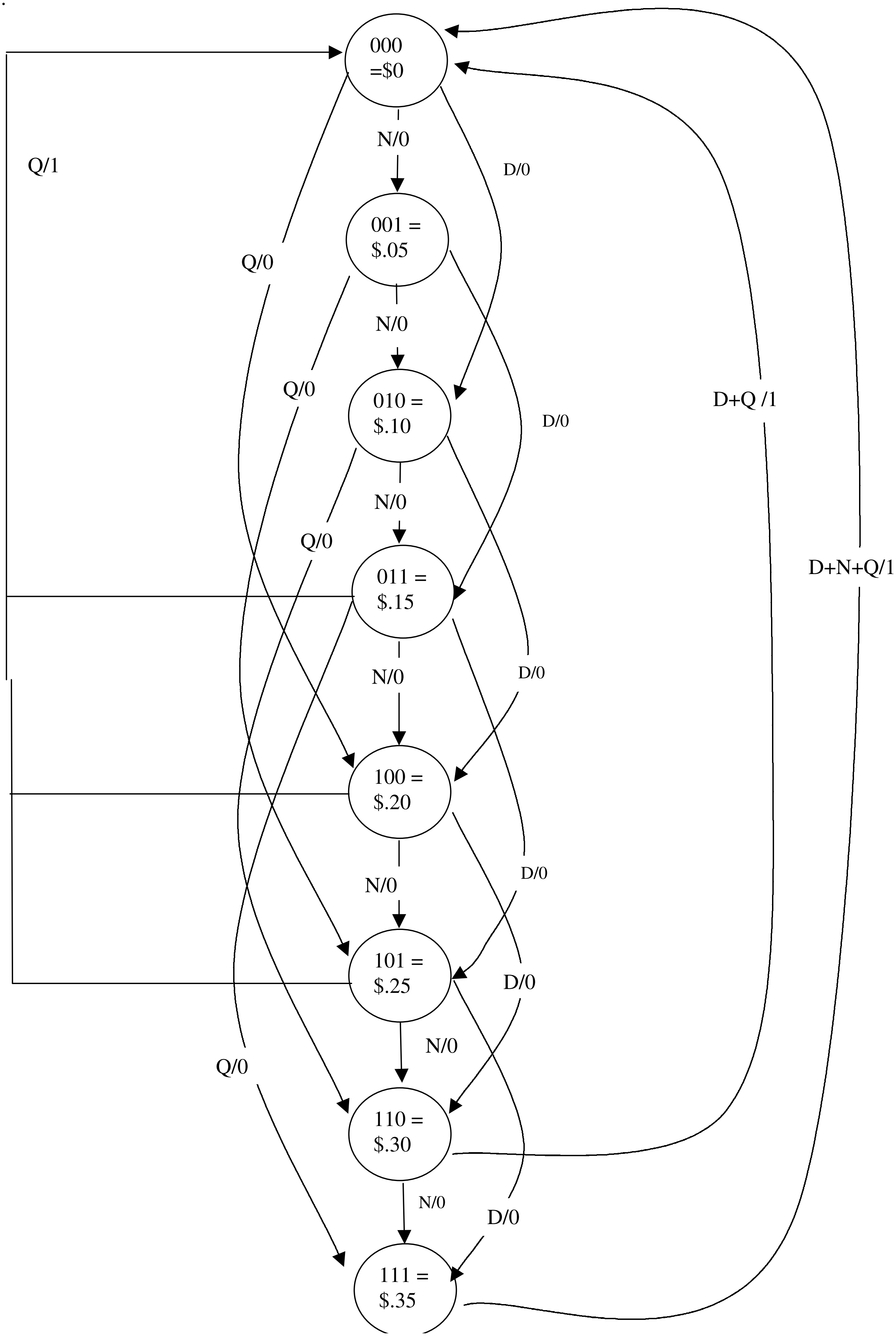
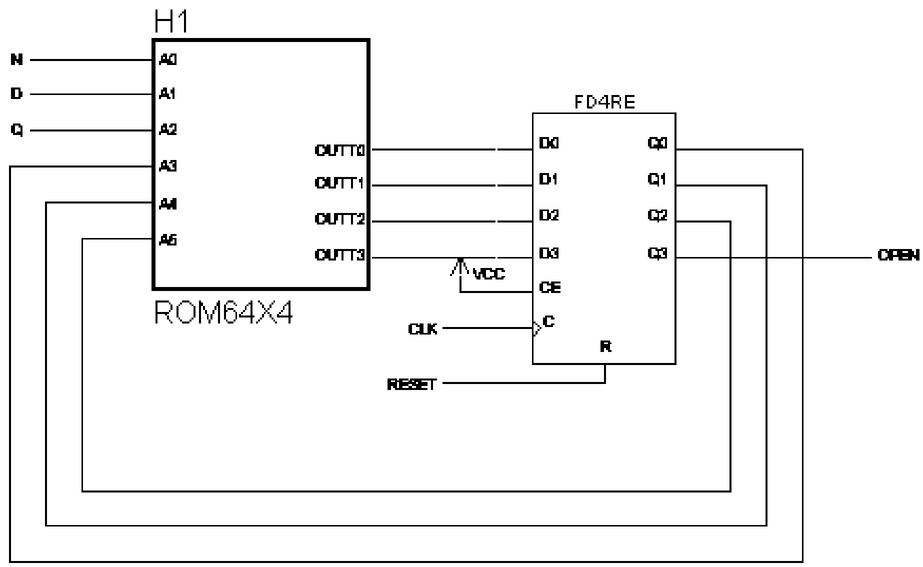
1.


