EXPERIMENT O7
FINITE STATE MACHINE DESIGN

MANCHESTER ENCODING

For each input bit, two bits need to be extracted in order to design a *Manchester Encoder*. If the input bit is *1*, the output *ordered pair of bits* should *10* and if the input bit is *0*, the output *ordered pair of bits* should *01*. Note that in this design we will be using a *D Flip Flop* that works at the *negative/falling edge*. This means that a new bit from the input stream (*synchronized* with the clock connected to the DFF) is taken only when the clock strikes *0*. No input from the stream is taken when the clock pulse is in the *high half of the clock cycle*. This characteristic can be exploited to utilize the *idle high half* of the clock cycle and design a *manchester encoder*.

We could have simply output the two bits for each input bit together. But we want the *manchester encoding* of the *input binary stream* to also be output as a *binary stream*. In other words, the two output bits for any input bit should not be output together, but one after the other, like a pulse. So we can utilize the *high half* (when the clock is showing 1) to yield the second bit in the 2-bit output, while the first bit is yielded in the *low half* (when the clock is showing 0) itself when a new bit is taken from the input stream.

INPUT (D)	*CLOCK (C)*	*EXPECTED OUTPUT (O)*
0	*0*	*0*
	1	*1*
1	*0*	*1*
	1	*0*

Notice one thing that here the output stream is varying at a pace twice as that of the input stream. We need to remember only one bit for the *high half* of the clock cycle to compute the output bit. Therefore, we will design an *FSM* with *only one state*. Consequently, only one *DFF* is enough for this design.

If the table is looked at closely, it is nothing but the *truth table* of *XOR* operation. Therefore, the *output function* for the *FSM* would be

$$O = C \oplus D$$

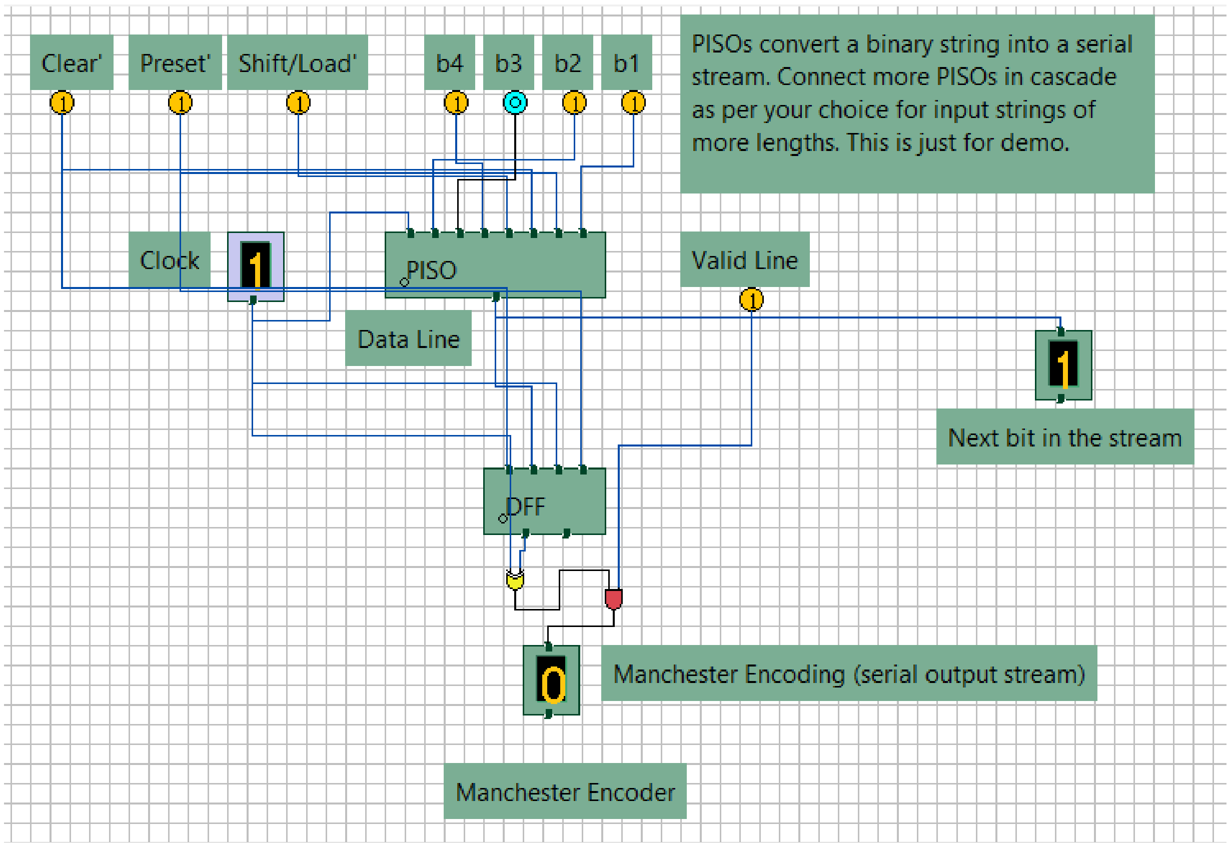
The *next state function* for the *FSM* is very straight forward because there is only one state. So, in any transition, the current configuration of the machine will always be at the *only state S* of the machine and naturally this would also be the *start state*. Therefore, the *next state function* is

