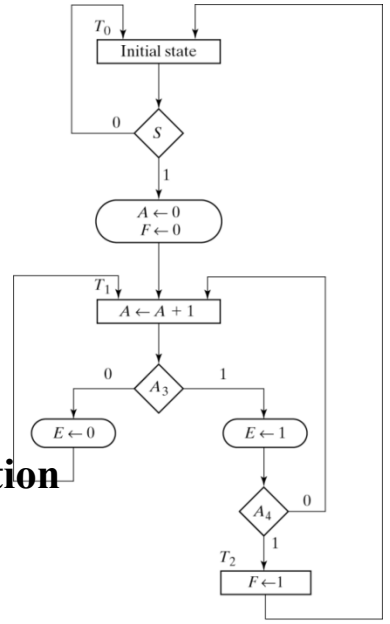
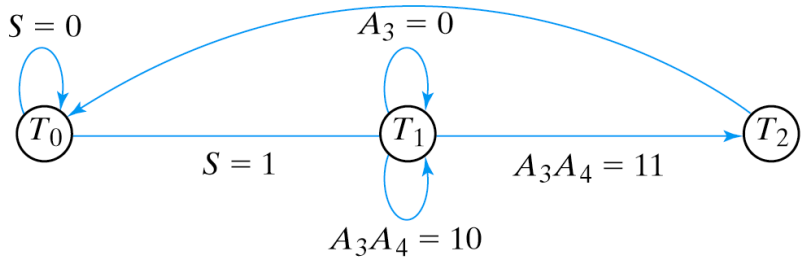


Fig. 8-9 ASM Chart for Design Example



(a) State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

if $(A_3 = 1)$ then $E \leftarrow 1$

if $(A_3 = 0)$ then $E \leftarrow 0$

T_2: $F \leftarrow 1$

(b) Register transfer operations

Fig. 8-11 Register Transfer Level Description of Design Example

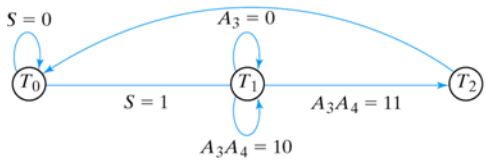
State table for control logic

Convert the state diagram in Fig. 8-11 to the state table to design the controller circuit

Table 8-3
State Table for Control of Fig. 8-10

Present-State Symbol	Present State		Inputs			Next State		Outputs			FF inputs					
	G_1	G_0	S	A_3	A_4	G_1	G_0	T_0	T_1	T_2	J_1	K_1	J_0	K_0 /	D_1	D_0
T_0	0	0	0	X	X	0	0	1	0	0						
T_0	0	0	1	X	X	0	1	1	0	0						
T_1	0	1	X	0	X	0	1	0	1	0						
T_1	0	1	X	1	0	0	1	0	1	0						
T_1	0	1	X	1	1	1	1	0	1	0						
T_2	1	1	X	X	X	0	0	0	0	1						

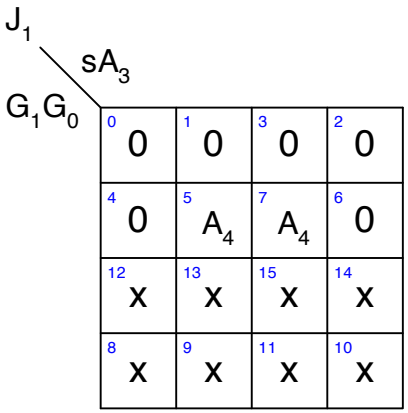
1. one row for each possible transition between states
2. # of rows = # of paths between the states in the ASM chart or state diagram
3. large # of don't care conditions
4. # of inputs = 5



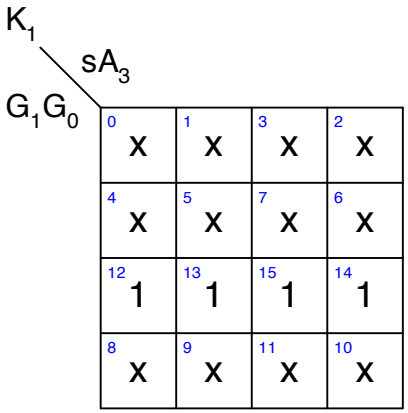
Conventional design method I

1. Choose JK

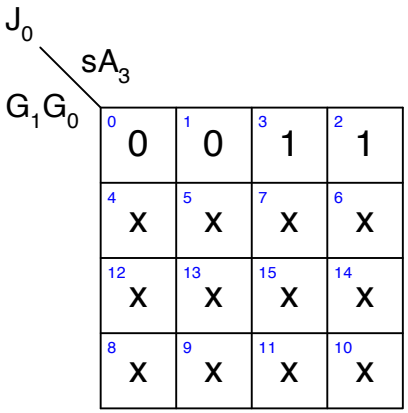
   => 5 inputs and 7 outputs ; complex



$J_1 = G_0 A_3 A_4$



$K_1 = 1$



$J_0 = s$

$K_0 = G_1$

$T_0 = G_0'$

$T_1 = G_1'G_0$

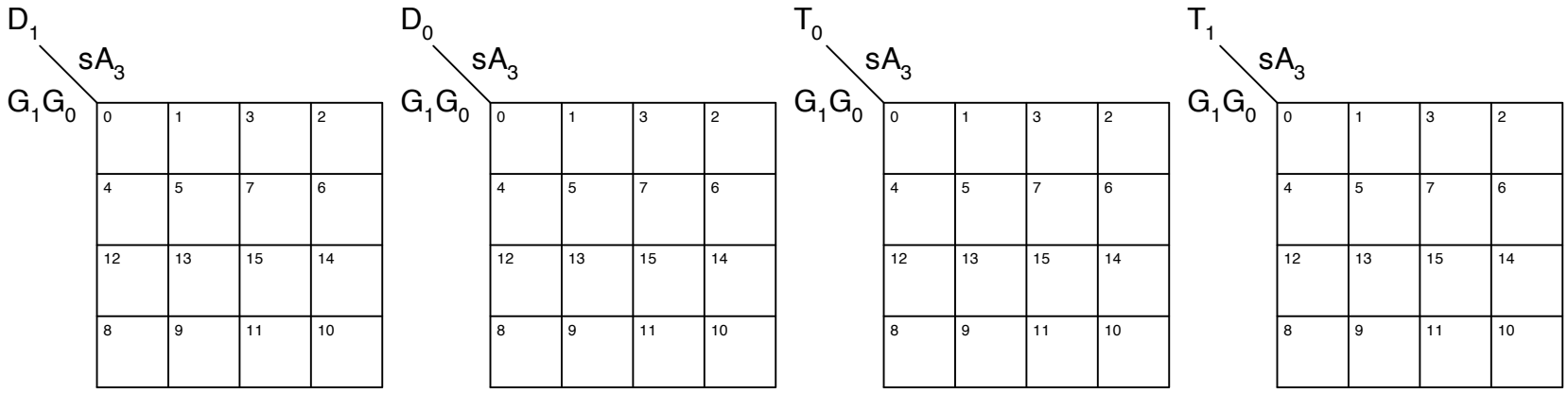
$T_2 = G_1$

* 7 maps --> cumbersome and difficult to manage !!

Conventional design method II

2. Choose D

=> 5 inputs and 5 outputs ; still complex



$D_1 = G_1' G_0 A_3 A_4 = T_1 A_3 A_4$

$D_0 = G_0' s + G_1' G_0 = T_0 s + T_1$

$T_0 = G_0'$

$T_1 = G_1' G_0$

$T_2 = G_1$

Control logic obtained

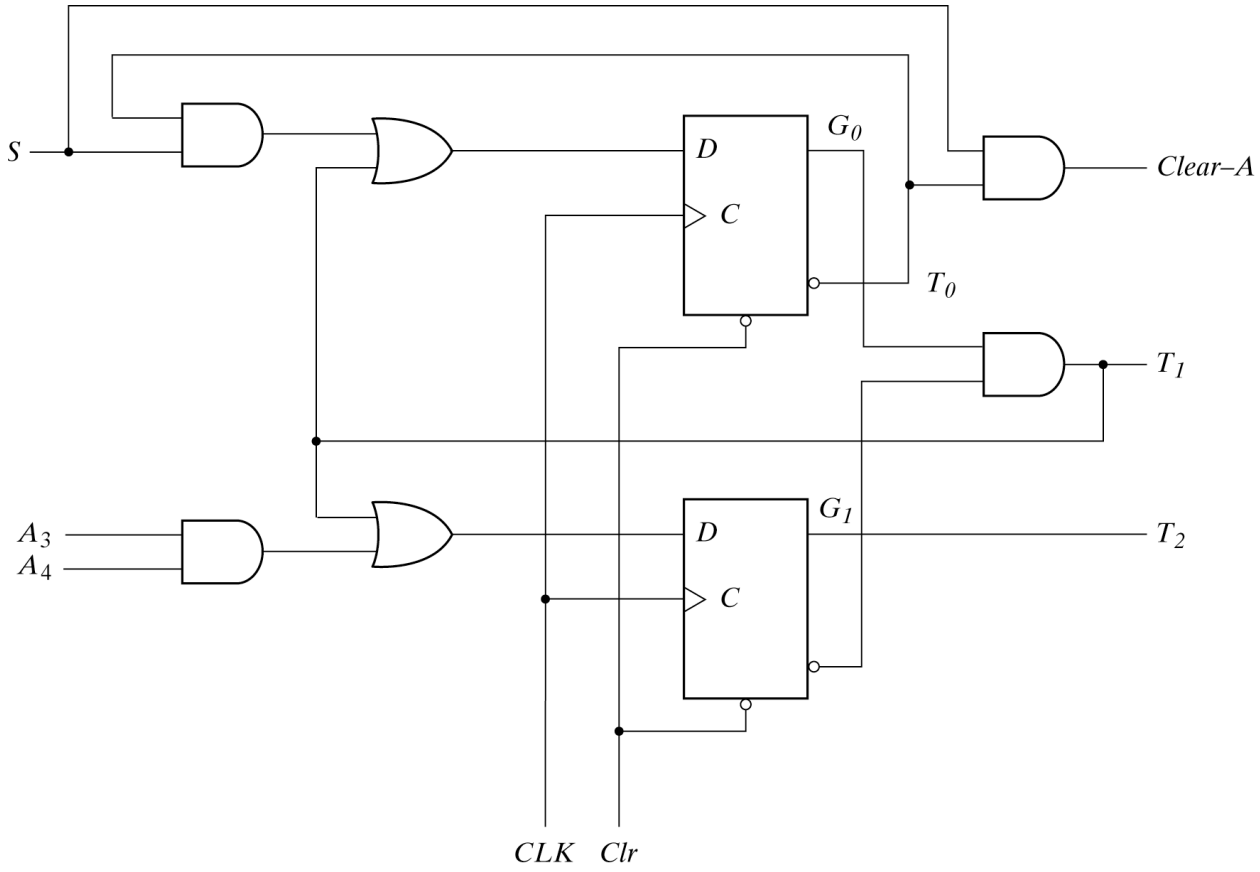
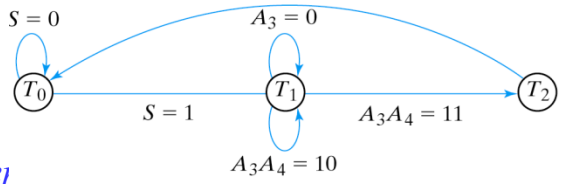


Fig. 8-12 Logic Diagram of Control

8.6 HDL description of ASM example

Structural
RTL *(1. Define I/O, 2. Control sequence, 3. Register transfer oper*
and ***Behavioral*** *description*



(a) State diagram for control

Fig. 8-11 Register Transfer Level Description

```
//HDL Example 8-2,  RTL description of design example (Fig.8-9)
module Example_RTL (S,CLK,Clr,E,F,A);  // define i/o
  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 -- control sequence
//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; else nstate = T0;
       T1: if(A[3] & A[4]) nstate = T2; else nstate = T1;
       T2: nstate = T0;
     endcase
```

```
//Register transfer operations, See list of operations Fig.8-11(b)
  always @(posedge CLK)  // nonblocking stmt for synch
    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
```

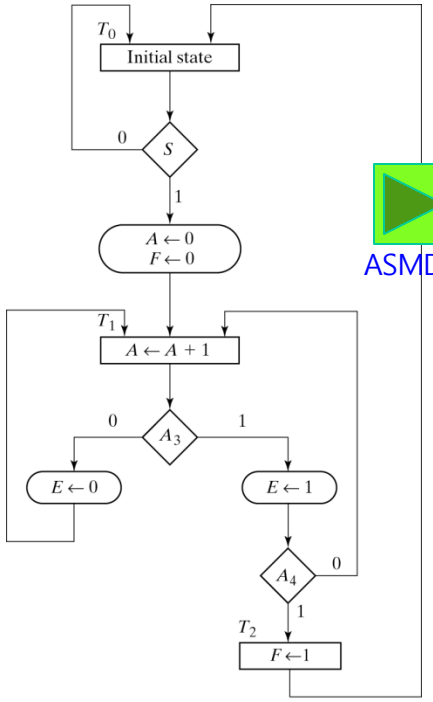


Fig. 8-9 ASM Chart for Design Example

ASMD 설계

전체개요

Check errors!

