SM Charts and Microprogramming

Objectives

- Introduction
- State Machine Charts
- Derivation of SM Charts (hardwiring)
  - Binary Multiplier Controller
  - Dice-Game
- Realization of SM Charts
- Implementation of the Dice Game
- Microprogramming
- Linked State Machines
- Summary

Introduction

- A state machine is often used to control a digital system that carries out a step-by-step procedure or algorithm.
- State diagrams or state graphs with circles representing states and arcs representing transitions have traditionally been used to specify the operation of the controller state machine.
- An alternative to using state graphs, a special type of flowchart, called a state machine chart (SM chart), may be used to describe the behavior of a state machine. These charts are also called algorithmic state machine charts (ASM charts).
- SM charts are often used to design control units for digital systems.

SM Charts

- Resemble software flowcharts.
- Useful in software design as well as hardware design.
- Advantages over state graphs:
  - Easier to understand compare to equivalent state graph.
  - Conditions of a state graph (1. one and exactly one transition from a state must be true at any time and 2. the next state must be uniquely defined for every input combination.) are automatically fulfilled in an SM chart.
  - Directly interprets to a hardware realization.
- May be converted into different equivalent forms resulting in different implementations. Hence, designer may optimize and transform SM charts to suit the required implementation style/technology.

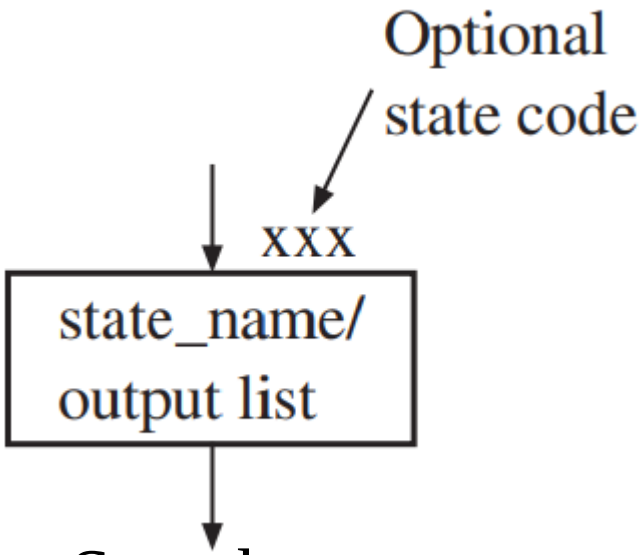
SM Charts (continued)

An SM chart differs from an ordinary flowchart in that certain specific rules must be followed in constructing the SM chart. When these rules are followed, the SM chart is equivalent to a state graph, and it leads directly to a hardware realization.

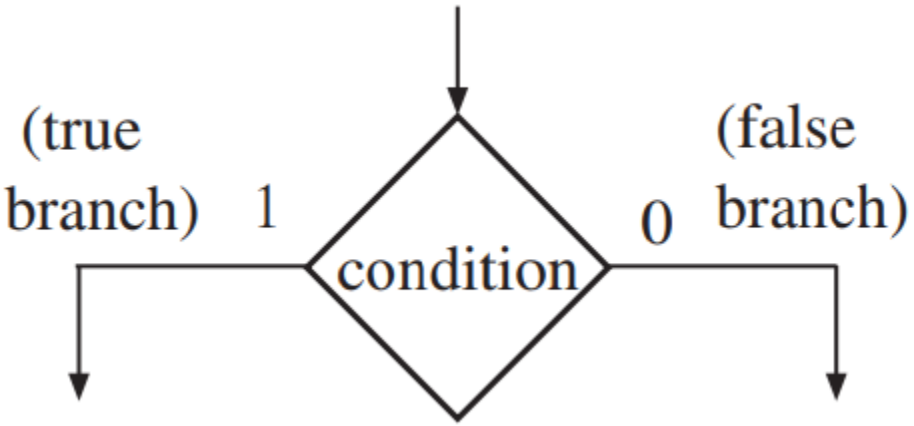
- Three components:
  - State box: rectangular box; contains a state name, followed by a slash (/) and an optional output list. After a state assignment has been made, a *state code* may be placed outside the box at the top.
  - Decision box: a diamond-shaped symbol with true and false branches. The condition placed in the box is a Boolean expression that is evaluated.
  - Conditional output box: box with curved ends; contains a conditional output list. The conditional outputs depend on both the state of the system and the inputs.

