2. Combinational and Sequential Circuits Design

2.1. Combinational Logic

A **combinational system (device)** is a digital system in which the value of the output at any instant depends only on the value of the input at that same instant (and not on previous values).

High-Level Specification of Combinational Systems
The specification consists of the following three components:
The set of values for the input, called the **input set**;
The set of values for the output, called the **output set**;

The description of the **input-output function:**

- a table (discrete function)
- an arithmetic expression (elements of the input and output sets are functions)
- a conditional expression (if the function can be partitioned into subfanctions)
- a logical expression (Boolen formula, preposition)
- a composition of simpler functions

Example
Informal specification: A radix-4 digit comparator module compares two radix-4 digits and produces one output with values G (grater), E(equal), and S(smaller).

The high-level specification is:

Inputs: $x, y \in \{0, 1, 2, 3\}$

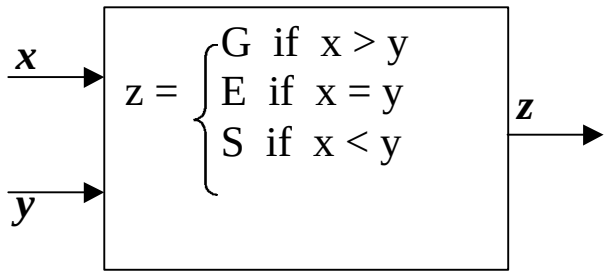
Outputs: $z \in \{G, E, S\}$

Function (a conditional expression):

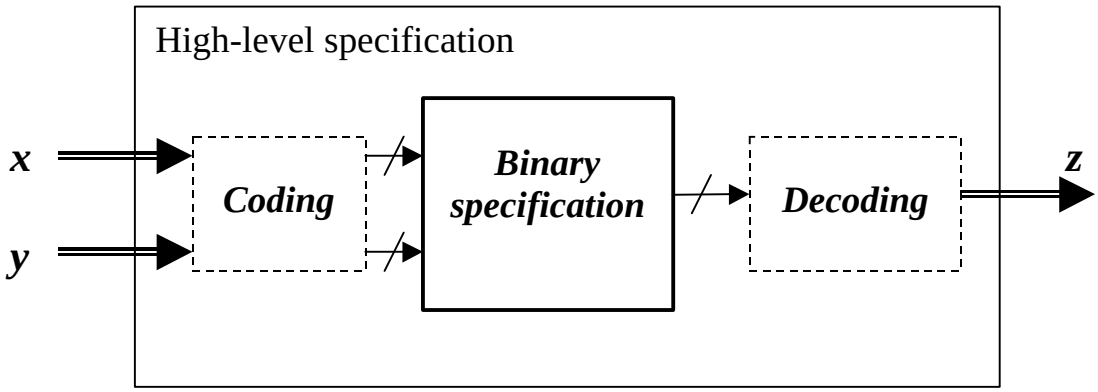
$$z = \begin{cases} G & \text{if } x > y \\ E & \text{if } x = y \\ S & \text{if } x < y \end{cases}$$

The tabular description of this function is

x \ ***y***	0	1	2	3
0	E	S	S	S
1	G	E	S	S
2	G	G	E	S
3	G	G	G	E



To obtain a binary description we have to code the input and output values on bit vectors.



The three-values output requires at least two binary variables. We use three binary variables. In this case, there are possible 8*7*6=336 code systems. We choose one of them

At the logic level we must work with both logic expression and gate networks to find the best implementation of a function, keeping in mind the relationships:

- combinational logic expressions are the specification;
- logic gate networks are the implementation;
- area (number of gates), delay, and power are the cost (restrictions).

z	z_2	z_1	z_0
G	1	0	0
E	0	1	0
S	0	0	1

For input, we take the most used binary code in which the radix-4 digit is represented by the correspondingradix-2 bit-vector

$(x = 2*x_1+x_0; \quad y = 2*y_1+y_0)$

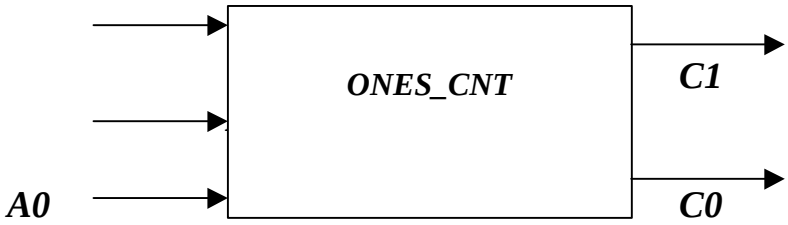
x (y)	x_1 (y_1)	x_2 (y_2)
1	*0*	*0*
2	*0*	*1*
3	*1*	*0*
4	*1*	*1*

Now, we can obtain the switching functions described by the following table (*binary specification of combinational system)*:

$x_1\ x_0$	$y_1\ y_0$ 00	01	10	11
00	010	001	001	001
01	100	010	001	001
10	100	100	010	001
11	100	100	100	010

$z_2\ z_1\ z_0$

Example ONE'S COUNT




```
entity ONES_CNT is
port (A: in BIT_VECTOR(2 downto 0);
      C: out BIT_VECTOR(1 downto 0));
end ONES_CNT;
--Truth table:
-------------------------------------
-- A2   A1  A0       C1   C0
-------------------------------------
-- 0    0   0        0    0
-- 0    0   1        0    1
-- 0    1   0        0    1
-- 0    1   1        1    0
-- 1    0   0        0    1
-- 1    0   1        1    0
-- 1    1   0        1    0
-- 1    1   1        1    1
-------------------------------------
end ONES_CNT;
architecture PURE_BEHAVIOR of ONES_CNT is
begin
      process(A)
            variable NUM: INTEGER range 0 to 3;
      begin
            NUM := 0;
            for I in 0 to 2 loop
                  if A(I) = '1' then
                        NUM := NUM + 1;
                  end if;
            end loop;
            case NUM is
                  when 0 => C <= '00';
                  when 1 => C <= '01';
                  when 2 => C <= '10';
                  when 3 => C <= '11';
            end case;
      end process;
end PURE_BEHAVIOR;
```

