Figure 5-1 Components of an SM Chart

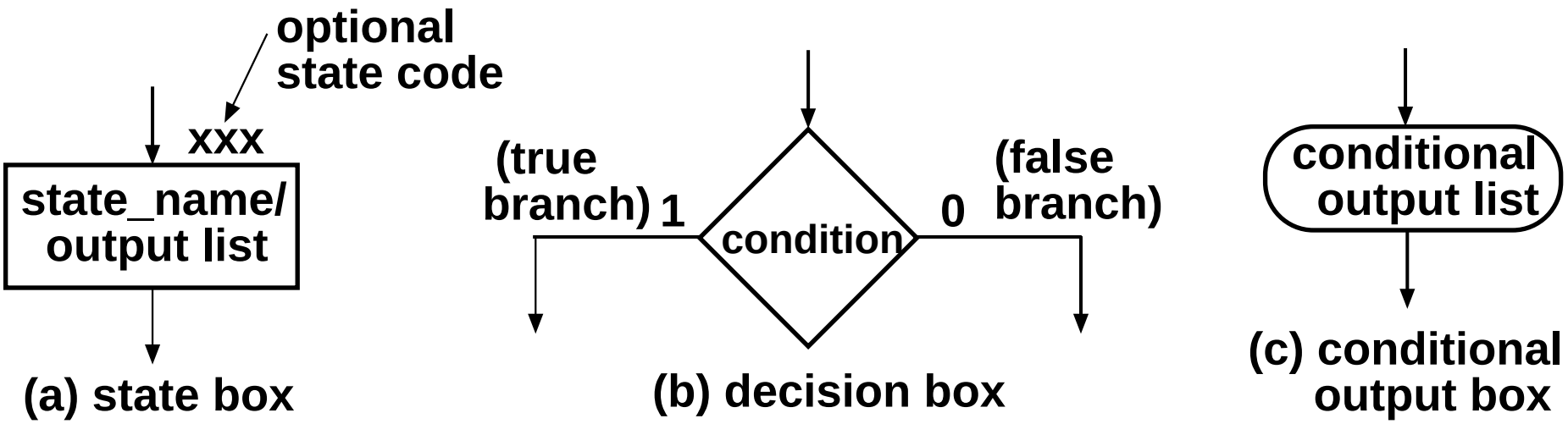


Figure 5-2 Example of an SM Block

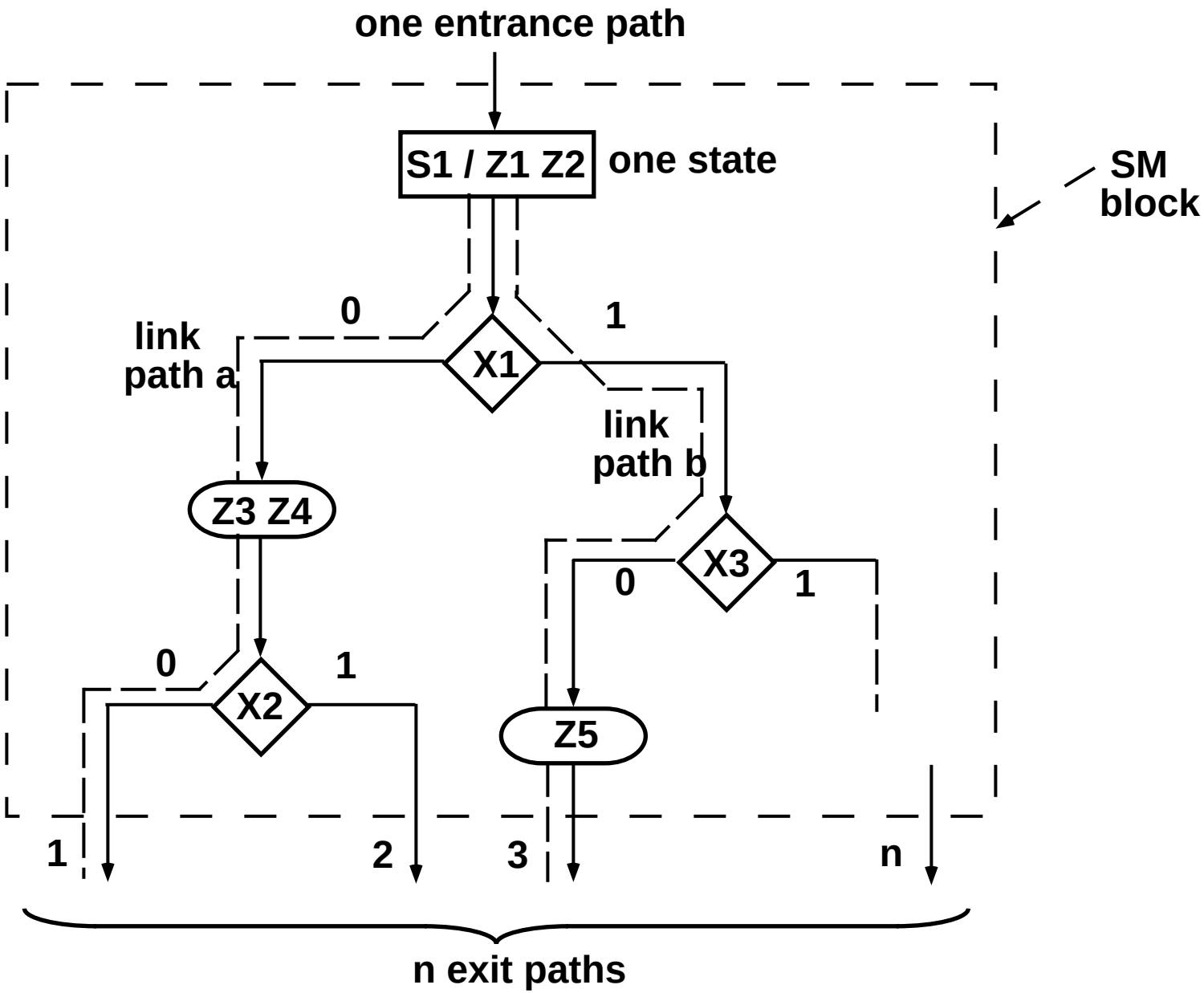


Figure 5-3 Equivalent SM Blocks

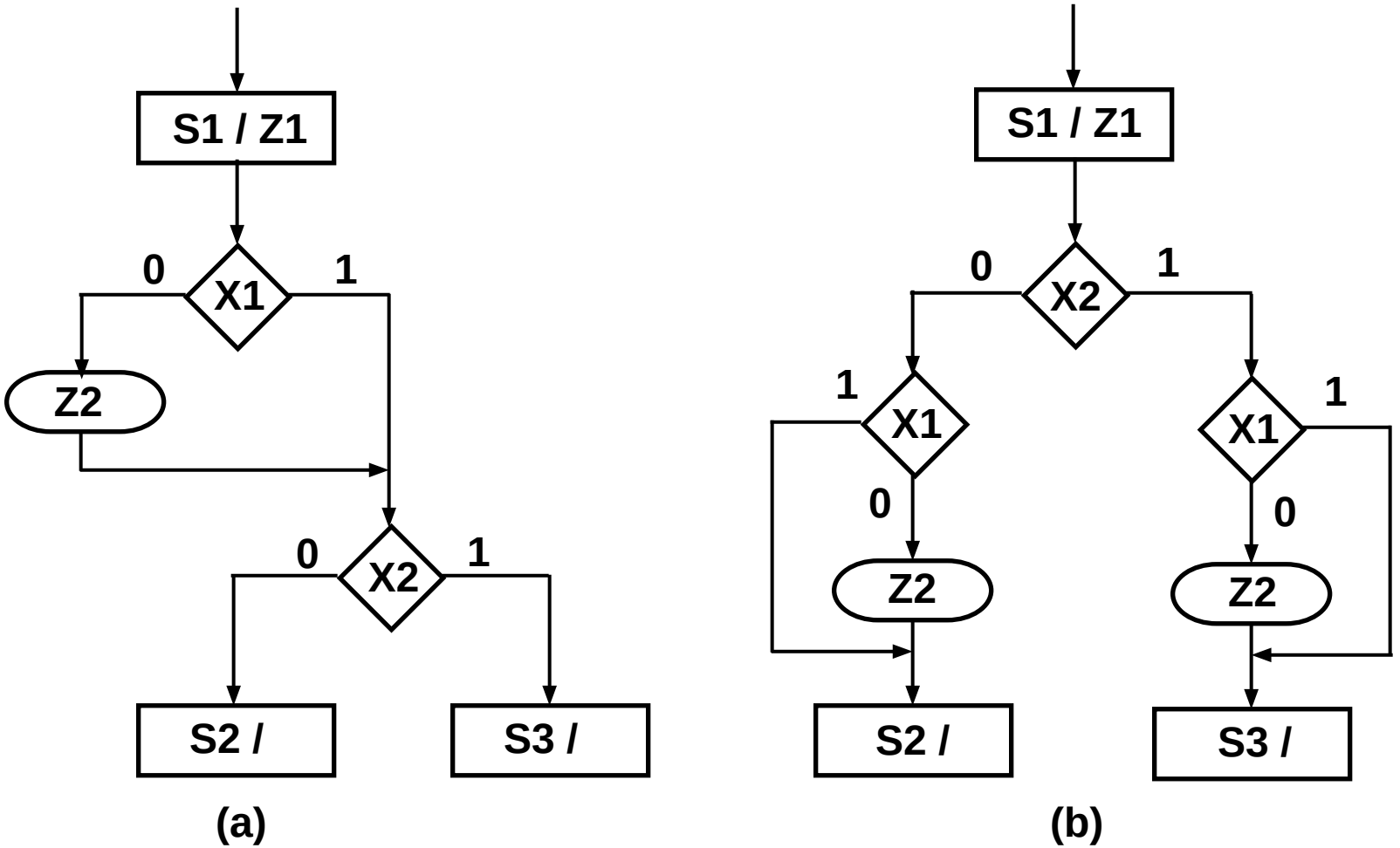


Figure 5-4: Equivalent SM Chart for a Combinational Network

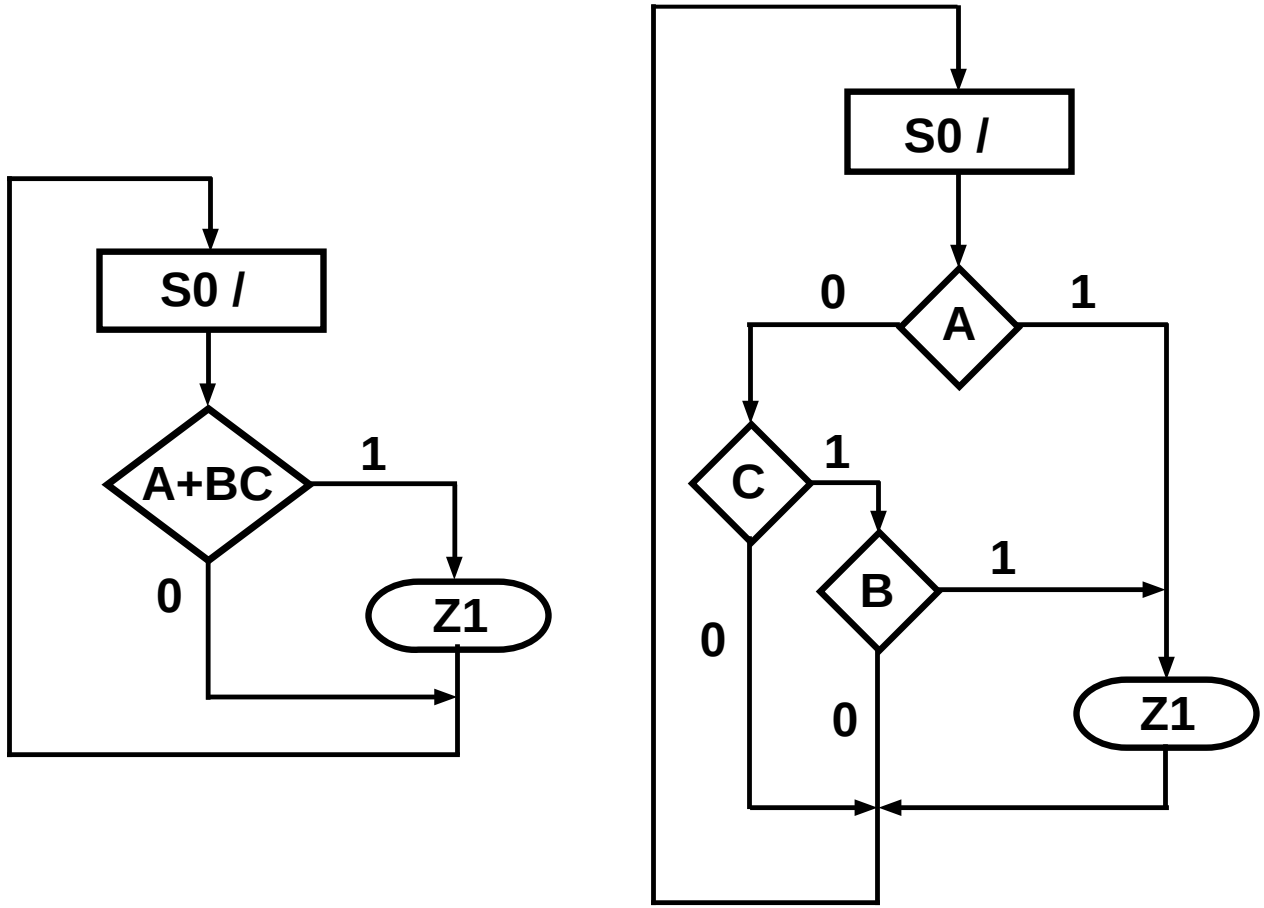


Figure 5-5 SM Block with Feedback

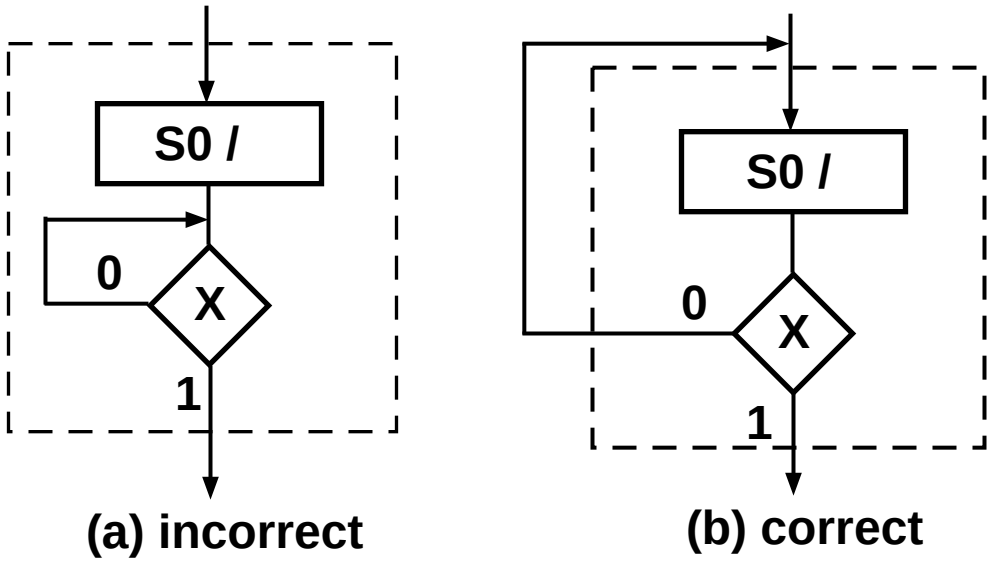
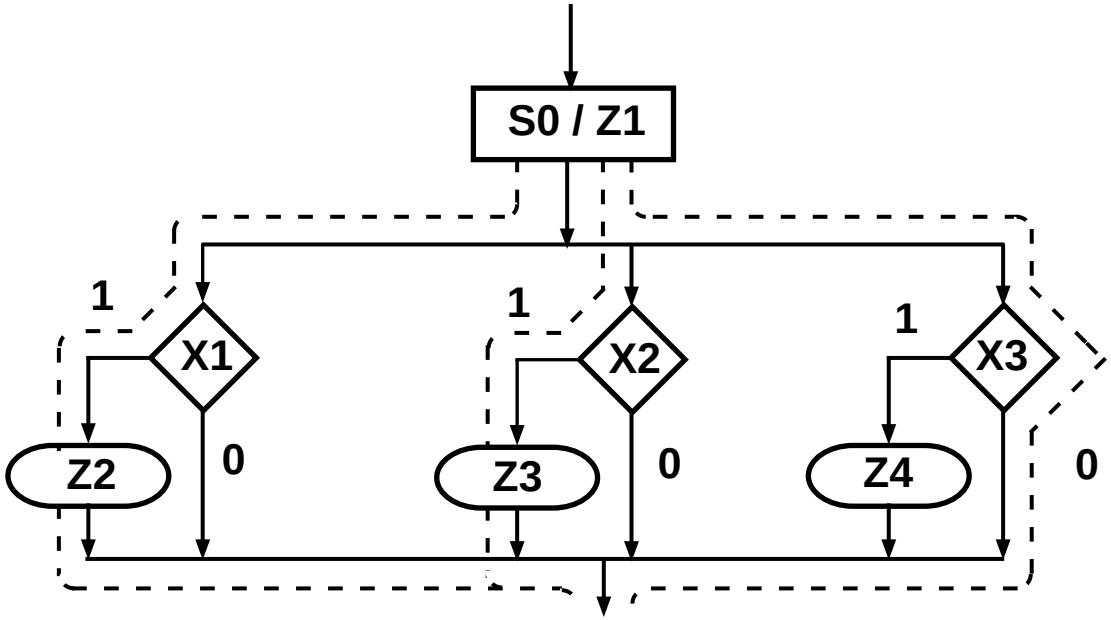
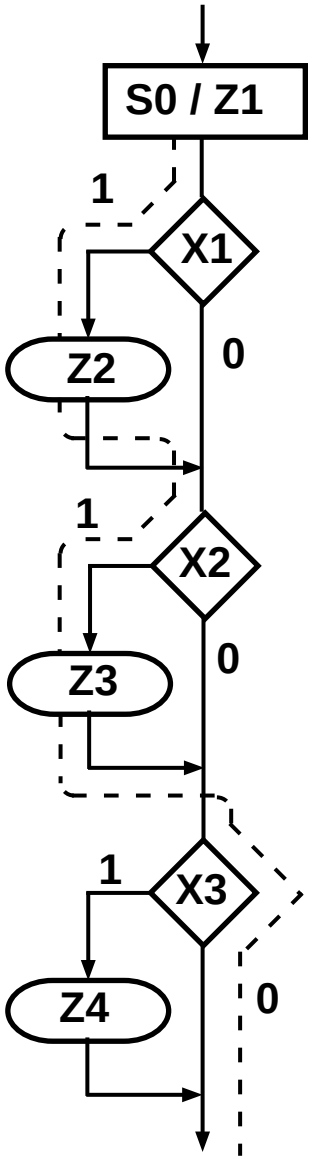


