Rutgers University
School of Engineering

Fall 2022

332:231 – Digital Logic Design

Sophocles J. Orfanidis
ECE Department
orfanidi@rutgers.edu

Unit 7 – Registers, shift registers, counters, LFSRs

Course Topics

1. Introduction to DLD, Verilog HDL, MATLAB/Simulink
2. Number systems
3. Analysis and synthesis of combinational circuits
4. Decoders/encoders, multiplexers/demultiplexers
5. Arithmetic systems, comparators, adders, multipliers
6. Sequential circuits, latches, flip-flops
7. Registers, shift registers, counters, LFSRs
8. Finite state machines, analysis and synthesis

Text: J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018
additional references on Canvas Resources

Sequential circuits (Wakerly, Chapters 9, 10 ,11)

Topics discussed are:

- Registers
- Shift registers
- Counters
- Binary counters
- Counter design with D and T flip-flops
- Counters of arbitrary sequences
- Counter initialization
- BCD counters
- Counters for task control (e.g., traffic light control)
- Ring and Johnson counters
- Linear feedback shift registers (LFSR)

Contents:

1. Registers
2. Shift registers
3. Counters
4. Binary counters
5. Counter design with D and T flip-flops
6. Counters for task control (e.g., traffic light control)
7. Counters of arbitrary sequences
7. Counter initialization
8. BCD counters
9. Ring and Johnson counters
10. Linear feedback shift registers (LFSR)

References

J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018.

S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with Verilog Design*, 3/e, McGraw-Hill, 2014.

D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*, 2/e, Elsevier, 2013.

M. Mano, C. R. Kime, and T. Martin, *Logic and Computer Design Fundamentals*, 5/e, Pearson, 2016.

A. F. Kana, *Digital Logic Design*, [on Canvas].

E. O. Hwang, *Digital Logic and Microprocessor Design with Interfacing*, 2/e, Cengage, 2018.

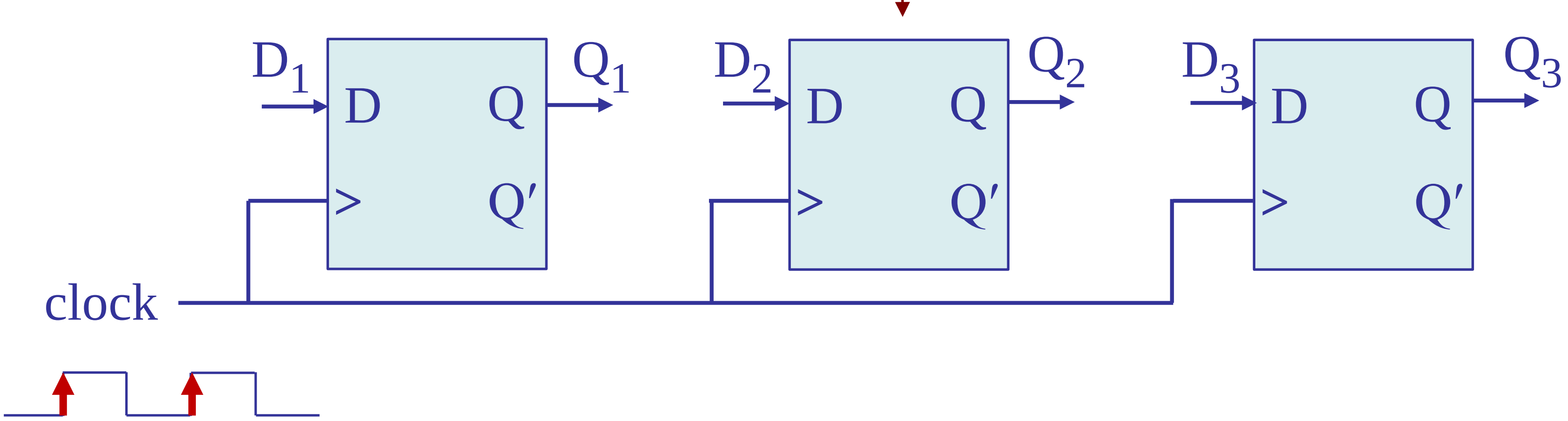
C. Maxfield, *Bebop to the Boolean Boogie*, 2/e, Newnes, 2009.

Wikipedia articles on Ring and Johnson counters and LFSRs.

registers

an *n*-bit register consists of *n* flip-flops, each independently storing *one bit* of information

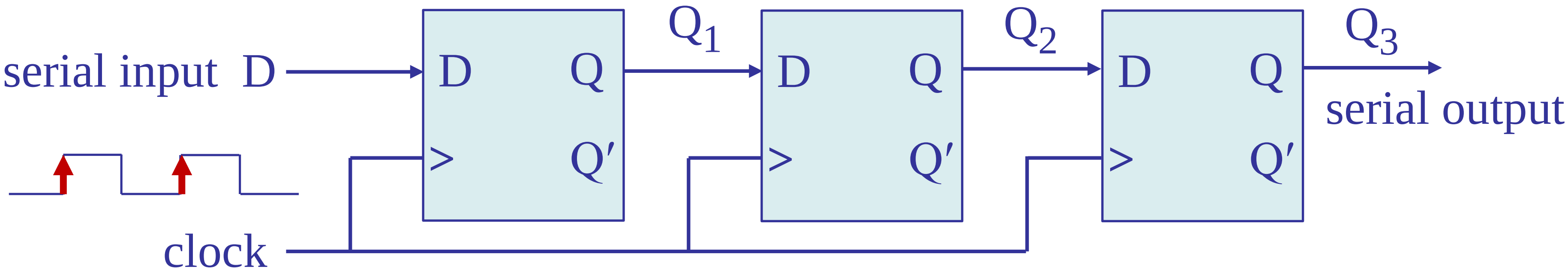
example: 3-bit register holding the bits Q_1, Q_2, Q_3



typically, all flip-flops are synchronously driven by the same clock, and additional external inputs and/or interconnections can be used to load and/or sequence the register contents.

shift registers

serial-in / serial-out



example

clock ticks

t	D(t)	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

$Q_1(t) = D(t\text{-}1)$, at clock edges

$Q_2(t) = Q_1(t\text{-}1) = D(t\text{-}2)$

$Q_3(t) = Q_2(t\text{-}1) = D(t\text{-}3)$

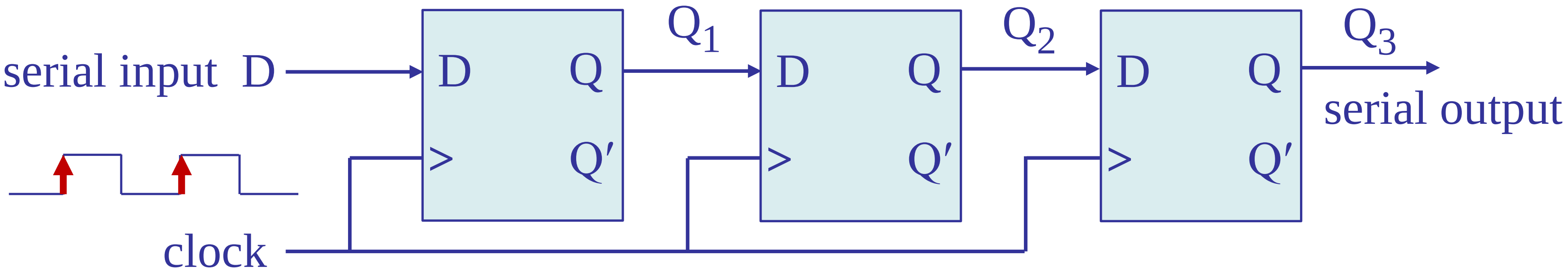
shift registers act as delay elements

$Q(t+1) = D(t)$, at clock edges

$Q(t) = D(t\text{-}1)$, at clock edges

shift registers

serial-in / serial-out



example

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

clock ticks

$Q_1(t) = D(t\text{-}1)$, at clock edges

$Q_2(t) = Q_1(t\text{-}1) = D(t\text{-}2)$

$Q_3(t) = Q_2(t\text{-}1) = D(t\text{-}3)$

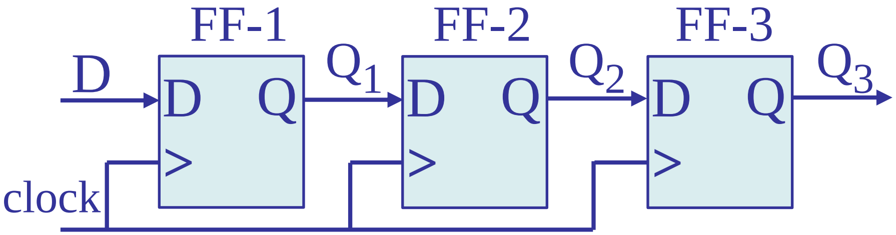
shift registers act as delay elements

$Q(t+1) = D(t)$, at clock edges
$Q(t) = D(t\text{-}1)$, at clock edges

state contents at successive (rising-edge) clock ticks

t	D(t)	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	-	0	0	0

← initial values at time t = 0–



state contents at successive (rising-edge) clock ticks

t	D(t)	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	-	1	0	0

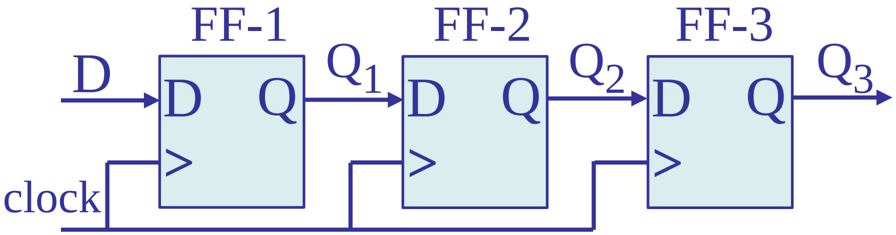
t=0

D is read and transferred to Q_1

Q_1 is transferred to Q_2

Q_2 is transferred to Q3

and wait until next clock tick at t=1



state contents at successive (rising-edge) clock ticks

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	-	0	1	0

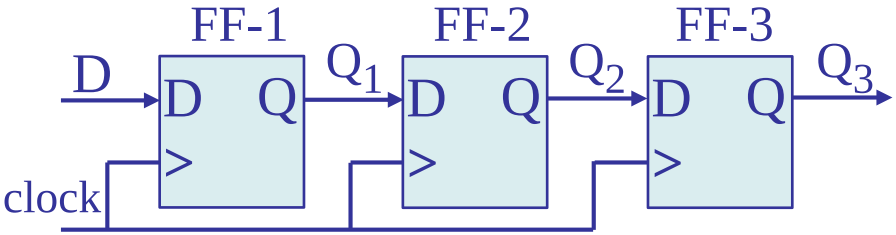
t=1

D is read and transferred to Q_1

Q_1 is transferred to Q_2

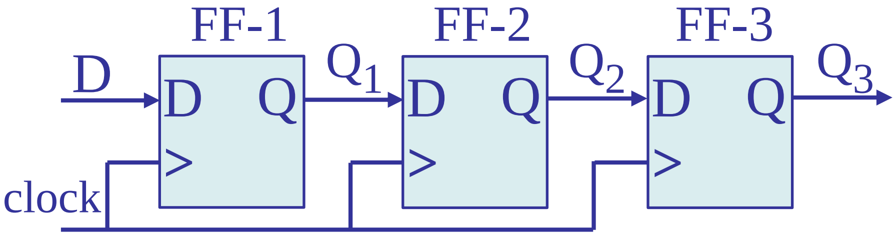
Q_2 is transferred to Q3

and wait until next clock tick at t=2



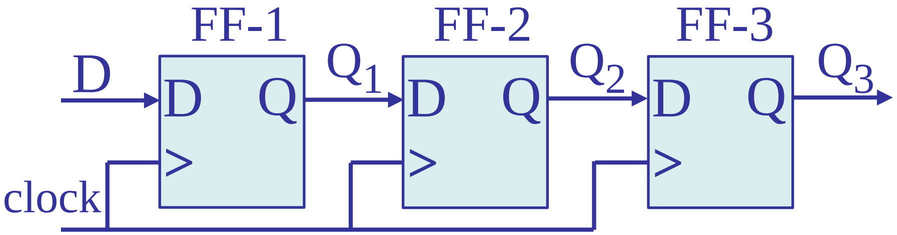
state contents at successive (rising-edge) clock ticks

t	D(t)	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	-	1	0	1



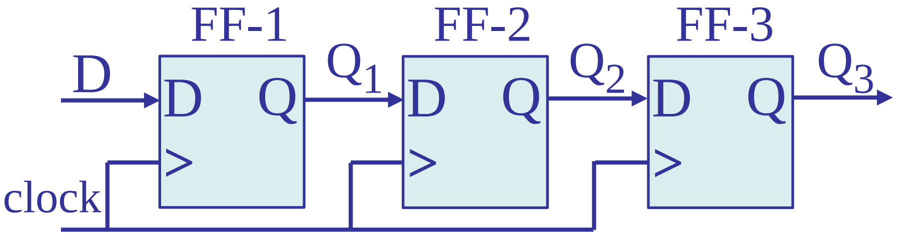
state contents at successive (rising-edge) clock ticks

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	-	1	1	0



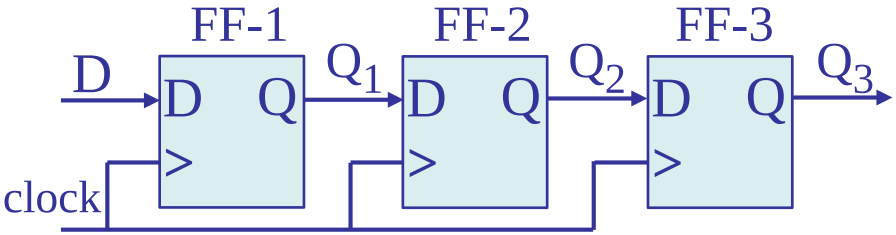
state contents at successive (rising-edge) clock ticks

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	-	1	1	1



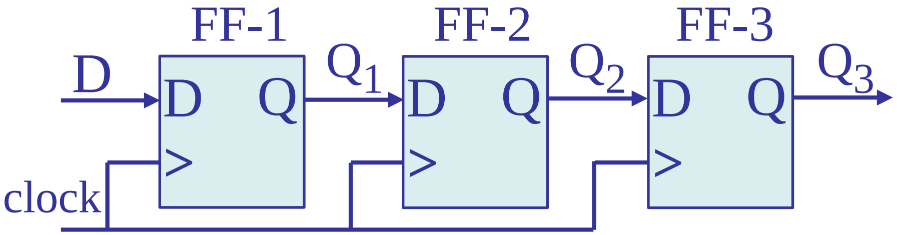
state contents at successive (rising-edge) clock ticks

t	D(t)	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1
6	-	0	1	1



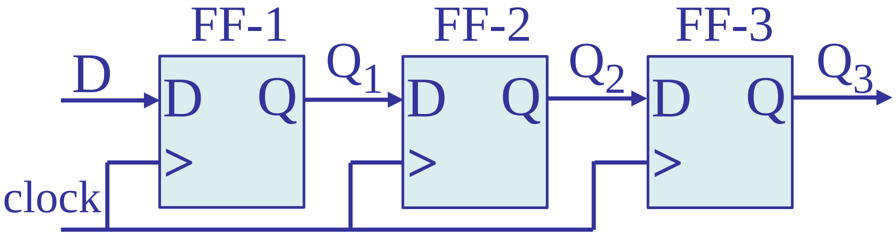
state contents at successive (rising-edge) clock ticks

t	D(t)	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1
6	0	0	1	1
7	-	0	0	1



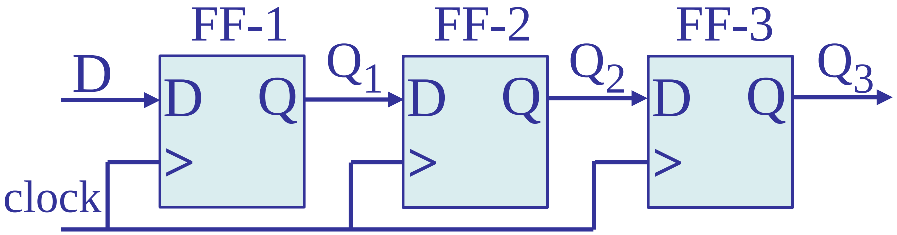
state contents at successive (rising-edge) clock ticks