SM Charts (continued)

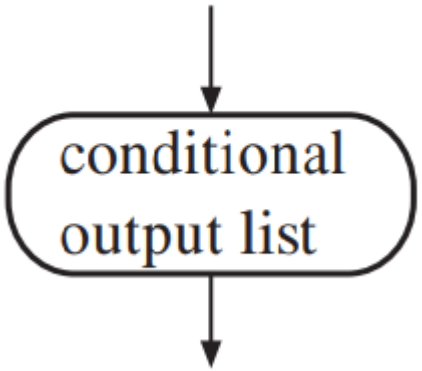
Principal Components of an SM Chart



State box



Decision box

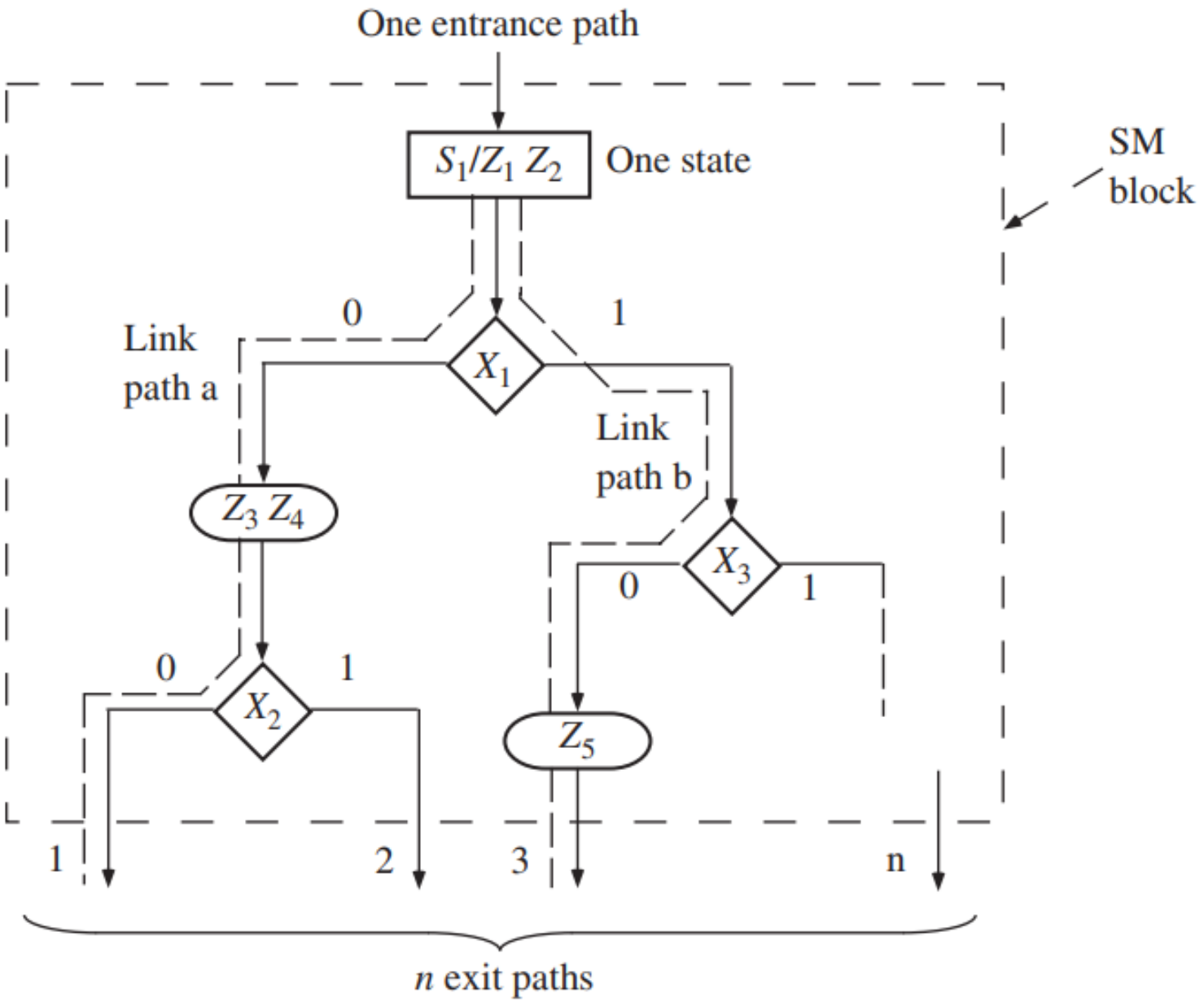


Conditional
output box

SM Charts (continued)

- An SM chart is constructed from SM blocks:
  - Each one contains exactly one state box, together with the decision boxes and conditional output boxes associated with that state.
  - Has one *entrance path* and one or more *exit paths*.
  - Each SM block describes the machine operation during the time that the machine is in one state.
  - When a digital system enters the state associated with a given SM block, the outputs on the output list in the state box become true.

