Register Transfer Level

CSE3201

RTL

- A digital system is represented at the register transfer level by these three components
  1. The set of registers in the system
  2. The operation that are performed on the data stored in the registers
  3. The control that supervises the sequence of operations in the system.
- The operations executed on the information stored in the registers are elementary operations and performed in parallel during one clock cycle.

RTL

- Comma is used to separate 2 or more operations that are executed in the same time

  `If(T3=1) then (R2←R1, R1←R2)`
- That is possible with registers that have edge triggered flip-flop
- Increment R1 then store it in R2

  `R2←R1+1`

RTL in HDL

```
assign s=A+B;

always  @(A or B)
      s=A+B;
```

Continuous assignment, for combinational circuits only, output can not be a reg

```
always @ (posedge clock)
      begin
         RA=RA+RB;
         RD=RA;
      end
```

Blocking procedural assignment, new value of RA is assigned to RD

```
always @ (negedge clock)
      begin
         RA<=RA+RB;
         RD<=RA;
      end
```

Non-blocking procedural assignment, old value of RA is assigned to RD

HDL Operations

- Arithmetic: + - * / %
- Logic (bit wise): ~ & | ^
- Logical ! && ||
- Shift >> << { , }
- Relational > < == != >= <=
- In shifting, the vacant bits are filled with zeros
- L

Loop Statements

-

```
integer count
initial
        begin
        count = 0;
        while (count <16)
        #5 count = count+1;
        end
```

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

Loop Statements

```
module decoder
        input [1:0] IN;
        output [3:0]Y;
        reg [3:0]Y;
        integer I;
        always @(IN)
          for (I=0; I<=3; I=I+1)
            if (IN == I)    Y[I]=1;
            else Y[I]=0;
endmodule
```

Synthesis

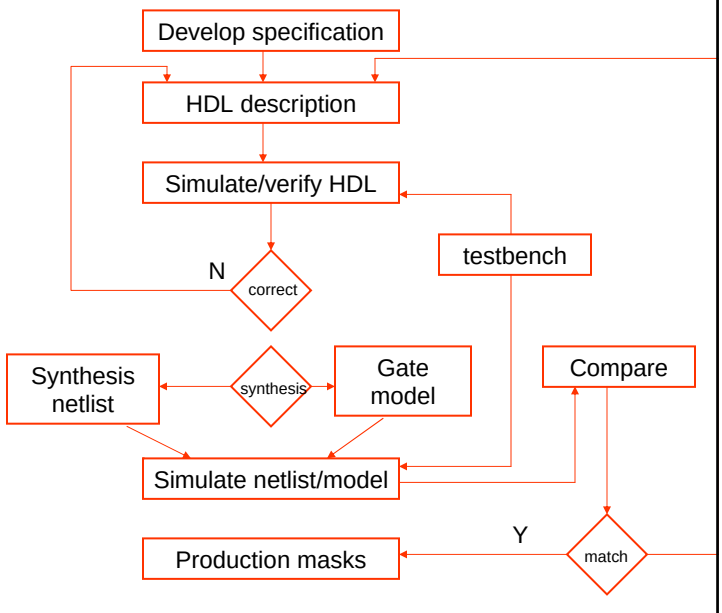
- Logic synthesis tools interpret the source code and translate it into an optimized gate structure.
- Statement like assign **`s=a&b`** is translated into an AND gate
- Statement like assign **`s=a+b`** is translated into adder
- Statemen like assign **`y=s? in_1:in_2`** is translated into a 2-to-1 multiplexer

Synhesis

- Always may imply combinational or sequential circuit based on the event control expression (level or edge)

```
always @(L1 or L0 or S)
  if (S) Y=I1;
  else Y=I0;
```

  Is translated into a 2-to-1 multiplexer
- If the event control is “posedge”, that is a synchronous sequential circuit



Algorithmic State Machine (ASM)