Figure 5-6 Equivalent SM Blocks



(a) Parallel form



(b) Serial form

Figure 5-7 Conversion of State Graph to an SM Chart

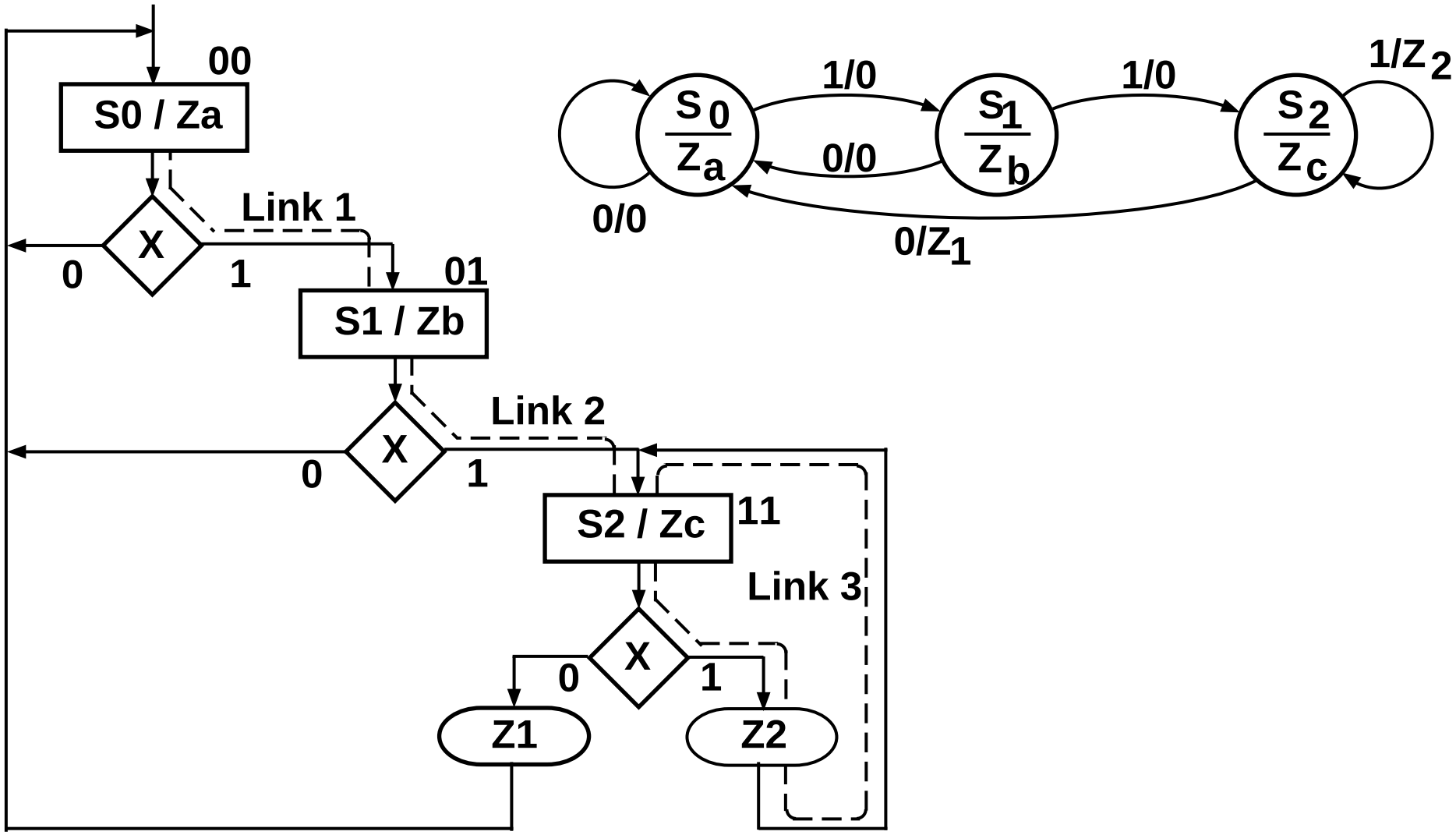


Figure 5-8 Timing Chart for Figure 5-7

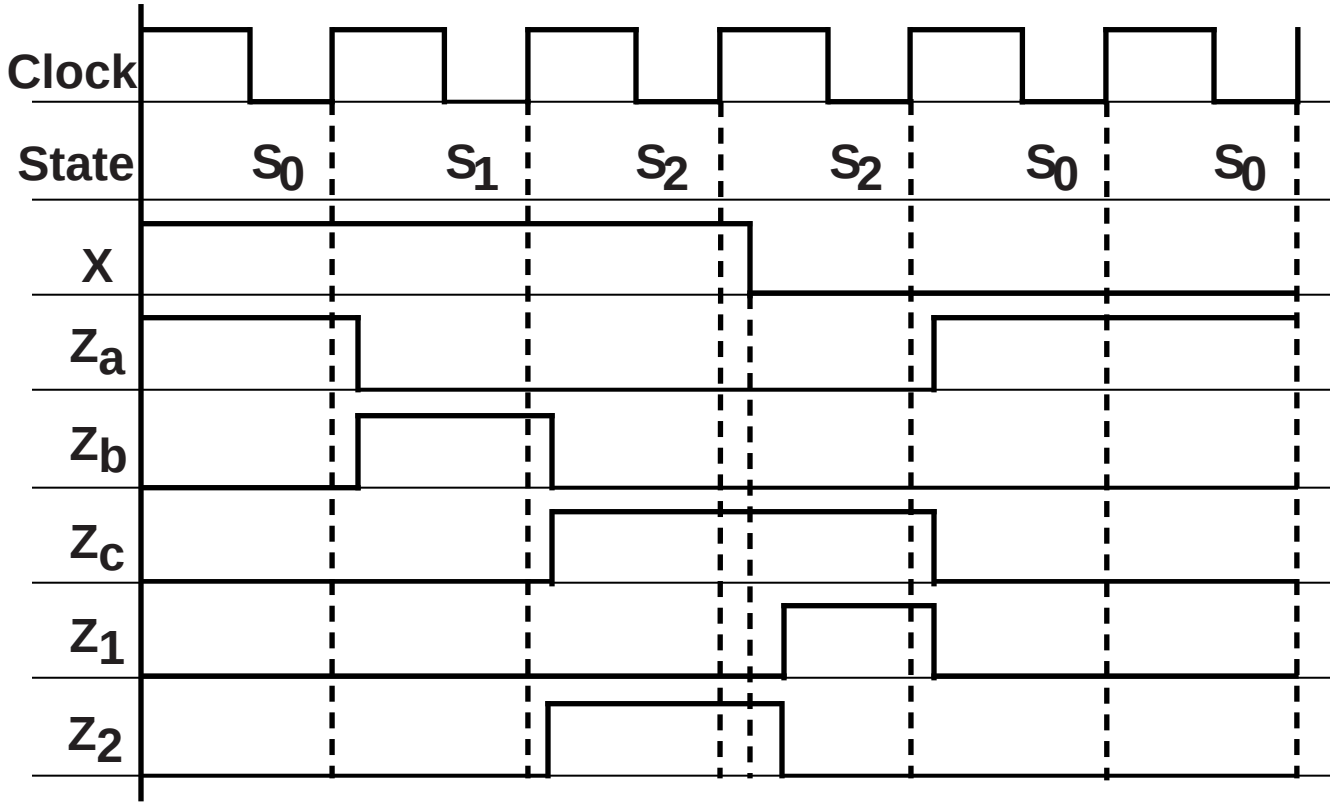


Figure 5-9 SM Chart for Binary Multiplier

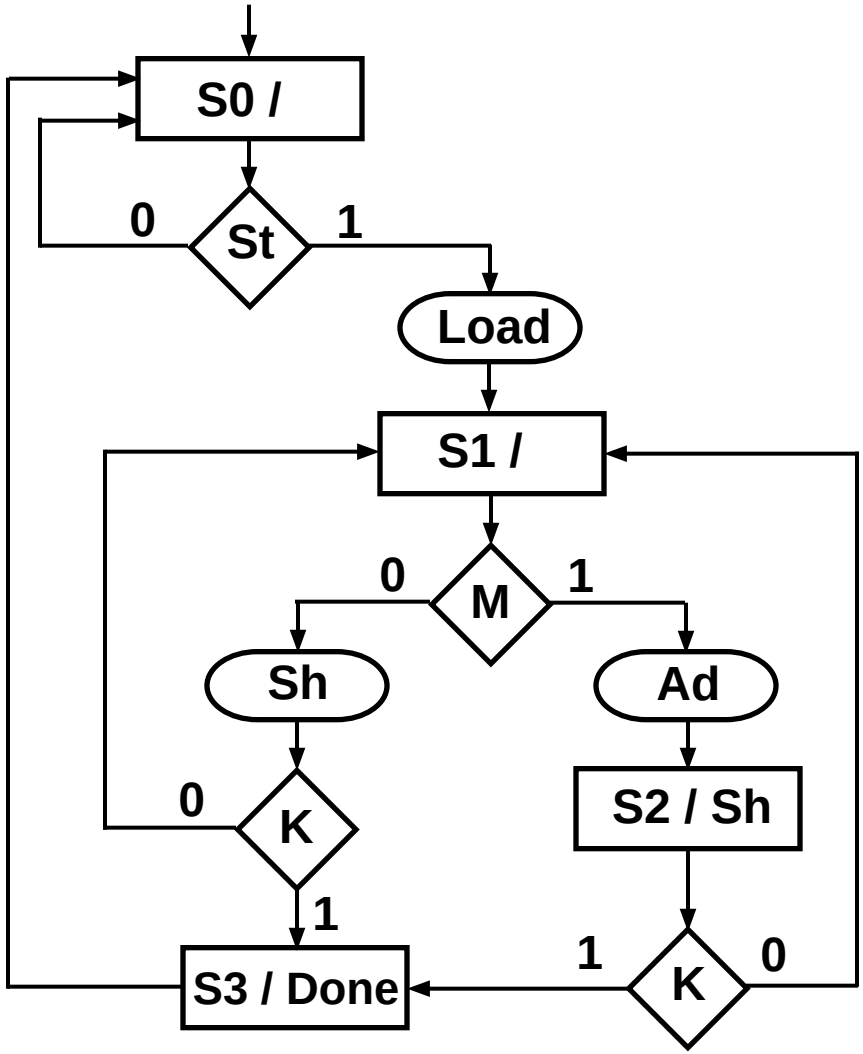


Figure 5-10(a) VHDL for SM Chart of Figure 5-9

```
entity Mult is
      port(CLK,St,K,M: in bit;
            Load,Sh,Ad,Done: out bit);
end mult;

architecture SMbehave of Mult is
      signal State, Nextstate: integer range 0 to 3;
begin
      process(St, K, M, State)                              -- start if state or inputs change
      begin
            Load <= '0'; Sh <= '0'; Ad <= '0';
            case State is
                  when 0 => if St = '1' then                -- St (state 0)
                              Load <= '1';  Nextstate <= 1;
                        else Nextstate <= 0;                -- St'
                        end if;
                  when 1 => if M = '1' then                 -- M (state 1)
                              Ad <= '1';  Nextstate <= 2;
                        else                                                -- M'
                              Sh <= '1';  if K = '1' then Nextstate <= 3;   -- K
                              else Nextstate <= 1;                          -- K'
                              end if;
                        end if;
                  when 2 => Sh <= '1';                      -- (state 2)
                              if K = '1' then Nextstate <= 3;     -- K
                              else Nextstate <= 1;                -- K'
                              end if;
```

Figure 5-10(b) VHDL for SM Chart of Figure 5-9

```
                when 3 => Done <= '1';                    -- (state 3)
                      Nextstate <= 0;
          end case;
     end process;
     process(CLK)
     begin
          if CLK = '1' then
                State <= Nextstate;                       -- update state on rising edge
          end if;
     end process;
end SMbehave;
```