SM Charts (continued)

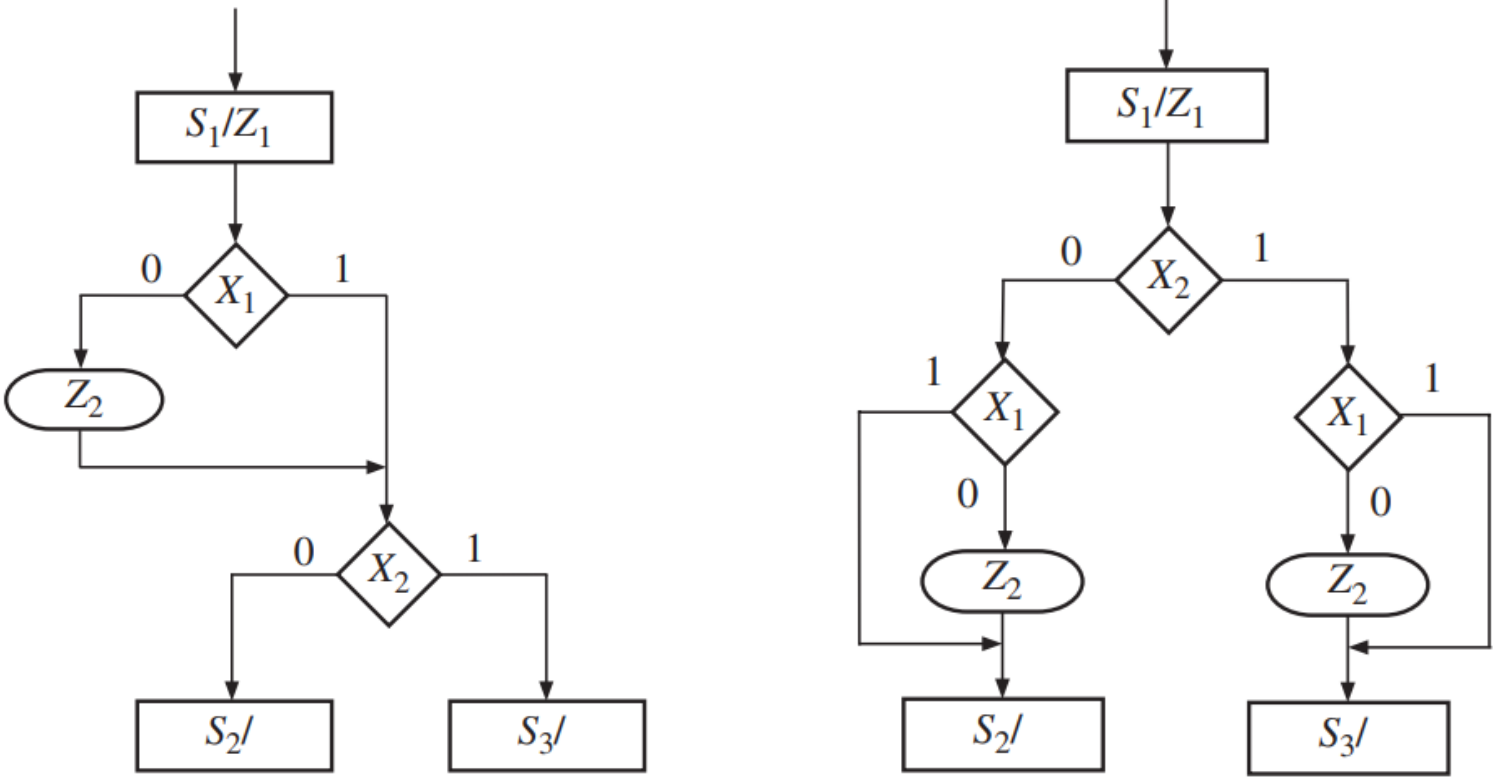


SM Charts (continued)

- Conditions in the decision boxes are evaluated.
- When a conditional output box is encountered along a path, the corresponding conditional outputs become true.
- If an output is not encountered along a path, that output is false by default.
- A path through an SM block from entrance to exit is referred to as a *link path*.

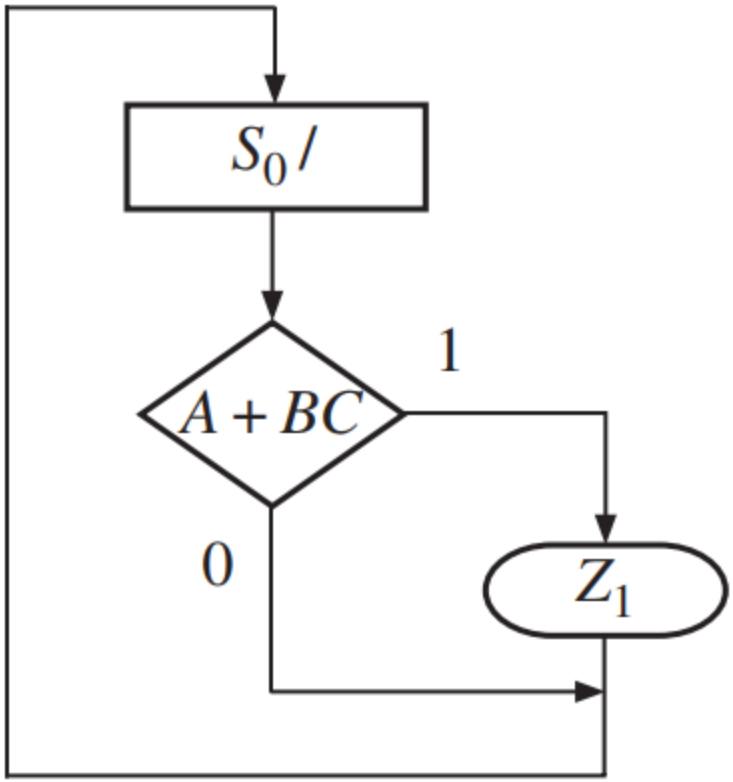
SM Charts (continued)

- Two equivalent SM blocks:
  - In both (a) and (b), the output $Z_2 = 1$ if $X_1 = 0$; the next state is S_2 if $X_2 = 0$ and S_3 if $X_2 = 1$.
  - The order in which the inputs are tested may affect the complexity of the SM chart.

