Register Transfer Level

ECE470

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

RTL in HDL

```
assign s=A+B;

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

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

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

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

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

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

Loop Statements

•

```
integer count
initial
        begin
        count = 0;
        while (count <0)
        #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
```

```
assign Y=s ? L1: L0;
Or
always @(L1 or L0 or S)
        if (S) Y=I1;
        else Y=I0;
```

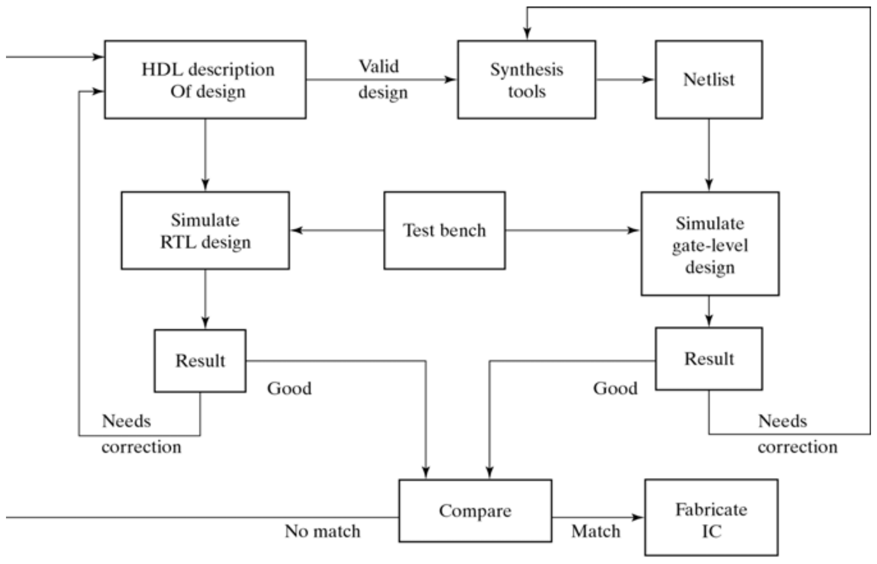


Fig. 8-1 Process of HDL Simulation and Synthesis

Algorithmic State Machine (ASM)

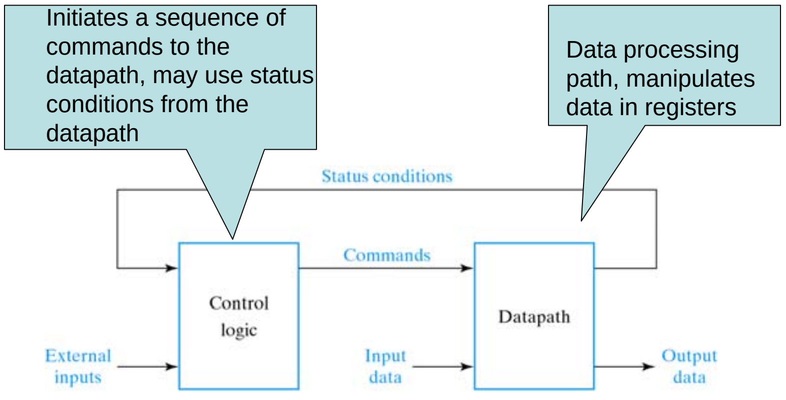

Initiates a sequence of commands to the datapath, may use status conditions from the datapath
Data processing path, manipulates data in registers
Status conditions
Commands
Control logic
Datapath
External inputs
Input data
Output data

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

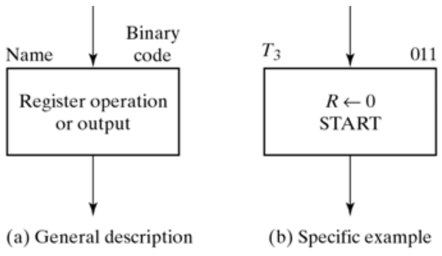


Fig. 8-3 State Box

The state is given a symbolic name (T3)

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

The operation is performed when we leave T_3 to the next state

Decision Box

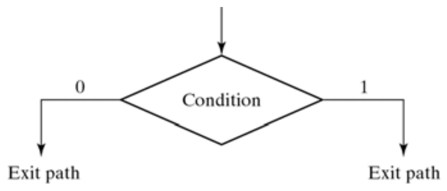


Fig. 8-4 Decision Box

Conditional Box

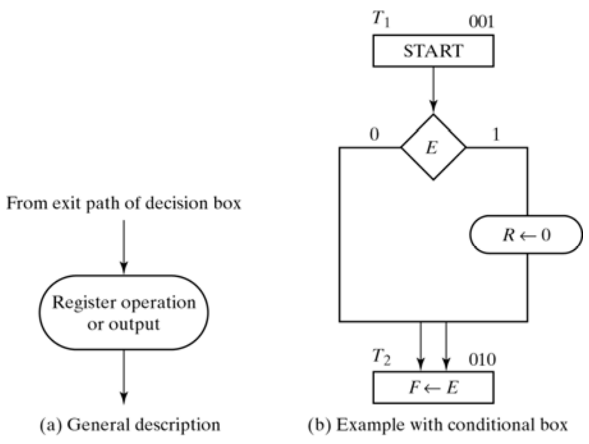

From exit path of decision box
Register operation or output
(a) General description
T_1
001
START
0
E
1
$R \leftarrow 0$
T_2
010
$F \leftarrow E$
(b) Example with conditional box

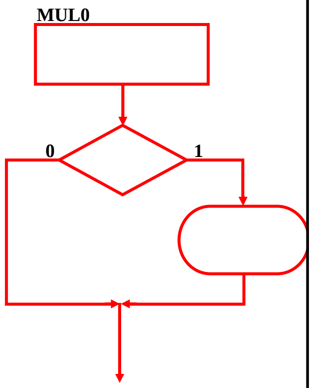
Fig. 8-5 Conditional Box

Input to the conditional 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

ASM

