

|  |
| --- |
| Group 2 |
| CST 2540 Group Coursework 2 G9 |
|  |

|  |
| --- |
|  |

## Amal Ali Anwar M00734729

## Mirza Mohammad Faiq Baig M00696479

## Hamad Mohamed Asif M00695912

## **Tutor: Madhumita Das**

## **Module Leader: Dr Purav Shah**

14th April 2021

**INDIVIDUAL TASKS:**

A sequential circuit has two inputs (X1, X2) and one output (Z). The output remains a constant value unless one of the following input sequences occurs:

a. The input sequence X1 X2 = 01, 11 causes the output to become ‘0’.

b. The input sequence X1 X2 = 10, 11 causes the output to become ‘1’.

c. The input sequence X1 X2 = 10, 01 causes the output to change value.

The notation X1 X2 = 10, 01 means X1 = 0, X2 = 1 followed by X1 = 1, X2 = 1.

Derive a Moore state graph and state table for the circuit.

We have a sequential circuit with two inputs (X1, X2) and one output (Z). Z is said to remain unchanged (either 1 or 0) unless certain input sequences take place. They are

1. X1 X2 = 01, 11 will result in Z = 0.
2. X1 X2 = 10, 11 will result in Z = 1.
3. X1 X2 = 10, 01 will result in Z = changed value (could be either 1 or 0, depending on the situation)

With respect to the above input sequences and condition of Z mentioned, a state graph for the sequential circuit has to be derived.

To start off, for six inputs (01,11,10,11,10,01) six states can be defined.

|  |  |  |
| --- | --- | --- |
| Previous Input (X1 X2) | Output (Z) | State Designation |
| 11 or 00 | 0 | S0 |
| 11 or 00 | 1 | S1 |
| 01 | 0 | S2 |
| 01 | 1 | S3 |
| 10 | 0 | S4 |
| 10 | 1 | S5 |

The input can either be 11 **or** 00 because neither input will start a changed sequence in output. From the input 00 or 11, the output can be in either of two basic states – 1 or 0. The output will remain latched unless an input aside from 00/11 is given.

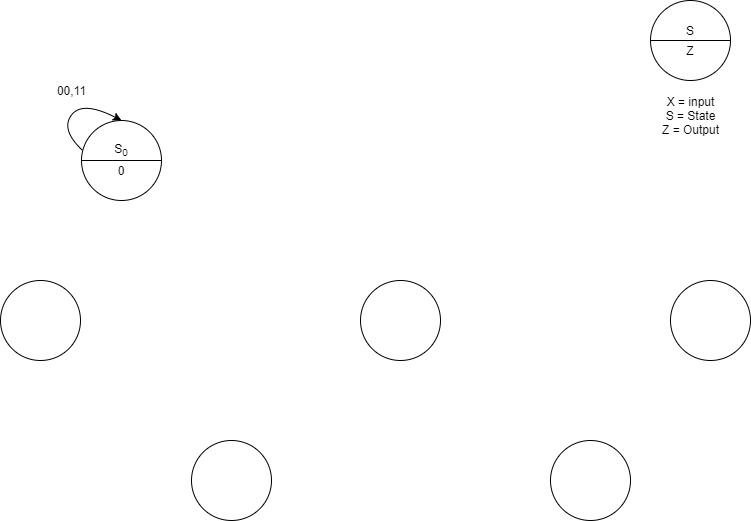
Now, a state table can be made: -

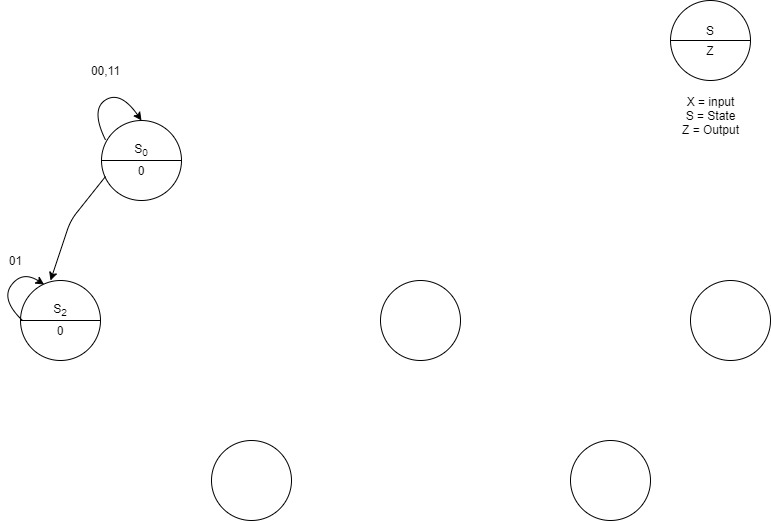
Next State (Input Sequence)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| State Designation | Previous State | Output (Z) | 00 | 01 | 11 | 10 |
| S0 | 11 or 00 | 0 | 0 | 0 | 0 | 0 |
| S1 | 11 or 00 | 1 | 1 | 1 | 1 | 1 |
| S2 | 01 | 0 | 0 | 0 | 0 | 0 |
| S3 | 01 | 1 | 1 | 1 | 0 | 1 |
| S4 | 10 | 0 | 0 | 1 | 1 | 0 |
| S5 | 10 | 1 | 1 | 0 | 1 | 0 |