		A1 A0			
		00	01	11	10
A2	0	0	0	1	0
	1	0	1	1	1

C1 = A1 A0 Ú A2 A0 Ú A2A1

		A1 A0			
		00	01	11	10
A2	0	0	1	0	1
	1	1	0	1	0

C0 = A2A1A0 Ú A2A1A0 Ú A2A1A0 Ú A2A1A0

C1 is the MAJORITY FUNCTION

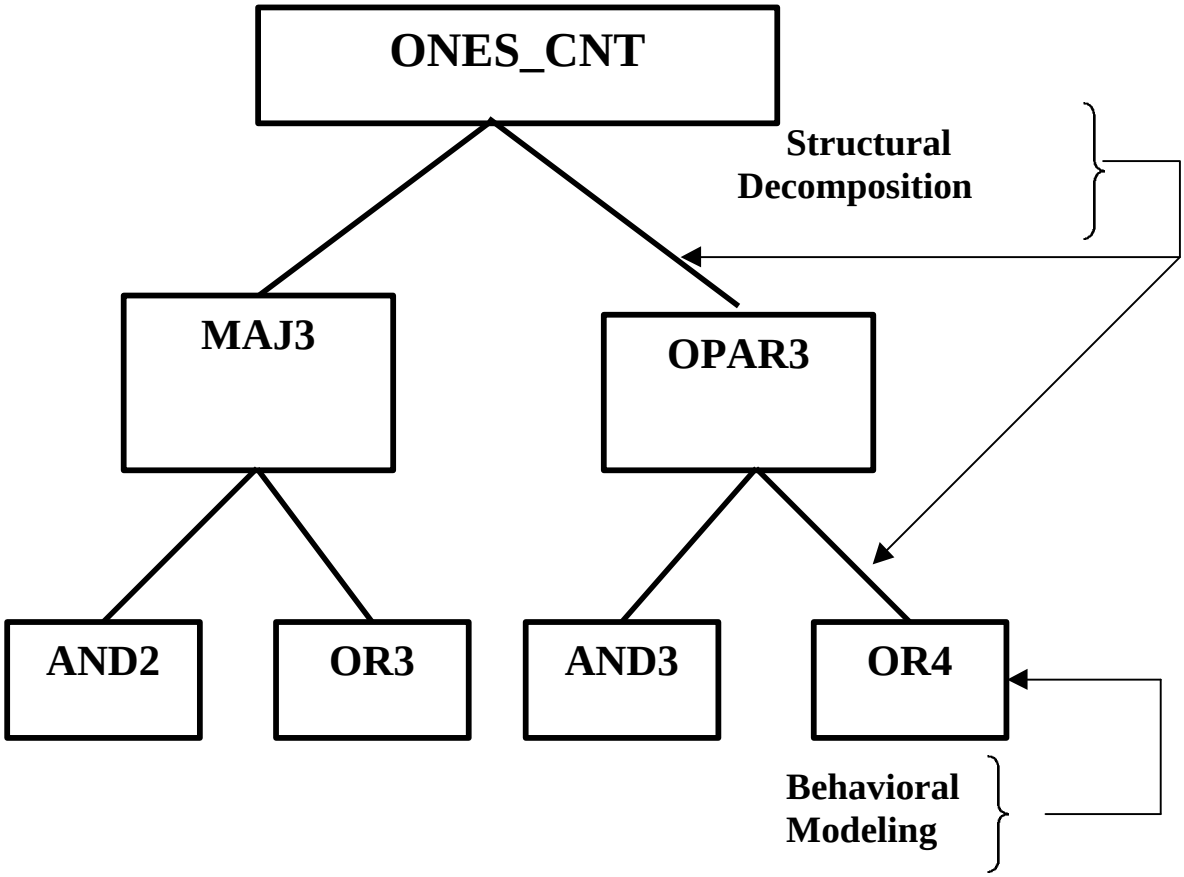
C0 is ODD-PARITY FUNCTION

```
--Macro-based architectural body:
architecture MACRO of ONES_CNT is
begin
        C(1) <= MAJ3(A);
        C(0) <= OPAR3(A);
end MACRO;
```

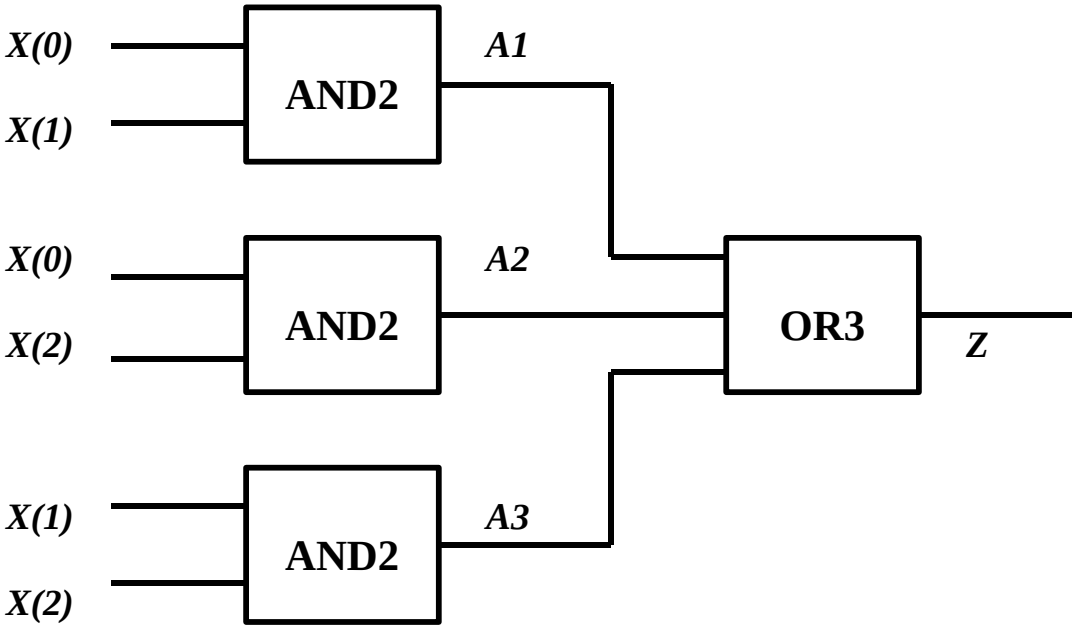
This architectural body implies the existence of MAJ and OPAR gates at the hardware level. In terms of a VHDL description, it requires that the functions MAJ3 and OPAR3 must have been declared and defined previously.

```
architecture DATA_FLOW of ONES_CNT is
begin
        C(1) <= (A(1) and A(0)) or (A(2) and A(0)) or (A(2)
                and A(1));
        C(0) <= (A(2) and not A(1) and not A(0)) or
                (not A(2) and not A(1) and A(0)) or
                (A(2) and A(1) and A(0)) or
                (not A(2) and A(1) and not A(0))
end DATA_FLOW;
```