- The operation in the state box or conditional box are nott executed in the current state.
- Rather, a control signal is asserted in the current state if Q0 is 1 and the operation is done at the transition from this state to the next one (with the next clock cycle)



One entrance

ASM Block

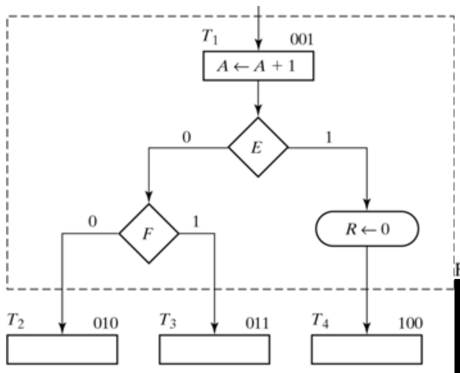


Fig. 8-6 ASM Block

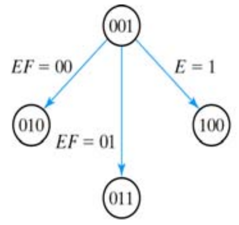


Fig. 8-7 State Diagram Equivalent to the ASM Chart of Fig. 8-6

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

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 at the clock pulse when the system is leaving T_1 to T_2, T_3, or T_4

Timing Consideration

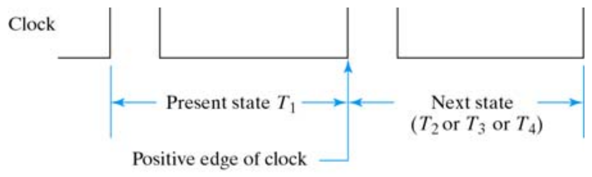


Fig. 8-8 Transition Between States

Design Example

- Design a system with 2 flip-flops E and F, and one 4 bit binary counter (A_4, A_3, A_2, A_1).
- 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_3 = 0, E is cleared and continue
  - If A3=1, E is set; then if A4=0 continue, if A4=1 F is set to 1 on the next clock cycle and the system stops.

BLOCK

BLOCK

BLOCK

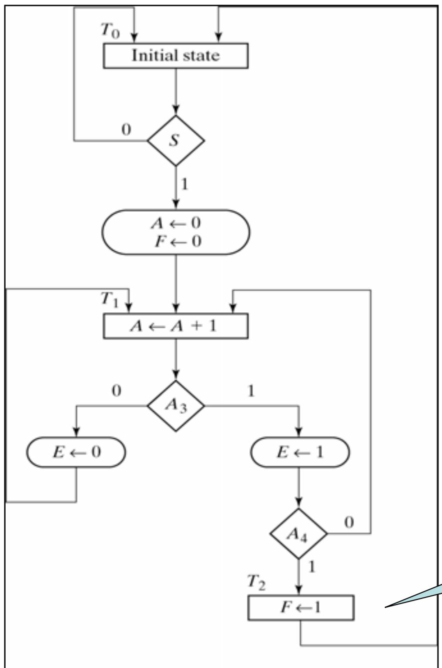


Fig. 8-9 ASM Chart for Design Example

F is set to 1 in the next clock cycle, must be a separate state

Counter				Flip-Flops			
A_4	A_3	A_2	A_1	E	F	Condition	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

Timing Sequence

- That illustrates the difference between ASM and flowchart.
  - When the system is in state 1011, It checks A_3 is 0, so it sets E to 0 and increment counter to 1100, the next cycle will start with 1100 and E set to 0.
  - Then checks A_3 and A_4 (both are 1), it sets E to 1, and increments counter.
  - Next cycle counter is 1101, and E=1 and now it is in state 2
  - Then it set F to 1 and goes to state 0

Datapath Design

- The requirements for the design of the datapath are specified in the state and conditional boxes.