**S0 ROW**

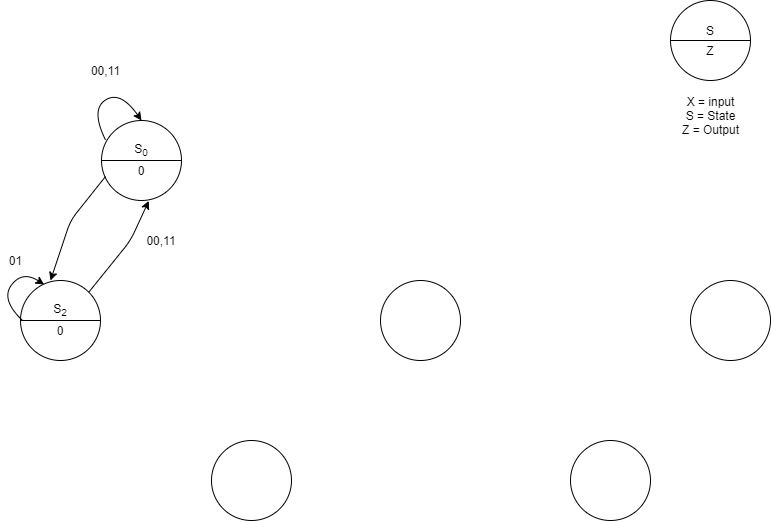
When 00 is received then the input sequence is 00,00, we go back to the same state (S0).



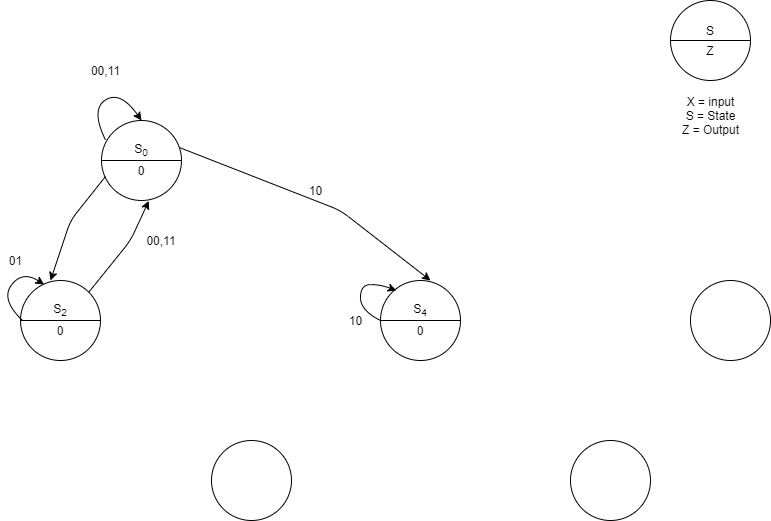
When 01 is received, then the input sequence is 00,01 we will move to S2 (output will be 0).

01

When 11 is received, then the input sequence is 00,11 and we go back to S0 (output is 0).



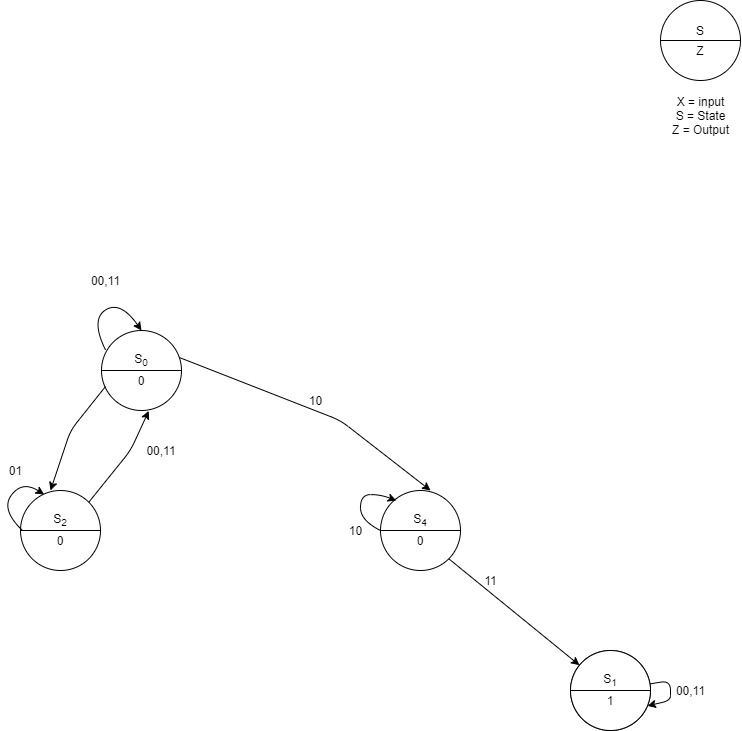
01

When 10 is received, the input sequence is 00,10 and so we go to S4 (output is 0).