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	-	0	0	0



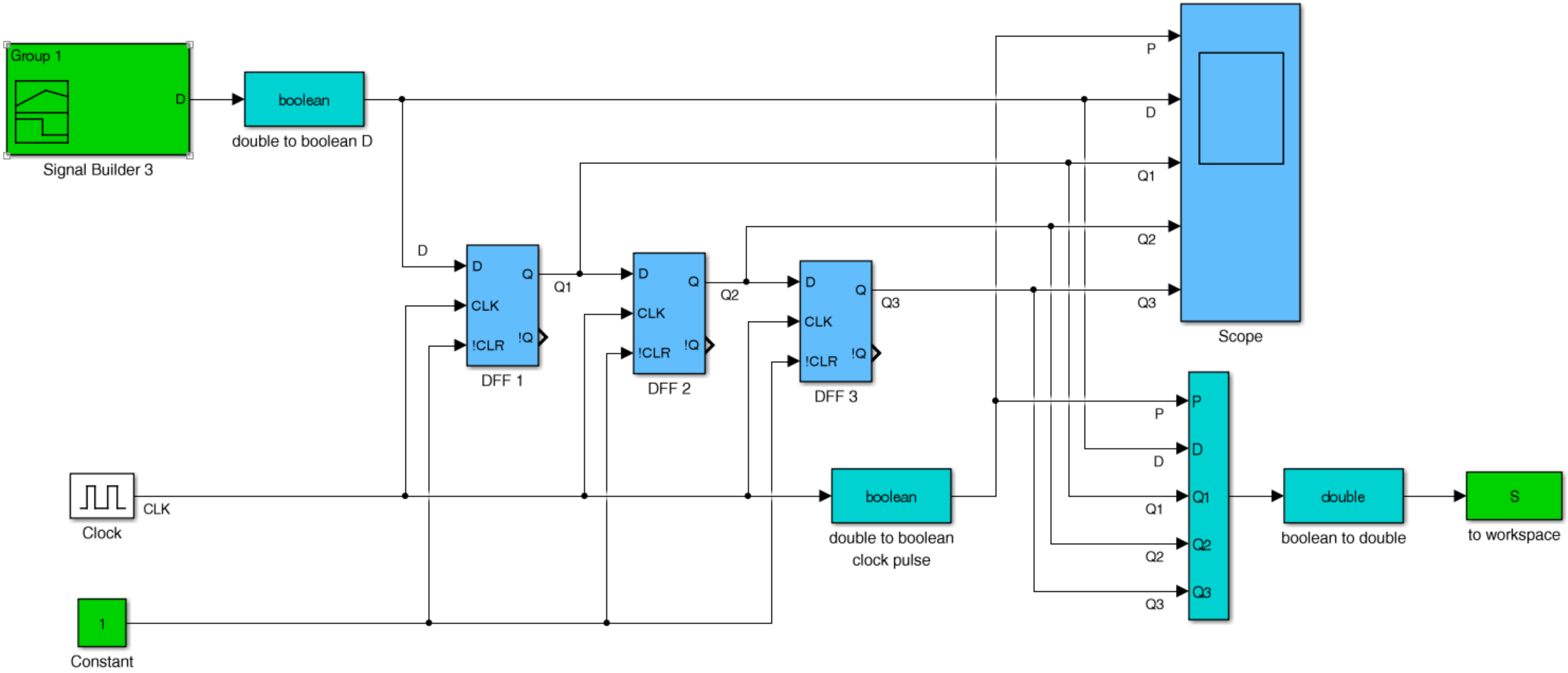
state contents at successive (rising-edge) clock ticks

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	1	0	1
4	1	1	1	0
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0



shift registers

serial-in / serial-out

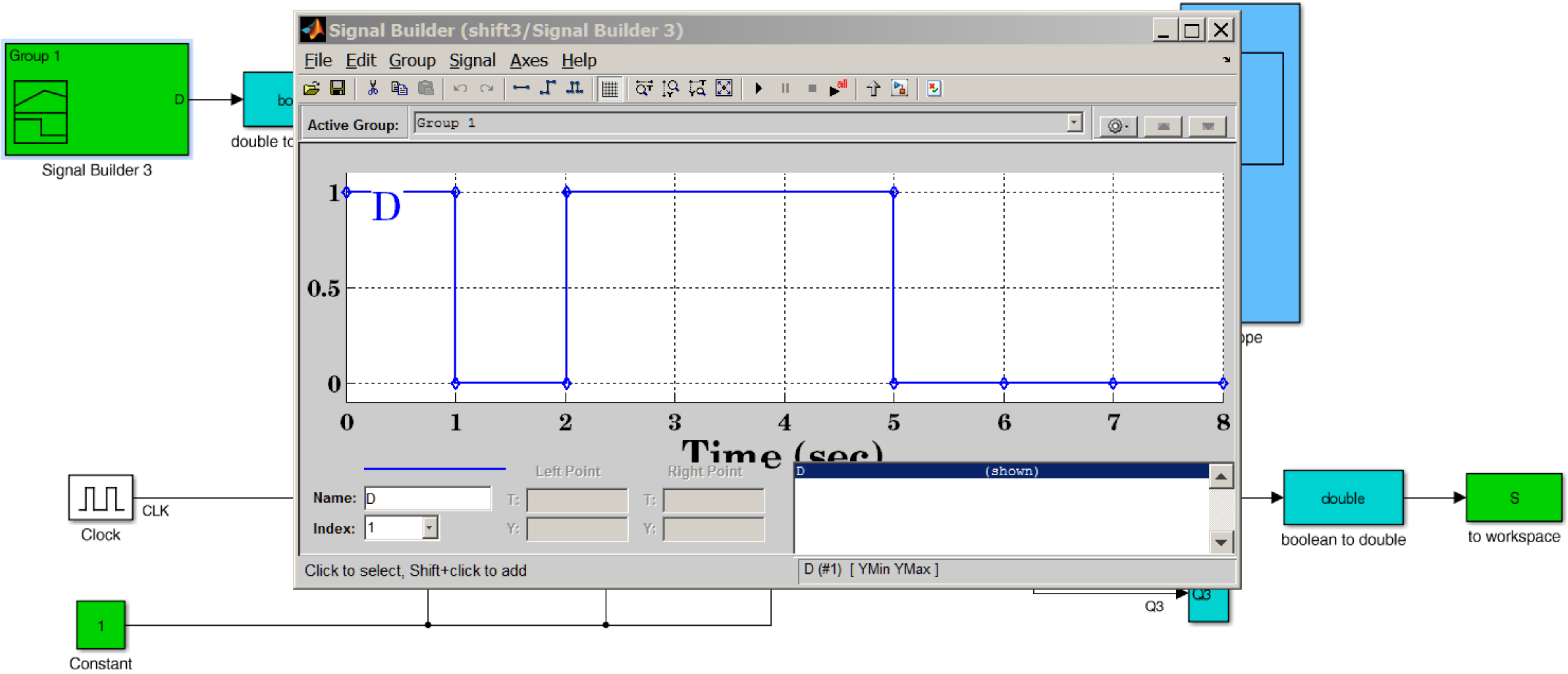


Simulink example: shifting the sequence 1 0 1 1 1 0 0 . . .

file: shift3.slx

shift registers

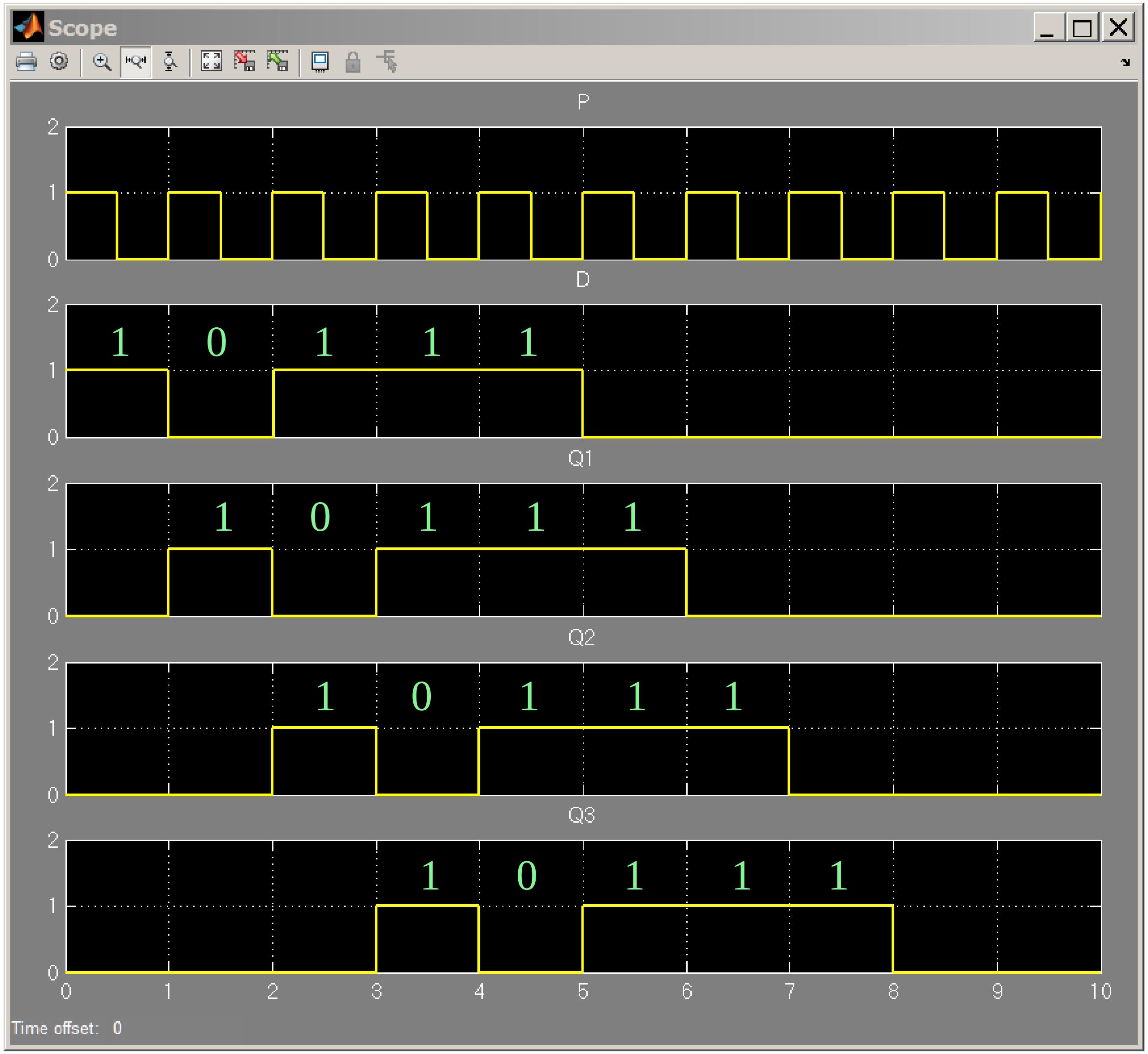
serial-in / serial-out



generate input sequence 1 0 1 1 1 0 0 . . .