- The control logic is determined from the decision and the required state transition.
- A look at the datapath design of the previous example.

Datapath design

- In state T_0, clear the counter and the F flip-flop (the and gate and f inputs).
- In state T_1, if A_3=0, set $E \leftarrow 0$ (will generate 01 on the JK inputs of E if the state is T1).
- In state T_1, if A_3A_4=10; $E \leftarrow 0$ (note the inputs of E).
- In state T_2, if A_3A_4=11, $F \leftarrow 1$, and (the F-F is set).

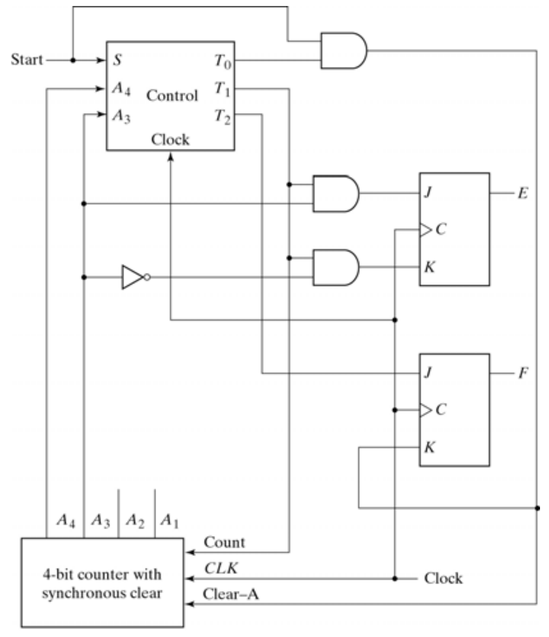
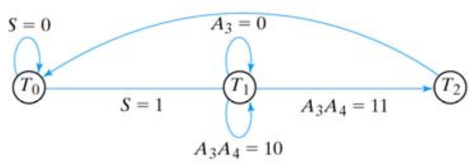


Fig. 8-10 Datapath for Design Example



(a) State diagram for control

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

T_1: $A \leftarrow A + 1$

if ($A3 = 1$) then $E \leftarrow 1$

if ($A3 = 0$) then $E \leftarrow 0$

T_2: $F \leftarrow 1$

(a) Register transfer operations

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

Sometimes it is useful to separate the control operation from the register transfer of the datapath.

The state diagram represents the control sequence; while the register transfer operation represents what happens in every state.

State Table

- The state diagram can be converted into a state table.
- Three states (T_0, T_1, and T_2), represents as the output of two registers (G_1 G_0) as 00, 01, and 11.
- The following table shows the state table for the previous example.

	Present State		**Inputs**			**next State**		**Outputs**		
Present (symbol)	**G1**	**G0**	**S**	**A3**	**A4**	**G1**	**G0**	**T0**	**T1**	**T2**
T0	0	0	0	X	X	0	0	1	0	0
T0	0	0	1	X	X	0	1	1	0	0
T1	0	1	X	0	X	0	1	0	1	0
T1	0	1	X	1	0	0	1	0	1	0
T1	0	1	X	1	1	1	1	0	1	0
T2	1	1	X	X	X	0	0	0	0	1

$T_0=G_0'$

$T_1=G_1'G_0$

$T_2=G_1$

$D_{G1}=T_1A_3A_4$

$D_{G2}=T_0S+T_1$

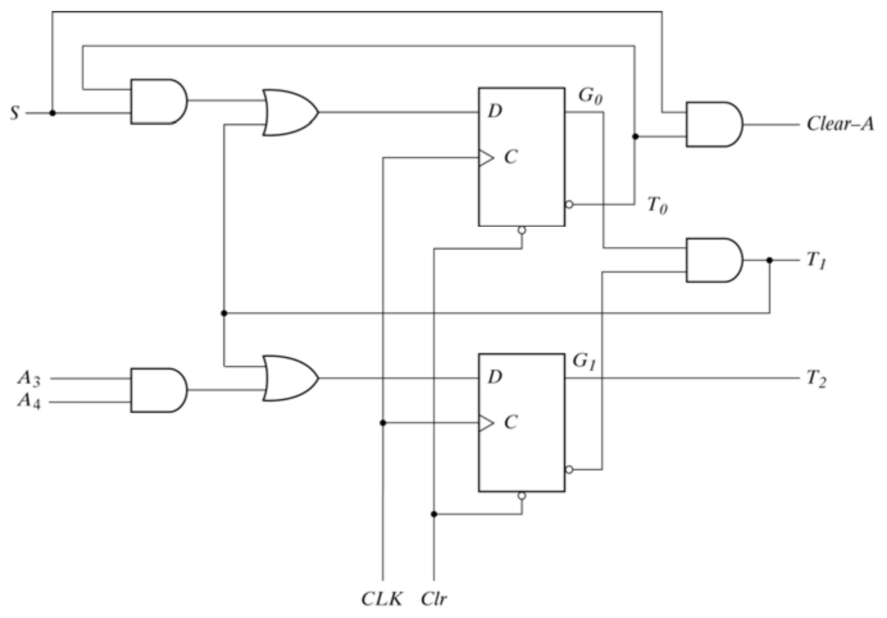


Fig. 8-12 Logic Diagram of Control

HDL Description

- The description could be one 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

