ASM Chart: Multiplier Control

COE608: Computer Organization and Architecture

Dr. Gul N. Khan

http://www.ee.ryerson.ca/~gnkhan

Electrical and Computer Engineering

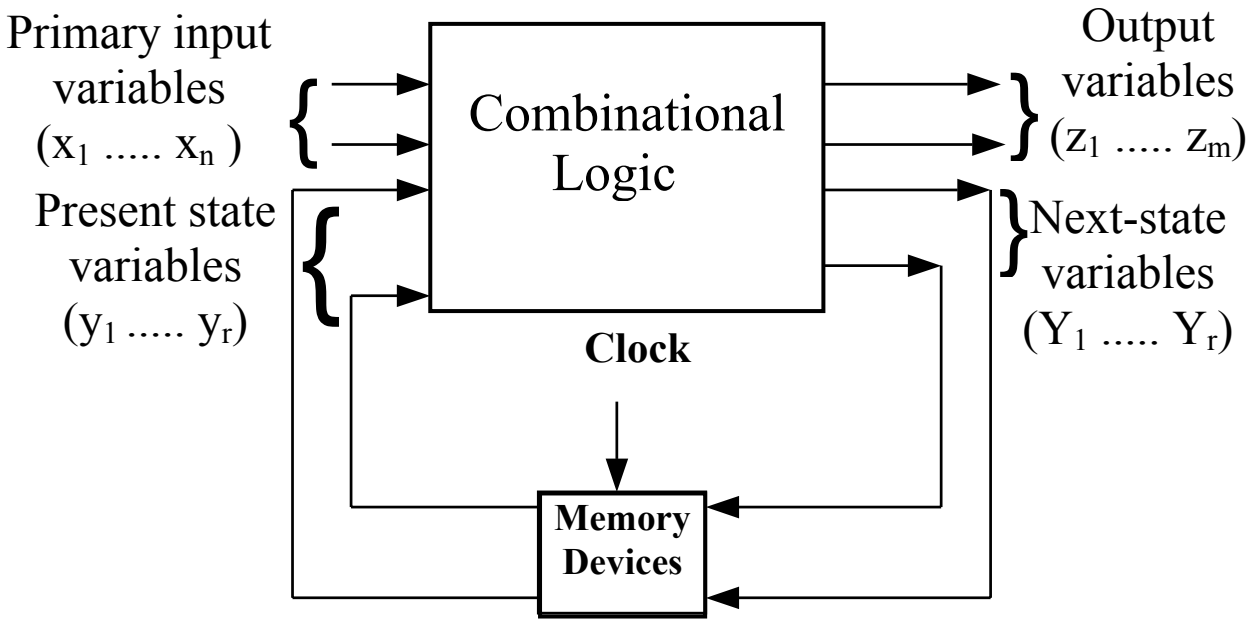
Ryerson University

Overview

- Types of Sequential Circuits
  - Mealy and Moore Machine Models
  - Sequence Detector Implementations
- Algorithmic State Machines: Introduction
- Realization of ASM
- Control Unit Design of the Multiplier
- Hardwired Control
  - Sequence Register and Decoder Method
  - One Flip-Flop per State Method

Part of Chapter 8, section 8.3 - 8.5 of Text by Mano and Kime

Sequential Logic Circuits



Inputs

Primary Inputs
State variables

Outputs

Output variables
Next state variables.

Synchronous Sequential Circuits

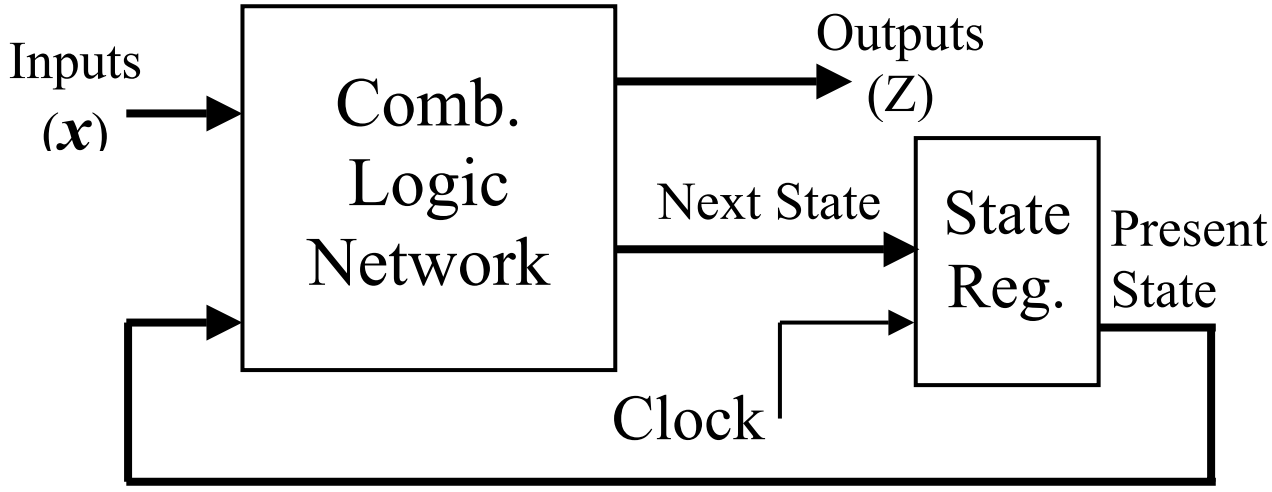
Clock is used to ensure occurrence of event (change of state) at a specified instant of time.

Asynchronous Sequential Circuits

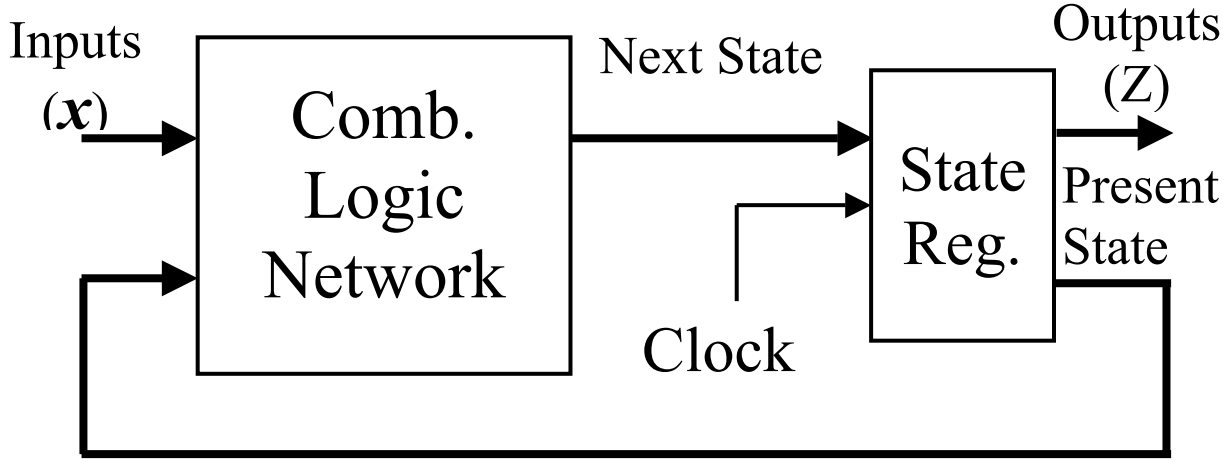
Sequential Machine Models

Main Models of Sequential Circuits or Machines are: Mealy and Moore Model

Mealy Machines: Their outputs depend on both the present state and the present inputs.

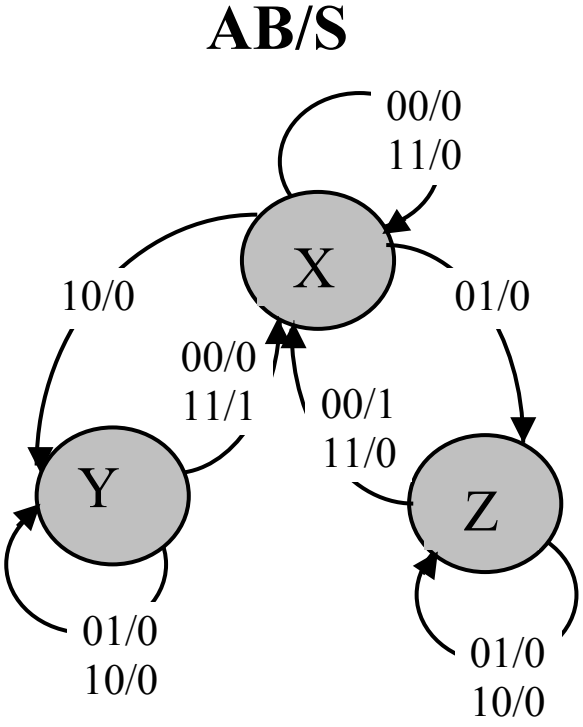


Moore Machines: The outputs depend on the present state only.

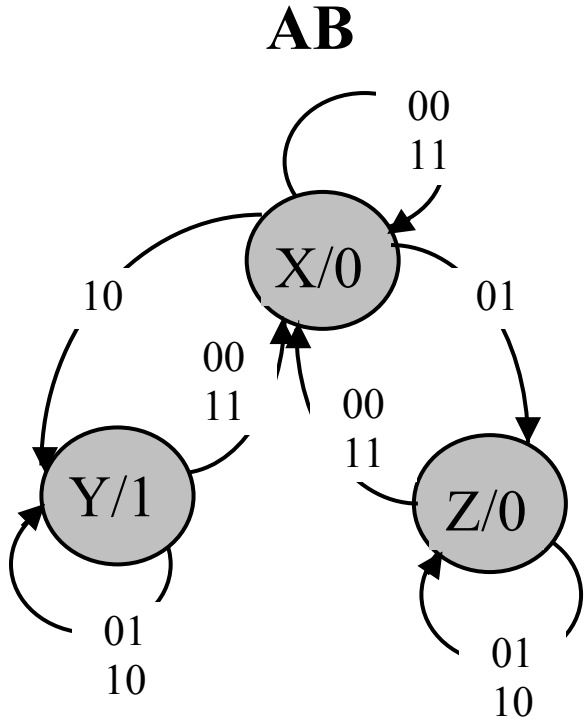


Sequential Machine Models

Mealy Model



Moore Model



PS	NS			
	AB			
	00	01	11	10
x	x/0	z/0	x/0	y/0
y	x/0	y/0	x/1	y/0
z	x/1	z/0	x/0	z/0

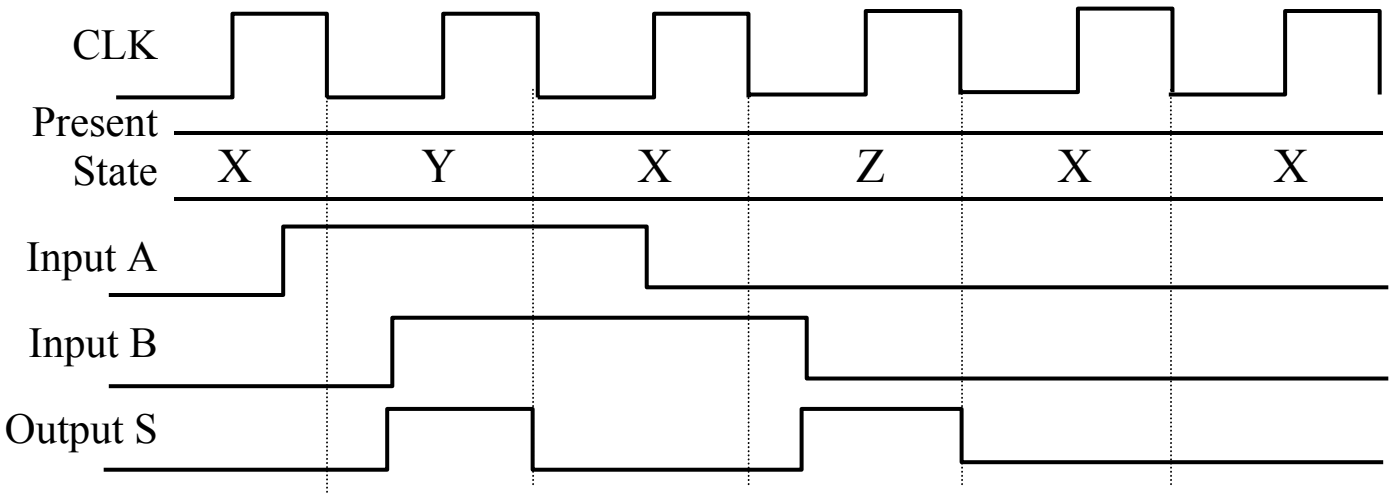
Mealy State Table

PS	NS				Output
	AB				S
	00	01	11	10	
x	x	z	x	y	0
y	x	y	x	y	1
z	x	z	x	z	0

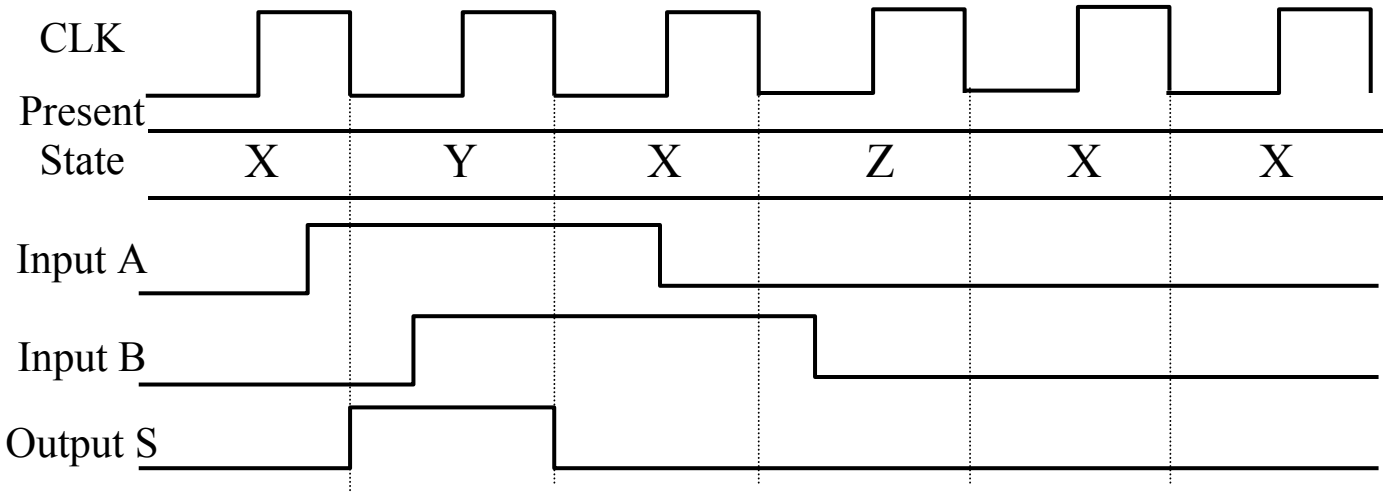
Moore State Table

Sequential Machine Models

Mealy Model Timing Diagram



Moore Model Timing Diagram



Sequential Machine Models

Main Features:

- Moore machine realization is more complex than Mealy due to additional state requirements to derive the required outputs.
- Outputs of a Moore machine are generally robust and independent of external (primary) inputs.
- Mealy model is useful for applications where faster respond is needed.
- In the case of Mealy machines, all unspecified states must end up in a specified state after the next or consecutive clock cycle.
  Otherwise, oscillation/hang-up may occur for certain input combinations.