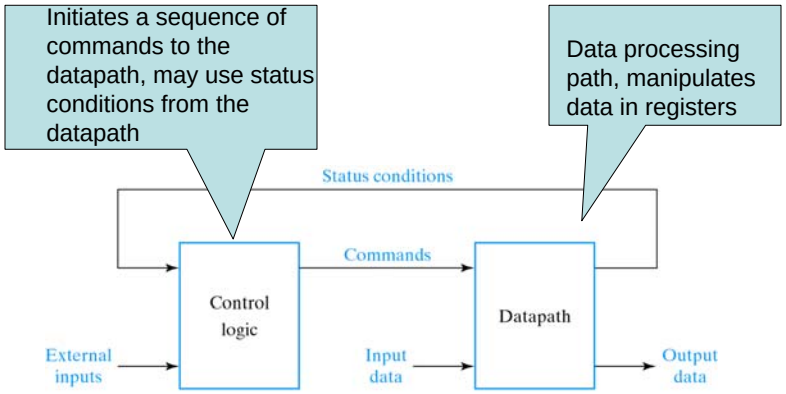


Fig. 8-2 Control and Datapath Interaction

ASM

- ASM is similar to flowchart in the sense that it specifies a sequence of procedural steps and decision paths for an algorithm.
- However, ASM is interpreted differently than a flowchart. While the flow chart is interpreted as a sequence of operations, ASM describes the sequence of events as well as the timing relationship between the states (as we will see shortly).

State Box

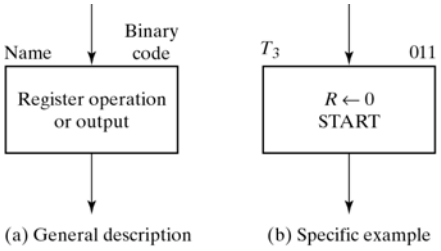


Fig. 8-3 State Box

The state is given a symbolic name (T3) (name may be inside the box)

Binary code for the assigned state (011)

The operations that are performed in this state R←0; and START could be an output signal is generated to start some operation

Operation happens when the machine makes a transition from T3 to next state

Decision Box

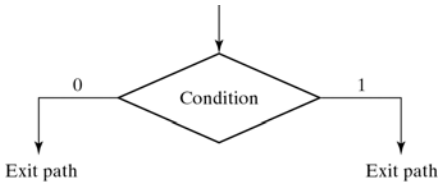


Fig. 8-4 Decision Box

Conditional Box

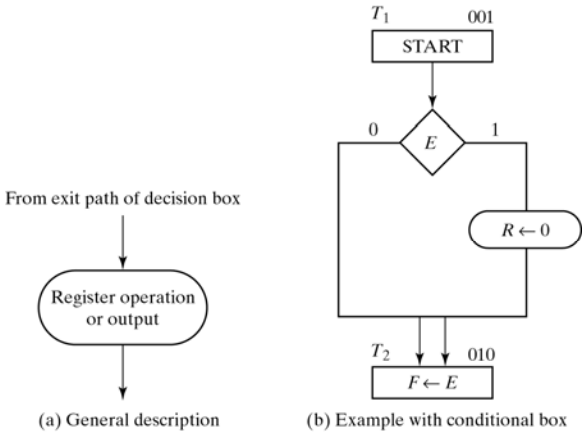


Fig. 8-5 Conditional Box

Input to the decision box must come from one of the exit paths of a decision box.

The register operation or outputs listed inside the conditional box are generated during a given state, if the input condition is satisfied of course

Depending on E, R is cleared or left unchanged

One entrance

ASM Block

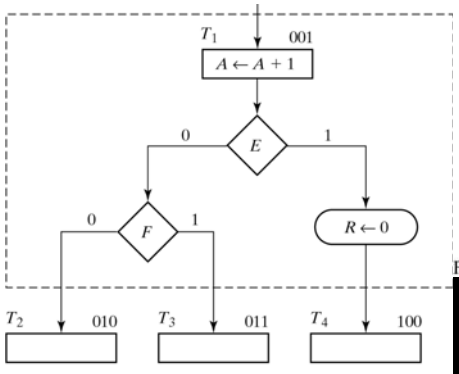


Fig. 8-6 ASM Block

If flow chart, A is incremented, then E is tested

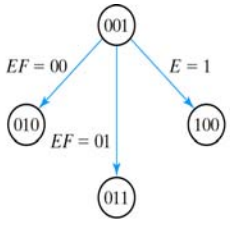


Fig. 8-7 State Diagram Equivalent to the ASM Chart of Fig. 8-6

ASM block is a structure consisting of one state box and all the decision and conditional boxes connected to its exit path.