shift registers

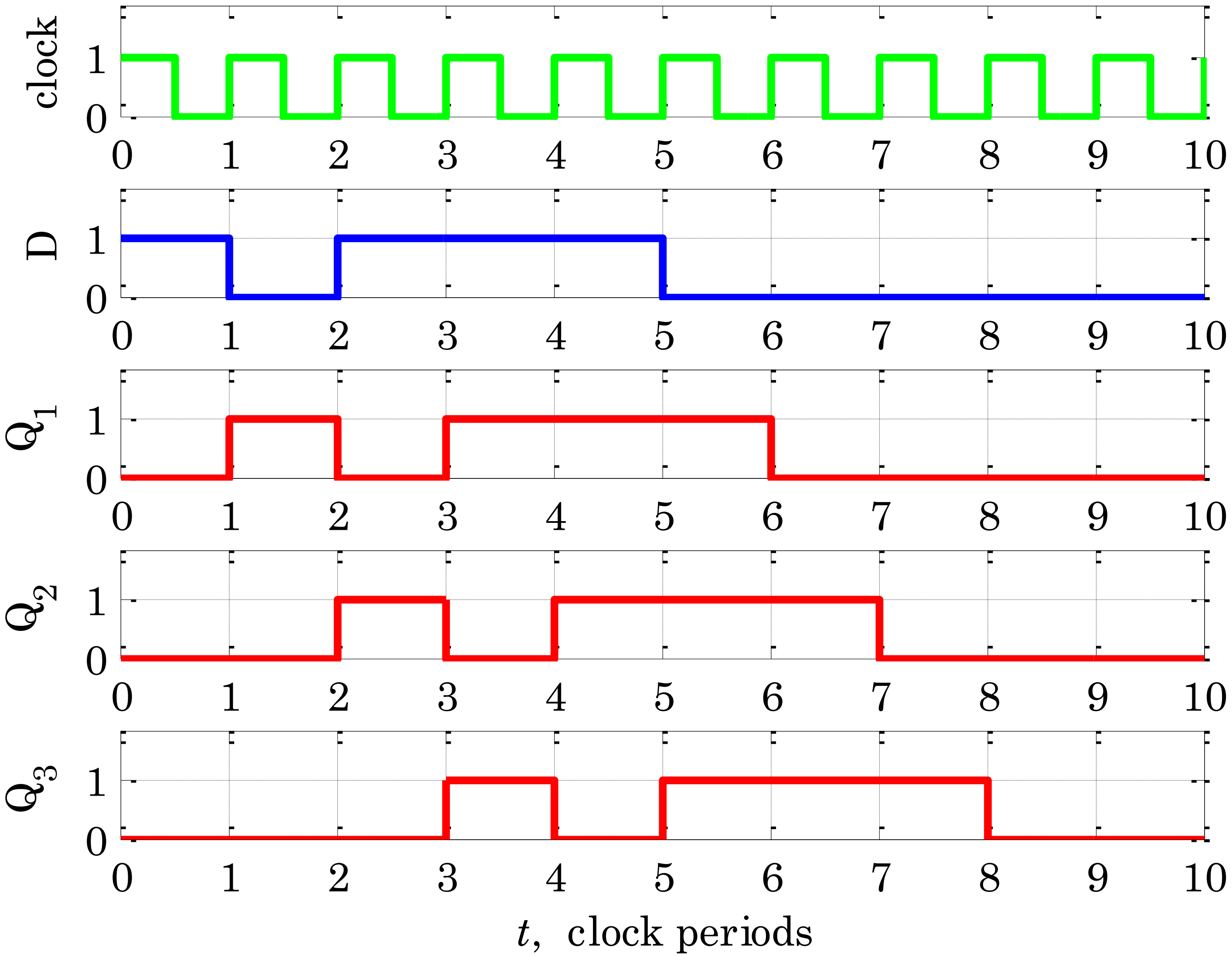
serial-in / serial-out



scope output

shift registers

serial-in / serial-out



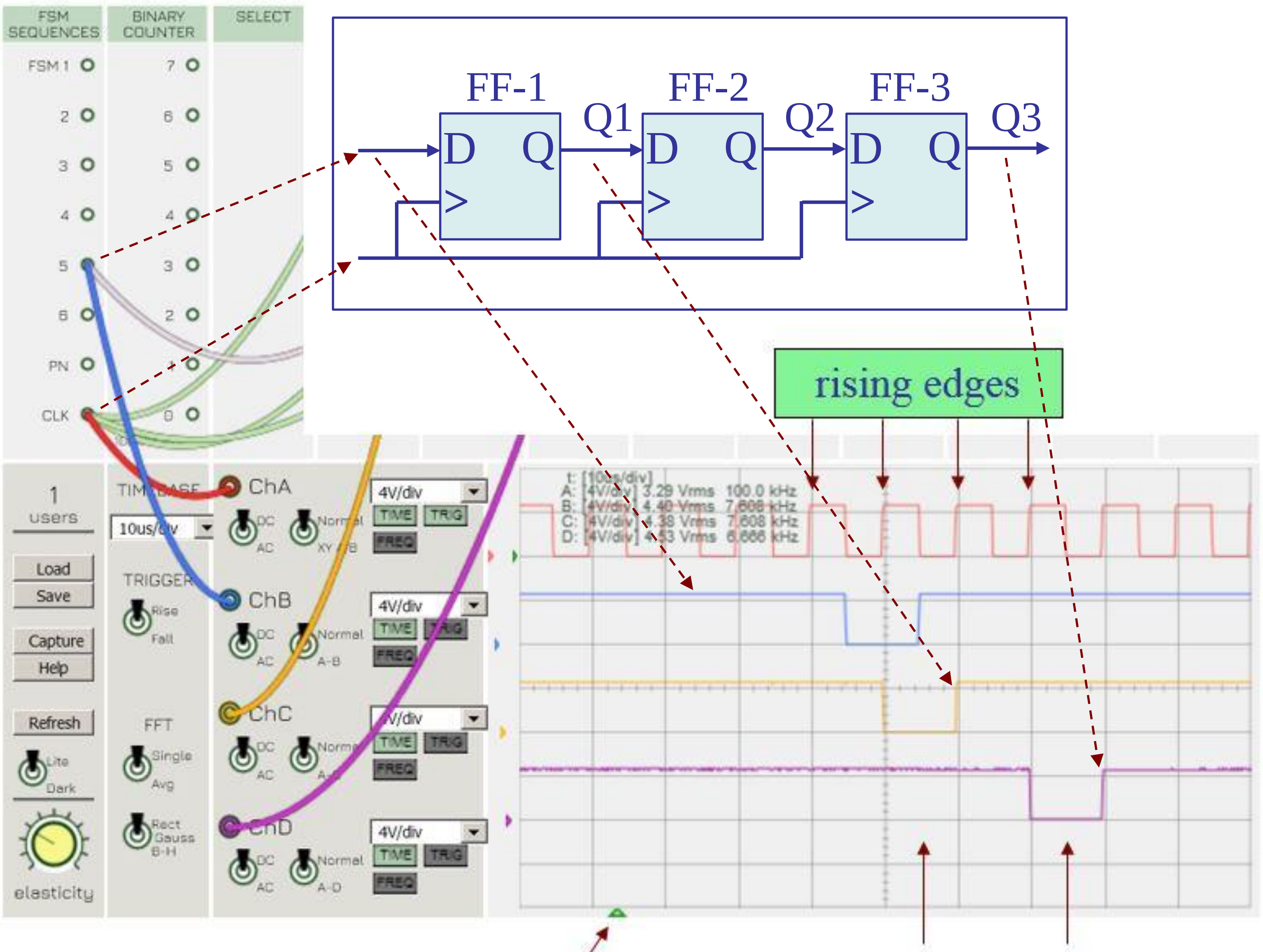
shift registers

serial-in / serial-out

```
% read Simulink data from timeseries structure S

t =  S.time;         % time
P =  S.data(:,1);    % clock pulse
D =  S.data(:,2);    % overall input
Q1 = S.data(:,3);    % flip-flop outputs
Q2 = S.data(:,4);
Q3 = S.data(:,5);

set(0,'DefaultAxesFontSize',10);

figure;
subplot(5,1,1); stairs(t,P,'g-');  ylabel('clock')
subplot(5,1,2); stairs(t,D,'b-');  ylabel('D');
subplot(5,1,3); stairs(t,Q1,'r-'); ylabel('Q_1');
subplot(5,1,4); stairs(t,Q2,'r-'); ylabel('Q_2');
subplot(5,1,5); stairs(t,Q3,'r-'); ylabel('Q_3');
xlabel('{\itt},  clock periods')
```

shift registers

Emona board – Lab 6



```
t: [10us/div]
A: [4V/div] 3.29 Vrms  100.0 kHz
B: [4V/div] 4.40 Vrms  7.608 kHz
C: [4V/div] 4.38 Vrms  7.608 kHz
D: [4V/div] 4.53 Vrms  6.666 kHz
```

rising edges

input

FF-1

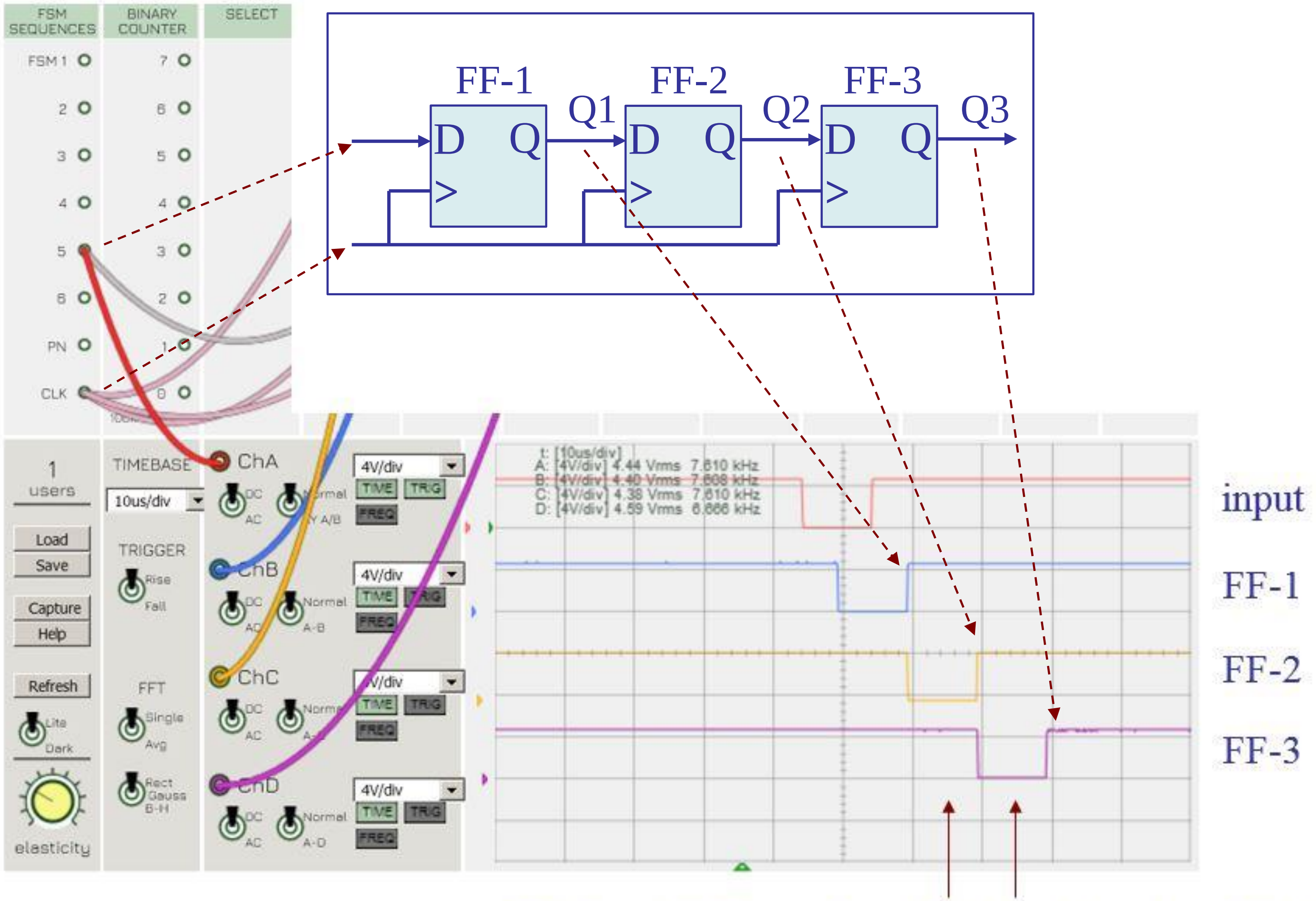
FF-3

adjust time-base here

FF-3 is two delays later than FF-1

shift registers

Emona board – Lab 6



10us/div

4V/div

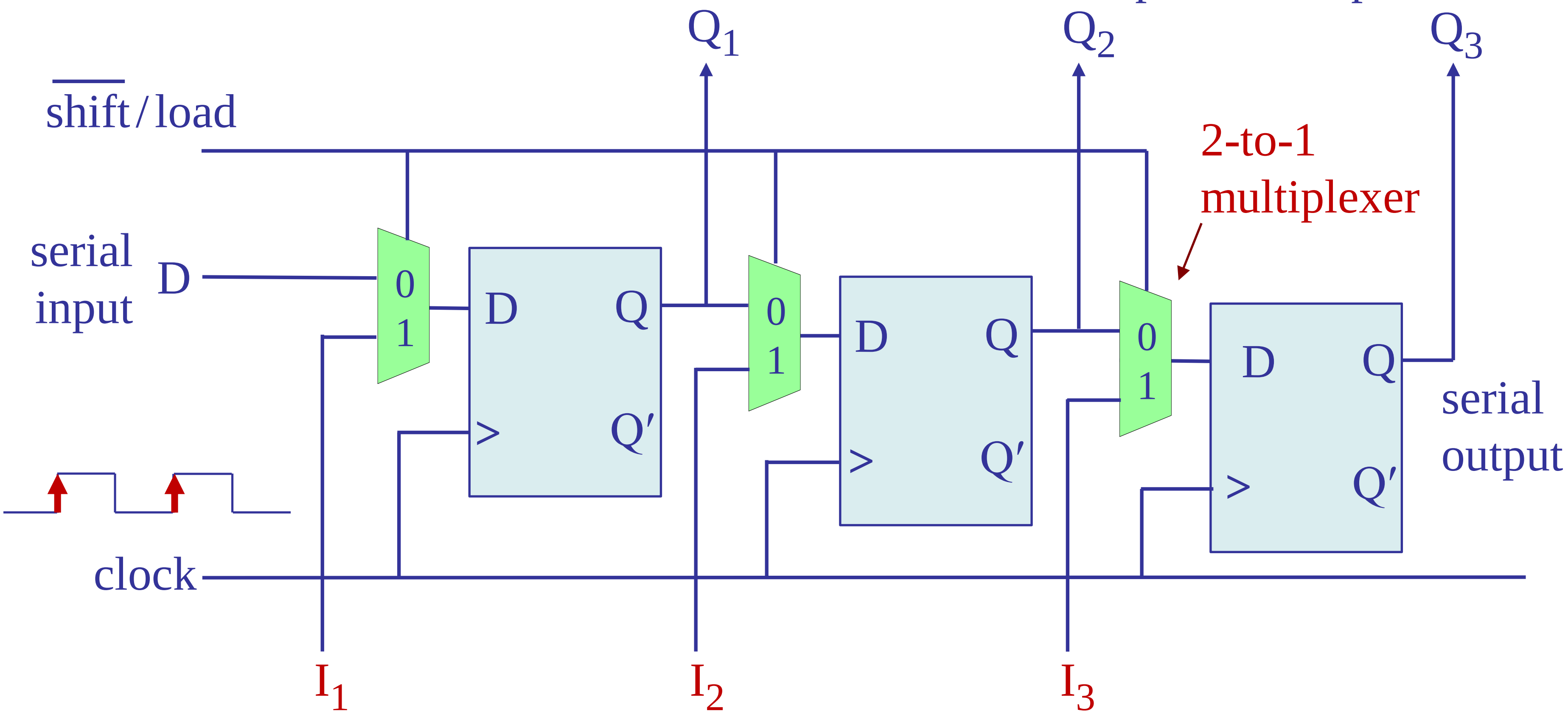
```
t: [10us/div]
A: [4V/div] 4.44 Vrms  7.610 kHz
B: [4V/div] 4.40 Vrms  7.608 kHz
C: [4V/div] 4.38 Vrms  7.610 kHz
D: [4V/div] 4.59 Vrms  6.666 kHz
```

FF-2 and FF-3 are 2 and 3 delays later than FF-1

shift registers

serial-to-parallel converter
serial-in / parallel-in / parallel-out / serial-out

parallel output



2-to-1 multiplexer

serial output

initial contents

shift registers

serial-to-parallel converter
serial-in / parallel-in / parallel-out / serial-out

clock ticks →

with initial contents

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	D_0	I_1	I_2	I_3
1	D_1	D_0	I_1	I_2
2	D_2	D_1	D_0	I_1
3	D_3	D_2	D_1	D_0
4	D_4	D_3	D_2	D_1
5	D_5	D_4	D3	D_2
6	D_6	D_5	D_4	D_3
7	D_7	D_6	D_5	D_4
8	D_8	D_7	D_6	D_5

. . .

zero initial contents

t	$D(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$
0	D_0	0	0	0
1	D_1	D_0	0	0
2	D_2	D_1	D_0	0
3	D_3	D_2	D_1	D_0
4	D_4	D_3	D_2	D_1
5	D_5	D_4	D3	D_2
6	D_6	D_5	D_4	D_3
7	D_7	D_6	D_5	D_4
8	D_8	D_7	D_6	D_5

. . .

counters

counters have many uses in DLD:

- generating time intervals for application control (e.g., for traffic light control)
- counting events
- measuring elapsed time between events

there are many types of counters:

- binary counters
- gray-code counters
- BCD counters
- counting in arbitrary order (see also, lab-6)
- generating arbitrary sequences
- ring counters
- Johnson counters

and specialized ones, such as linear feedback shift registers (LFSR) for generating random-like sequences used in error control coding and cryptography applications, in the testing of digital circuits, and in secure wireless communications, spread spectrum and frequency hopping (see Hedy Lamarr & George Antheil's frequency hopping patents, and Prof. Soljanin's PBS podcast on Hedy Lamarr, on ECE)

counters

Synchronous counters are specialized types of finite state machines which are sequentially clocked to generate a desired periodic sequence of numbers.

With *n* flip-flops, one can generate up to 2^n sequence values (states), which can be labeled either according to their decimal values or in ta coded binary form.

design procedure:

1. Specify the state transition table for the desired sequence, i.e., how the values are to be sequenced in time.
2. Convert the state table into a state-assigned table (i.e., assigning state variables) using a binary representation of each number in the sequence.
3. Use K-maps to simplify the next-state functions, $Q \rightarrow Q^{next}$, for each bit.
4. The individual bits Q become the digital states and are stored in flip-flops. If D-flip-flops are used, the inputs to the flip-flops are trivial because, $D = Q^{next}$. If T or JK flip-flops are used, their inputs T or J,K are also straightforward (see p. 49).

counters

For example, in order to generate the length-8 sequence of numbers, [s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7], we may list in a table how these numbers (states) are sequenced from one clock time instant to the next.

state transition table

t	s_t	next s_{t+1}
0	s_0	s_1
1	s_1	s_2
2	s_2	s_3
3	s_3	s_4
4	s_4	s_5
5	s_5	s_6
6	s_6	s_7
7	s_7	s_0
8	s_0	s_1
	etc.	

clock ticks

reset/repeat

state diagrams are a graphical representation of state tables

counters

state diagrams (to be discussed further in unit-8) are directed graphs that convey the same information as state tables, but in a graphical form, where the states are listed inside the circles and the arrows indicate the state transitions taking place at successive clock-edge time instants.

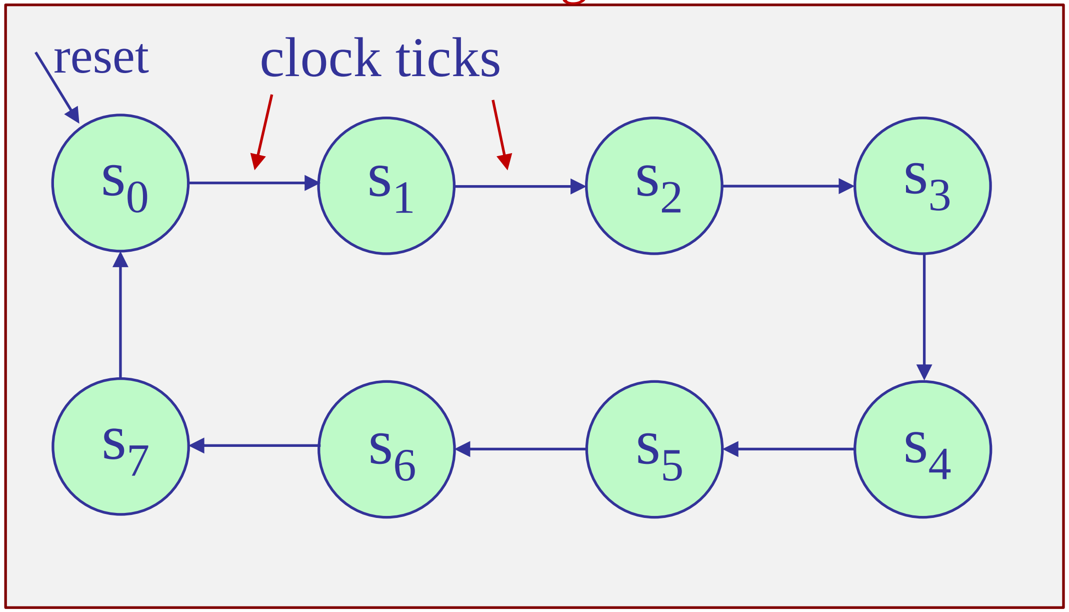
state table

t	s_t	next s_{t+1}
0	s_0	s_1
1	s_1	s_2
2	s_2	s_3
3	s_3	s_4
4	s_4	s_5
5	s_5	s_6
6	s_6	s_7
7	s_7	s_0
8	s_0	s_1
	etc.	

clock ticks

repeat

state diagram



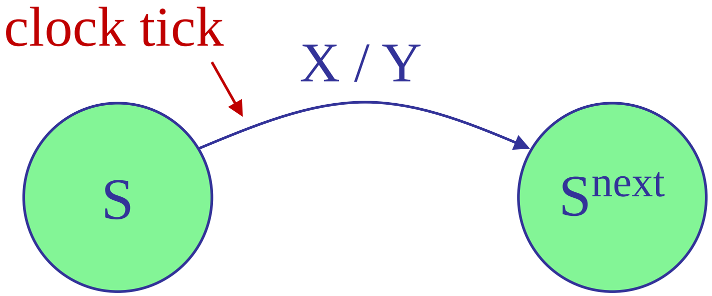
the number of states is finite, hence, the name "finite state machines". More general state diagrams can have more complicated transitions between states, including additional inputs/outputs (see unit-8)

Conventions for drawing state diagrams:

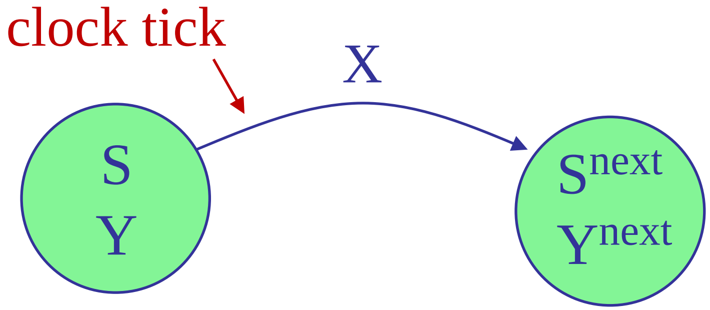
Along each arrow connecting two successive states S and S^{next}, indicate the values of the inputs X that caused this transition from the current state S.