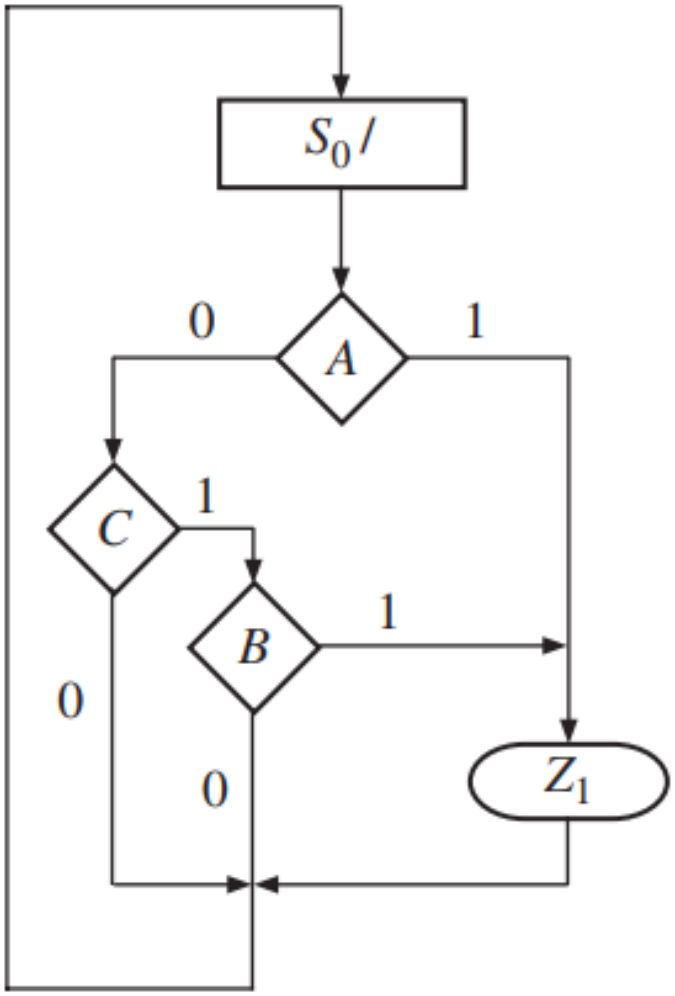


SM Charts (continued)

Equivalent SM Charts for a Combinational Circuit



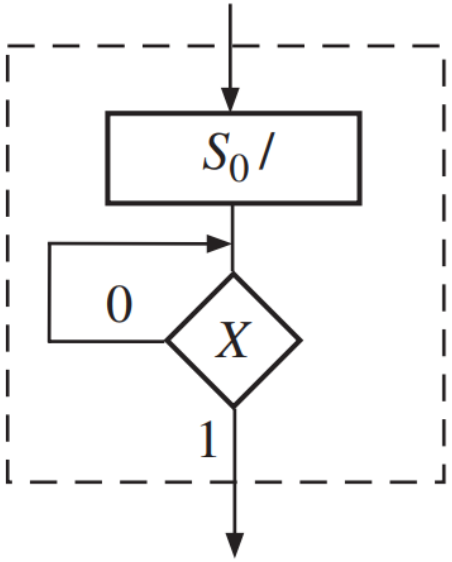
Z1 = A + BC



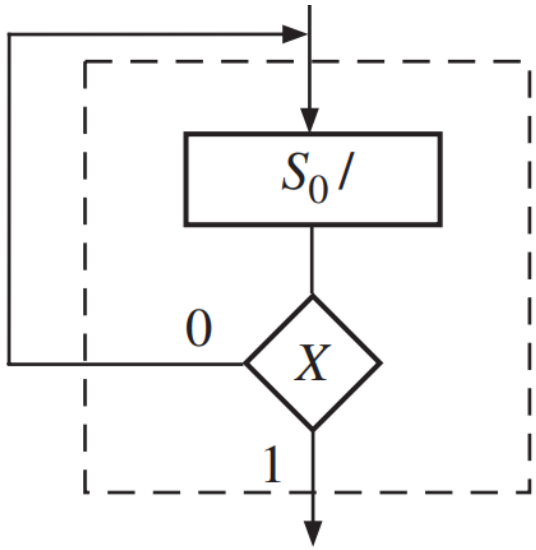
Z1 = A + A'BC = A + BC

SM Charts (continued)

- **Rules for creating an SM block:**
  - For every combination of input variables, there must be exactly one exit path defined. This is necessary since each allowable input combination must lead to a single next state.
  - No internal feedback within a block is allowed.