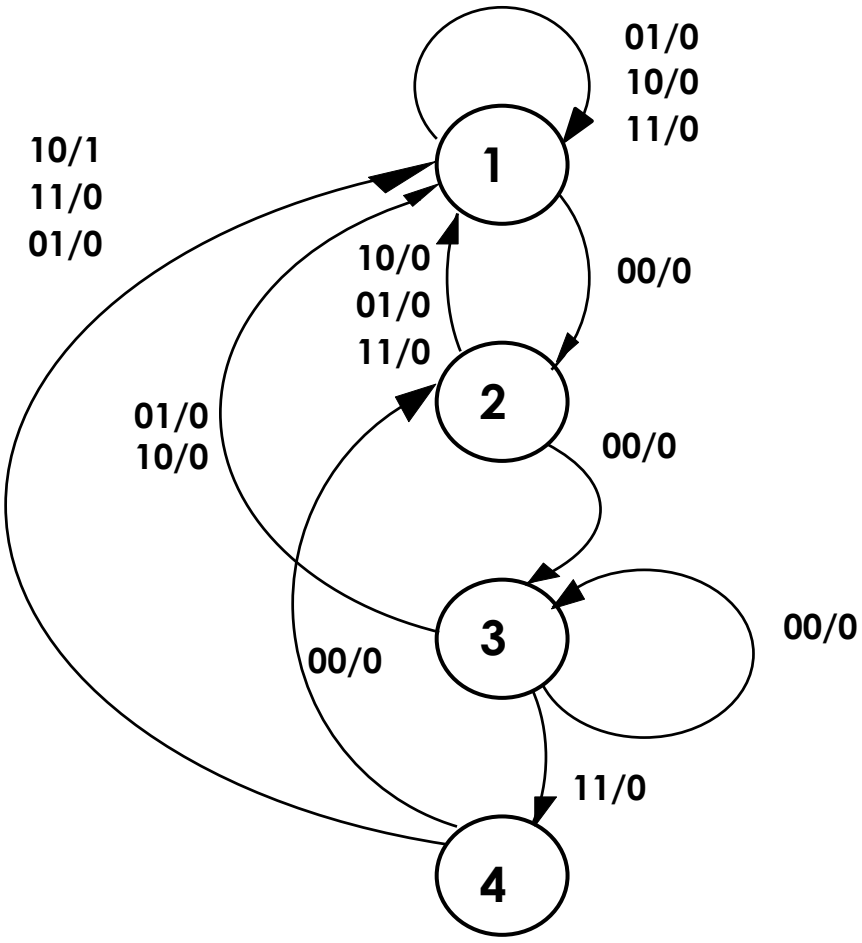
Finite State Machine Example

Problem Statement:

- The occurrence of sequence of pairs of inputs $00 \rightarrow 00 \rightarrow 11 \rightarrow 10$ is to be detected.
- The machine will output logic 1 after detecting the sequence successfully.

The Mealy Model:

- State transitions are labeled with inputs and the output values.



State Assignment

The occurrence of sequence of pairs of inputs $00 \rightarrow 00 \rightarrow 11 \rightarrow 10$ is to be detected.

Mealy Machine Design Approach

State-Transition Table

Present state	Inputs X_1, X_2							
	Next state				Output z			
	00	01	11	10	00	01	11	10
1	2	1	1	1	0	0	0	0
2	3	1	1	1	0	0	0	0
3	3	1	4	1	0	0	0	0
4	2	1	1	1	0	0	0	1

State Assignment: Assign binary codes to the states. **1 = 00, 2 = 01, 3 = 11 and 4 = 10**

Present State y_1y_2	Inputs x_1x_2							
	Next State, w_1w_2				Output z			
	00	01	11	10	00	01	11	10
00	01	00	00	00	0	0	0	0
01	11	00	00	00	0	0	0	0
11	11	00	10	00	0	0	0	0
10	01	00	00	00	0	0	0	1

State Assignment

- Any state assignment is satisfactory as long as each state is assigned a unique binary code.
- However, one particular assignment may be optimal that requires least number of gates.

Guidelines for State Assignment

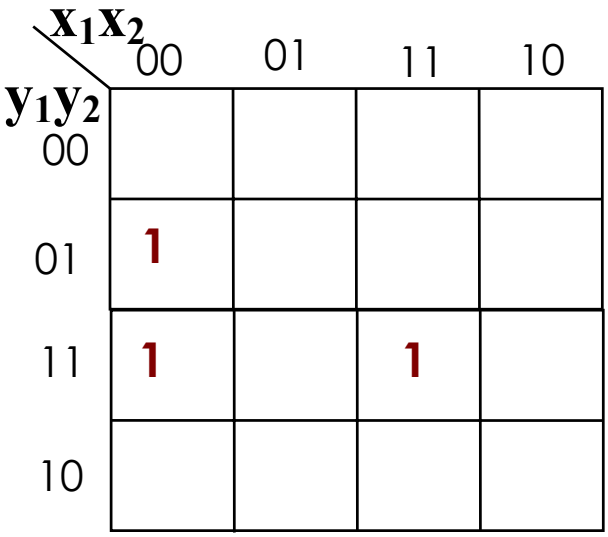
- States having the same NEXT STATES for a given input condition should be given adjacent assignments.
- States, which are NEXT STATES of a single-state, should be given adjacent assignments.
- States, which have identical output specification, should be given adjacent assignments.

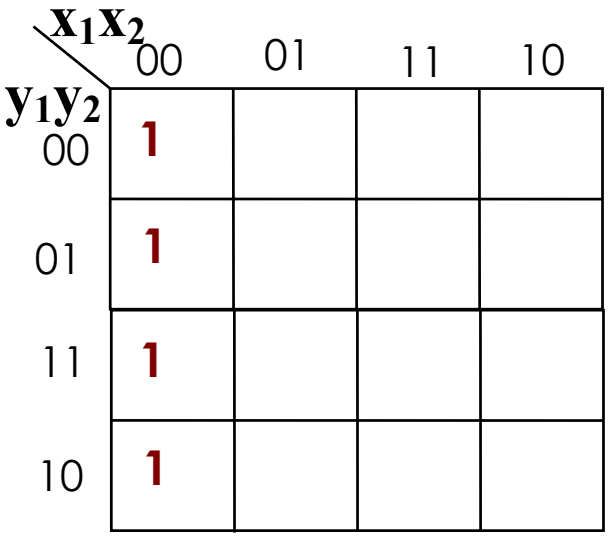
Overall

Minimize the number of Sate Variable changes as you move through the state diagram.

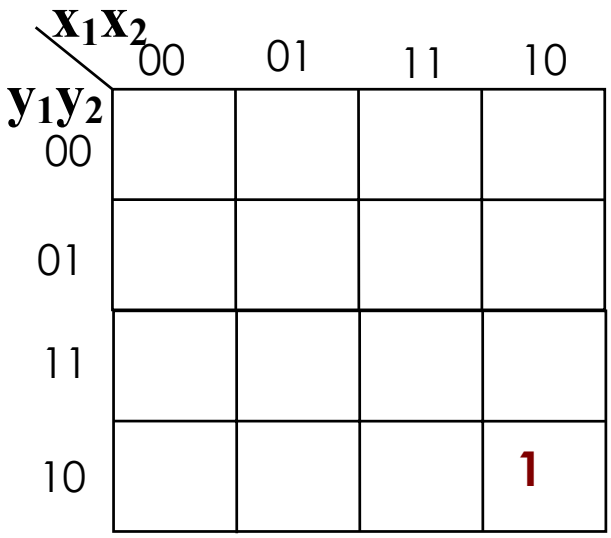
Mealy Machine Design

Using D-type FFs





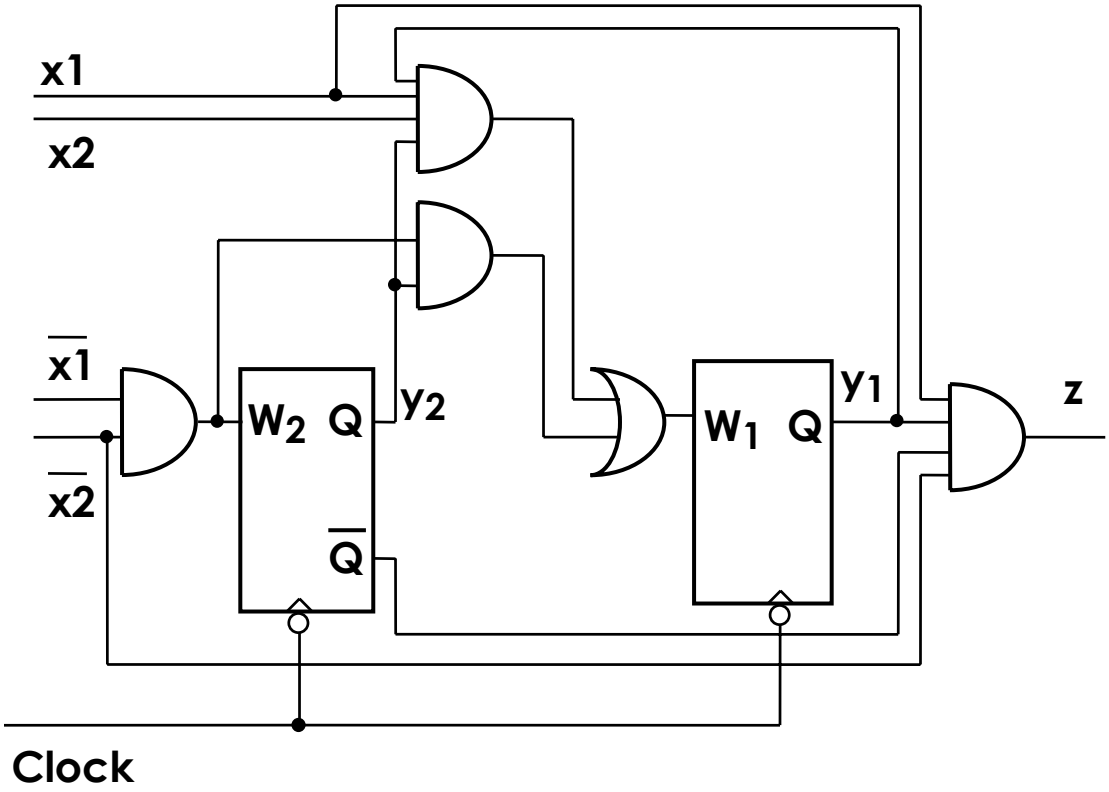
$$w_1 = \overline{x_1}\,\overline{x_2}y_2 + x_1x_2y_1y_2 \qquad w_2 = \overline{x_1}\,\overline{x_2}$$



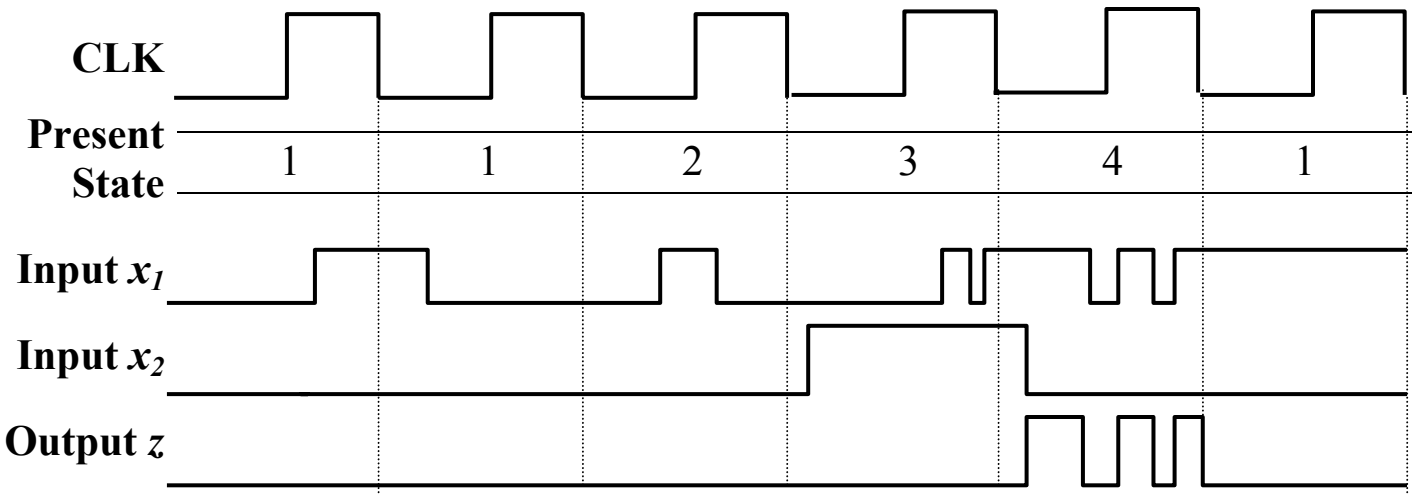
BY INSPECTION OUTPUT $z = x_1\overline{x_2}y_1\overline{y_2}$