000
=$0
001 =
$.05
010 =
$.10
011 =
$.15
100 =
$.20
101 =
$.25
110 =
$.30
111 =
$.35
N/0
N/0
N/0
N/0
N/0
N/0
N/0
D/0
D/0
D/0
D/0
D/0
D/0
Q/0
Q/0
Q/0
Q/0
Q/1
D+Q /1
D+N+Q/1

Q2	Q1	Q0	Q	D	N	OPEN	Q2+	Q1+	Q0+	Hex
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	2
0	0	0	1	0	0	0	1	0	1	5
0	0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	1	0	0	1	0	2
0	0	1	0	1	0	0	0	1	1	3
0	0	1	1	0	0	0	1	1	0	6
0	1	0	0	0	0	0	0	1	0	2
0	1	0	0	0	1	0	0	1	1	3
0	1	0	0	1	0	0	1	0	0	4
0	1	0	1	0	0	0	1	1	1	7
0	1	1	0	0	0	0	0	1	1	3
0	1	1	0	0	1	0	1	0	0	4
0	1	1	0	1	0	0	1	0	1	5
0	1	1	1	0	0	1	0	0	0	8
1	0	0	0	0	0	0	1	0	0	4
1	0	0	0	0	1	0	1	0	1	5
1	0	0	0	1	0	0	1	1	0	6
1	0	0	1	0	0	1	0	0	0	8
1	0	1	0	0	0	0	1	0	1	5
1	0	1	0	0	1	0	1	1	0	6
1	0	1	0	1	0	0	1	1	1	7
1	0	1	1	0	0	1	0	0	0	8
1	1	0	0	0	0	0	1	1	0	6
1	1	0	0	0	1	0	1	1	1	7
1	1	0	0	1	0	1	0	0	0	8
1	1	0	1	0	0	1	0	0	0	8
1	1	1	0	0	0	0	1	1	1	7
1	1	1	0	0	1	1	0	0	0	8
1	1	1	0	1	0	1	0	0	0	8
1	1	1	1	0	0	1	0	0	0	8