Each block in the ASM describes the state of the system during one clock-pulse interval.

The operations within the state and conditional boxes are executed while the system is in state T_1 The same clock pulse transfer the system into T_2, T_3, or T_4

Timing Consideration

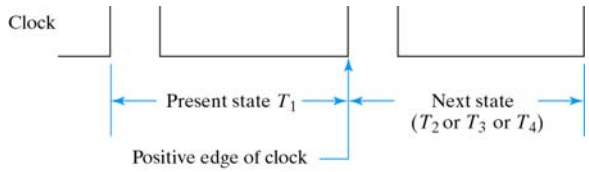
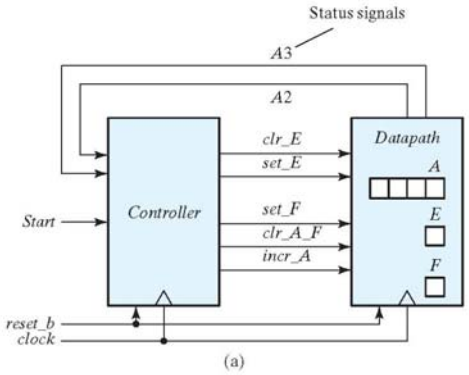


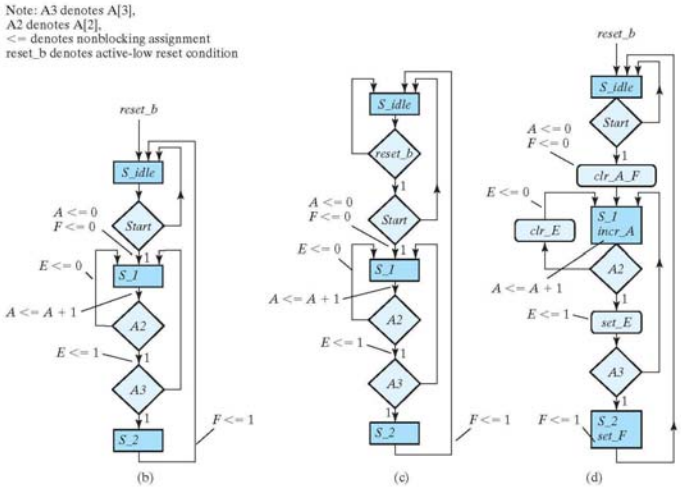
Fig. 8-8 Transition Between States

Design Example

- Design a system with 2 flip-flops E and F, and one 4 bit binary counter (A_3, A_2, A_1, A_0).
- A start signal initiates the operation by clearing A and F.
- Then the counter is incremented by one starting from the next clock pulse and continues to increment until the operation stops. A_3 and A_4 determine the operations.
  - If A_2 = 0, E is cleared and continue
  - If A_2=1, E is set; then if A_3=0 continue, if A_3=1 F is set to 1 on the next clock cycle.
- If *start* = 0, the system stays in the initial state, otherwise repeat

