Asynchronous 4-bit Modulo-*n* Counter using SEQUEL

INTEGRATED CIRCUITS LAB

Project Report

Sambhav R Jain - 107108103

Vangapandu Srivijay - 107108078

Kommanaboina Pramod - 107108060

The use of SEQUEL (A Solver for circuit EQuations with User-defined ELements) to implement an asynchronous 4-bit MOD-*n* counter is shown. The design employs JK flip-flops, multiplexers and de-multiplexers along with the necessary logic gates to reset the counter when required. The counter is thus able to produce a sequence from 0 to any value up to 16.

TITLE: ***ASYNCHRONOUS 4-BIT MODULO-n COUNTER***

OBJECTIVE:

To construct an asynchronous 4 bit modulo-*n* counter using SEQUEL and to verify its output as anticipated.

COMPONENTS USED:

Serial No.	Name	Type	Quantity
1.	JK flip-flop	negative edge triggered	4
2.	Multiplexer	4×1	4
3.	De-multiplexer	1×4	1
3.	AND gate	2-input	17
4.	OR gate	2-input	3
5.	Clock	25 kHz	1

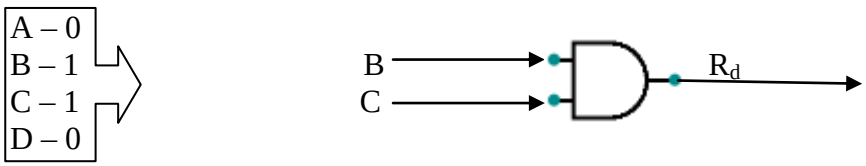
PRINCIPLE OF OPERATION:

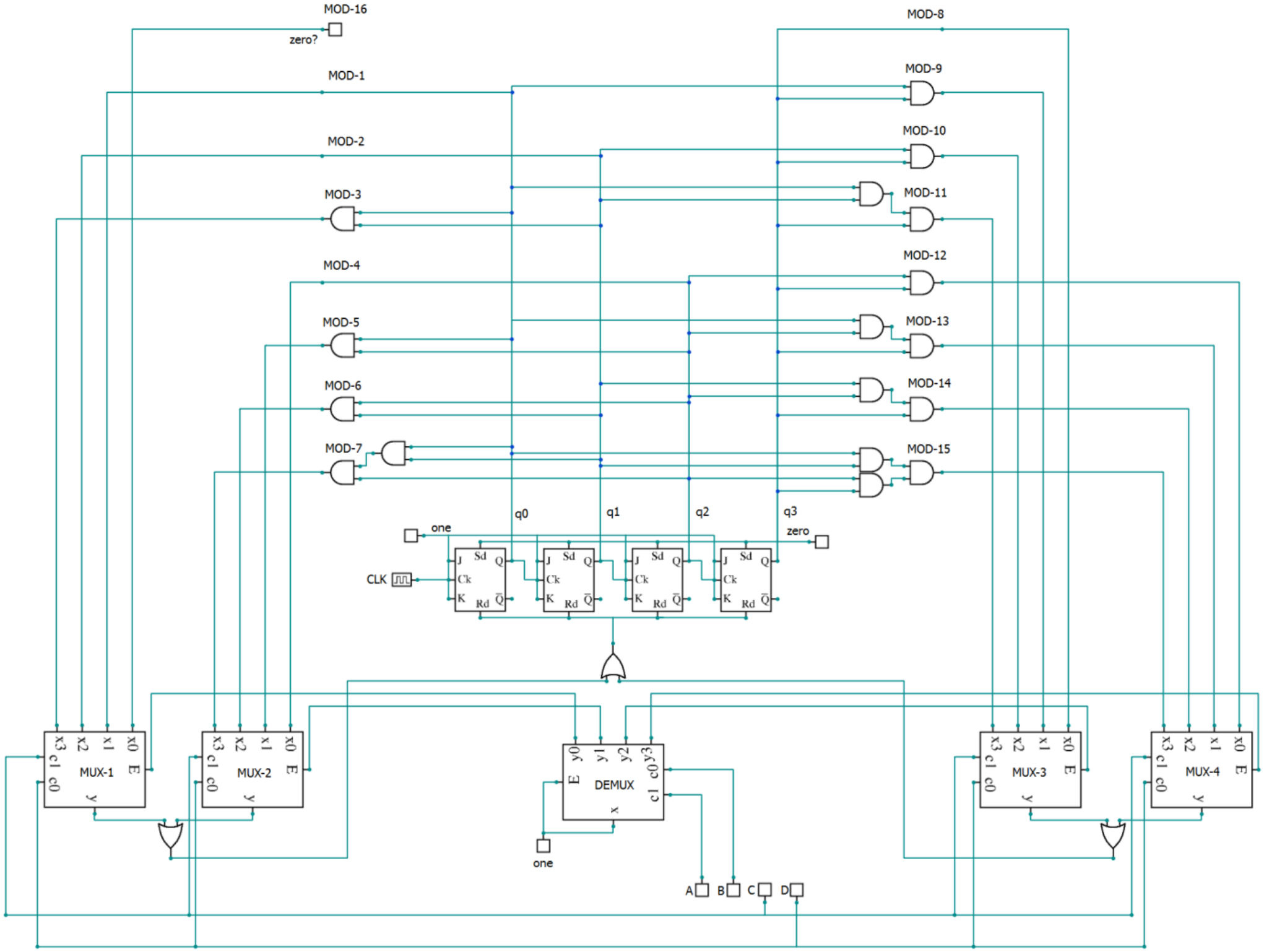
Asynchronous counters are made up of a series of flip-flops, each clocked by the previous state, one after the other. Since all the stages of the counter are not clocked simultaneously, a ripple effect propagates as various flip-flops are clocked. For this reason they are called ripple counters.