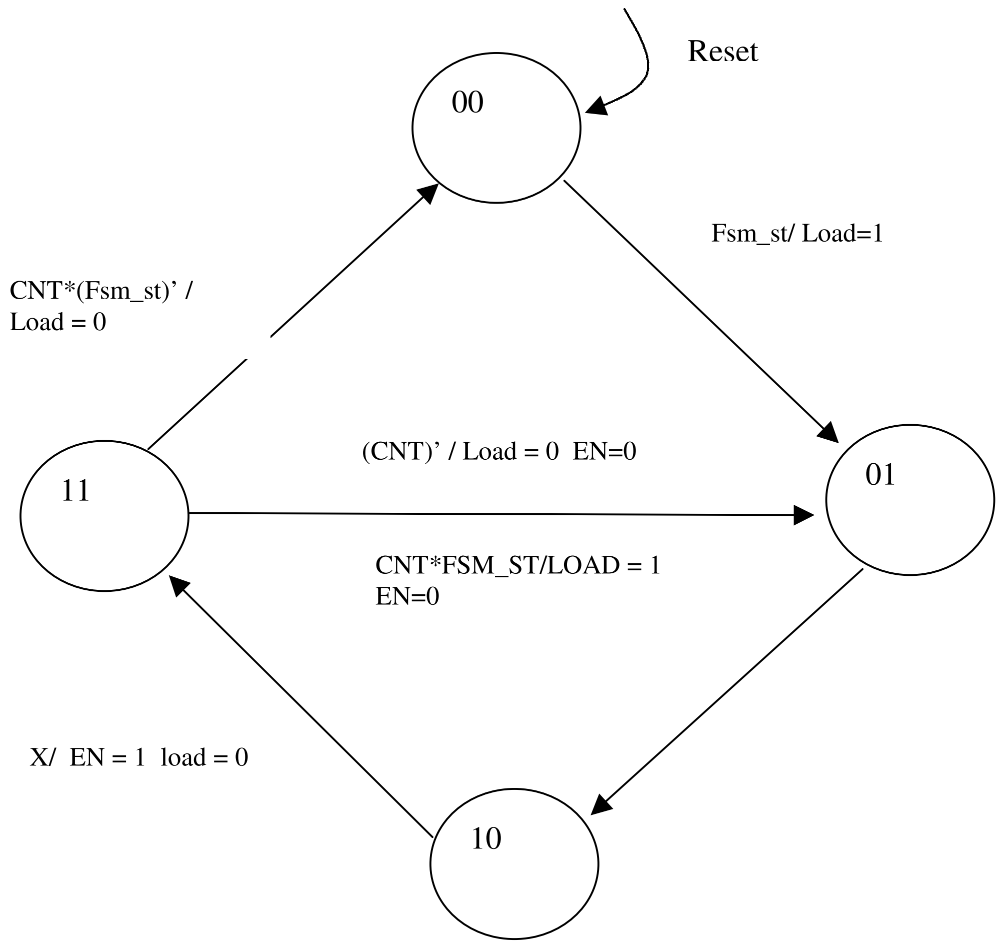
ROM contents, address 0…63 : 0,1,2,0,5,0,0,0,1,2,3,0,6,0,0,0,
2,3,4,0,7,0,0,0,3,4,5,0,8,0,0,0,
4,5,6,0,8,0,0,0,5,6,7,0,8,0,0,0,
6,7,8,0,8,0,0,0,7,8,8,0,8,0,0,0

We can simply make the data for all not listed addresses 0x0. This will send us back to the starting state if we get to an invalid state somehow. Also, this state machine assumes only one of N,D,and Q can be asserted at one time.



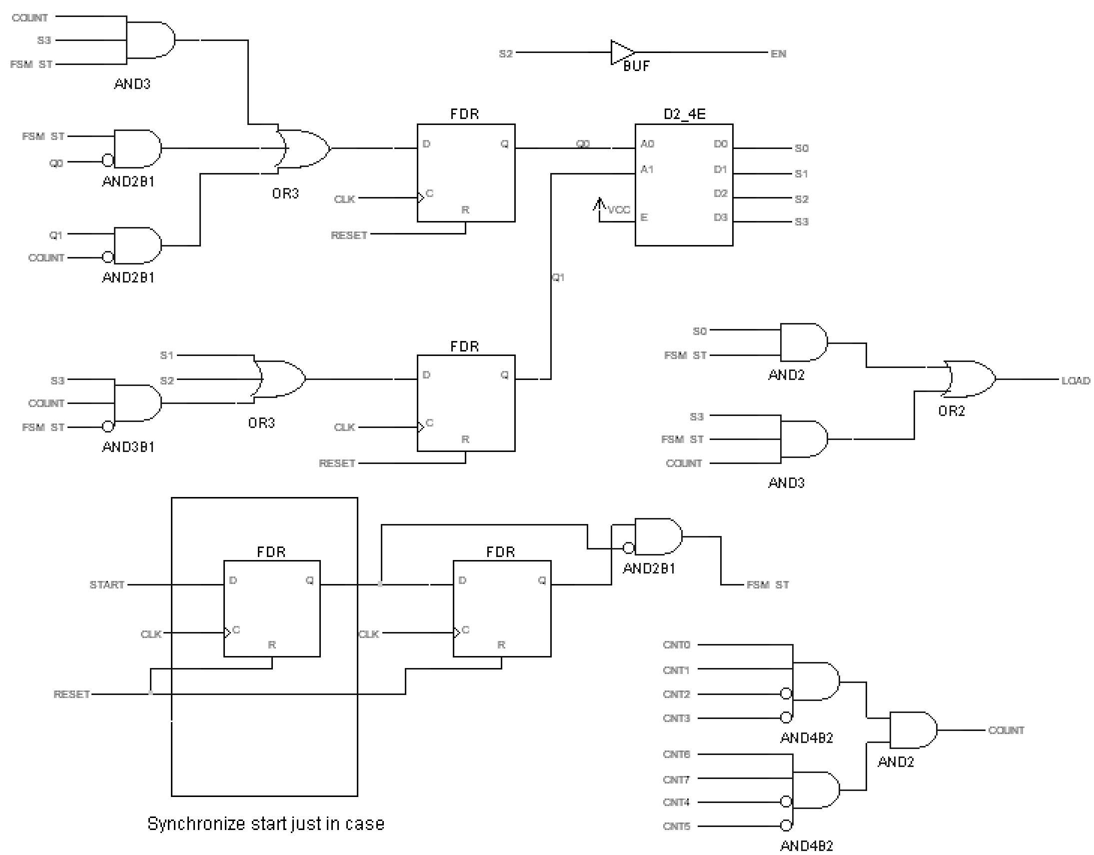
b. 2^6 x 4 ROM means we need 2^6 x 4 bits for storage which is equal to 2^8= 256 bits. That takes up about 4 CLBs which is 2% of the XC 4005E.