01

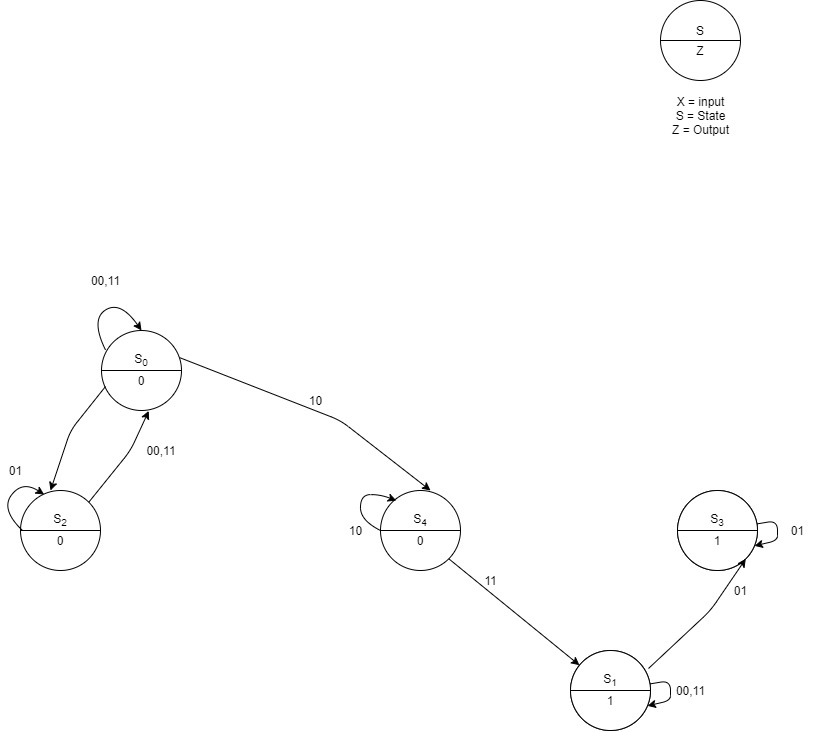
**S1 ROW**

Moving on, when 00 is received, the input sequence is 00,11 and so we go to S1 and the output will now change to 1.



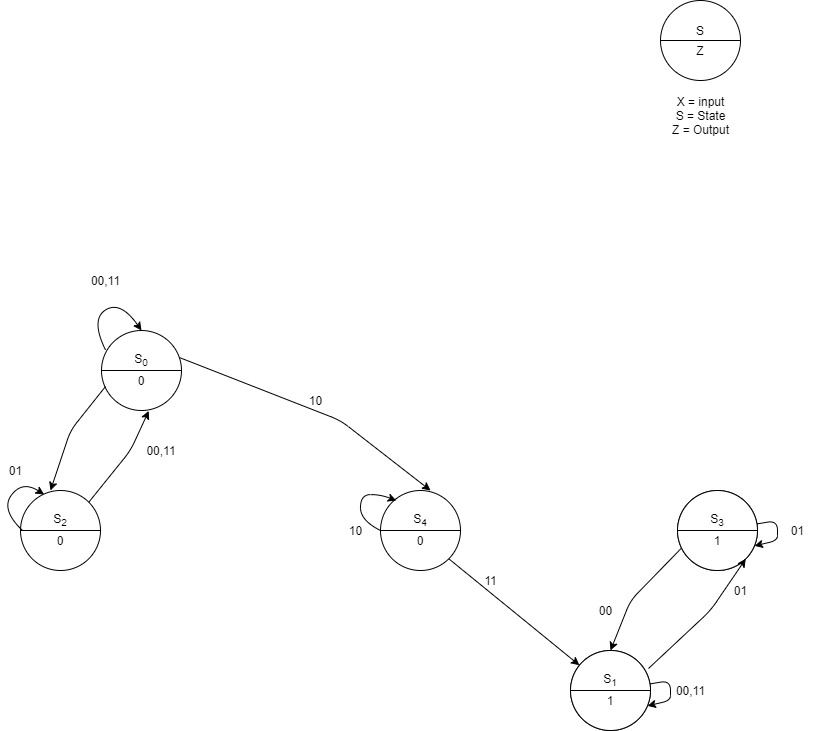
01

When 01 is received, the input sequence will be 11,01 and we will move to S3 (output will be 1).