Example





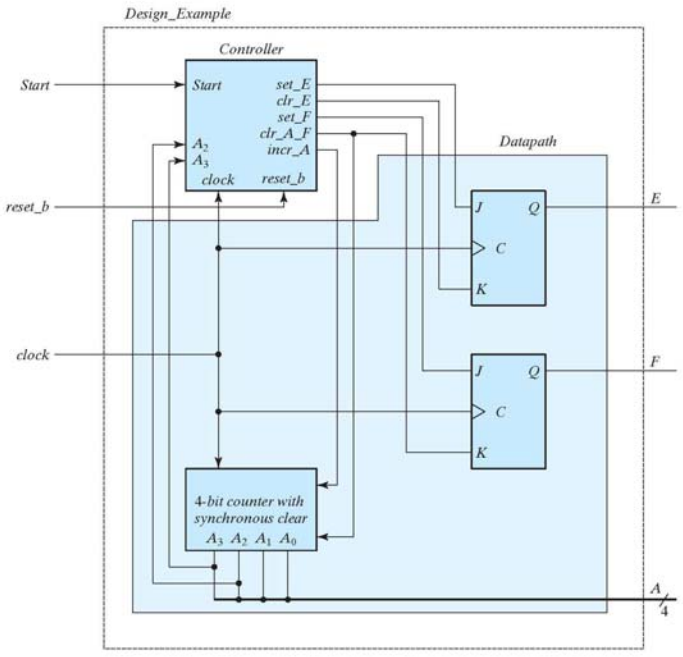
Asynchronous erset

Synchronous reset

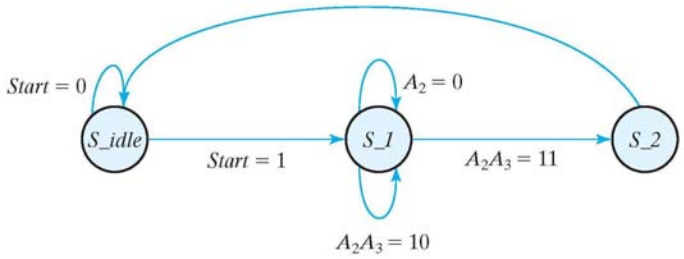
Table 8.3
Sequence of Operations for Design Example

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

Figure Number: Table08 03
Mano/Ciletti
Digital Design, 4e



RTL Description



(a)

S_idle ⟶ S_1, clr_A_F: A ⟵ 0, F ⟵ 0

S_1 ⟶ S_1, incr_A: A ⟵ A + 1

if (A2 = 1) then set_E: E ⟵ 1

if (A2 = 0) then clr_E: E ⟵ 0

S_2 ⟶ S_idle, set_F: F ⟵ 1

(b)

Table 8.4
State Table for the Controller of Fig. 8.10

	Present State		Inputs			Next State		Outputs				
Present-State Symbol	G_1	G_0	*Start*	A_2	A_3	G_1	G_0	*set_E*	*clr_E*	*set_F*	*clr_A_F*	*incr_A*
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