Figure 5-11 Block Diagram for Dice Game

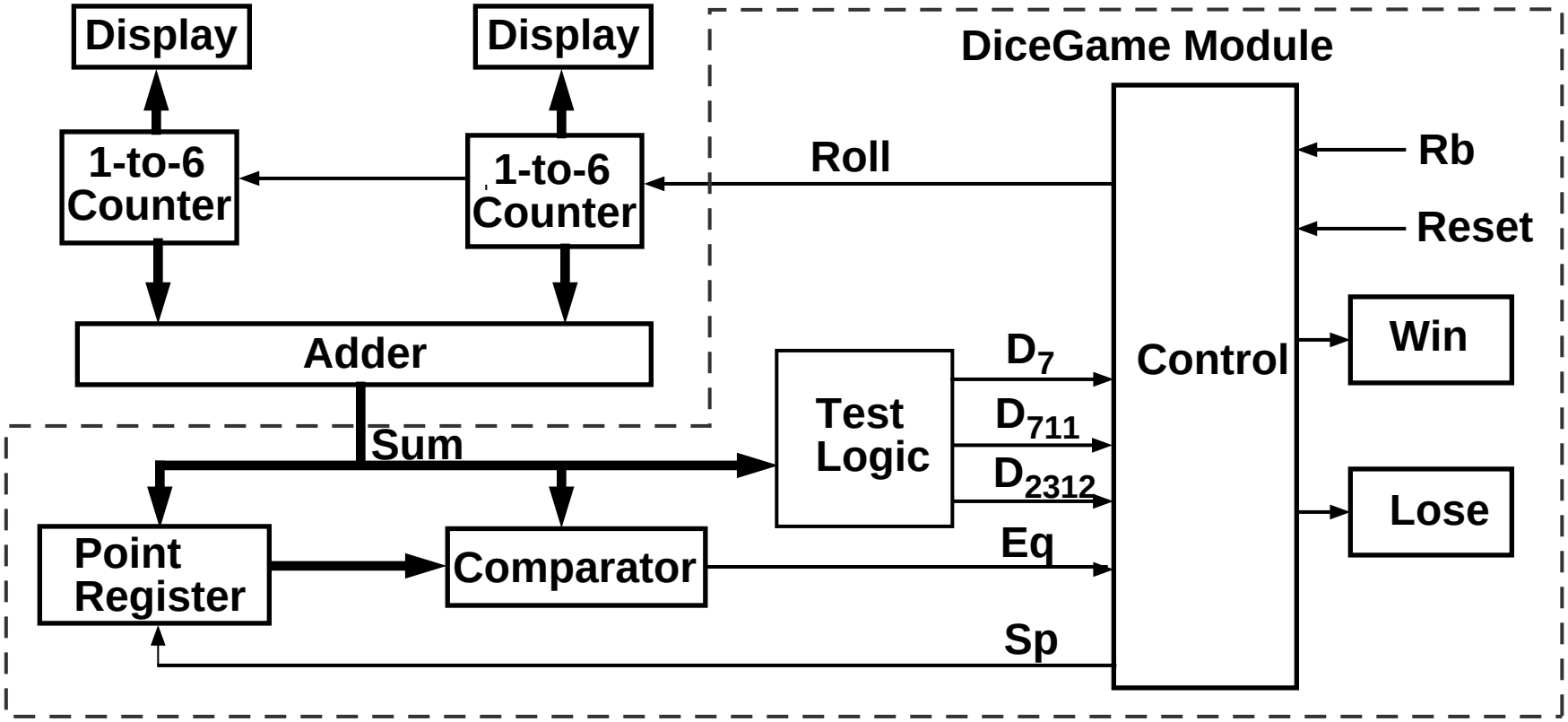


Figure 5-12 Flowchart for Dice Game

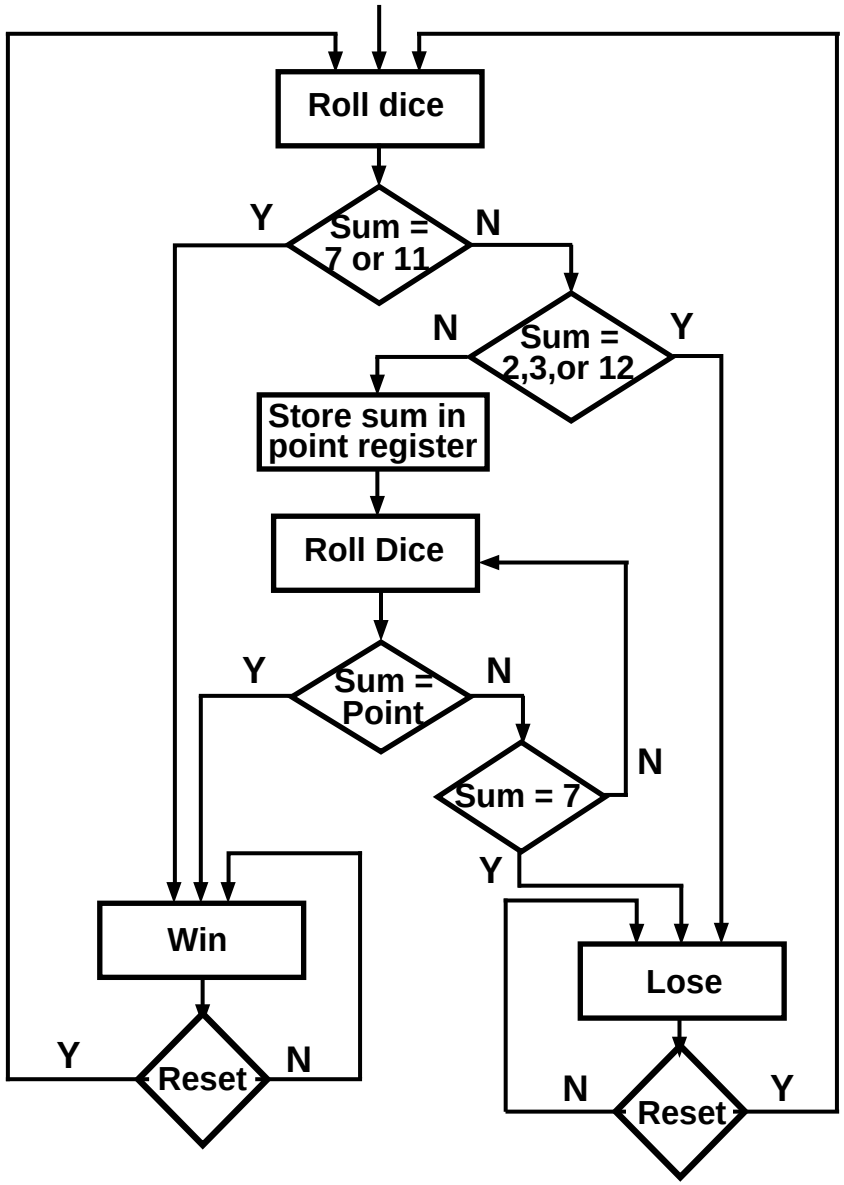


Figure 5-13 SM Chart for Dice Game

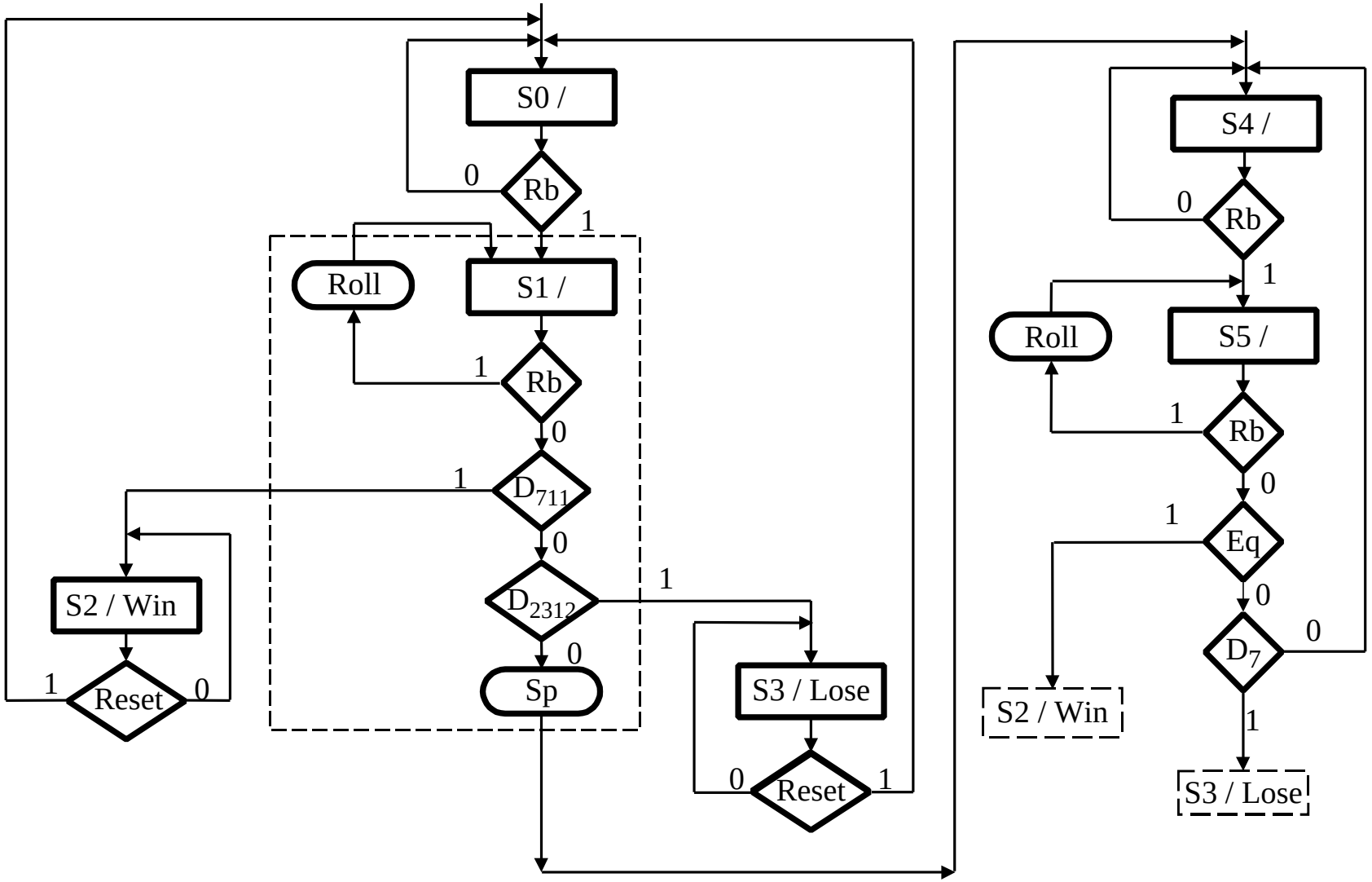


Figure 5-14 State Graph for Dice Game Controller

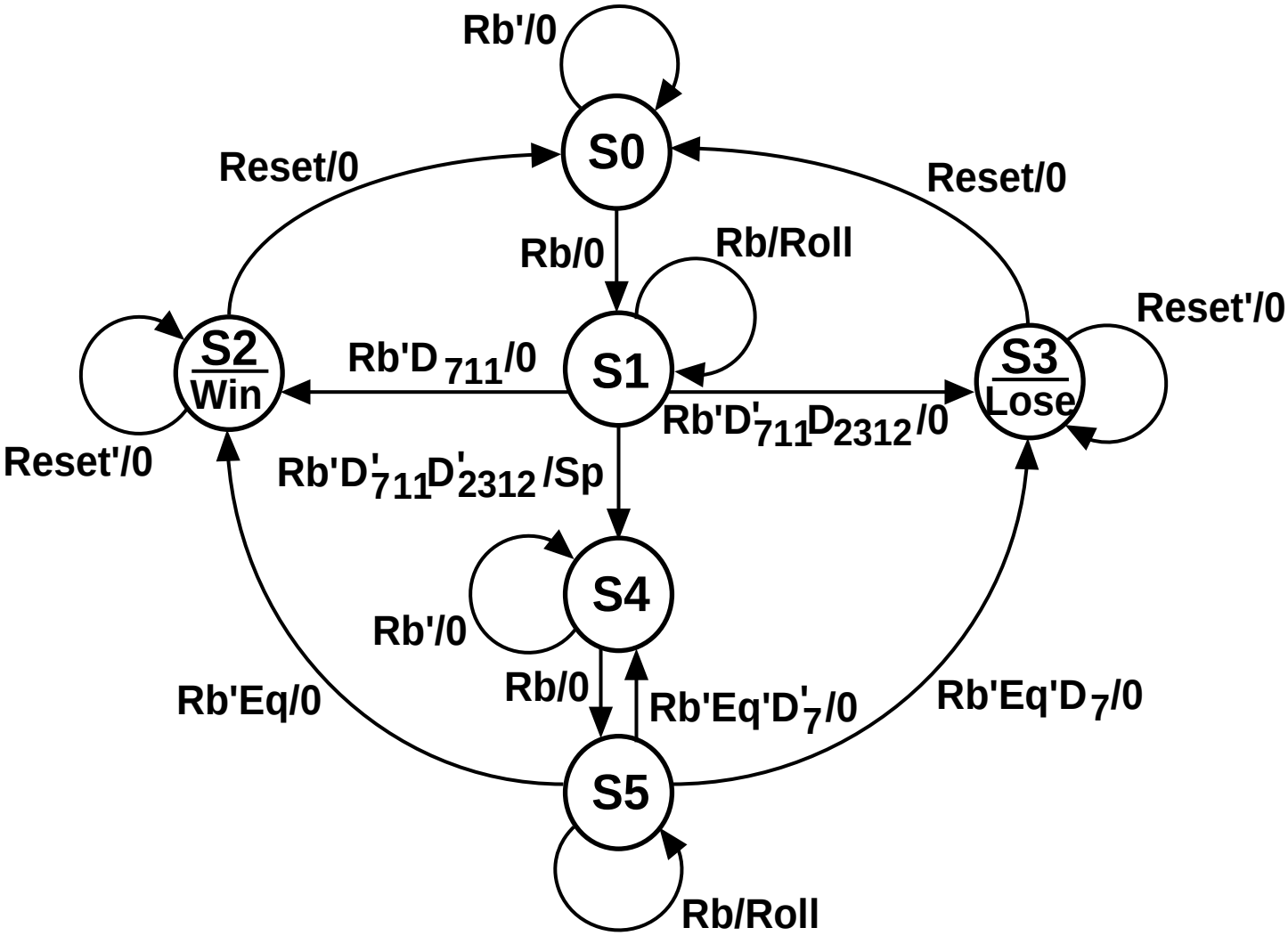


Figure 5-15(a) Behavioral Model for Dice Game

```
entity DiceGame is
      port (Rb, Reset, CLK: in bit;
            Sum: in integer range 2 to 12;
            Roll, Win, Lose: out bit);
end DiceGame;

library BITLIB;
use BITLIB.bit_pack.all;

architecture DiceBehave of DiceGame is
      signal State, Nextstate: integer range 0 to 5;
      signal Point: integer range 2 to 12;
      signal Sp: bit;
begin
      process(Rb, Reset, Sum, State)
      begin
            Sp <= '0'; Roll <= '0'; Win <= '0'; Lose <= '0';
            case State is
                  when 0 => if Rb = '1' then Nextstate <= 1;  end if;
                  when 1 =>
                        if Rb = '1' then Roll <= '1';
                              elsif Sum = 7 or Sum = 11 then Nextstate <= 2;
                              elsif Sum = 2 or Sum = 3 or Sum =12 then Nextstate <= 3;
                              else Sp <= '1'; Nextstate <= 4;
                        end if;
                  when 2 => Win <= '1';
                        if Reset = '1' then Nextstate <= 0; end if;
```