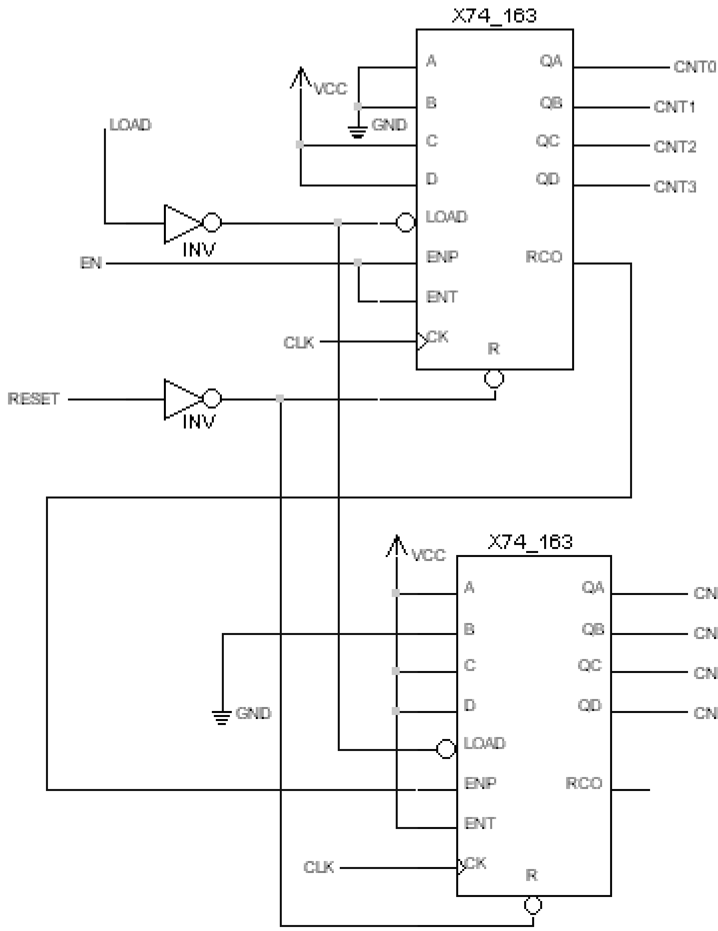
2.



Q1	Q0	Fsm_st	CNT		EN	Load	Q1+	Q0+
0	0	0	X		0	0	0	0
0	0	1	X		0	1	0	1
0	1	X	X		0	0	1	0
1	0	X	X		1	0	1	1
1	1	0	1		0	0	0	0
1	1	X	0		0	0	0	1
1	1	1	1		0	1	0	1

Q1+ = Q1'*Q0 + Q1*Q0'
Q0+ = Q0'*FSM_ST + Q1*CNT' + Q1*Q0*FSM_ST*CNT
LOAD = Q1'*Q0'*FSM_ST + Q1*Q0*FSM_ST*CNT   EN = Q1*Q0'
In the above FSM_start will be a 1 for one clock cycle after the falling edge of Start, and 0 otherwise. This will be done with a falling edge detect circuit.