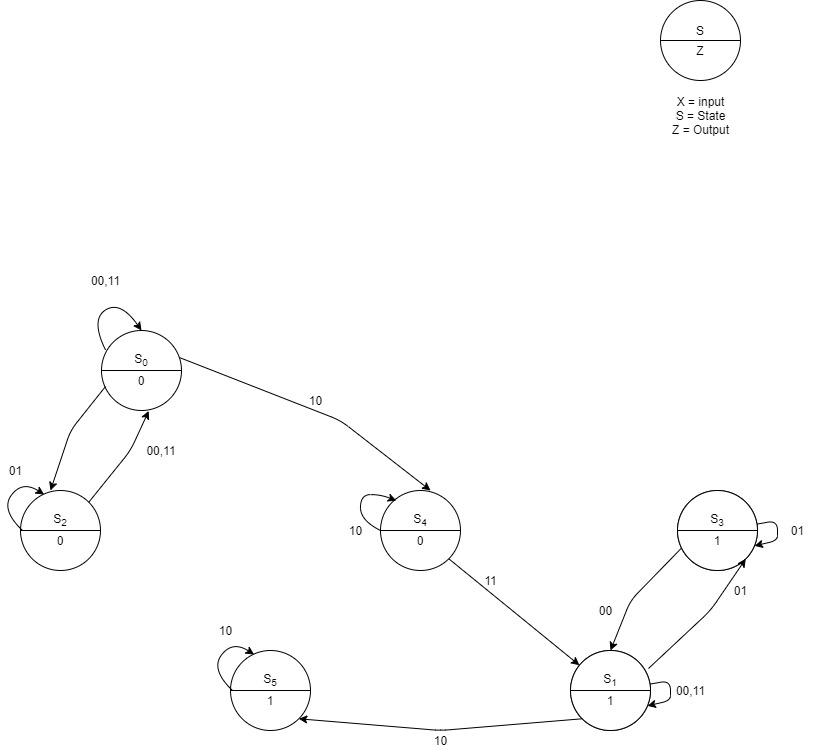
01

When 11 is received, the input sequence becomes 11,11 and so we go back to S1. (output will be 1).



01

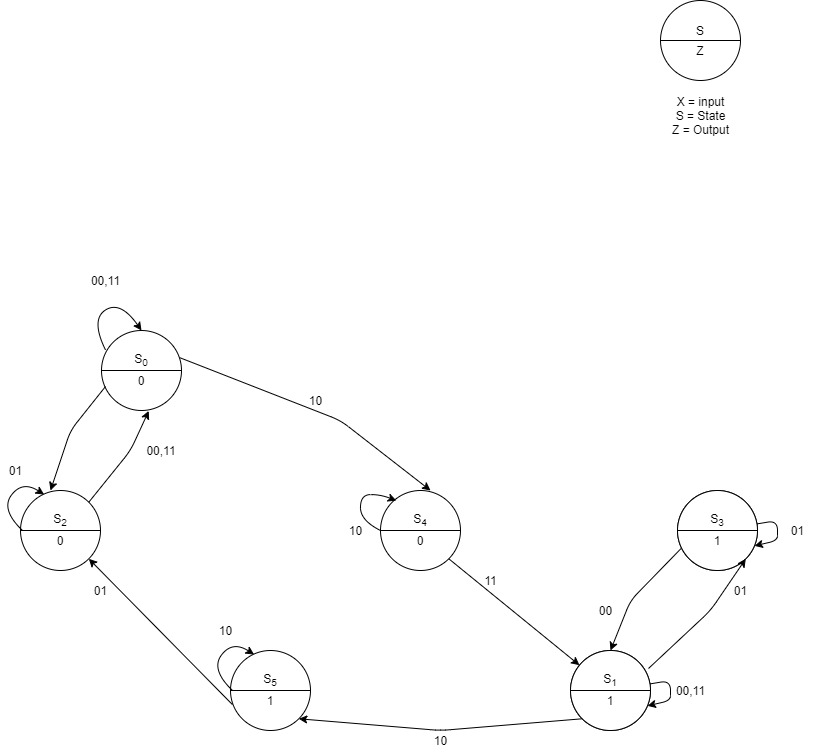
When 10 is received, the input sequence is 11, 10 and we move to S5 (output will be 1).



01

**S2 ROW**

When 00 is received, the input sequence is 01,00 and we move to S0 because the output will be 0.

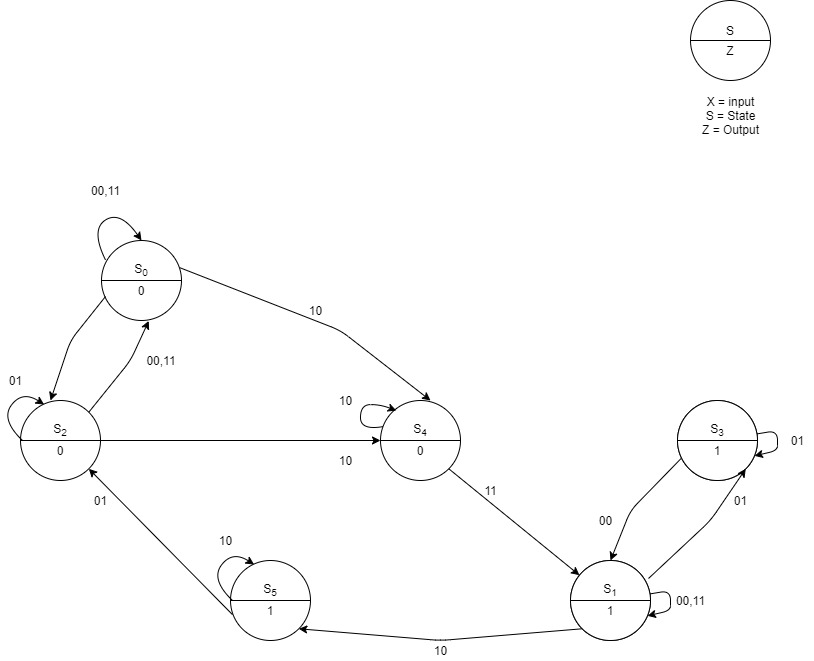


01

When 01 is received, the input sequence is 01, 01 and we move to S2 (output will be 0).