$$S_{next} = S$$

We also want to implement an *enabler* or *valid* line/input to decide if the encoder should output the *manchester encoding* of the input stream or it should output *nothing* (a stream of *0*s). Another input *V* is used for this matter. If *V* is *1*, the data is valid and the encoder should output the manchester encoding and otherwise it should output a stream of zeroes. So the *output function* can be modified as follows.

$$O = (C \oplus D) \,.\, V$$

These are the overall semantics of the *finite state machine* that we have designed for *a Manchester Encoder*. The following circuit implements this design.


Clear'
Preset'
Shift/Load'
b4
b3
b2
b1
PISOs convert a binary string into a serial stream. Connect more PISOs in cascade as per your choice for input strings of more lengths. This is just for demo.
Clock
1
PISO
Valid Line
Data Line
1
Next bit in the stream
DFF
0
Manchester Encoding (serial output stream)
Manchester Encoder

TRAFFIC LIGHT CONTROLLER

The *traffic light controller* has *3 inputs (H, V* and *T)* and therefore *8 input strings* are possible for the machine. The main activities or semantics that we need at any point to decide which transition should take place include -- *whether the horizontal or vertical traffic was passed in the last instance* and *whether or not the timeout was active in the last instance*. Therefore, the states must have *two bits* of information, one bit to remember each of them.

Bit	*Convention*
P	If *0* then the *horizontal traffic* is passed. If *1* then *vertical traffic* is passed.
Q	If *0* then the *timeout* is active. If *1* then *timeout* is inactive.

Since a *state* is encoded as a *two bit binary string*, there are *4 states* in the *finite state machine* we are designing.

P	*Q*	*State*
0	0	00 (*S1*)
0	1	01 (*S2*)
1	0	10 (*S3*)
1	1	11 (*S4*)

Given the conditions in the problem statement, let us make a *next state transition table*.

PS	*NS (given the input HVT)*							
	000	***001***	***011***	***010***	***110***	***111***	***101***	***100***
00	00	00	11	11	11	11	01	01
01	00	00	01	01	01	01	01	01
11	10	10	11	11	11	11	11	11
10	10	10	11	11	01	01	01	01

Since the state comprises of two bits, two functions will have to be given to define each of the $P_{next}$and Q_{next}. Let us consider one bit at a time in the above *table* and convert it into a *Karnaugh map* that can be reduced into a compact function to define the value for that bit associated with the next state, depending on the inputs and the current state.

PS	P_{next}*(given the input HVT)*							
	000	***001***	***011***	***010***	***110***	***111***	***101***	***100***
00	0	0	1	1	1	1	0	0
01	0	0	0	0	0	0	0	0

11	1	1	1	1	1	1	1	1
10	1	1	1	1	0	0	0	0

HVT PQ	000	001	011	010	110	111	101	100
00	0	1	**3**	**2**	**6**	**7**	5	4
01	8	9	11	10	14	15	13	12
11	**24**	**25**	**27**	**26**	**30**	**31**	**29**	**28**
10	**16**	**17**	**19**	**18**	22	23	21	20

Therefore the function for the value *P* in the *next state* can be given by

$$P_{next} = P.Q + H'P + VP'Q'$$

PS	Q_{next}*(given the input HVT)*							
	000	***001***	***011***	***010***	***110***	***111***	***101***	***100***
00	0	0	1	1	1	1	1	1
01	0	0	1	1	1	1	1	1
11	0	0	1	1	1	1	1	1
10	0	0	1	1	1	1	1	1

HVT PQ	000	001	011	010	110	111	101	100
00	0	1	**3**	**2**	**6**	**7**	**5**	**4**
01	8	9	**11**	**10**	**14**	**15**	**13**	**12**
11	24	25	**27**	**26**	**30**	**31**	**29**	**28**

10	16	17	**19**	**18**	**22**	**23**	**21**	**20**

Therefore the function for the value Q in the *next state* can be given by

$$Q_{next} = (H'.V')' = H + V$$

These two functions together define the *next state function*.