8.5 Design example – new approach (ASMD chart)

Design a digital system with

2 FF's (E and F)

1 4-bits binary counter A ($A_3A_2A_1A_0$) * *was ($A_4A_3A_2A_1$) in ASM chart*

1 start input, Start

Operation

"Start=1" initiate the system operation by clearing A and F, then

A counts up until the operation stop

A_2 and A_3 determine the operation sequence

If A_2=0, E ← 0 and **keeps counting**

else E ← 1 then

if A_3=0, **keeps counting**

else **F ← 1 on the next CP** and **stops counting**

Then if Start=0, the system remains in the **initial state**, else the cycle repeats by clearing A and F

States

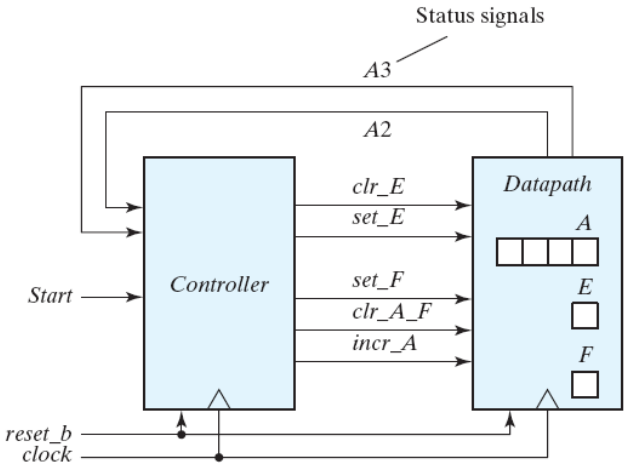
initial state : *S_idle*

counting : *S_1*

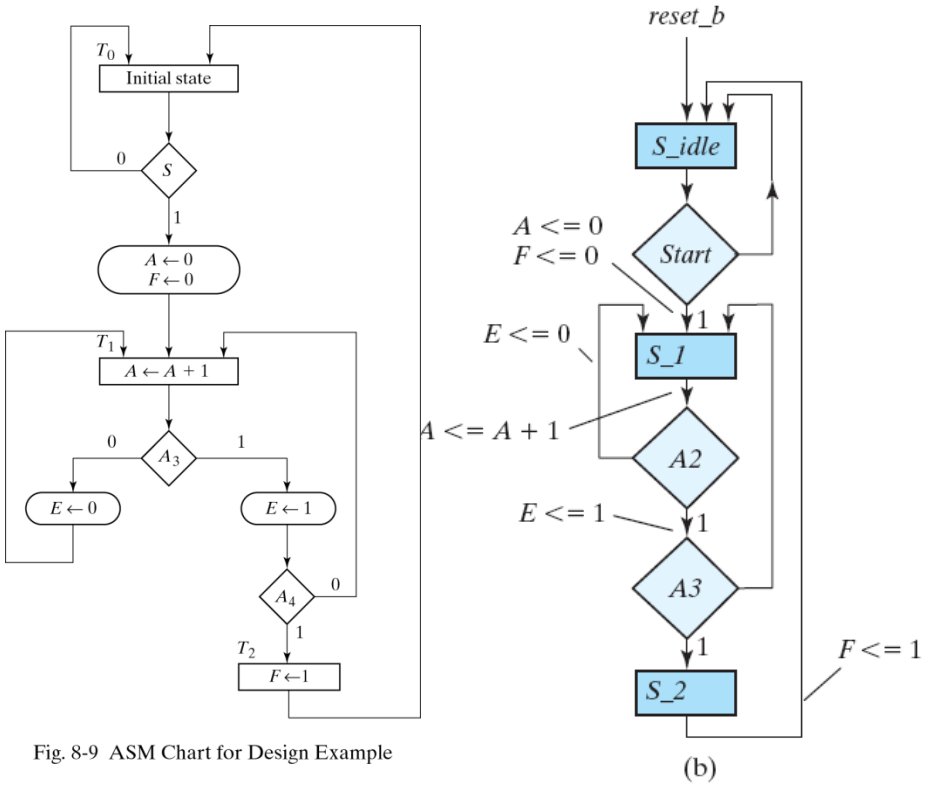
F ← 1 and stop : *S_2*

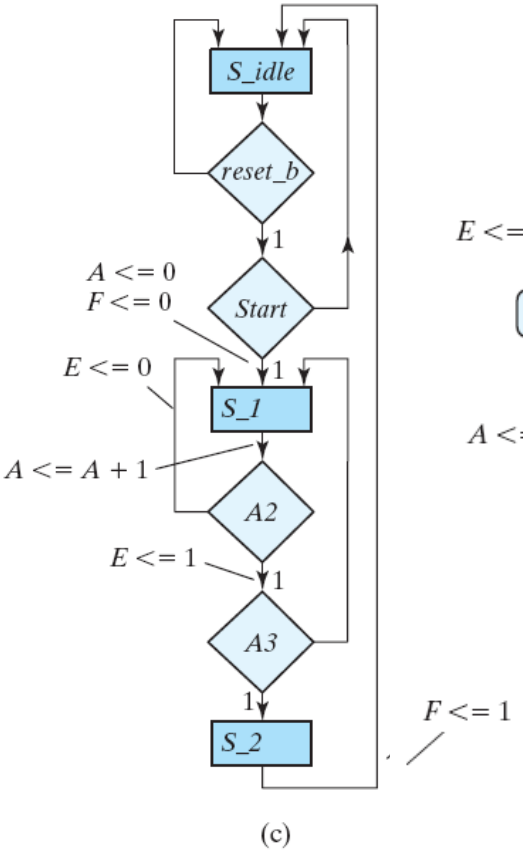
Explain POR!

FIGURE 8.9
(a) Block diagram for design example
(b) ASMD chart for controller state transitions, asynchronous reset
(c) ASMD chart for controller state transitions, synchronous reset
(d) ASMD chart for a completely specified controller, asynchronous reset



ment
condition





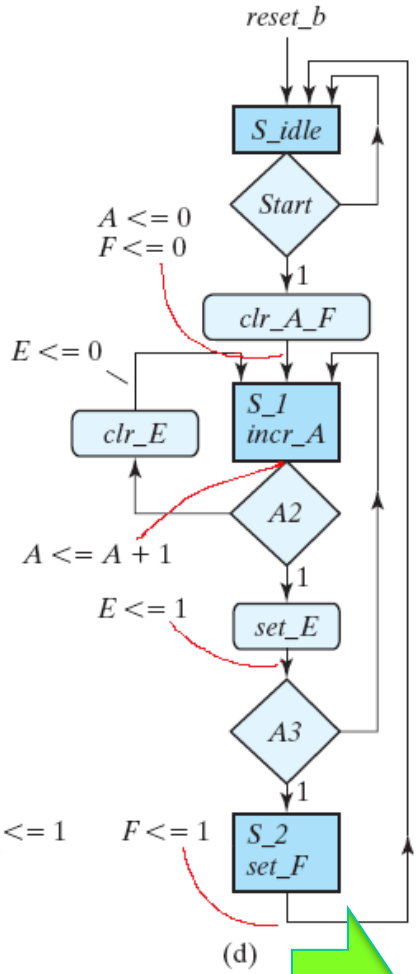


Fig. 8-9 ASM Chart for Design Example

HDL code

Timing sequence for the example

Table 8.3
Sequence of Operations for Design Example

Counter				Flip-Flops			
A_3	A_2	A_1	A_0	E	F	Conditions	State
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	$A_2 = 1, A_3 = 1$	
1	1	0	1	1	0		S_2
1	1	0	1	1	1		S_idle

Controller and datapath hardware design

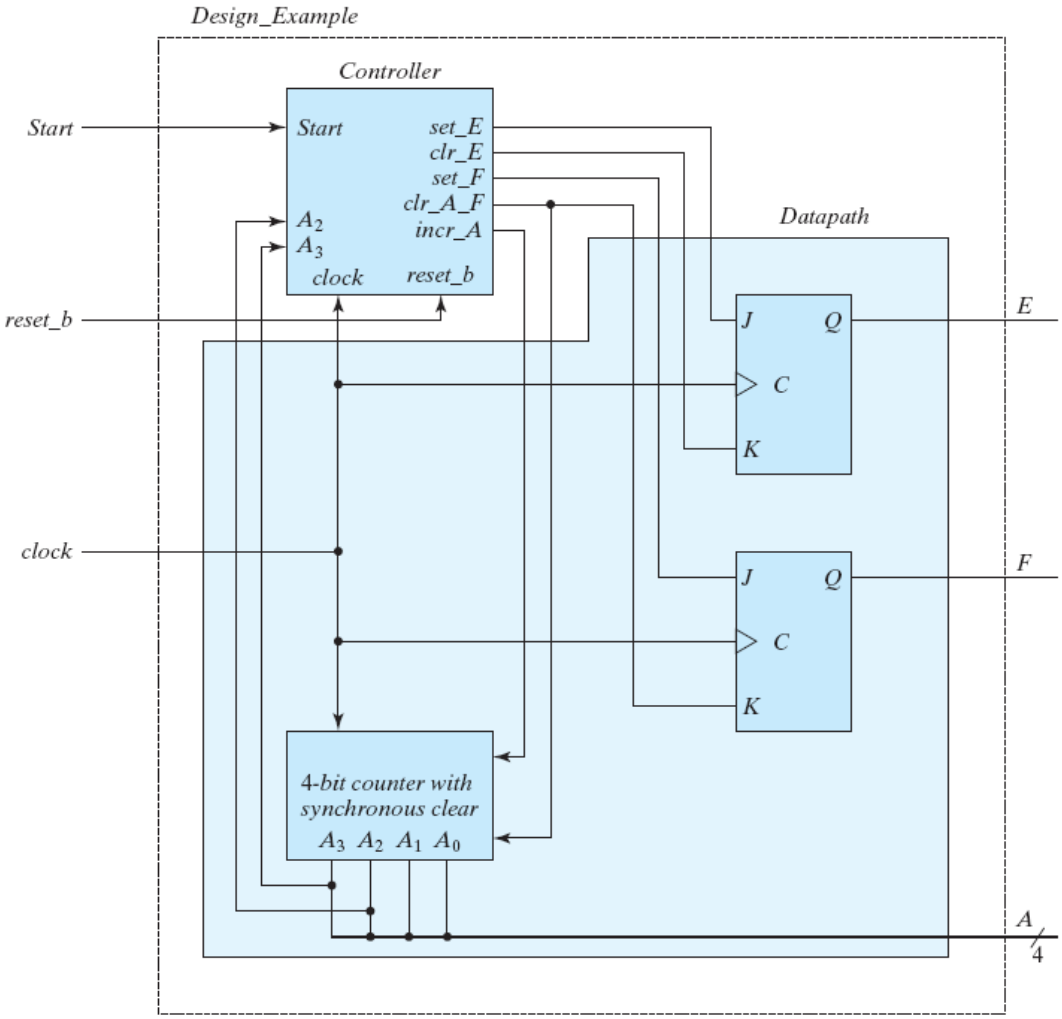
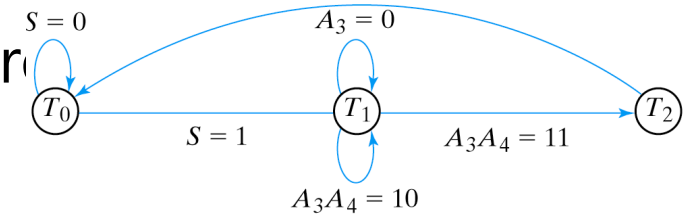


FIGURE 8.10
Datapath and controller for design example

HDL 설계

Register transfer r



(a) State diagram for control

T_0: if $(S = 1)$ then $A \leftarrow 0, F \leftarrow 0$

T_1: $A \leftarrow A + 1$

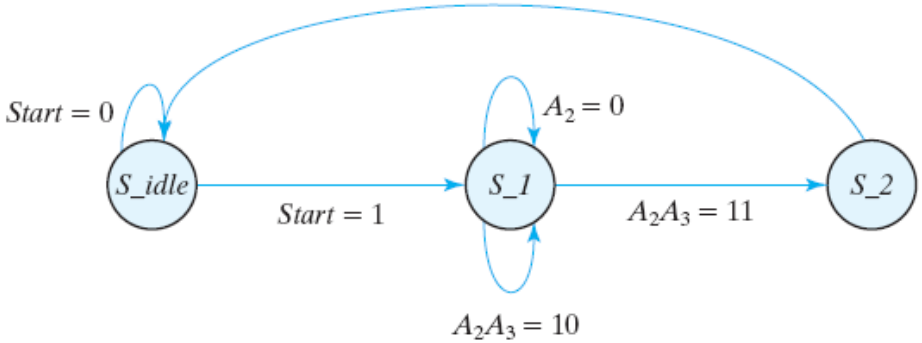
if $(A_3 = 1)$ then E $\leftarrow 1$

if $(A_3 = 0)$ then E $\leftarrow 0$

T_2: $F \leftarrow 1$

(a) Register transfer operations

Fig. 8-11 Register Transfer Level Description of Design Example



(a)

S_idle ⟶ *S_1, clr_A_F:* $A \leftarrow 0, F \leftarrow 0$

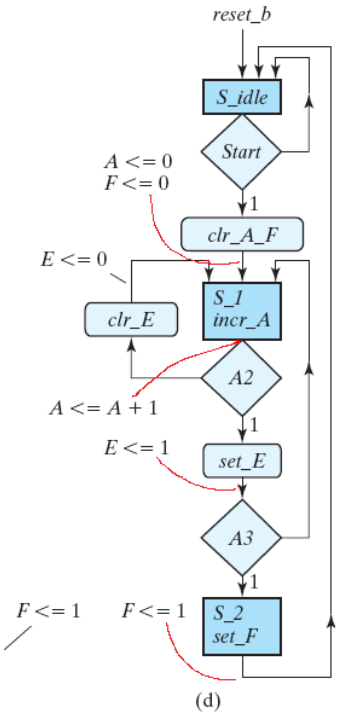
S_1 ⟶ *S_1, incr_A:* $A \leftarrow A + 1$

if $(A_2 = 1)$ *then set_E:* $E \leftarrow 1$

if $(A_2 = 0)$ *then clr_E:* $E \leftarrow 0$

S_2 ⟶ *S_idle, set_F:* $F \leftarrow 1$

(b)



(d)