Structural Design Hierarchy for the Ones Counter (ONES_CNT)



Majority Function Gate Structure



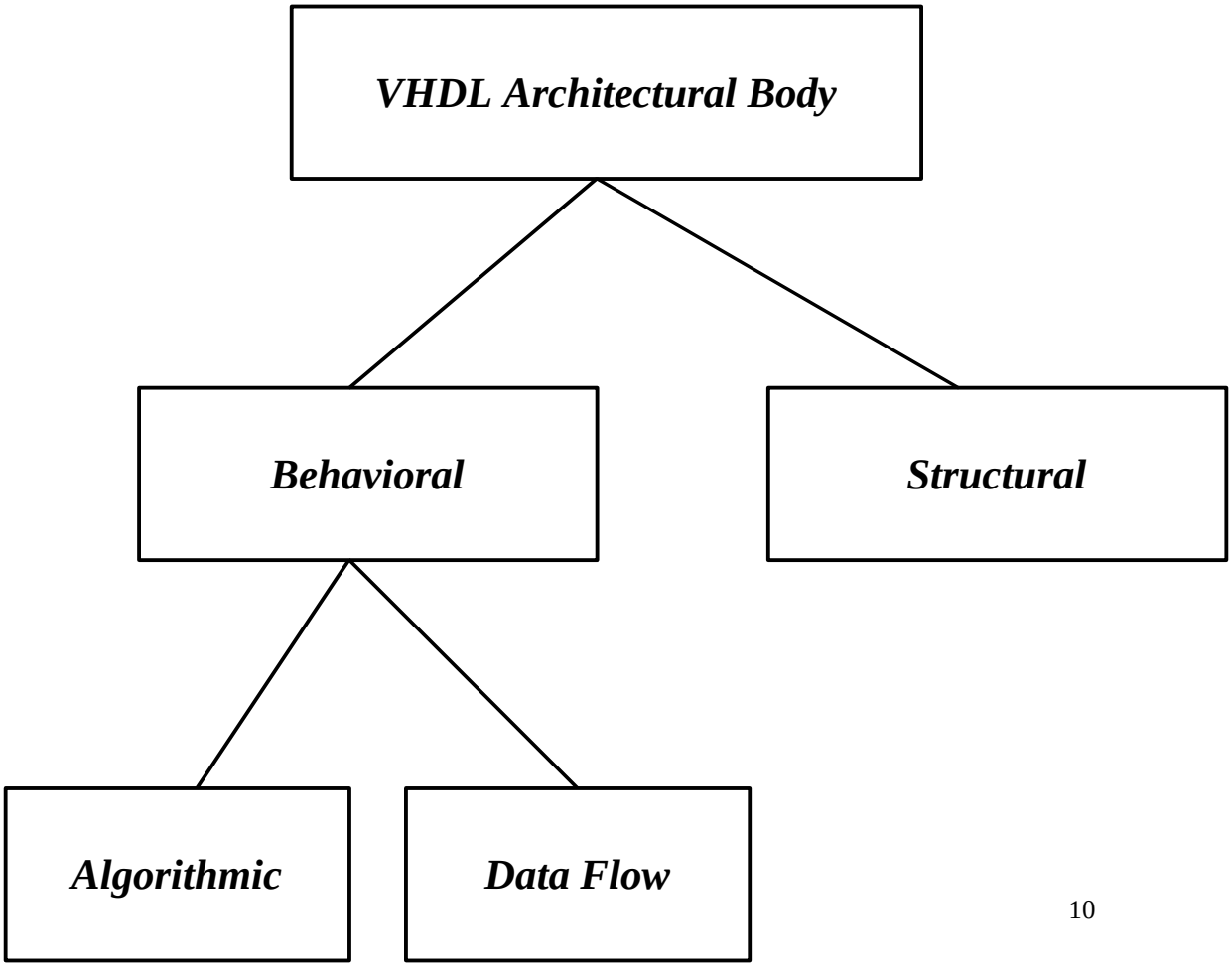

```
entity AND2 is
        port (I1, I2:in BIT; O out BIT);
end AND2

architecture BEHAVIOR of AND2 is
begin
        O <= I1 and I2;
end BEHAVIOR;
```

```
entity OR3 is
        port(I1, I2, I3: BIT; O: out BIT);
end OR3;

architecture BEHAVIOR of OR3 is
begin
        O <= I1 or I2 or I3;
end BEHAVIOR;
use work.all;
entity MAJ3 is
        port(X: in BIT_VECTOR(2 downto 0); Z: out BIT);
end MAJ3;

architecture AND_OR of MAJ3 is
component AND2C
        port (I1, I2: in BIT; O: out BIT);
end component;
component OR3C
        port (I1, I2, I3: in BIT; O: out BIT);
end component
for all: AND2C use entity AND2(BEHAVIOR);
for all: OR3C use entity OR3(BEHAVIOR);
signal A1, A2, A3: BIT;
begin
        G1: AND2C
                port map (X(0), X(1), A1);
        G2: AND2C
                port map (X(0), X(2), A2);
        G3: AND2C
                port map (X(1), X(2), A3;
        G4: OR3C
                port map (A1, A2, A3, Z);
end AND_OR;
```

```
use work.all;
architecture STRUCTURAL of ONES_CNT is
        component MAJ3C
                port (X: in BIT_VECTOR(2 downto 0);
                        Z: out BIT);
        end component;
        component OPAR3C
                port (X: in BIT_VECTOR(2 downto 0);
                        Z: out BIT);
        end component;
        for all: MAJ3C use entity MAJ3 (AND_OR);
        for all: OPAR3C use entity OPAR3C (AND_OR);
        begin
                COMP1: MAJ3C
                        port map (A, C(1));
                COMP2: OPAR3C
                        port map (A, C(0));
end STRUCTURAL;
```

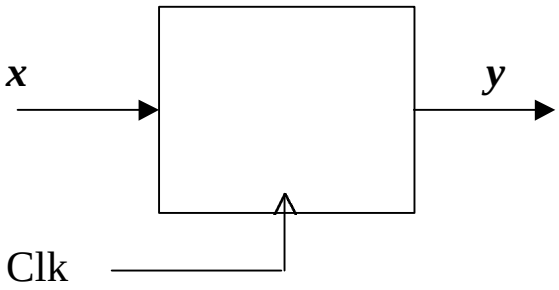
Structural Architectural Body for the Ones Counter.