Figure 5-15(b) Behavioral Model for Dice Game

```
                when 3 => Lose <= '1';
                    if Reset = '1' then Nextstate <= 0; end if;
                when 4 => if Rb = '1' then Nextstate <= 5; end if;
                when 5 =>
                    if Rb = '1' then Roll <= '1';
                        elsif Sum = Point then Nextstate <= 2;
                        elsif Sum = 7 then Nextstate <= 3;
                        else Nextstate <= 4;
                    end if;
            end case;
        end process;

        process(CLK)
        begin
            if rising_edge(CLK) then
                State <= Nextstate;
                if Sp = '1' then Point <= Sum; end if;
            end if;
        end process;
end DiceBehave;
```

Figure 5-16 Dice Game with Test Bench

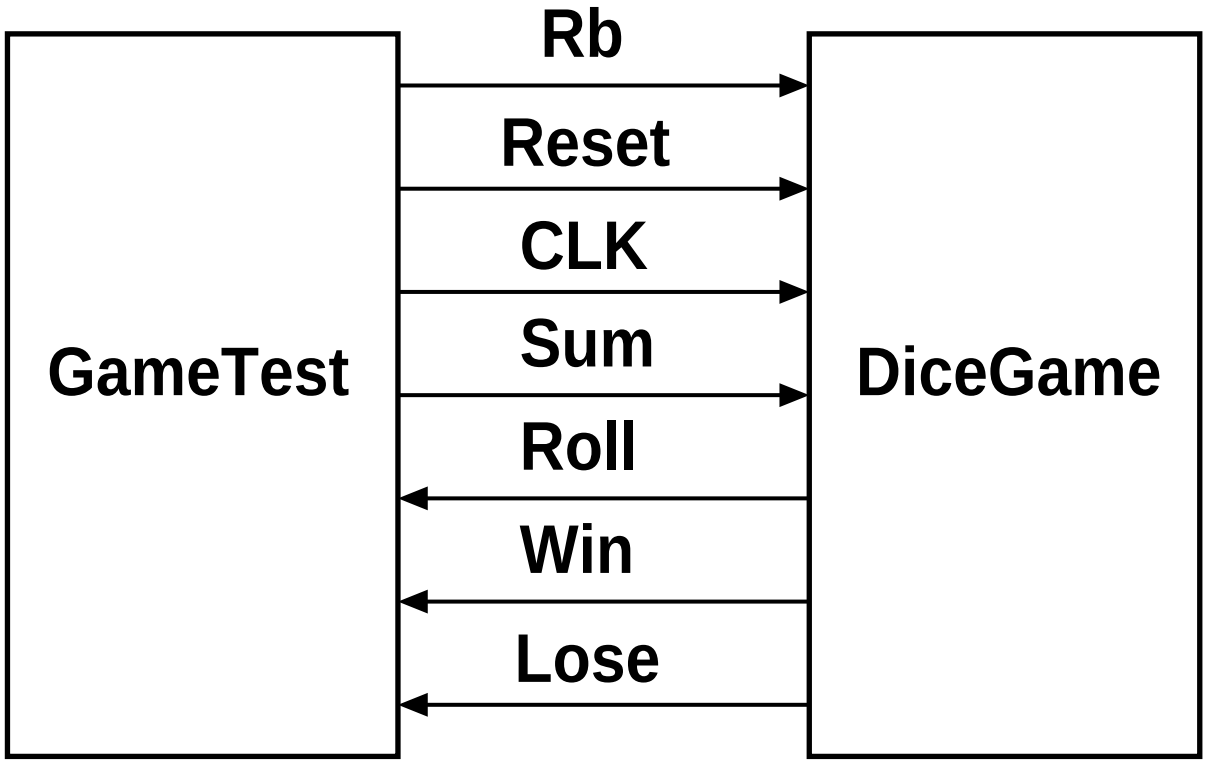


Figure 5-17 SM Chart for Dice Game Test

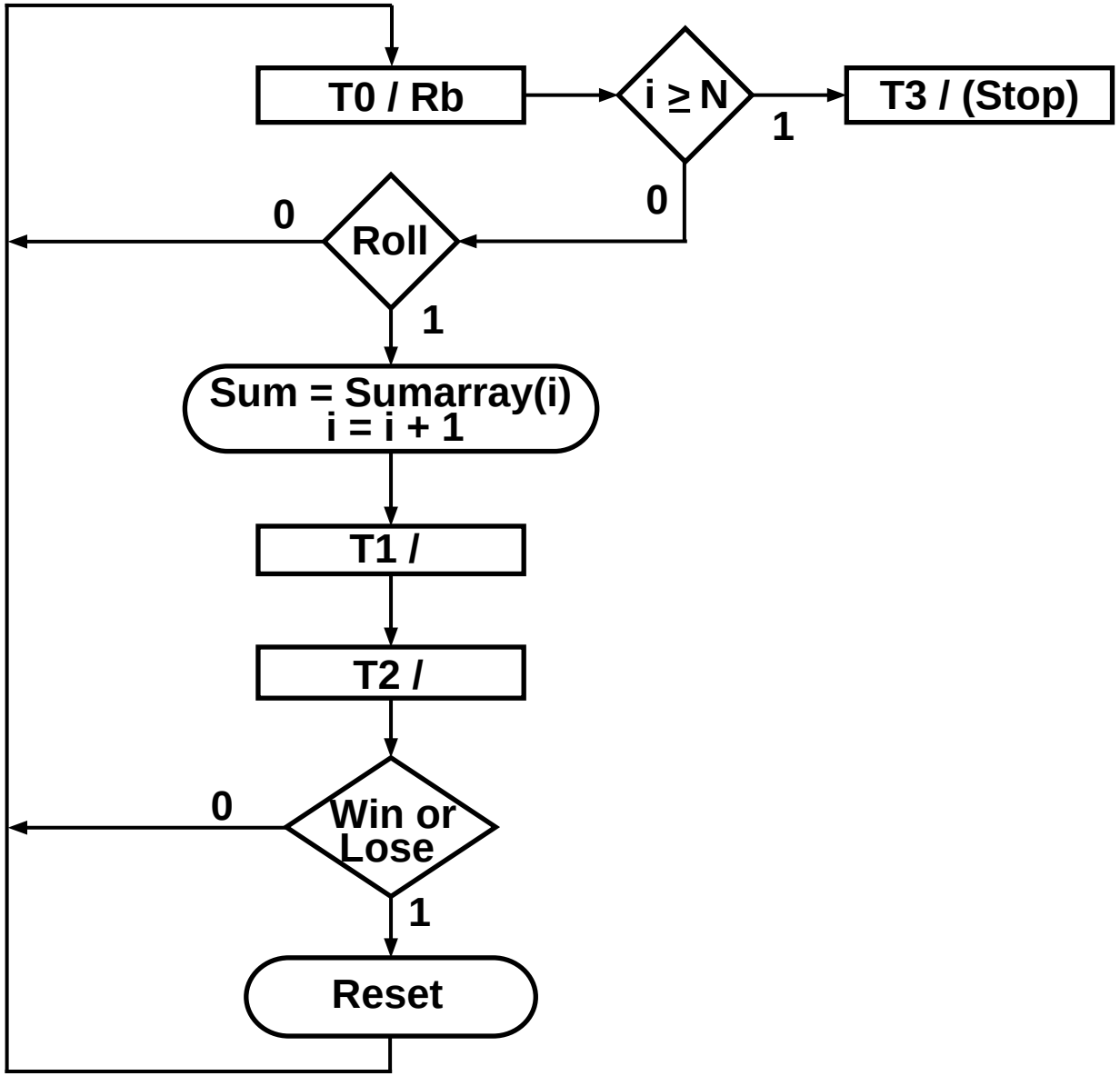


Figure 5-18(a) Dice Game Test Module

```
entity GameTest is
      port(Rb, Reset:  out bit;  Sum: out integer range 2 to 12;
            CLK: inout bit;  Roll, Win, Lose: in bit);
end GameTest;

library BITLIB;
use BITLIB.bit_pack.all;

architecture dicetest of GameTest is
      signal Tstate, Tnext: integer range 0 to 3;
      signal Trig1: bit;
      type arr is array(0 to 11) of integer;
      constant Sumarray:arr := (7,11,2,4,7,5,6,7,6,8,9,6);
      begin
            CLK <= not CLK after 20 ns;
```

