

Logic Design Lab

ASSIGNMENT-2 DOCUMENTATION

ONE BIT SEQUENCE DETECTOR

SUBMITTED BY: GROUP 12

Astha Agarwal (110050018)

Ashish Sonone (110050022)

Anand Soni (110050037)

Mridul Jain (110040083)

Vikash Challa (110050077)

Objective

To implement a 1 bit sequence detector to detect the following sequences:

- * 100100
- * 110101
- * 1100

using a shift-register.

Output is single bit indicating valid sequence and a two-bit vector indicating the sequence detected.

Example: 1-> valid sequence detected / 0 otherwise,

01-> "100100" detected

10-> "110101" detected

11-> "1100" detected

00-> otherwise.

Preview

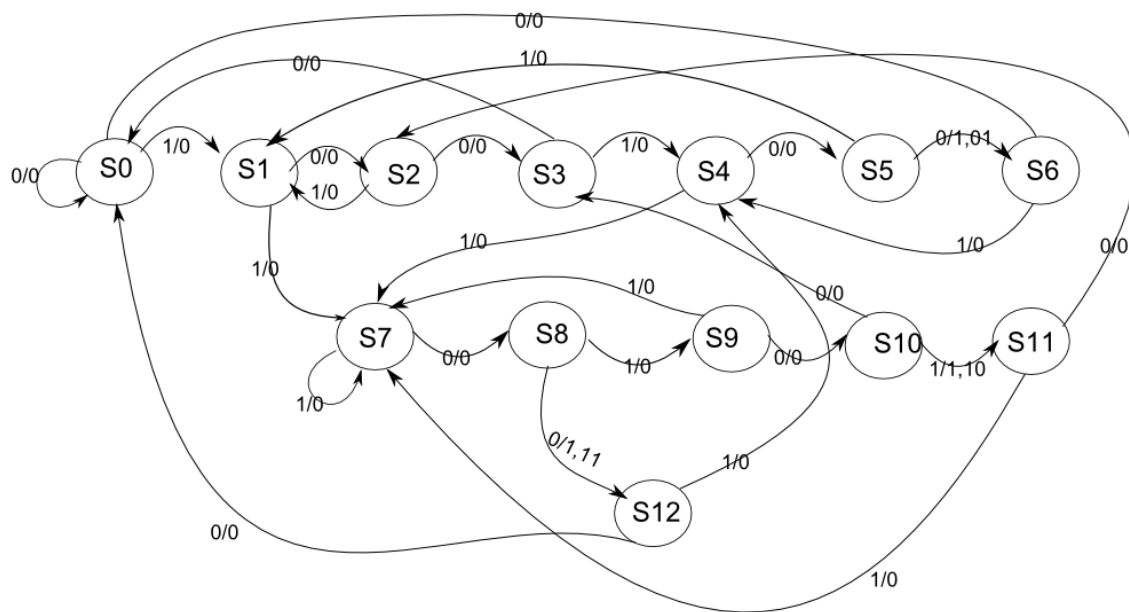
We are asked to design a 1-bit sequence detector. For each bit taken as input, we need to see whether the input sequence matches one of three given sequences: 100100, 110101 or 1100.

At any point of time we need to keep track of the sequence seen so far and on taking the current input bit, we either detect the sequence or not. Hence, our output is a function of present state and input. Thus we need to implement a sequential circuit. As we provide the input at discrete instants of time, therefore it must be a synchronous sequential circuit.

Since the question requires us to design it using a shift register, we are taking 8 bit-vector as input. We then use the bits one at a time to detect if any one of the above 3 sequences is present. Essentially we have implemented it using a parallel-in-serial-out shift register. The bits are however, considered to be continuous i. e. if 11101011 is first 8-bit input and 001101000 is our next 8-bit input then the input sequence will be 11101011001101000 and will detect 1100 formed by last two bits and first two bits bit of first and second input respectively.

Procedure

Our state machine starts in a state in which we have received no bits. We will call this state S_0 . The state transitions will be depending on whether the input X is a 0 or 1. For each state there will be two transitions one for each input 1 and 0. In this way we construct the following **state-transition diagram**:



Finite State Machine (Non-Minimised)

The directed lines are labelled with two binary numbers separated by a slash. The input value during the present state is labelled first and the number after the slash gives output of the present state with the given input. The above FSM detects the sequences 100100, 110101 and 1100.