For Mealy machines, indicate along the same arrow the values of the outputs Y that resulted from the values of X and the current state S.

For Moore machines, because the outputs Y depend only on the current state Q, indicate the values of Y inside the circle for the current state S.



Mealy



counters

once the state table for the desired sequence is specified, one must make state assignments, that is, representing each number in the sequence in binary, e.g., 3 bits in the present example. These bits serve as the states and are stored in flip-flops.

state table

t	s_t	s_{t+1}
0	s_0	s_1
1	s_1	s_2
2	s_2	s_3
3	s_3	s_4
4	s_4	s_5
5	s_5	s_6
6	s_6	s_7
7	s_7	s_0
8	s_0	s_1
	etc.	

→

state-assigned table or characteristic table

t	$Q_2\ Q_1\ Q_0$	$Q_2^{\text{next}}\ Q_1^{\text{next}}\ Q_0^{\text{next}}$
0	. . .	. . .
1	. . .	. . .
2		
3		
4		
5		
6		
7		
8	. . .	. . .
	etc.	etc.

for D-flip-flops, these become the flip-flop inputs, i.e.,

$$D_2 = Q_2^{\text{next}}$$
$$D_1 = Q_1^{\text{next}}$$
$$D_0 = Q_0^{\text{next}}$$

counters

In general, for a length-N counter, one needs, $n = \text{ceiling}(\log_2 N)$, state variables, each to be stored in a separate flip-flop. Each of the N numbers in the counter is encoded with n state variables, and represented as an n-bit binary number (state assignment).

Example: $N=6$, $n = \text{ceiling}(\log_2(6)) = \text{ceiling}(2.5850) = 3$

state table

t	s_t	s_{t+1}
0	s_0	s_1
1	s_1	s_2
2	s_2	s_3
3	s_3	s_4
4	s_4	s_5
5	s_5	s_0
6	s_0	s_1
7	s_1	s_2
8	s_2	s_3
	etc.	

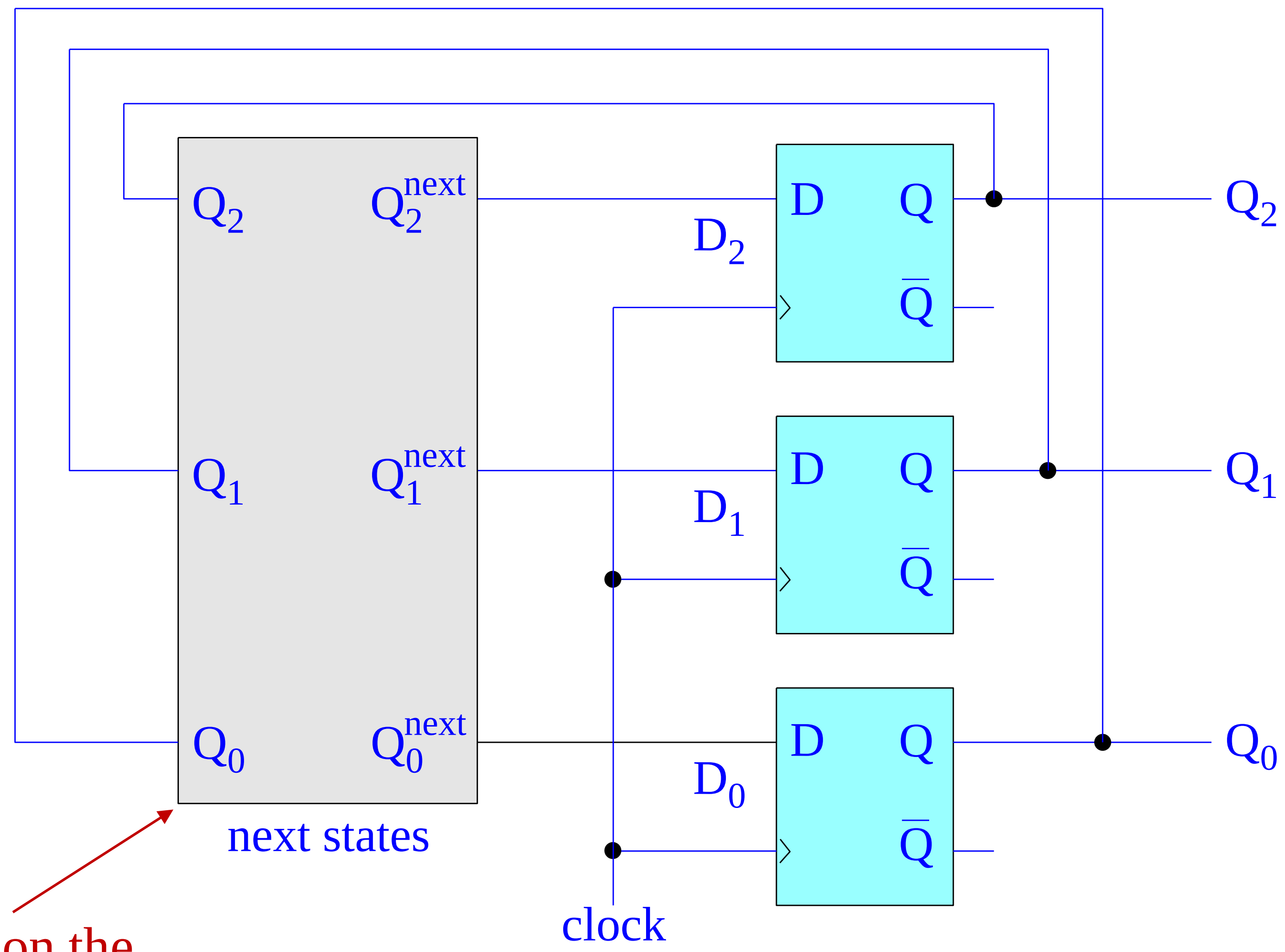
N=6 example

repeat

characteristic table

t	$Q_2\ Q_1\ Q_0$	$Q_2^{next}\ Q_1^{next}\ Q_0^{next}$
0	. . .	. . .
1	. . .	. . .
2	. . .	. . .
3	. . .	. . .
4	. . .	. . .
5	. . .	. . .
6	. . .	. . .
7	. . .	. . .
8	. . .	. . .
	etc.	etc.

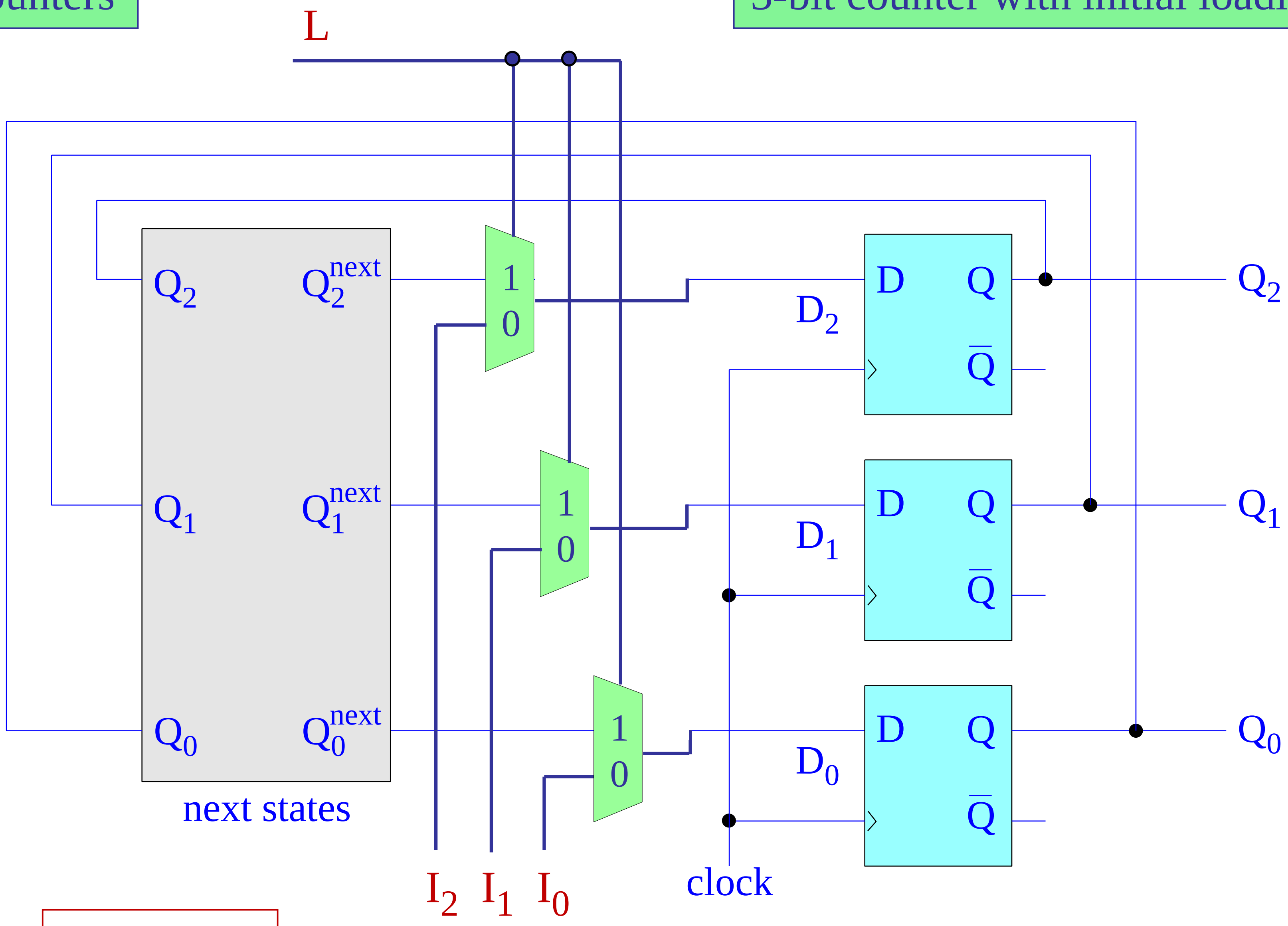
3-bit counter



depends on the counting sequence

counters

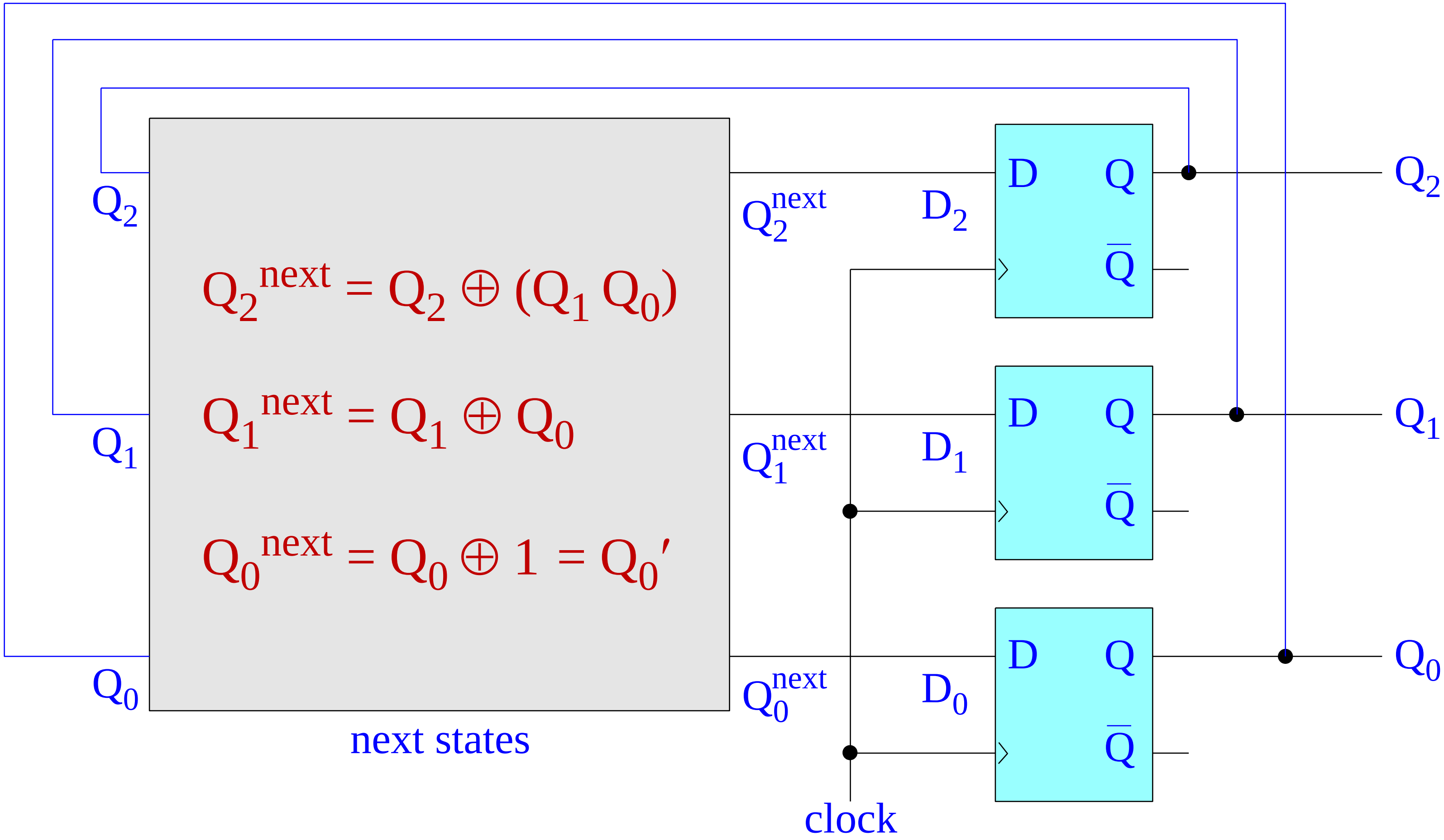
3-bit counter with initial loading



L=0, load
L=1, count

load initial values
with 2-to-1 multiplexers

Example 1 – three-bit binary counter. Derive and implement with logic gates the next-state logic for an ordinary 3-bit binary counter, with a periodically repeating length-8 sequence, [0, 1, 2, 3, 4, 5, 6, 7].



$$Q_2^{next} = Q_2 \oplus (Q_1\, Q_0)$$

$$Q_1^{next} = Q_1 \oplus Q_0$$

$$Q_0^{next} = Q_0 \oplus 1 = Q_0'$$

characteristic table

s_t	s_{t+1}	Q_2	Q_1	Q_0	Q_2^{next}	Q_1^{next}	Q_0^{next}
0	1	0	0	0	0	0	1
1	2	0	0	1	0	1	0
2	3	0	1	0	0	1	1
3	4	0	1	1	1	0	0
4	5	1	0	0	1	0	1
5	6	1	0	1	1	1	0
6	7	1	1	0	1	1	1
7	0	1	1	1	0	0	0