FIGURE 8.11
Register transfer-level description of design example

State table

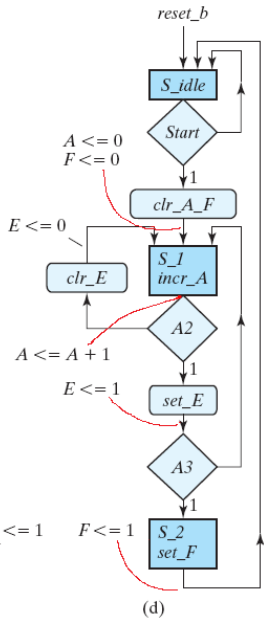


Table 8.4
State Table for the Controller of Fig. 8.10

Present-State Symbol	Present State G_1	Present State G_0	Inputs *Start*	Inputs A_2	Inputs A_3	Next State G_1	Next State G_0	Outputs *set_E*	Outputs *clr_E*	Outputs *set_F*	Outputs *clr_*	Outputs *incr*
S_idle	0	0	0	X	X	0	0	0	0	0	0	0
S_idle	0	0	1	X	X	0	1	0	0	0	1	0
S_1	0	1	X	0	X	0	1	0	1	0	0	1
S_1	0	1	X	1	0	0	1	1	0	0	0	1
S_1	0	1	X	1	1	1	1	1	0	0	0	1
S_2	1	1	X	X	X	0	0	0	0	1	0	0

$$D_{G1} = S\_1A_2A_3$$
$$D_{G0} = Start\ S\_idle + S\_1$$

$$set\_E = S\_1A_2$$
$$clr\_E = S\_1A_2'$$
$$set\_F = S\_2$$
$$clr\_A\_F = Start\ S\_idle$$
$$incr\_A = S\_1$$

Control logic

Instead of using maps to simply the input equations, obtain them directly from the state table **by inspection**

$$D_{G1} = S\_1A_2A_3$$
$$D_{G0} = Start\ S\_idle + S\_1$$
$$set\_E = S\_1A_2$$
$$clr\_E = S\_1A_2'$$
$$set\_F = S\_2$$
$$clr\_A\_F = Start\ S\_idle$$
$$incr\_A = S\_1$$

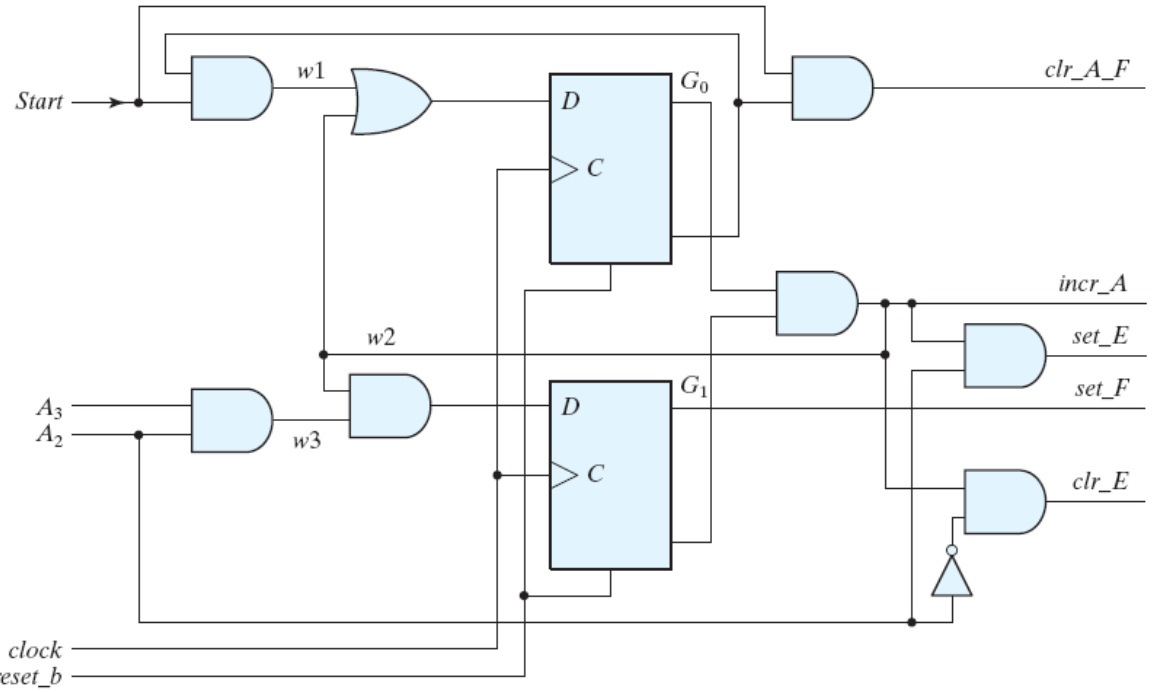
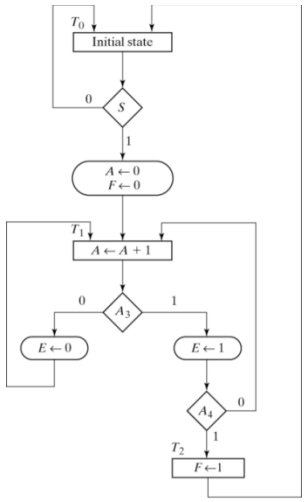


FIGURE 8.12
Logic diagram of the control unit for Fig. 8.10

8.5 Design example – In summary

Design a digital system with

- 2 FF's (E and F)
- 1 4-bits binary counter A ($A_4A_3A_2A_1$)
- 1 start input, S



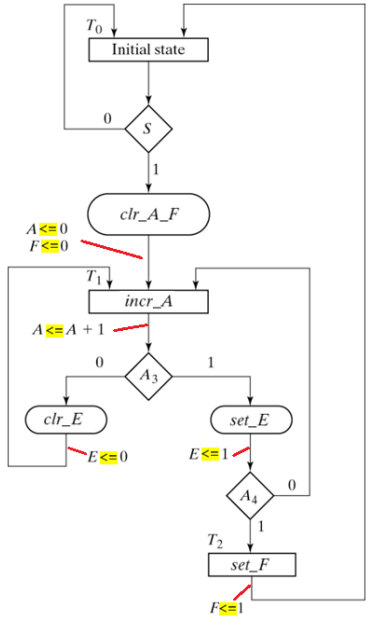
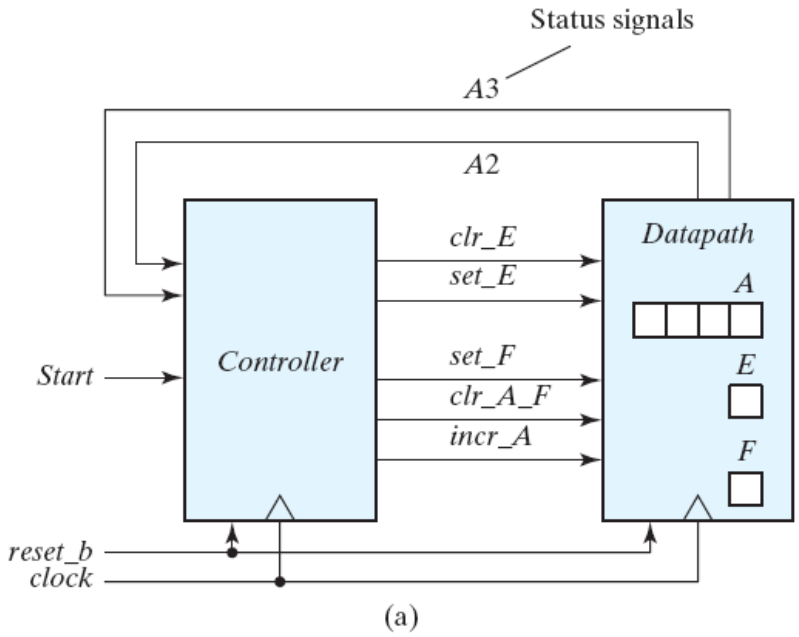
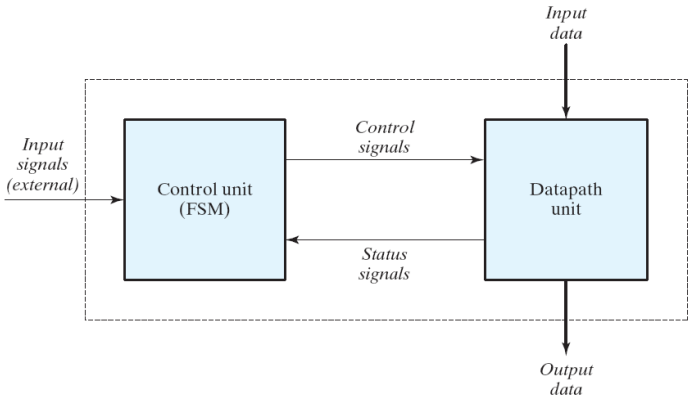


Fig. 8-9 ASM Chart for Design Example



(a)



종래방식 설계

8.6 HDL description of ASMD example

***RTL** description*

(1. Define I/O, 2. Control sequence, 3. Register transfer operations and outputs)

```
//HDL Example 8-2,  RTL description of design example (Fig.8-11)

module Example_RTL (A, E, F, Start, clock, reset_b);  // define i/o
  input Start, clock, reset_b;
  output [3:0] A;
  output E,F;

//Specify system registers                          *S_idle is renamed as S_0
  reg [3:0] A;                 //A register
  reg E, F;                     //E and F flip-flops
  reg [1:0] pstate, nstate;   //control register
  reg set_E, clr_E, set_F, clr_A_F, incr_A;

//Encode the states
  parameter S_0 = 2'b00, S_1 = 2'b01, S_2 = 2'b11;

//State transition for control logic -- control sequence
  always @(posedge clock or negedge reset_b)
    if (~reset_b) pstate <= S_0;        //initial state
    else          pstate <= nstate;     //clocked operations

//Code next_state logic directly from ASMD chart. Fig. 8-9(d)
  always @ (Start or A or pstate)
    case (pstate)
      S_0: if(Start) nstate <= S_1; else nstate <= S_0;
      S_1: if(A[2] & A[3]) nstate <= S_2; else nstate <= S_1;
      S_2: nstate <= S_0;
    endcase
```

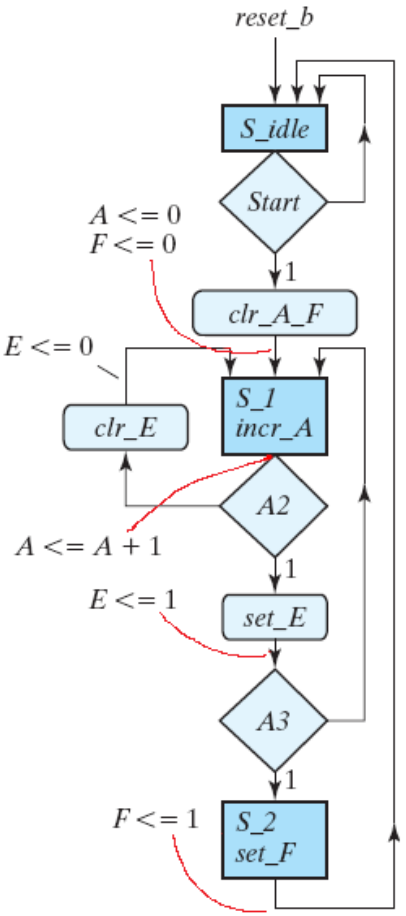


FIGURE 8.9 (d)

8.6 HDL description of ASMD example – *cont'd*

```
//Code output logic directly from ASMD chart. Fig. 8-9(d)
  always @ (Start or A or pstate) begin
    set_E    <= 0;
    clr_E    <= 0;
    set_F    <= 0;
    clr_A_F  <= 0;
    incr_A   <= 0;

    case (pstate)
      S_0: if(Start) clr_A_F <= 1;
      S_1: begin
           incr_A <= 1;
           if(A[2]) set_E <= 1;
           else clr_E <= 1; end
      S_2: set_F <= 1;
    endcase
  end
//Code register transfer operations, from Fig.8-9(d) -- register operation
  always @(posedge clock) begin
    if(set_E)     E <= 1;
    if(clr_E)     E <= 0;
    if(set_F)     F <= 1;
    if(clr_A_F) begin A <= 0; F <= 0; end
    if(incr_A) A <= A + 1;
  end

endmodule
```

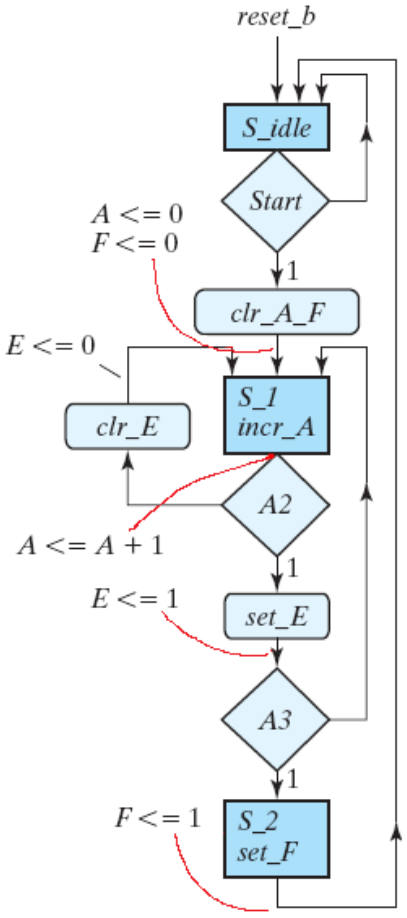
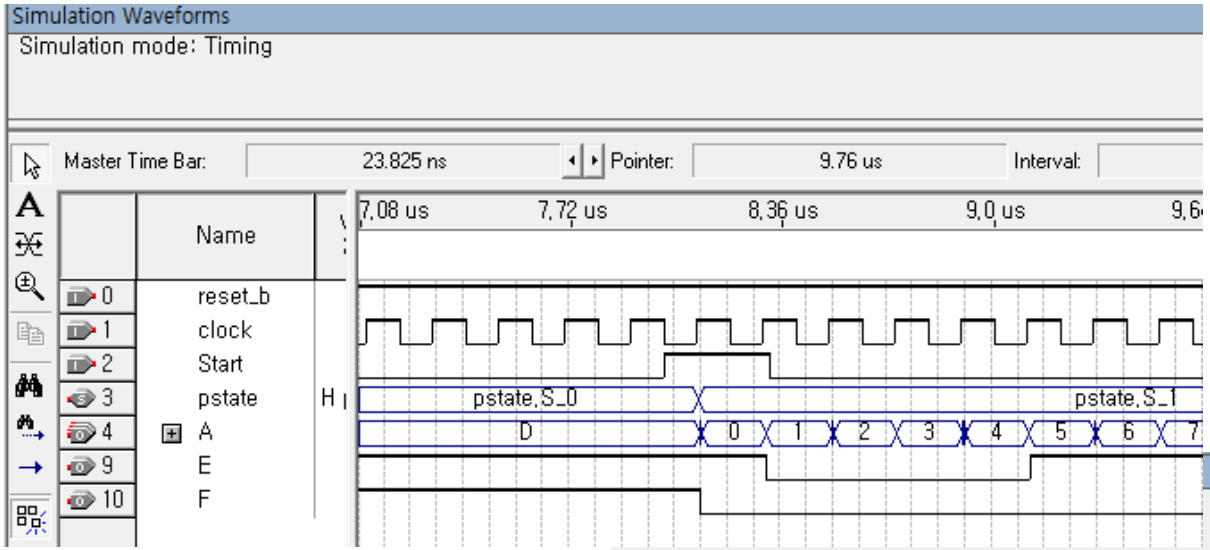