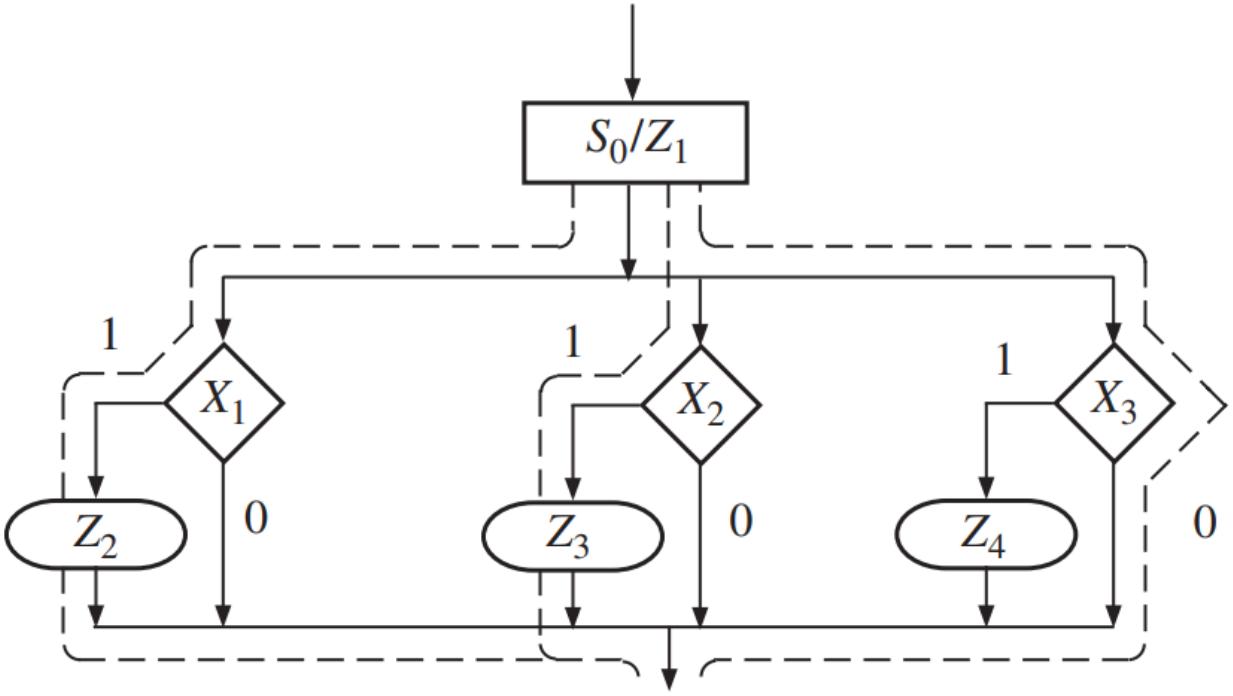
(a) Incorrect



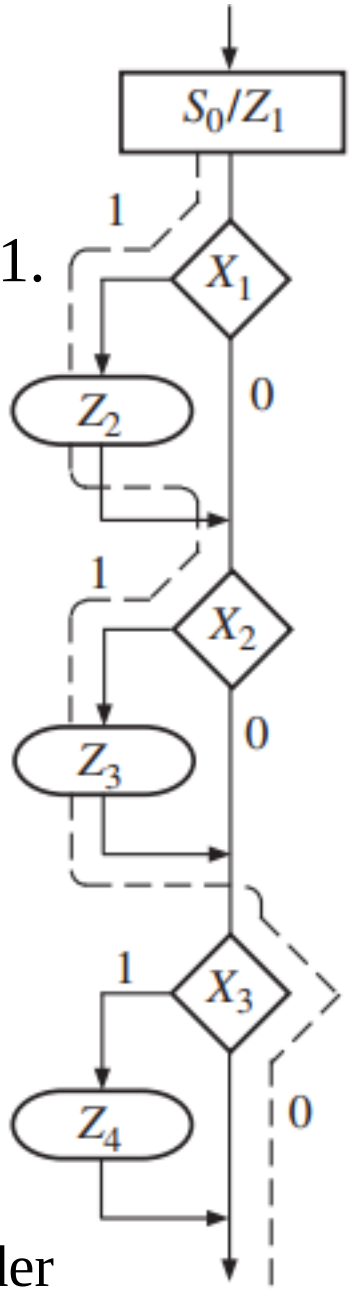
(b) Correct

SM Charts (continued)

If X1 = X2 = 1 and X3 = 0, the link paths marked with dashed lines are active, and the outputs Z1, Z2, and Z3 are 1.



