Register Transfer Level

- Something between the logic level and the architecture level
- A convenient way to describe synchronous sequential systems
- State diagrams for pros

Hierarchy of Designs

- The design of a digital system happens in many different levels of abstraction
  - Physical level
  - Electronic level
  - Logic level
  - Architectural level
  - System level

Theory-Practice

- In theory, theory and practice are the same.
- In practice they are not
- Real systems tend to have many states with many transitions that depend on many inputs
- Simple state diagrams are not enough; we need a more powerful language

Modularity

- Using better notation is not enough
- We have to use another design concept called modularity
- We partition our design into many modules each one with its own specifications

Advantages

- Division of labor among members of the team
- Reuse the design of the modules in other systems
- Keep the size manageable.

They are not unrelated

- Modules and hierarchies of abstraction are not unrelated
- They both try to hide unnecessary information.

What we do here

- Design simple modules
- Implement our designs in the logic level
- Explore various possibilities

Components

- Registers (collections of F-F)
- Operations on registers (using combinational circuits)
- Control (someone to boss everything around)

Registers

- Registers are collections of F-F that can execute LOAD and other operations
- Sometimes the other operations are incrementing, decrementing, shifting, etc
- Sometimes these operations are done with the help of separate combinational circuits

For Example

- A 16-bit incrementer requires 16 half adders
  - more hardware if we seek efficiency
- If we need two counters that do not increment at the same time we might decide to let them share the incrementer.

Transfers

- This is done by transfering the data to the incrementer
- and then transfering the output of the incrementer back to the register

Transfer Notation

- We use statements like
  - R1 <= R2
- e.g. R2 is copied into R1
- The transfer may be conditional
  - if (T1=1) then (R1<=R2)

Anything Goes

  - R1 <= R2+R3
  - R3 <= R3+1
  - R4 <= shl R4
  - R5 <= 0
- The similarity with Verilog is obvious...

Like Verilog

- We want to describe
  - Transfer operations
  - Arithmetic operations
  - Logic operations
  - Shift operations

Clocked Transfer

- The transfer happens only at the edge of the clock
- Before the clock the combinational circuits are computing the input to the F-F
- After the clock we are computing the F-F input for the next state.

All Transfers at Once

- Since all transfers happen at once
- The natural procedural assignment is the non-blocking one:
  - R1 <= R2
  - R2 <= R1

Loops

- There are two uses of the loops:
  - describe test benches
  - describe repeated hardware

Example: for loop

```
module decoder (IN, Y);
   input  [1:0] IN;    //Two binary inputs
   output [3:0] Y;    //Four binary outputs
   reg [3:0] Y;
   integer I;          //control variable for loop
   always @ (IN)
        for (I = 0; I <= 3; I = I + 1)
          if (IN == I) Y[I] = 1;
          else Y[I] = 0;
endmodule
```

The Equivalent

- The **for** loop can be replaced by
  - if (N=00) F[0]=1; else F[0]=0;
  - if (N=01) F[1]=1; else F[1]=0;
  - if (N=10) F[2]=1; else F[2]=0;
  - if (N=11) F[3]=1; else F[3]=0;

Synthesis

- The hardware compiler should know how to do:
  - B = A+C // addition
  - **assign** Y = S ? I1 : I0 // 2-1 MUX
  - **case** ... // larger MUX
  - **always** @ (**posedge** ...) // edge triggered F-F

The Design Process

- Like any design process, we have two phases
  - Compose
  - Verify
- Composition is (nowadays) done in a HDL
- Verification is done (mostly) with simulations

Various Simulations

- A simulation is just an approximation of the reality
- There are several kinds of approximations
  - RTL simulations
  - Gate level simulations
  - Electronic level simulations

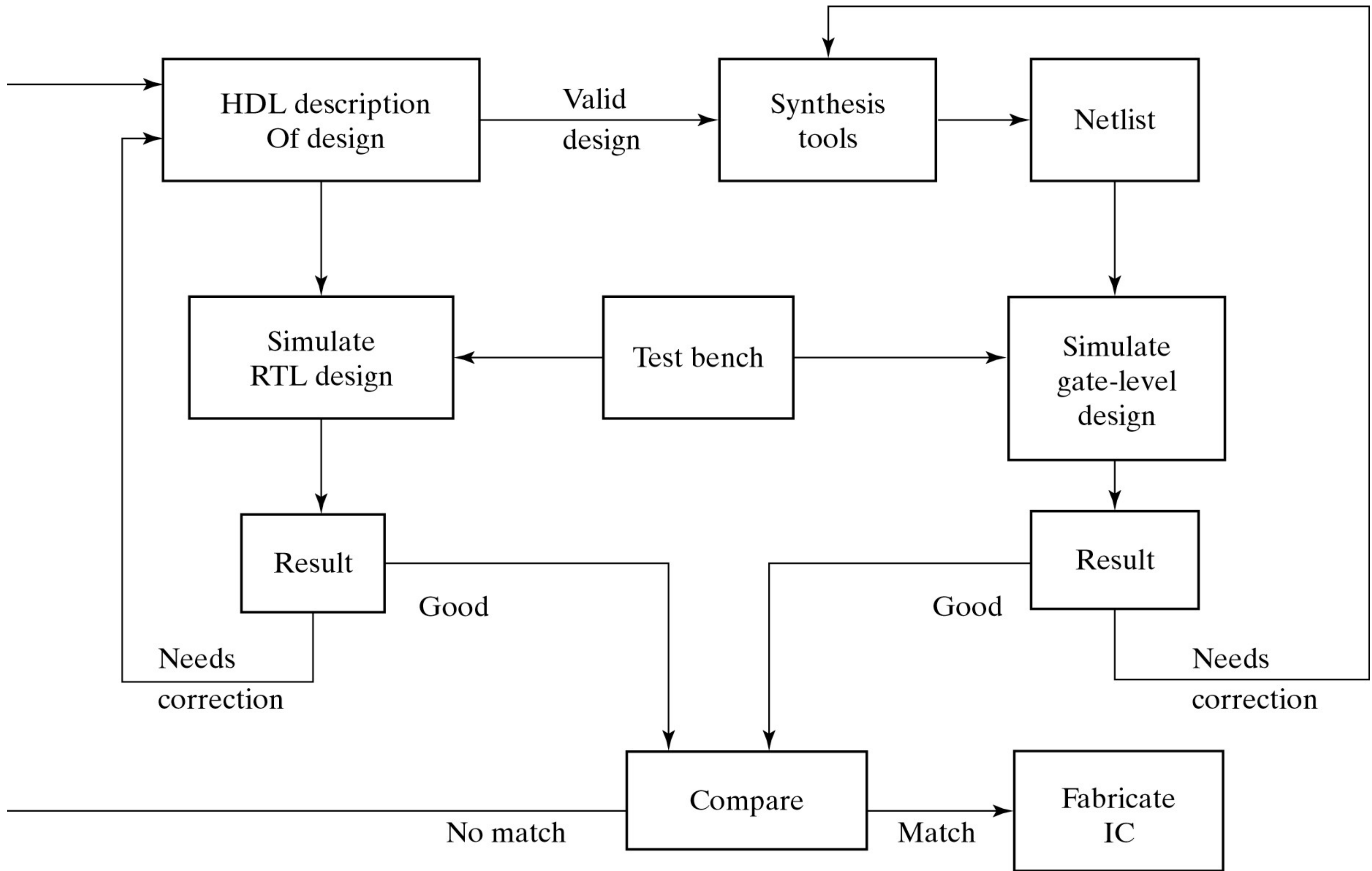


Fig. 8-1 Process of HDL Simulation and Synthesis

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Algorithmic State Machines

- Synchronous sequential circuits can be thought of as having two parts
  - The *data* part that is concerned with the processing of the contents of the registers
  - The *control* part that is concerned with the sequencing of states

The Datapath

- Contains all the registers
- All the arithmetic etc logic that operates on the data
- All the outputs (to the world and to the logic unit)
- Receives commands from the control unit

Control Unit

- Contains all the “state” F-F
- All the logic to decide the next state
- Receives feedback from the data path
- Generates commands for things to happen in the datapath or just outputs its state

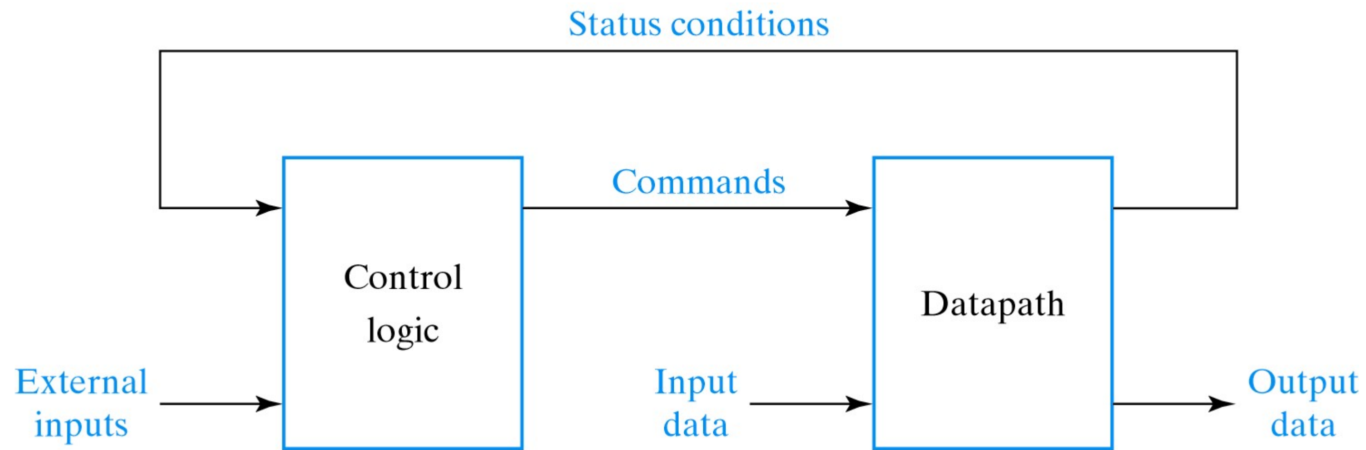


Fig. 8-2 Control and Datapath Interaction

© 2002 Prentice Hall, Inc.