FIGURE 8.9 (d)

ASM 설계

8.6 Simulation output



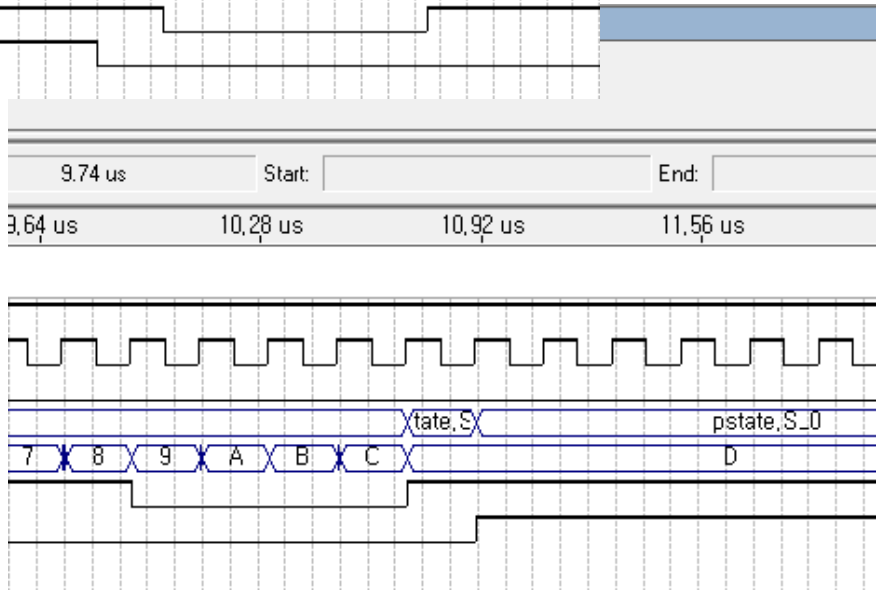


Table 8.3
Sequence of Operations for Design Example

Counter				Flip-Flops		
A_3	A_2	A_1	A_0	E	F	State
0	0	0	0	1	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	
0	1	0	1	1	0	
0	1	1	0	1	0	
0	1	1	1	1	0	
1	0	0	0	1	0	
1	0	0	1	0	0	
1	0	1	0	0	0	
1	0	1	1	0	0	
1	1	0	0	0	0	
1	1	0	1	1	0	*S_2*
1	1	0	1	1	1	*S_idle*

8.7 Sequential Binary Multiplier

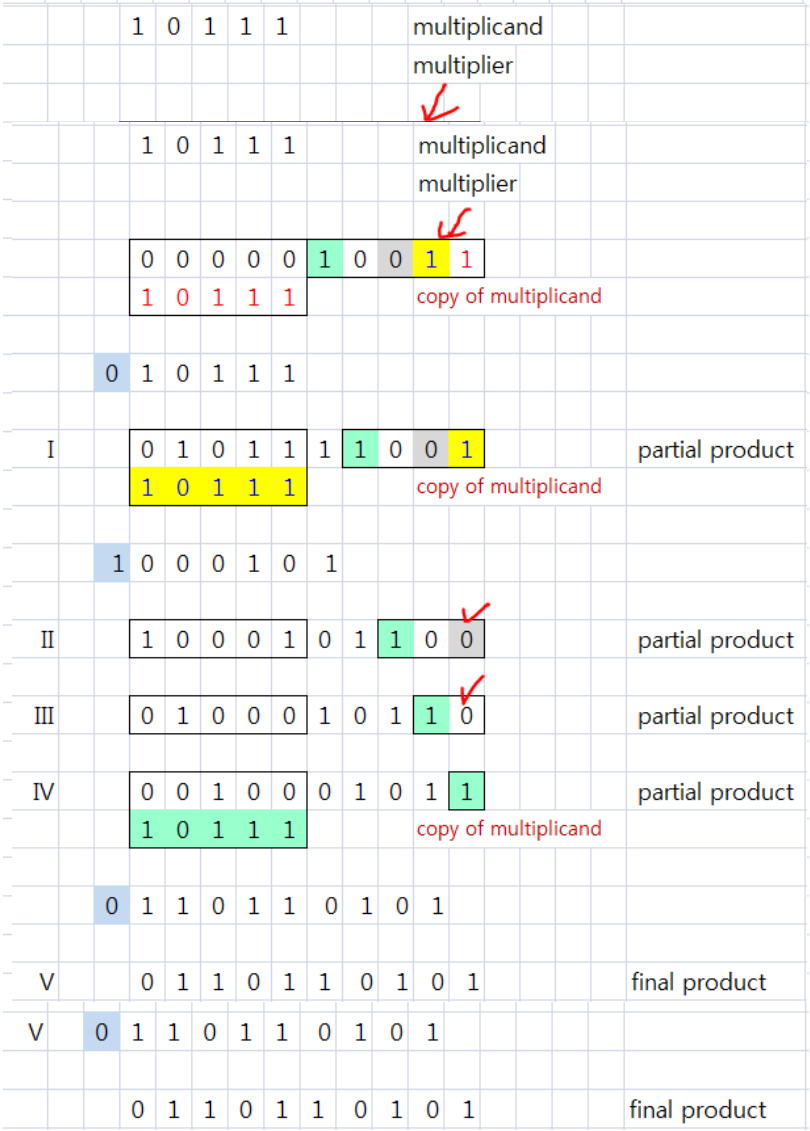
Multiplication of two unsigned binary numbers

* *paper and pencil method for reference*

23	*10111*	*multiplicand*
19	*10011*	*multiplier*
	10111	
	10111	
	00000	
	00000	
	10111	
437	*110110101*	*product*

Binary Multiplication

23	*10111*	*multiplicand*
19	*10011*	*multiplier*
	10111	
	10111	
	00000	
	00000	
	10111	
437	*110110101*	*product*



Design an efficient sequential multiplier through

1. Sum only two binary numbers
   and accumulate the partial products in a register
2. Shift the partial product to the right
3. when the corresponding bit of the multiplier is 0,
   neglect addition

Sequential Binary Multiplier (multiplication of two unsigned binary numbers)

Design an efficient sequential multiplier through

1. Sum only two binary numbers and accumulate the partial products in a register
2. Shift the partial product to the right
3. when the corresponding bit of the multiplier is 0, neglect addition

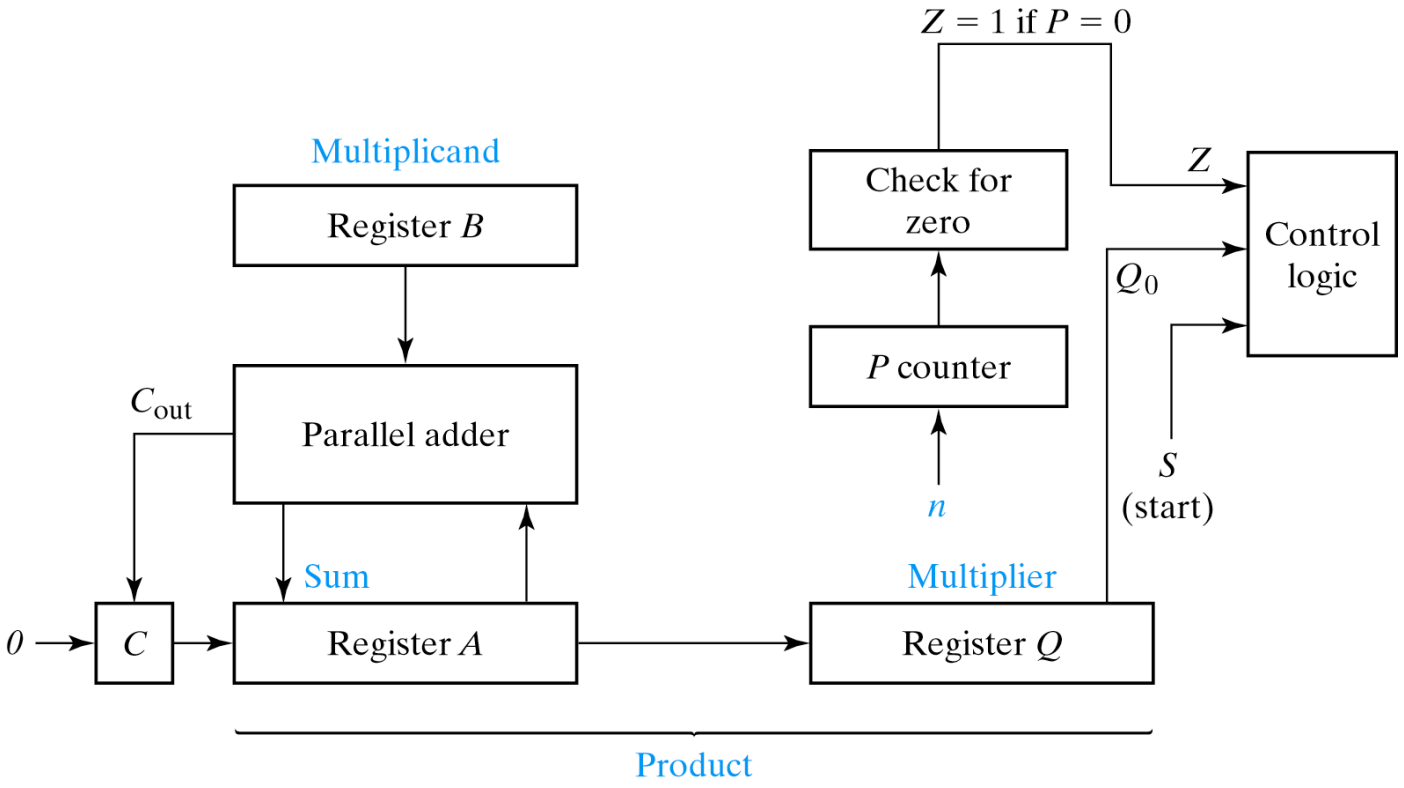


Fig. 8-13 Block Diagram of Binary Multiplier

ASM chart for the example

** paper and pencil method for reference*

23	*10111*	*multiplicand*
19	*10011*	*multiplier*
	10111	
	10111	
	00000	
	00000	
	10111	
437	*110110101*	*product*