$P_{next} = PQ + H'P + VP'Q'$

$Q_{next} = (H'.V')' = H + V$

Let us verify the correctness of the *next state function* by checking if all the conditions in the problem statement are satisfied.

- Timeout is activated after a pass signal is given.
  Giving a *pass signal* means at least one of the inputs *H* and *V* are *true*. In this case $Q_{next} = H + V = 1$and hence the *timeout* has been activated in the next state.
  Hence verified.
- If there's no traffic after a timeout, then timeout is not active.
  No traffic means values of both *H* and *V* is *false*. In this case $Q_{next} = H + V = 0 + 0 = 0$and hence the *timeout* is de-activated in the next state.
  Hence verified.
- The pass signal does't change when timeout is active.
  Active timeout means Q is *1*. Therefore we have
  $P_{next} = P.Q + H'P + VP'Q' = P.1 + H'P + VP'0 = P + H'P = P$
  This implies that the value of *P* does not change if the *timeout* is active. Therefore, the *pass signal* does not change.
  Hence verified.
- If only horizontal or vertical traffic is present when timeout is inactive, that should be passed and timeout activated.
  Inactive timeout means Q is 0. Therefore we have
  $P_{next} = P.Q + H'P + VP'Q' = P.0 + H'P + VP'1 = H'P + VP'$

If only horizontal traffic is present (*H=1 and V=0*) then $P_{next} = 0$ and therefore *horizontal traffic is passed.* Also $Q_{next} = H + V = 1 + 0 = 1$ that means the *timeout is activated*.
Similarly, if only vertical traffic is present (*H=0 and V=1*) then $P_{next} = P + P' = 1$ and therefore *vertical traffic is passed.* Also $Q_{next} = H + V = 0 + 1 = 1$ that means the *timeout is activated*.
Hence verified.

- If there's no traffic, the current pass signal is to be maintained.
  No traffic means values of both *H* and *V* is *false*. In this case $P_{next} = P.Q + H'P + VP'Q' = P.Q + 1.P + 0.P'Q' = P.Q + P = P$
  This implies that the value of *P* does not change if the *there's no traffic*. In other words, *the current pass signal is maintained*.
  Hence verified.
- If both traffic are present when timeout is inactive, the one that was not favoured last time should be favoured this time and timeout should be activated
  Inactive timeout means Q is 0. Therefore we have
  $P_{next} = P.Q + H'P + VP'Q' = P.0 + H'P + VP'1 = H'P + VP'$
  If both traffic is present then *H=1 and V=1*. Therefore,
  $P_{next} = H'P + VP' = 0.P + 1.P' = P'$
  This implies that the value of *P* is flipped in the next state, that is if in the present state *P* is 1 (vertical traffic is passing) then in the next state *P* is 0 (now horizontal traffic is favoured); and vice-versa.
  Hence verified.

So the *next state function* of the *FSM* is designed (and also validated). Now the *output function* has to be designed. There are *3 bits* in the output.

Bit	*Convention*
PH	If *1* then the *horizontal traffic* is blocking *vertical traffic*, and not otherwise
PV	If *1* then the *vertical traffic* is blocking *horizontal traffic*, and not otherwise

ST	If *1* then the *timer is activated*

PH output will be *1 if and only if* the *horizontal traffic is passing* (*P* bit in the next state is *0*) and the *vertical traffic is present* (*V* input bit is 1).
If the horizontal traffic is passing and the vertical traffic is not present (*V* input bit is 0), then *no blocking is happening* because there is no traffic to be blocked. Therefore,

$$PH = P_{next}^{'} . V = (P.Q + H'P + VP'Q')'.V = (P' + Q').(H + P').(V' + P + Q).V$$

PV output will be *1 if and only if* the *vertical traffic is passing* (*P* bit in the next state is *1*) and the *horizontal traffic is present* (*H* input bit is 1).
If the vertical traffic is passing and the horizontal traffic is not present (*H* input bit is 0), then *no blocking is happening* because there is no traffic to be blocked. Therefore,