Mealy Machine Design

Sequential Circuit



Timing Diagram

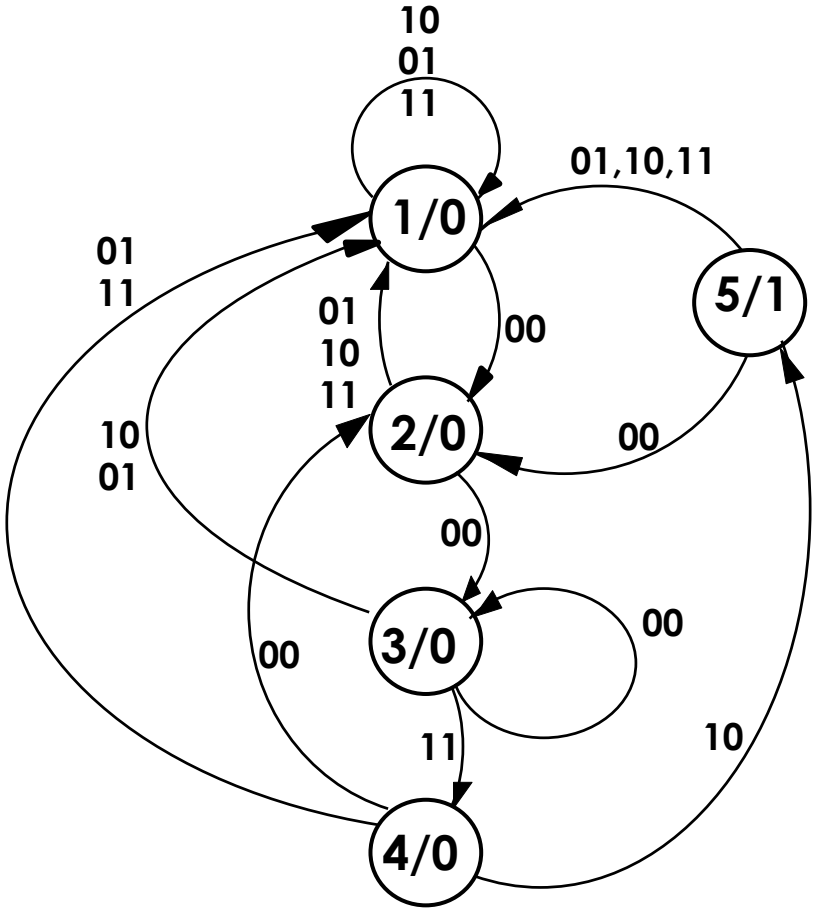


Moore Machine Implementation

The Moore Model:

- Suitable for asynchronous sequential systems.
- Transitions are labeled with the inputs only.

State Diagram



Moore Machine Example

State-transition Table

Present State	Next State xy				Output *z*
	00	01	11	10	
1	2	1	1	1	0
2	3	1	1	1	0
3	3	1	4	1	0
4	2	1	1	5	0
5	2	1	1	1	1

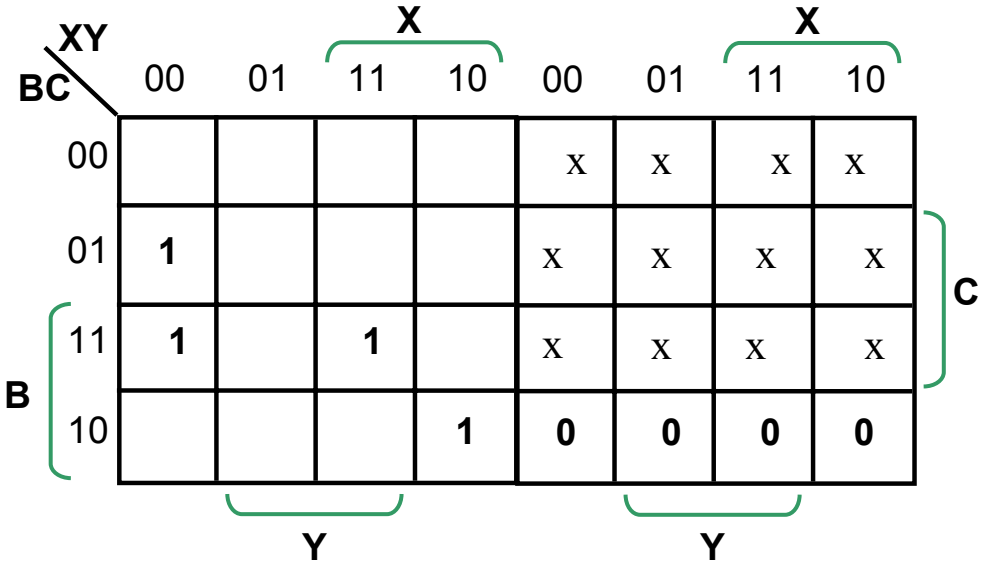
State Assignment

Present State ***ABC***	**Next state** $(A^+B^+C^+)$ ***XY*** 00	01	11	10	**Output** ***z***
000	001	000	000	000	0
001	011	000	000	000	0
011	011	000	010	000	0
010	001	000	000	110	0
110	001	000	000	000	1

Using D-type FFs

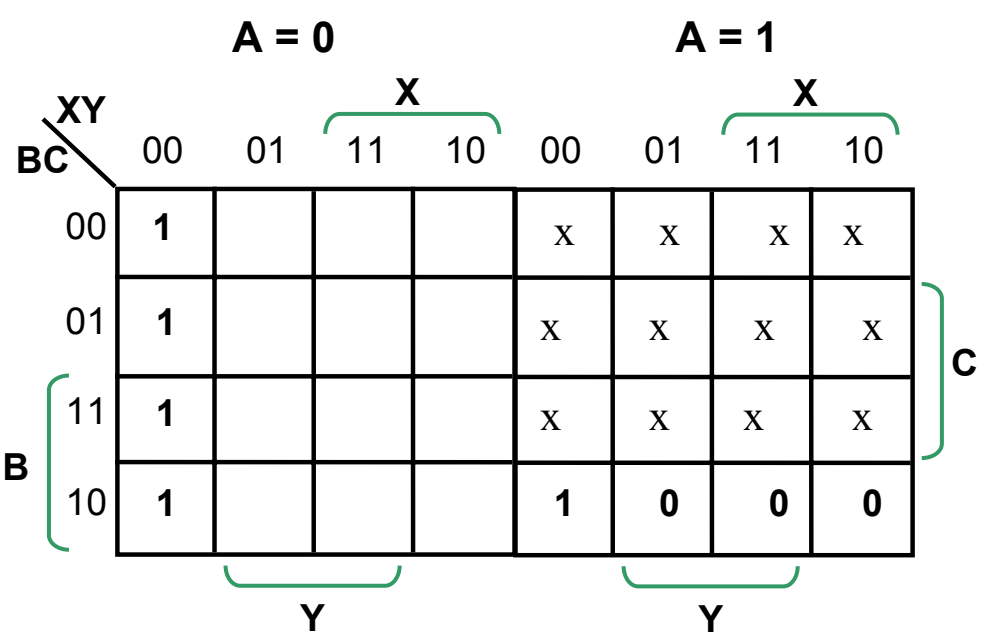
Moore Machine Design

B^+



$$B^+ = C\overline{xy} + BCxy + \overline{A}B\overline{C}x\overline{y}$$

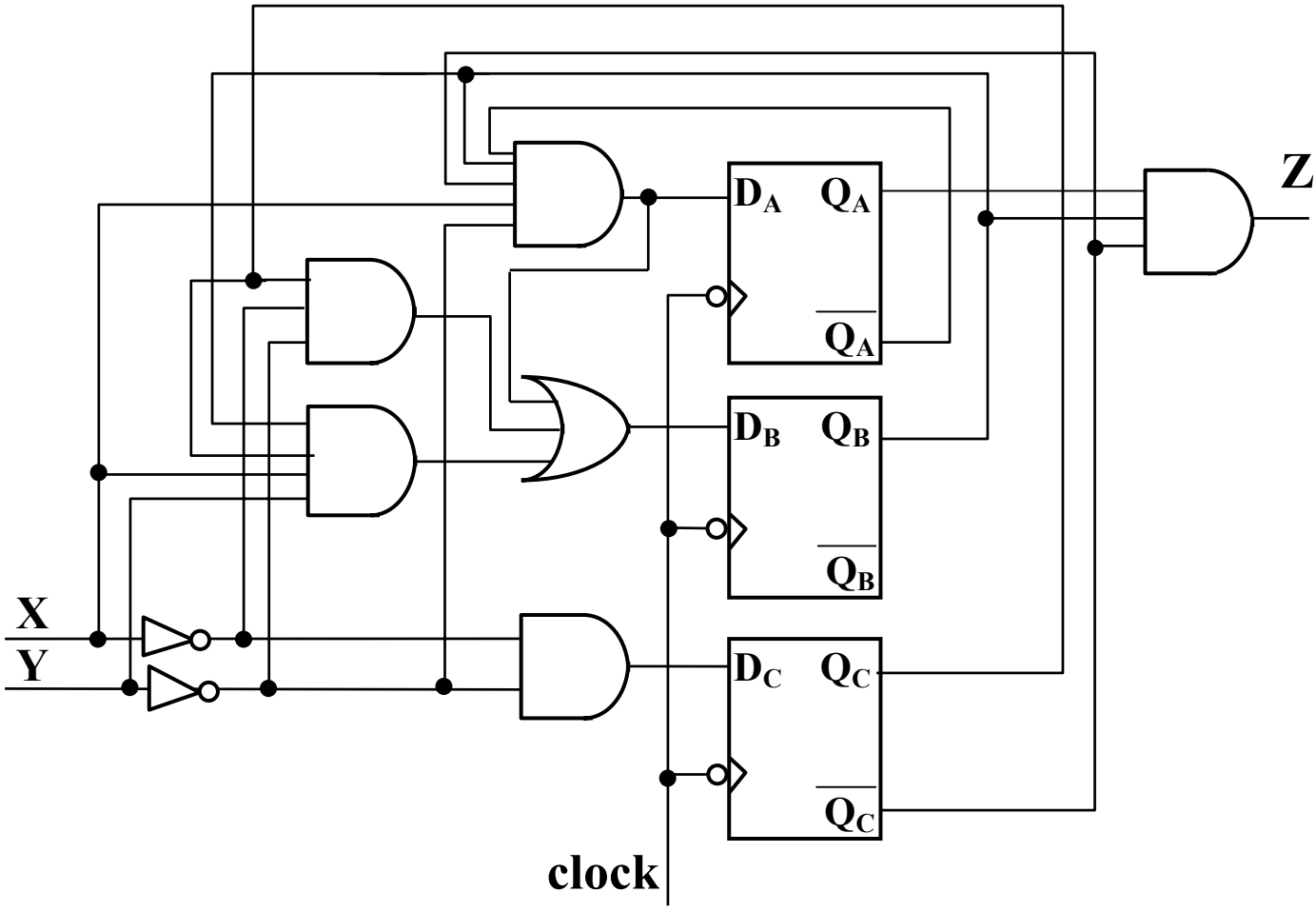
C^+



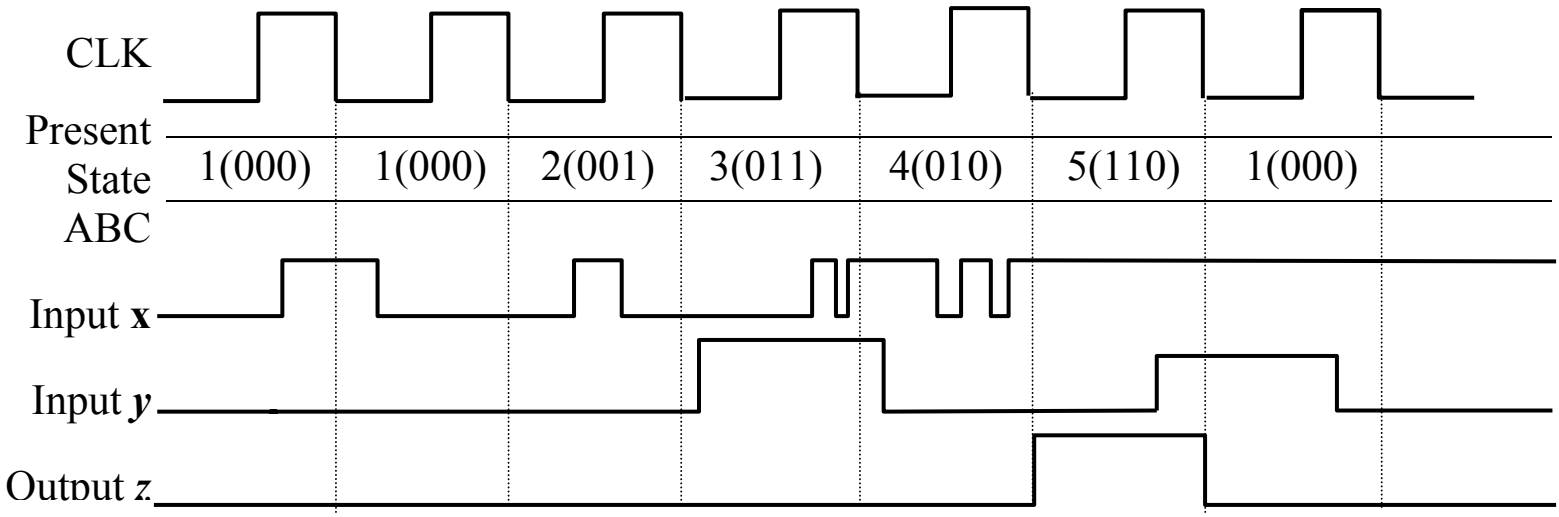
$$C^+ = \overline{xy}$$

Moore Machine Design

Sequential Circuit



Timing Diagram



Algorithmic State Machine (ASM)

- Flow Charts are mostly used for software design. They are also useful for digital system design.
- The ASM or State Machine charts offer several advantages over state diagrams.