← repeat

start by listing the binary representation of the given sequence and the next states

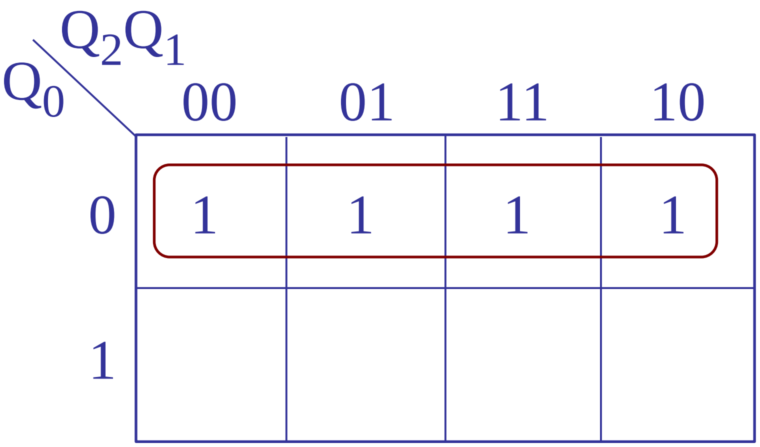
then, use K-maps to derive the next-state logic functions,

$$Q \rightarrow Q^{\text{next}}$$

recalling the property: $a \oplus b = a\, b' + a'\, b$

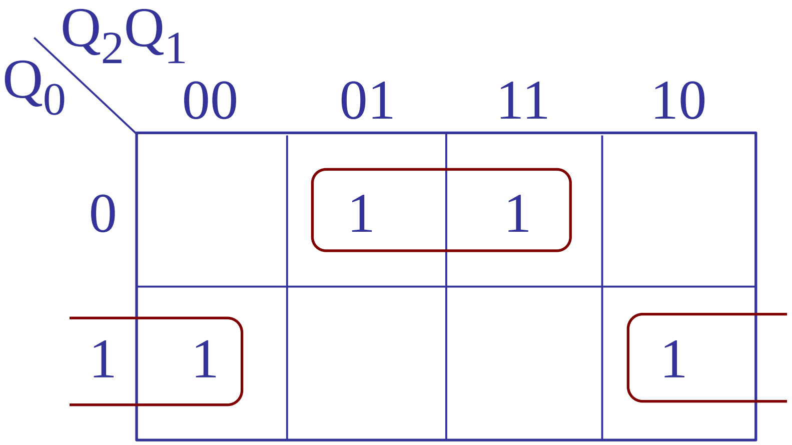
characteristic table

s_t	Q_2	Q_1	Q_0	Q_2^{next}	Q_1^{next}	Q_0^{next}
0	0	0	0	0	0	1
1	0	0	1	0	1	0
2	0	1	0	0	1	1
3	0	1	1	1	0	0
4	1	0	0	1	0	1
5	1	0	1	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	0	0



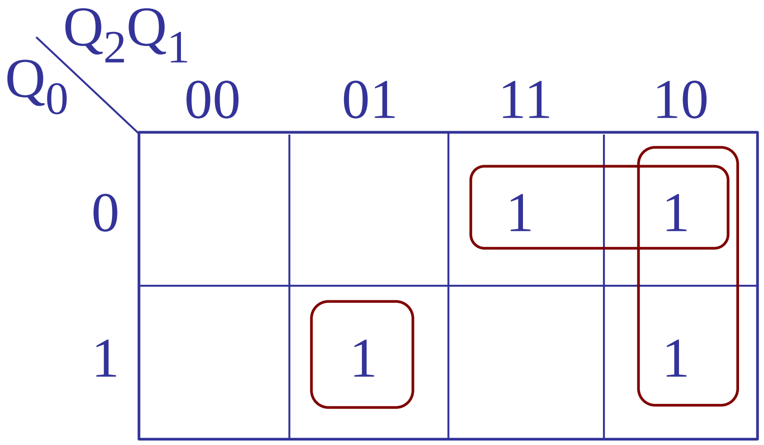
$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	1	1	1	1
1				

$$Q_0^{next} = Q_0' = Q_0 \oplus 1$$



$Q_0 \backslash Q_2Q_1$	00	01	11	10
0		1	1	
1	1			1

$$Q_1^{next} = Q_1\, Q_0' + Q_1'\, Q_0 = Q_1 \oplus Q_0$$

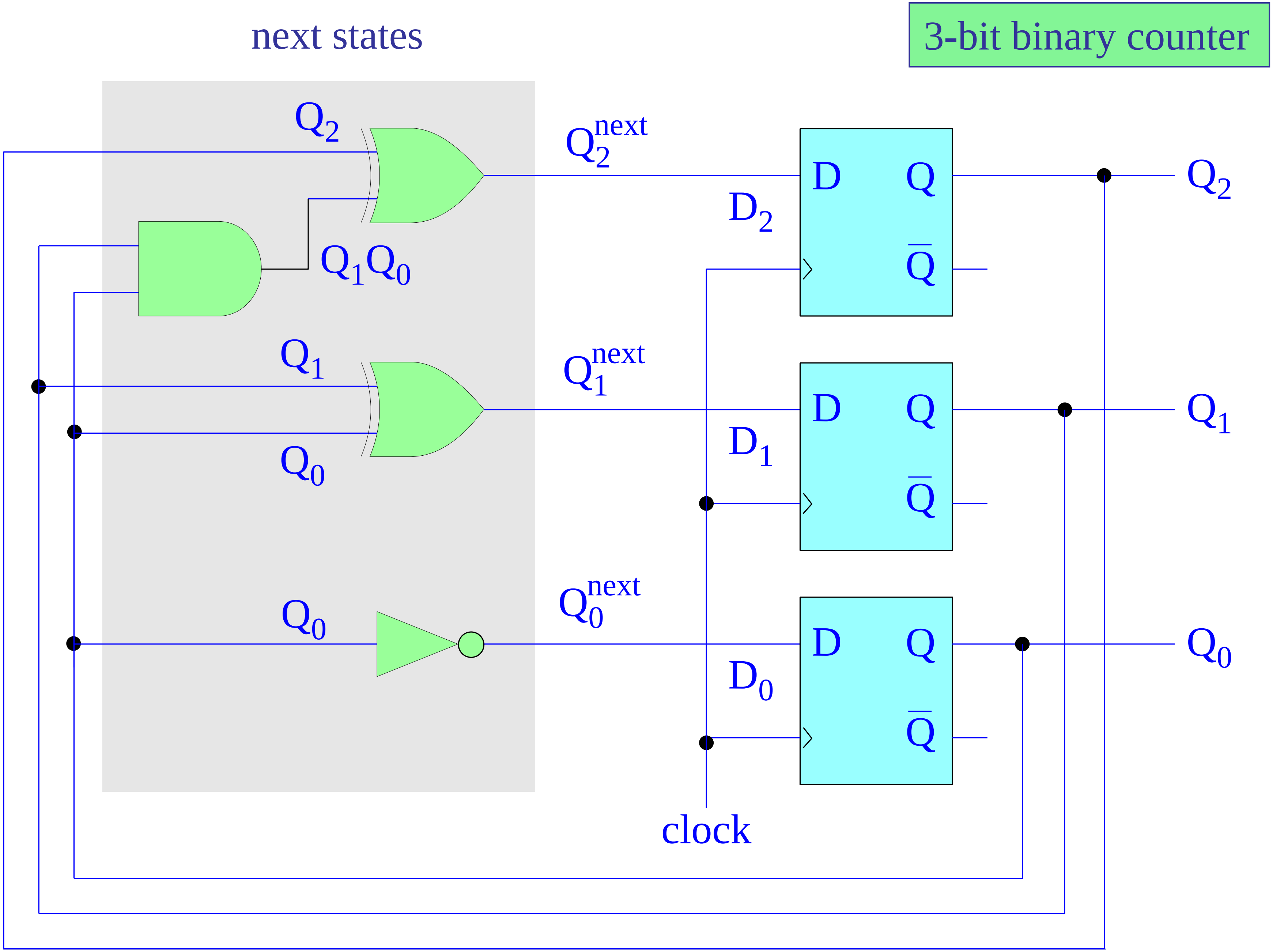


$Q_0 \backslash Q_2Q_1$	00	01	11	10
0			1	1
1		1		1

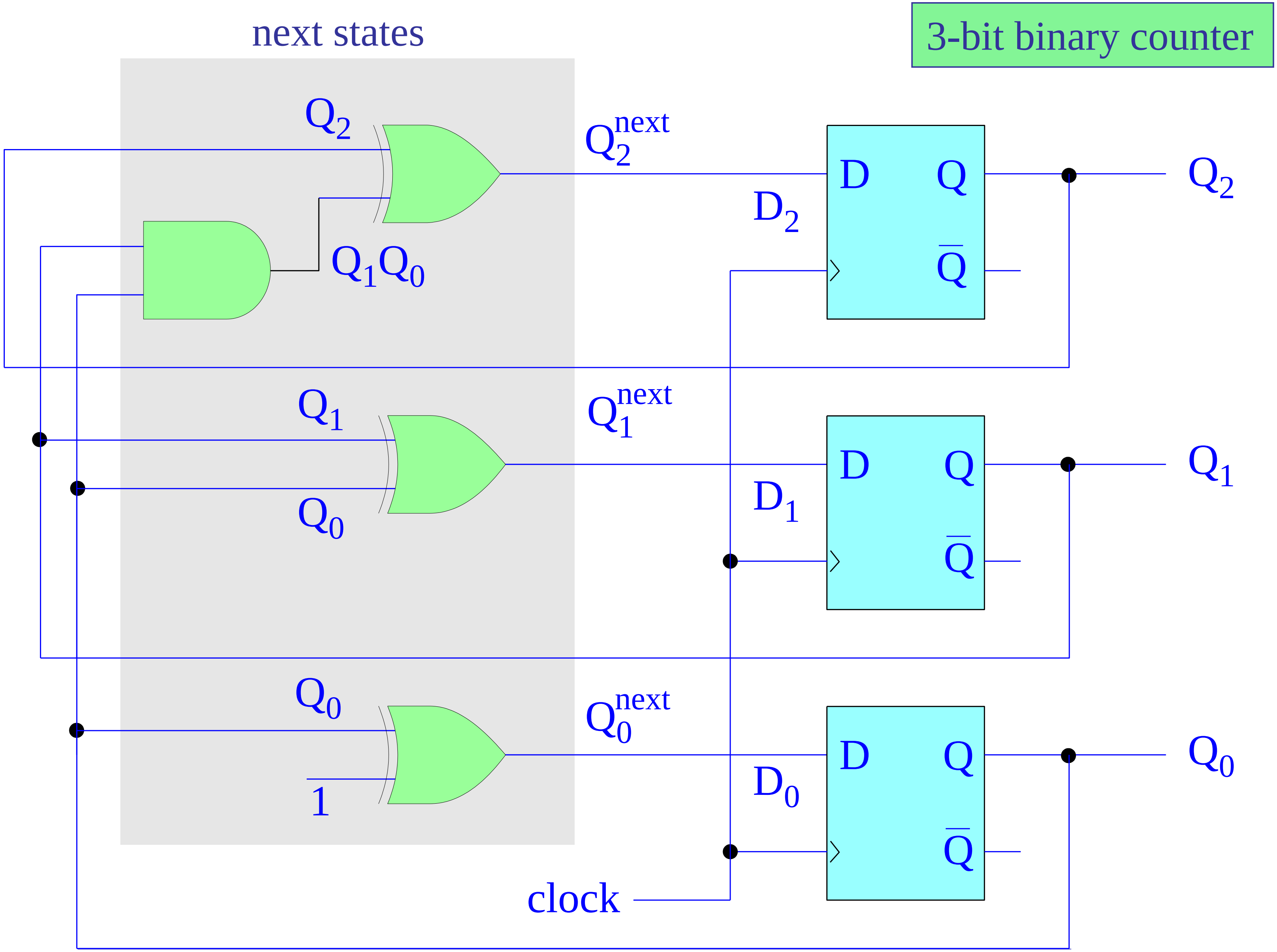
$$\begin{aligned} Q_2^{next} &= Q_2\, Q_0' + Q_2\, Q_1' + Q_2'\, Q_1\, Q_0 \\ &= Q_2\, (Q_0' + Q_1') + Q_2'\,(Q_1\, Q_0) \\ &= Q_2 \oplus (Q_1\, Q_0) \end{aligned}$$