```
•   //RTL description of design example 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 operatons
•   //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
```

Testing

- Note that because we used non-blocking assignment we did not have to worry about the order of the statements in every state.
- Had we used a blocking assignment, we have to worry about the order.

Testing

- //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
- endmodule

- 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);

Structural Description

Structural Description

- //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-flips
-    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

Structural 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 flipf-lops
- 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

Structural Description

```
• //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
```

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
```

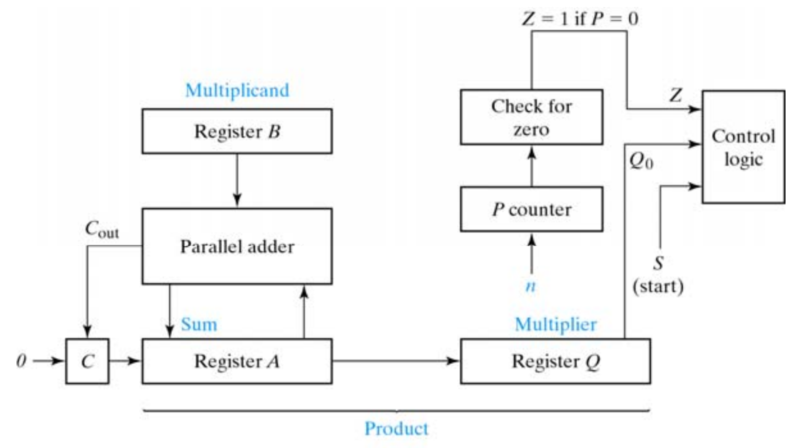


Fig. 8-13 Block Diagram of Binary Multiplier

Binary Multiplier

- Assume that the multiplicand in B, and the multiplier in Q.
- P contains *n* the length of the multiplier

Partial product is shifted one bit at a time into Q and eventually replaces the multiplier