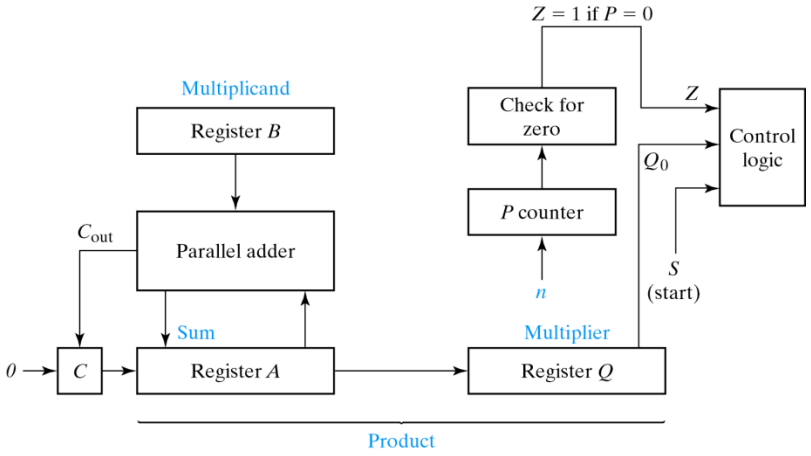


Fig. 8-13 Block Diagram of Binary Multiplier

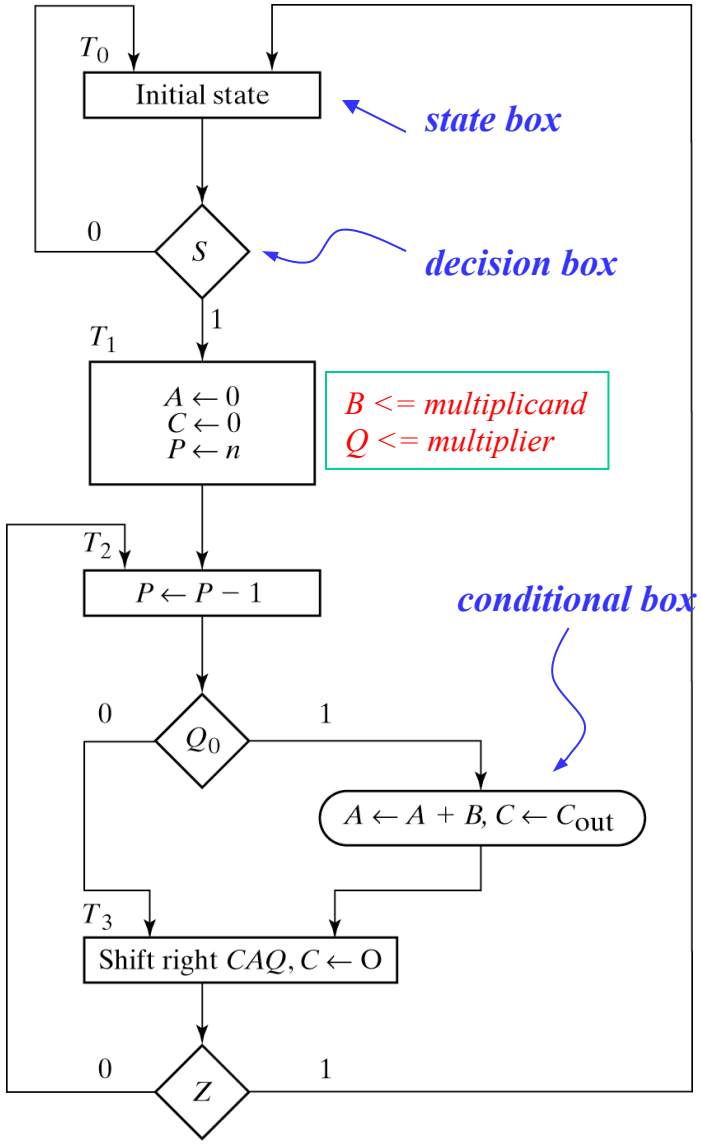


Fig. 8-14 ASM Chart for Binary Multiplier

Step-by-step analysis

clock	C	A	Q	State	P
		B = 1 0 1 1 1	Q = 1 0 0 1 1		
				T_1	x
	0	0 0 0 0 0	1 0 0 1 1	T_2	5
	0	1 0 1 1 1		T_3	4
	0	0 1 0 1 1	1 1 0 0 1	T_2	4
	1	0 0 0 1 0		T_3	3
	0	1 0 0 0 1	0 1 1 0 0	T_2	3
				T_3	2
	0	0 1 0 0 0	1 0 1 1 0	T_2	2
				T_3	1
	0	0 0 1 0 0	0 1 0 1 1	T_2	1
	0	1 1 0 1 1		T_3	0
	0	0 1 1 0 1	1 0 1 0 1	T_0	0

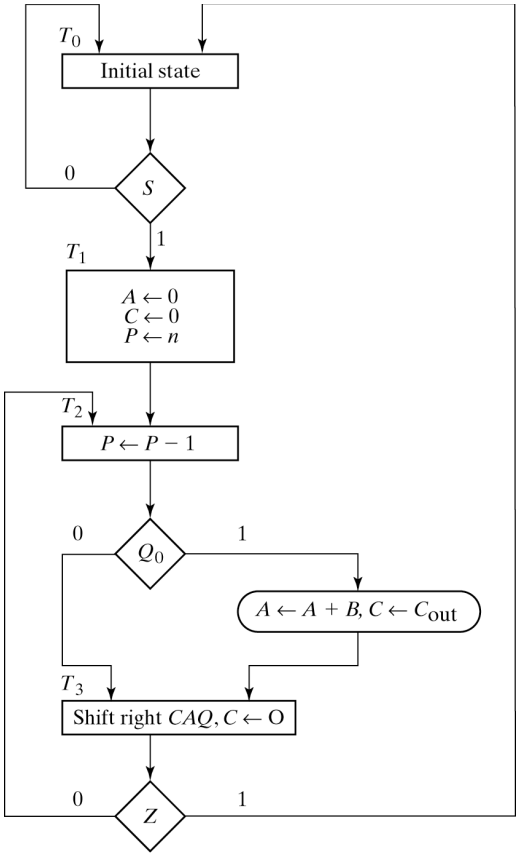


Fig. 8-14 ASM Chart for Binary Multiplier

Table 8-4
Numerical Example For Binary Multiplier

Multiplicand B = 10111

	C	A	Q	P
Multiplier in Q	0	00000	10011	101
$Q_0 = 1$; add B		10111		
First partial product	0	10111		100
Shift right CAQ	0	01011	11001	
$Q_0 = 1$; add B		10111		
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		10111		
Fifth partial product	0	11011		
Shift right CAQ	0	01101	10101	000

Final product in AQ = 0110110101

8.8 Control logic

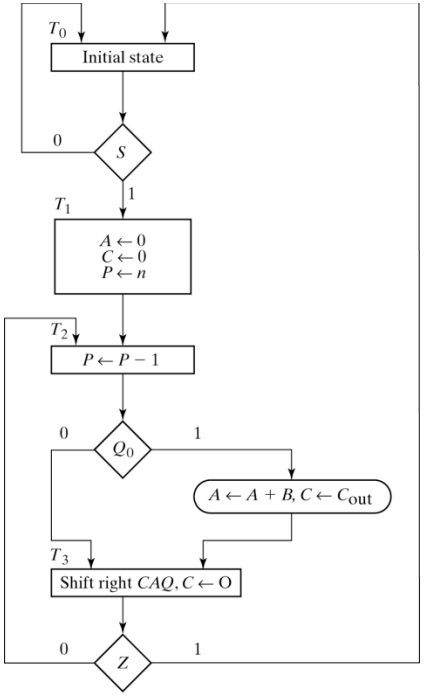
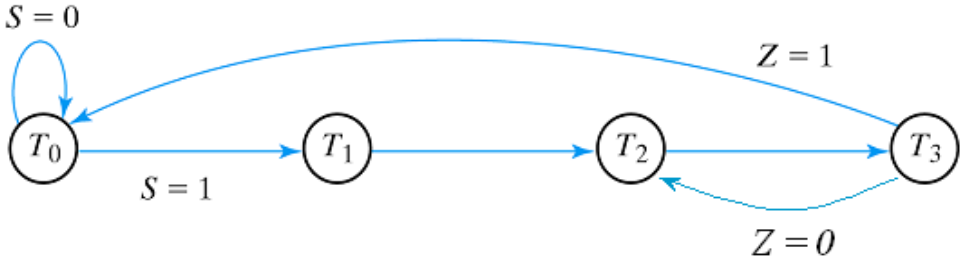


Fig. 8-14 ASM Chart for Binary Multiplier



(a) State diagram

For control logic design
From decision box

T_0: Initial state

T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$

T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

T_3: shift right $CAQ, C \leftarrow 0$

For datapath design
From state & conditional box

(b) Register transfer operations

Fig. 8-15 Control Specifications for Binary Multiplier

8.8 Control logic implementation

To do :

establish the required sequence of states,

provide signals to control the datapath operations (*conditional operations*)

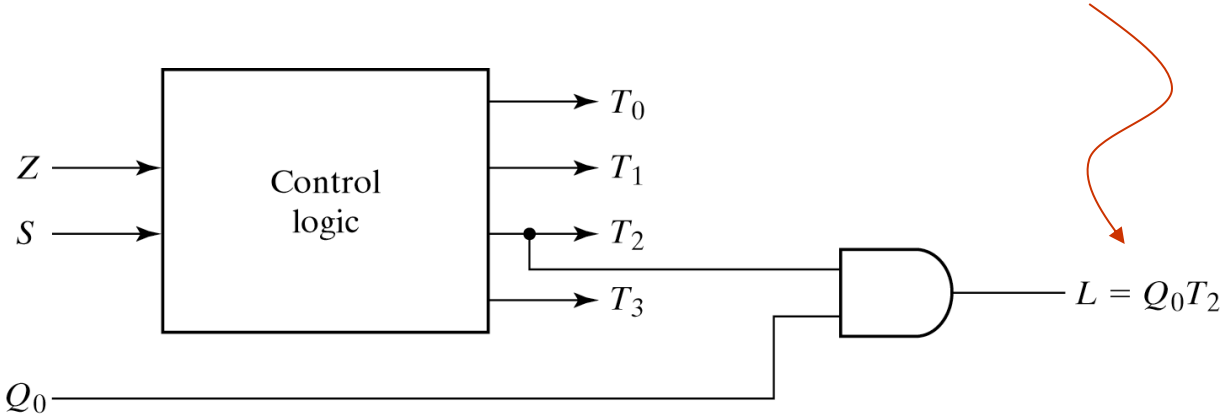


Fig. 8-16 Control Block Diagram

How ?

State assignment

When (# of inputs + # of states) is large, not easy to design

⇒ use specialized methods

A. sequence register and decoder method

B. one-hot (one FF per state) assignment

State assignment

Table 8-5
State Assignment for Control

State	Binary	Gray Code	One-Hot
T_0	00	00	0001
T_1	01	01	0010
T_2	10	11	0100
T_3	11	10	1000

Method I: Sequence register and decoder method

In the multiplier, 2 FF's and 2×4 decoder

1. State table
2. Use K-map if simple
3. For bigger or complex design
   by assuming T_o~T_3 available,
   find N/S equations directly from inspection of the state table
   → *The result is not the simplest. Thus, need to analyze the circuit*

1. State table

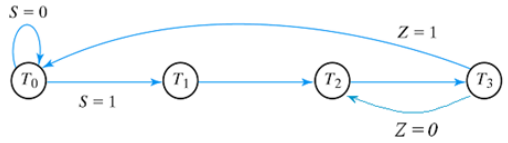


Table 8-6
State Table for Control Circuit

Present State		Input		Next State		Outputs			
G_1	G_0	S	Z	G_1	G_0	T_0	T_1	T_2	T_3
0	0	0	X	0	0	1	0	0	0
0	0	1	X	0	1	1	0	0	0
0	1	X	X	1	0	0	1	0	0
1	0	X	X	1	1	0	0	1	0
1	1	X	0	1	0	0	0	0	1
1	1	X	1	0	0	0	0	0	1

3. Next state equations

$D_1 = T_1 + T_2 + T_3 Z'$

$D_0 = T_0 S + T_2$

Control logic by *sequence register and decoder method*

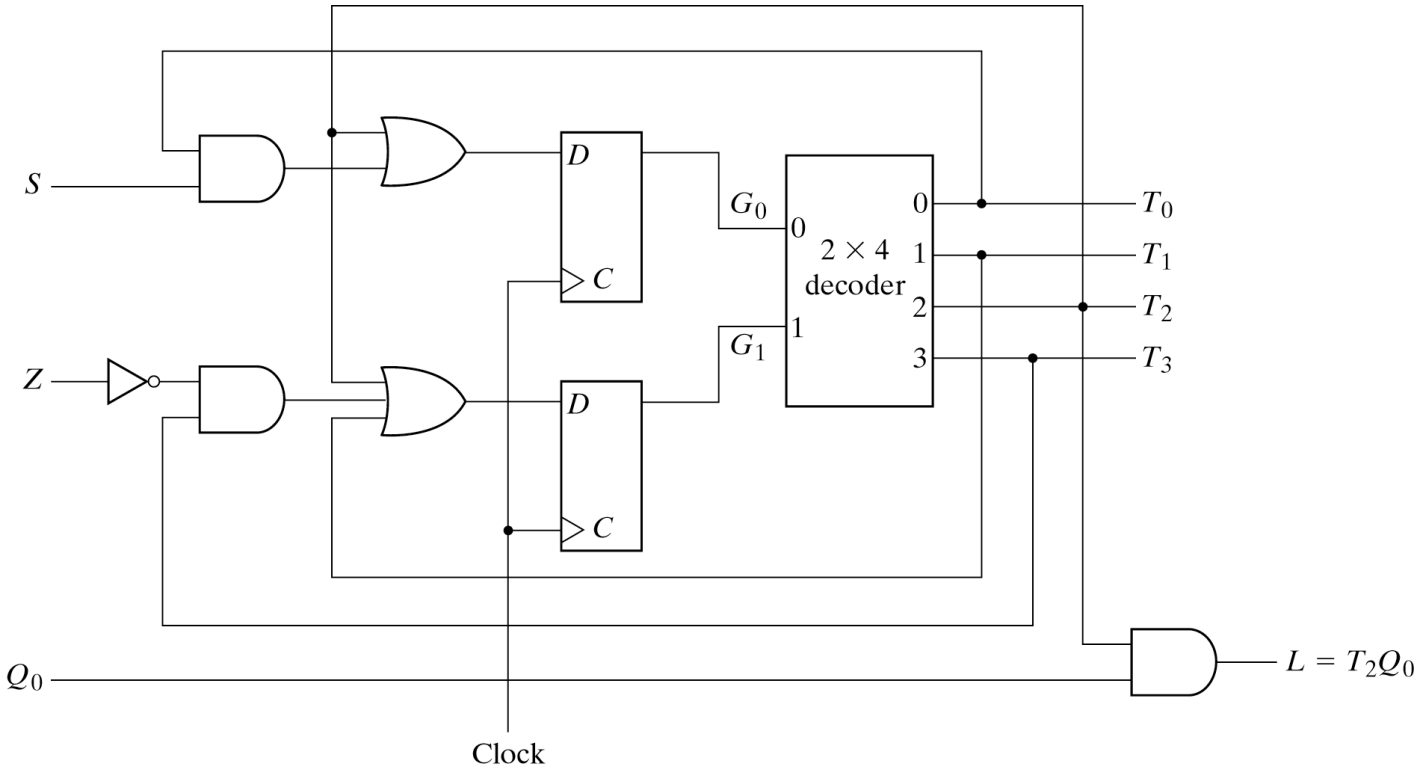
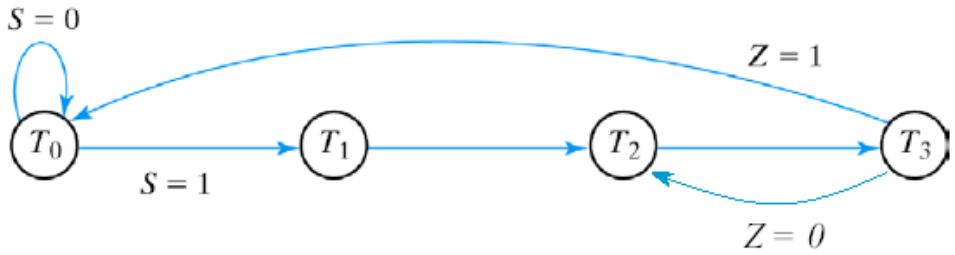