Figure 5-18(b) Dice Game Test Module

```
    process(Roll, Win, Lose, Tstate)
        variable i: natural;                                    -- i is initialized to 0
        begin
        case Tstate is
            when 0 => Rb <= '1';                                -- wait for Roll
                Reset <='0';
                if i>=12 then Tnext <= 3;
                elsif Roll = '1' then
                    Sum <= Sumarray(i);
                    i:=i+1;
                    Tnext <= 1;
                end if;
            when 1 => Rb <= '0'; Tnext <= 2;
            when 2 => Tnext <= 0;
                Trig1 <= not Trig1;                             -- toggle Trig1
                if (Win or Lose) = '1' then Reset <= '1';  end if;
            when 3 => null;                                     -- Stop state
        end case;
    end process;
    process(CLK)
        begin
        if CLK = '1' then
            Tstate <= Tnext;
        end if;
    end  process;
end dicetest;
```

Figure 5-19 Tester for Dice Game

```
entity tester is
end tester;
architecture test of tester is
component GameTest
      port(Rb, Reset:  out bit;
            Sum: out integer range 2 to 12;  CLK: inout bit;  Roll, Win, Lose: in bit);
end component;
component DiceGame
      port (Rb, Reset, CLK: in bit;  Sum: in integer range 2 to 12 ;  Roll, Win, Lose: out bit);
end component;
signal rb1, reset1, clk1, roll1, win1, lose1: bit;  signal sum1: integer range 2 to 12;
begin
      Dice: Dicegame port map(rb1,reset1,clk1,sum1,roll1,win1,lose1);
      Dicetest: GameTest port map(rb1,reset1,sum1,clk1,roll1,win1,lose1);
end test;
```

Figure 5-20 Simulation and Command File for Dice Game Tester

```
list /dicetest/ trig1  -NOTrigger sum1 win1 lose1 /dice/point
run 2000
```

ns	delta	trig1	sum1	win1	lose1	point
0	+0	0	2	0	0	2
100	+3	0	7	1	0	2
260	+3	0	11	1	0	2
420	+3	0	2	0	1	2
580	+2	1	4	0	0	4
740	+3	1	7	0	1	4
900	+2	0	5	0	0	5
1060	+2	1	6	0	0	5
1220	+3	1	7	0	1	5
1380	+2	0	6	0	0	6
1540	+2	1	8	0	0	6
1700	+2	0	9	0	0	6
1860	+3	0	6	1	0	6

Table 5-1 PLA Table for Multiplier Control

	A	B	St	M	K	A^+	B^+	Load	Sh	Ad	Done
S0	0	0	0	–	–	0	0	0	0	0	0
	0	0	1	–	–	0	1	1	0	0	0
S1	0	1	–	0	0	0	1	0	1	0	0
	0	1	–	0	1	1	1	0	1	0	0
	0	1	–	1	–	1	0	0	0	1	0
S2	1	0	–	–	0	0	1	0	1	0	0
	1	0	–	–	1	1	1	0	1	0	0
S3	1	1	–	–	–	0	0	0	0	0	1

A^+ = A'BM'K + A'BM + AB'K = A'B(M + K) + AB'K

B^+ = A'B'St + A'BM'(K'+K) + AB'(K'+K) = A'B'St + A'BM' + AB'

Sh = A'BM'(K'+K) + AB'(K'+K) = A'BM' + AB'

Load = A'B'St *Ad* = A'B M *Done* = A B

Figure 5-21 PLA Realization of Dice Game Controller

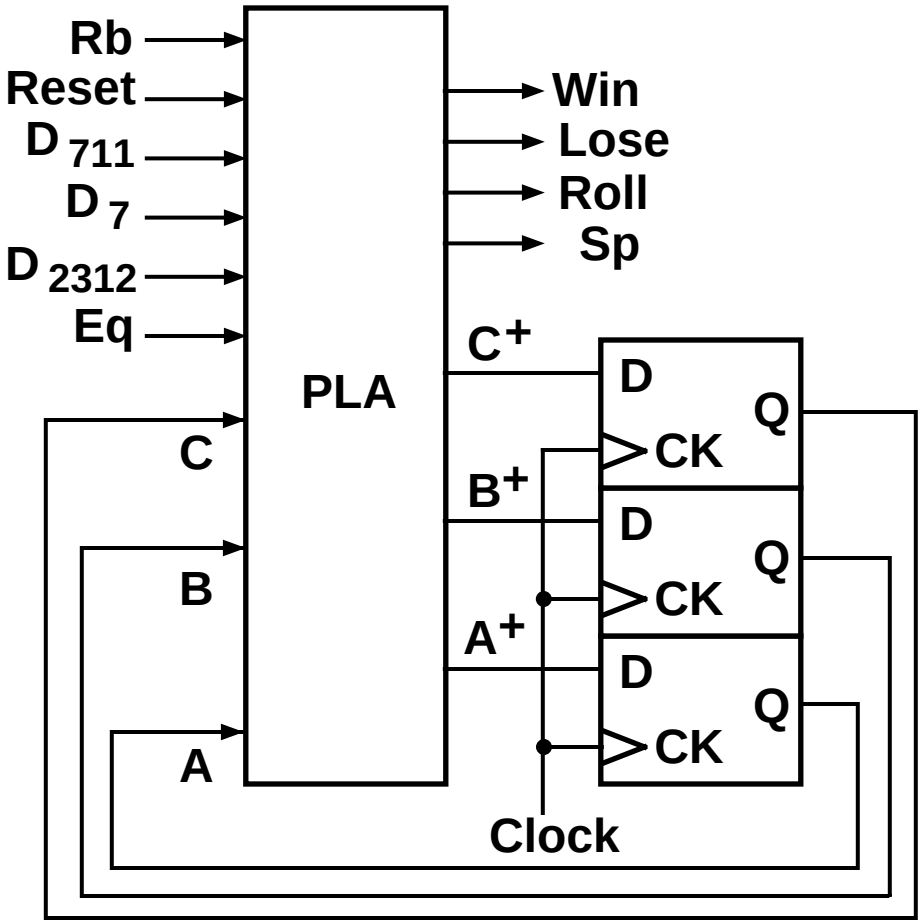
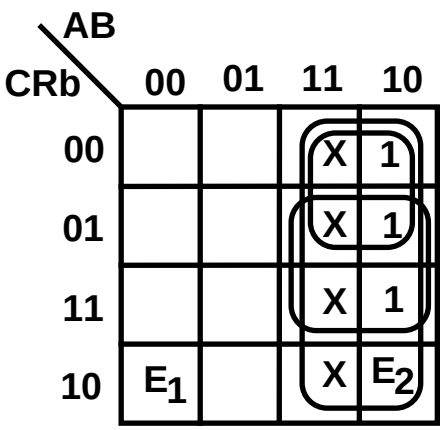
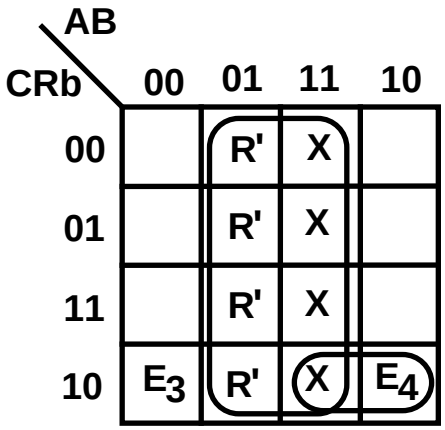


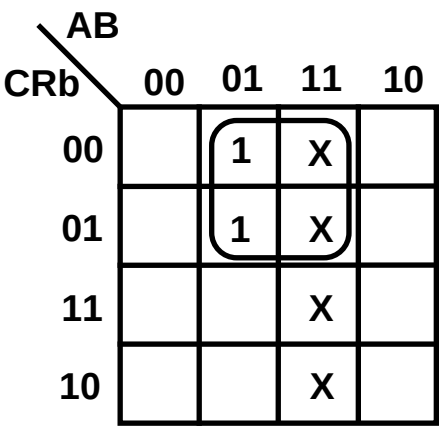
Table 5-2 PLA Table for Dice Game

	ABC	Rb	Reset	D7	D711	D2312	Eq	A+	B+	C+	Win	Lose	Roll	Sp
1	000	0	–	–	–	–	–	0	0	0	0	0	0	0
2	000	1	–	–	–	–	–	0	0	1	0	0	0	0
3	001	1	–	–	–	–	–	0	0	1	0	0	1	0
4	001	0	–	–	0	0	–	1	0	0	0	0	0	1
5	001	0	–	–	0	1	–	0	1	1	0	0	0	0
6	001	0	–	–	1	–	–	0	1	0	0	0	0	0
7	010	–	0	–	–	–	–	0	1	0	1	0	0	0
8	010	–	1	–	–	–	–	0	0	0	1	0	0	0
9	011	–	1	–	–	–	–	0	0	0	0	1	0	0
10	011	–	0	–	–	–	–	0	1	1	0	1	0	0
11	100	0	–	–	–	–	–	1	0	0	0	0	0	0
12	100	1	–	–	–	–	–	1	0	1	0	0	0	0
13	101	0	–	0	–	–	0	1	0	0	0	0	0	0
14	101	0	–	1	–	–	0	0	1	1	0	0	0	0
15	101	0	–	–	–	–	1	0	1	0	0	0	0	0
16	101	1	–	–	–	–	–	1	0	1	0	0	1	0
17	110	–	–	–	–	–	–	–	–	–	–	–	–	–
18	111	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 5-22 Maps Derived from Table 5-2







$E_1 = D'_{711} D'_{2312}$

$E_2 = D'_7 Eq'$

R = Reset

$E_3 = D_{711} + D'_{711} D'_{2312} = D_{711} + D'_{2312}$

$E_4 = Eq + D_7 Eq' = Eq + D_7$

Figure 5-23 Data Flow Model for Dice Game

```
library BITLIB;
use BITLIB.bit_pack.all;

architecture Dice_Eq of DiceGame is
      signal Sp,Eq,D7,D711,D2312: bit:='0';  signal DA,DB,DC,A,B,C :bit:='0';
      signal Point: integer range 2 to 12;
begin
process(Clk)
begin
      if rising_edge(Clk) then
            A <= DA;  B <= DB;   C <= DC;
            if Sp = '1' then Point <= Sum; end if;
      end if;
end process;
Win <= B and not C;
Lose <= B and C;
Roll <= not B and C and Rb;
Sp <= not A and not B and C and not Rb and not D711 and not D2312;
D7 <= '1' when Sum = 7 else '0';
D711 <= '1' when (Sum = 11) or (Sum = 7) else '0';
D2312 <= '1' when (Sum = 2) or (Sum = 3) or (Sum = 12) else '0';
Eq <= '1' when Point=Sum else '0';
DA <= (not A and not B and C and not Rb and not D711 and not D2312) or
      (A and not C) or (A and Rb) or (A and not D7 and not Eq);
DB <= ( (not A and not B and C and not Rb) and (D711 or D2312) )
      or (B and not Reset) or ( (A and C and not Rb) and (Eq or D7) );
DC <= (not B and Rb) or (not A and not B and C and not D711 and D2312) or
      (B and C and not Reset) or (A and C and D7 and not Eq);
end Dice_Eq;
```