A *k*-bit counter can count up to 2^k clock cycles. However, by using the direct-reset (R_d) pin to clear the flip flop at any intermediate cycle, the same *k*-bit counter can be utilized to count to any value within 2^k. The modulus of a counter is the number of different output states the counter may take; e.g. MOD-6 means that the counter has six output states.

E.g. A 4-bit counter can count 16 clock cycles. But if it is expected to construct a MOD-6 counter, then the output should repeat 0000 till 0101. Thus as soon as the next stage viz. 0110 is achieved, combinational logic is used to set the R_d to HIGH, which would reset the counter to repeat its next sequence. This way, it would only have six output states.

To generalize the circuit, the user can be made to choose the required modulus value '*n*' by specifying its binary equivalent; e.g. for a MOD-6, enter:




MOD-16
zero?
MOD-1
MOD-2
MOD-3
MOD-4
MOD-5
MOD-6
MOD-7
MOD-8
MOD-9
MOD-10
MOD-11
MOD-12
MOD-13
MOD-14
MOD-15
q0
q1
q2
q3
one
zero
CLK
J
Sd
Q
Ck
K
Rd
MUX-1
MUX-2
MUX-3
MUX-4
DEMUX
A
B
C
D

CIRCUIT OPERATION:

In our circuit, depending upon the binary input A-B-C-D being given, the counter would count up to one less than the decimal equivalent of the binary input; i.e. if a MOD-6 counter is required, the binary input to the circuit should be 0110.

The first and second most significant bits of the input (i.e. A and B) are fed to the select lines of the DEMUX whose outputs are coupled to the ENABLE pins of each of the four MUX used. Thus if the inputs A and B are as shown then the MUX controls are as below.

A	B	MUX	*n*
0	0	MUX-1	16,1,2,3
0	1	MUX-2	4,5,6,7
1	0	MUX-3	8,9,10,11
1	1	MUX-4	12,13,14,15

The bits C and D help the MUX to select the value of *n*.

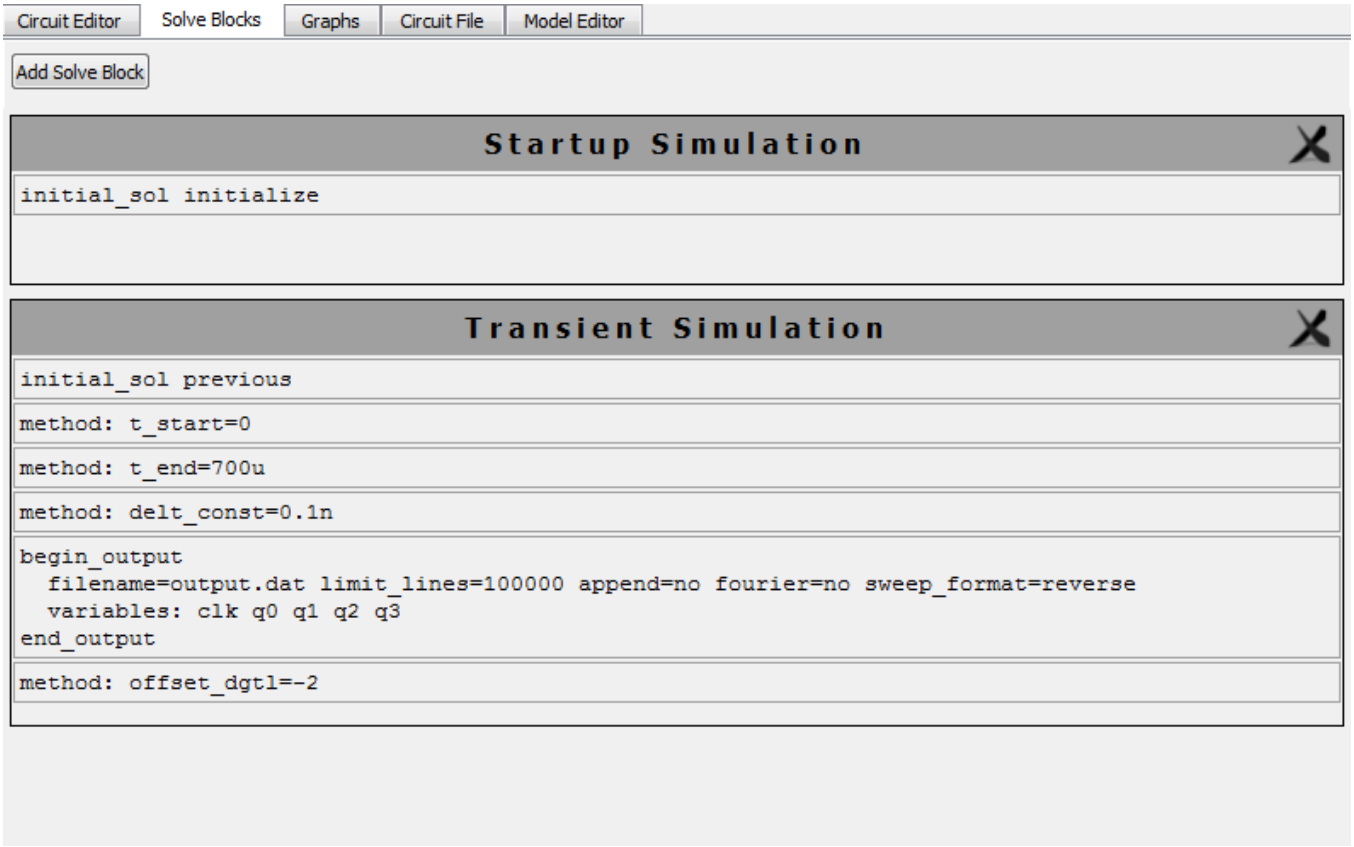
C	D	Modulus (*n*)			
		MUX-1	MUX-2	MUX-3	MUX-4
0	0	16	4	8	12
0	1	1	5	9	13
1	0	2	6	10	14
1	1	3	7	11	15

The following combinational logic is employed to reset the flip flops at the required cycle.