$$PV = P_{next} . H = (P.Q + H'P + VP'Q').H = PQH + VHP'Q'$$

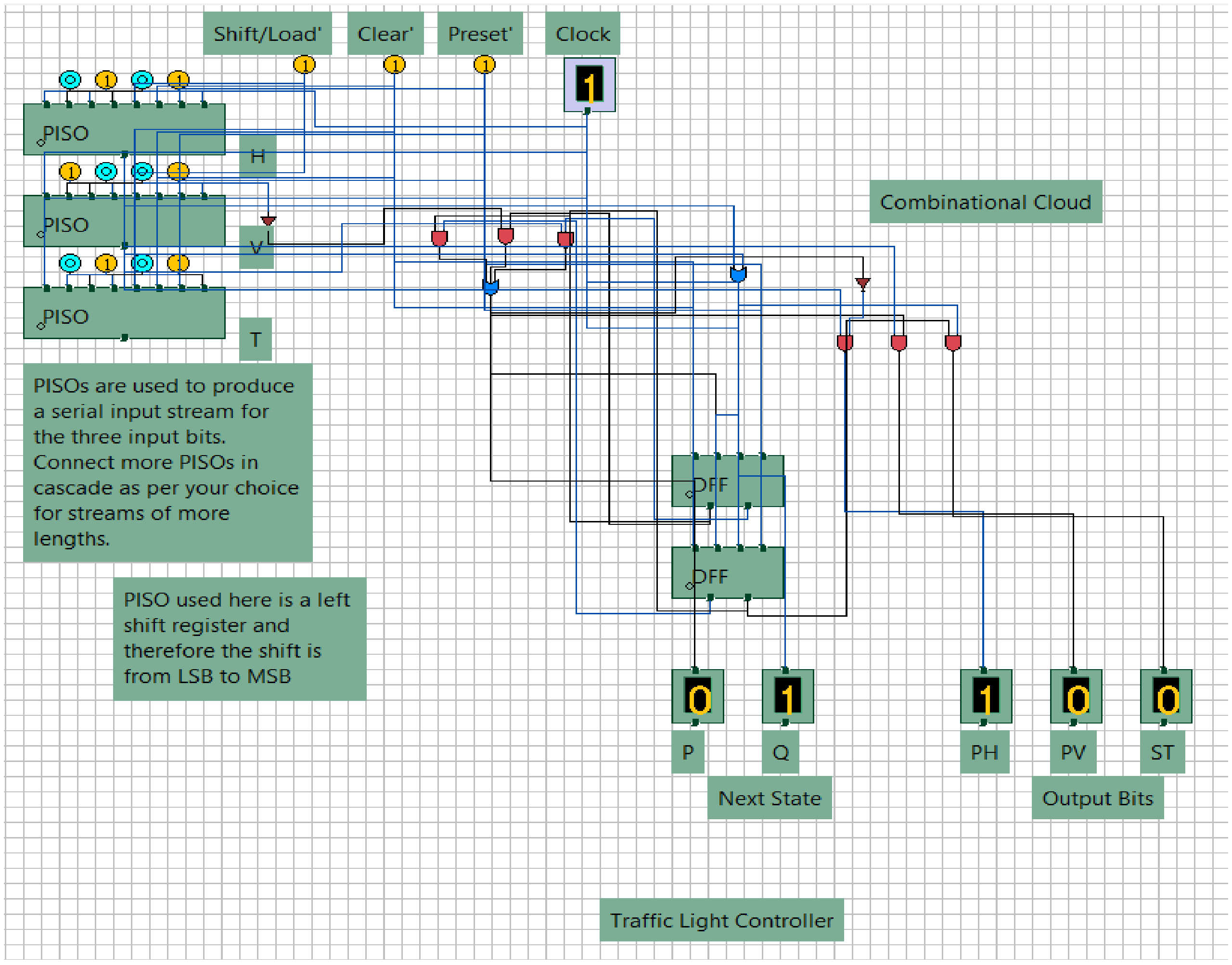
ST output is *1* if and only if in the *present state timer is inactive* (*Q is 0*) and in the *next state it becomes active* (Q_{next} *is 1*). In this case, if the timer was already active in the present state, then it is not needed to be activated and hence *ST* will be *0*.

$$ST = Q'.Q_{next} = Q'.(H + V) = Q'H + Q'V$$

These three functions together define the *output functions*.

$$PH = (P' + Q').(H + P').(V' + P + Q).V$$
$$PV = PQH + VHP'Q'$$
$$ST = Q'.(H + V) = Q'H + Q'V$$

Now the design of the *finite state machine* is complete.
These are the overall semantics of the *finite state machine* that we have designed for *a Traffic Light Controller*. The following circuit implements this design.



PISOs are used to produce a serial input stream for the three input bits. Connect more PISOs in cascade as per your choice for streams of more lengths.

PISO used here is a left shift register and therefore the shift is from LSB to MSB