ASM

- Part of the work to be done is to
  - Define a set of states
  - The operations that take place in every state
  - And the transitions between the states
- All this in a way that they solve a problem
- This is called an ASM

ASM Charts

- An ASM can be described with a HDL
- Can also be described with a kind of flowchart
  - Similar to s/w flowcharts, but adapted to hardware

State Box

- Represents a state (of course!)
- Contains
  - The symbolic state name
  - The binary state name (if available)
  - Any number of unconditional operations

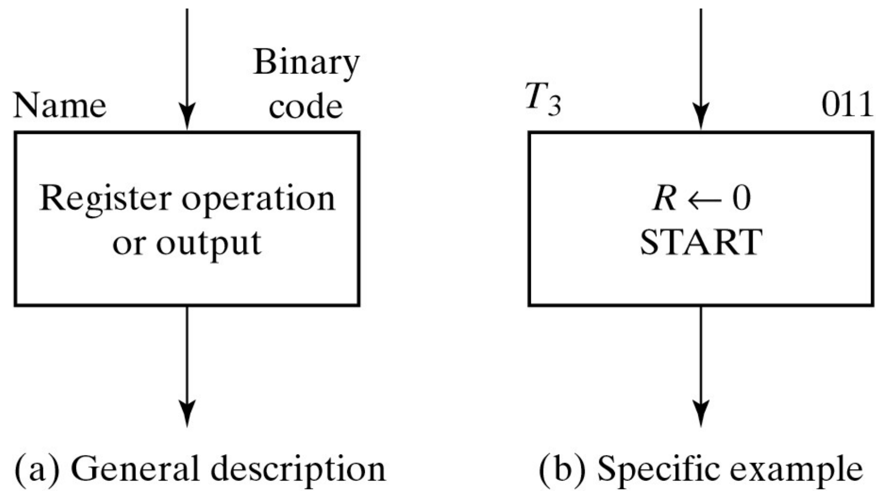


Fig. 8-3 State Box

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Decision Box

- Represents a decision (of course!)
- Normally a binary decision
- Has one incoming arrow and two outgoing
- The condition is written inside
- The outcome on the outgoing arrows

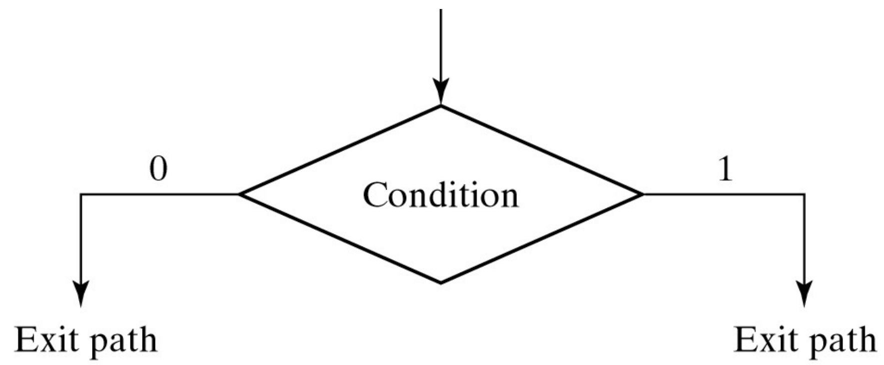
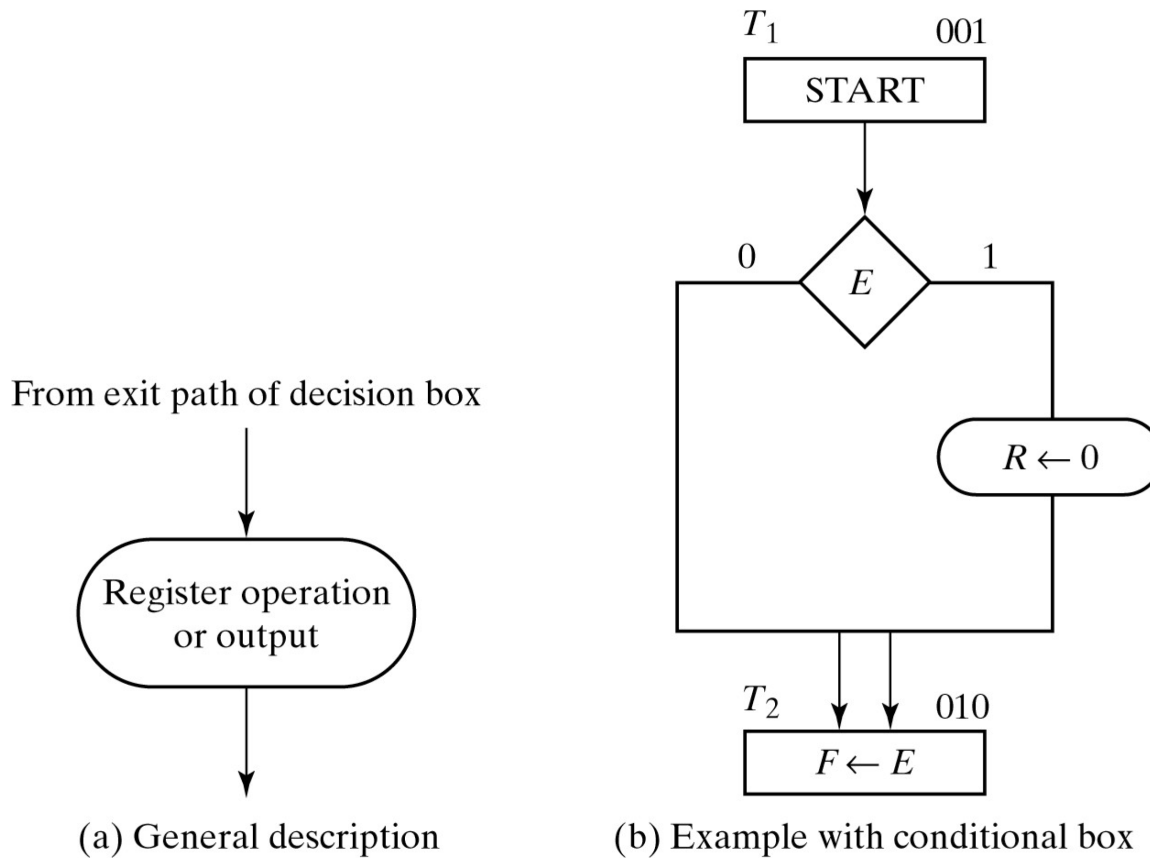


Fig. 8-4 Decision Box

© 2002 Prentice Hall, Inc.

Conditional Box

- Represents a conditional statement (of course!)
- Always follows a decision box
- Contains the operations that will be executed if we reach it



(a) General description

(b) Example with conditional box

Fig. 8-5 Conditional Box

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

ASM Block

- It is not what happens to an author that cannot compose an ASM chart
- Represents a complete state
- Contains one state box
- And all associated decision and condition boxes

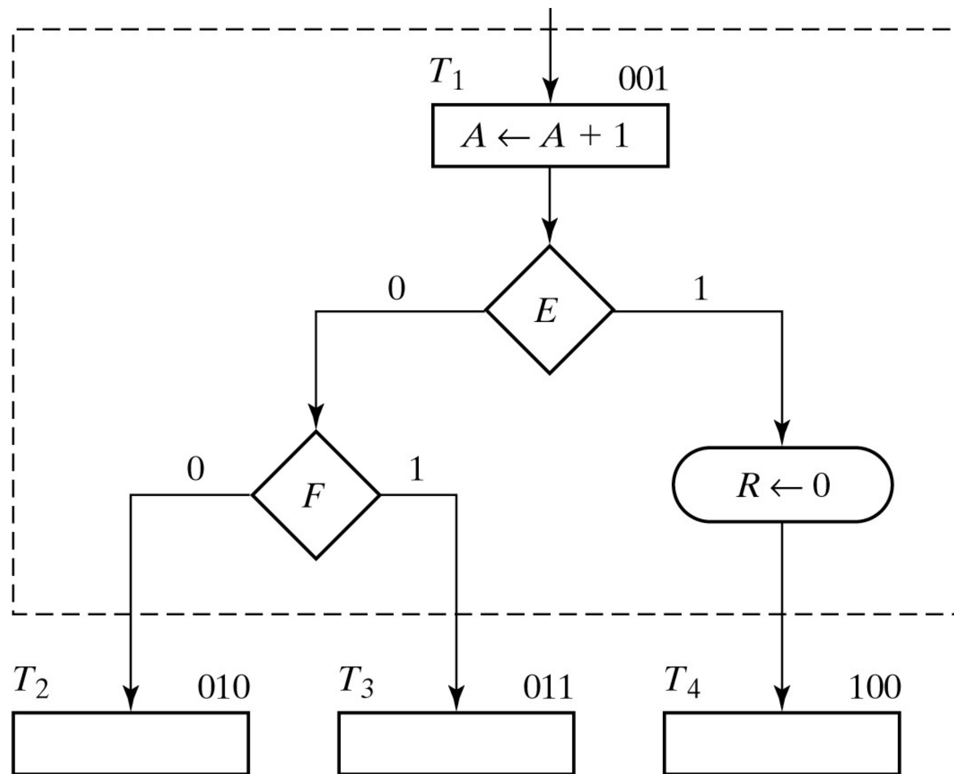


Fig. 8-6 ASM Block

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

State Diagram

- A state diagram does exactly the same as an ASM chart
- The ASM chart is better suited for real problems that may have more detail
- The diagram that follows does the same as the chart before

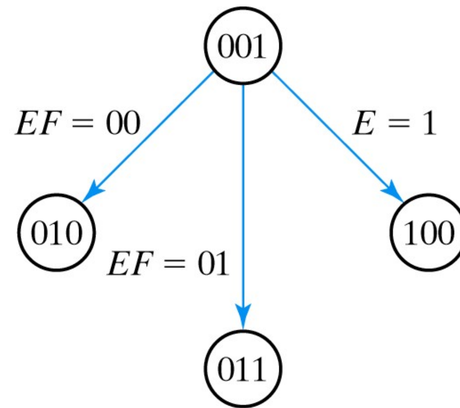


Fig. 8-7 State Diagram Equivalent to the ASM Chart of Fig. 8-6

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Timing

- All F-F, both state F-F and registers are connected to a common clock and triggered in the same fashion.
- All operations within an ASM block take place at once, but the results are stored in the F-F at the clock edge