When 11 is received, the input sequence becomes 01, 11 and we go to S0 (output will be 0).

When 10 is received, the input sequence is 01, 10 and we go to S4 (output will be 0).



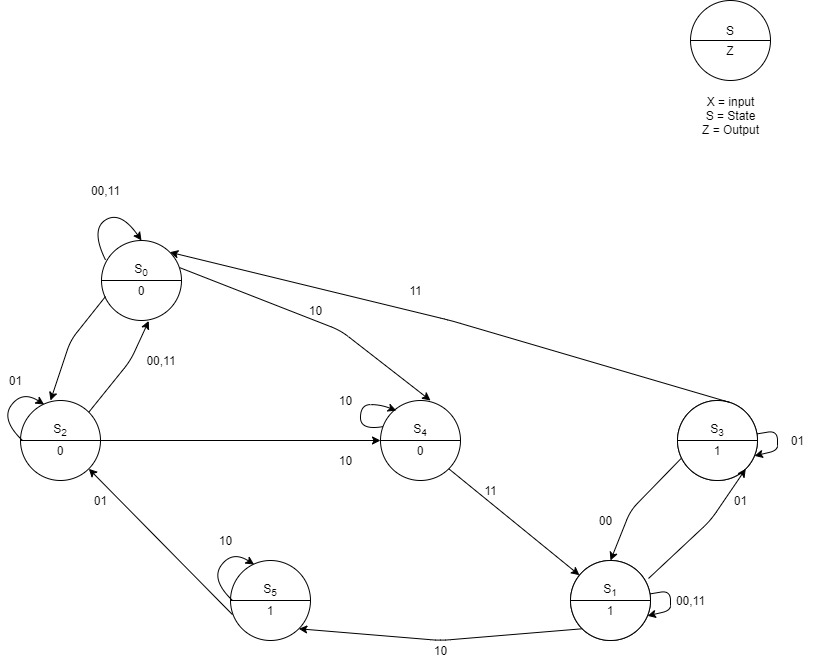
01

**S3 ROW**

When 00 is received, the input sequence is 01,00 and we go to S1 (output will be 1).

When 01 is received, the input sequence is 01, 01 and we go to S3 (output will be 1).

When 11 is received, the input sequence is 01, 11 and we go to S0 (output will be 0).

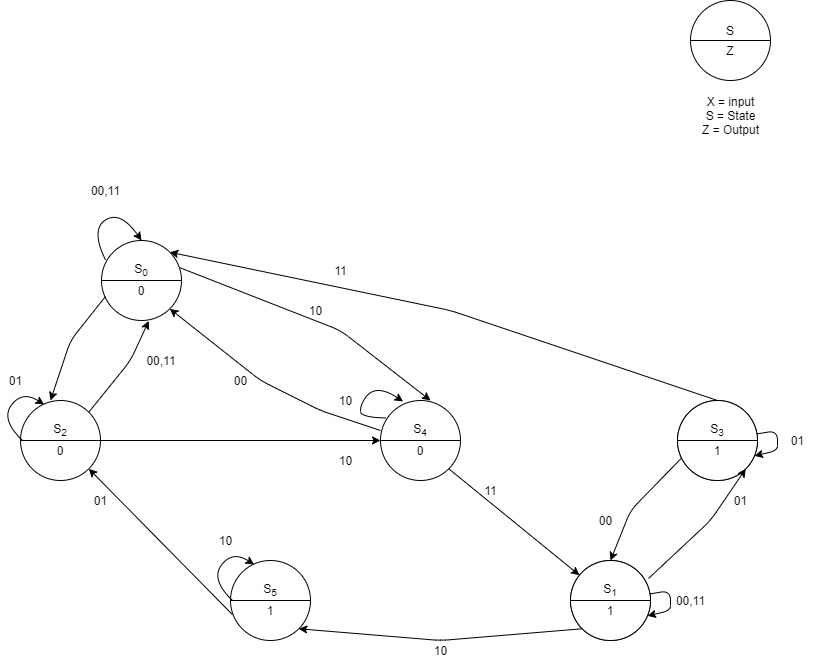


01

When 10 is received, the input sequence is 01, 10 and we move to S5 (output will be 1).