$A \leftarrow$ shr A, $A_{n-1} \leftarrow C$

$Q \leftarrow$ shrQ,$Q_{n-1} \leftarrow A_0$

$C \leftarrow 0$

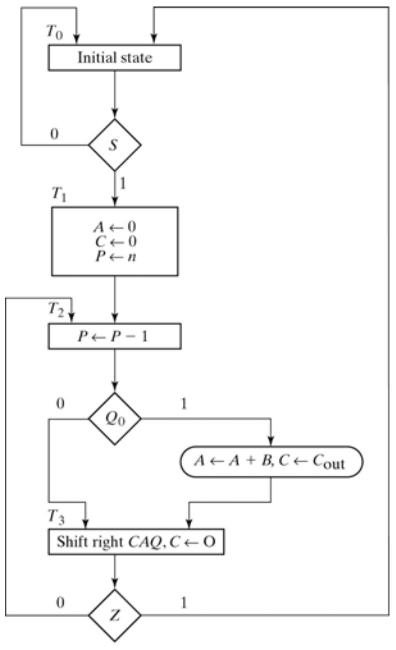
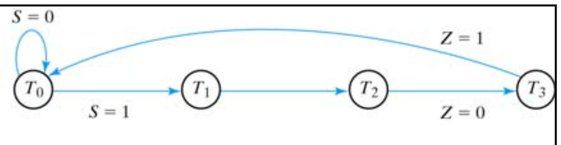


Fig. 8-14 ASM Chart for Binary Multiplier



(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

Fig. 8-15 Control Specifications for Binary Multiplier

Mistake an arrow from T_3 to T_2 if Z=0

State Table

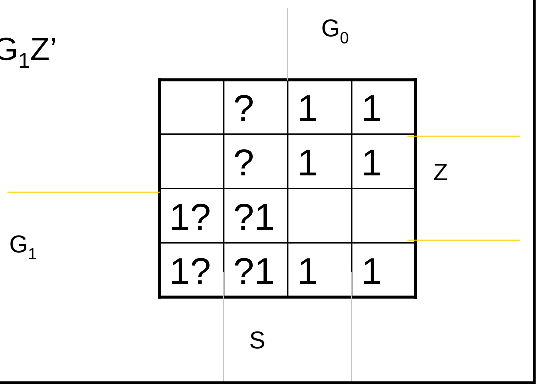
Present State		Inputs		next State		Outputs			
G1	G0	S	Z	G1	G0	T0	T1	T2	T3
0	0	0	X	0	0	1	0	0	0
0	0	1	X	0	1	1	0	0	0
0	1	X	X	1	0	0	1	0	0
1	0	X	X	1	1	0	0	1	0
1	1	X	0	1	0	0	0	0	1
1	1	X	1	0	0	0	0	0	1

Controller Design

- We can use conventional sequential circuit design for the controller, if we did using 2 D type Flip-Flops

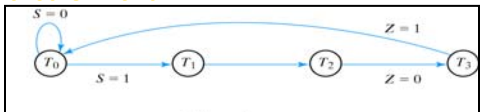
$D_{G1}=G_1G'_0 + G_0\, G'_1 + G_1Z'$

$D_{G0}=SG'_0 + G_1\, G'_0$



	?	1	1
	?	1	1
1?	?1		
1?	?1	1	1

Z is a status signal that checks P for 0.



(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

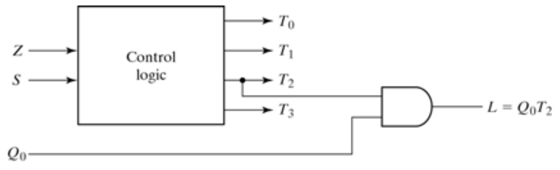


Fig. 8-16 Control Block Diagram

Sequence Register and Decoder

- If the number of variables is large, conventional design is difficult.
- Need specialized methods for the control design.
- Uses a register to control the states, and a decoder to provide an output corresponding to each of the states.
- A register with n flip-flops can have up to 2^n states, and n-to-2^n line decoder has up to 2^n outputs.