So we have a state machine that has 13 states. 13 states is quite a lot. There must be some states which are equivalent. By equivalent we mean:

- Both states lead to the same next state.
- The current outputs for both states are the same.

We try to apply state reduction – look for redundancy or unnecessary repetition in the diagram, thereby replacing the multiple equivalent states by one and thus minimizing the FSM.

So for this purpose we convert the state diagram to **state-transition table** as below:

Present State	Next State X=0	Output X=0	Next State X=1	Output X=1	Last known symbols
S0	S0	0	S1	0	Nil
S1	S2	0	S7	0	1
S2	S3	0	S1	0	10
S3	S0	0	S4	0	100
S4	S5	0	S7	0	1001
S5	S6	1,01	S1	0	10010
S6	S0	0	S4	0	100100
S7	S8	0	S7	0	11
S8	S12	1,11	S9	0	110
S9	S10	0	S7	0	1101
S10	S3	0	S11	1,10	11010
S11	S2	0	S7	0	110101
S12	S0	0	S4	0	1100

From above state-transition table we can clearly see that states S_3 , S_6 and S_{12} are equivalent. So we replace all occurrences of S_6 and S_{12} by S_3 .

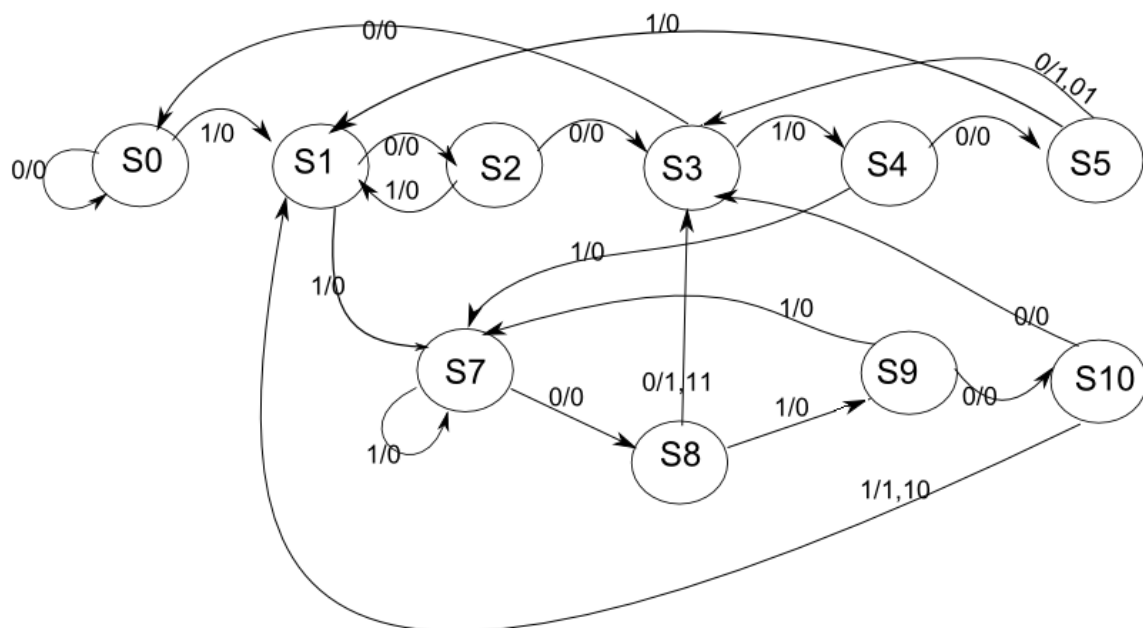
Also the states S_1 and S_{11} are equivalent. So, we again replace all occurrences of S_{11} by S_1 .

Present State	Next State X=0	Output X=0	Next State X=1	Output X=1	Last known symbols
S0	S0	0	S1	0	Nil
S1	S2	0	S7	0	1
S2	S3	0	S1	0	10
S3	S0	0	S4	0	100
S4	S5	0	S7	0	1001
S5	S3	1,01	S1	0	10010
S7	S8	0	S7	0	11
S8	S3	1,11	S9	0	110
S9	S10	0	S7	0	1101
S10	S3	0	S11	1,10	11010
S11	S2	0	S7	0	110101

Once we have eliminated enough states, write the minimized state table. The **minimized state table** is shown below. In producing the minimal state table the following state equivalences were identified: $S_3=S_6=S_{12}$, $S_1=S_{11}$.