$DG_1 = S\_1\ A_2\ A_3$

$DG_0 = S\_1 + S\_Idle\ Start$

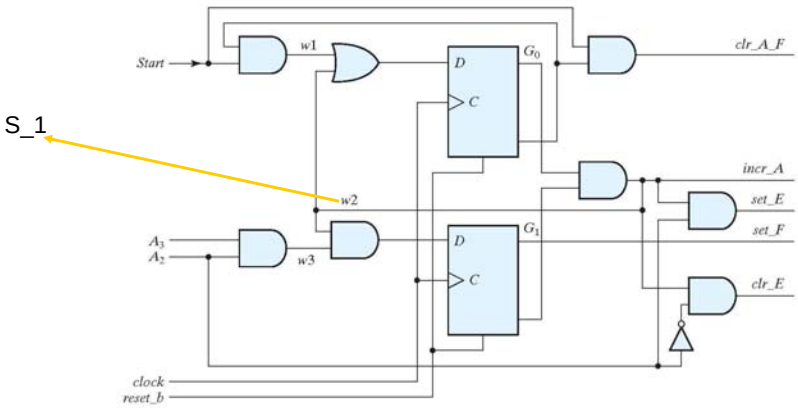
$Set\_E = S\_1\ A_2$

$Clr\_E = S\_1\ A'_2$

$Set\_F = S_{\_2}$

$Clr\_A\_F = Start\ S\_Idle$

$Incr\_A = s\_1$



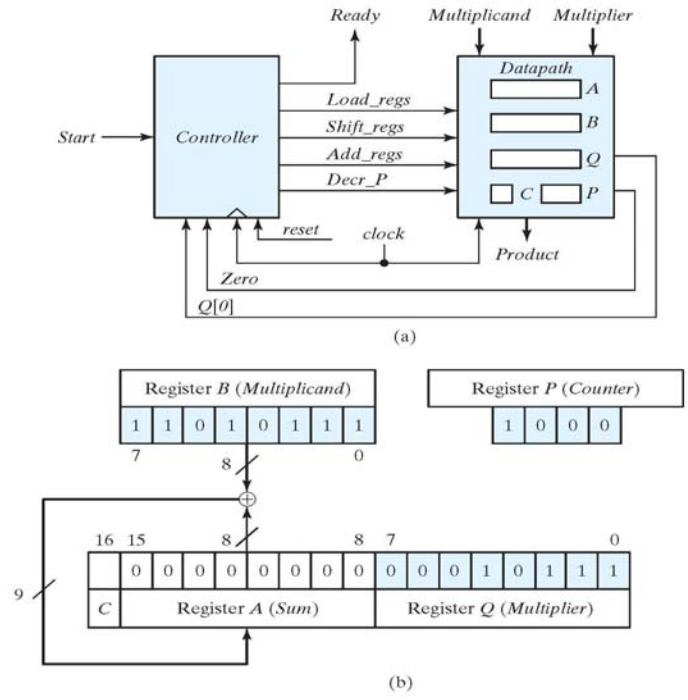
HDL Description

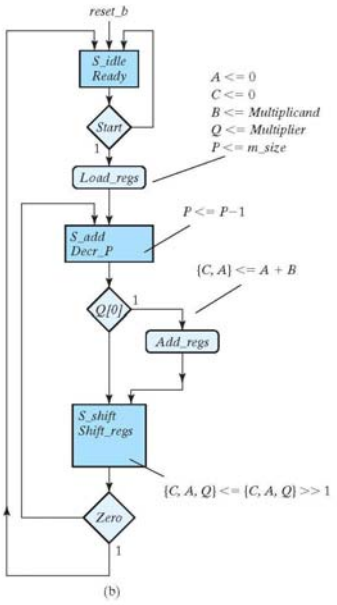
- The description could be on three different levels
  - Behavioral description on the RTL level
  - Behavior description on the algorithmic level
  - Structural description
- Note that the algorithmic level, is used only to verify the design *ideas* in the early stages. Some of the constructs might not be *synthesizable*
- Following RTL behavior description

Binary Multiplier

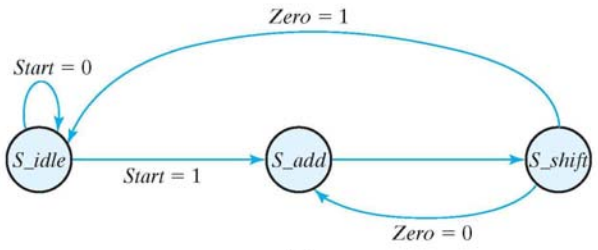
- We did this before using combinational circuit (adders, gaters, ..).
- Use one adder and shift registers.
- Instead of shifting multiplicand to the left, shift the partial product to the right.

23	10111
19	10011
	10111
	10111
	00000
	00000
	10111
437	110110101





(b)



(a)

State Transition		Register Operations
From	To	
S_idle		Initial state
S_idle	*S_add*	$A <= 0, C <= 0, P <= dp\_width$
S_add	*S_shift*	$P <= P - 1$ if $(Q[0])$ then $(A <= A + B, C <= C_{out})$
S_shift		shift right $\{CAQ\}$, $C <= 0$

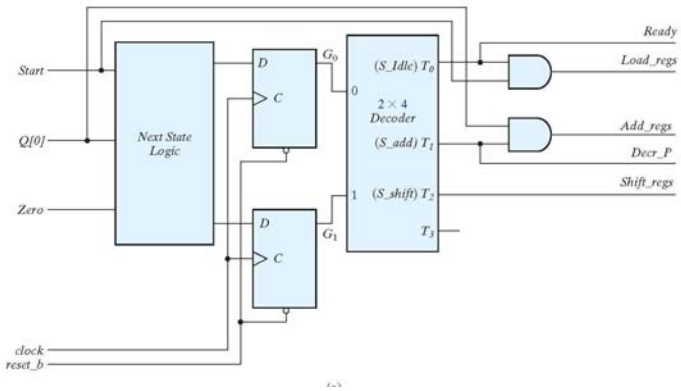
(b)