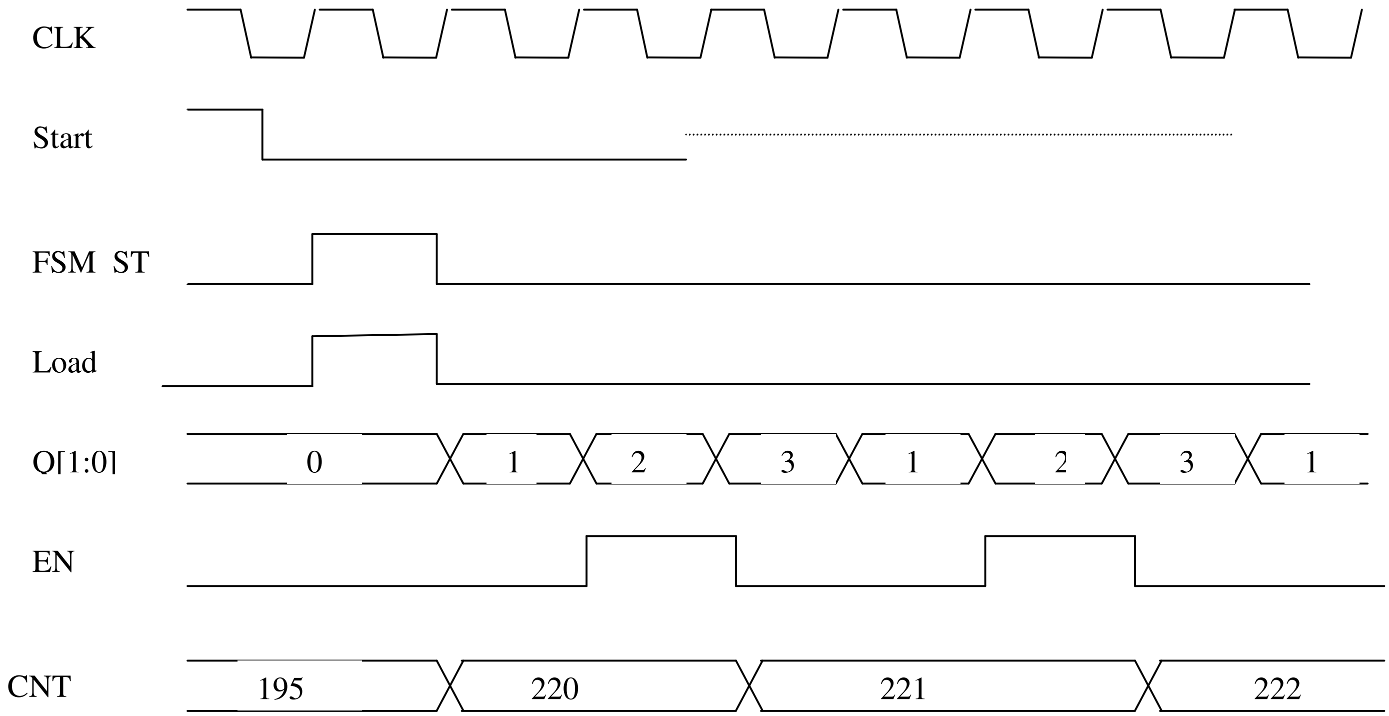

COUNT
S3
FSM ST
AND3
S2
BUF
EN
FDR
D2_4E
FSM ST
Q0
AND2B1
OR3
D
Q
C
R
CLK
RESET
A0
A1
E
VCC
D0
D1
D2
D3
S0
S1
S2
S3
Q1
COUNT
AND2B1
Q1
S1
S2
S3
COUNT
FSM ST
AND3B1
OR3
FDR
CLK
RESET
S0
FSM ST
AND2
OR2
LOAD
S3
FSM ST
COUNT
AND3
FDR
FDR
AND2B1
START
CLK
CLK
RESET
FSM ST
CNT0
CNT1
CNT2
CNT3
AND4B2
AND2
COUNT
CNT6
CNT7
CNT4
CNT5
AND4B2
Synchronize start just in case