As a Result

- Max one assignment per F-F per ASM block
- The F-F do not change value in between clock edges
- A state is the time between the (triggering) clock edges

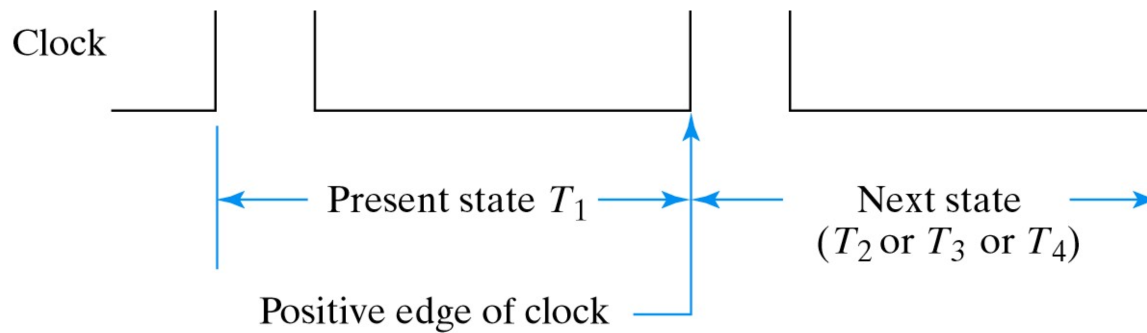


Fig. 8-8 Transition Between States

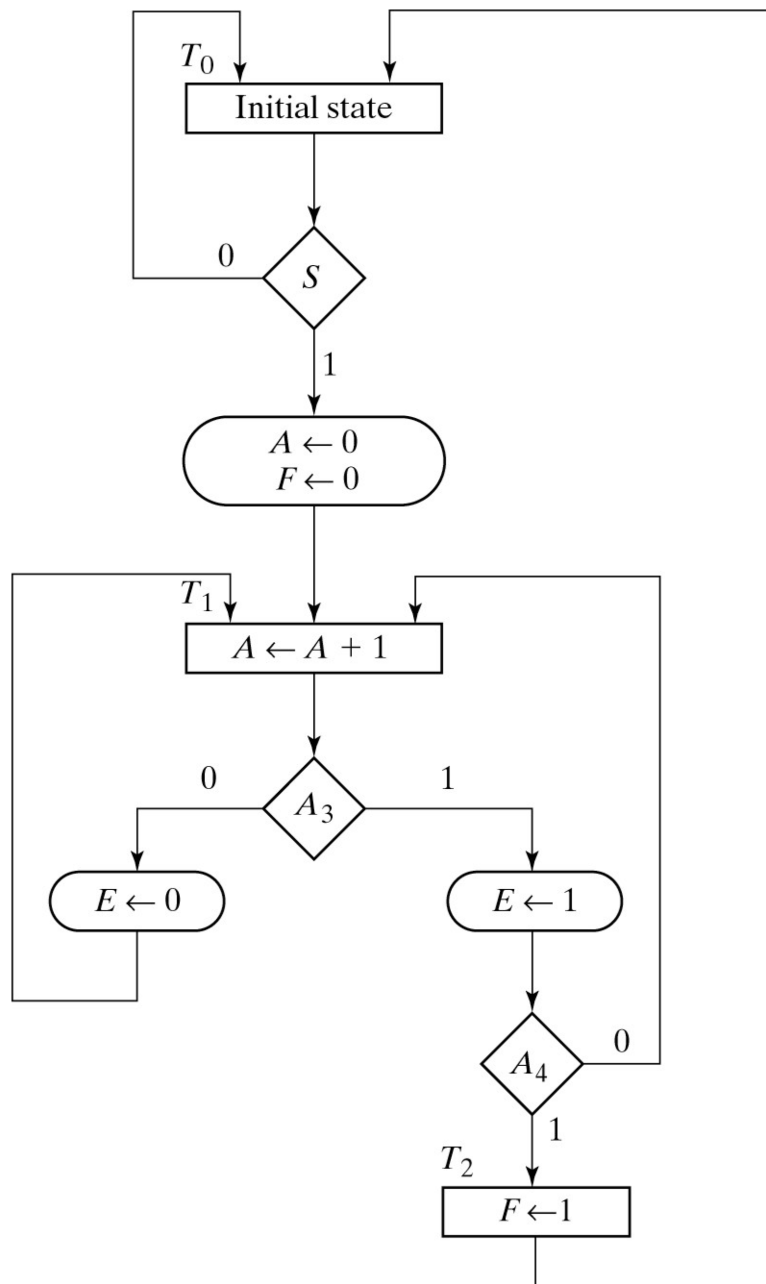
© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Design Example

- Design a sequential circuit that has
  - A counter: A[4:1]
  - Two F-F: E and F
  - An input S
- When the system is in the initial state and the input S becomes 1 the system goes in the counting state and F and A are reset

Design Example

- If the input S is 0 and the system is in the initial state, it remains in the initial state
- If in the counting state, E is set is A[3] is 1, o/w reset
- If in the counting state, we go to the output state if A[3:4]==11



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Fig. 8-9 ASM Chart for Design Example

Design the Datapath

- We can design it with state tables etc
- We can design it in an *ad hoc* fashion
  - This often the best

Assumptions

- We have:
  - A counter A with <u>synchronous</u> clear
  - Two flip-flops E and F of the J-K variety
  - An input S
- And also
  - The system has three inputs T0, T1, T2 that correspond to the three states

Let's Fry Some Bits

- The counter counts when in state T1
- The counter is reset when in state T0 and S=1

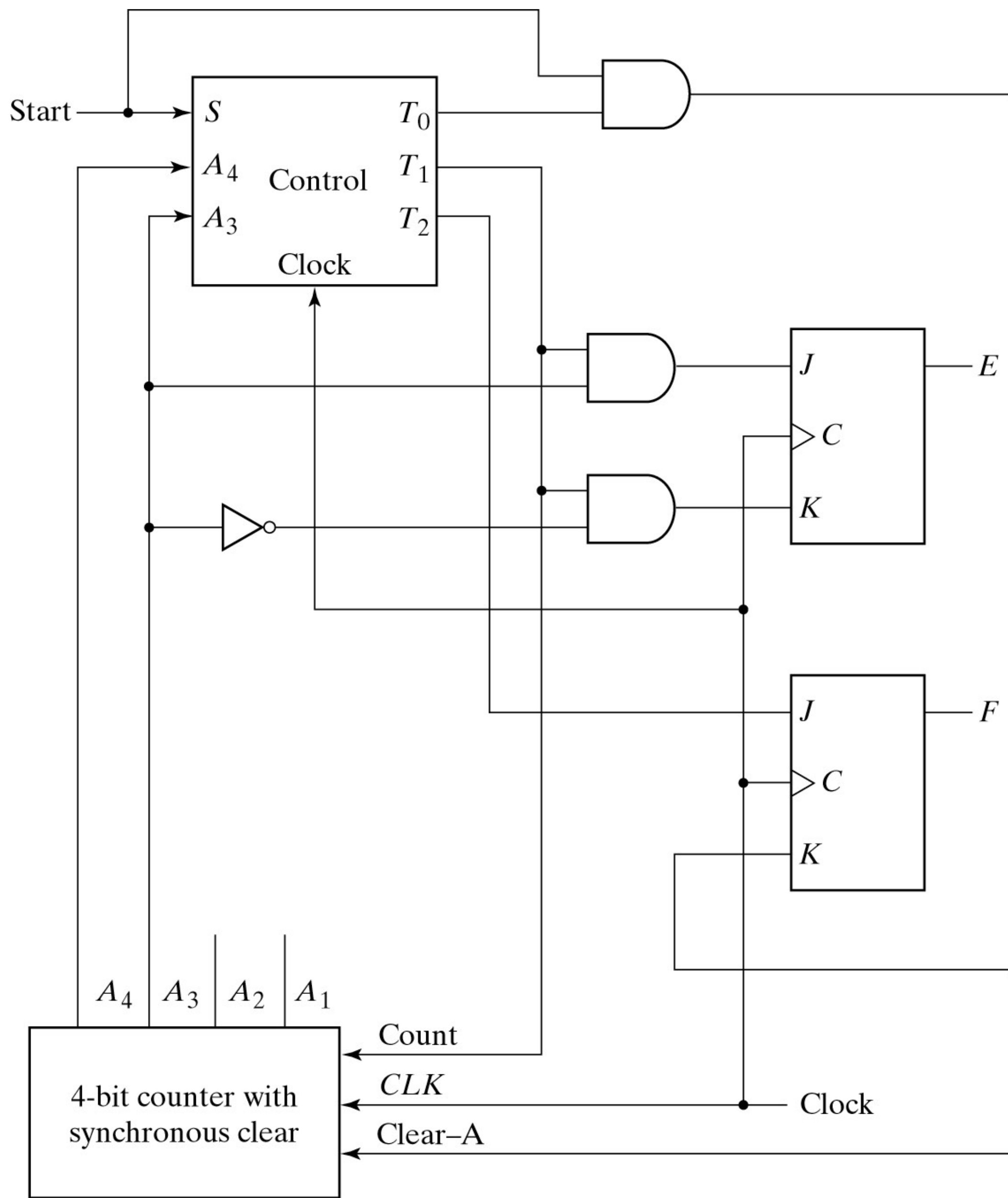


Fig. 8-10 Datapath for Design Example

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Fry more Bits

- F-F E is set when in state T1 and A3=1
- F-F E is reset when in state T1 and A3=0
- F-F F is set when in state T2
- F-F F is reset when in state T0 and S=1