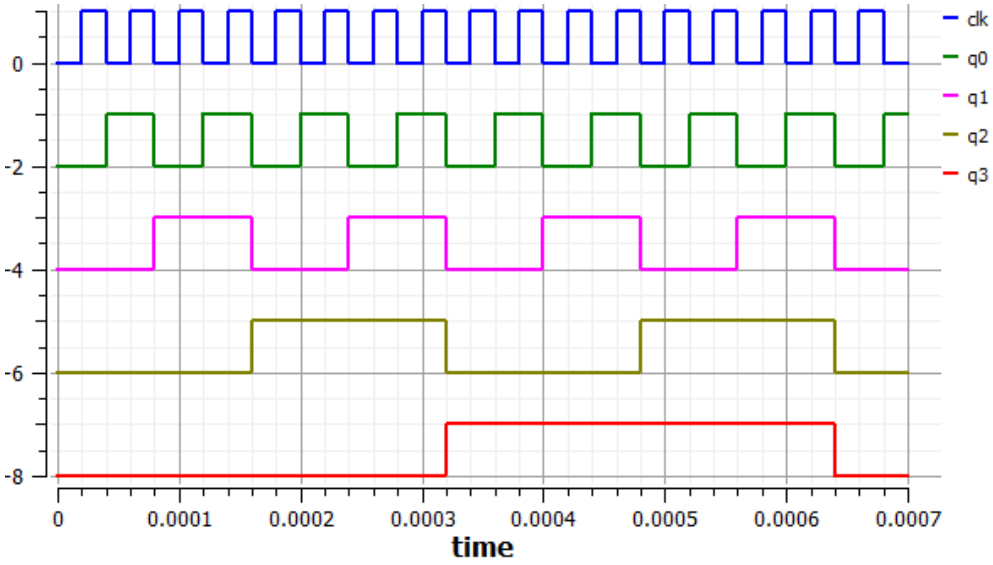
n	R_d
1	q_0
2	q_1
3	$q_0.q_1$
4	q_2
5	$q_0.q_2$
6	$q_1.q_2$
7	$q_0.q_1.q_2$
8	q_3
9	$q_0.q_3$
10	$q_1.q_3$
11	$q_0.q_1.q_3$
12	$q_2.q_3$
13	$q_0.q_2.q_3$
14	$q_1.q_2.q_3$
15	$q_0.q_1.q_2.q_3$
16	0

SOLVE BLOCKS:

- The offset digital block is set to -2, to prevent the waveforms from overlapping each other
- The transient simulation is started at t = 0 s and stopped at t = 700µs
- Output variables to be monitored are clk, q_0, q_1, q_2, q_3