Figure 5-24 Counter for Dice Game

```
entity Counter is
      port(Clk, Roll: in bit;
            Sum: out integer range 2 to 12);
end Counter;

architecture Count of Counter is
signal Cnt1,Cnt2: integer range 1 to 6 := 1;
begin
      process (Clk)
      begin
            if Clk='1' then
                  if Roll='1' then
                        if Cnt1=6 then Cnt1 <= 1; else Cnt1 <= Cnt1+1; end if;
                        if Cnt1=6 then
                              if Cnt2=6 then Cnt2 <= 1; else Cnt2 <= Cnt2+1; end if;
                        end if;
                  end if;
            end if;
      end process;
  Sum <= Cnt1 + Cnt2;
end Count;
```

Figure 5-25 Complete Dice Game

```
entity Game is
      port (Rb, Reset, Clk: in bit;
            Win, Lose: out bit);
end Game;

architecture Play1 of Game is
      component Counter
            port(Clk, Roll: in bit;
                  Sum: out integer range 2 to 12);
      end component;
      component DiceGame
            port (Rb, Reset, CLK: in bit;
                  Sum: in integer range 2 to 12;
                  Roll, Win, Lose: out bit);
      end component;
            signal roll1: bit;
            signal sum1: integer range 2 to 12;
begin
      Dice: Dicegame port map(Rb,Reset,Clk,sum1,roll1,Win,Lose);
      Count:  Counter port map(Clk,roll1,sum1);
end Play1;
```

Figure 5-26 Control Network Using an input Mux to Select the Next State

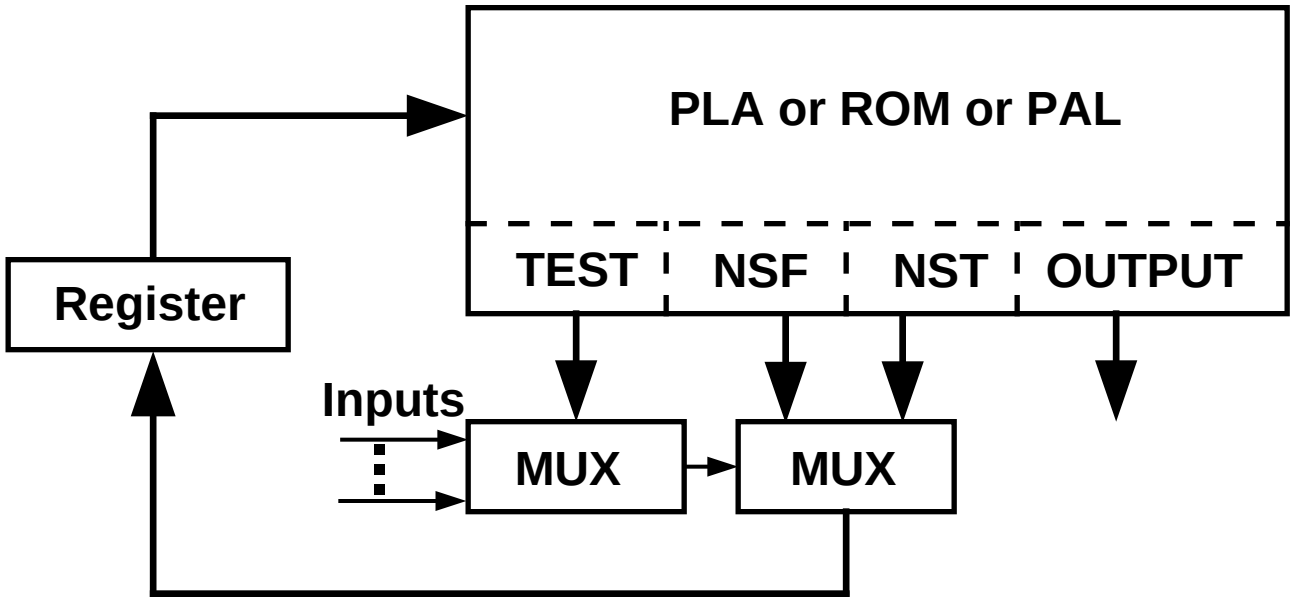


Figure 5-27(a) SM Chart with Moore Outputs and One Test per State

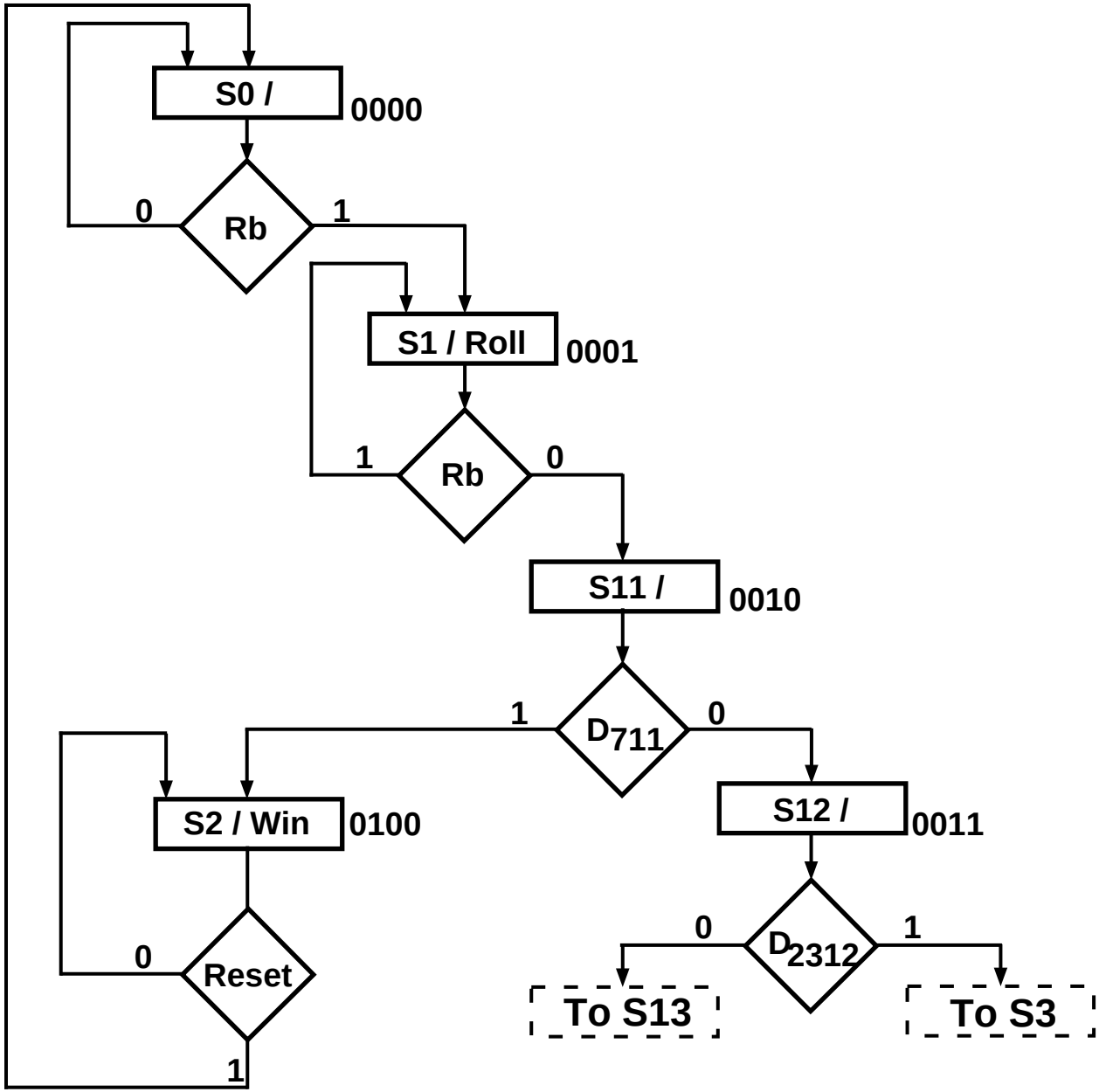


Figure 5-27(b) Chart with Moore Outputs and One Test per State

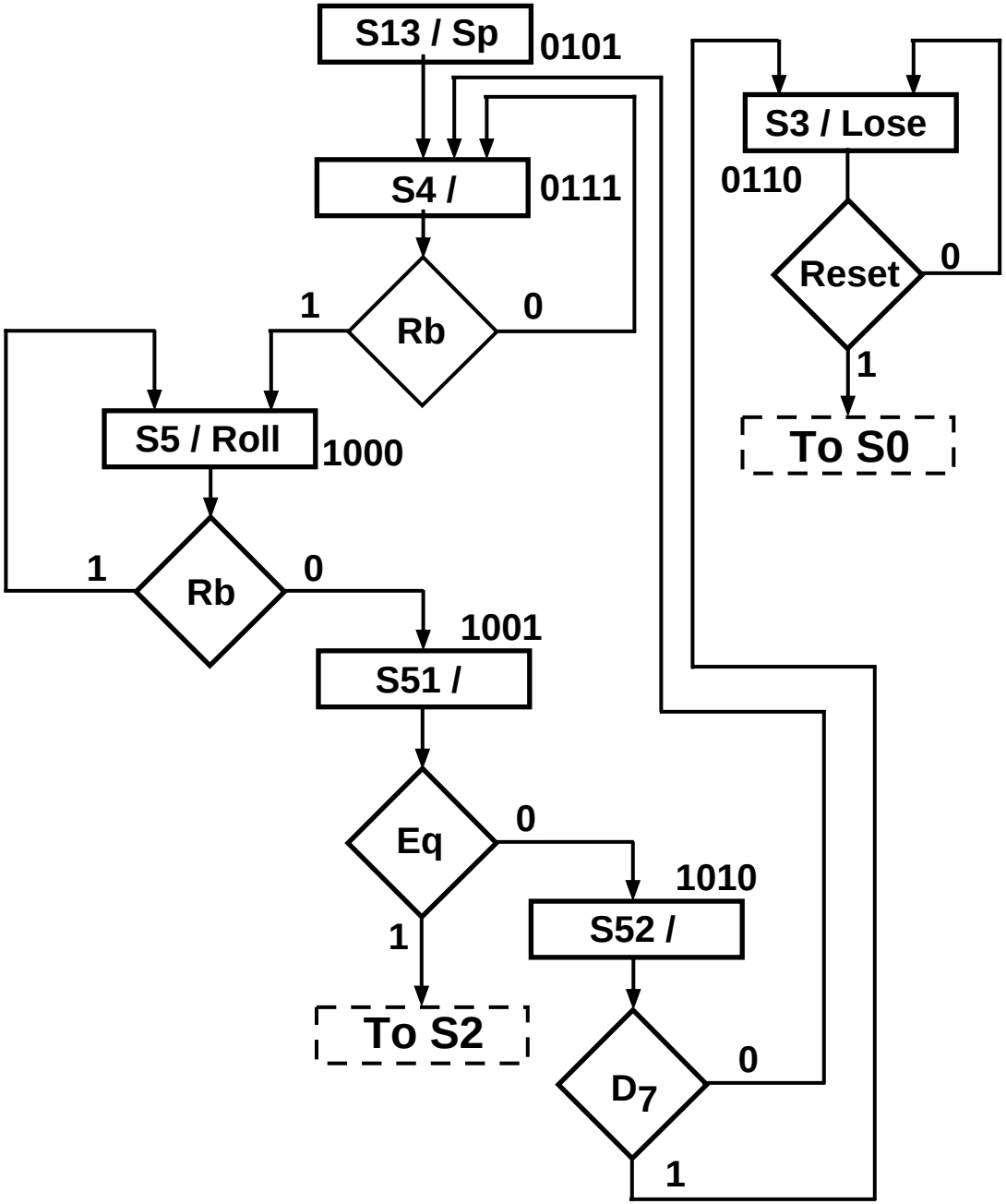


Figure 5-28 MUX for SM Chart of Figure 5-27

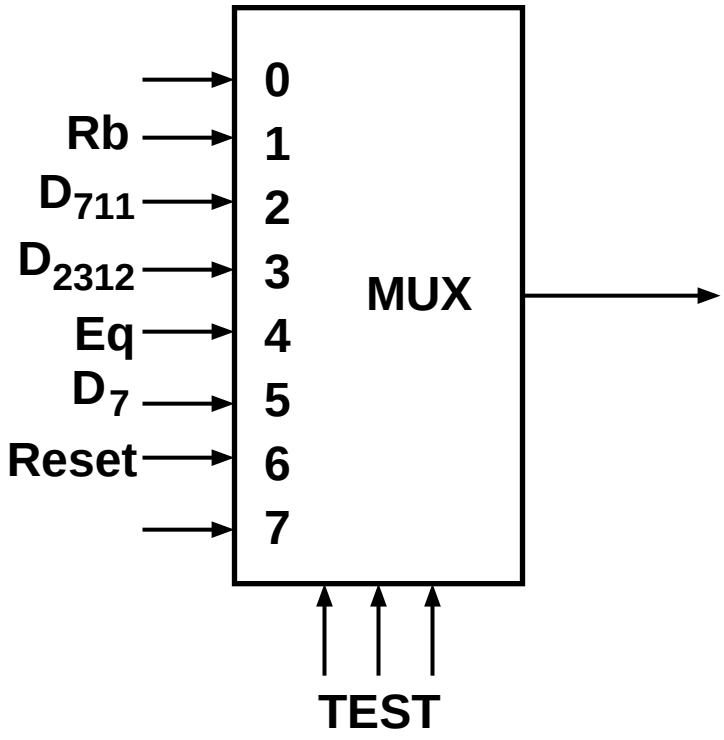


Table 5-3 PLA/ ROM Table for Figure 5-27

State	ABCD	TEST	NSF	NST	ROLL	Sp	Win	Lose
S0	0000	001	0000	0001	0	0	0	0
S1	0001	001	0010	0001	1	0	0	0
S11	0010	010	0011	0100	0	0	0	0
S12	0011	011	0101	0110	0	0	0	0
S2	0100	110	0100	0000	0	0	1	0
S13	0101	xxx	0111	0111	0	1	0	0
S3	0110	110	0110	0000	0	0	0	1
S4	0111	001	0111	1000	0	0	0	0
S5	1000	001	1001	1000	1	0	0	0
S51	1001	100	1010	0100	0	0	0	0
S52	1010	101	0111	0110	0	0	0	0