MODELING STYLES



2.2. Sequential Logic

A **sequential circuit** is a circuit with memory. **A Finite State Machine** (FSM) is a mathematical model of a system with discrete inputs, discrete outputs and a finite number of internal configurations or states. The state of a system completely summarizes the information concerning the past inputs to the system that is needed to determine its behavior on subsequent inputs.
This high-level FSM model has only one input channel and only one output channel. Variables in the specification of a design are *multiple-valued* or *symbolic*. The symbolic variable takes on *symbolic values*.

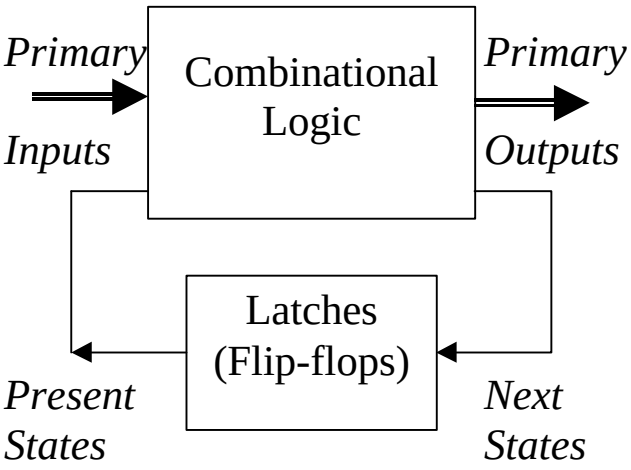


A sequential circuit is said to be synchronous if the internal state of the machine changes at specific instants of of time as governed by a clock.

Circuits with an acyclic underlying topology are combinational. Feedback (cyclic) is a necessary condition for a circuit to be sequential.

FSM as algebraic system is a quintuple A = ***á*** *S, I, O,* ***d****,* ***l*** ***ñ****,* where
S is a finite non-empty set of states,
I is a finite non-empty set of inputs and
O is a finite set of outputs.
d*: I ´ S ® S* is called transition (or next state function) and
λ is called the output function of *A*
l*: I ´ S ® O* for Mealy type FSM,
l*: S ® O* for a Moore machine,.
Note that any Moore machine can be converted into a Mealy machine with the same number of states and state transitions.

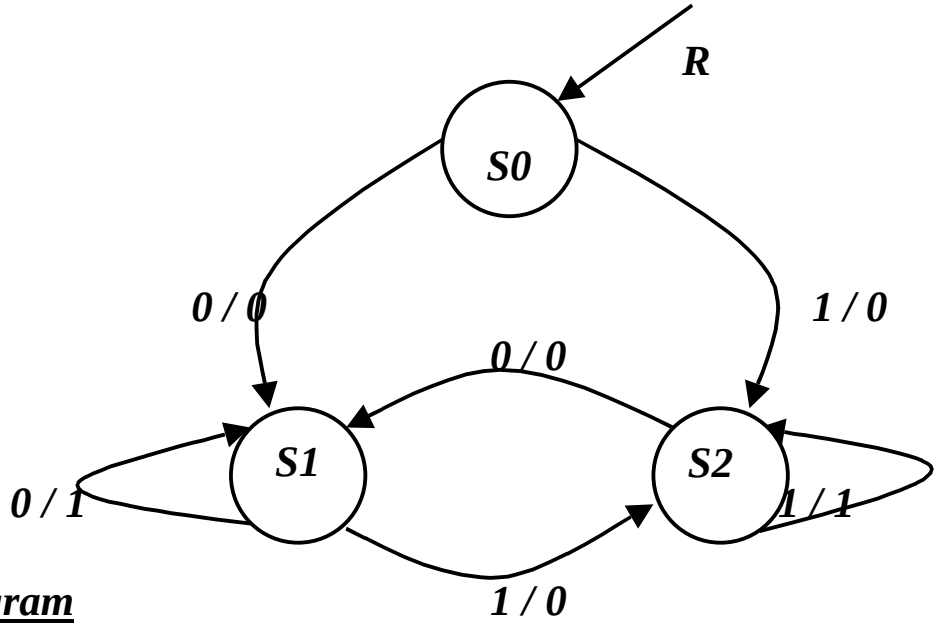
A general logic-level synchronous sequential circuit



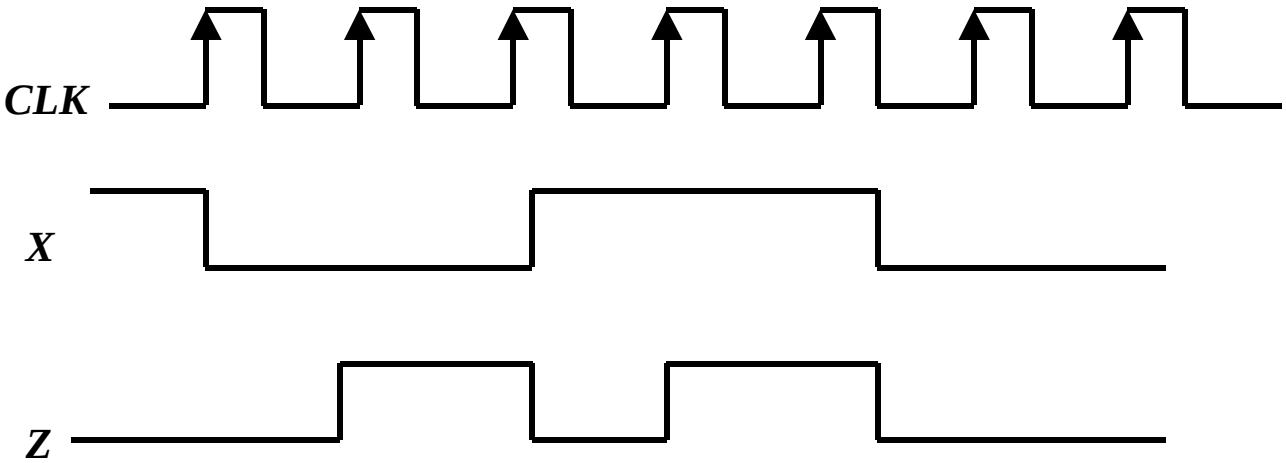
Logic-level description consists of a combinational logic block and state registers (latches or flip-flops) that hold the state information.
The combinational block is an interconnection of gates that implements the mapping between the **primary** input (PI) and present-state (PS), and primary output (PO) and next-state (NS). A state is a bit vector of length equal to the number of memory elements (latches or flip-flops) in the sequential circuit. Each state has a unique bit vector representing that state, and this bit vector is known as the **state code**. The process of assigning a code to each state is known as **state assignment** or **state encoding**.