Example 1

3-bit binary counter

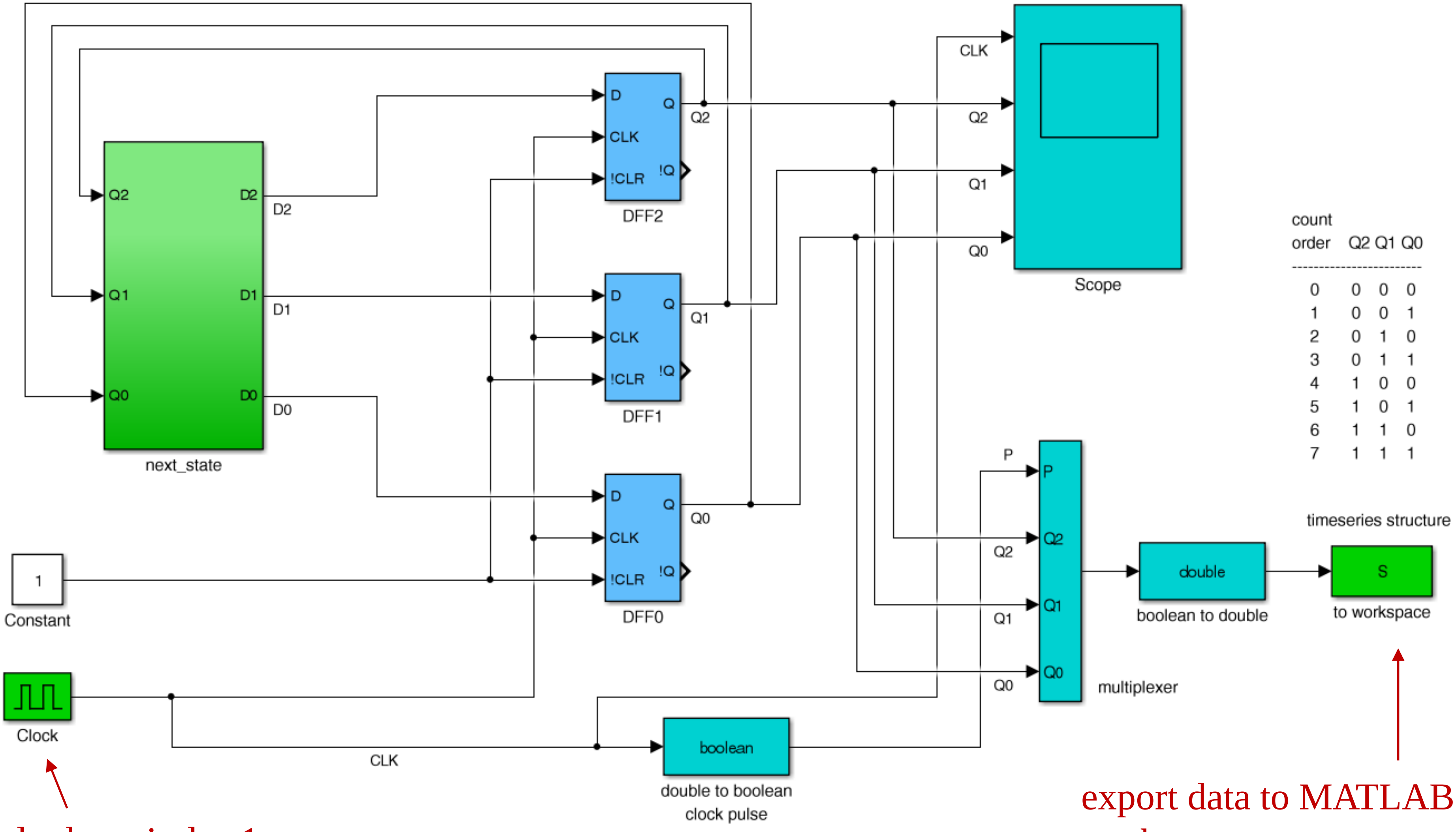


3-bit binary counter



Example 1

3-bit binary counter



count order	Q2	Q1	Q0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

clock period = 1

export data to MATLAB workspace

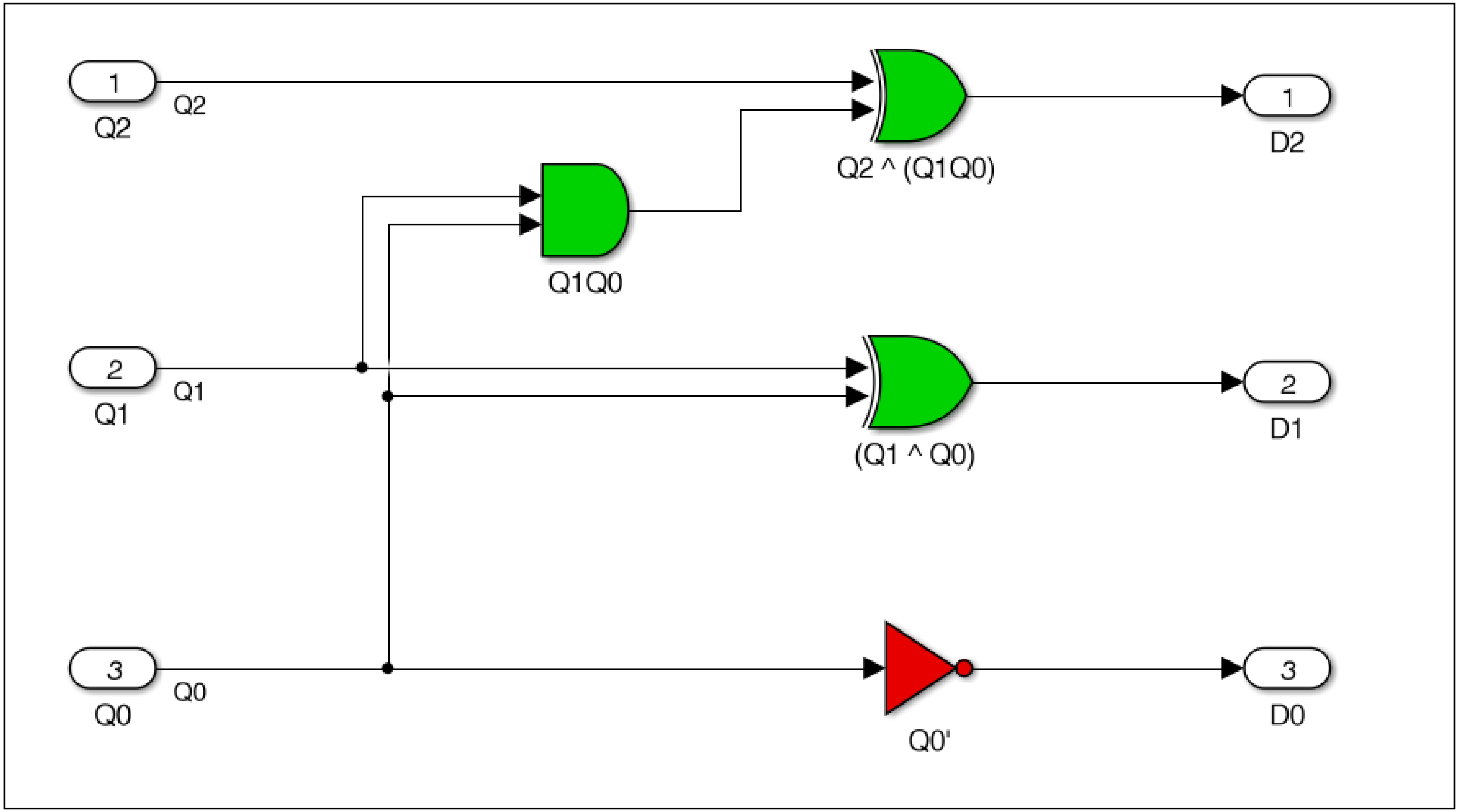
$$Q_2^{\text{next}} = Q_2 \oplus (Q_1\,Q_0)$$

$$Q_1^{\text{next}} = Q_1 \oplus Q_0$$

$$Q_0^{\text{next}} = Q_0 \oplus 1 = Q_0'$$

next-state sub-function

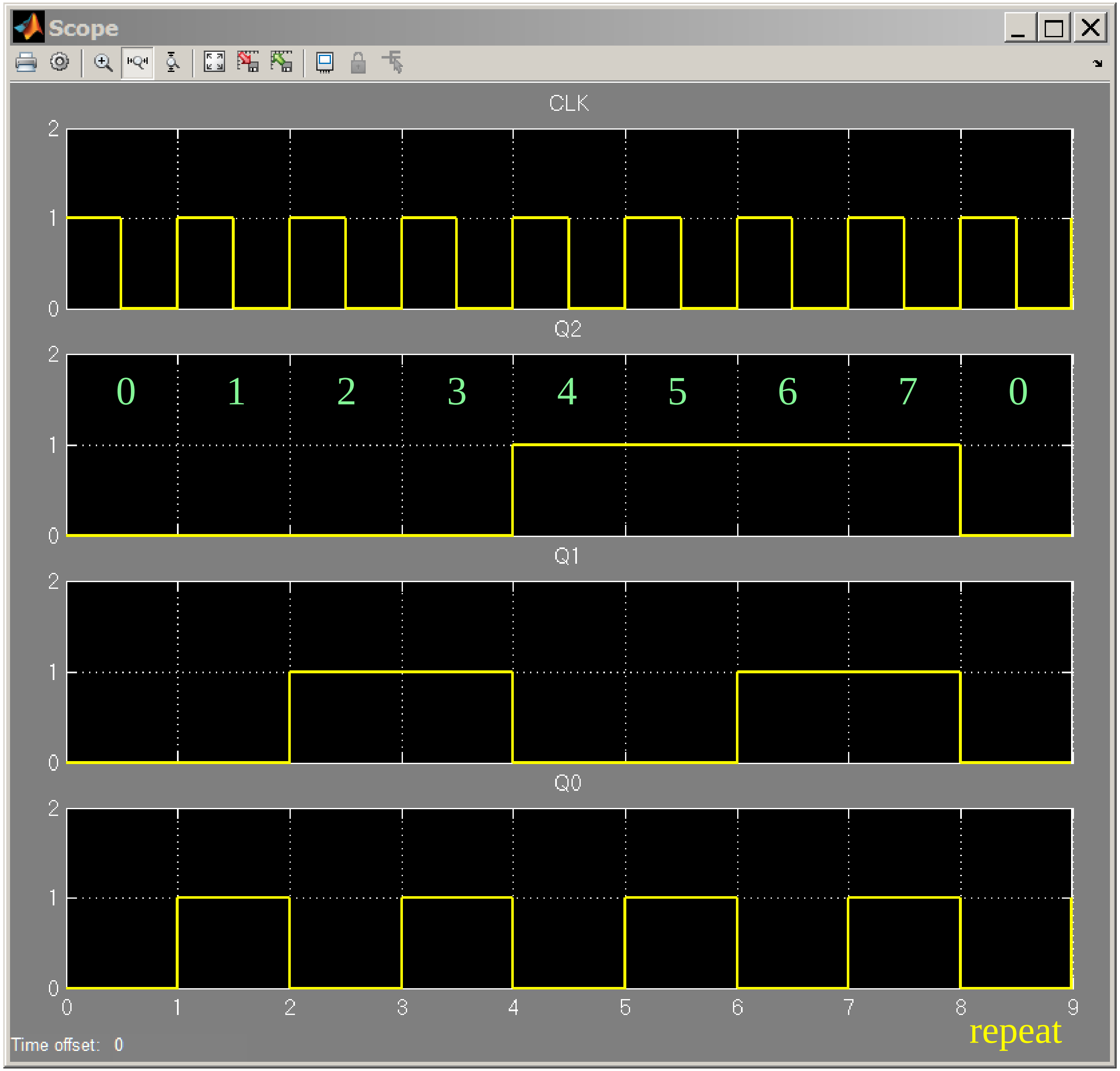
3-bit binary counter



Example 1

scope display

3-bit binary counter

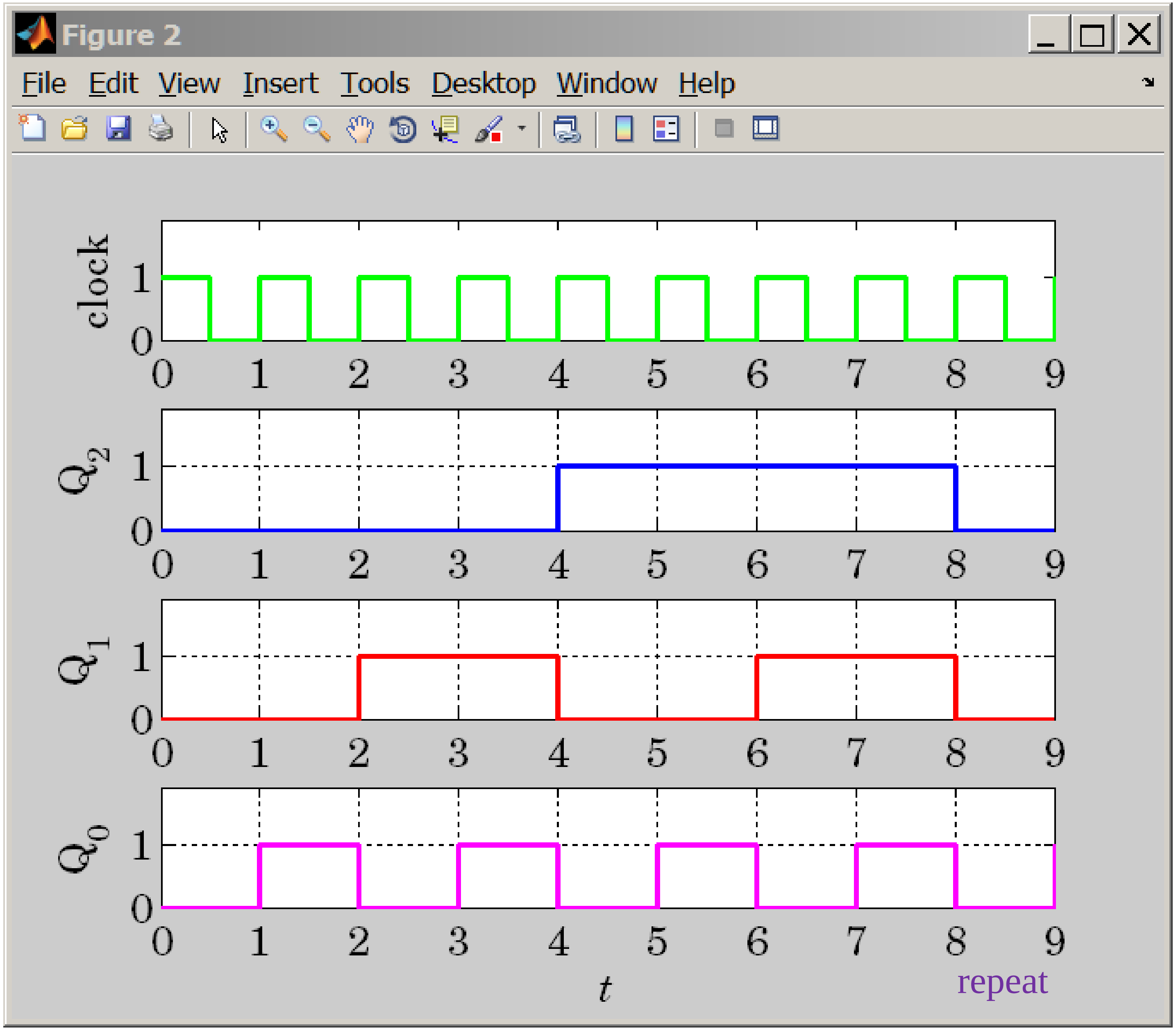


Example 1

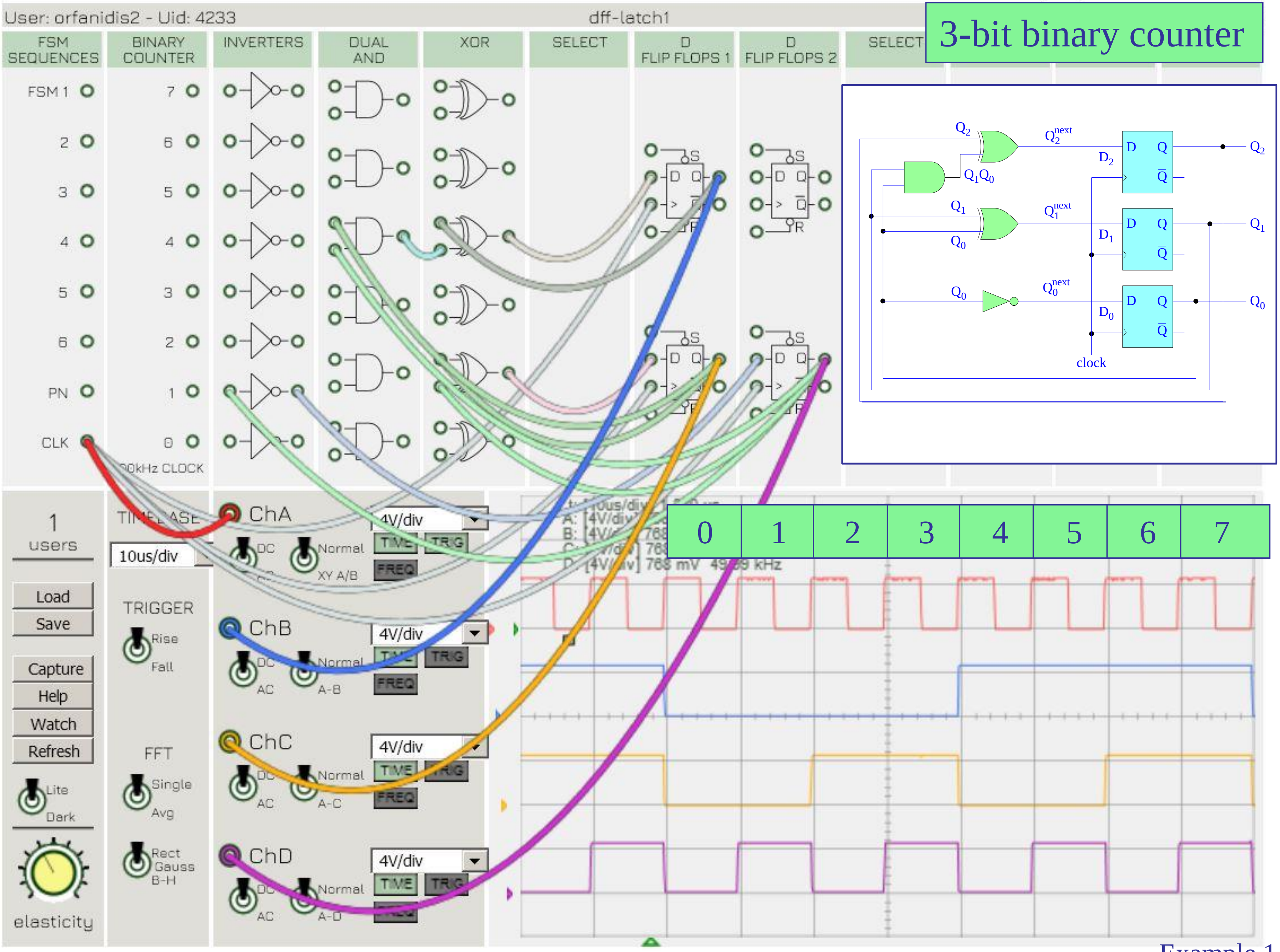
3-bit binary counter

```
% plot timing diagram from exported Simulink data

t =  S.time;        % time
P =  S.data(:,1);   % clock pulse
Q2 = S.data(:,2);
Q1 = S.data(:,3);
Q0 = S.data(:,4);

set(0,'DefaultAxesFontSize',15);
figure;
subplot(4,1,1); stairs(t,P,'g-');  ylabel('clock')
subplot(4,1,2); stairs(t,Q2,'b-'); ylabel('Q_2');
subplot(4,1,3); stairs(t,Q1,'r-'); ylabel('Q_1');
subplot(4,1,4); stairs(t,Q0,'m-'); ylabel('Q_0');
xlabel('{\itt}')
```