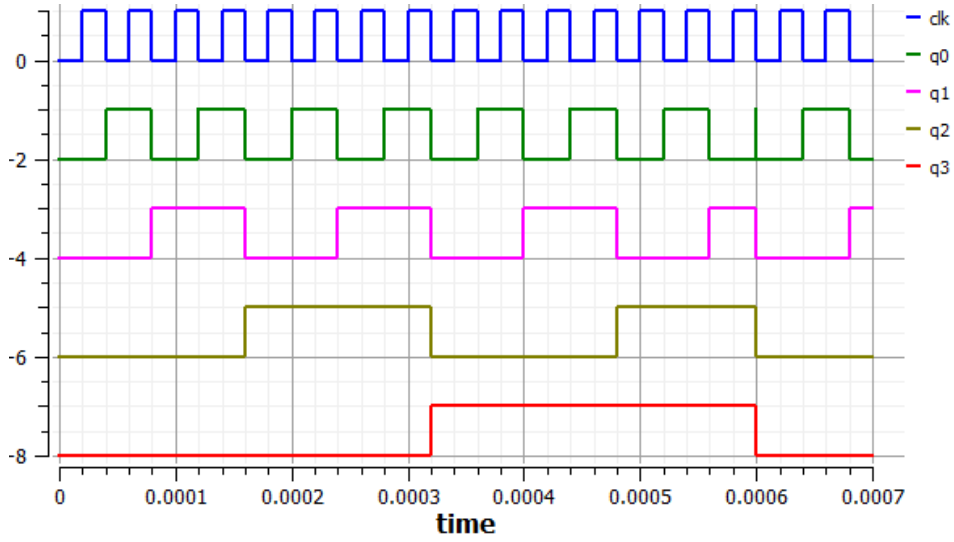


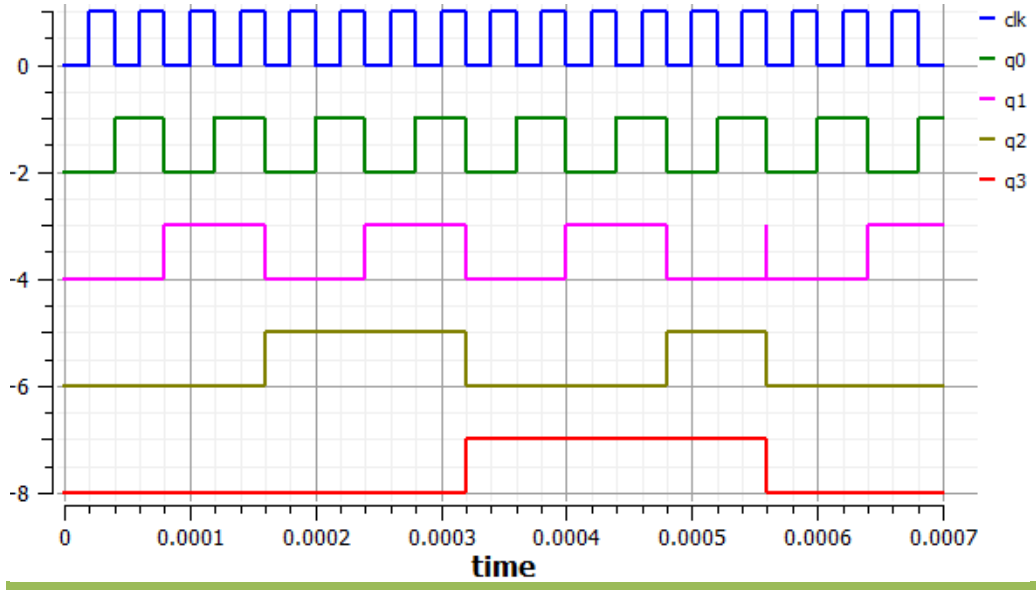
SIMULATED WAVEFORMS:



MOD-16

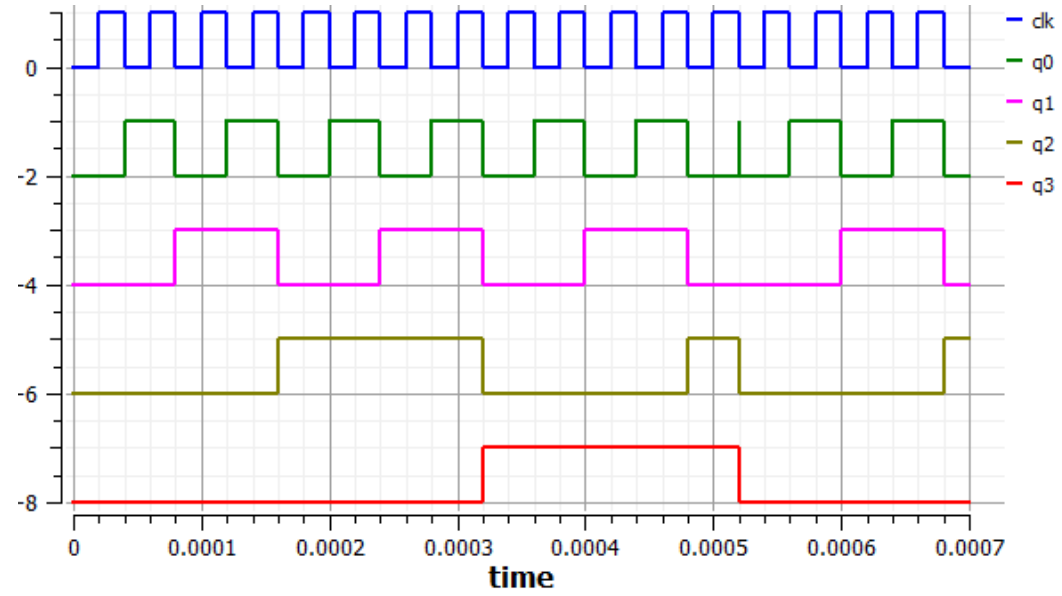
MOD-15

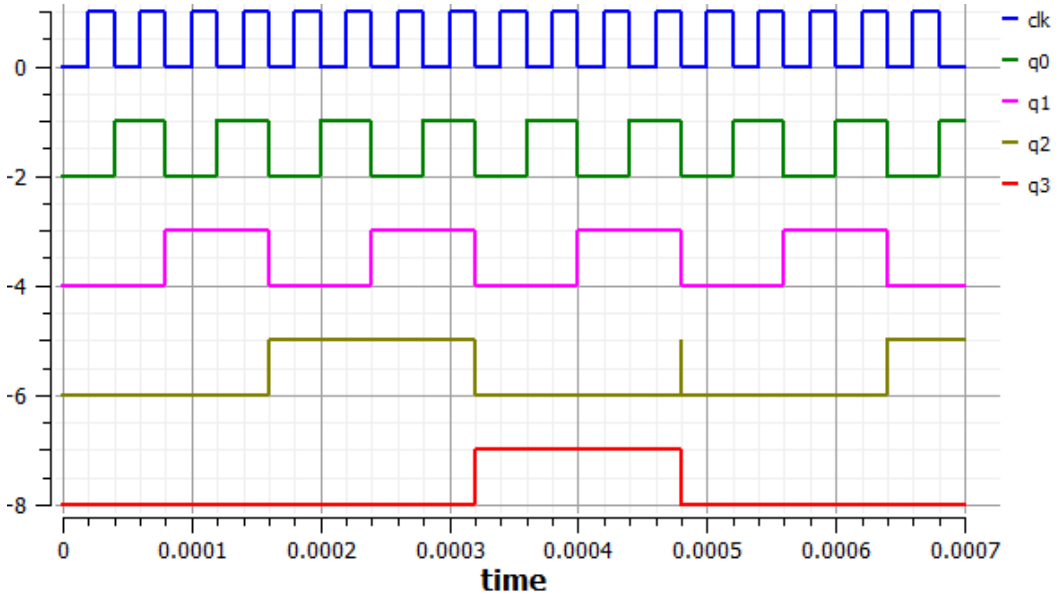




MOD-14

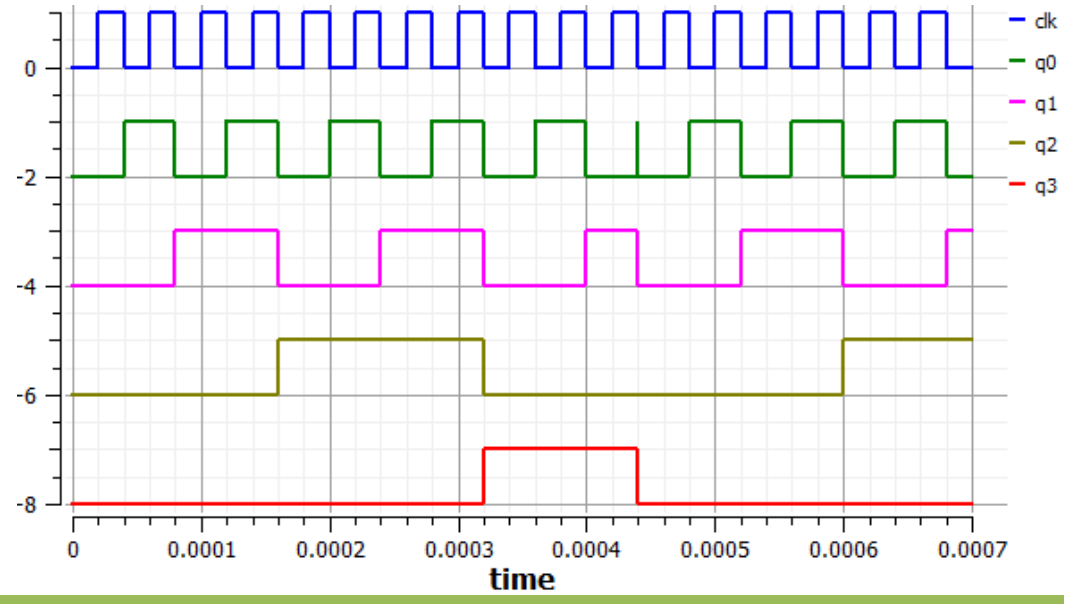
MOD-13

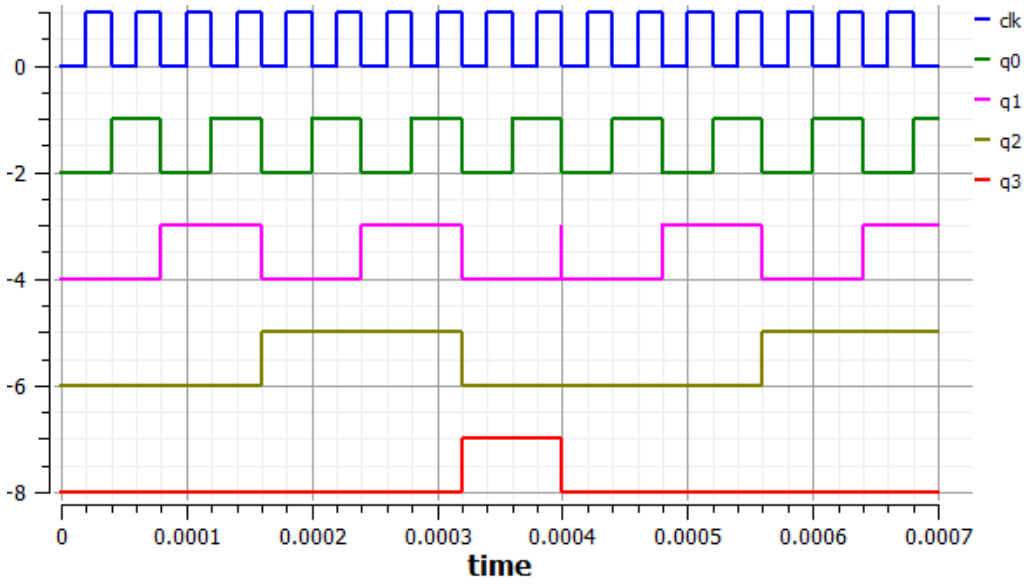




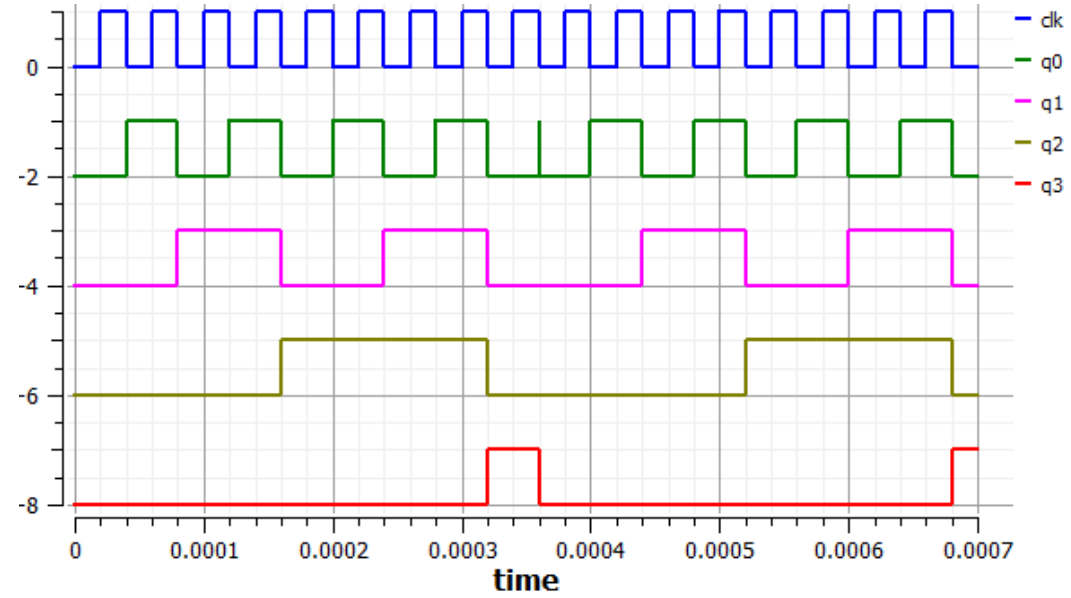
MOD-12

MOD-11

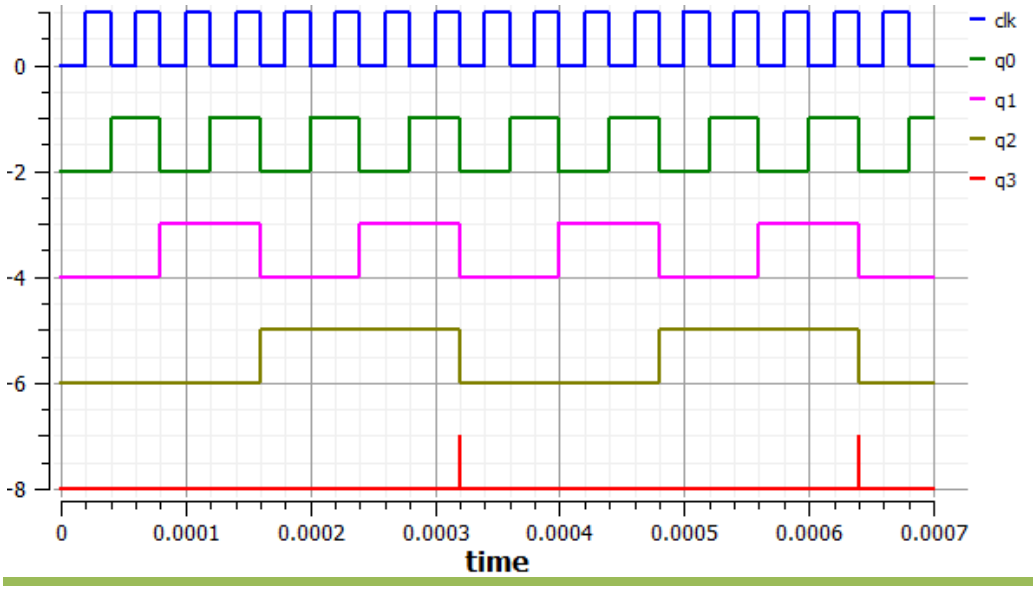




MOD-10