Fig. 8-17 Logic Diagram of Control for Binary Multiplier Using a Sequence Register and Decoder

Method II: One-hot assignment method

1. One FF per state
2. Uses more FF's, but simpler the design process (and less gates, no decoder)
3. Can be designed from inspection of ASM chart or state diagram

   i.e. no need to have a state table if DFF is used



(a) State diagram

$D_0 = T_0 S' + T_3 Z$

$D_1 = T_0 S$

$D_2 = T_1 + T_3 Z'$

$D_3 = T_2$

Control logic by *one-hot assignment method*

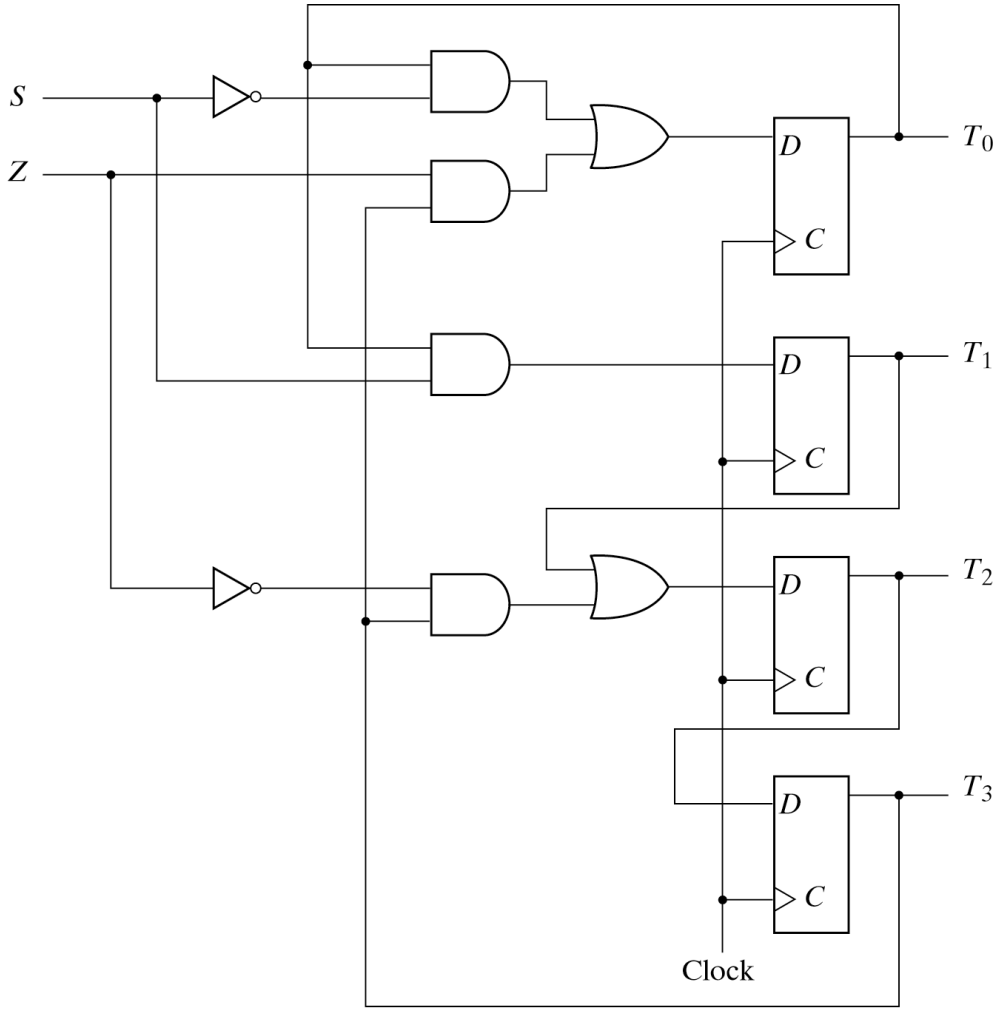


Fig. 8-18 One Flip-Flop Per State Controller

ASM chart for the example

** paper and pencil method for reference*

23	*10111*	*multiplicand*
19	*10011*	*multiplier*
	10111	
	10111	
	00000	
	00000	
	10111	
437	*110110101*	*product*

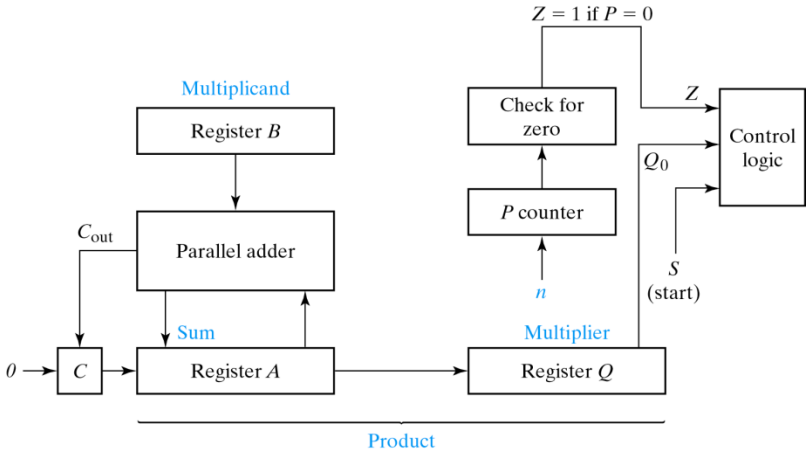


Fig. 8-13 Block Diagram of Binary Multiplier

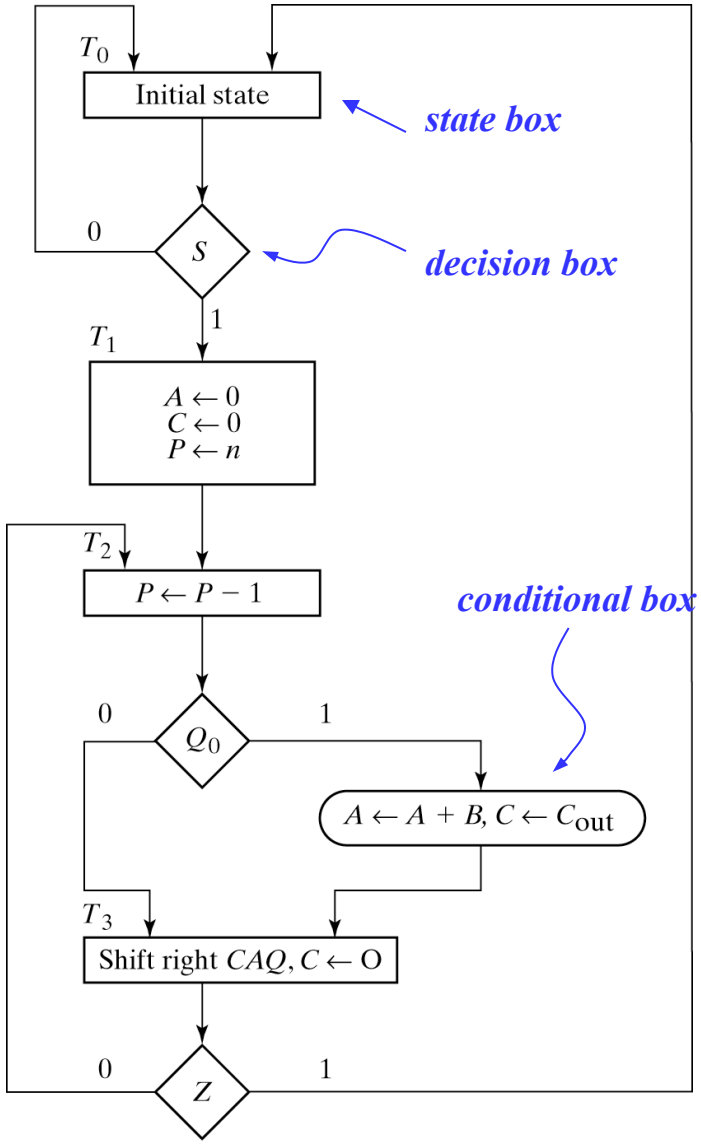


Fig. 8-14 ASM Chart for Binary Multiplier

8.9 HDL description of bi

- I/O define
- reg declaration and state enc
- combination circuit
- control logic (state transition
- datapath operations

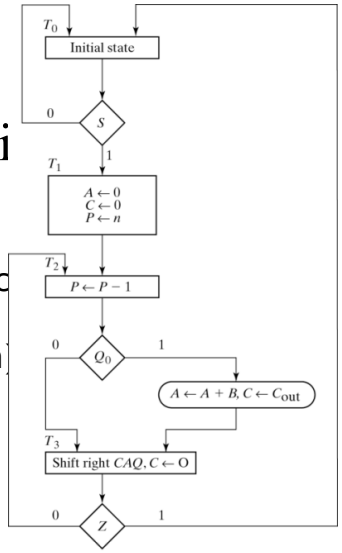


Fig. 8-14 ASM Chart for Binary Multiplier

```
//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 -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; else nstate = T0;
      T1: nstate = T2;
      T2: nstate = T3;
      T3: if (Z) nstate = T0; else   nstate = T2;
    endcase
//Register transfer operations, 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
```

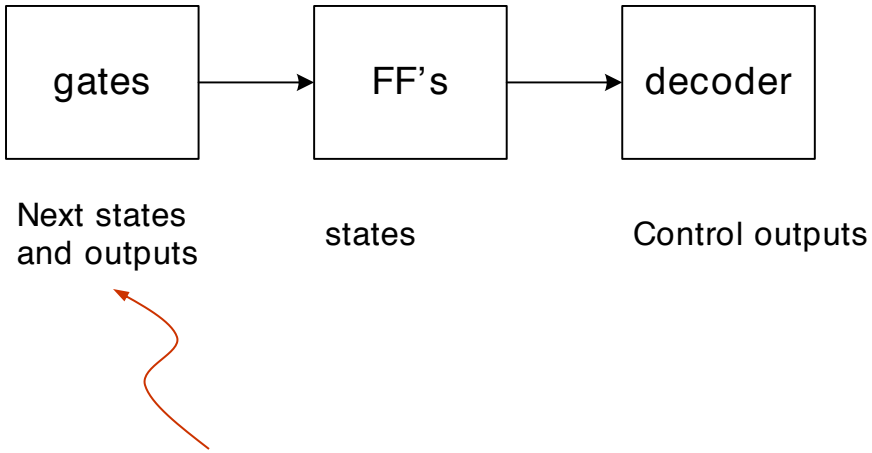
Behavioral description of binary multiplier