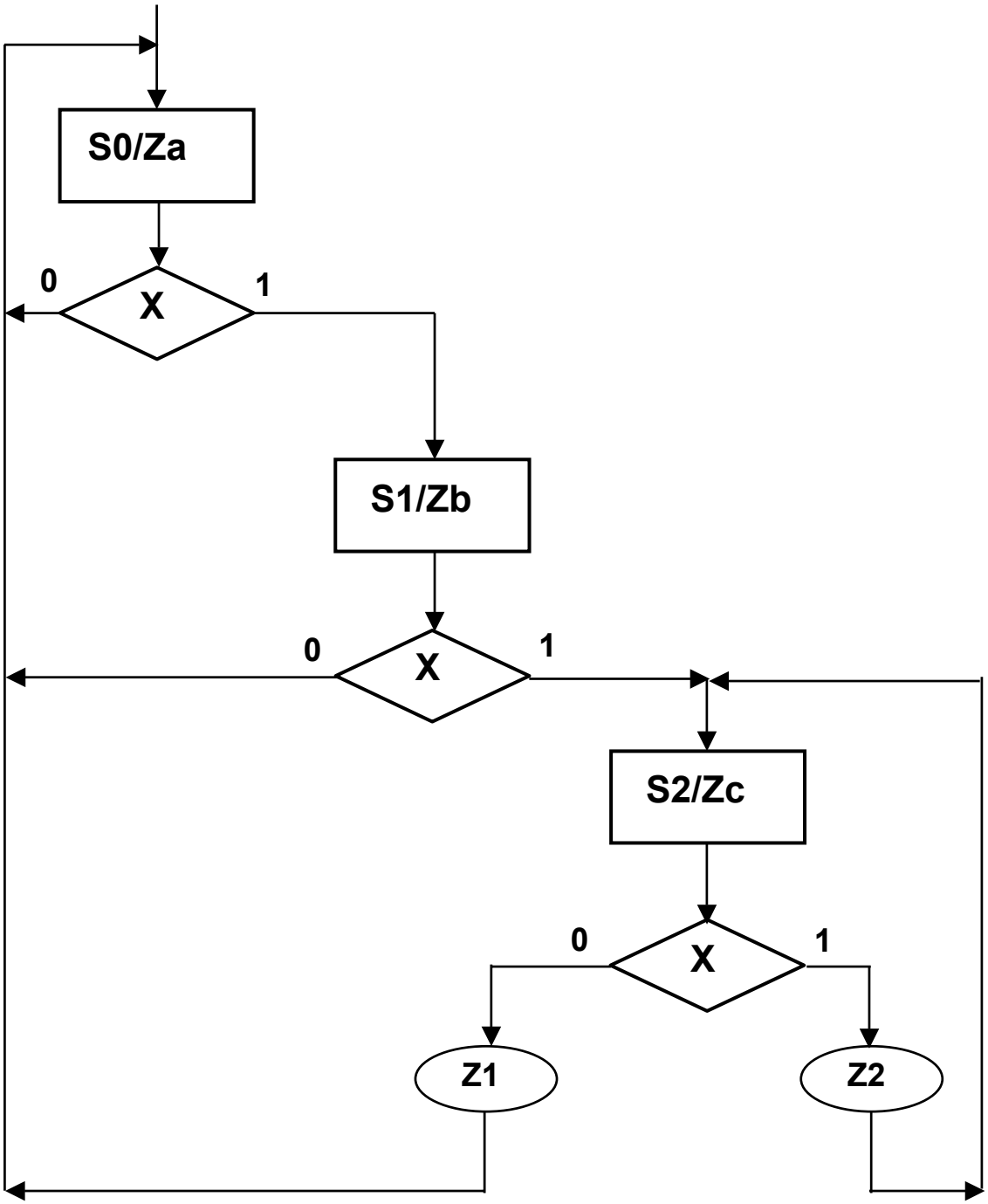
Main Features of ASM Charts

- Operation of a digital system represented by an ASM chart is easier to understand.

- An ASM chart can be converted into several equivalent forms and each form leads directly to a hardware realization.
- The conditions for a proper state diagram are completely satisfied by the ASM chart.
- ASM chart based digital system design is equivalent to software design.

ASM Charts

A Typical ASM Chart



ASM Charts

The state diagram/table based design approach becomes impractical for systems with large number of inputs.

- The number of columns for the state table doubles with every additional input.
- All the inputs are not relevant at each clock pulse/transition (don't care conditions). On the other hand, ASM approach only shows the active inputs on the chart.
- State diagrams are not suitable for gradual refinement of FSM.

Typical State Table

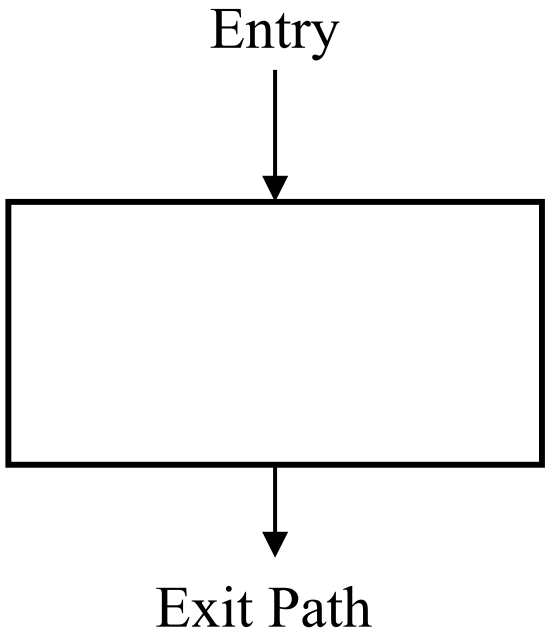
Present State	**Next state** $(A^+B^+C^+)$				**Output**
ABC	***XY*** 00	01	11	10	***z***
000	001	000	000	000	0
001	011	000	000	000	0
011	011	000	010	000	0
010	001	000	000	110	0
110	001	000	000	000	1

ASM Charts

Basic Elements of an ASM chart are:

State Box

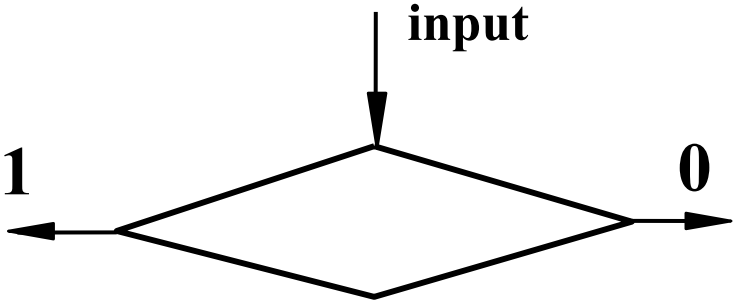
- It represents one state of the ASM.
- The sequential machine resides in a state box for one state time (one clock cycle).
- It consists of a *state name, state assignment code* and *state output* (Moore).
- State box has a single exit/entry point unlike to a state node in state diagram.



ASM Charts

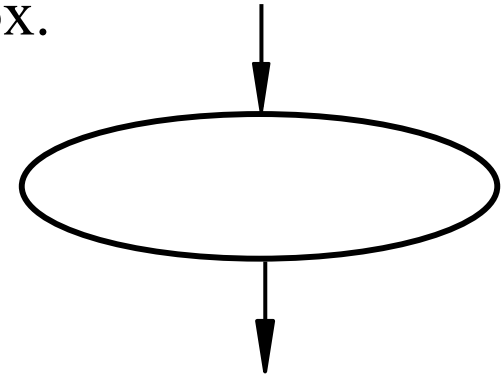
Decision Box

- The *decision box* takes machine inputs.
- It contains Boolean variables to be tested and gives conditions that *control* or *qualify* conditional state transition and outputs.
- Single entry path and two exit paths define the condition for true or false exit.



Conditional Output Box

- It describes those outputs that only become active on true conditions.
- It is always connected to the exit-path of a decision box.



ASM Charts

ASM charts are equivalent to state diagrams:

- Sate Box ⇔ State diagram node
- Decision Box ⇔ Input values on the state transition lines.
- Outputs in the State Box ⇔ Output values in the state nodes. (Moore Machine)
- Outputs in Conditional Output Box ⇔ Output values on the state transition lines. (Mealy Machine)

ASM Block

ASM charts are constructed from ASM Blocks

An ASM block consists of:

- Exactly one state box.
- Decision and conditional output boxes associated with the state.
- One entry path and one or more exit paths.

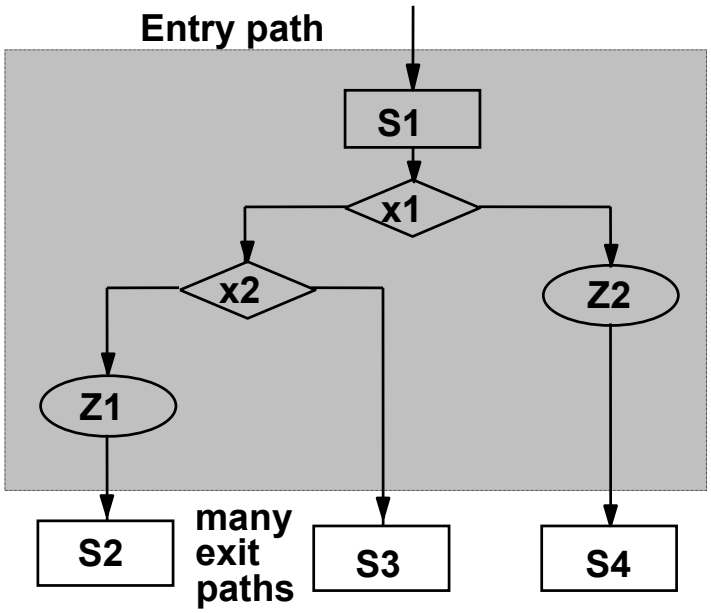
A pure combinational circuit can be described by one ASM block.

An ASM block describes the machine operation during the time that the machine is in that state.

ASM Block

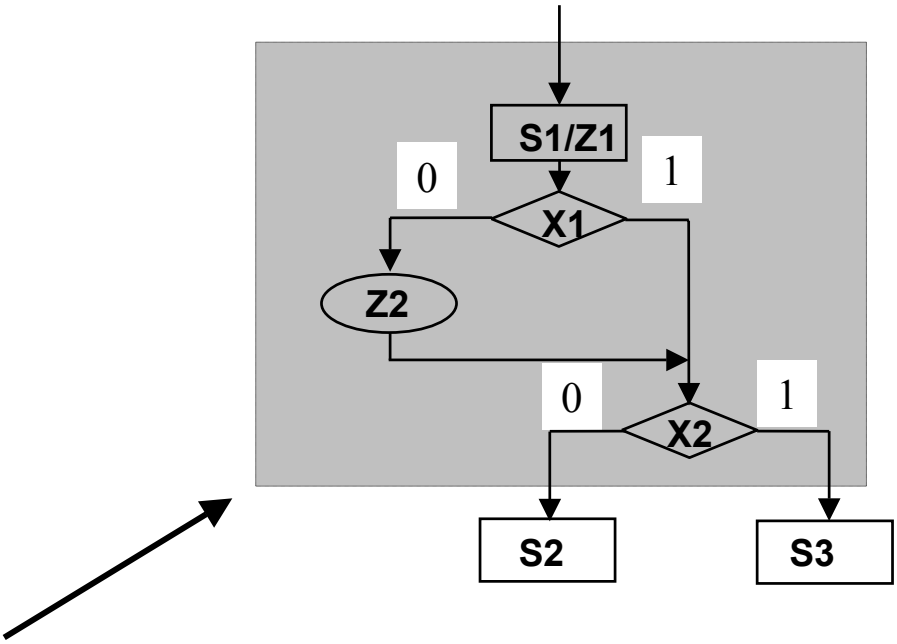
When a digital system enters the state associated with an ASM block:

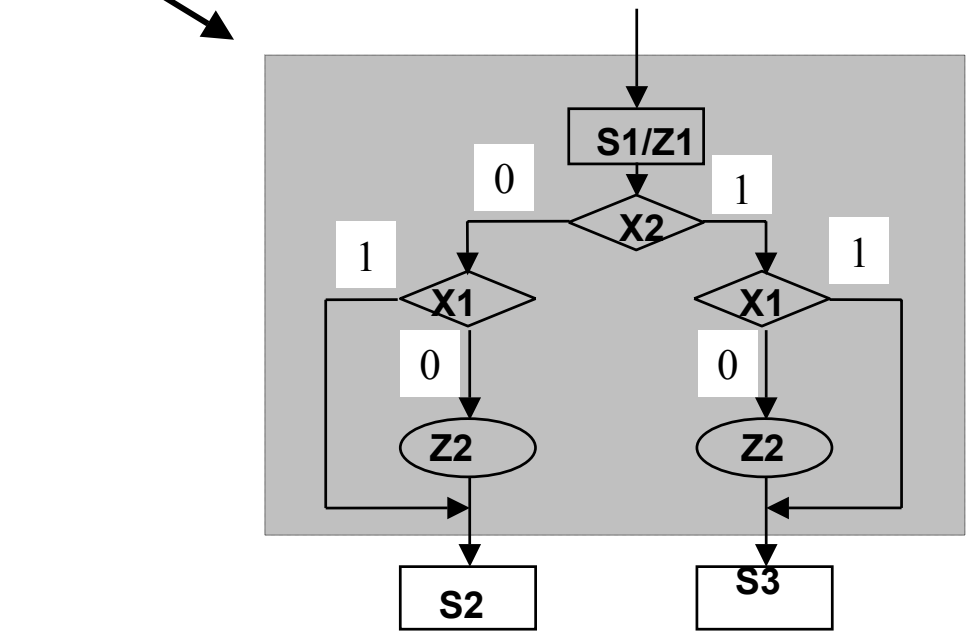
- Outputs on the output list of the state box become true.
- The conditions in the decision boxes are evaluated to determine which path(s) are to be followed.
- When a conditional output box is encountered along such a path, the corresponding conditional outputs become true.
- If an output is not encountered along a path that output is assigned a FALSE (by default).
- Each exit path of an ASM block must lead to another state.
- Each possible path through an ASM block from entrance to exit is termed as link path.



ASM Block

An ASM Block can be drawn in several ways.