Control Logic

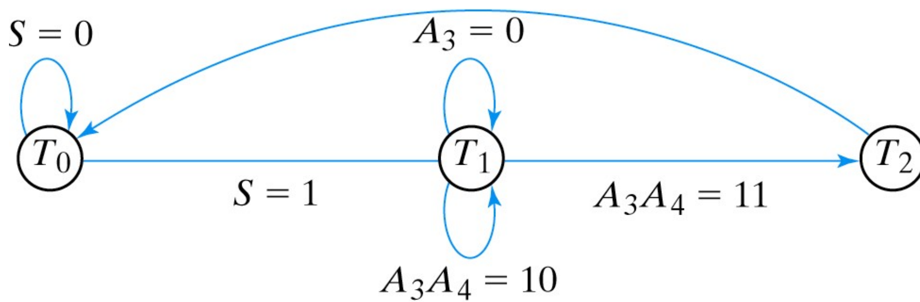
- We will see a few techniques to design the control logic
- There are many we will not see:
  - Either proprietary
  - Or historical

State Table

- We need the state table
- The state table can be big
- For this abysmally small example it has 32 entries
- We compress it

The Table

Present State		Inputs			Next State		Outputs		
G1	G0	S	A3	A4	G1	G0	T1	T2	T3
0	0	0	x	x	0	0	1	0	0
0	0	1	x	x	0	1	1	0	0
0	1	x	0	x	0	1	0	1	0
0	1	x	1	0	0	1	0	1	0
0	1	x	1	1	1	1	0	1	0
1	1	x	x	x	0	0	0	0	1



(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

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

The logic

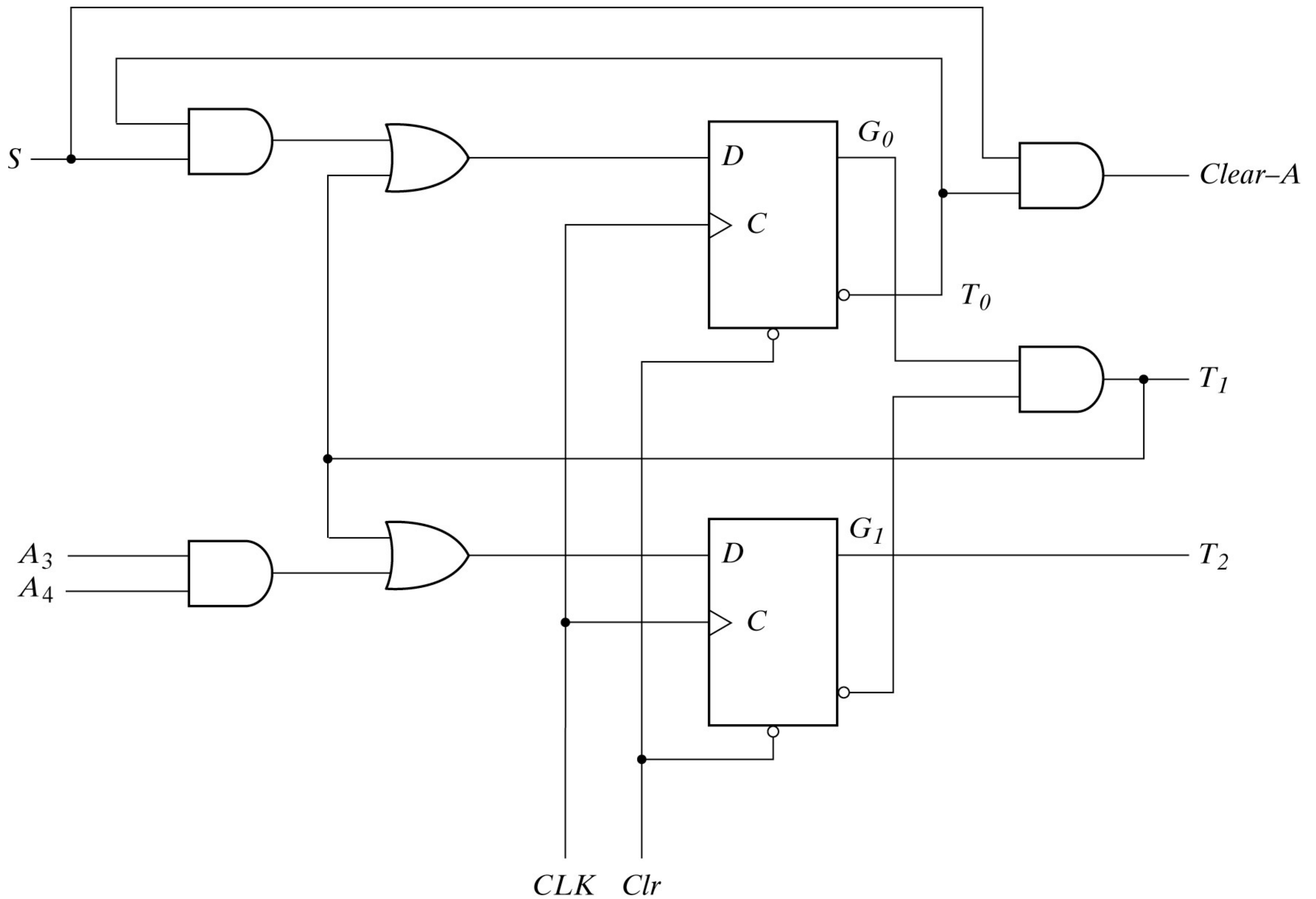
- With a bit of symbolic manipulation
  - D1 = G1’ G0 A3 A4
  - D0 = G1’ G0 + S G1’ + G1 G0’ + S G0’
- And
  - T0 = G1’ G0’
  - T1 = G1’ G0
  - T2 = G1 G0

With Maps

- The F-F inputs are
  - $D1 = G1' G0 A3 A4$
  - $D0 = G1'G0 + G0'S$
- And the outputs are
  - $T0 = G0'$
  - $T1 = G1'G0$
  - $T2 = G1$

With Intuition

- The F-F inputs are:
  - D1 = T1 A3 A4
  - D0 = T0 S + T1
- And the outputs are
  - T0 = G0'
  - T1 = G1'G0
  - T2 = G1



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Fig. 8-12 Logic Diagram of Control

With Verilog

- Behavioral description
  - the most abstract
  - h/w compiler is the king
- Structural description
  - we have to do the work
  - we give all the detail

The Preliminaries

```
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;
```

The Control Section

```
//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
```

The Register Transfer Logic

```
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
```

Multiplier

- An extremely useful thing
- An astonishingly complex thing to do if speed is important
- An outrageously tricky thing to do with floating point arithmetic

Back to Elementary School

- Let’s multiply 23 by 19
    - `     10111`
    - `  x  10011`
    - `----------`
    - `     10111`
    - `    10111`
    - ` 10111`
    - `----------`
    - ` 110110101`

Still in Elementary School

- The first number is called multiplicant
- The second is called multiplier
- The result is called product

Register Configuration

- We store the multiplicant in Reg. B
- We store the multiplier in Reg. Q
- Reg. Q will be shifted out to oblivion
- The product will be stored half in the accumulator A and half in Q

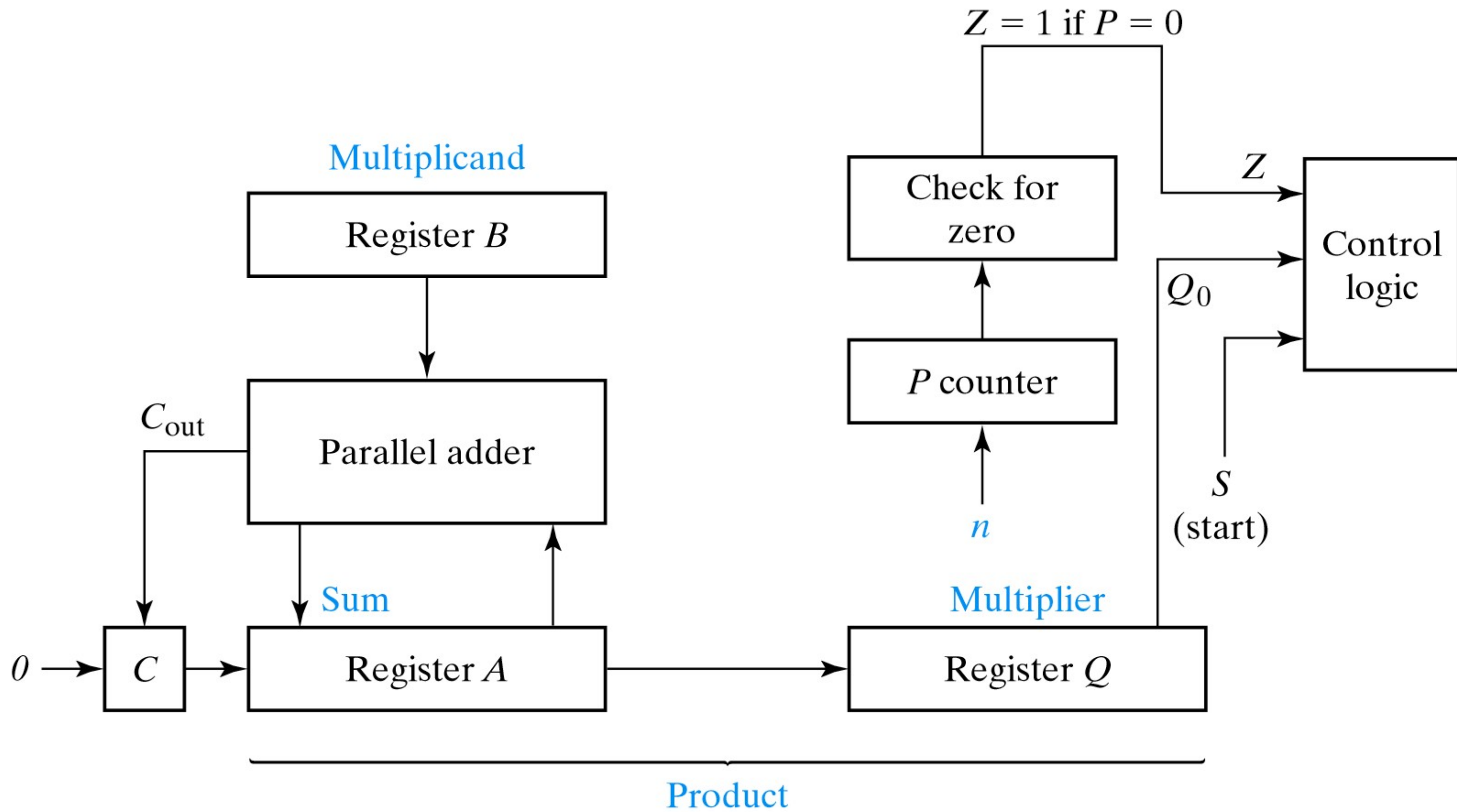
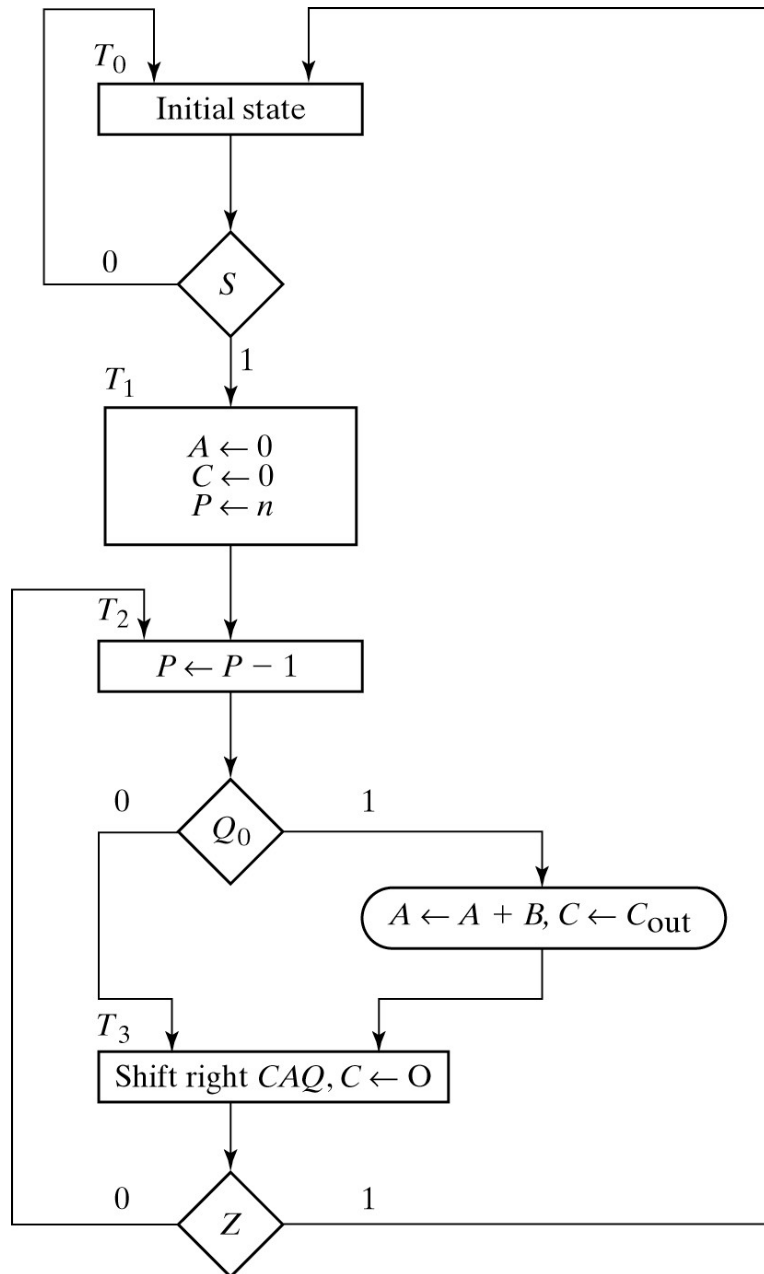