- simple (one multiplication statement)
- not always synthesizable (depends on tools used)

```
//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 multiplexer

Structure of the control logic
by sequence register and decoder method?



Replace this with MUX (to generate D input)

→ ***standard cells in many IC's***

Example problem

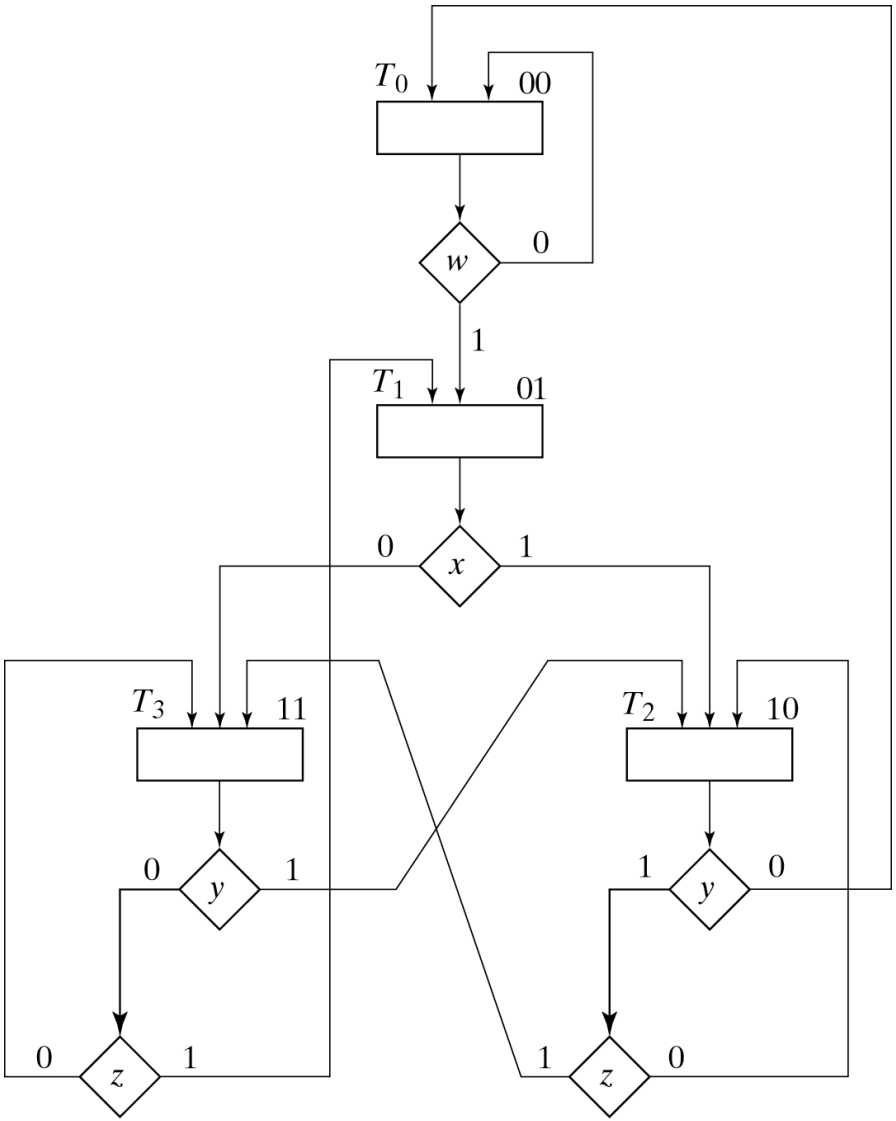


Fig. 8-19 Example of ASM Chart with Four Control Inputs

To evaluate the MUX inputs

prepare a table showing the input conditions
for each transition in the ASM chart

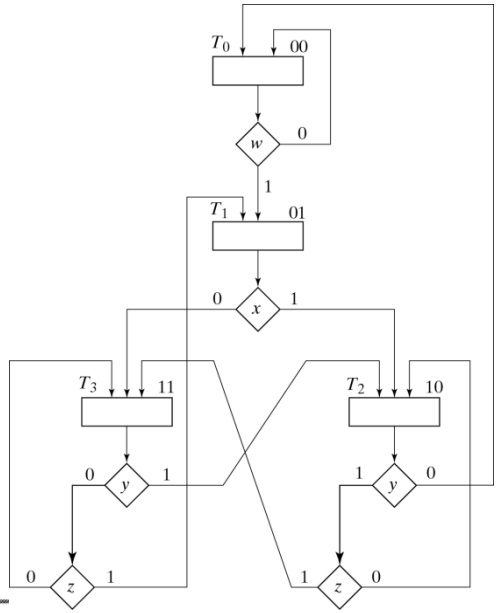


Fig. 8-19 Example of ASM Chart with Four Control Inputs

Table 8-7
Multiplexer Input Conditions

Present State		Next State		Input	Inputs (to D_1 and D_2)	
G_1	G_0	G_1	G_0	Conditions	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'	$yz' + yz = y$	yz
1	0	1	1	yz		
1	1	0	1	$y'z$		
1	1	1	0	y		
1	1	1	1	$y'z'$	$y + y'z' = y + z'$	$y'z + y'z' = y'$

Control logic obtained

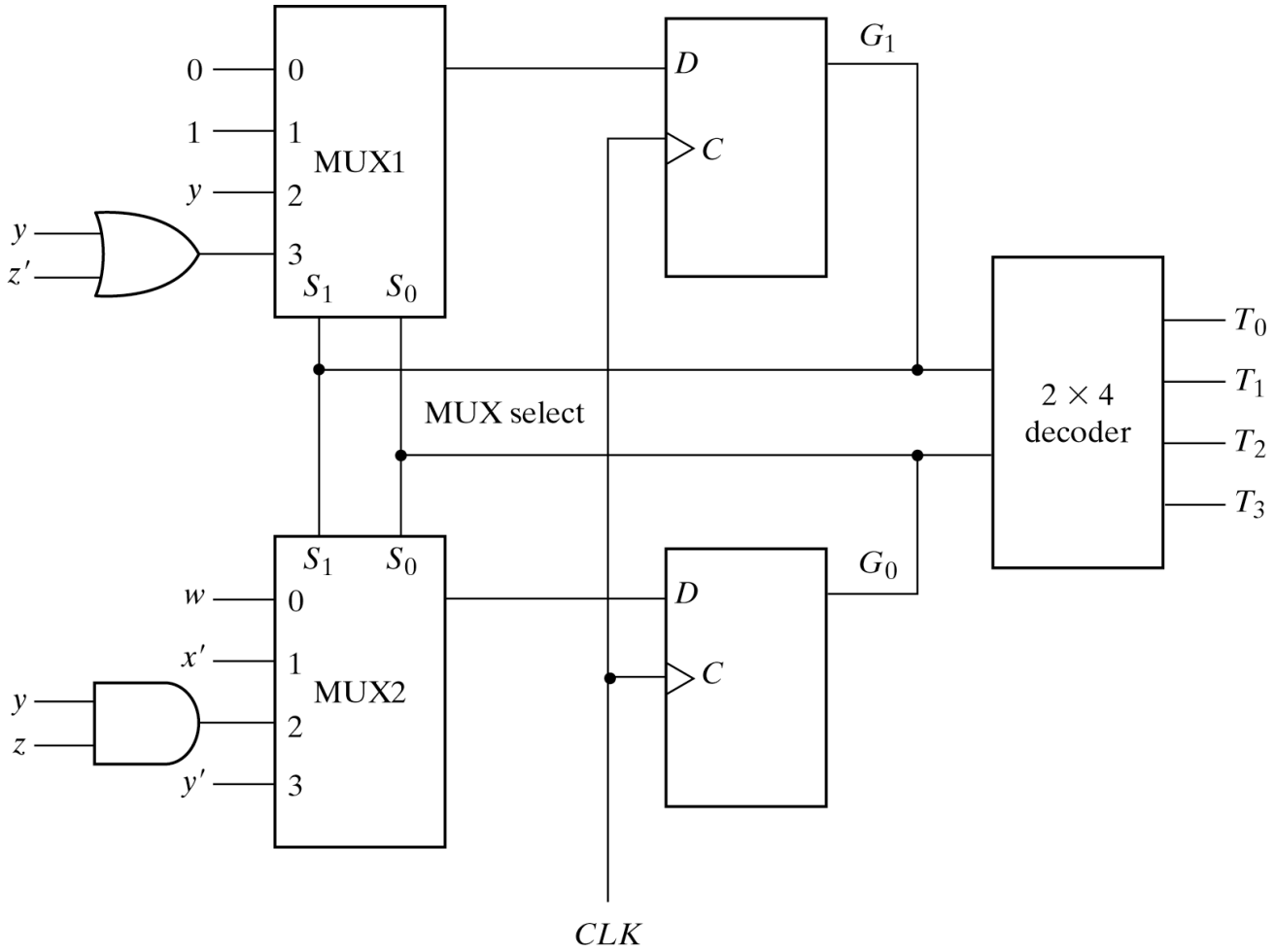


Fig. 8-20 Control Implementation with Multiplexers

Another example

Count the # of 1's in a register, R_1 and set R_2 to that number

How?
- shift each bit from R_1 into E
- When E=1, increment R_2

Control logic inputs?
- S for "start" ← external
- E ← from datapath
- Z (R1=0?)

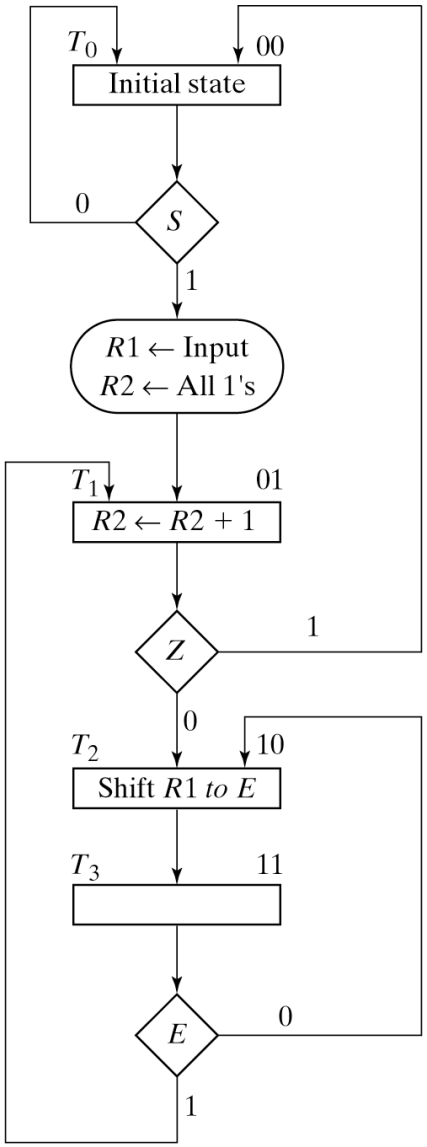


Fig. 8-21 ASM Chart for Count-of-Ones Circuit

Block diagram for the example

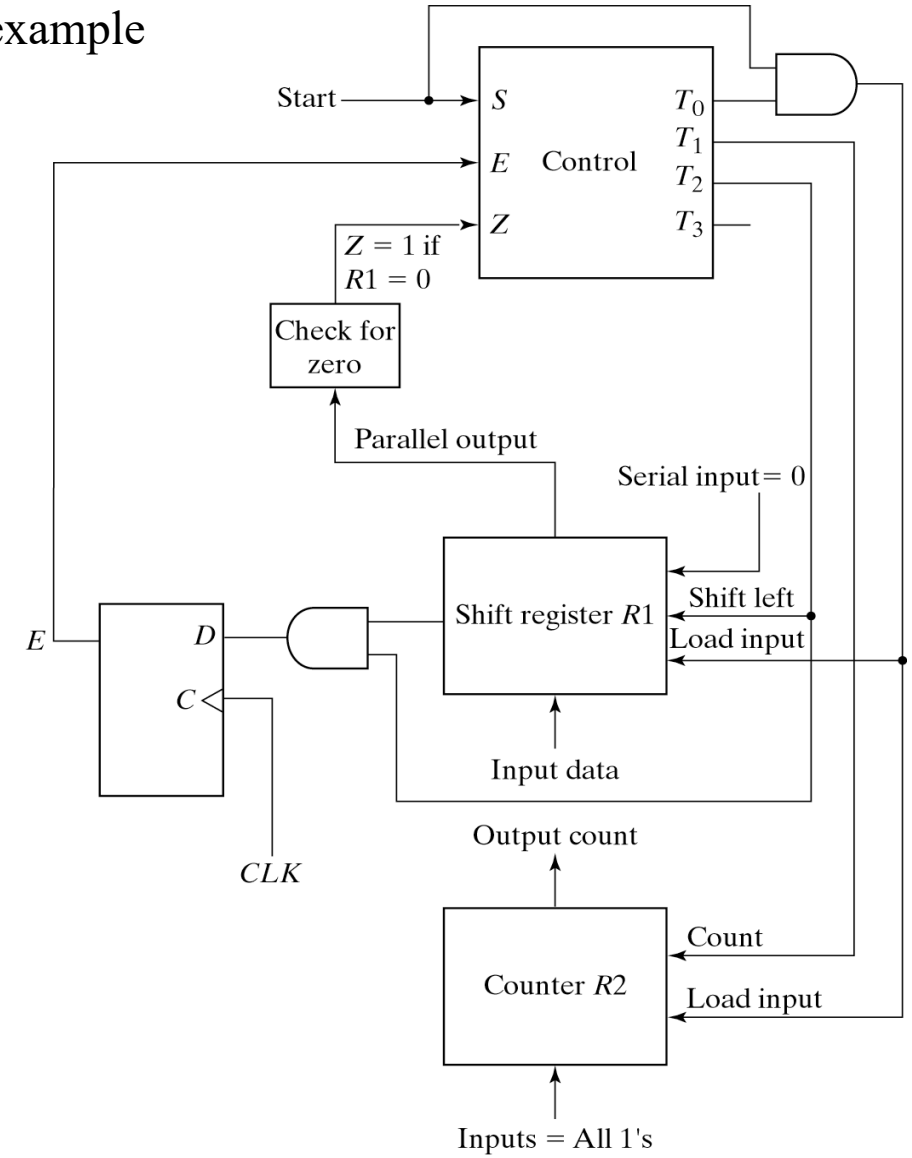


Fig. 8-22 Block Diagram for Count-of-Ones

MUX input conditions

Table 8-8
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	S'		
0	0	0	1	S	0	S
0	1	0	0	Z		
0	1	1	0	Z'	Z'	0
1	0	1	1	None	1	1
1	1	1	0	E'		
1	1	0	1	E	E'	E

Control logic obtained

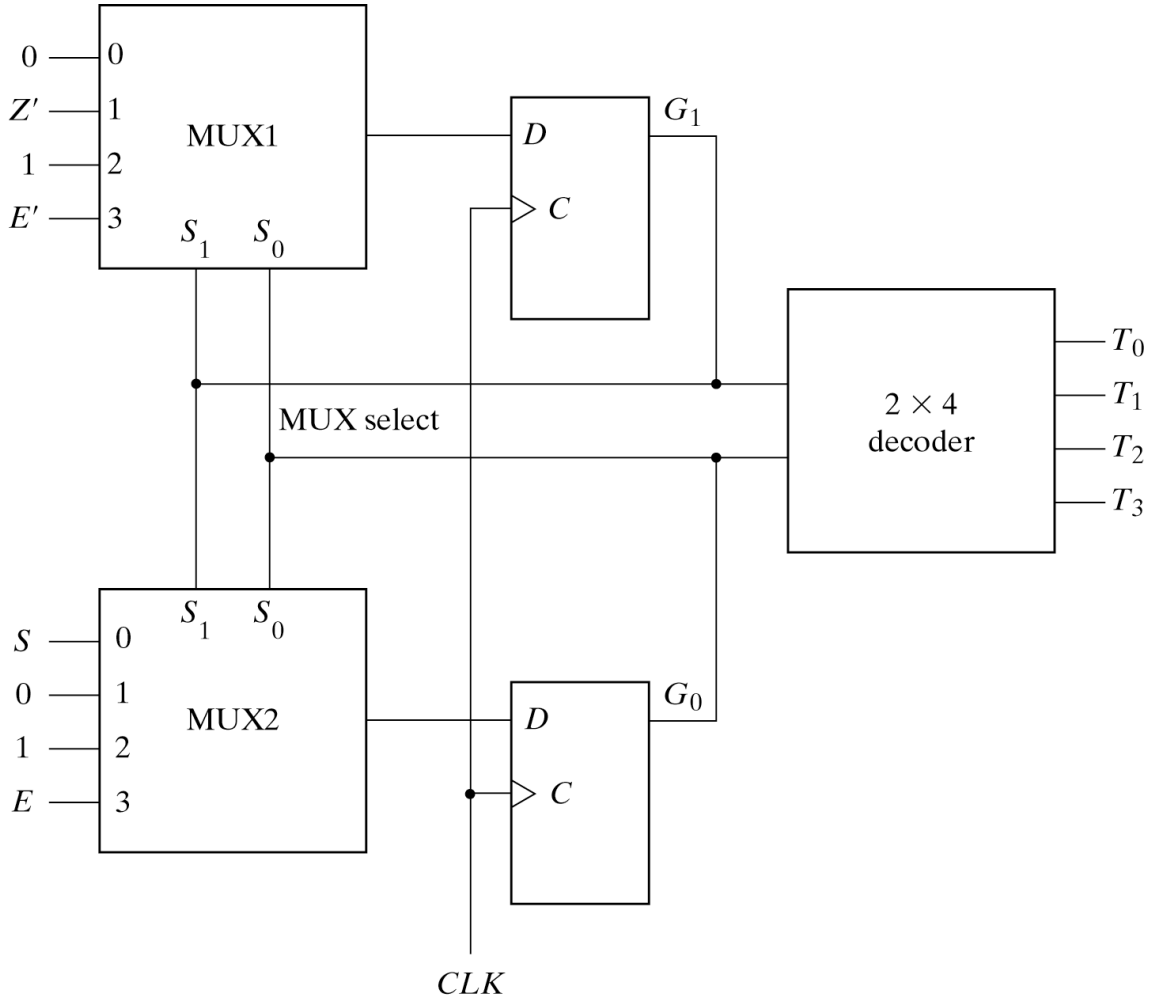


Fig. 8-23 Control Implementation for Count-of-Ones Circuit

RTL description of binary multiplier

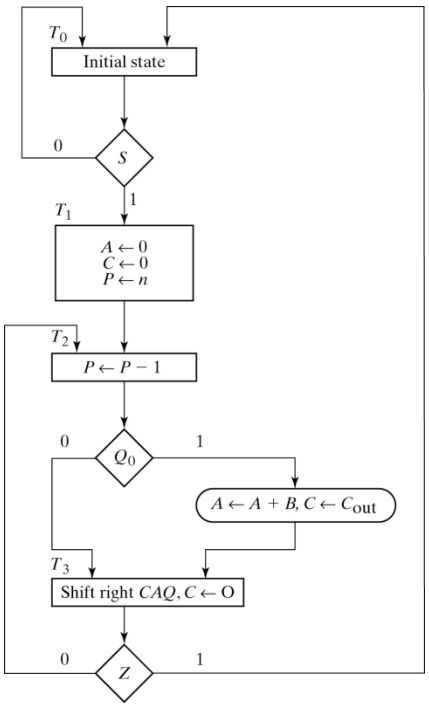
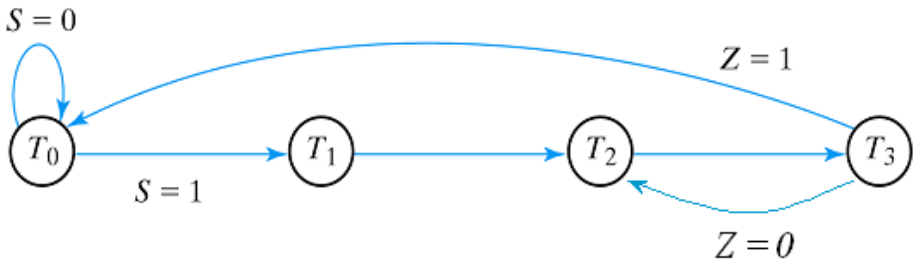


Fig. 8-14 ASM Chart for Binary Multiplier



(a) State diagram

T_0: Initial state

T_1: $A \leftarrow 0, C \leftarrow 0, P \leftarrow n$

T_2: $P \leftarrow P - 1$

if $(Q_0) = 1$ then $(A \leftarrow A + B, C \leftarrow C_{out})$

T_3: shift right $CAQ, C \leftarrow 0$

(b) Register transfer operations

Control Specifications for Binary Multiplier

RTL description of binary multiplier (new)

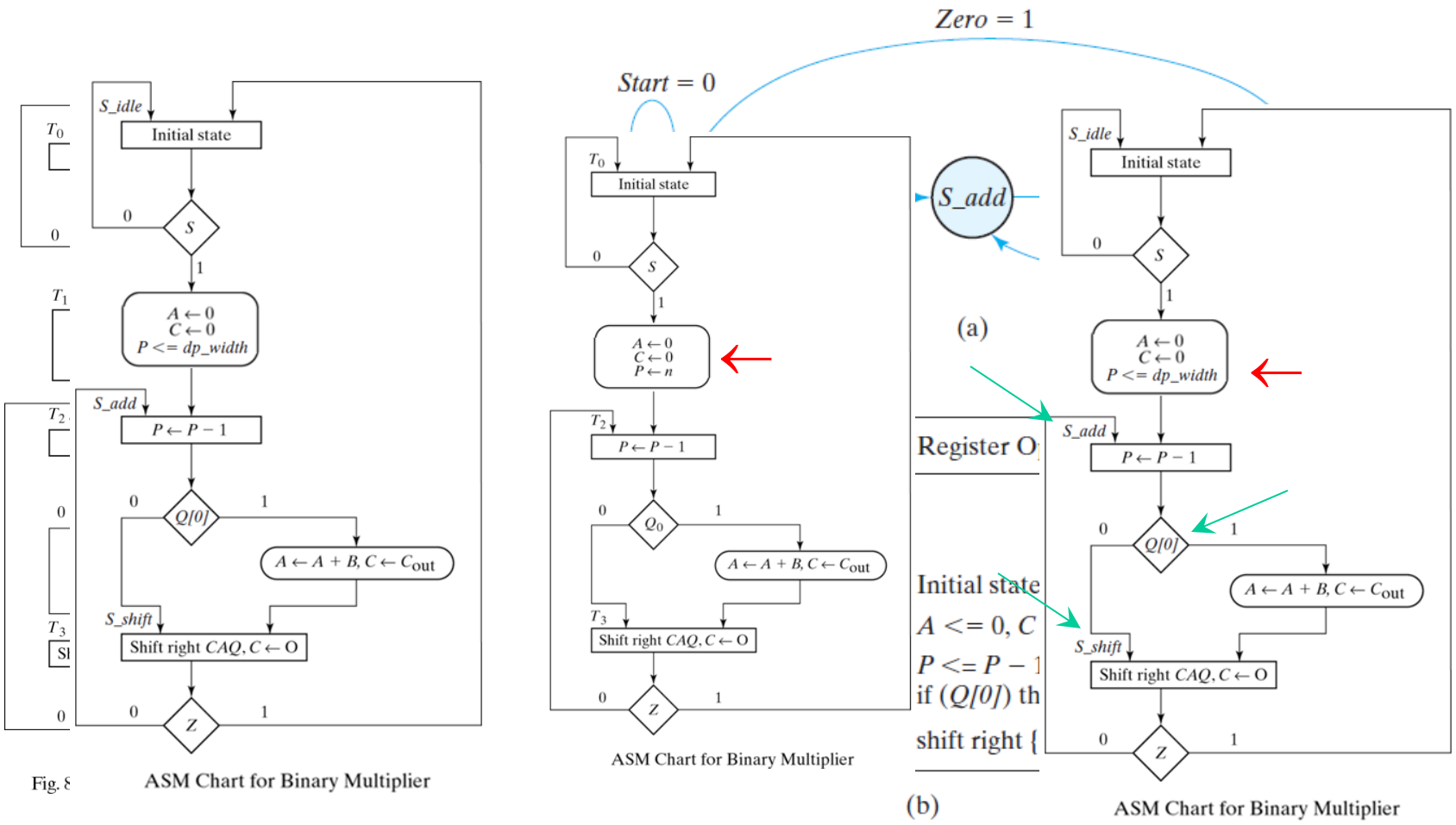


FIGURE 8.16
Control specifications for binary multiplier

ASMD chart of binary multiplier

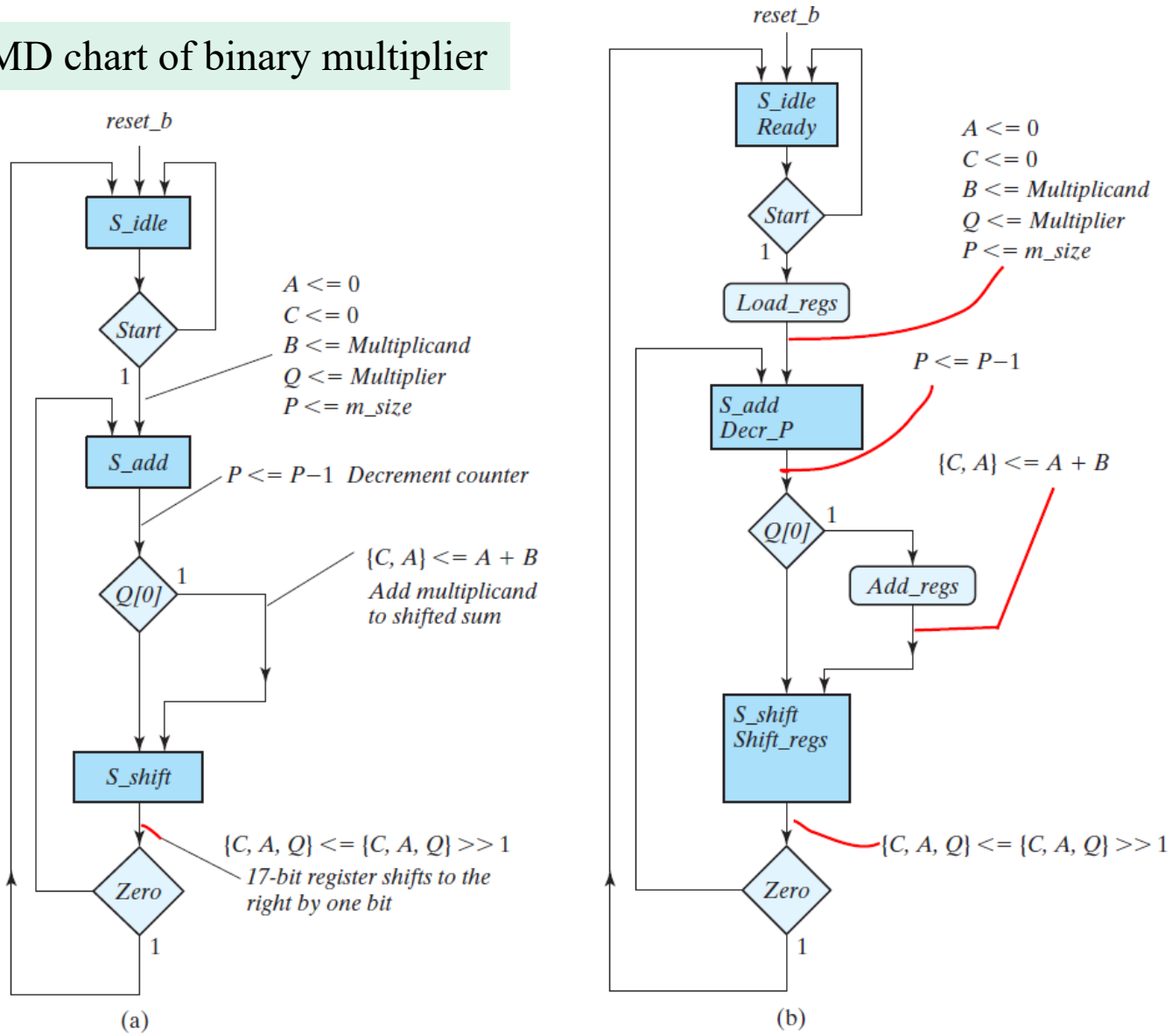


FIGURE 8.15
ASMD chart for binary multiplier

HDL description of ASMD chart (binary multiplier)

HDL Example 8.5