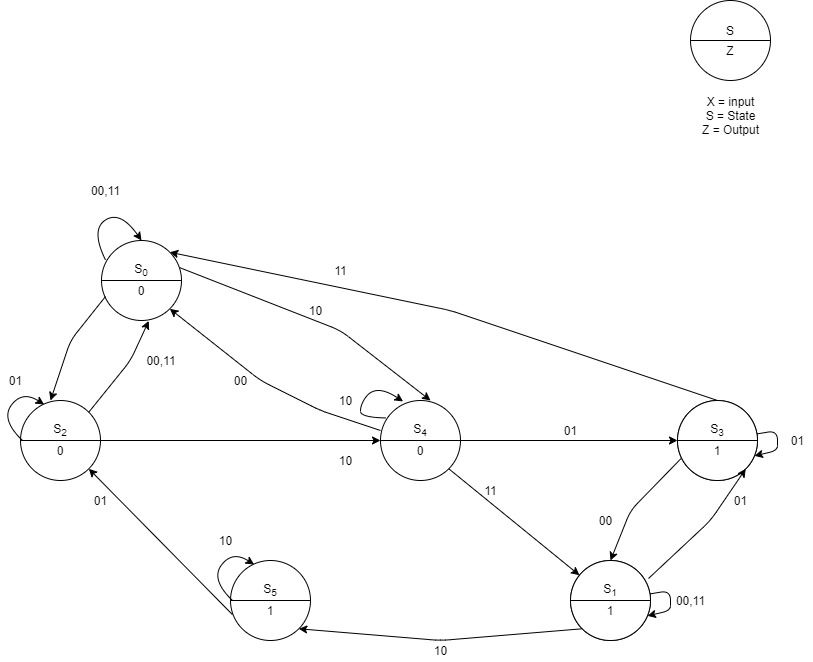
**S4 ROW**

When 00 is received, the input sequence is 10, 00 and we go to S0 (output will be 0).



01

When 01 is received, the input sequence is 10, 01 and we move to S3 (output will be 1).



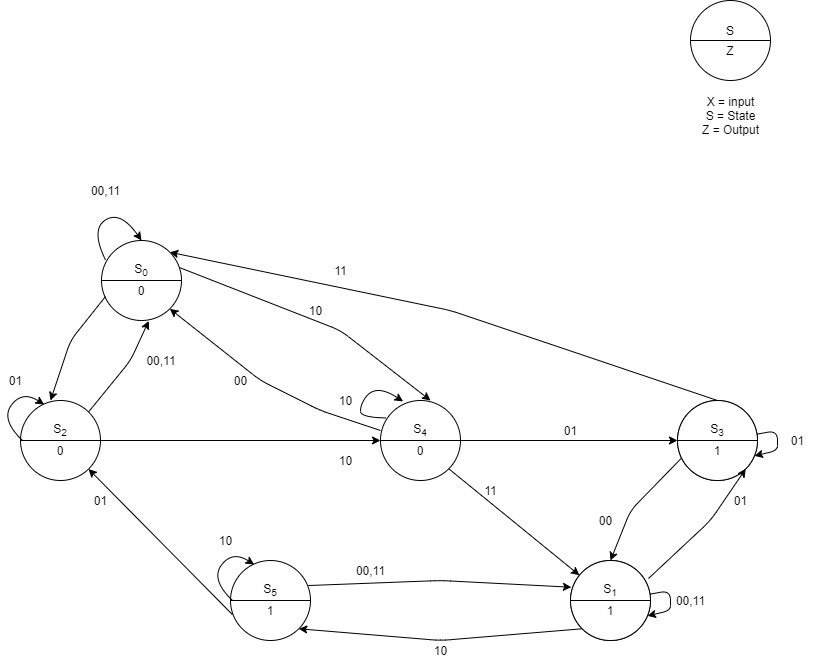
01

When 11 is received, the input sequence is 10, 11 and we move to S1 rather than moving to S0 because the output is 1.

When 10 is received, the input sequence is 10,10 and we move to S4 (output will be 0).

**S5 ROW**

When 00 is received, the input sequence is 10, 00 and we go to S1



01

When 01 is received, the input sequence is 10, 01 and we move to S2

When 11 is received, the input sequence is 10, 11 and we move to S1

When 10 is received, the input sequence is 10,10 and we move to S5

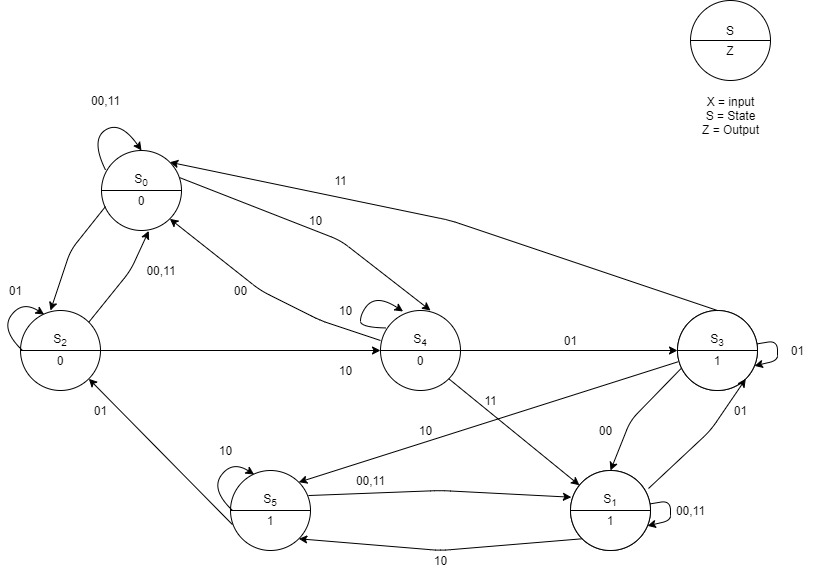
A simpler version of the state table: -

Next State (Input Sequence)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Previous Input | Output (Z) | 00 | 01 | 11 | 10 |
| S0 | 0 | S0 | S2 | S0 | S4 |
| S1 | 1 | S1 | S3 | S1 | S5 |
| S2 | 0 | S0 | S2 | S0 | S4 |
| S3 | 1 | S1 | S3 | S0 | S5 |
| S4 | 0 | S0 | S3 | S1 | S4 |
| S5 | 1 | S1 | S2 | S1 | S5 |