Table 8.6
State Assignment for Control

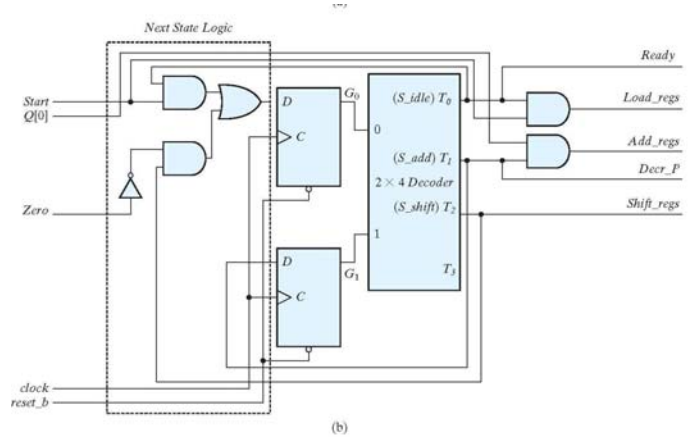
State	Binary	Gray Code	One-Hot
S_idle	00	00	001
S_add	01	01	010
S_shift	10	11	100

Table 8.7
State Table for Control Circuit

	Present State		Inputs			Next State						
Present-State Symbol	G_1	G_0	*Start*	*Q[0]*	*Zero*	G_1	G_0	*Ready*	*Load_regs*	*Decr_P*	*Add_regs*	*Shift_regs*
S_idle	0	0	0	X	X	0	0	1	0	0	0	0
S_idle	0	0	1	X	X	0	1	1	1	0	0	0
S_add	0	1	X	0	X	1	0	0	0	1	0	0
S_add	0	1	X	1	X	1	0	0	0	1	1	0
S_shift	1	0	X	X	0	0	1	0	0	0	0	1
S_shift	1	0	X	X	1	0	0	0	0	0	0	1

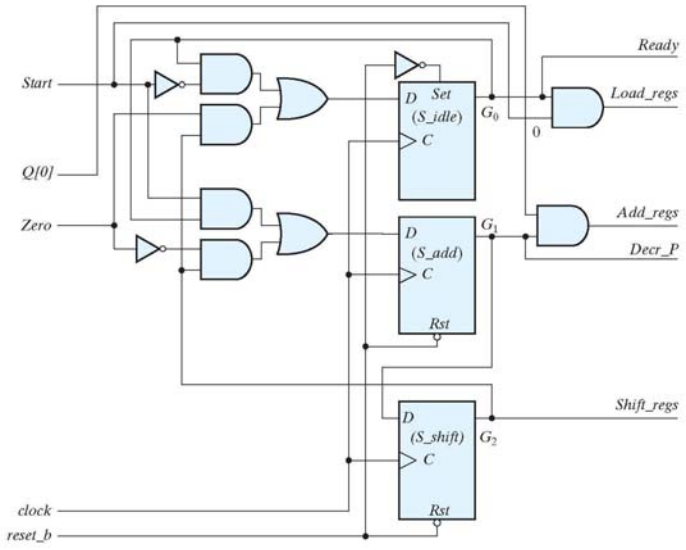


Using Sequence register and a decoder



One-Hot Design

- Every state is represented by a flip flop
- Only one contains a value of 1 at any time
- Simple but uses more FF



Design with multiplexers

- The previous design consists of flip-flops, decoder, and gates.
- Replacing gates with multiplexers results in a regular pattern of the design.
  - First level contains multiplexers (possibly added gates, but only one level.
  - The second level is the registers to hold the present state information
  - The last stage has a decoder that provides a separate output for every state

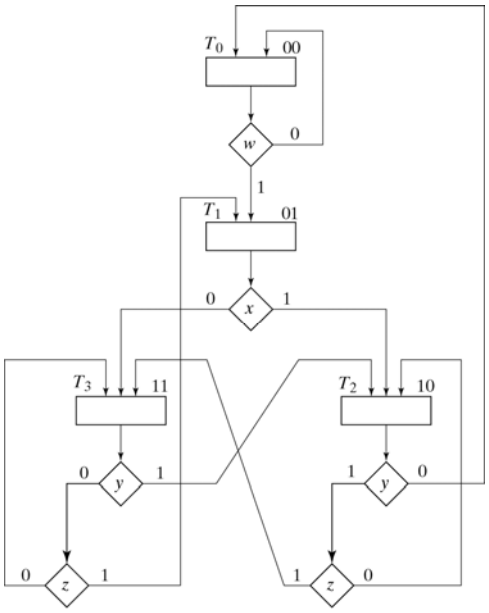


Fig. 8-19 Example of ASM Chart with Four Control Inputs

Multiplexer input condition

Present State		next State		I/P	inputs	
G1	G0	G1	G0	cond.	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	y+y’z=y+z	y’z+y’z’=y’
1	1	1	1	y’z’		