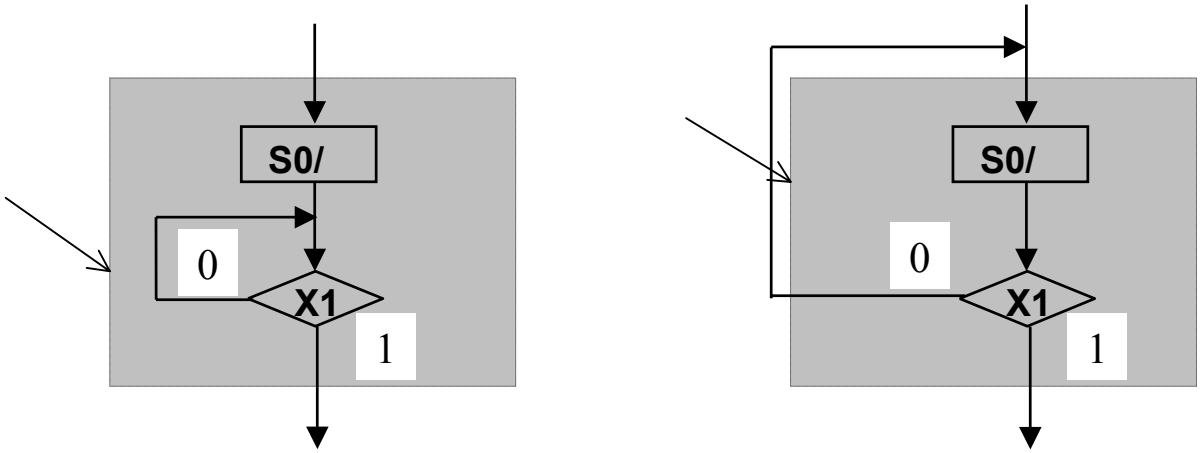




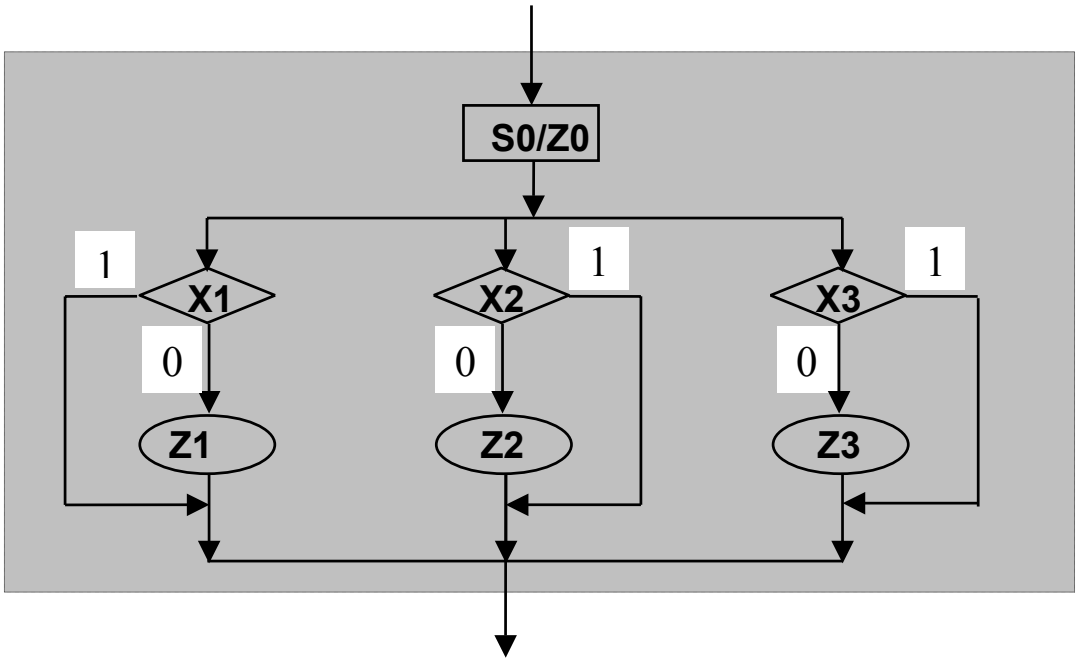
ASM Block

Rules to Construct an ASM Block

- For every valid combination of input variables, there must be one exit path.
- No internal feedback within an ASM block is allowed.



- An ASM block can have several parallel paths that lead to the same exit path and more than one of these paths can be active at the same time.

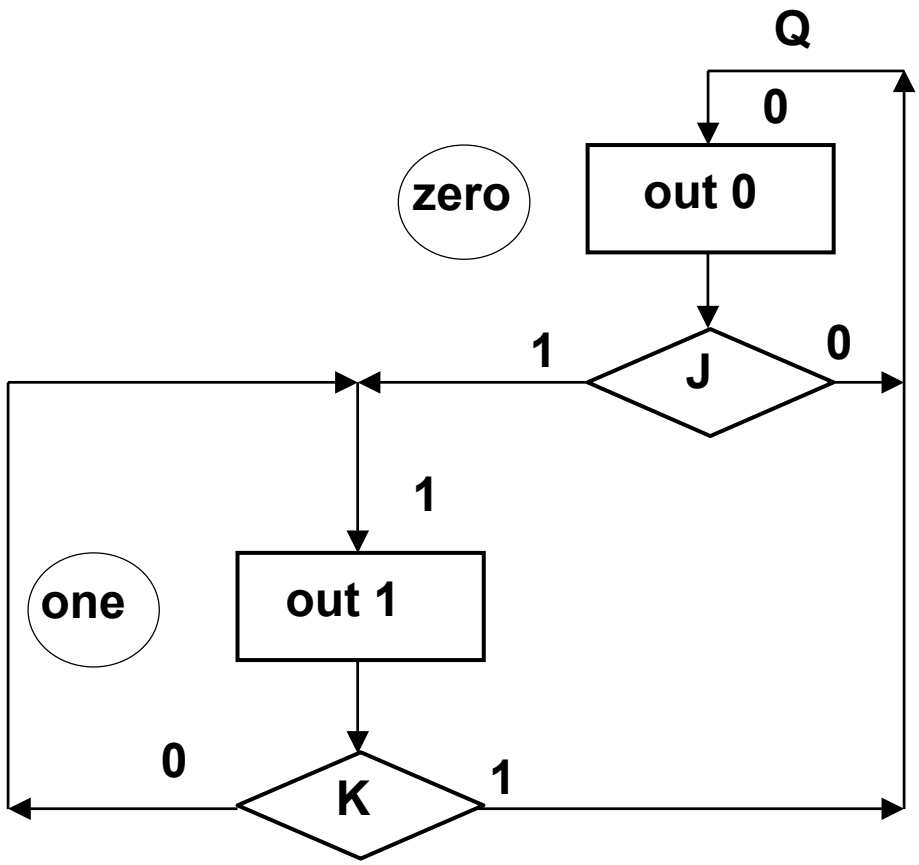


ASM Chart

- An ASM chart consists of one or more ASM blocks connected in a consistent manner.
- In the case of autonomous sequential circuits

  ASM chart will consist of state boxes connected by direct transition link paths.

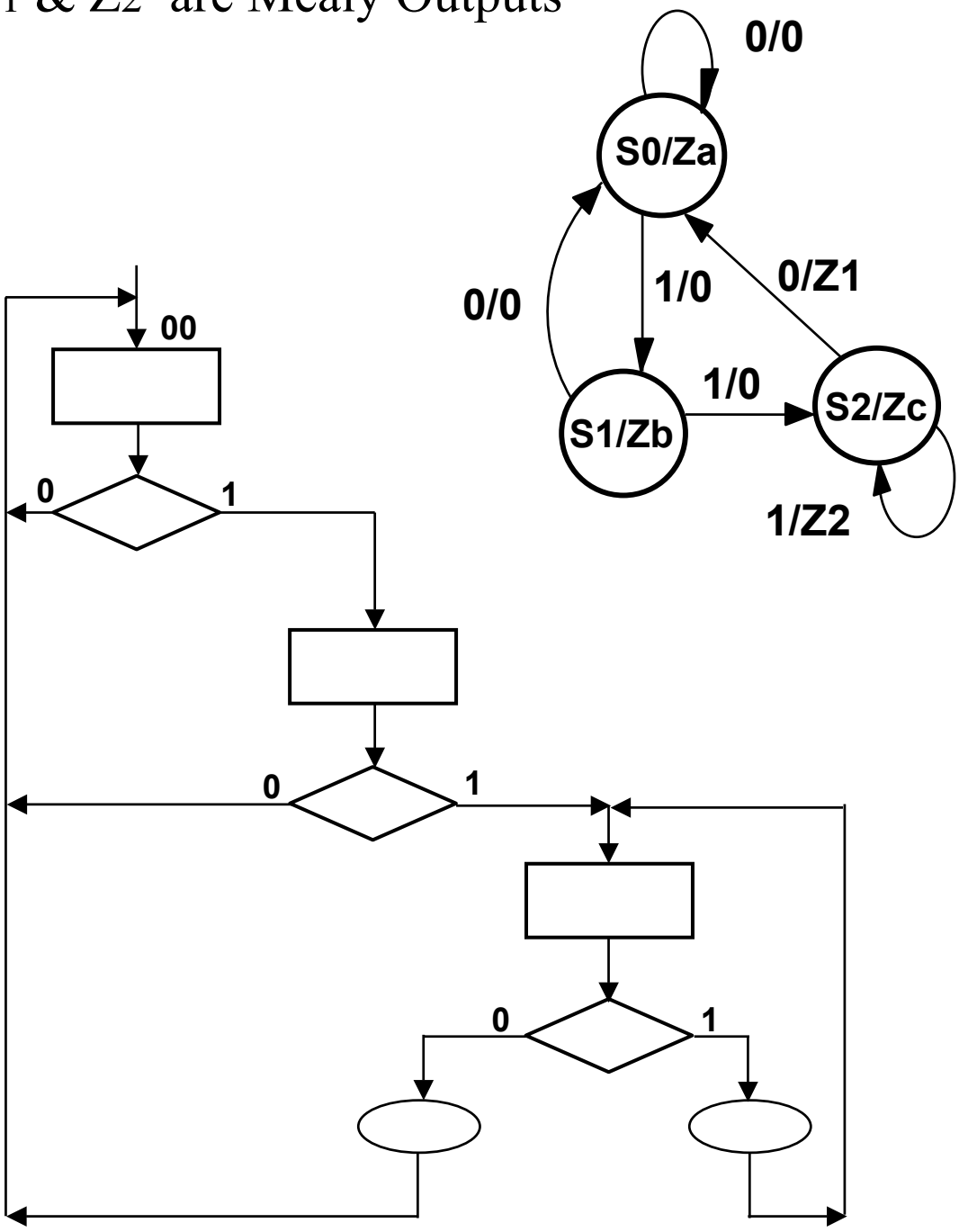
The ASM chart of a JK Flip-Flop



ASM Chart

From State Diagram to ASM Chart

- X is an input
- Z_a, Z_b & Z_c are Moore Outputs
- Z_1 & Z_2 are Mealy Outputs



Realization of ASM Chart

Main Steps

- For each state variable (e.g. Qa, Qb, etc.), identify all states in which Q = 1

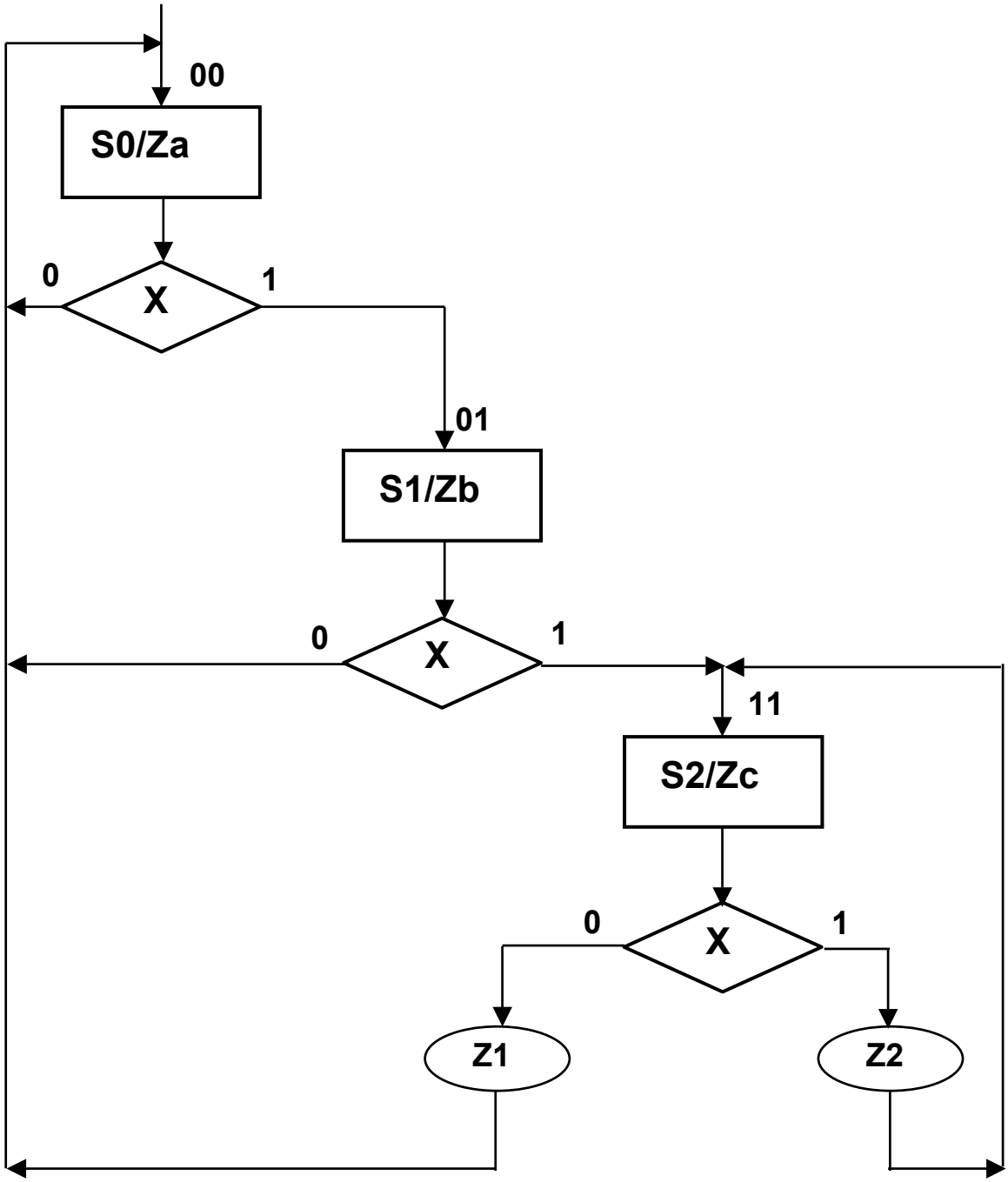
  For each of these states, find all the link-paths that lead into the state.

  - For each of these link paths, find a product-term that is logic-1 when the link-path is followed.
  - e.g. For a link path from state Si to Sj, the product term will be 1 if the machine is in state Si and the conditions for Sj entry are satisfied.

- The expression for Q^+ is formed by ORing all the product-terms found for a particular state variable as above.

Realization from ASM Chart

Example



Realization from ASM Chart

Working Example

For Next State: Consider Variable B Link-paths for States that has B = 1 are S1 and S2 states.