(a) Parallel form



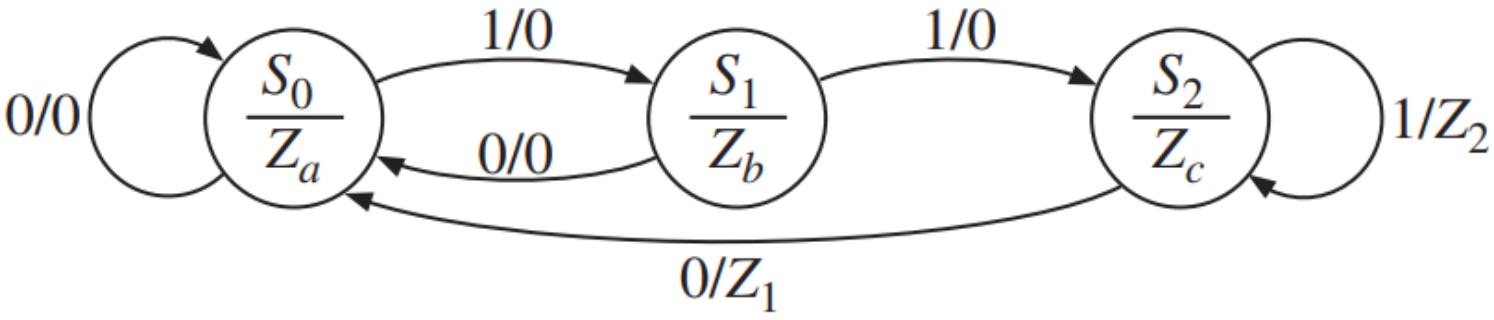
(b) Serial form

Regardless of whether the SM block is drawn in serial or parallel form, all the tests take place within one clock time. In the remainder of this text, **we use only the serial form for SM charts**.

SM Charts (continued)

Conversion of a State Graph to an SM Chart

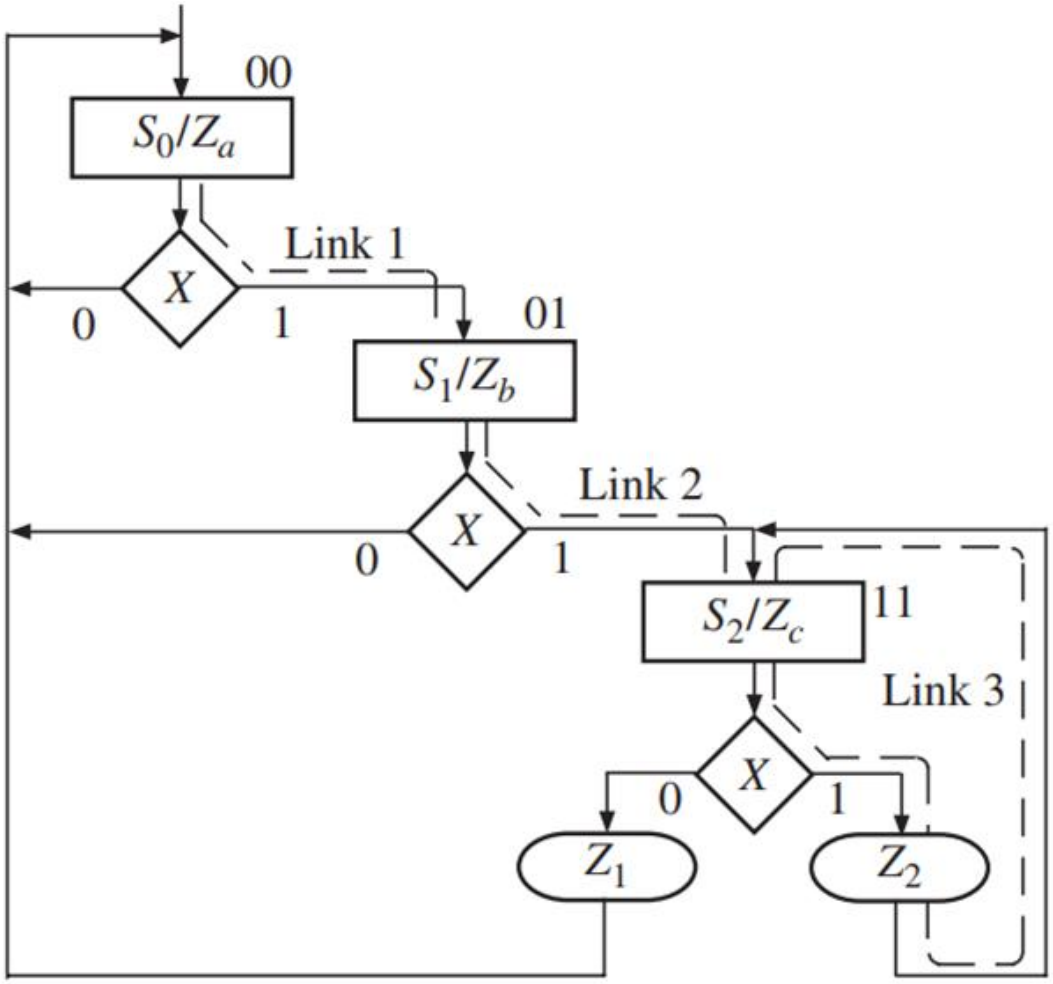
This state graph of has both Moore and Mealy outputs.