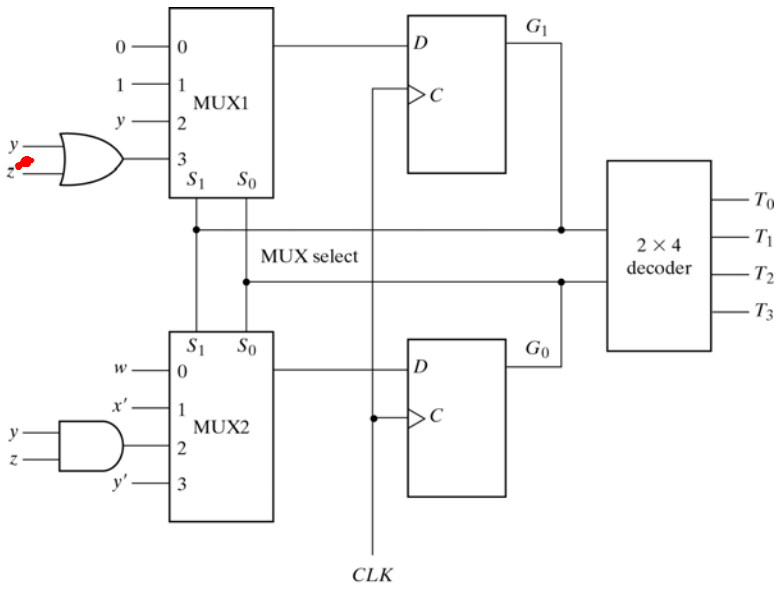
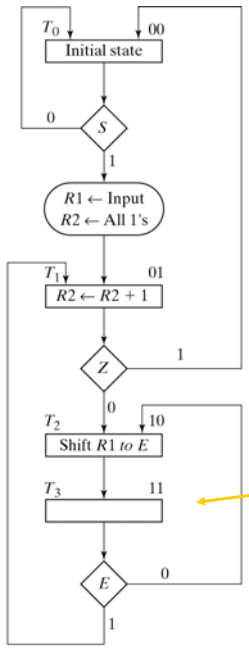


Fig. 8-20 Control Implementation with Multiplexers

Counting the number of 1's

- The system counts the number of 1's in R1, and set R2 accordingly.
- The bits in R1 are shifted one at a time, checking if the shifted out bit is 1 or 0, and incrementing R2
- Z is a signal to indicate if R1 contains all 0's or not.
- E is the output of the flip-flop (the shifted out bit).



?

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

E could not be checked in the same block as T2 since the shift to E will not happen unstill the end of the cycle.