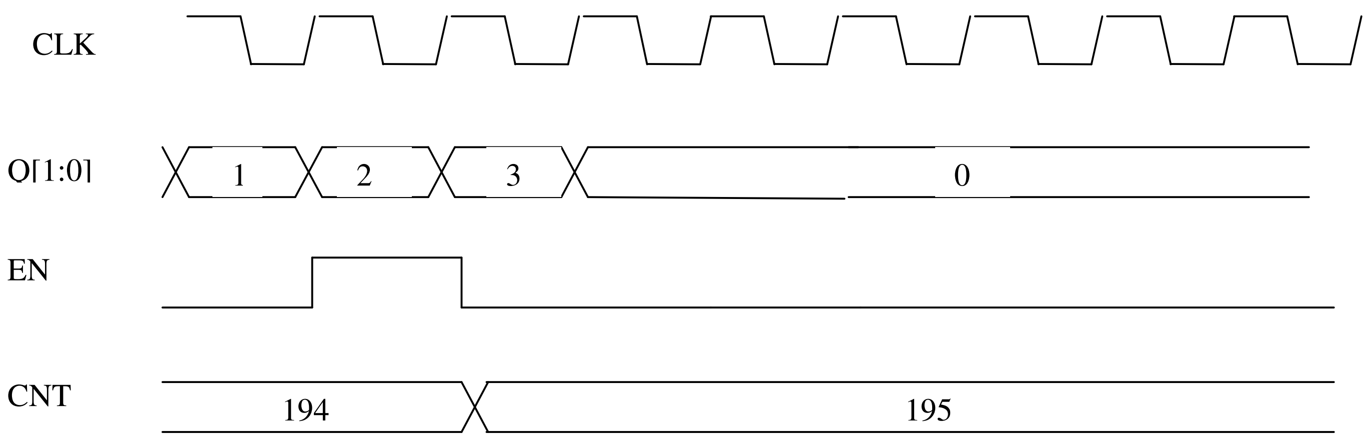
220d = 11011100b

195d = 11000011b

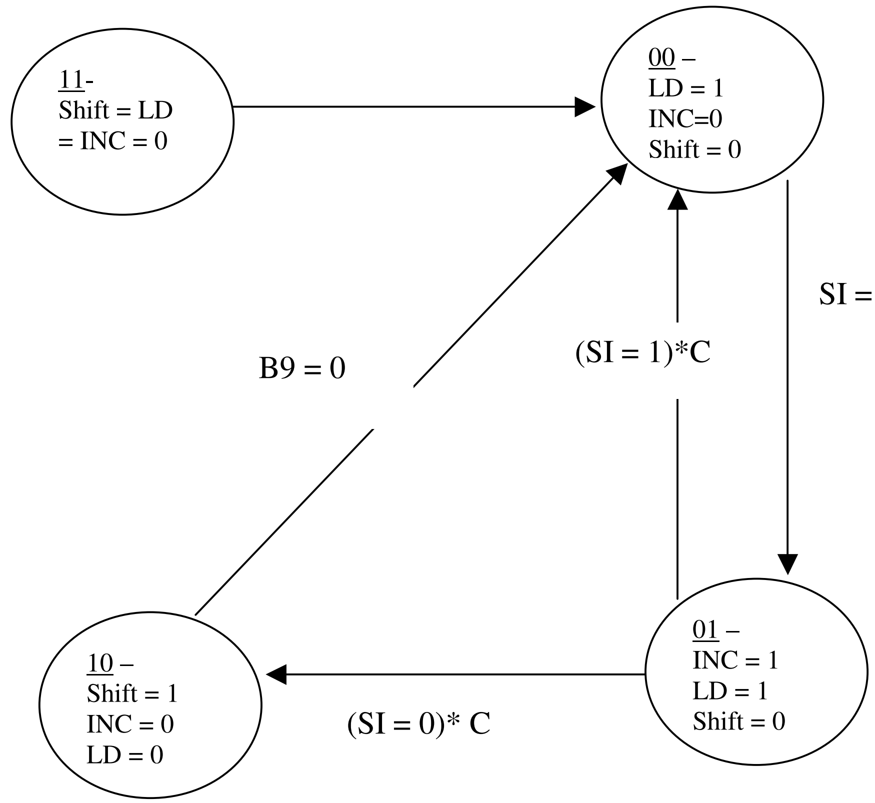
Beginning of count:



End of count:



3.