Example represetatations of the device which ditects two or more consecutive 1's or two consecutive 0' on its input *X*.

Block Diagram



State Diagram



Timing Diaram

state	*X*	
	0	**1**
S0	S1 / 0	S2 / 0
S1	S1 / 1	S2 / 0
S2	S1 / 0	S2 / 0

State Table

state	*code*
	y1y0
S0	*00*
S1	*01*
S2	*11*

State Assignment

code *y1 y0*	*X*	
	0	**1**
00	0	1
01	0	1
11	0	1
10	-	-

Truth Table (K-map)

Y1

code *y1 y0*	***X***	
	0	**1**
00	1	1
01	1	1
11	1	1
10	-	-

Truth Table (K-map)

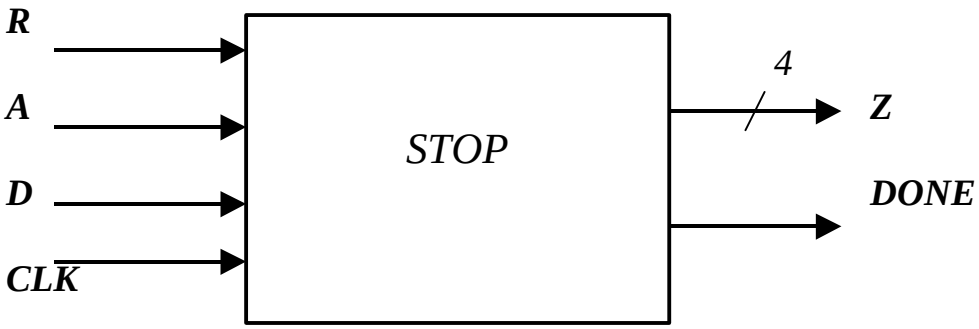
Y0

code *y1 y0*	***X***	
	0	**1**
00	0	0
01	1	0
11	0	1
10	-	-

Truth Table (K-map) Z

2.3. Multilevel Design of Sequential Logic Circuits.

Design a serial to parallel converter.



A word description of an example device:

Reset signal R is synchronous. If R = 1 at the end of any clock period, the device must enter the reset state.

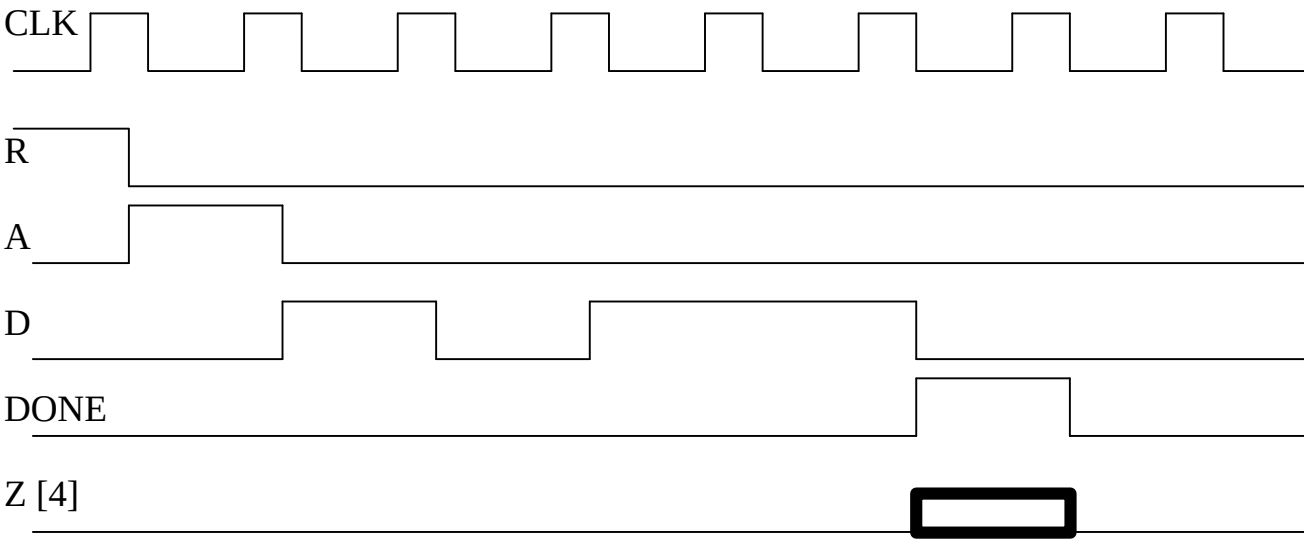
Input A will be asserted for exactly one clock period prior to the arrival of serial data on input D.

For the next four clock periods, data will arrive serially on line D.

The device must collect the 4 bits of serial data and output the 4 bits in parallel at output Z, which is a 4-bit vector.

During the clock period when the parallel data is present at Z, signal DONE will be asserted. The outputs Z and DONE must remain asserted for one full clock period.

DONE alerts the destination device that data is present on Z.



Design process:

1. A word description for a device.

2. Moore or Mealy decision:

The **State Transition Graph** (STG) or state diagram of the Moore machine has the output associated with the states, while in the Mealy model, the outputs are associated with the edges.

- A Moore machine may require more states than the corresponding Mealy machine.
- Moore device isolates the outputs from the inputs
- Mealy machine can respond one clock period one clock period earlier than the Moore machine to input changes, but noise on the input lines may be transferred to the outputs.