Fig. 8-13 Block Diagram of Binary Multiplier

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

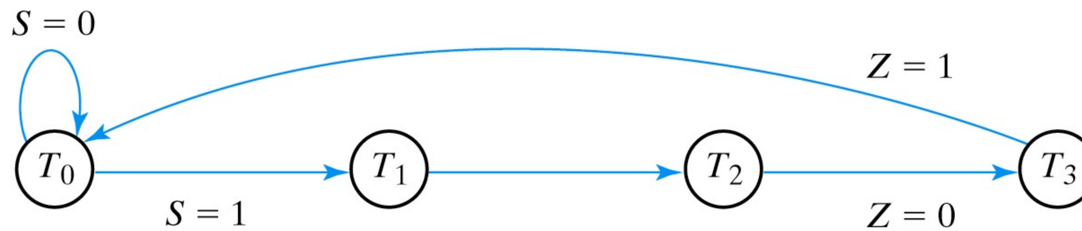
Fig. 8-14 ASM Chart for Binary Multiplier

Control Logic

- We have three choices
  - Binary
  - Gray Code
  - One-hot (1 F-F per state)
- We have three inputs:
  - S, Q0(?), Z

We know this stuff...

- We can distill the ASM chart into a simple state diagram
- We do the tables
- Simplify
- And we are done



(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

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

How it looks like

- Q0 can be dealt with separately since it does not affect the sequence of states but only the commands to the datapath (e.g. the output of the control logic)

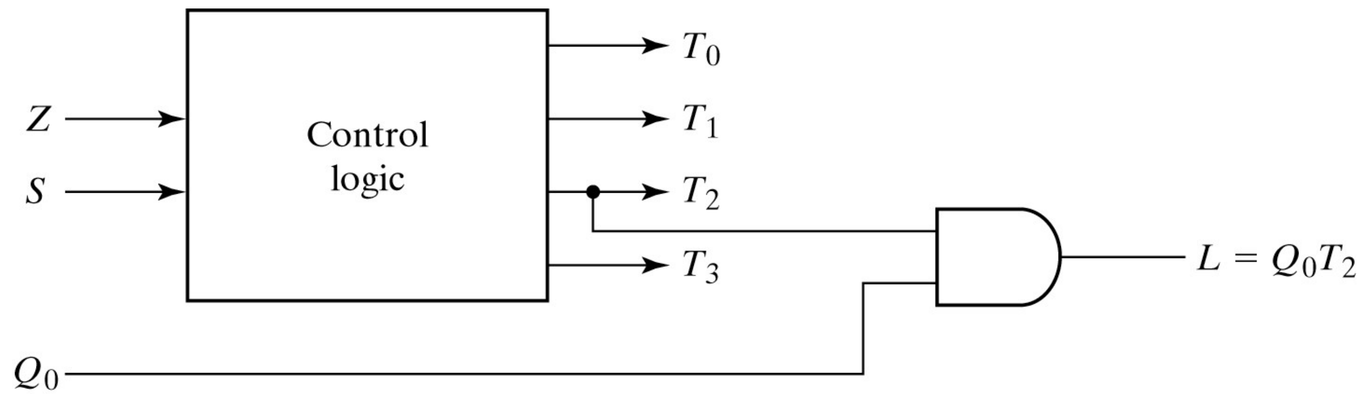


Fig. 8-16 Control Block Diagram

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

The Table

Present State		Input		Next State		Output			
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

Simplify

- Assuming binary state assignment
  - D1 = G1' G0 + G1 G0' + G1Z'
  - D0 = G1 G0' + G0'S
- Assuming Gray state assignment
  - D1 = G0 + G1Z'
  - D0 = G1 G0 + G1'G0'S

Using the Outputs

- We need a decoder for this since there is not much minimization for the output circuits
- Plugging the outputs in
  - D1 = T1 + T2 + T3Z’
  - D0 = T2 + T0 S
- Does not matter if we use Gray or binary

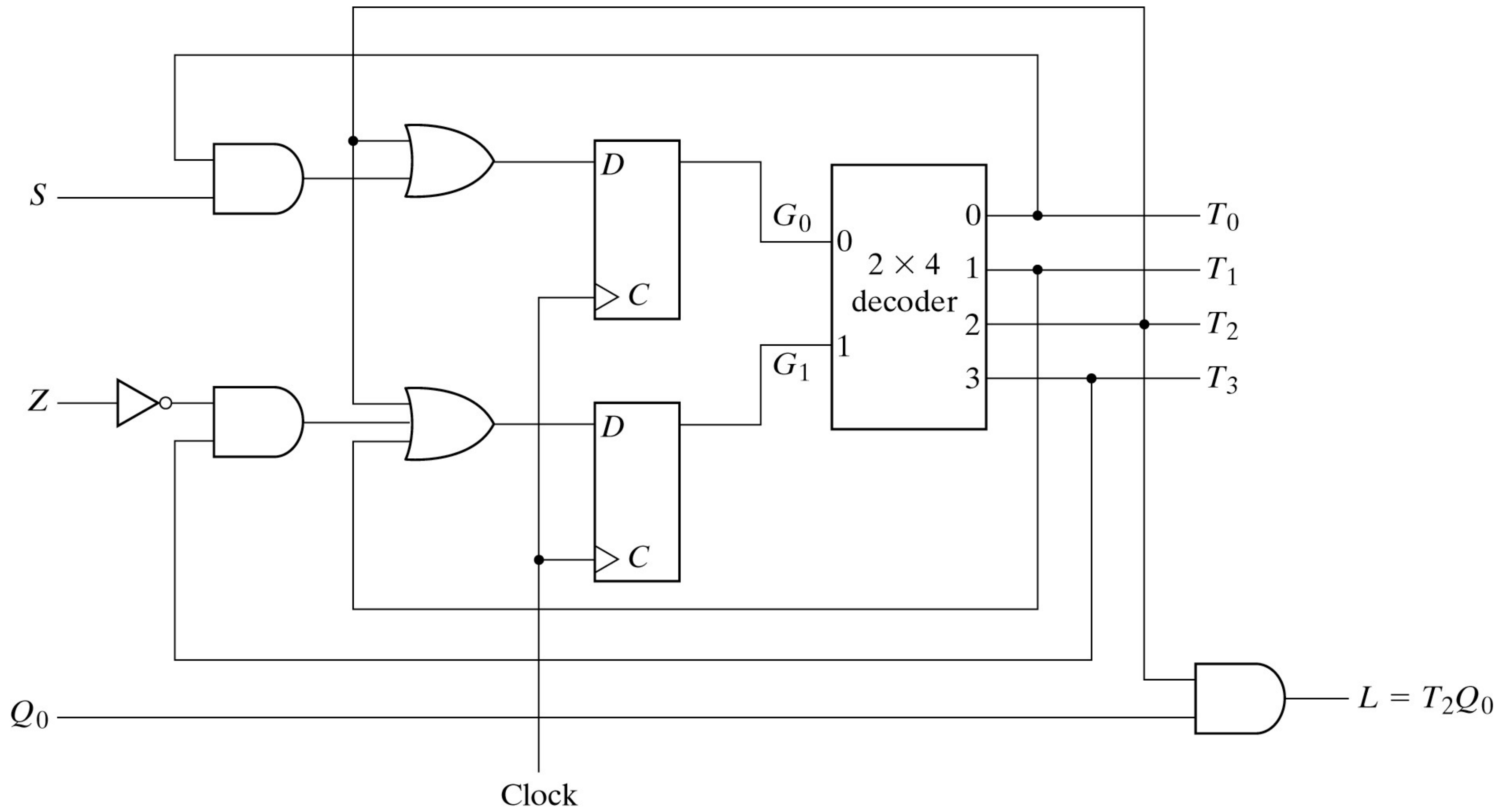


Fig. 8-17 Logic Diagram of Control for Binary Multiplier Using a Sequence Register and Decoder