Present State	Next State X=0	Output X=0	Next State X=1	Output X=1	Last known symbols
S0	S0	0	S1	0	Nil
S1	S2	0	S7	0	1
S2	S3	0	S1	0	10
S3	S0	0	S4	0	100
S4	S5	0	S7	0	1001
S5	S3	1,01	S1	0	10010
S7	S8	0	S7	0	11
S8	S3	1,11	S9	0	110
S9	S10	0	S7	0	1101
S10	S3	0	S1	1,10	11010

And finally we are left with 10 states which are all different and therefore the state transition diagram reduces to:




Minimised Finite State Machine (10 states)

Sample input-output

Now let us test our FSM on a sample input and see whether we get the output as expected. Suppose our input sequence is: 1101101001001001101100100 (25-bits)

On such above input sequence we get the following state transitions in the above FSM

1->7->8->9->7->8->9->10->3->4->5->3-> 4-> 5->3->4->7->8->9->7->8->3-> 4->5->3



Input	State	Output
1	S ₁	0
1	S ₇	0
0	S ₈	0
1	S ₉	0
1	S ₇	0
0	S ₈	0
1	S ₉	0
0	S ₁₀	0
0	S ₃	0
1	S ₄	0
0	S ₅	0
0	S ₃	1,01
1	S ₄	0
0	S ₅	0
0	S ₃	1,01
1	S ₄	0
1	S ₇	0
0	S ₈	0
1	S ₉	0
1	S ₇	0
0	S ₈	0
0	S ₃	1,11
1	S ₄	0
0	S ₅	0
0	S ₃	1,01

The above input sequence contains 4 detectable sequences (one 1100 and three 100100s) and are detected by the finite state machine. Thus, our obtained result matches the expected result.

Shift register

The shift register being a synchronous sequential circuit has a master clock – **clk_s**. Also there is a **load** input that loads the 8 input bits in the register simultaneously. When load is at logic-1, the data is transferred into the register at next falling edge of the clock. When load input is at logic-0, the data is transferred on bit at a time at each falling edge of the clock, in MSB first manner. The other two inputs are: a **parallel_input**, which is the 8-bit input `std_logic_vector` and **serial_output**, which is the `parallel_input`'s one bit.

There are also some intermediate signals use to control the register operation. One such signal is the **temp: std_logic_vector** which is stores the temporary state of the `parallel_input`(reversed, otherwise we get and LSB first output) after each bit is transferred. For eg. if `parallel_input` is 11001011 then the state of temp signal after each iteration would be:

c	temp	serial_output
1	01101001	1
2	00110100	1
3	00011010	0
4	00001101	0
5	00000110	1
6	00000011	0
7	00000001	1
8	00000000	1

where **c** : is another variable used to maintain the above count. When no output is to be transmitted, the `serial_output` is set to Z-high impedance. Following timing diagram shows the functionality of the shift-register for input 11001101.



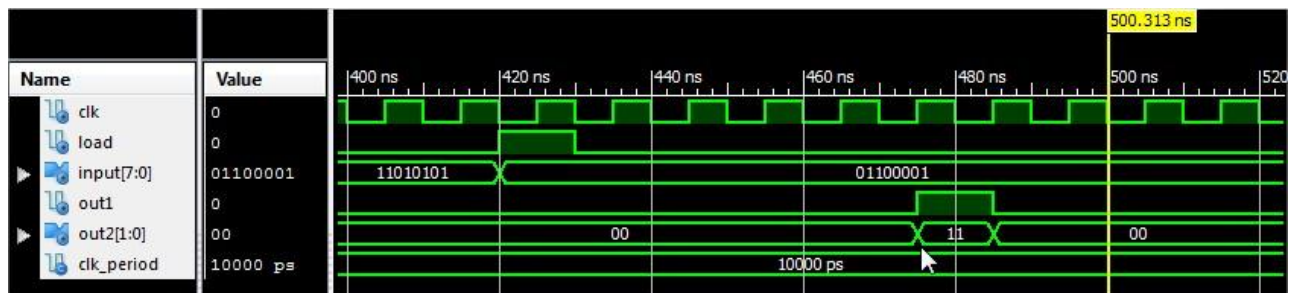
Results captured

A single input bit is parsed at the falling edge of the clock by the shift-register. out1 goes high at the first rising edge of the clock when a sequence is detected. out2 shows the encoding of sequence detected.

Sequence detected: 100100 – encoded 01



Sequence detected: 1100 – encoded 11



Sequence detected: 110101 – encoded 10