Figure 5-29 Control Network Using a Counter for the State Register

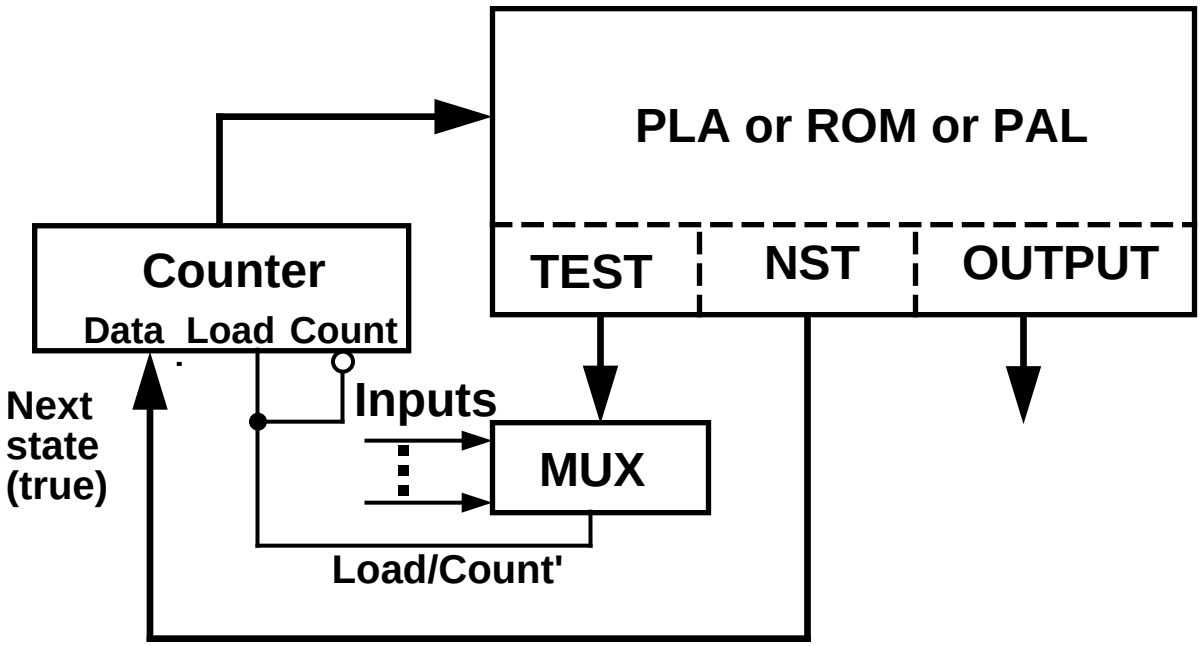


Figure 5-30(a) SM chart with Serial State Assignment and Added X-states

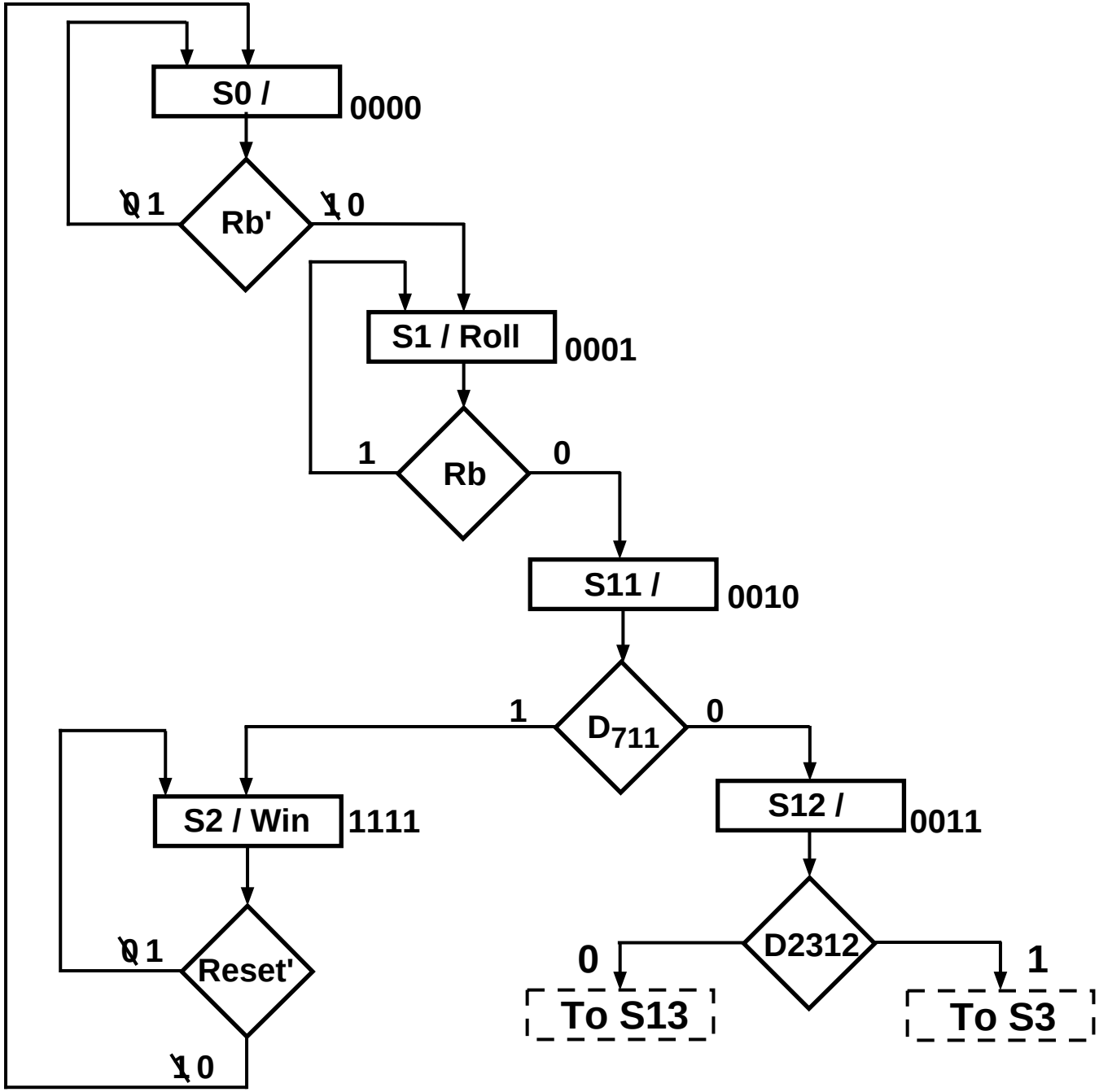


Figure 5-30(b) SM Chart with Serial State Assignment and Added X-state

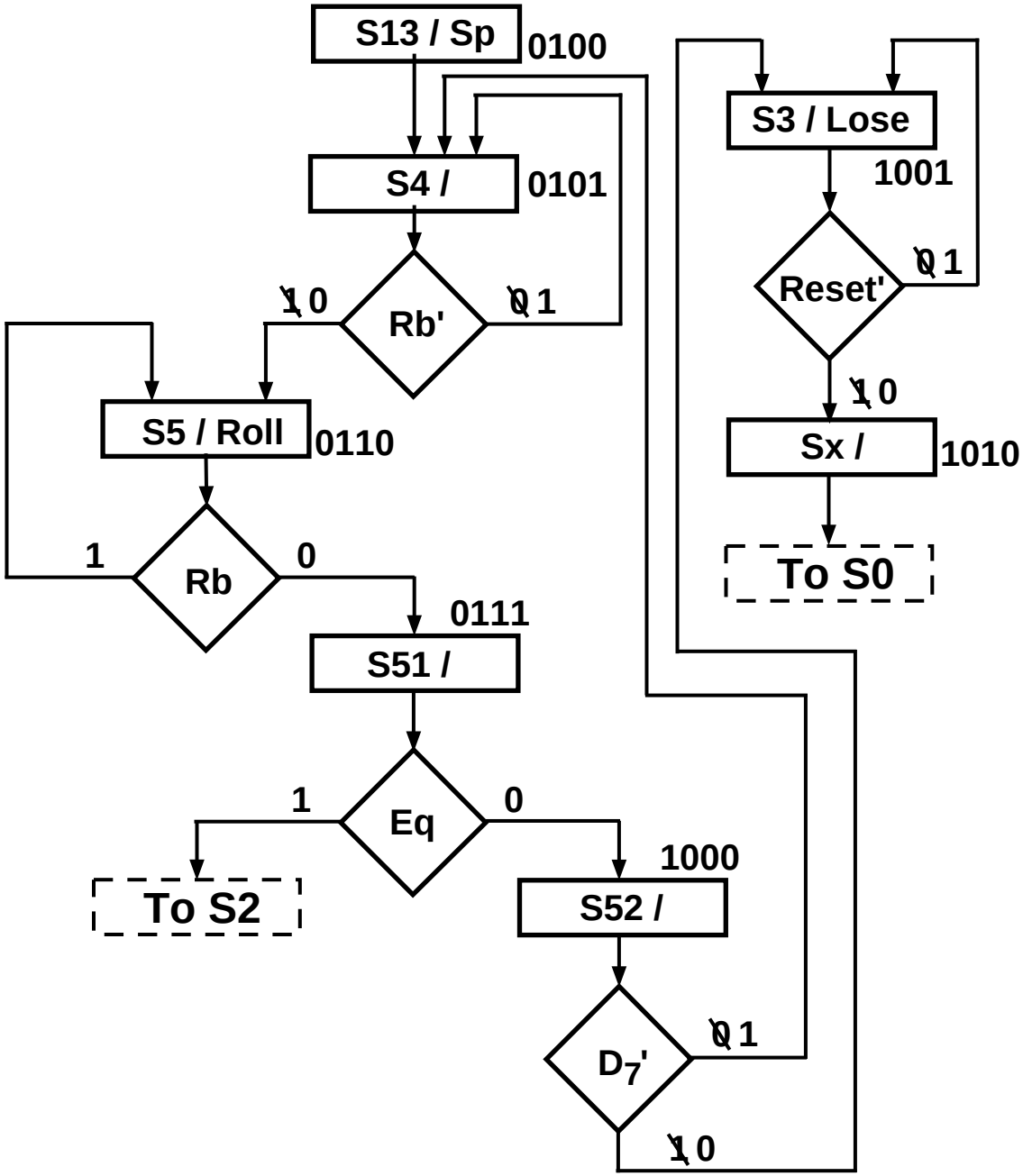


Figure 5-31 MUX for SM chart of Figure 5-30

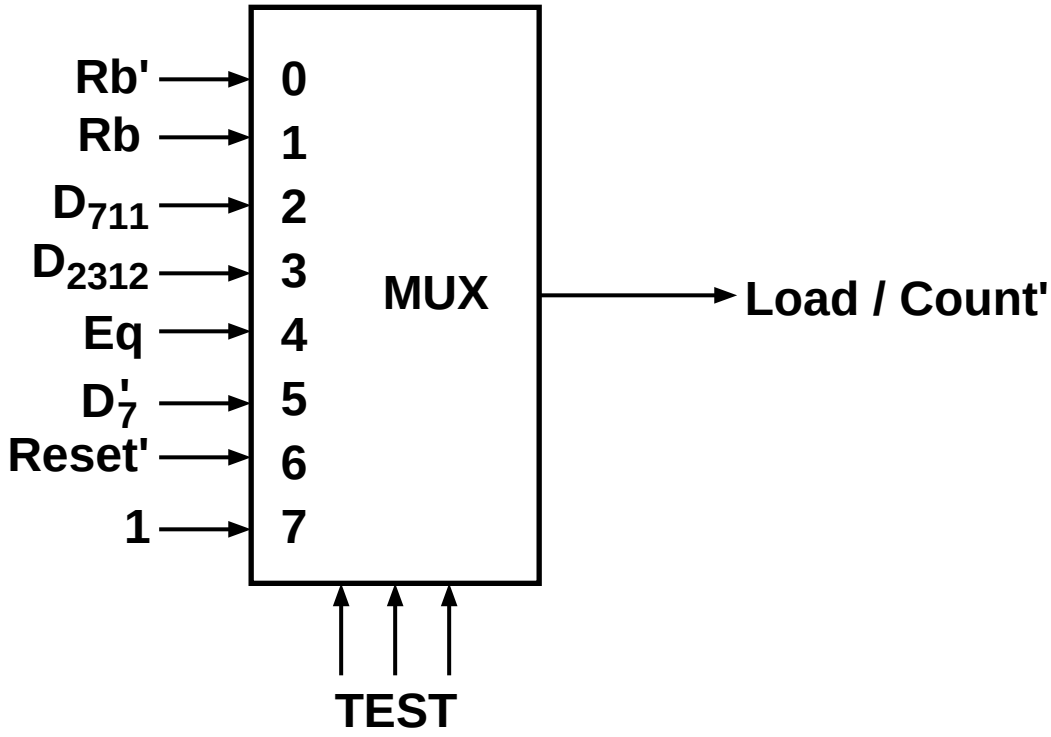


Table 5-4 PLA Table for Figure 5-31

State	ABCD	TEST	NST	ROLL	Sp	Win	Lose
S0	0000	000	0000	0	0	0	0
S1	0001	001	0001	1	0	0	0
S11	0010	010	1111	0	0	0	0
S12	0011	011	1001	0	0	0	0
S13	0100	111	0101	0	1	0	0
S4	0101	000	0101	0	0	0	0
S5	0110	001	0110	1	0	0	0
S51	0111	100	1111	0	0	0	0
S52	1000	101	0101	0	0	0	0
S3	1001	110	1001	0	0	0	1
Sx	1010	111	0000	0	0	0	0
S2	1111	110	1111	0	0	1	0