Sequence Register and Decoder

- The circuit could be obtained directly from the table by inspection (keep in mind that the states are available as inputs).
- Directly from the table, there are three 1's for G_1, which means

$$D_{G1} = T_1 + T_2 + T_3\overline{Z}$$
$$D_{G0} = T_0 S + T_2$$

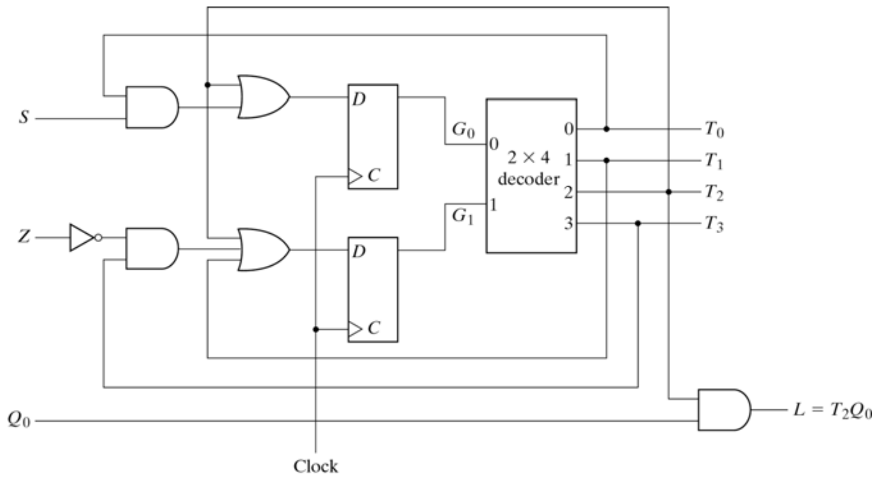
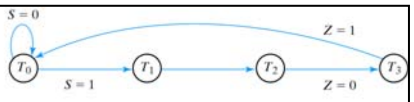


Fig. 8-17 Logic Diagram of Control for Binary Multiplier Using a Sequence Register and Decoder

One Flip-Flop per State

- We need n flip-flops for every state
- In this case, we need 4 flip-flops.
- The circuits are very simple to implement and can be obtained directly from the state diagram.
- For example, we move from state 0 to 1 if S=1 which means $D_{T1}=T_0S$

One Flip-Flop per State



(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

$$D_{T0} = T_0\overline{S} + T_3Z$$
$$D_{T1} = T_0S$$
$$D_{T2} = T_1 + T_3\overline{Z}$$
$$D_{T3} = T_2$$

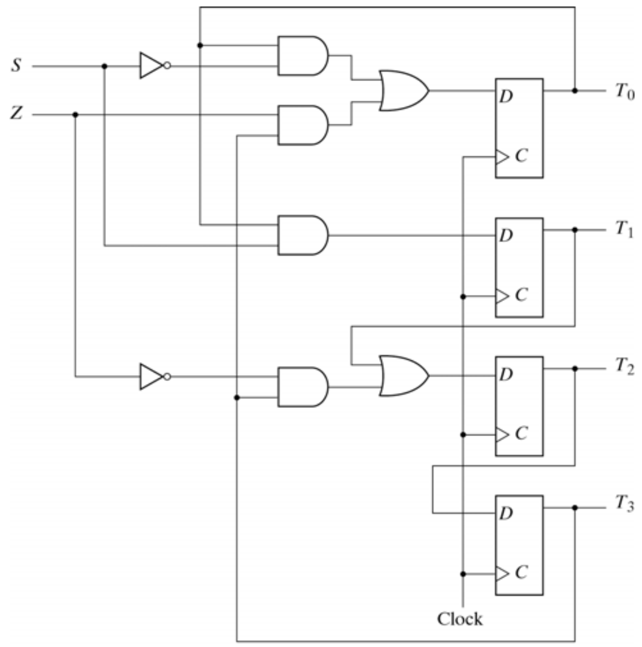


Fig. 8-18 One Flip-Flop Per State Controller

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

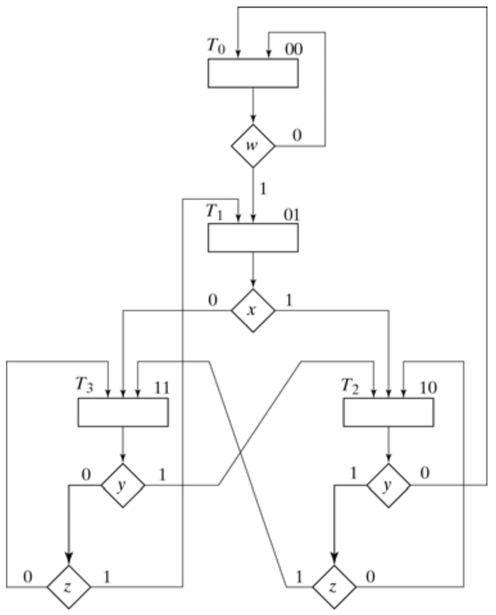


Fig. 8-19 Example of ASM Chart with Four Control Inputs

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’zy=y’
1	1	1	1	y’z’		