At last, the final State Graph: -

01



**2.** Following are non-valid BCD encodings for the decimal digits:

* 1010
* 1011
* 1100
* 1101
* 1110
* 1111

|  |  |
| --- | --- |
| State | Description |
| *S0* | *The 3 most recent bits are x00* |
| *S1* | *The 3 most recent bits are 001* |
| *S2* | *The 3 most recent bits are 010* |
| *S3* | *The 3 most recent bits are 011* |
| *S4* | *The 3 most recent bits are 101* |
| *S5* | *The 3 most recent bits are 110* |
| *S6* | *The 3 most recent bits are 111* |

The state machines would need to detect when the significant bits would be 101 or 11. The table below would define the states.

*Table 1*

I will use the states defined above and obtain the *Next State (N.S)* and the *Output (Z)* for the *Present State* and the *Input (X).*

Consider the *present state* as *S0*,

If the input is 1, then the *next state* is *S1* and the output is *Z = 0*.

If the input is 0, then the *next state* is *S0*and the output is *Z = 0.*

Consider the *present state* as *S1*,

If the input is 1, then the *next state* is *S3* and the output is *Z = 0.*

If the input is 0, then the *next state* is *S2* and the output is *Z = 0.*

Consider the *present state* as *S2*,

If the input is 1, then the *next state* is *S4* and the output is *Z = 0.*

If the input is 0, then the *next state* is *S0* and the output is *Z = 0.*

Consider the *present state* as *S3*,

If the input is 1, then the *next state* is *S6* and the output is *Z = 0.*

If the input is 0, then the *next state* is *S5* and the output is *Z = 0.*

Consider the *present state* as *S4*,

If the input is 1, then the *next state* is *S3* and the output is *Z = 1.*

If the input is 0, then the *next state* is *S2* and the output is *Z = 1.*

Consider the *present state* as *S5*,

If the input is 1, then the *next state* is *S4* and the output is *Z = 1.*

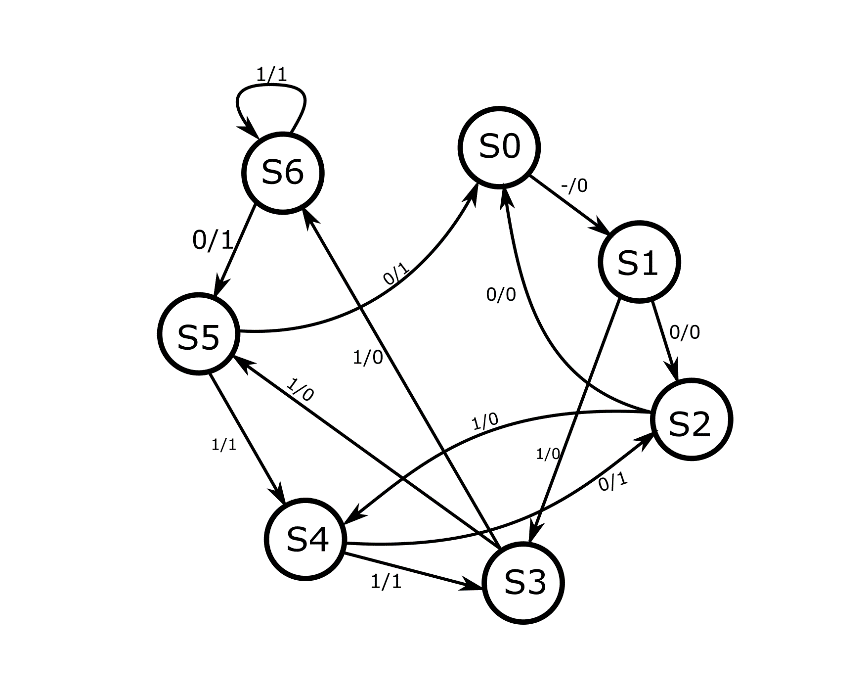
If the input is 0, then the *next state* is *S0* and the output is *Z = 1.*

Consider the *present state* as *S6*,

If the input is 1, then the *next state* is *S6* and the output is *Z = 1.*

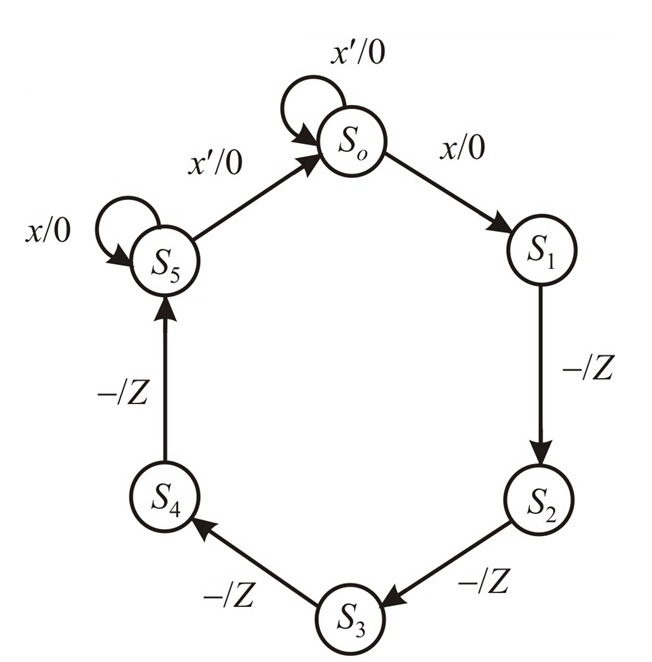
If the input is 0, then the *next state* is *S5* and the output is *Z = 1.*