For the several to parallel converter (STOP), the output must be present during the clock period following the last input. Since the last input is no longer available, the outputs cannot depend on the inputs. Also, since the outputs are specified to be constant during the entire clock period, a Mealy machine cannot be used. Therefore we must design a Moore machine.

3. Construction of a state table.

We follow the structured methodology.

Two techniques:

a) state diagrams
b) transition lists

3.1. Creating a state diagram (STG)

To construct a state diagram, start with a state that is easily described in words. If there is a reset state, that is always a very good place to start.

We recommend writing a complete word description of each state as it is created.

The process is iterative in nature.

State S0: The Reset State.

The device enters this state at the end of any clock period in which input R=1 regardless of the values of any input. The device will stay in this state until there is a logic 1 on line A.
When in state S0, DONE = 0which means that the data on line Z will be ignored by the destination. This means that we are free to place anything at all on output Z.

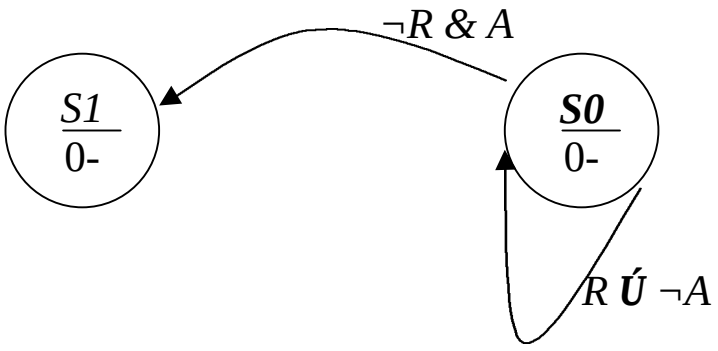
The next step is to decide what to do when device is in state S0 for various conditions that may exist on the device inputs. If the device is in state S0, it will stay in S0 if R=1 regardless of the values of any other input. When R=0, the device also will stay in state S0 when A=0. The complete condition for the device to state in state S0 is

$$R \acute{U} (\neg R \& \neg A) = R \acute{U} \neg A$$

If R = 0 and A = 1 during some clock period, then the device must get ready to receive data on line D during the next 4 clock periods. This means that it must enter a new state at the end of clock period.

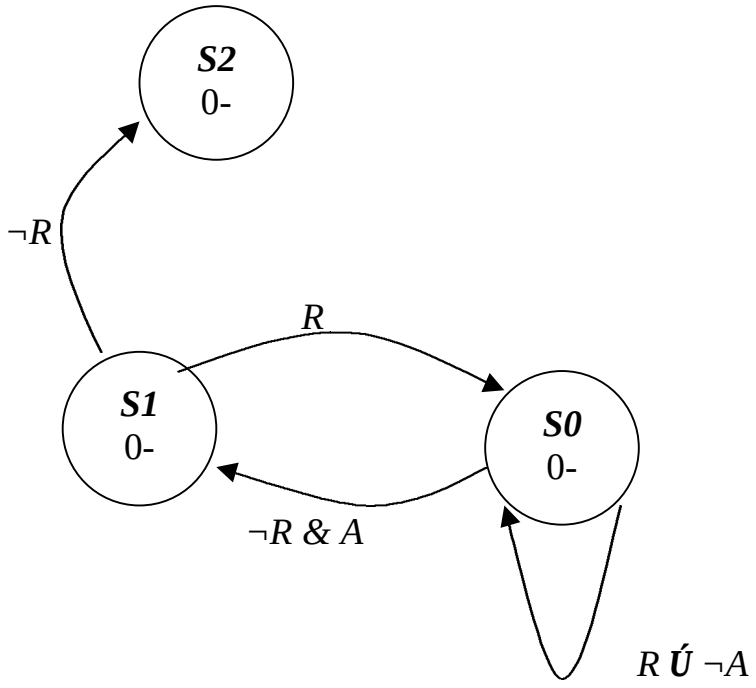
State S1.

The device enters this state from state S0 when R = 0 and A = 1.
When the device is in state S1 the first data value will be present on line D and must be saved at the end of clock period for later output.
When in state S1, output DONE = 0 and Z is unspecified.

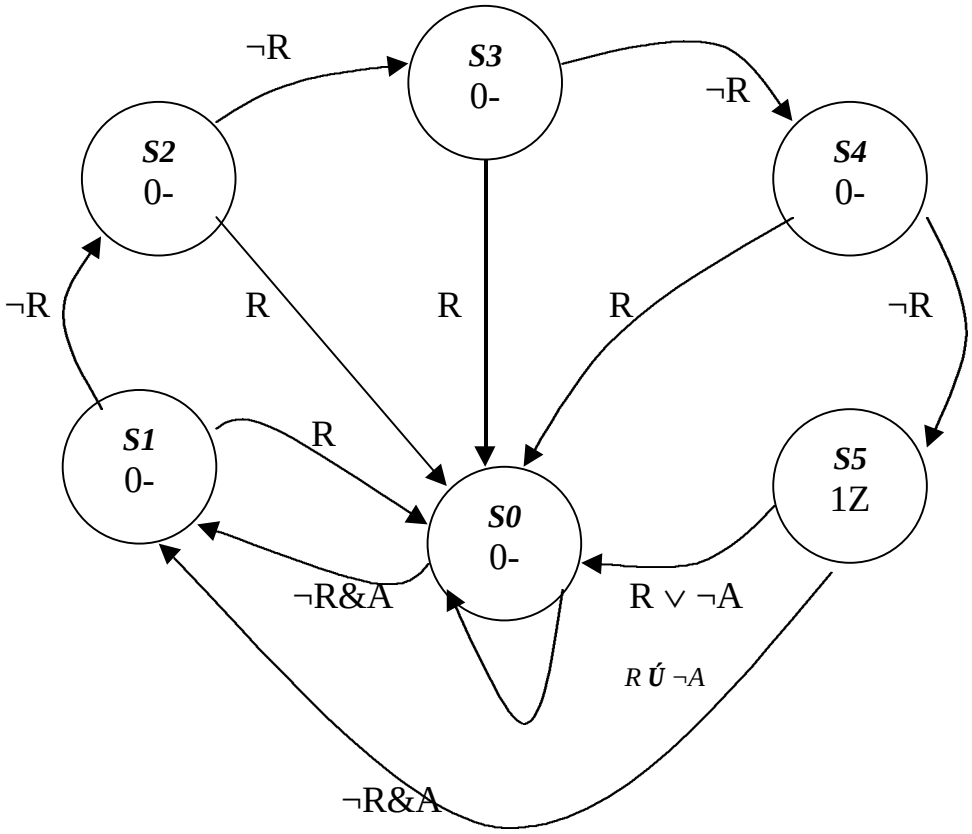


State S2.