The Moore outputs (Za, Zb, Zc)

The Mealy outputs (Z1, Z2)

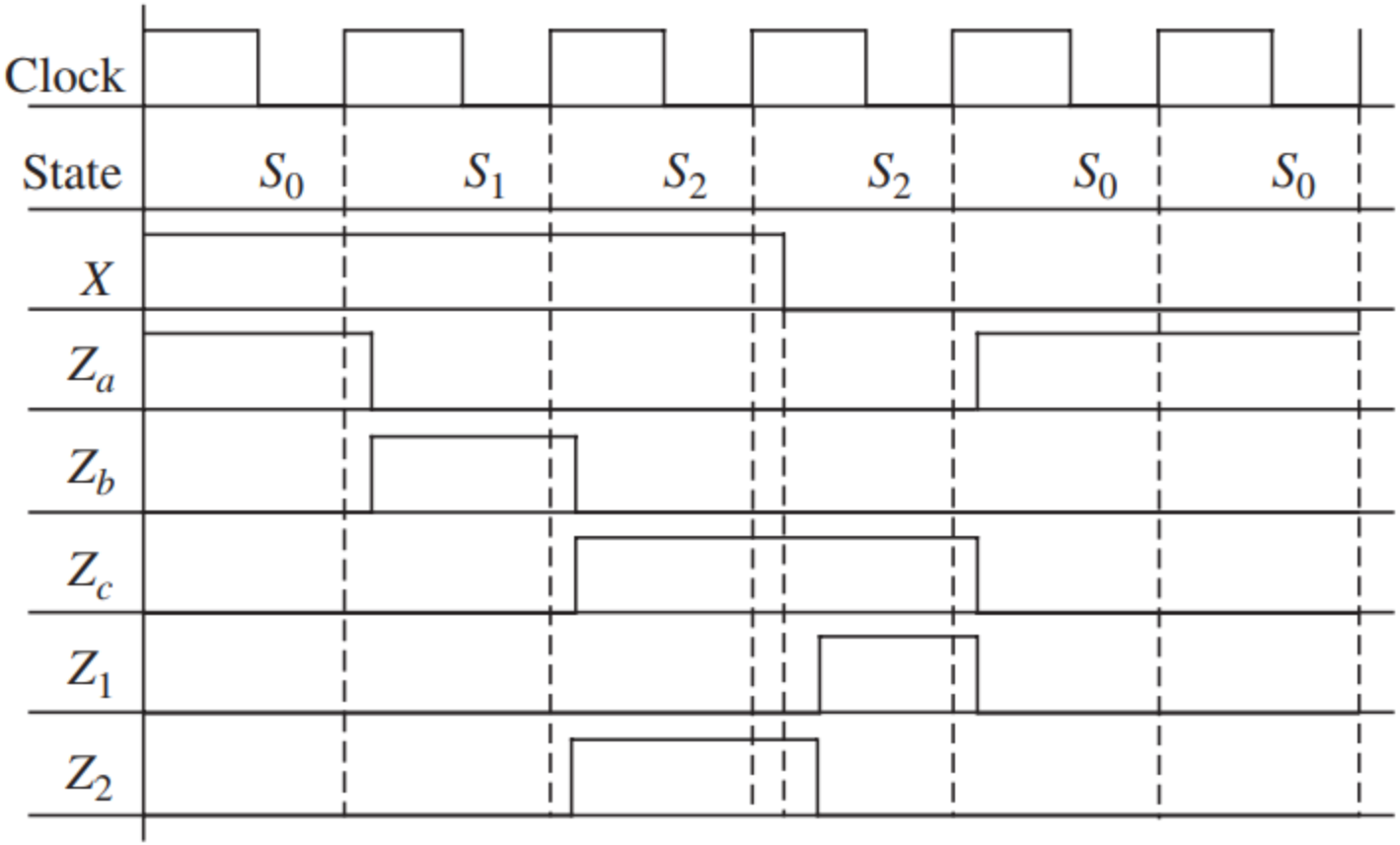
SM Charts (continued)



(b) Equivalent SM chart

SM Charts (continued)

An input sequence X = 1, 1, 1, 0, 0, 0.



Timing Chart

Derivation of SM Charts

- The method used to derive an SM chart for a sequential control circuit is similar to that used to derive the state graph:
  - 1. Draw a block diagram of the system one is controlling.
  - 2. Define the required input and output signals to the control circuit.
  - 3. Construct a SM chart that tests the input signals and generates the proper sequence of output signals.

Derivation of SM Charts: Binary Multiplier Example

The add-shift control generates the required sequence of add and shift signals. The counter counts the number of shifts and outputs K = 1 just before the last shift occurs.