3-bit binary counter



3-bit binary counter



User: orfanidis2 - Uid: 4233

dff-latch1

TIMEBASE 10us/div

ChA 4V/div

ChB 4V/div

ChC 4V/div

ChD 4V/div

A: [4V/div] 768 mV

B: [4V/div] 768 mV

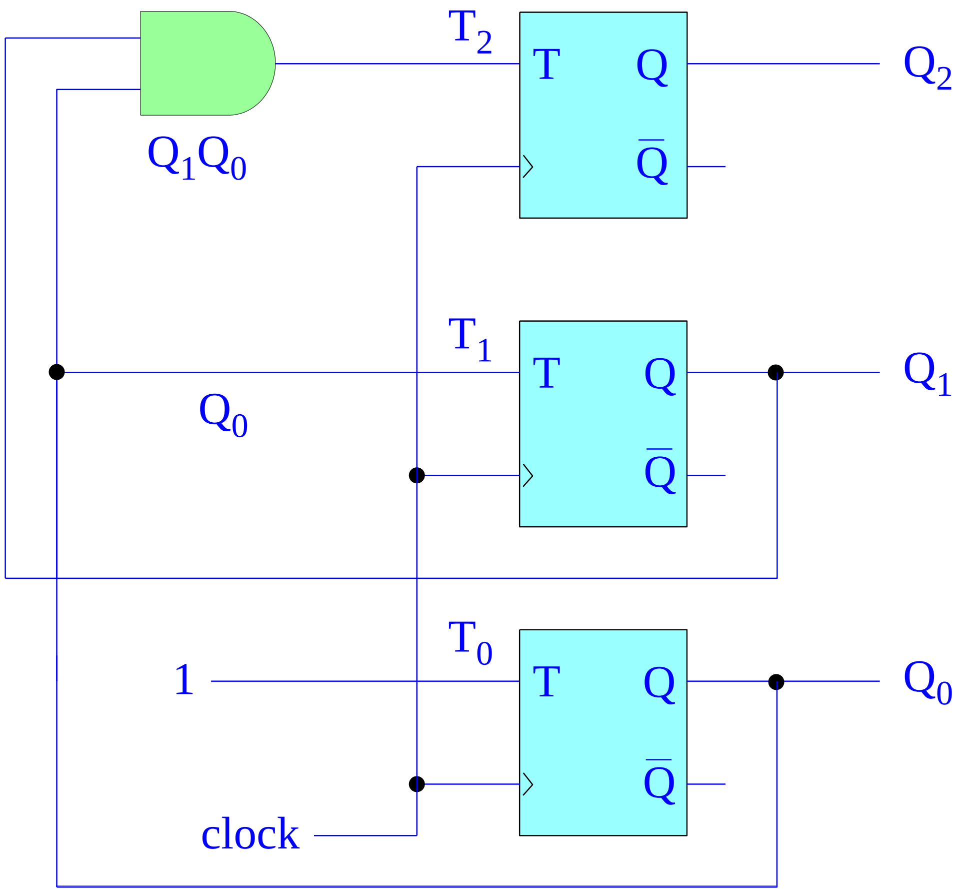
C: [4V/div] 768 mV

D: [4V/div] 768 mV 49.99 kHz

0	1	2	3	4	5	6	7

Example 1

3-bit binary counter using T-flip-flops



characteristic equation of T-flip-flop

$$Q^{next} = Q \oplus T$$

$$Q_2{}^{next} = Q_2 \oplus (Q_1\, Q_0) = Q_2 \oplus T_2$$

$$Q_1{}^{next} = Q_1 \oplus Q_0 = Q_1 \oplus T_1$$

$$Q_0{}^{next} = Q_0 \oplus 1 = Q_0{}' = Q_0 \oplus T_0$$

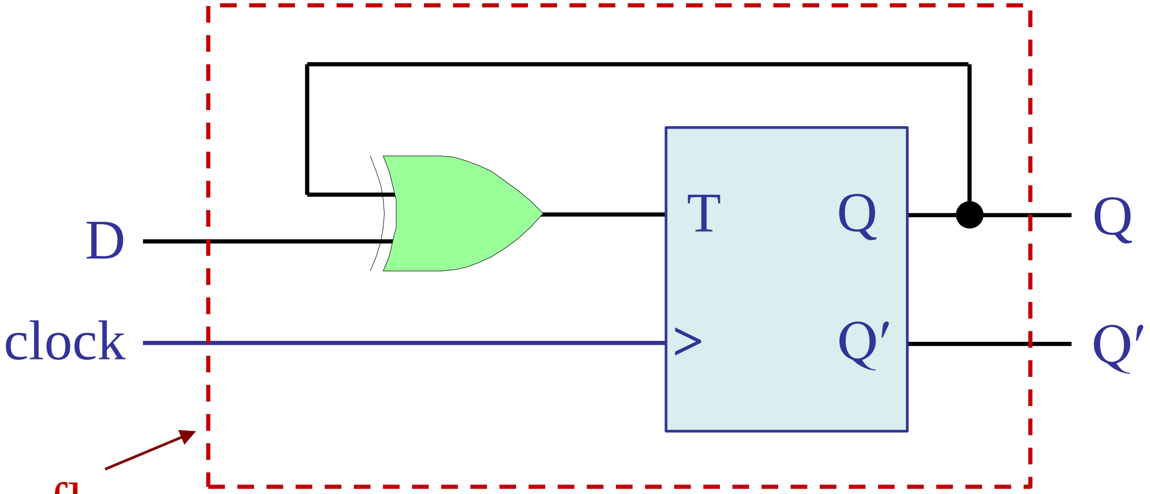
$$T_2 = Q_1\, Q_0$$

$$T_1 = Q_0$$

$$T_0 = 1$$

Example 1

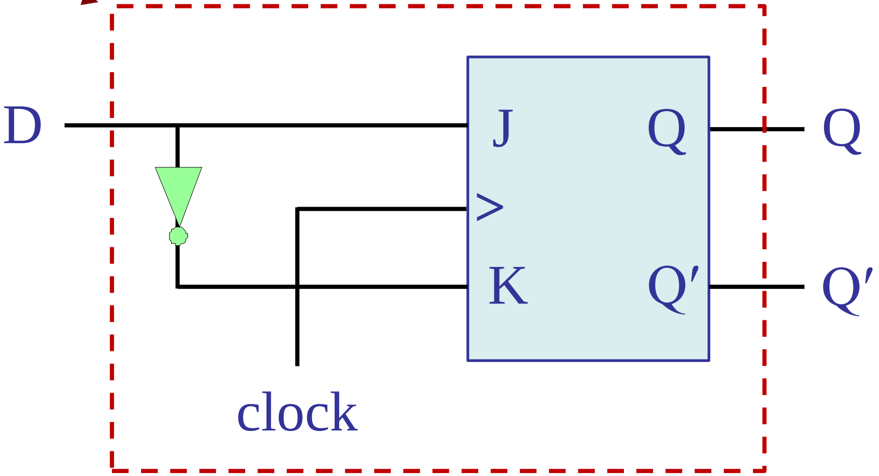
A general procedure for using T or JK flip-flops instead of D flip-flops is to carry out the design using D flip-flops, then, replace each D flip-flop by a T or a JK flip-flop as shown below (effectively converting them to D flip-flops, see unit-6)



$T = D \oplus Q$

$Q_{next} = T \oplus Q = D$

D flip-flops



$J = D$

$K = D'$

$Q_{next} = J\,Q' + K'\,Q = D$

Example 2 – four-bit binary counter. Derive and implement with logic gates the next-state logic equations,

$$Q_3^{\text{next}} = Q_3 \oplus (Q_2\, Q_1\, Q_0)$$

$$Q_2^{\text{next}} = Q_2 \oplus (Q_1\, Q_0)$$

$$Q_1^{\text{next}} = Q_1 \oplus Q_0$$

$$Q_0^{\text{next}} = Q_0 \oplus 1 = Q_0'$$

for an ordinary 4-bit binary counter, with a periodically repeating length-8 sequence, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15].

using D flip-flops

$D_3 = Q_3^{\text{next}}$

$D_2 = Q_2^{\text{next}}$

$D_1 = Q_1^{\text{next}}$

$D_0 = Q_0^{\text{next}}$

using T flip-flops

$T_3 = Q_2\, Q_1\, Q_0$

$T_2 = Q_1\, Q_0$

$T_1 = Q_0$

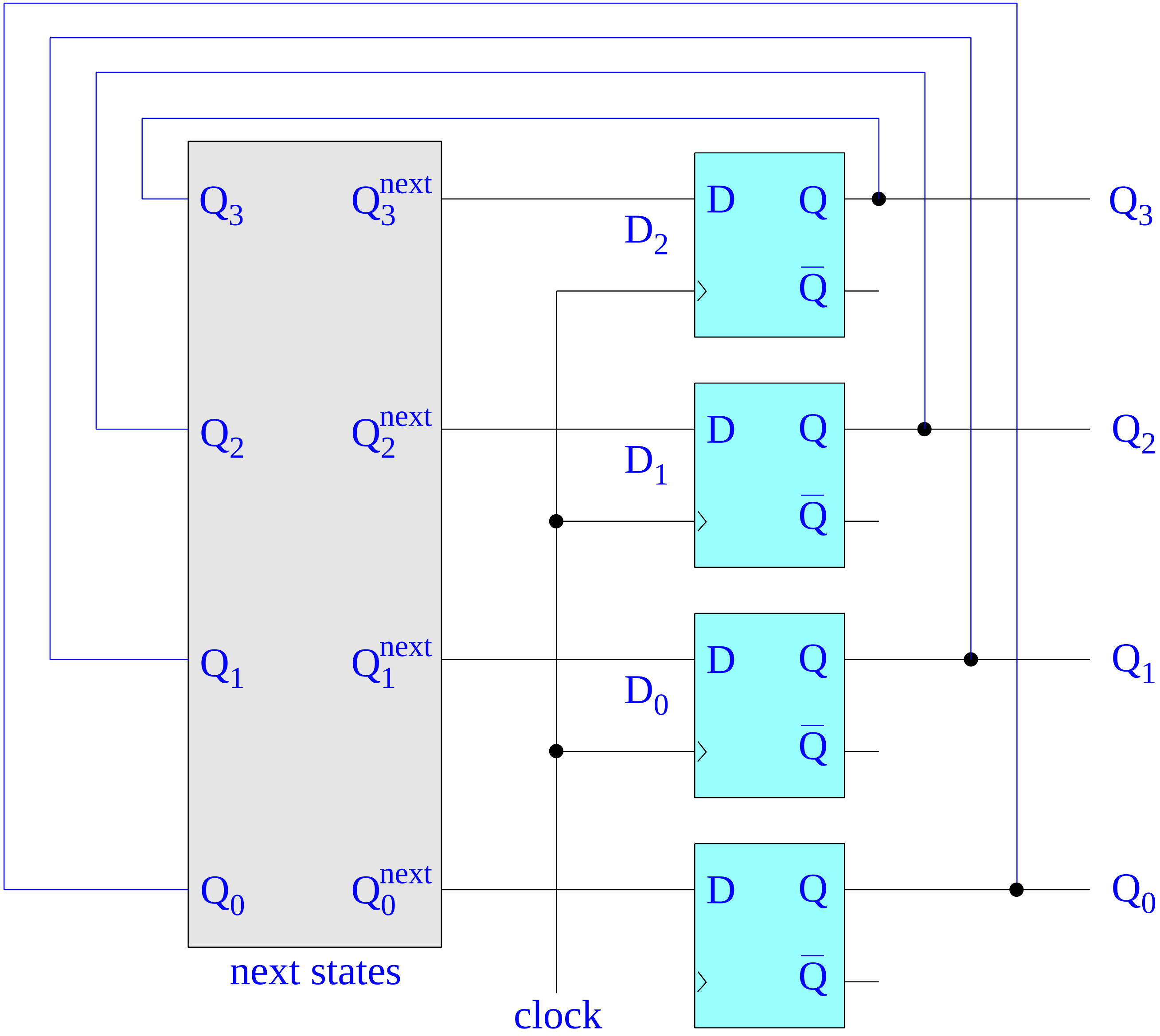
$T_0 = 1$

4-bit binary counter

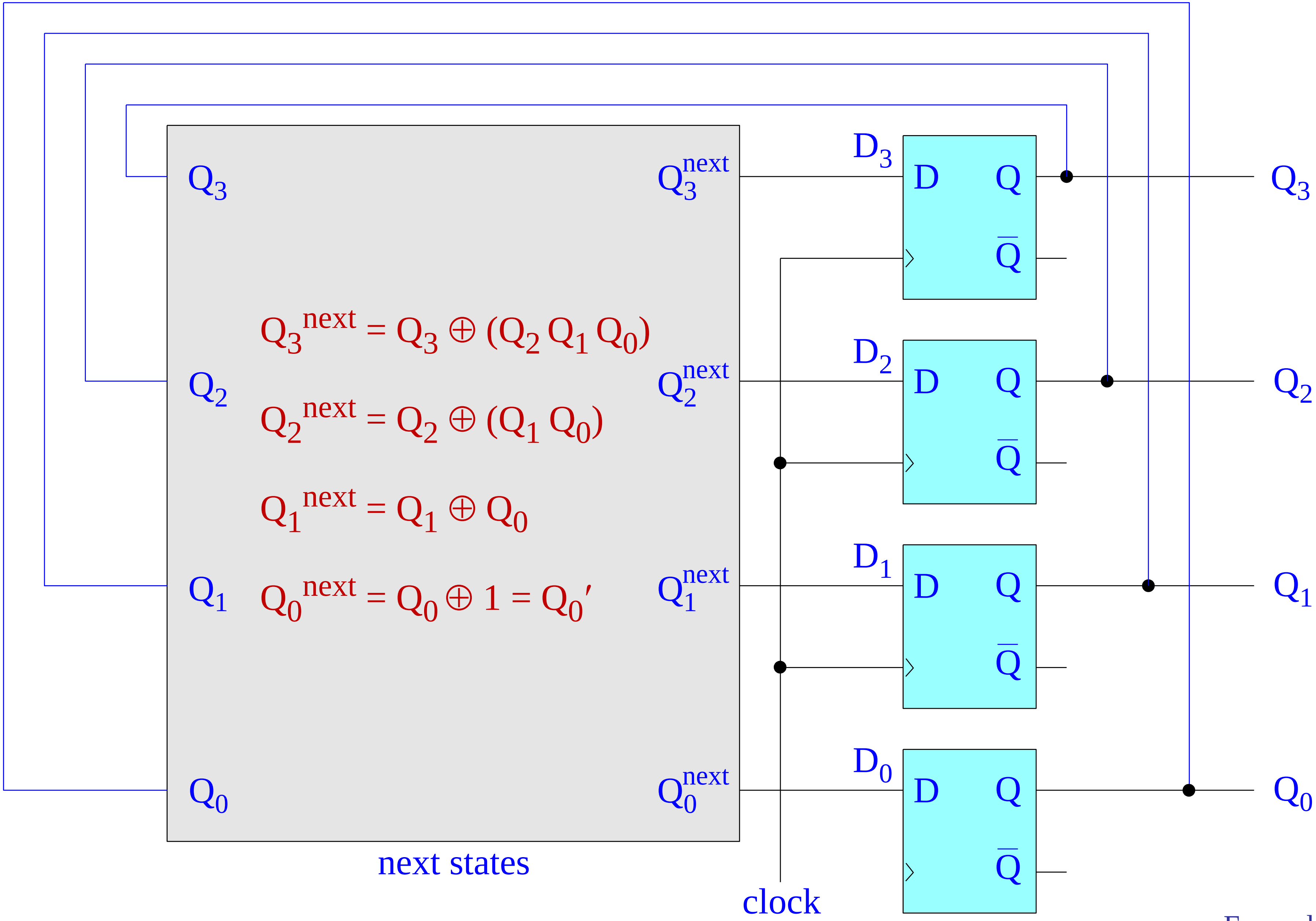
characteristic table

					next states				
s_t	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	s_{t+1}
0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	1	0	2
2	0	0	1	0	0	0	1	1	3
3	0	0	1	1	0	1	0	0	4
4	0	1	0	0	0	1	0	1	5
5	0	1	0	1	0	1	1	0	6
6	0	1	1	0	0	1	1	1	7
7	0	1	1	1	1	0	0	0	8
8	1	0	0	0	1	0	0	1	9
9	1	0	0	1	1	0	1	0	10
10	1	0	1	0	1	0	1	1	11
11	1	0	1	1	1	1	0	0	12
12	1	1	0	0	1	1	0	1	13
13	1	1	0	1	1	1	1	0	14
14	1	1	1	0	1	1	1	1	15
15	1	1	1	1	0	0	0	0	0