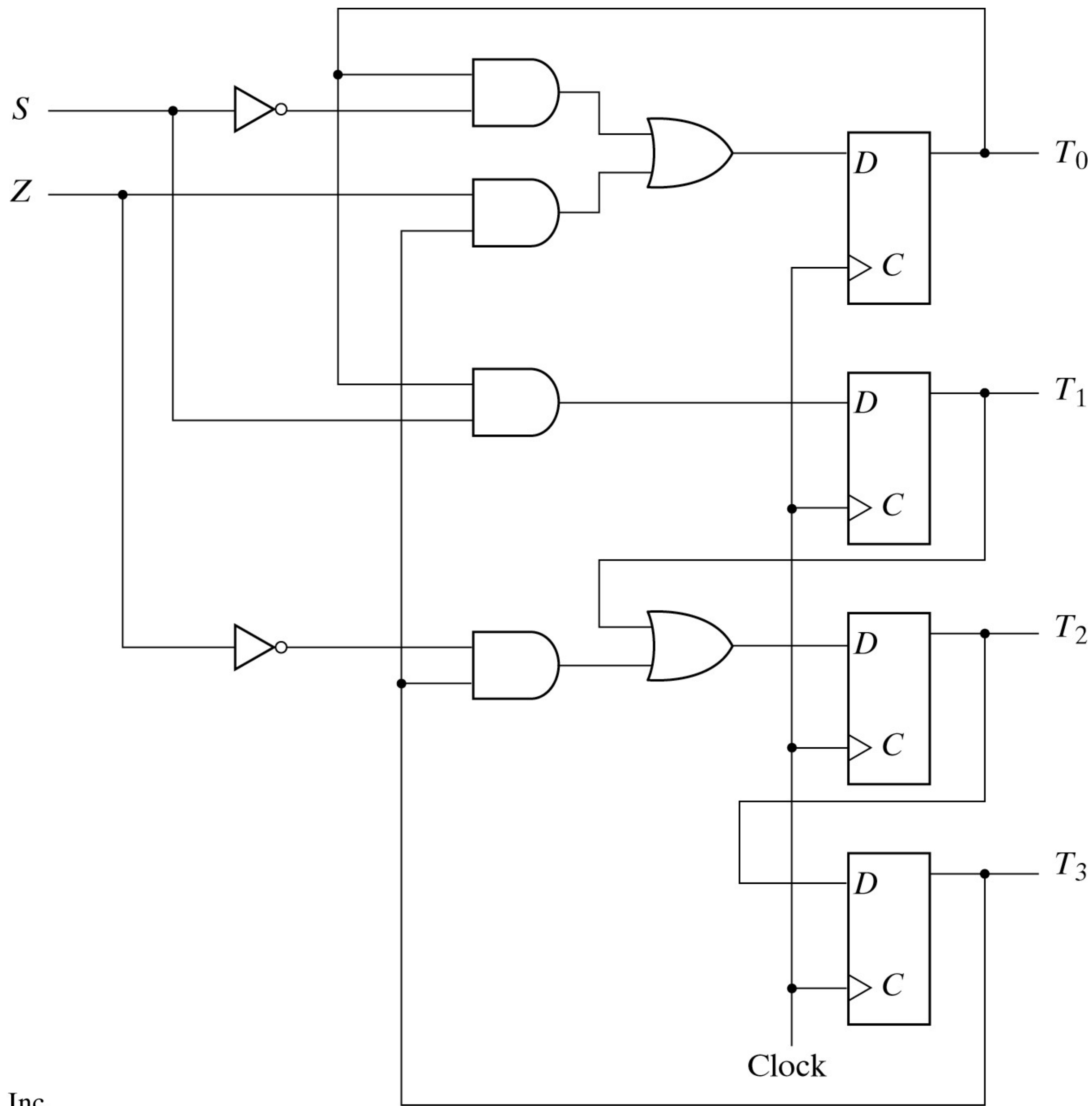
© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Some Observations

- We cannot always avoid a full decoder
- Decoders need one gate per state
- Control logic does not always offer much of a chance for simplification

One F-F per State

- It is not always as costly as it looks
- Permits direct implementation from the ASM chart
- It makes sense!



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

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

HDL for Multiplier

```
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 Transitions

```
always @(negedge CLK or negedge Clr)
   if (~Clr) pstate = T0;
   else pstate <= nstate;
 always @(S or Z or pstate)
   case (pstate)
     T0: if (S) nstate = T1;
     T1: nstate = T2;
     T2: nstate = T3;
     T3: if (Z) nstate = T0;
         else   nstate = T2;
   endcase
```

Register Transfer