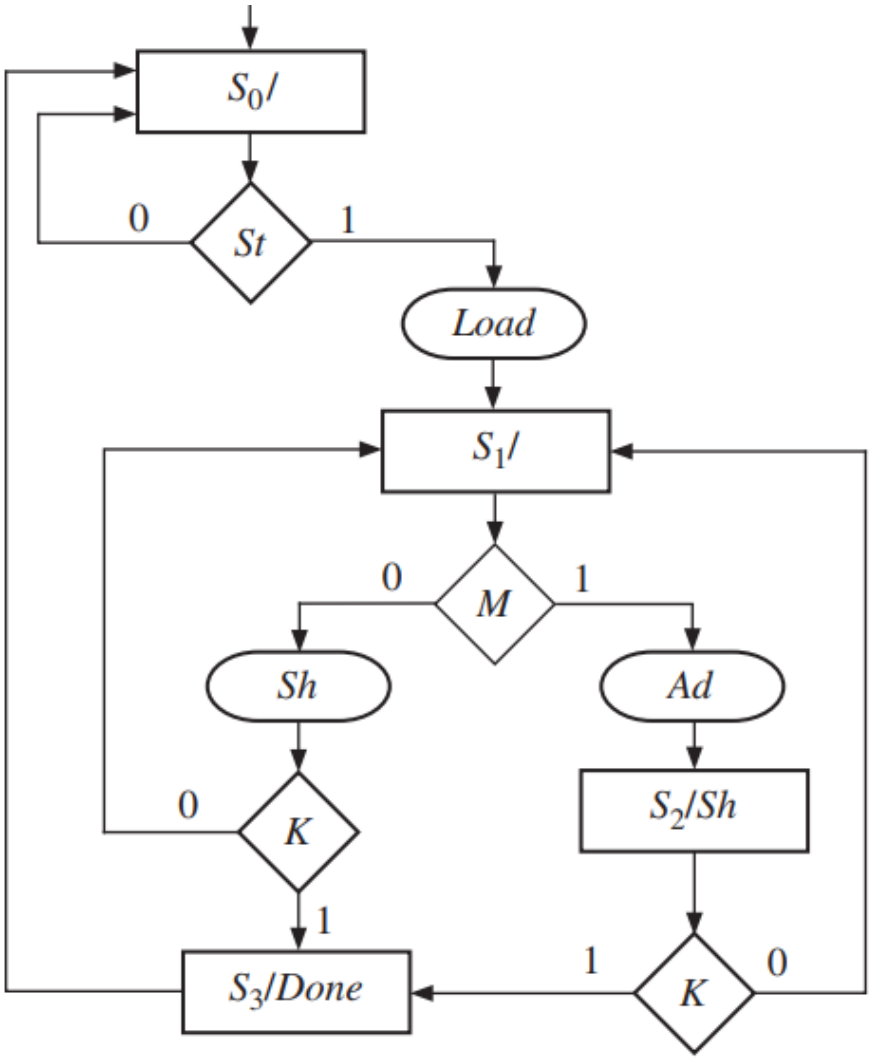
- Conversion of an SM chart to a Verilog process:
  - A case statement can be used to specify what happens in each state. Each condition box corresponds directly to an if statement (or an else).

Derivation of SM Charts: Binary Multiplier (continued)

- Verilog code: Two always statements are used:
  - 1. Represents the combinational part of the circuit.
  - 2. Updates the state register on the rising edge of the clock.
- The signals *Load*, *Sh*, and *Ad* are turned on in the appropriate states, and they must be turned off when the state changes; a way to do this is to set them all to 0 at the start of the process.

Derivation of SM Charts: Binary Multiplier (continued)

- SM chart:



Derivation of SM Charts: Binary Multiplier (continued)

```
module Mult (CLK, St, K, M, Load, Sh, Ad, Done);
  input CLK;
  input St;
  input K;
  input M;
  output Load;
  output Sh;
  output Ad;
  output Done;

  reg Ad;
  reg Sh;
  reg Load;
  reg Done;

  reg[1:0] State;
  reg[1:0] Nextstate;

  initial
  begin
       State = 0;
       Nextstate = 0;
  end

  always @(St or K or M or State)
  begin
       Load = 1'b0;
       Sh = 1'b0;
       Ad = 1'b0;
       Done = 1'b0;
```

Derivation of SM Charts: Binary Multiplier (continued)

```
case (State)
   0 :
            begin
               if (St == 1'b1) begin
                  Load = 1'b1;
                  Nextstate = 1;
               end
               else begin
                  Nextstate = 0;
               end
            end
   1 :
            begin
               if (M == 1'b1) begin
                  Ad = 1'b1;
                  Nextstate = 2;
               end
               else begin
                  Sh = 1'b1;
                  if (K == 1'b1) begin
                     Nextstate = 3;
                  end
                  else begin
                     Nextstate = 1;
                  end
               end
            end
```

Derivation of SM Charts: Binary Multiplier (continued)

```
            2 :
                        begin
                            Sh = 1'b1;
                            if (K == 1'b1) begin
                                Nextstate = 3;
                            end
                            else begin
                                Nextstate = 1;
                            end
                        end
            3 :
                        begin
                            Done = 1'b1;
                            Nextstate = 0;
                        end
        endcase
    end

    always @(posedge CLK)
    begin
        State <= Nextstate;
    end
endmodule
```