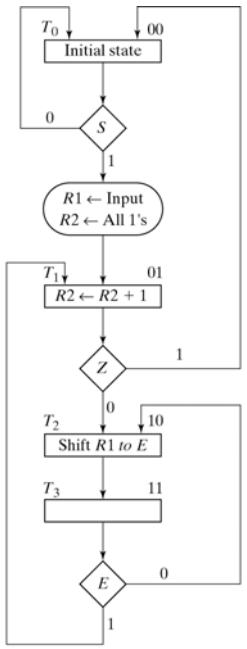


Fig. 8-21 ASM Chart for Count-of-Ones Circuit

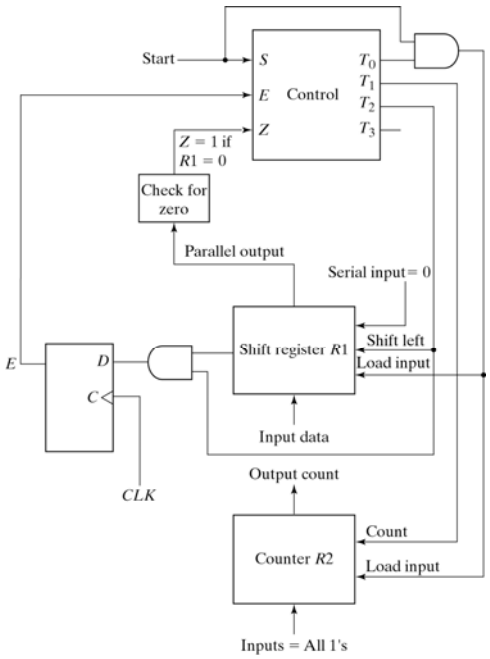


Fig. 8-22 Block Diagram for Count-of-Ones

Control (counting of 1's)

Present State		Next State		Conditions	MUX inputs	
G1	G0	G1	G0		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		1	1
1	1	1	0	E'		
1	1	0	1	E	E'	E

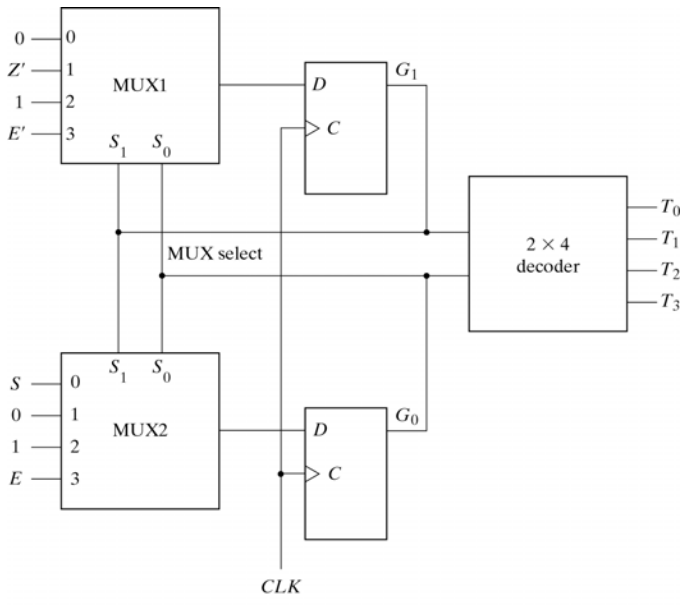


Fig. 8-23 Control Implementation for Count-of-Ones Circuit

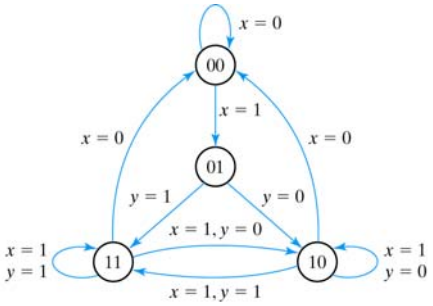


Fig. P8-10 Control State Diagram for Problems 8-10 and 8-11

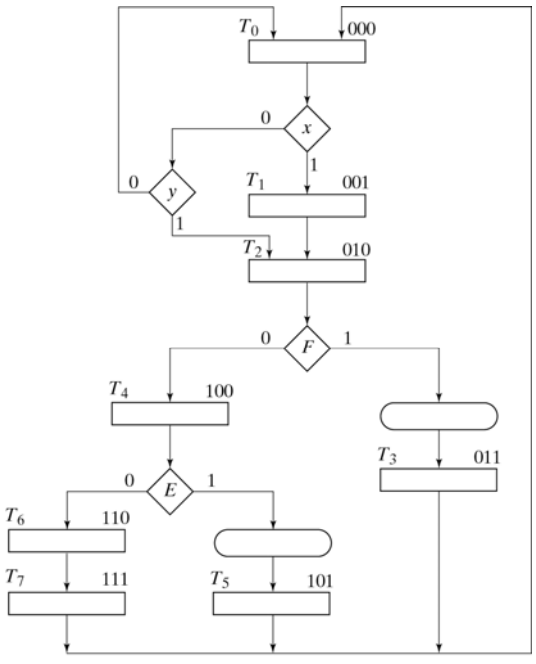


Fig. P8-20 ASM Chart for Problems 8-20