```
module Sequential_Binary_Multiplier (Product, Ready, Multiplicand, Multiplier, Start,
clock, reset_b);
// Default configuration: five-bit datapath
  parameter                              dp_width = 5;       // Set to width of datapath
  output        [2*dp_width -1: 0]       Product;
  output                                 Ready;
  input         [dp_width -1: 0]         Multiplicand, Multiplier;
  input                                  Start, clock, reset_b;

  parameter                              BC_size = 3;        // Size of bit counter
  parameter                              S_idle =   3'b001, // one-hot code
                                         S_add =    3'b010,
                                         S_shift =  3'b100;
  reg   [2: 0]                           state, next_state;
  reg   [dp_width -1: 0]       A, B, Q;              // Sized for datapath
  reg                          C;
  reg   [BC_size -1: 0]        P;
  reg                          Load_regs, Decr_P, Add_regs, Shift_regs;

// Miscellaneous combinational logic

  assign                       Product = {A, Q};
  wire                         Zero = (P == 0);          // counter is zero
                               // Zero = ~|P;             // alternative
  wire                         Ready = (state == S_idle); // controller status
```

HDL description of ASMD chart (binary multiplier)

```
// control unit
  always @ (posedge clock, negedge reset_b)
    if (~reset_b) state <= S_idle; else state <= next_state;

  always @ (state, Start, Q[0], Zero) begin
    next_state = S_idle;
    Load_regs = 0;
    Decr_P = 0;
    Add_regs = 0;
    Shift_regs = 0;
    case (state)
      S_idle:     begin if (Start) next_state = S_add; Load_regs = 1; end
      S_add:      begin next_state = S_shift; Decr_P = 1; if (Q[0]) Add_regs = 1; end
      S_shift:    begin Shift_regs = 1; if (Zero) next_state = S_idle;
                  else next_state = S_add; end
      default:    next_state = S_idle;
    endcase
  end
// datapath unit
  always @ (posedge clock) begin
    if (Load_regs) begin
      P <= dp_width;
      A <= 0;
      C <= 0;
      B <= Multiplicand;
      Q <= Multiplier;
    end
    if (Add_regs) {C, A} <= A + B;
    if (Shift_regs) {C, A, Q} <= {C, A, Q} >> 1;
    if (Decr_P) P <= P -1;
  end
endmodule
```