Using the explanation, below is the state table and state graph for the *Mealy Sequential circuit*:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Present State* | *Next State* | | | *Output* | |
|  | *X = 0*  *X = 1* | | | *X = 0*  *X =1* | |
| *S0* | *S1* | *S1* | 0 | | 0 |
| *S1* | *S2* | *S3* | 0 | | 0 |
| *S2* | *S0* | *S4* | 0 | | 0 |
| *S3* | *S5* | *S6* | 0 | | 0 |
| *S4* | *S2* | *S3* | 1 | | 1 |
| *S5* | *S0* | *S4* | 1 | | 1 |
| *S6* | *S5* | *S6* | 1 | | 1 |

**3.** The initial state, or reset state, is when neither the button has been pushed nor is Z high. This is S0. The next state would be S1, where the button is pushed and X is high, and Z is high for its first clock cycle. S2 would be Z’s second clock cycle while it is high. S3 would be its third, and S4 would be the fourth and final clock cycle where Z is high. During all 4 of Z’s high states, X is a don’t-care since Z remains high for these 4 states regardless of what X’s value is. S5 is where X is still 1 and Z returns to low, and the machine remains in this state as long as X stays pressed after Z is done with its clock cycles. Upon X going back to low (which means the button is no longer being pressed), the machine enters the reset state where X and Z are both low. Below is the state graph:



**GROUP TASKS:**

**1.a.**

**Design code:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity updown is

Port (clrn, clk, load, ent, enp, up: in std\_logic;

d: in std\_logic\_vector(3 downto 0);

q: out std\_logic\_vector(3 downto 0);

co: out std\_logic);

end updown;

architecture eqn of updown is

signal qint: std\_logic\_vector(3 downto 0) := "0000";

begin

q <= qint;

co <= ( qint(3) and qint(2) and qint(1) and qint(0) and ent and up)

or

( not qint(3) and not qint(2) and not qint(1) and not qint(0) and ent and not up);

process (clrn, clk)

begin

if clrn = '0' then

qint <= "0000";

elsif clk'event and clk = '1' then

if load = '0' then

qint <= d;

elsif (ent and enp and up) = '1' then

qint <= std\_logic\_vector( unsigned(qint) + 1 );

elsif (ent and enp and not up) = '1' then

qint <= std\_logic\_vector( unsigned(qint) - 1 );

end if;

end if;

end process;

end eqn;

**Testbench code:**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity testbench is

end entity;

architecture tb of testbench is

component updown is

port (clrn, clk, load, ent, enp, up: in std\_logic;

d: in std\_logic\_vector(3 downto 0);

q: out std\_logic\_vector(3 downto 0);

co: out std\_logic);

end component;

signal clk, co: std\_logic := '0';

signal load, clrn, ent, enp, up: std\_logic := '1';

signal d, q: std\_logic\_vector(3 downto 0) := "0000";

begin

uut: updown port map(clrn, clk, load, ent, enp, up, d, q, co);

stim: process begin

for i in 0 to 17 loop

clk <= not clk;

wait for 1ns;

end loop;

-- counts down

up <= '0';

for i in 0 to 17 loop

clk <= not clk;

wait for 1ns;

end loop;

wait;

end process;

end tb;

**1.b.**

**Design code:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity updown8bit is

Port (clrn, clk, load, ent, enp, up: in std\_logic;

d: in std\_logic\_vector(7 downto 0);

q: out std\_logic\_vector(7 downto 0);

co: out std\_logic);

end updown8bit;

architecture structure of updown8bit is

component updown is

Port (clrn, clk, load, ent, enp, up: in std\_logic;

d: in std\_logic\_vector(3 downto 0);

q: out std\_logic\_vector(3 downto 0);

co: out std\_logic);

end component;

signal co1: std\_logic;

signal q1,q2: std\_logic\_vector(3 downto 0);

begin

c1: updown port map (clrn, clk, load, ent, enp, up, d(3 downto 0),q1,co1);

c2: updown port map (clrn, clk, load, co1, enp, up, d(7 downto 4),q2, co);

q <= q2 & q1;

end structure;

**Testbench code:**

**2.a.** Line 12, the word “select” is used as a variable. This will not work since select is a reserved keyword in vhdl.

In line 14, the first muxel argument is pointless since muxel gets incremented as a result of the ‘if’ statement with 'sel' in the next line.

**2.b.**

The output is given in a compound form, where for example 01 means Z1 = 0 and Z2 = 1