Link-Path-1

- Starting with a present state AB = 00, takes the X=1 branch and terminates at state S1 during which B = 1.

Link-Path-2

- Starting state 01, takes X=1 branch & ends at state 11.

Link-Path-3

- Starting at state 11, takes X=1 branch and ends in state 11.

Overall $\mathbf{B^+ = A'B'X + A'BX + ABX}$

For Next State: Consider State Variable A.

- Two link paths terminate at S2 state

Moore Outputs

Za = A'B'; Zb = A'B; Zc = AB

Conditional Output

Binary Multipliers

Hand Multiplication:

```
   11                1 1 0 1      Multiplicand
 * 13                1 0 1 1      Multiplier
                     1 1 0 1
                   1 1 0 1
                 0 0 0 0
               1 1 0 1
 143         1 0 0 0 1 1 1 1
```

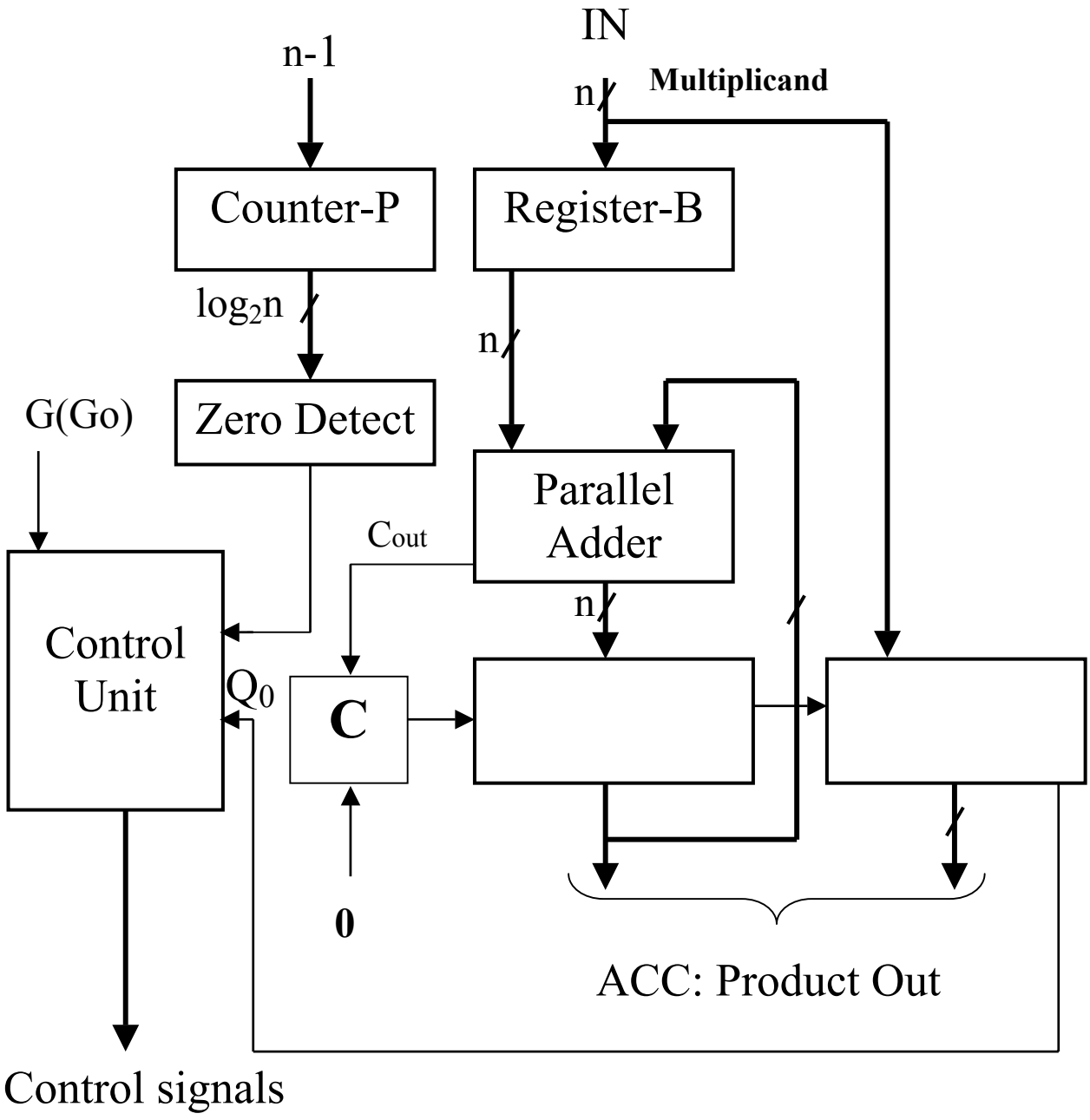
Modified (serial) Multiplication

```
 11   0 0 0 0 0 | 1 0 1 1     Initial contents of product register
* 13     1 1 0 1             M bit=1, add multiplicand
       0 1 1 0 1 | 1 0 1 1   Partial product before shift
       0 0 1 1 0 | 1 0 1 1   Partial product after shift
         1 1 0 1             Multiplier bit=1, add multiplicand
       1 0 0 1 1 | 1 0 1 1   Partial product before shift
       0 1 0 0 1 | 1 1 0 1   Partial product after shift
                             Multiplier bit=0  skip addition
       0 0 1 0 0 | 1 1 1 1   Partial product after shift
         1 1 0 1             Multiplier bit=1, add multiplicand
       1 0 0 0 1 | 1 1 1 1   Partial product before shift
       0 1 0 0 0 | 1 1 1 1   After shift (Final answer)
```

Final Result = 0 1 0 0 0 1 1 1 1

Multiplier

Block Diagram



Binary Multipliers

The multiplication of two binary numbers is performed by successive additions and shifting.
B ← Multiplicand; Q ← Multiplier
Partial product is formed in A and stored in A & Q.

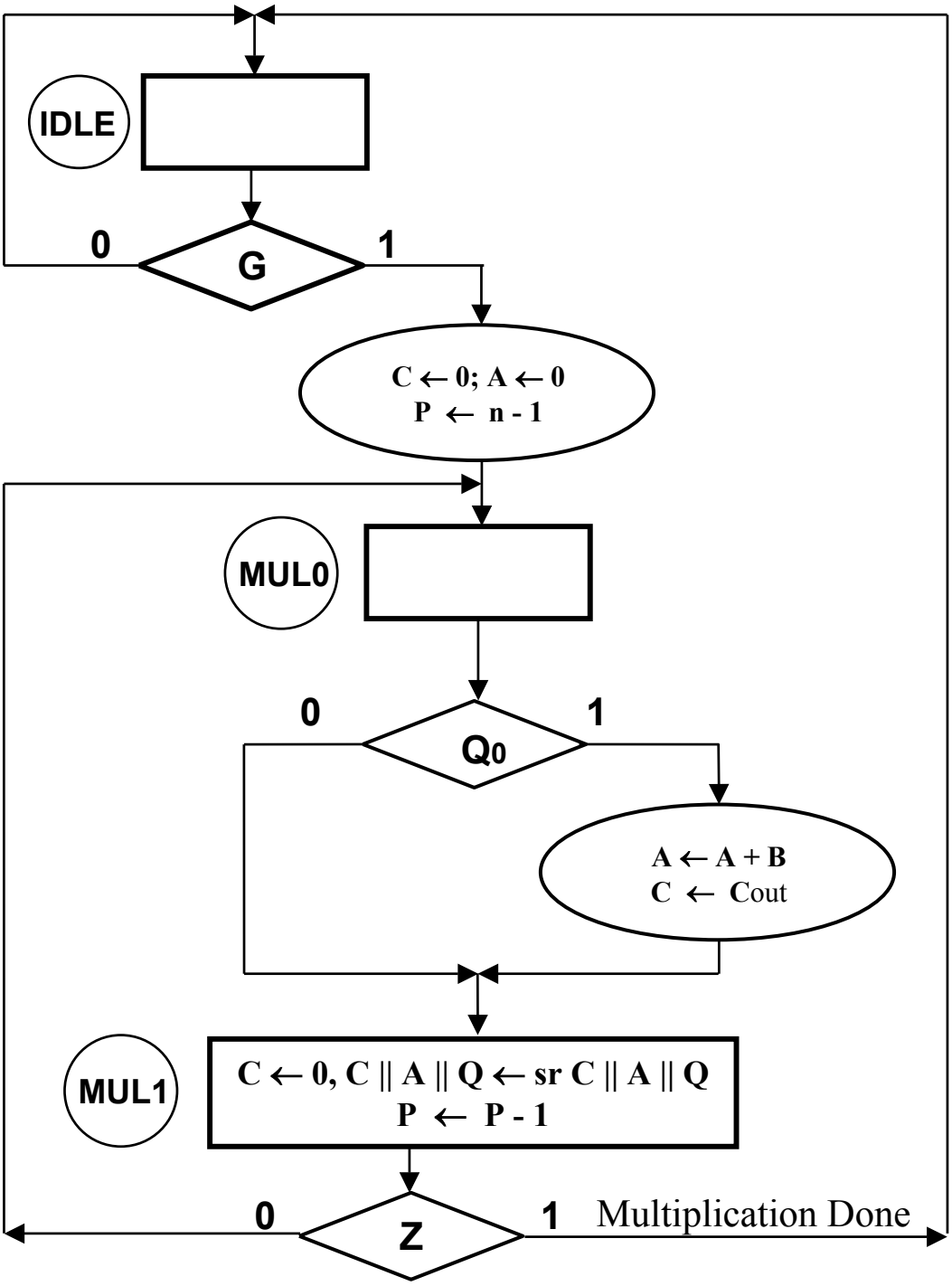
Multiplier Circuit Operation

- Q is an n-bit shift register where multiplier is loaded that is shifted right. It vacates 1-bit space every time. This space accepts the lower part of the partial product.
- An n-bit parallel adder produces Sums as
  A ← A + B
- C flip-flop stores the carry from addition. It is reset to zero during the right shift.
- Counter P counts the number of add-shift or shift actions. It is initially set at (n-1) & it counts down.
- When P counts 0, the final product is in the double register A and Q.

Control Unit is the heart of Multiplier:

- Its input, G initiate multiplication.
- Control unit generate control signals to perform add-shift or shift operations.
- It uses Q_0 (LSB of Q shift register) and counter zero-detect, Z signals.

Multiplier Control, ASM



sr = shift right and

C || A || Q ← sr C || A || Q is equivalent to 4 transfers

A(n-1) ← C, A ← sr A, Q(n-1) ← A(0), Q ← sr Q

Multiplier Control Unit

Control Unit is the Heart of Multiplier

- Its input, G initiates multiplication.
- It uses Q_0 (LSB of Q shift register) and counter zero-detect, Z signals.
- Control unit generates control signals to activate following micro-operations:
  - Sum of A and B.
  - PP transferred to A.
  - C_{out} transferred to C.
  - PP & multiplier in A:Q shifted right.
  - Carry from C is shifted to MSB of A:
    - LSB of Q is discarded.
    - After right shift, 1-bit of PP is transferred into Q and multiplier bits are shifted one bit right.
  - Control unit decides between add-shift and shift depending on the LSB of Q.
  - Control unit checks Z for an end.
  - Control unit checks G, to start multiplication.

Multiplier Control

ASM Chart Analysis

Multiplicand in register B

Multiplier in Q

State Changes from IDLE to MUL0

MUL0 State

MUL1 State

- Decrement Counter P
- Four transfers take place

  A(n-1) ← C;

  A ← sr A;

  Q(n-1) ← A(0);

  Q ← sr Q;

Control Unit Design

- Control unit design by using classical FSM design is impractical due to large number of inputs and states it may have.
- An attempt to minimize and simplify these circuits usually ends up in irregular networks that would be difficult to recognize and debug.
- An extension to the classical approach is used by experienced designer in designing control logic circuits:
  - Sequence register and decoder method.
  - One flip-flop per state method. (One-hot state assignment method)
  - Microprogramming.

The first two methods result in a hard-wired logic. Any modification will require rewiring.

- The micro-program control uses ROM/PROM.
- Modification of the PROM or replacing the ROM modifies the micro-program control.

Hardwired Control

Type of Registers Used for Datapath

- Register A is a shift register with parallel load and synchronous clear.
- Register Q is a shift register.
- C flip-flop needs a synchronous clear.
- Register B has a parallel load.
- Register Q has a parallel load.

To Implement Control Unit Consider:

- Control of micro-operations i.e. generate the control signals
- Sequencing of control unit and micro-operation i.e. to determine what happens next.

Control Unit Design Approach

- Simplify ASM chart to represent only state transitions.
- Generate a new table to define control signals in terms of states and inputs.