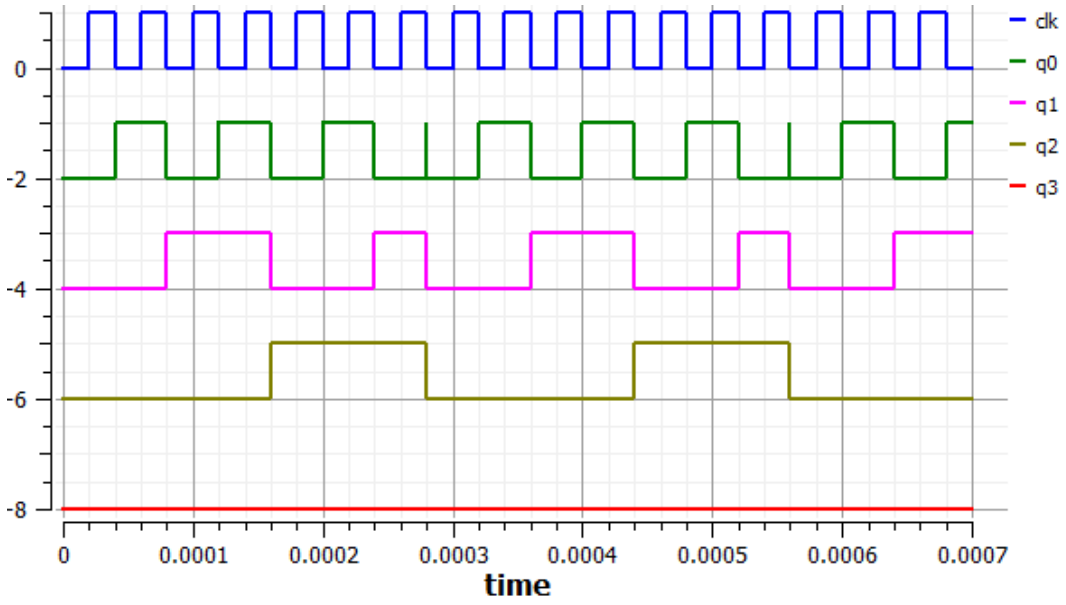


MOD-9

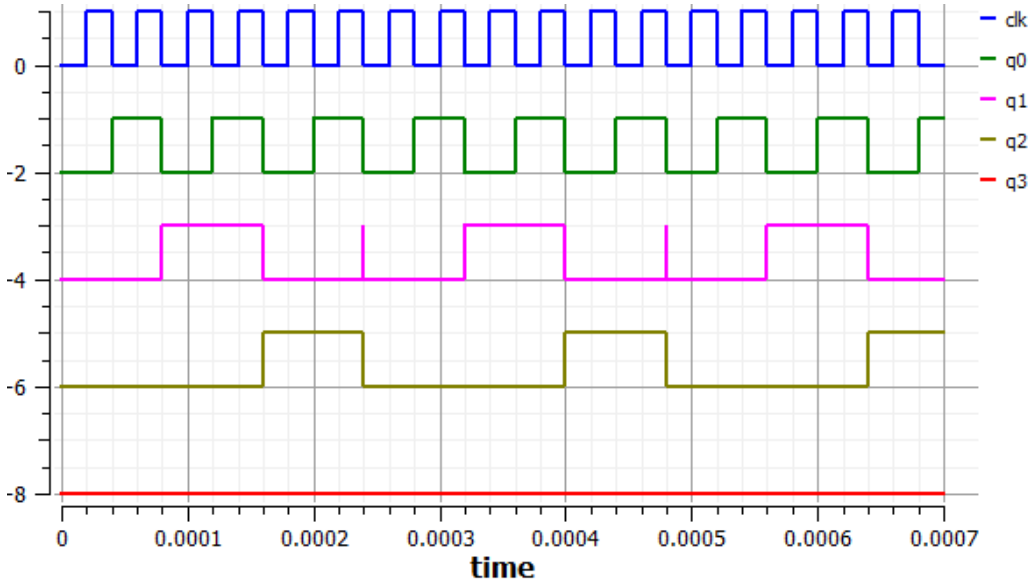


MOD-8

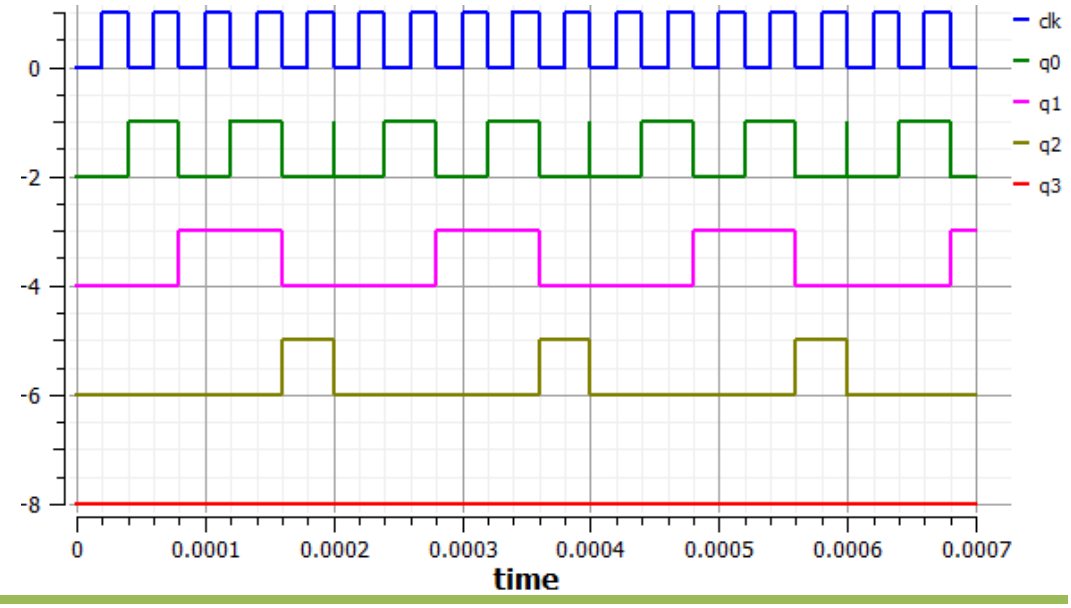
MOD-7

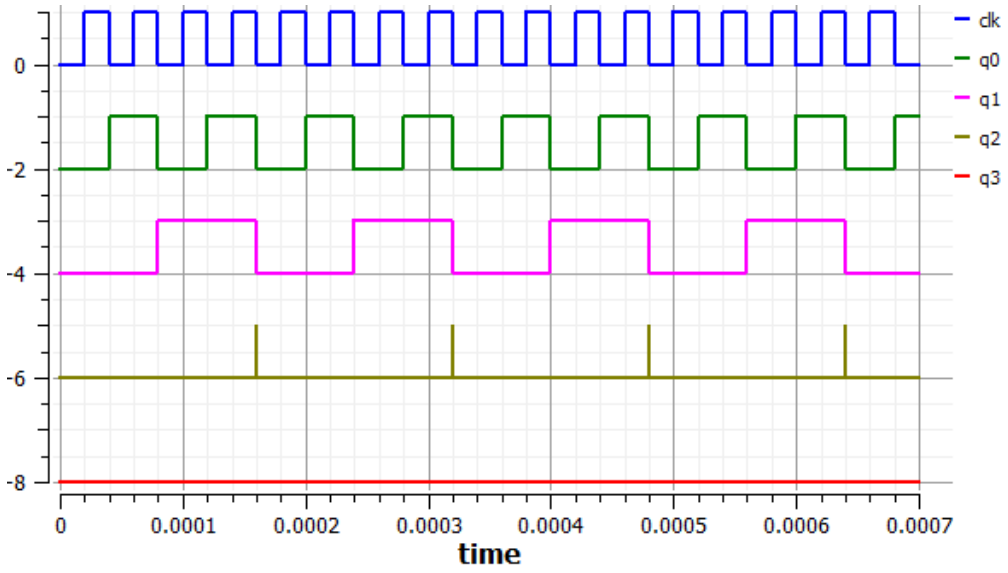


MOD-6

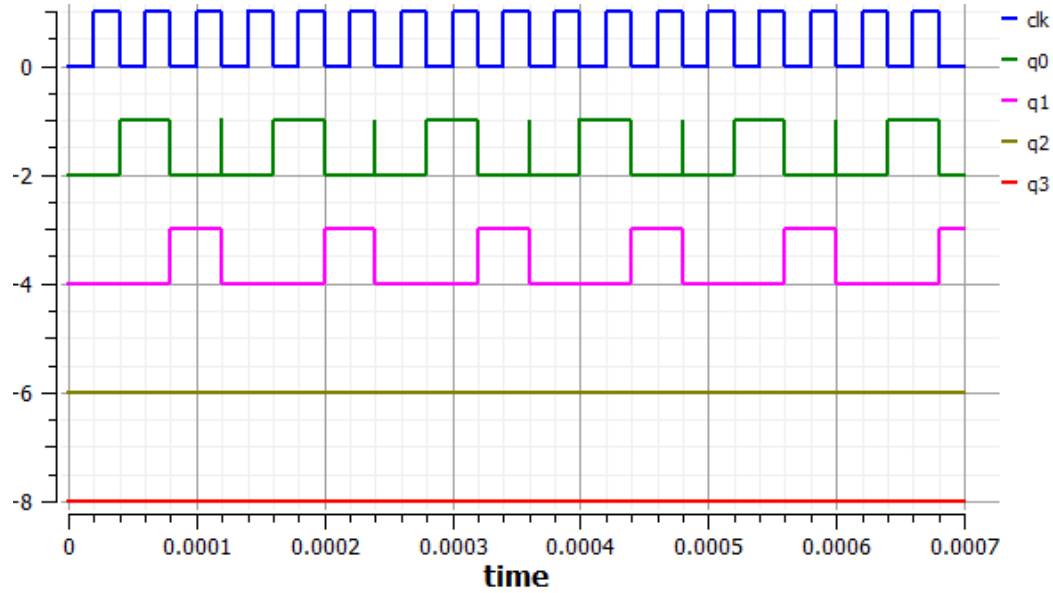


MOD-5

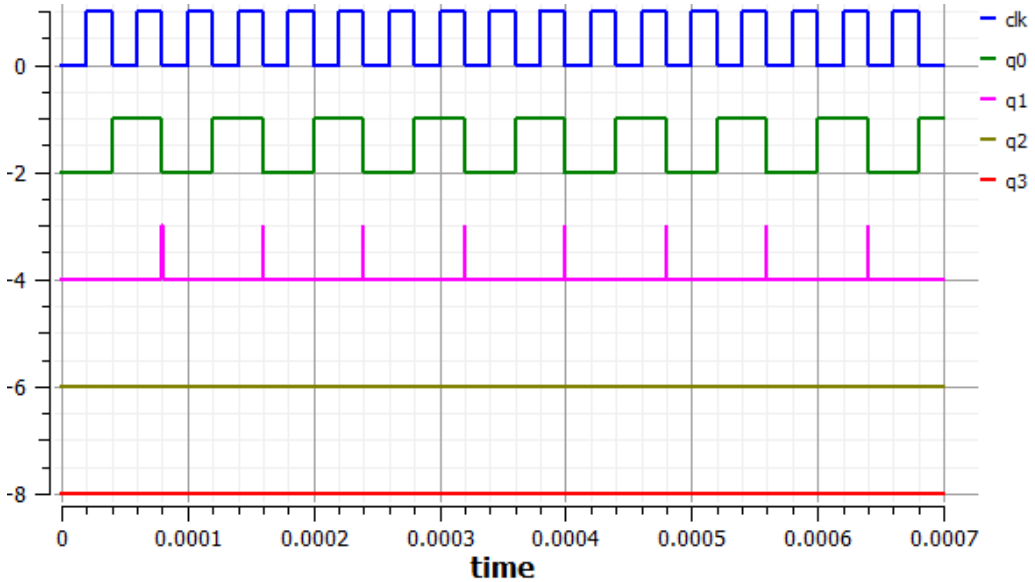




MOD-4