Q1+	Q0+	SI	B9	C		LD	INC	Shift	Q1+	Q0+
0	0	1	X	X		1	0	0	0	0
0	0	0	X	X		1	0	0	0	1
0	1	X	X	0		1	1	0	0	1
0	1	1	X	1		1	1	0	0	0
0	1	0	X	1		1	1	0	1	0
1	0	X	1	X		0	0	1	1	0
1	0	X	0	X		0	0	1	0	0
1	1	X	X	X		X	X	X	0	0

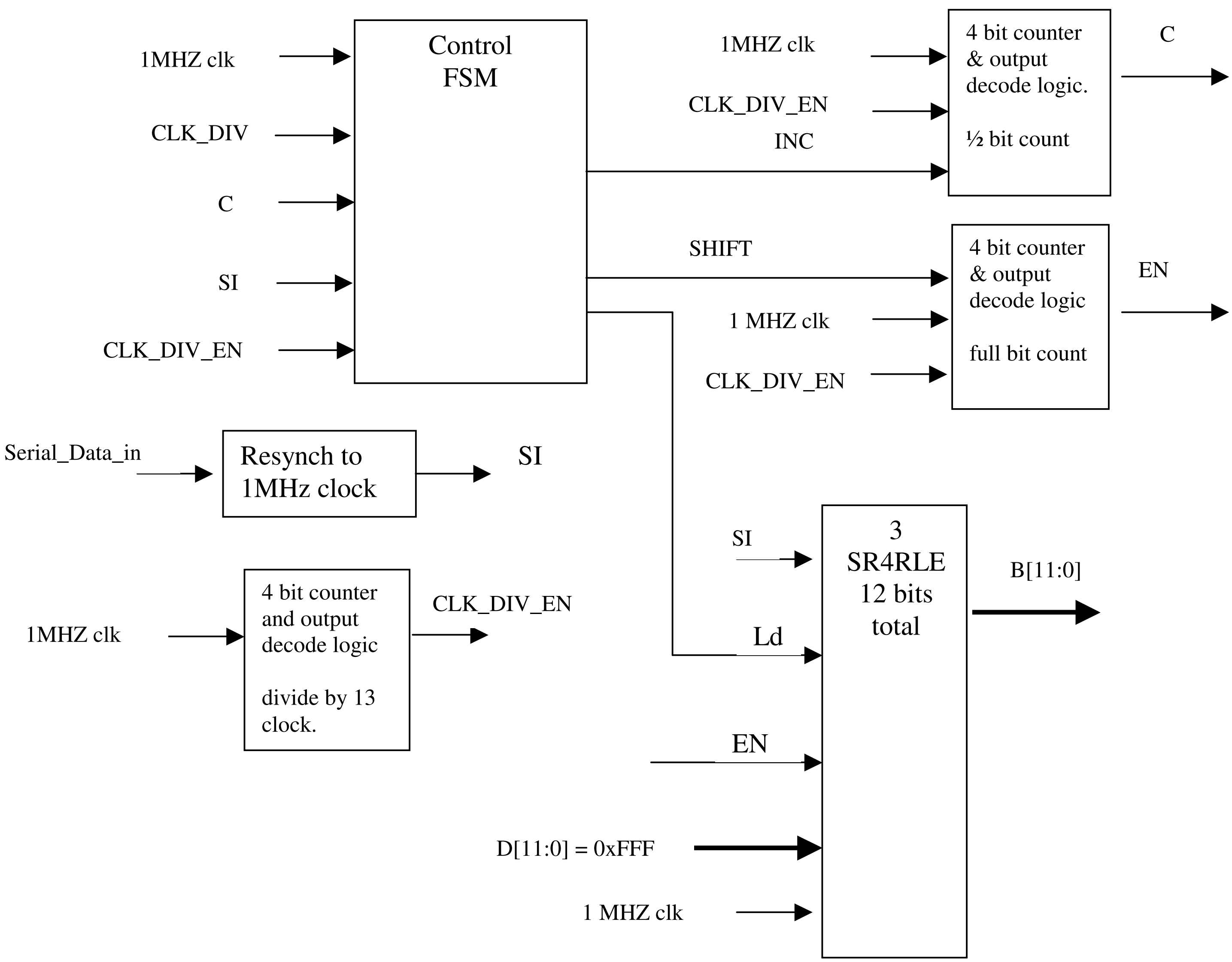
In the above we are letting C = (CNT = 3) , SI = Serial_Data_In_resynced, and B9 is bit 9 of the data word stored in the shift register (should be the start bit when all bits have been shifted in). Note that we divide the 1MHz clock by 13 and then use the method suggested in the problem. This creates a clock enable called CLK_DIV_EN which causes sampling at the centers of the data bits. We also use this clock_enable for all components in the circuit (FSM registers, counters and shift registers).