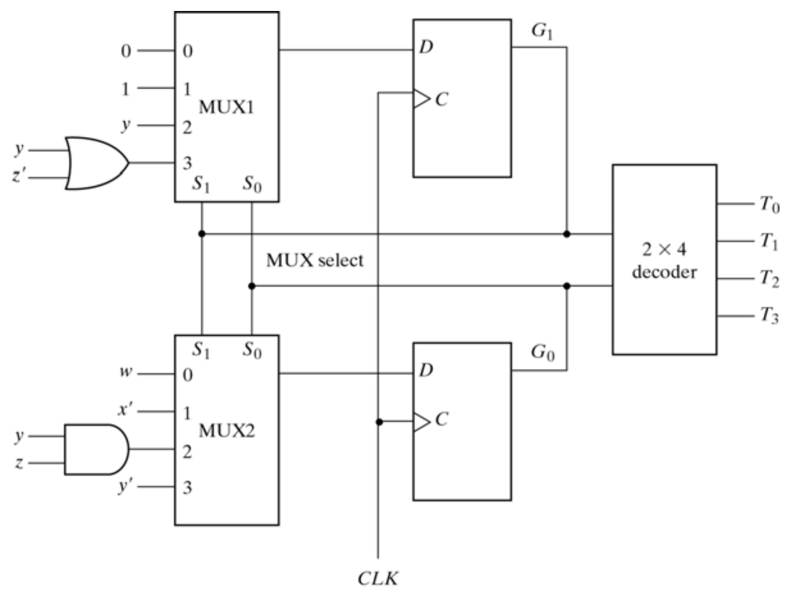


Fig. 8-20 Control Implementation with Multiplexers

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).

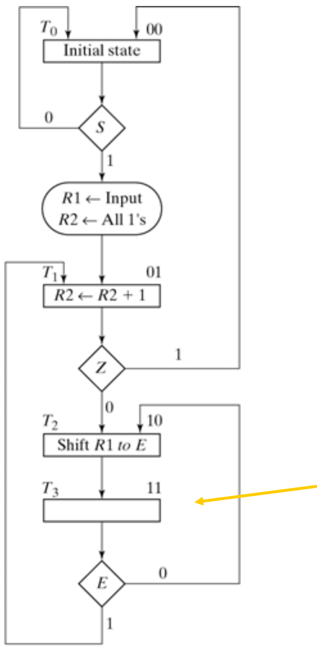


Fig. 8-21 ASM Chart for Count-of-Ones Circuit

?