Control Signals for Multiplier

Micro-operations for each register

Block Diag Module	Micro-Operation	Control Signal	Control Expression
Register A	$A \leftarrow 0$ $A \leftarrow A + B$ $C\|A\|Q \leftarrow sr\ C\|A\|Q$	Initialize Load Shift	
Register B	$B \leftarrow IN$	Load_B	
FF C	$C \leftarrow 0$ $C \leftarrow Cout$	Clear_C Load	
Register Q	$Q \leftarrow IN$ $C\|A\|Q \leftarrow sr\ C\|A\|Q$	Load_Q Shift	
Counter P	$P \leftarrow n - 1$ $P \leftarrow P - 1$	Initialize Decrement Count	

Same control signal for different registers:

Derive control signal logic from ASM

Initialize:

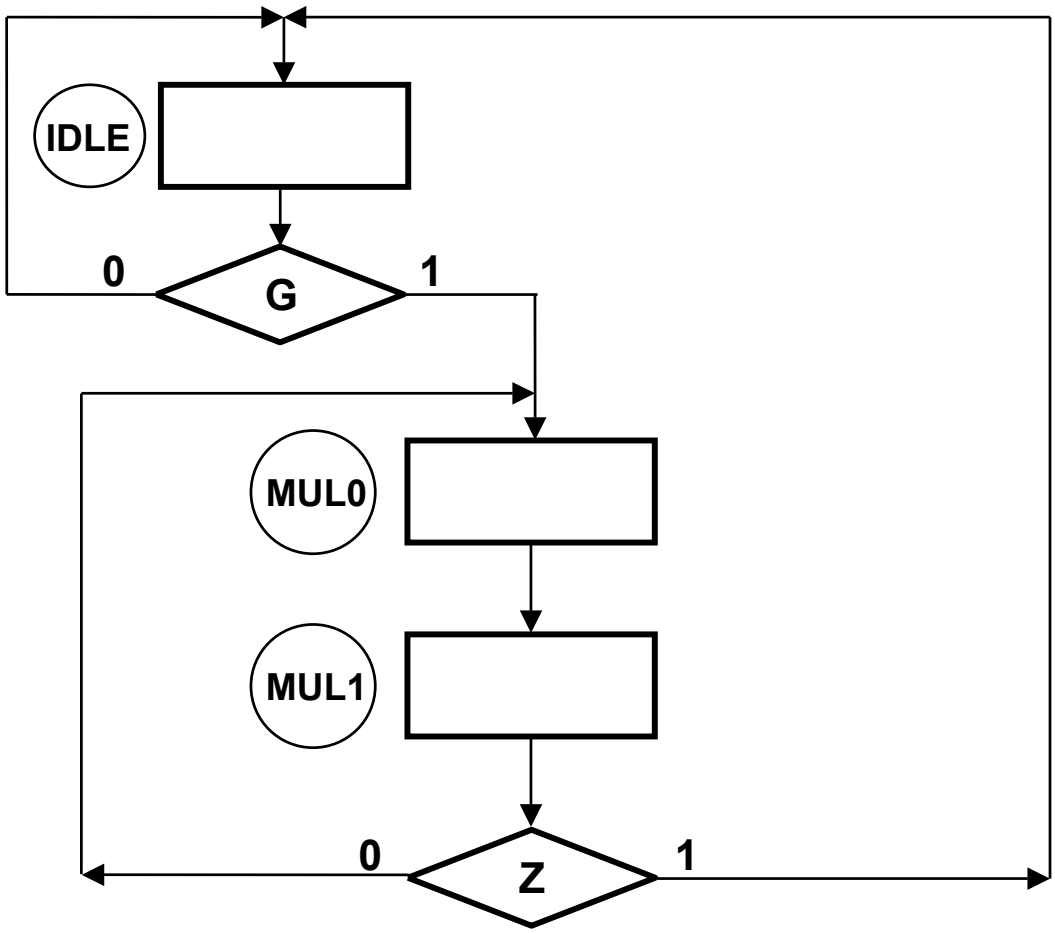
Clear_C:

Remove information on micro-operations and redraw the ASM for sequencing purposes only.

ASM for Sequencing Part

- Remove any decision boxes that do not affect the next state situation.

- Remove all the output boxes and any outputs in the state boxes.
- Design the sequencing part of the control unit using the simplified ASM chart



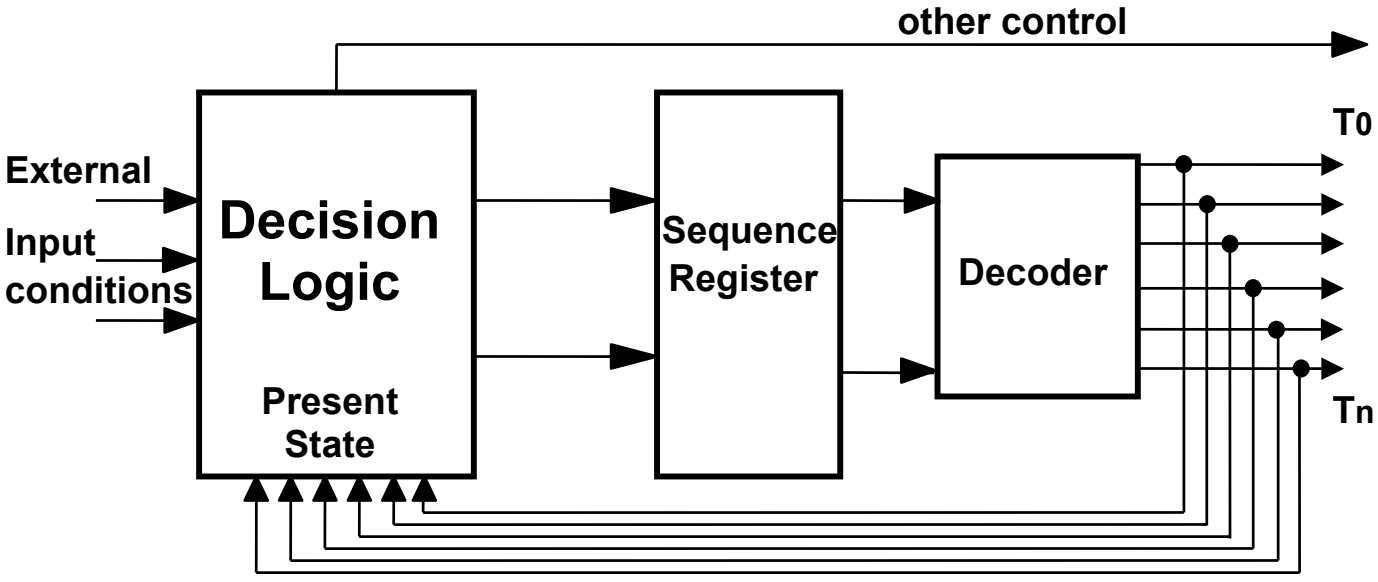
Sequence Register and Decoder Method

Sequence Register for control states

- Register with n-Flip-Flops can have 2^n states.
- n-bit sequence register has n-FFs & associated gates.

Decoder provides outputs corresponding to each state

Combination of the external inputs and feedback from the present state generates the next states.



If there is no external input then it reduces to a counter decoder control circuit.

Sequence Register and Decoder Method

Binary Multiplier Control Sequencer

- 3-states and 2-inputs:

State Table

Present State			**Inputs**		**N. State**		**Decoder**		
Name	M_1	M_0	G	Z	M_1^+	M_0^+	IDLE	MUL0	MUL1
IDLE	0	0	0	x			1	0	0
	0	0	1	x			1	0	0
MUL0	0	1	x	x			0	1	0
MUL1	1	0	x	0			0	0	1
	1	0	x	1			0	0	1
	1	1	x	x			x	x	x

2 Flip-flops : M1 M0

States 00, 01 and 10: IDLE, MUL0 and MUL1

$D_{M0} = M0^+$

$D_{M1} = M1^+$

Outputs: Initialize, Clear_C, Shift and Load

Initialize and Shift already available

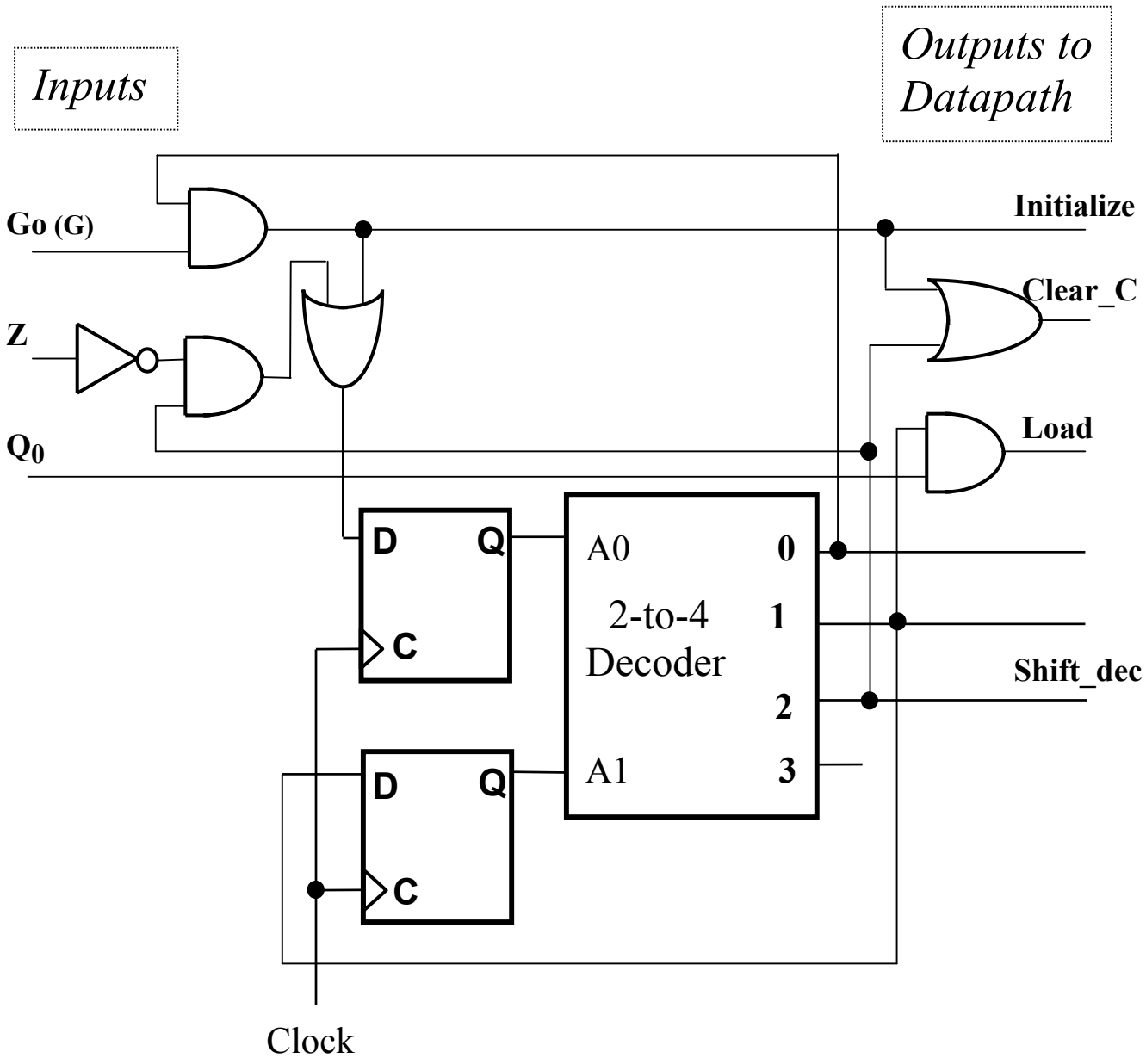
Gates required for Clear_C and Load

Clear_C =

Load =

Sequence Register and Decoder Method

Implementation



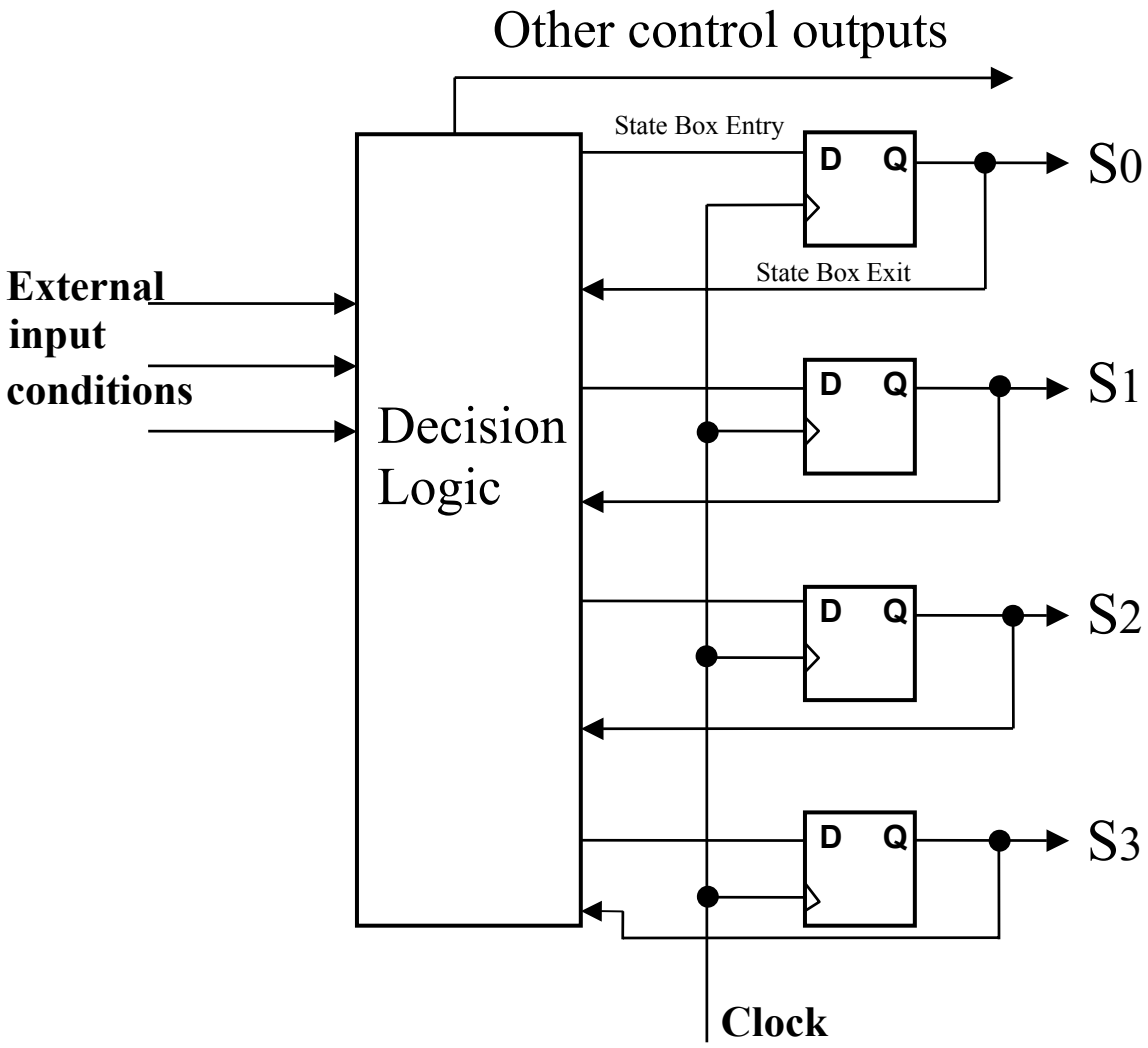
Multiplier VHDL Code

```
  -- A behavioral model of a multiplier for
  -- unsigned binary-numbers that multiplies a
  -- 4-bit multiplicand by a 4-bit multiplier
  -- to give an 8-bit product.

  -- The maximum number of clock cycles needed
  -- for a multiply is 10.


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity mult4X4 is
port ( Clk, St: in std_logic;
       Mplier, Mcand:
              in std_logic_vector(3 downto 0);
        Done: out std_logic);
end mult4X4;

architecture behave1 of mult4X4 is
  signal State: integer range 0 to 9;

-- accumulator
signal ACC: std_logic_vector(8 downto 0);

-- Q0 is bit 0 of ACC
alias Q0: std_logic is ACC(0);
begin
```

```
process
  begin
-- executes on rising edge of clock
  wait until Clk = '1';
  case State is
   when 0 =>                    --initial State
    if St='1' then
     ACC(8 downto 4)<= "00000"; --Begin cycle
      -- Load multiplier
     ACC(3 downto 0) <= Mplier;
      State <= 1;
    end if;
    when 1 | 3 | 5 | 7  => --"add/shift" State
     if Q0 = '1' then      --Add multiplicand
       ACC(8 downto 4) <=
       add4(ACC(7 downto 4),Mcand,'0');
        State <= State + 1;
        else         -- Shift accumulator right
           ACC <= '0' & ACC(8 downto 1);
           State <= State + 2;
     end if;
    when 2 | 4 | 6 | 8 =>     --"shift" State
      -- Right shift
     ACC <= '0' & ACC(8 downto 1);
     State <= State + 1;
    when 9 =>                  -- End of cycle
     State <= 0;
   end case;
end process;
Done <= '1' when State = 9 else '0';
end behave1;
```

One Flip-Flop per State Method

Every state is assigned to one flip-flop.