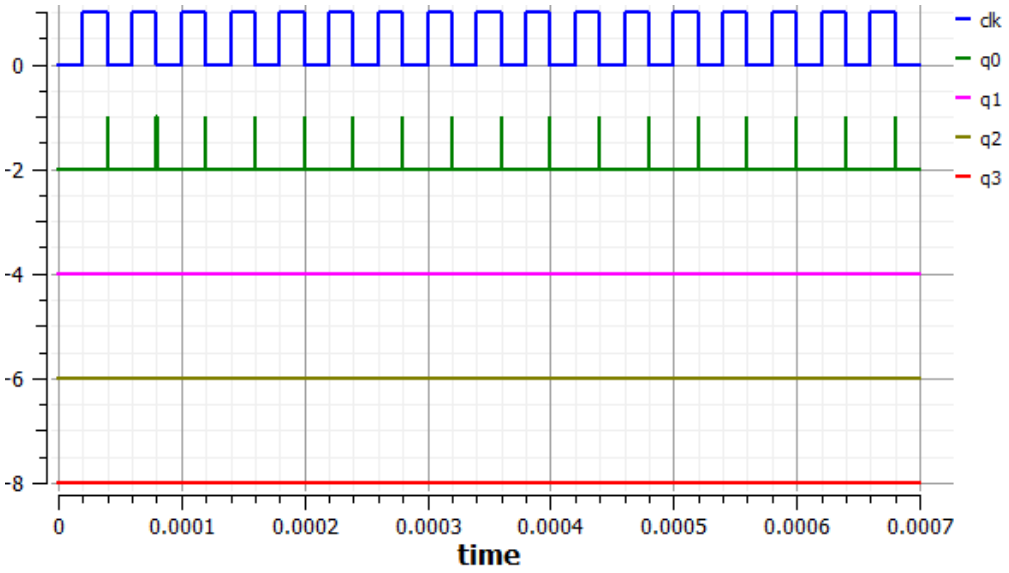


MOD-3



MOD-2

MOD-1



INFERENCES & CONCLUSIONS:

i. SEQUEL was found to be a user-friendly tool with good clarity in its use. It was easier to work with, as the documentation explained its use very elaborately.

ii. The presence of glitches at the end of every sequence is only because when the flip-flop tries to move from a LOW state to a HIGH state, the combinational logic would take some time to activate the direct-reset pins to all flip flops. This can thus be accounted due to the propagation delay of the gates.

iii. Because of the complexity of the circuit, physical realization would be difficult.