State S1 transitions to state S2 when R = 0.
When in state S2, the second data value will be present on line D and must be saved at the end of clock period.
In state S2, the output DONE = 0 and Z is unspecified.
Therefore, a new node is added to the state diagram under construction for state S2. An arc from S1 to S2 with label ¬R indicates the desired state transition.



This figure shows the final state diagram (STG) for device STOP.



Final state diagram for serial to parallel converter

3.2. **Transition List Approach**

Present state	Transition Expression	Next State	Data Transfers	Output
S0	R ∨ ¬A	S0	None	DONE =0
S0	¬R & A	S1		Z unspec.
S1	¬R	S2	Store bit 1	DONE = 0
S1	R	S0		Z unspec.
S2	¬R	S3	Store bit 2	DONE = 0,
S2	R	S0		Z unspec.
S3	¬R	S4	Store bit 3	DONE = 0,
S3	R	S0		Z unspec.
S4	¬R	S5	Store bit 4	DONE = 0
S4	R	S0		Z unspec.
S5	¬R & A	S1	None	DONE = 1, Z = paral. data out
S5	R ∨ ¬A	S0		

Principle of Mutual Exclusion.

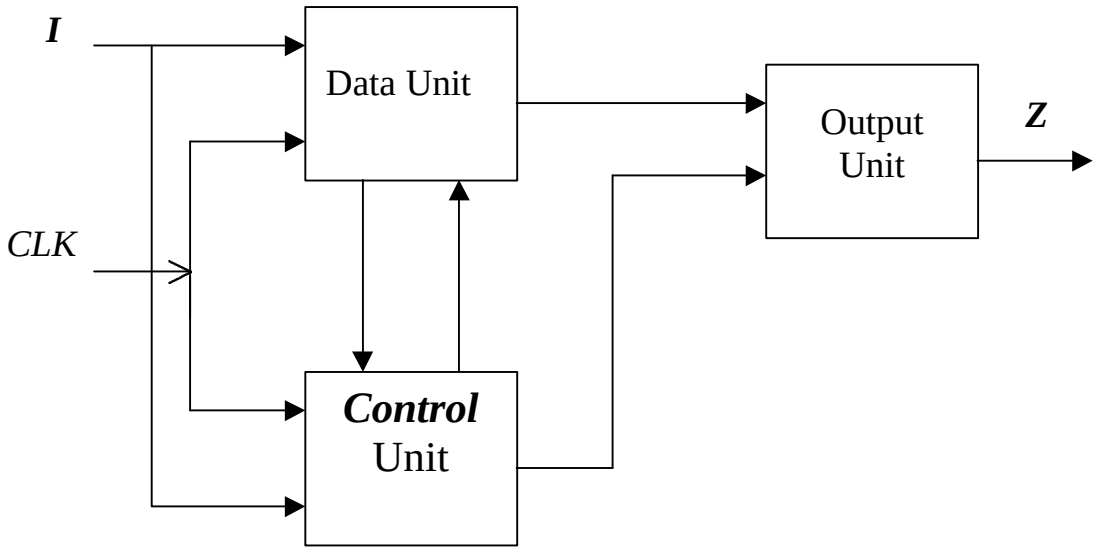
No two expressions on different arcs can be true simultaneously.

For example for node S0,

$$(\neg R \,\&\, A)\&(R \vee \neg A) = \neg RAR \vee \neg RA\neg A = 0$$

```
entity STOP is
port (R, A, D, CLK: in BIT;
        Z: out BIT_VECTOR (3 downto 0);
        DONE: out BIT);
end STOP;

architecture FSM_RTL of STOP is
        type STATE_TYPE is (S0, S1, S2, S3, S4, S5);
        signal STATE: STATE_TYPE;
        signal SHIFT_REG: BIT_VECTOR (3 downto 0);
begin
        STATE: process (CLK)
        begin
                if CLK = '1' then
                        case STATE is
                        when S0 =>
                                if R = '1' or A = '0' then
                                        STATE <= S0;
                                elsif R = '0' and A = '1' then
                                        STATE <= S1;
                                end if;
                        when S1 =>
                        SHIFT_REG <= D & SHIFT_REG(3 downto 1);
                                if R = '0' then
                                        STATE <= S2;
                                elsif R = '1' then
                                        STATE <= S0;
                                end if;
                        when S2 =>
                        SHIFT_REG <= D & SHIFT_REG (3 downto 1);
                                if R = '0' then
                                        STATE <= S3;
                                elsif R = '1' then
                                        STATE <= S0;
                                end if;
```

```
                when S3 =>
                SHIFT_REG <= D & SHIFT_REG (3 downto1);
                        if R = '0' then
                                STATE <= S4;
                        elsif R = '1' then
                                STATE <=S0;
                        end if;
                when S4 =>
                SHIFT_REG <= D & SHIFT_REG (3 downto1);
                        if R = '0' then
                                STATE <= S5;
                        elsif R = '1' then
                                STATE <=S0;
                        end if;
                when S5 =>
                        if R = '0' and A ='1' then
                                STATE <= S1;
                        elsif R = '1' or A = '0' then
                                STATE <= S0;
                        end if;
                end case;
        end if;
    end process STATE;

    OUTPUT: process (STATE)
    begin
        case STATE is
        when S0 to S4 =>
                DONE <= '0';
        when S5 =>
                DONE <= '1';
                Z <= SHIFT_REG;
        end case;
    end process OUTPUT;
end FSM_RTL;
```