The configuration of the 4-state control logic:

- Four D-type flip-flops
- One flip-flop for every state
- Only one flip-flop will be active (level HIGH) at any one time.

One Flip-Flop per State Method

Only one flip-flop is in active state or "1" at a time that signifies one state.

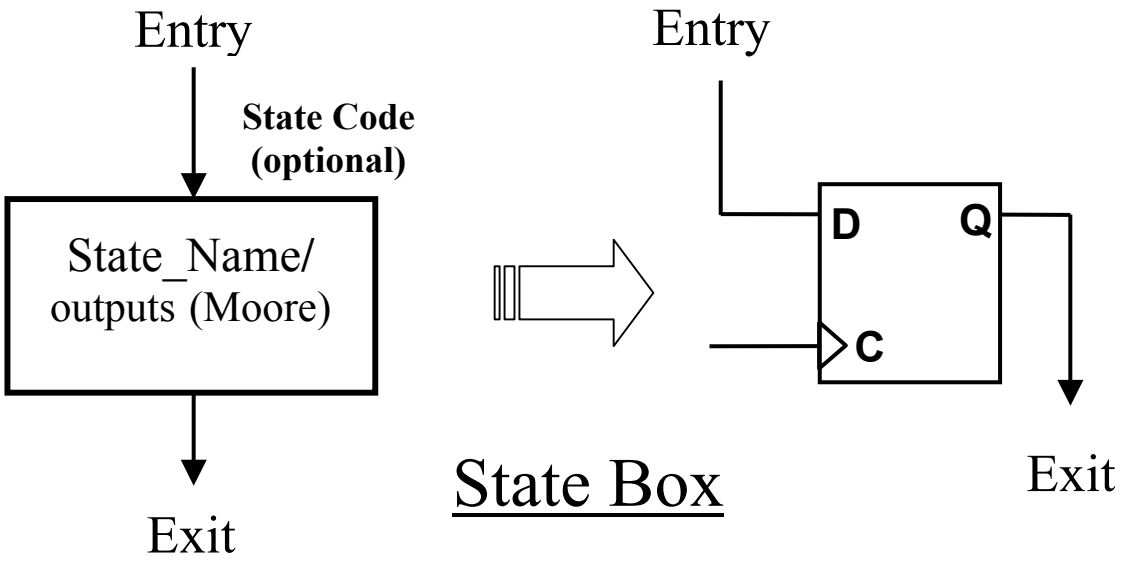
Main Features:

- The simplicity allows designers to design controller only by inspection from the ASM chart or state diagram.
- Cost saving in the design effort of controllers, however, it is not recommended for high volume production.
- Large number of flip-flops leads to high cost.
- Each of the flip-flop output is connected to the data-processing section of the digital system, to initiate certain micro-operations.
- If controllers do not have any input and the control needs to be repeated then it becomes a ring counter controller.

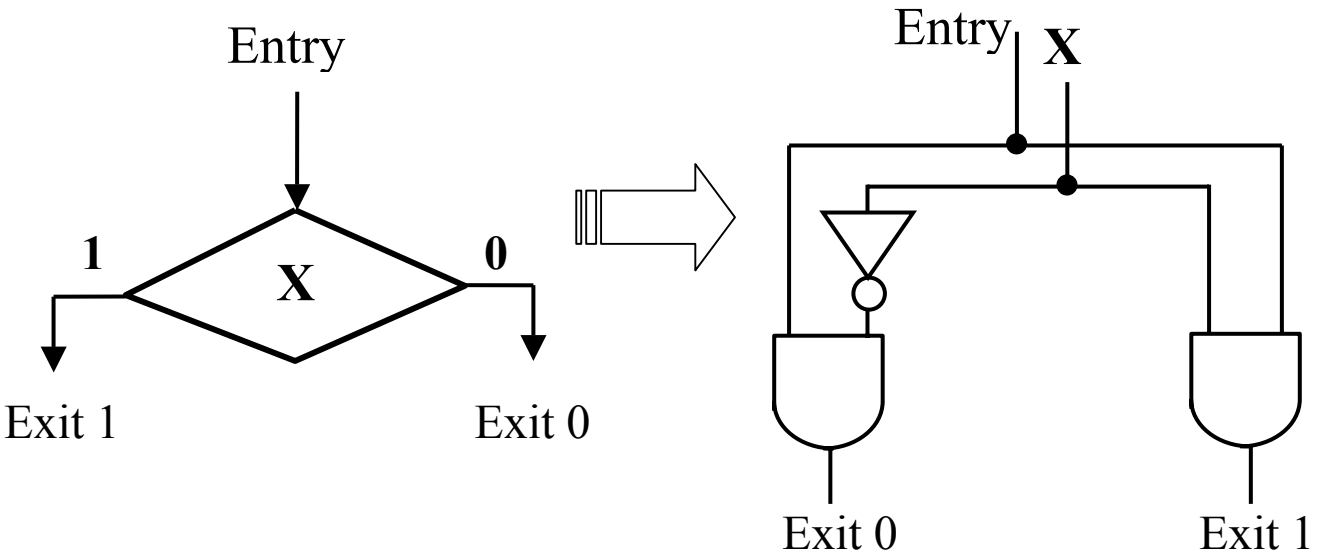
One FF/State Implementation

Suited for Implementing Control Unit from ASM charts.

ASM Transforming Rules



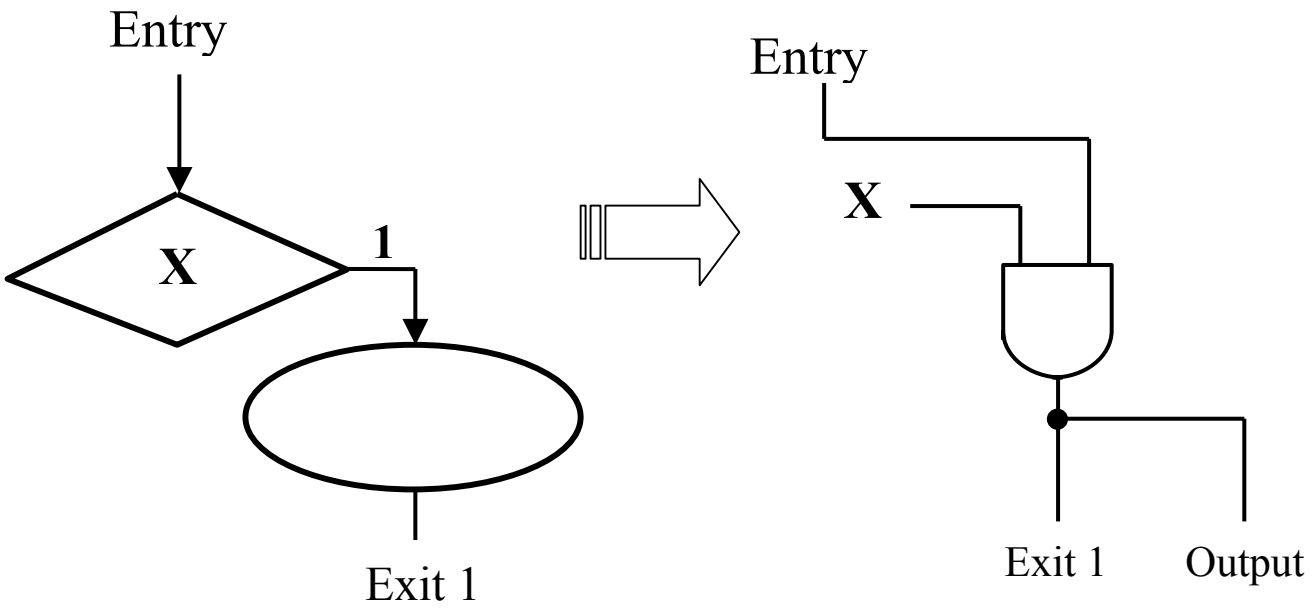
State Box



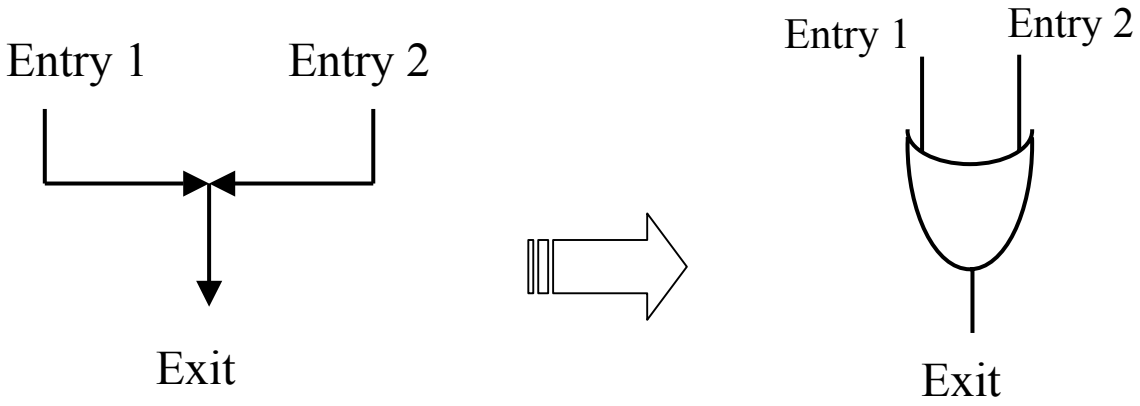
Decision Box

One FF/State Implementation

ASM Transforming Rules



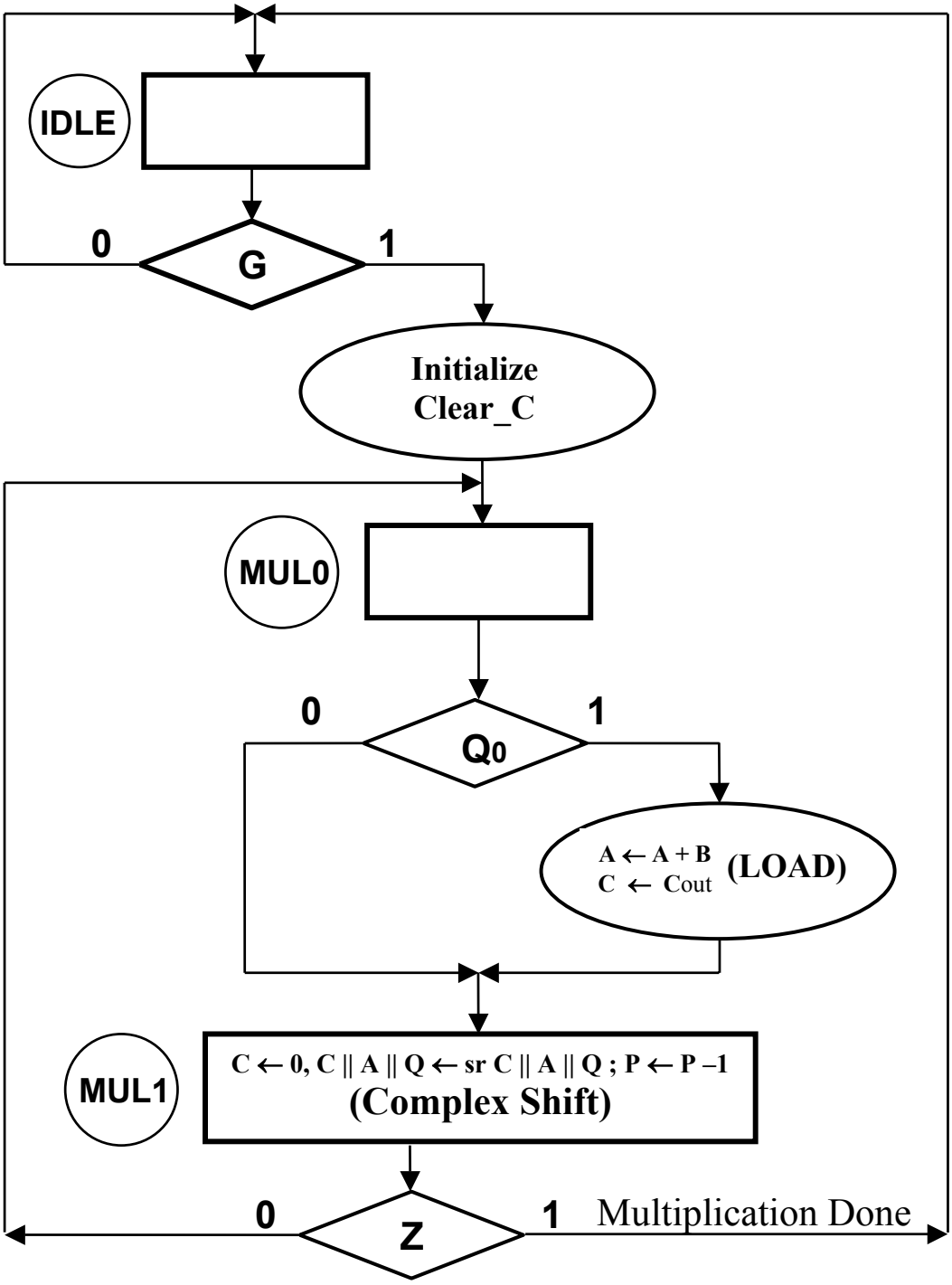
Conditional Output Box



Junction

One FF/State Implementation

ASM Chart for Binary Multiplier Control



One FF/State Implementation