Test(2) = B C'D' + B C D + A
Test(1) = B'C + B C'D' + A D
Test(0) = A'B'D + B D' + A D'
NST(3) = A'B'C + C D + A D
NST(2) = A'C D' + B + A C'D'
NST(1) = A'C D' + B C
NST(0) = D + A'B'C + B C' + A C'
Roll = A'B'C'D + B C D'
SP = B C'D'
Win = A B
Lose = A B'D

Figure 5-32 SM Charts for Serially Linked State Machine

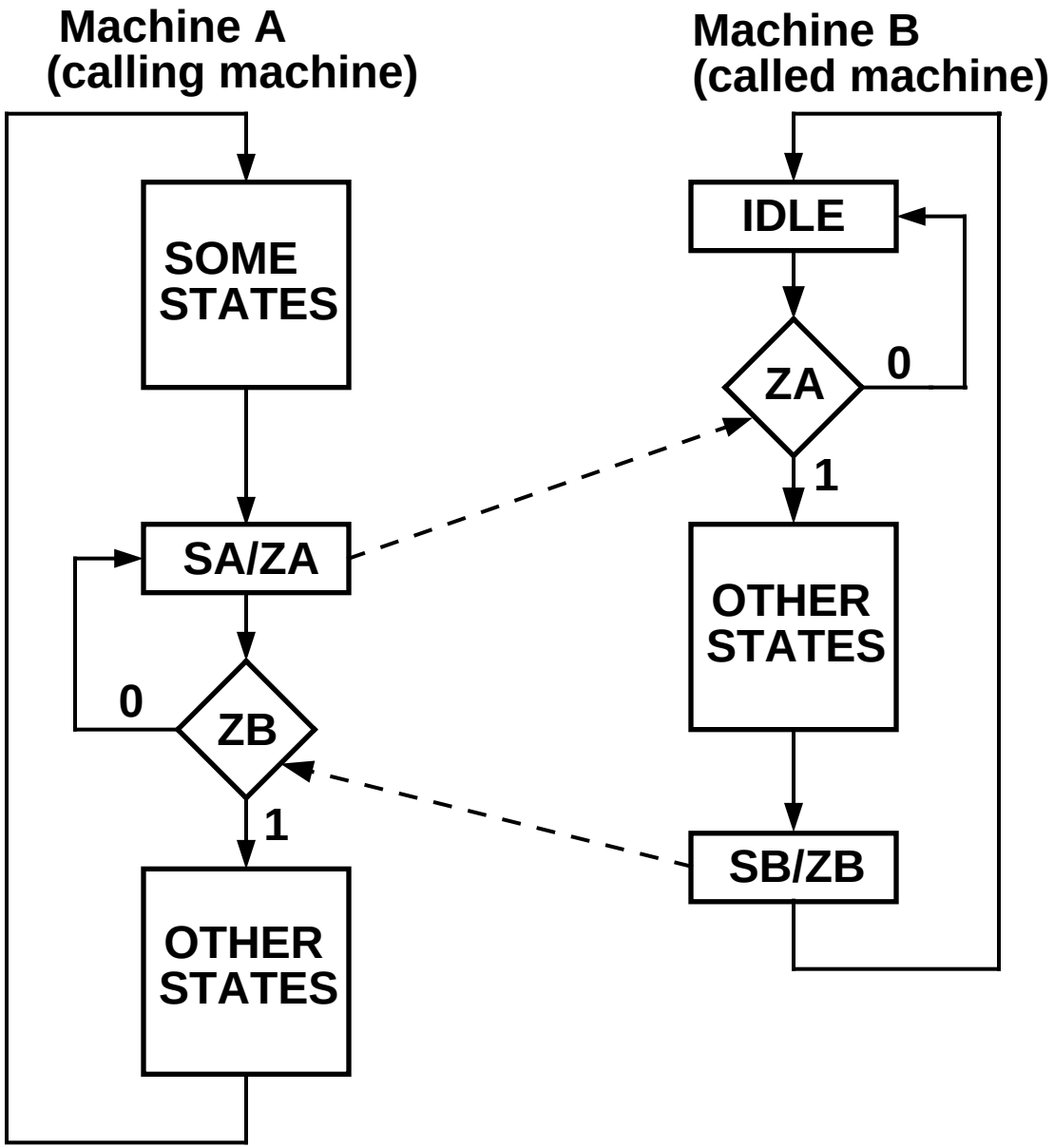
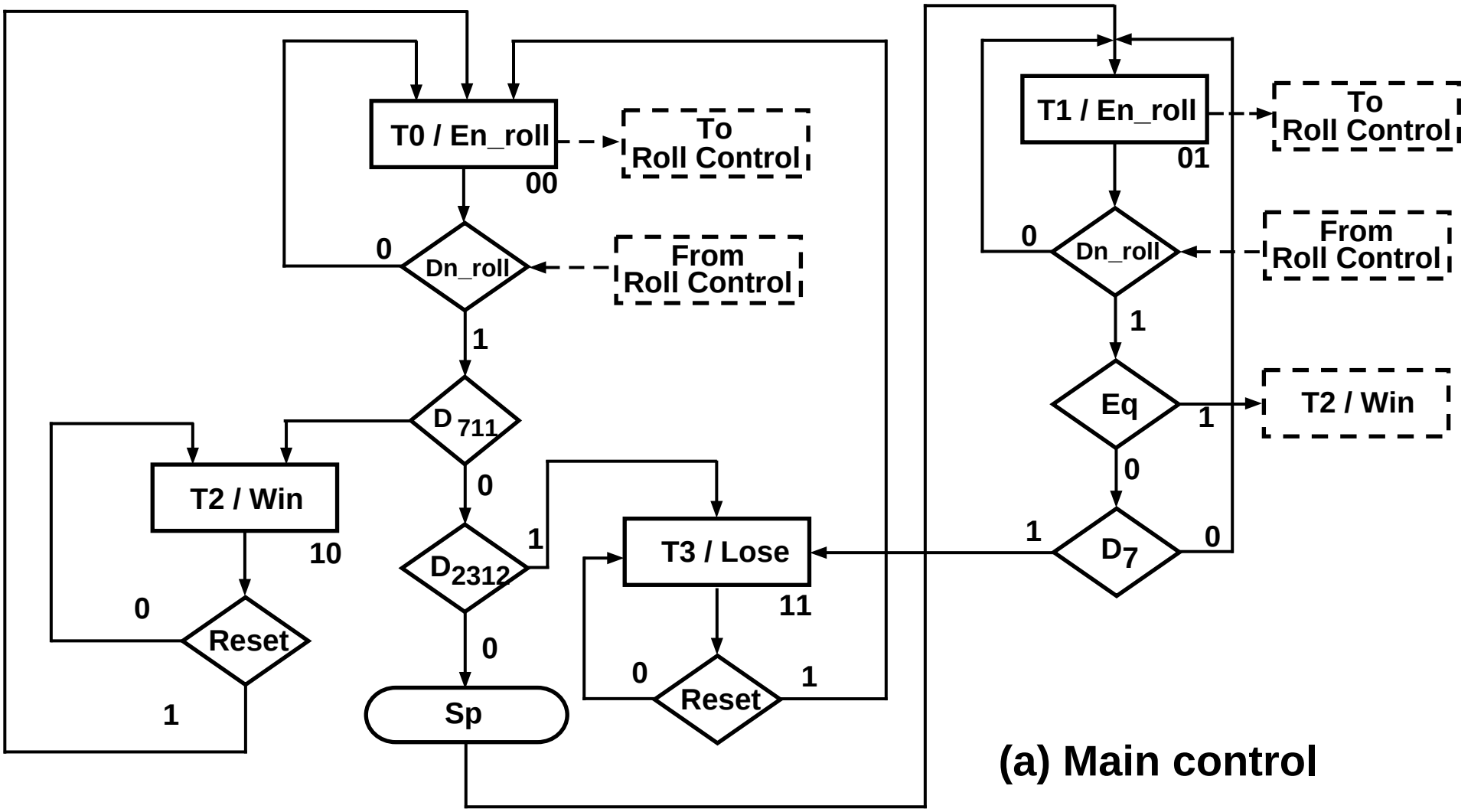
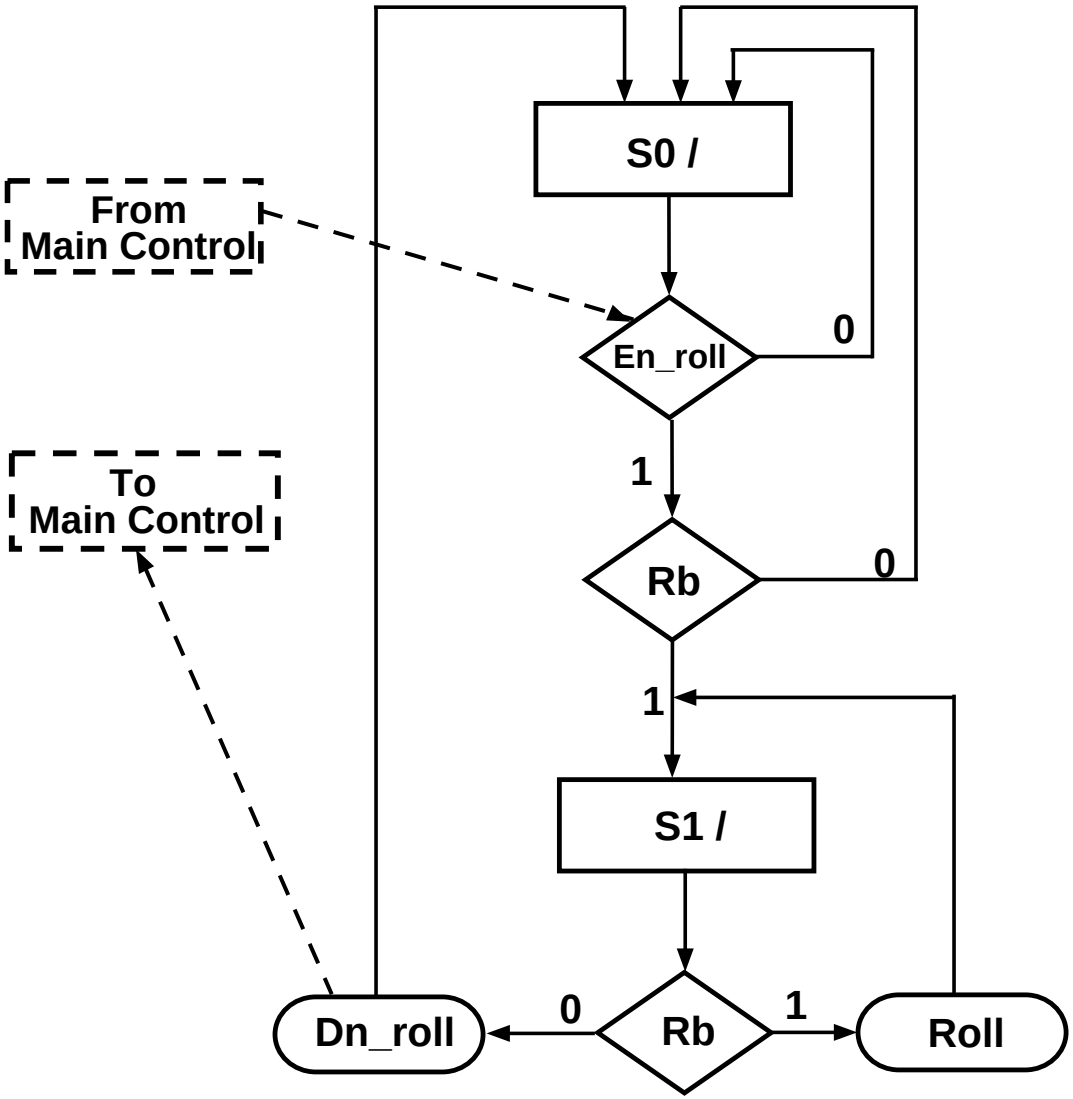


Figure 5-33a Linked SM Charts for Dice Game



(a) Main control

Figure 5-33b Linked SM Charts for Dice Game



(b) Roll Control