Also, there is an EN signal which enables the shift register in state 10. SH_CNT counts a full bit width and then asserts EN. So, EN = (SH_CNT0+SH_CNT1+SH_CNT2)'. The SH_CNT is only incremented in state 10.

Q1+= Q1'*Q0*SI'*C + Q1*Q0'*B9 Q0+ = Q1'*Q0'*SI' + Q1'*Q0*C'

LD = Q1' INC = Q1'*Q0 SHIFT = Q1*Q0'

Block Diagram:
(Note: each block will also have a global reset connection, not shown).



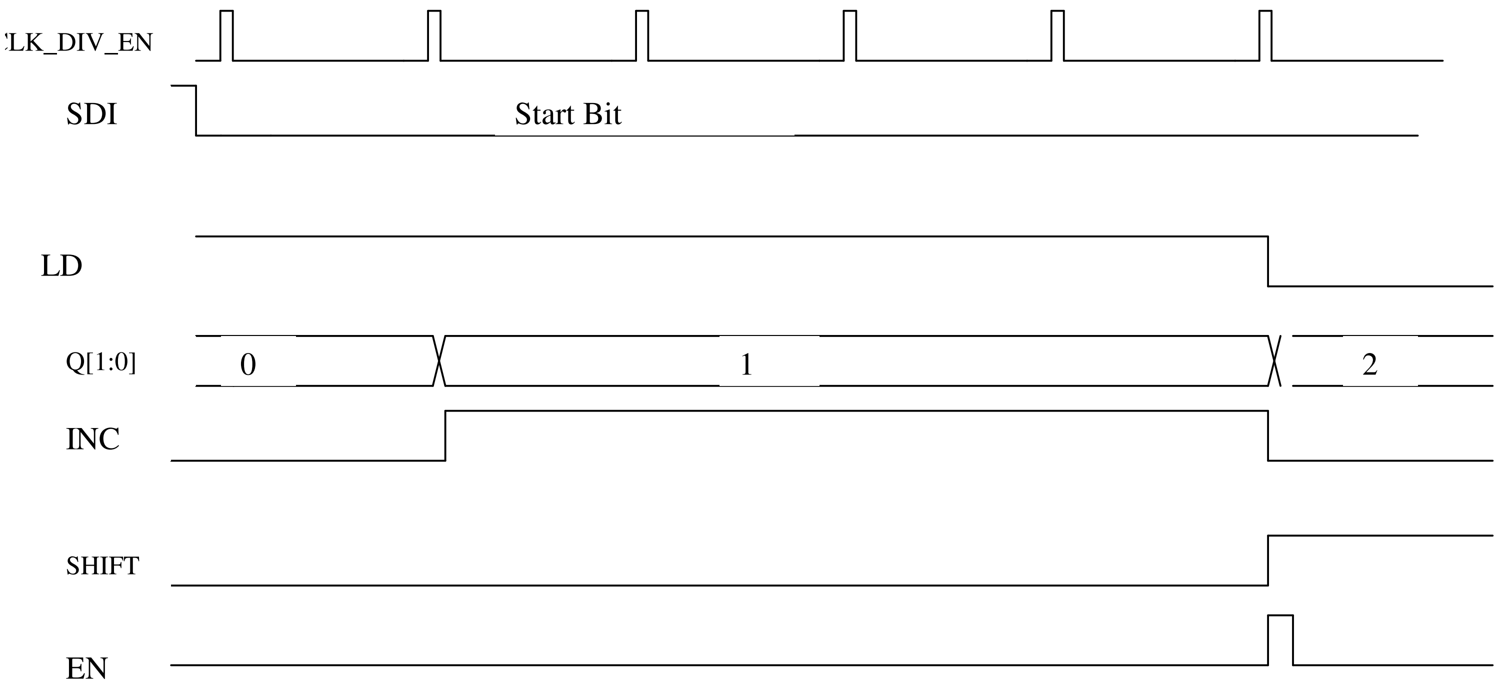
For outputs:

Data out: D[7:0] = B[8:1]

New_Data_Ready = B[9]'

In these waveforms only the clock enable (CLK_DIV) to the circuit is shown, not the clock. The 1 MHZ clock is continuously running and the FSM only changes state when CLK_DIV =1. So we can leave the 1MHz clock out of the waveform without loss of information, as long as we recognize there is one rising edge of the 1MHz clock each time that CLK_DIV is high.

The waveform for the start bit (beginning of data receiving):



End of data (stop bit) :