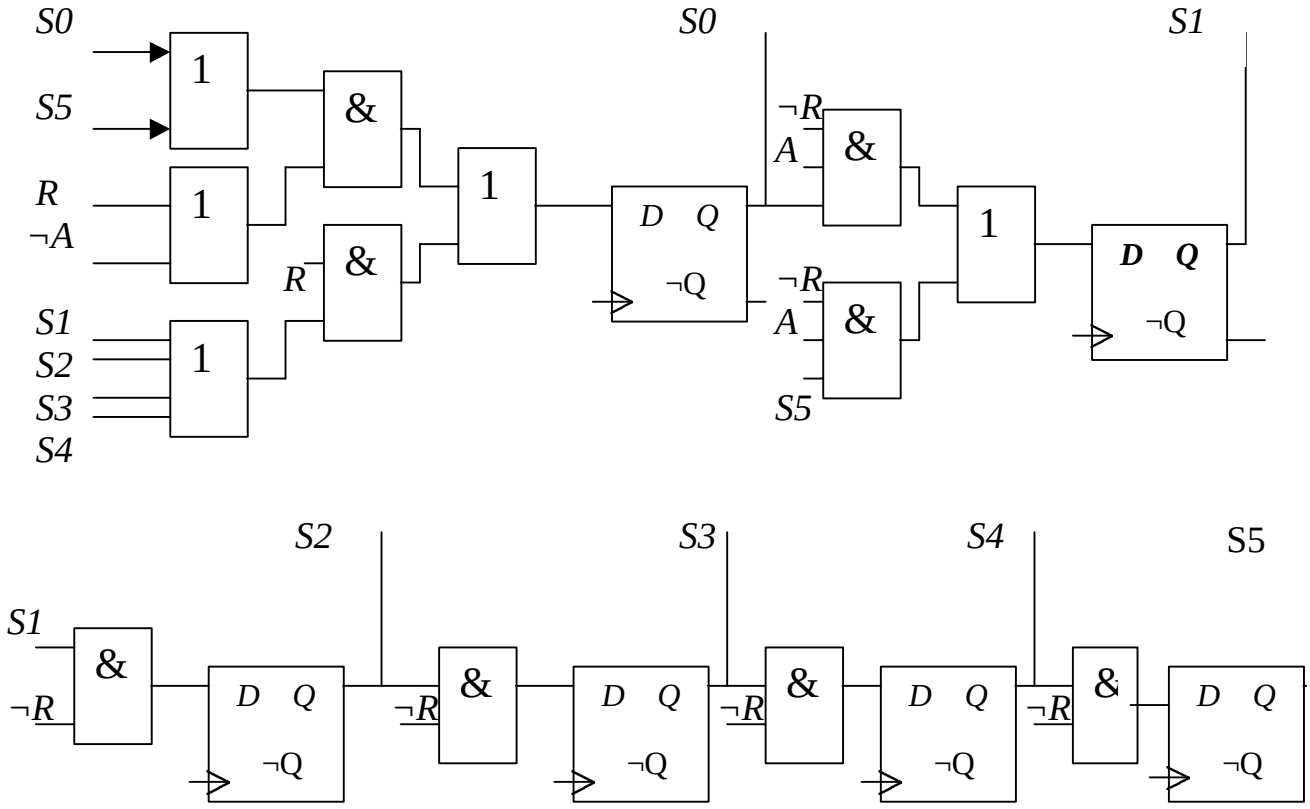
Block diagram for model of a state machine.

Control circuit synthesized from VHDL description (serial to parallel converter)

TO STATE	FROM STATE	CONDITION
S0	S0	R + ¬A
S0	S1	R
S0	S2	R
S0	S3	R
S0	S4	R
S0	S5	R + ¬A
S1	S0	¬R A
S1	S5	¬R A
S2	S1	¬R
S3	S2	¬R
S4	S3	¬R
S5	S4	R

Control section synthesis list for serial to parallel converter.

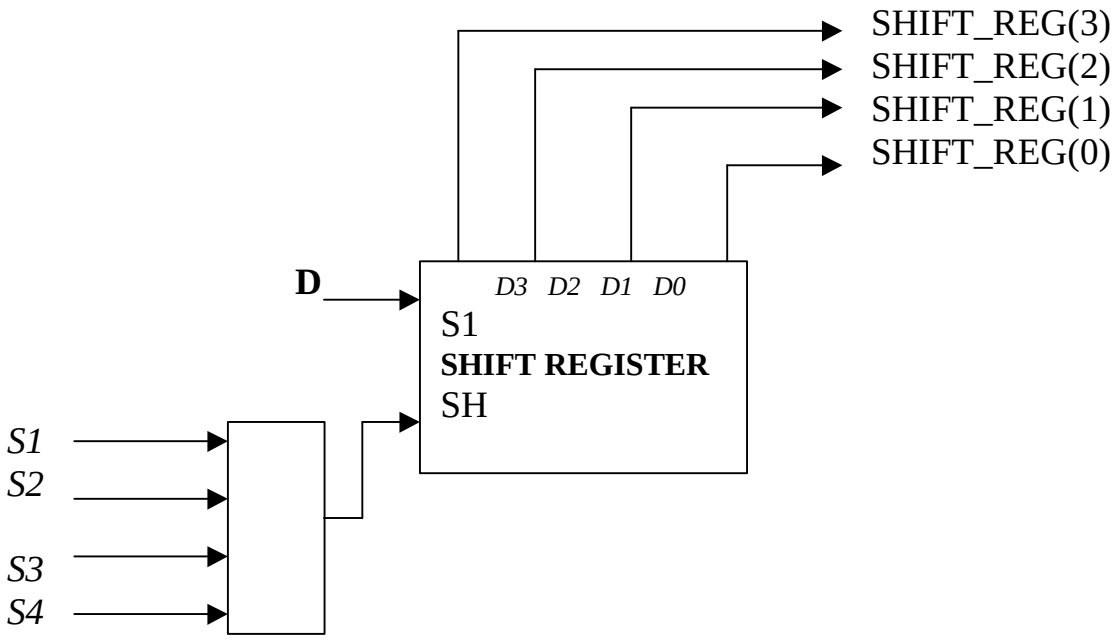
One-hot coding stile



Synthesis list for data unit for serial to parallel converter.

Data Transfer	State	Condition
SHIFT_REG <= D & SHIFT_REG (3 downto 1)	S1	1
SHIFT_REG <= D & SHIFT_REG (3 downto 1)	S2	1
SHIFT_REG <= D & SHIFT_REG (3 downto 1)	S3	1
SHIFT_REG <= D & SHIFT_REG (3 downto 1)	S4	1

Data unit for serial to parallel converter.



SHIFT = S1+S2+S3+S4

The output unit

$$\begin{cases} DONE = S5 \\ Z = SHIFT\_REG \end{cases}$$