```
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
```

Design with MUX

- Allow us to minimize the number of components
- Quite simple to do.

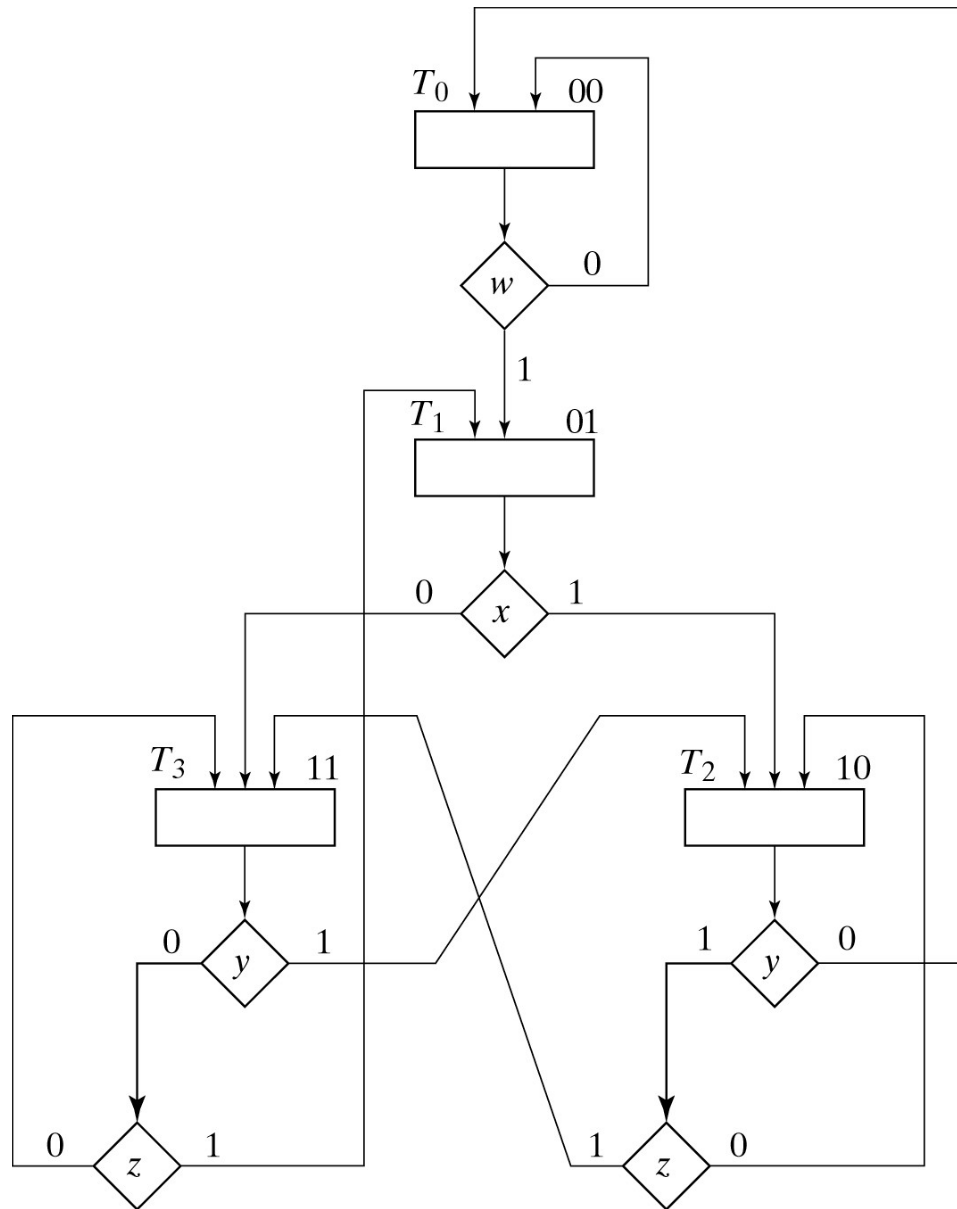


Fig. 8-19 Example of ASM Chart with Four Control Inputs

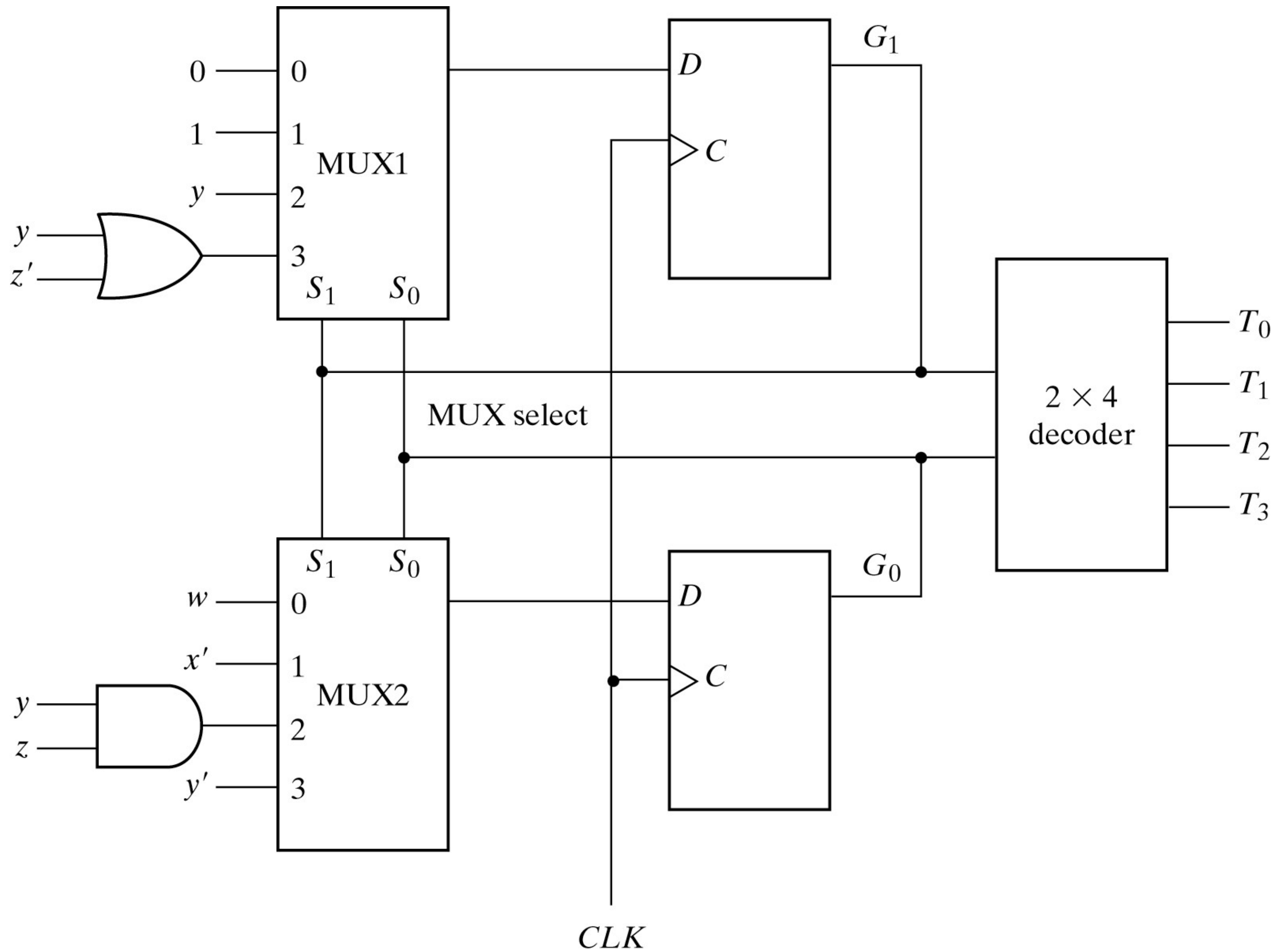
© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

The Advantages

- MUXes allow us to break the combinational circuit in many simpler smaller ones
- Are among the few ways to design a 3 level combinational circuit
- Really useful when we have many variables and not all of them affect every state.

The Table

Present State		Next State		Input Conditions	MUX 1	MUX 2
G1	G0	G1	G0			
0	0	0	0	w'	0	w
0	0	0	1	w		
0	1	1	0	x	1	x'
0	1	1	1	x'		
1	0	0	0	y'	yz'+yz=y	yz
1	0	1	0	yz'		
1	0	1	1	yz		
1	1	0	1	y'z	y+y'z' = y+z'	y'
1	1	1	0	y		
1	1	1	1	y'z'		



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Fig. 8-20 Control Implementation with Multiplexers

With One-Hot

- We do this with the tried and true method (glare at the chart til your eyes get dry)
  - $T0 = T0\, w' + T2\, y'$
  - $T1 = T0\, w + T3\, y'z$
  - $T2 = T3\, y + T1\, x + T2\, yz'$
  - $T3 = T3\, y'z' + T1\, x' + T2\, yz$

Count the Ones