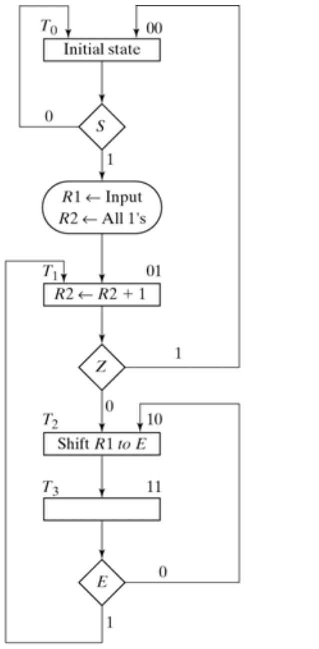


Fig. 8-21 ASM Chart for Count-of-Ones Circuit

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

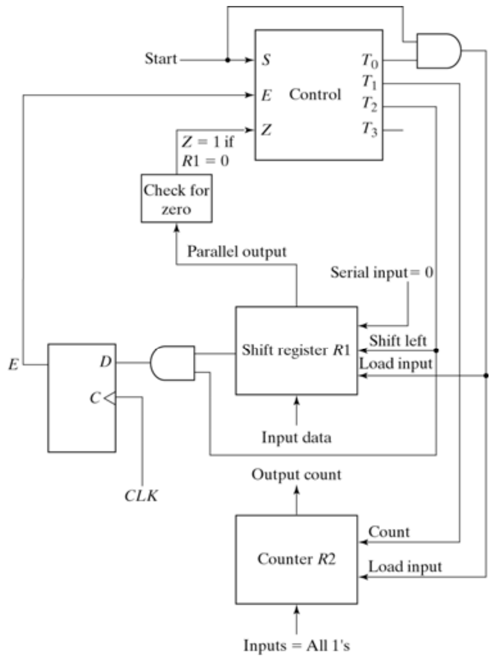


Fig. 8-22 Block Diagram for Count-of-Ones

Control (counting of 1's)

Present		Next		Conditions	MUX inputs	
State		State				
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

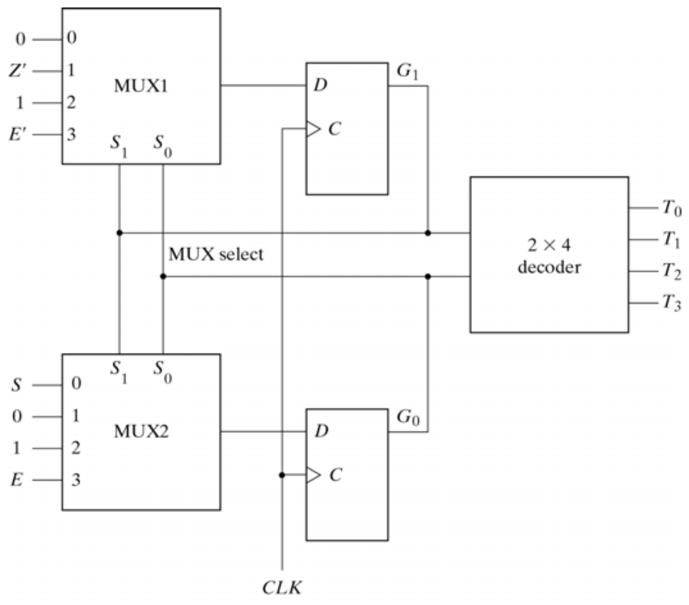


Fig. 8-23 Control Implementation for Count-of-Ones Circuit

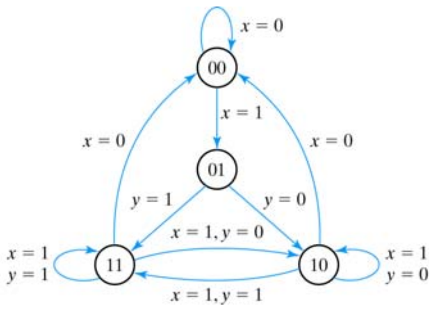


Fig. P8-10 Control State Diagram for Problems 8-10 and 8-11

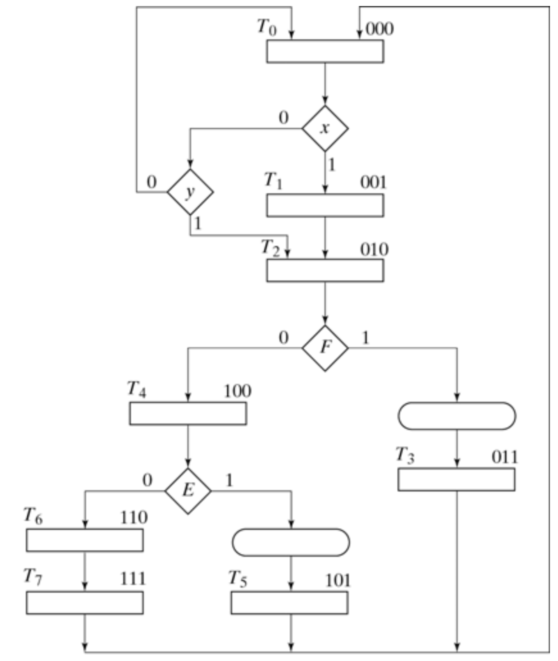


Fig. P8-20 ASM Chart for Problems 8-20