- Design a circuit that counts the ones in a register R1 and stores the result in counter R2
- The circuit keeps counting and shifting the contents of R1 out till R1 is all zeros
- R2 is initialized to all 1s.

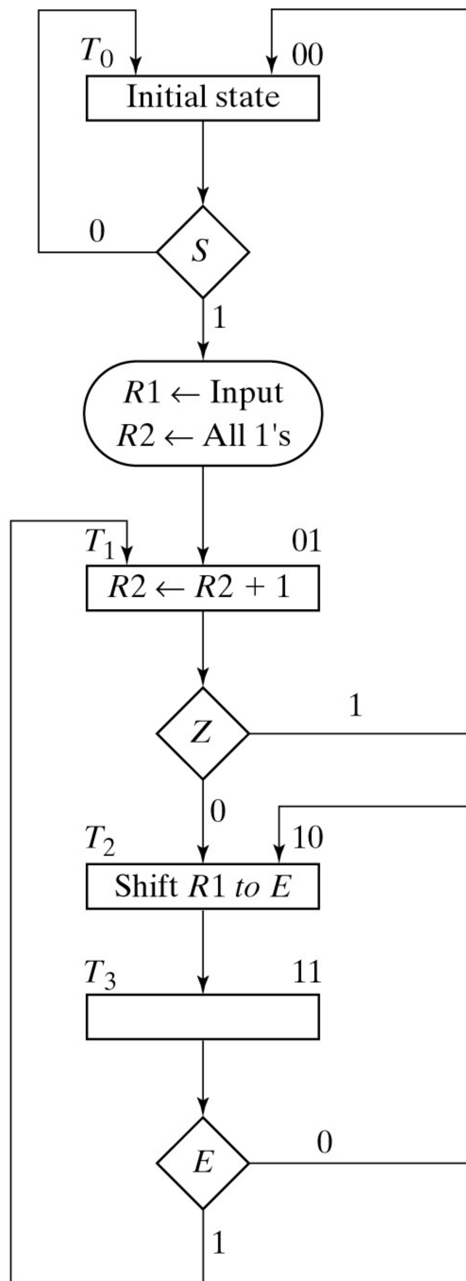
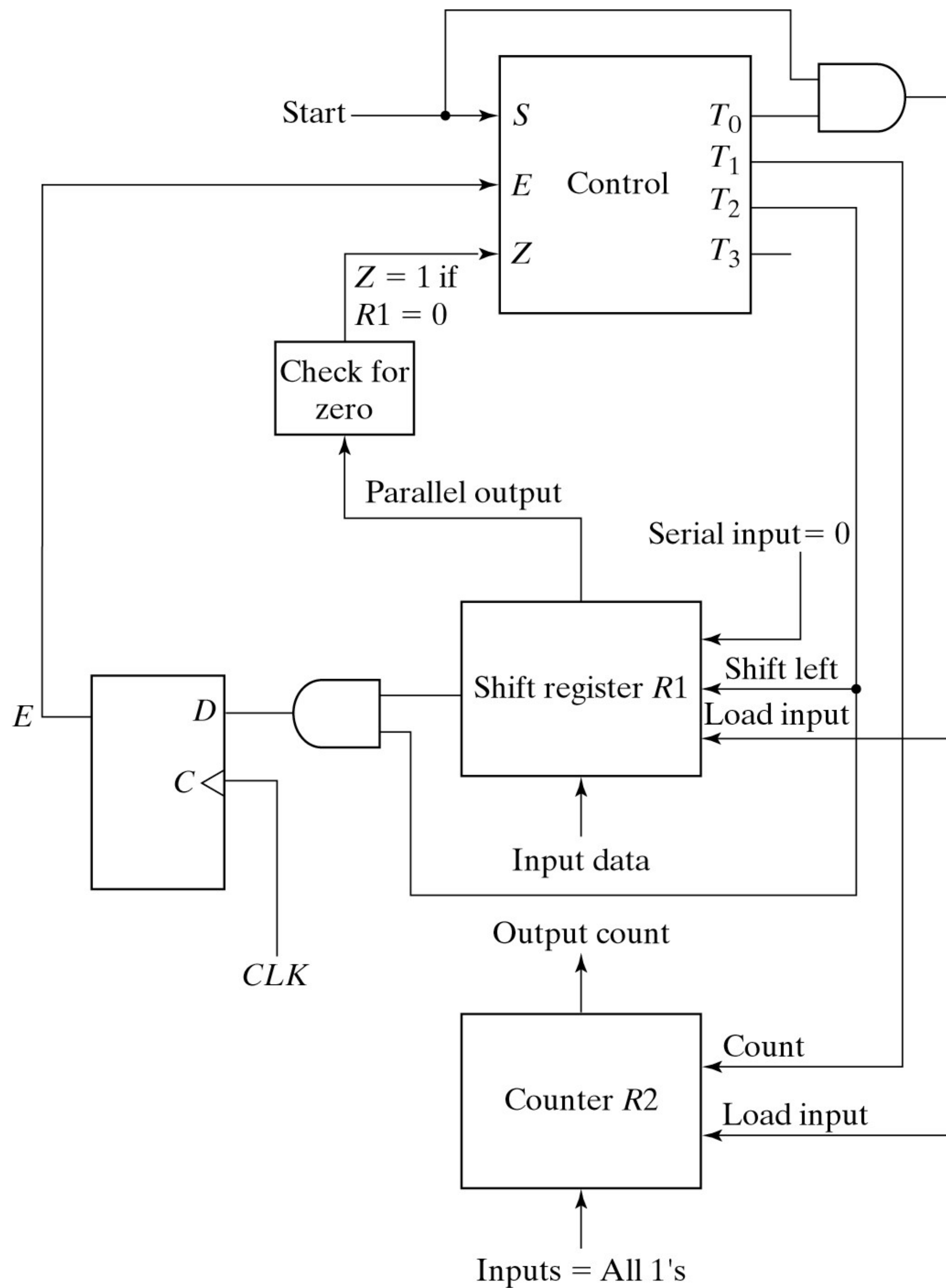


Fig. 8-21 ASM Chart for Count-of-Ones Circuit

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

We Also Need

- A F-F to store the bit shifted out of R1
- A combinational circuit to check if all bits of R1 are zero.



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

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

The Table

Pr. St.		N. St.		Inp. Cond	MUX1	MUX2
G1	G0	G1	G0			
0	0	0	0	S’	0	S
0	0	0	1	S		
0	1	1	0	Z	1	Z’
0	1	1	1	Z’		
1	0	1	1	-	1	1
1	1	1	0	E’	E’	E
1	1	0	1	E		

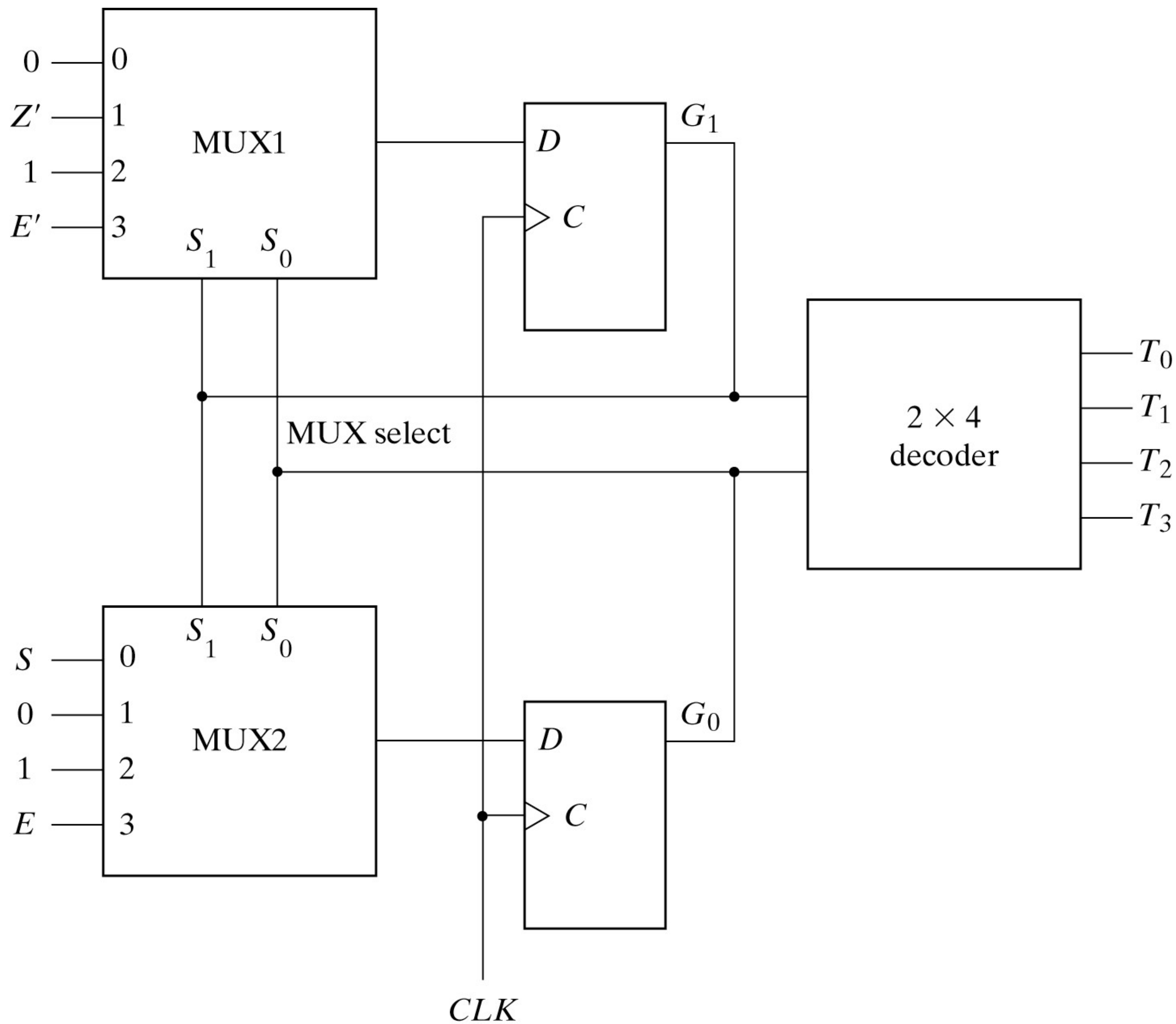


Fig. 8-23 Control Implementation for Count-of-Ones Circuit

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

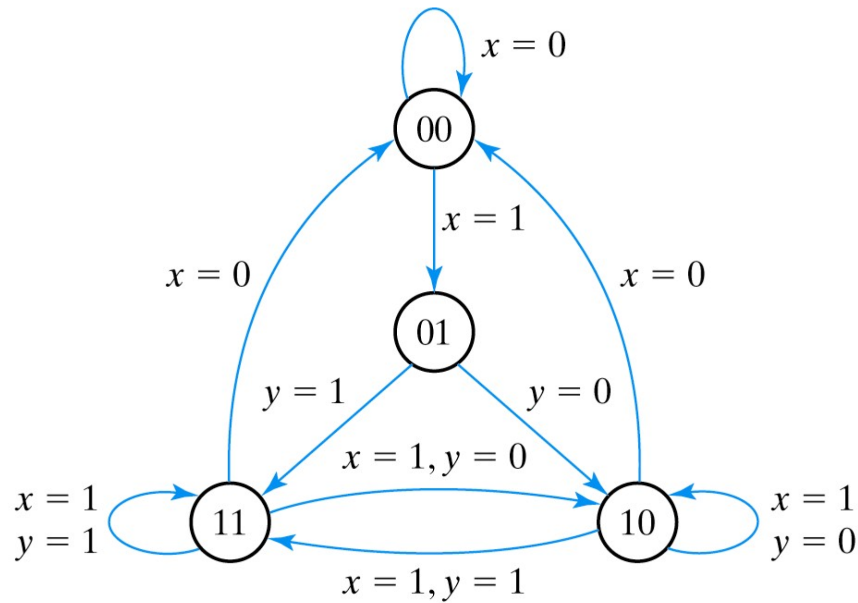
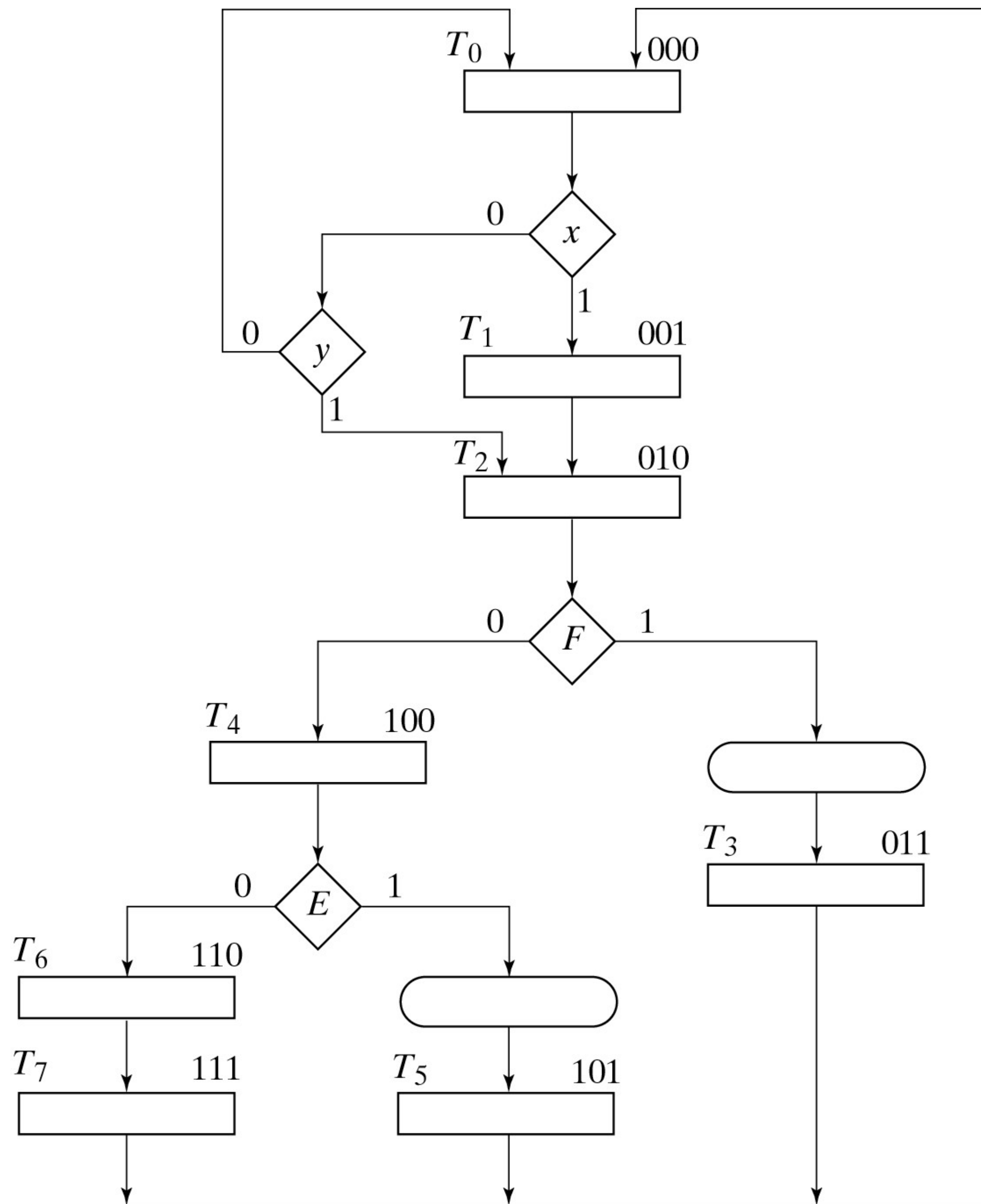


Fig. P8-10 Control State Diagram for Problems 8-10 and 8-11

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.



© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

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