4-bit binary counter



4-bit binary counter



$$Q_3^{next} = Q_3 \oplus (Q_2\, Q_1\, Q_0)$$

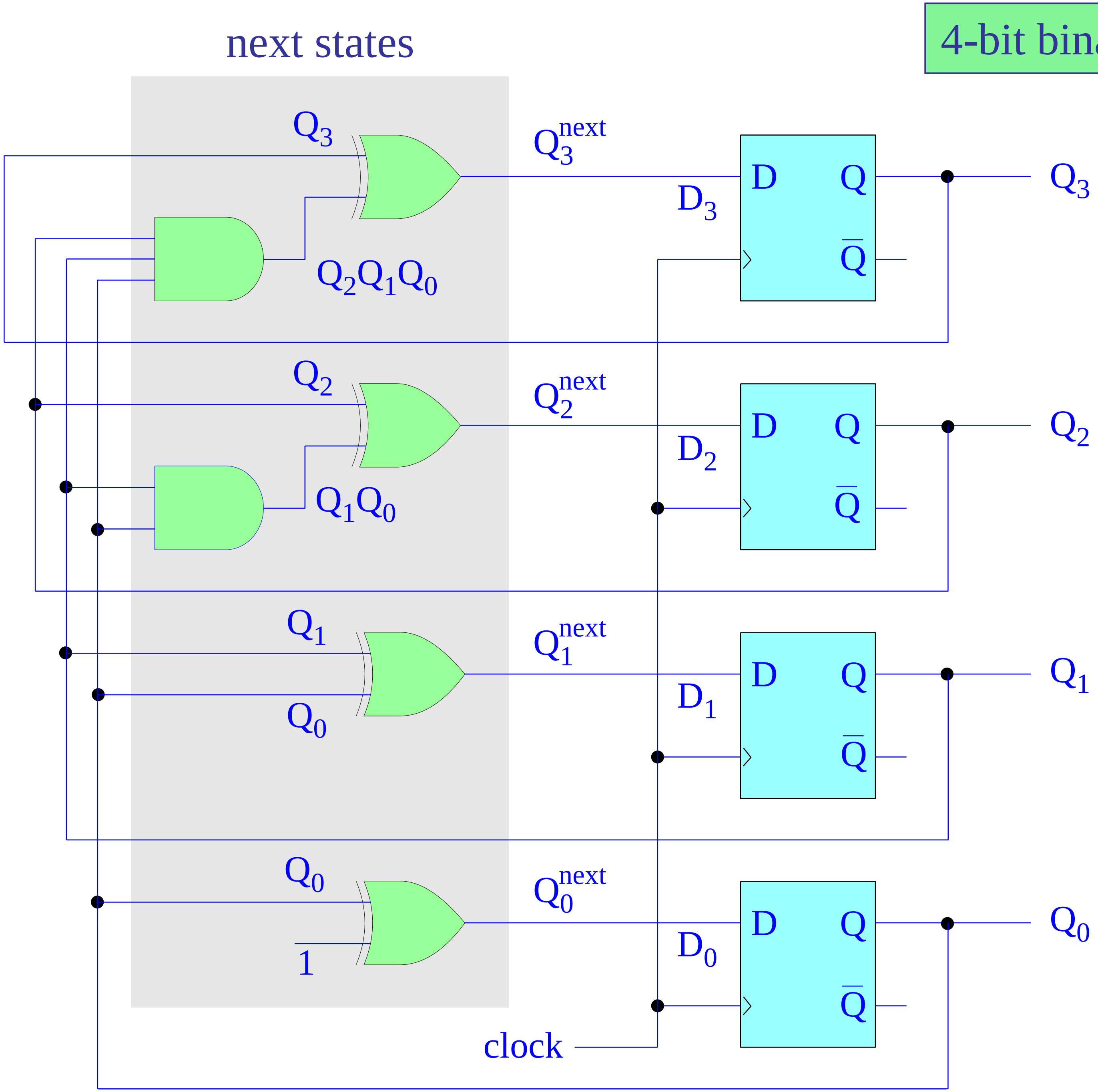
$$Q_2^{next} = Q_2 \oplus (Q_1\, Q_0)$$

$$Q_1^{next} = Q_1 \oplus Q_0$$

$$Q_0^{next} = Q_0 \oplus 1 = Q_0'$$

Example 2

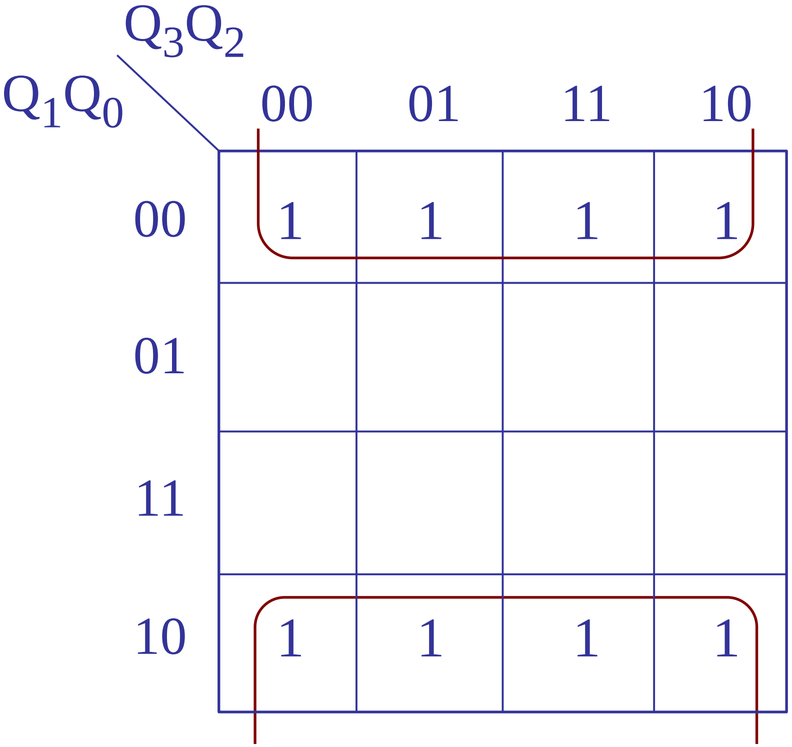
4-bit binary counter



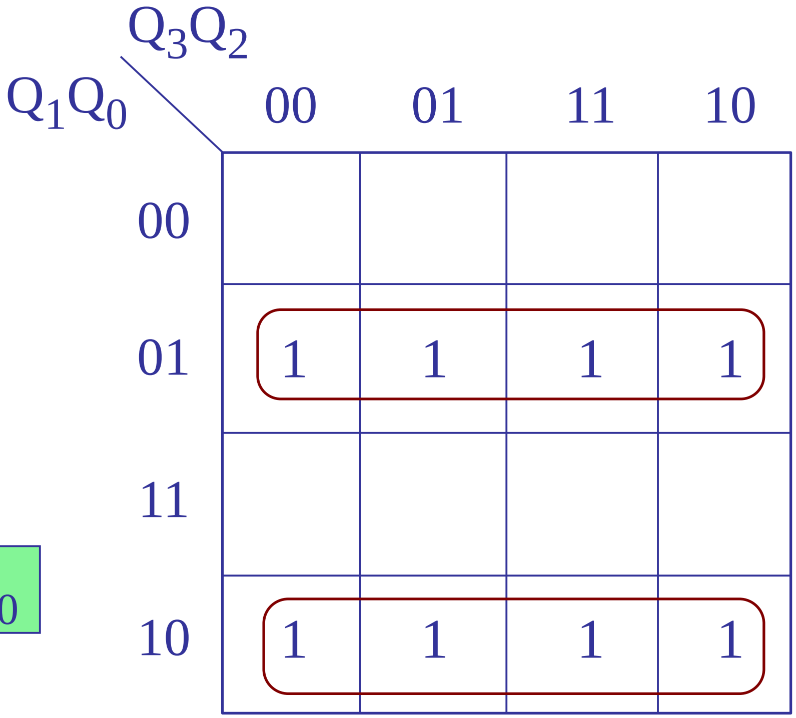
Example 2

4-bit binary counter

$Q_0^{next} = Q_0' = Q_0 \oplus 1$



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	1	1	1
01				
11				
10	1	1	1	1

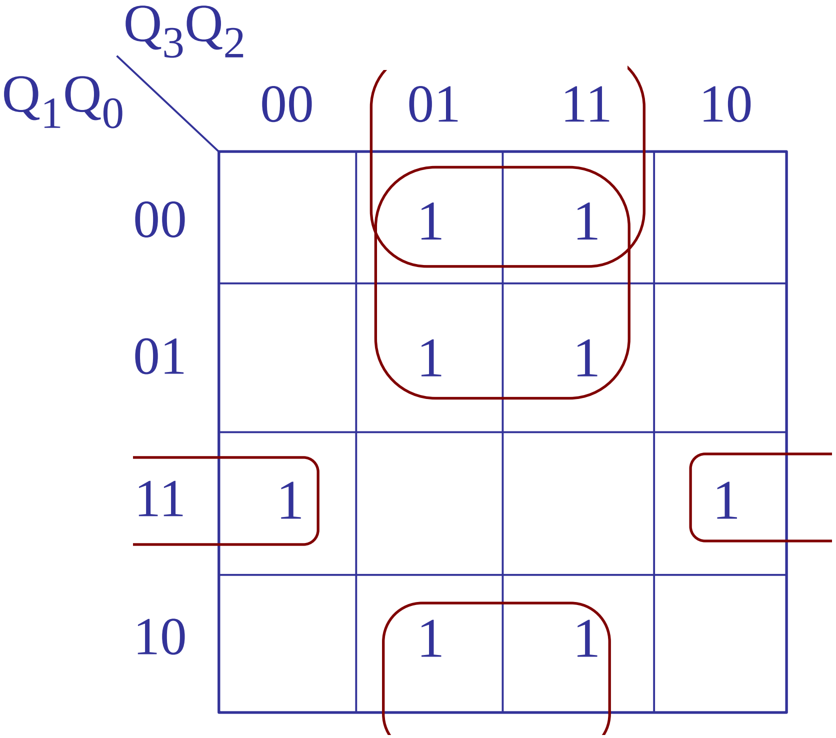


Q_1Q_0 \ Q_3Q_2	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$Q_1^{next} = Q_1 Q_0' + Q_1' Q_0 = Q_1 \oplus Q_0$

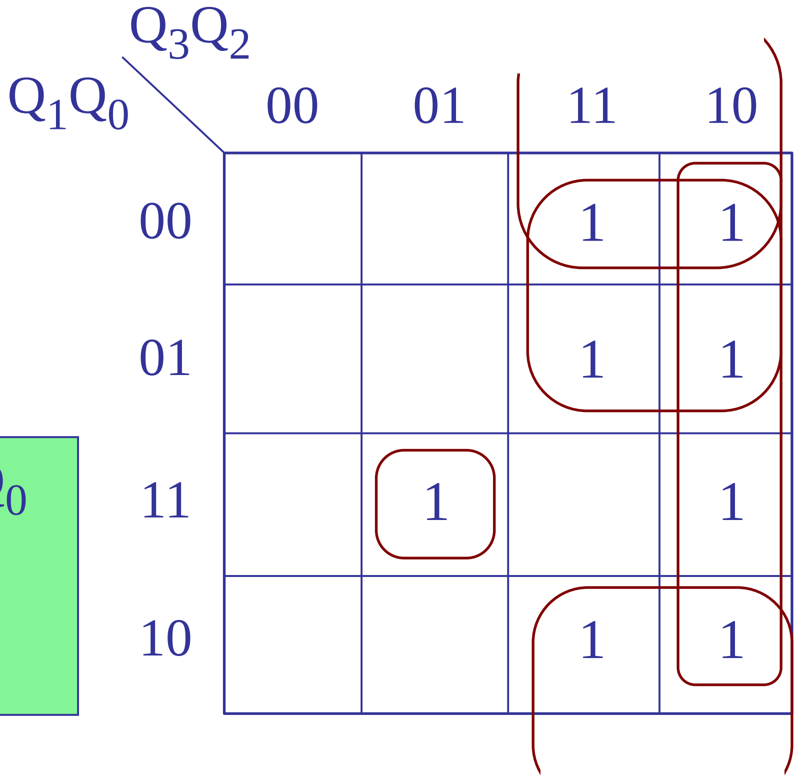
Example 2

4-bit binary counter



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00		1	1	
01		1	1	
11	1			1
10		1	1	

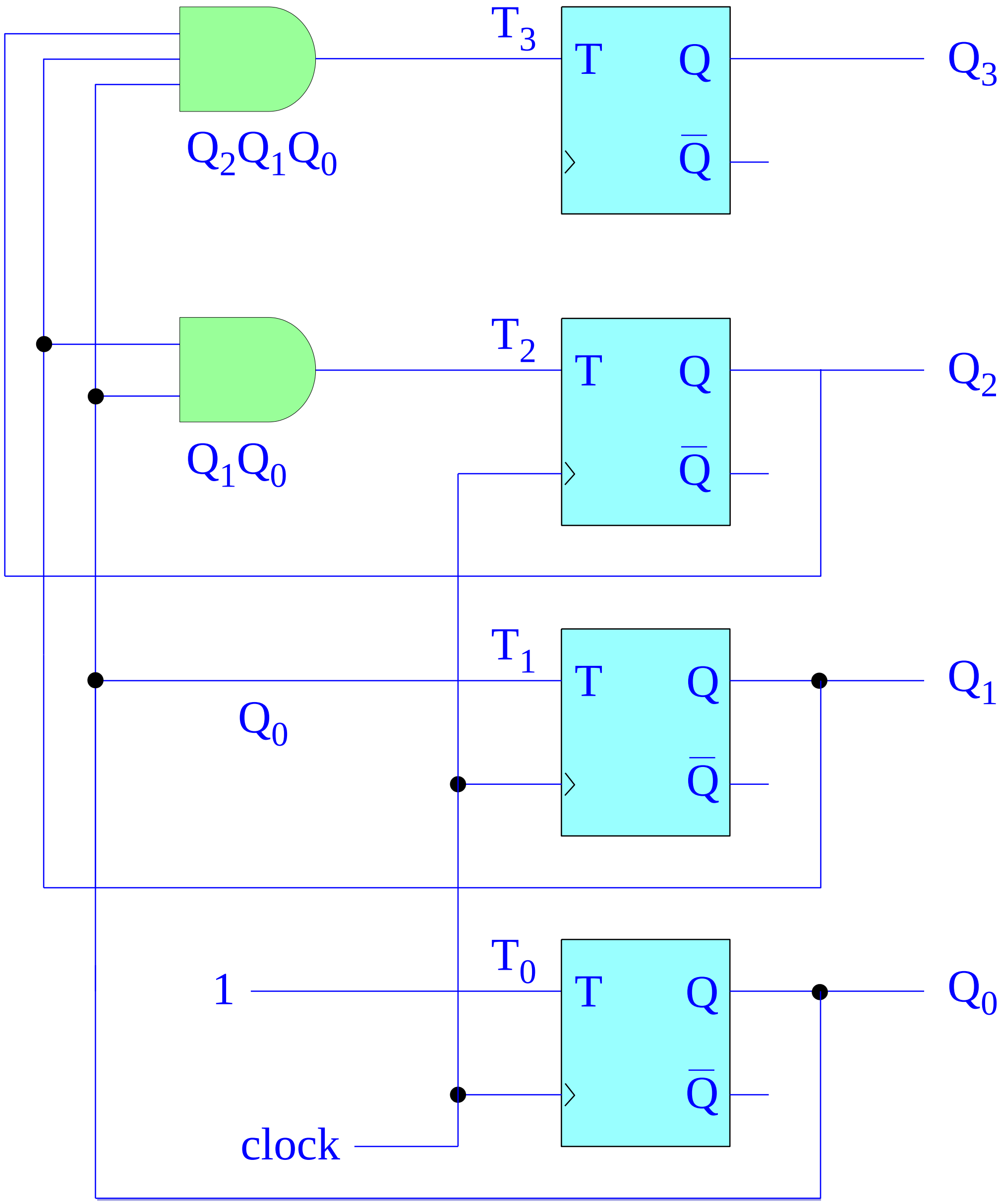
$$Q_2^{next} = Q_2\, Q_1' + Q_2\, Q_0' \; + Q_2'\, Q_1\, Q_0$$
$$= Q_2\, (Q_1' + Q_0') \; + Q_2'\, (Q_1\, Q_0)$$
$$= Q_2 \oplus (Q_1\, Q_0)$$



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00			1	1
01			1	1
11		1		1
10			1	1

$$Q_3^{next} = Q_3\, Q_1' + Q_3\, Q_2' + Q_3\, Q_0' + Q_3'\, Q_2\, Q_1\, Q_0$$
$$= Q_3\, (Q_1' + Q_2' \; + Q_0') \; + Q_3'\, Q_2\, Q_1\, Q_0$$
$$= Q_3 \oplus (Q_2\, Q_1\, Q_0)$$

Example 2



4-bit binary counter using T-flip-flops

characteristic equation of T-flip-flop

$$Q^{next} = Q \oplus T$$

↓

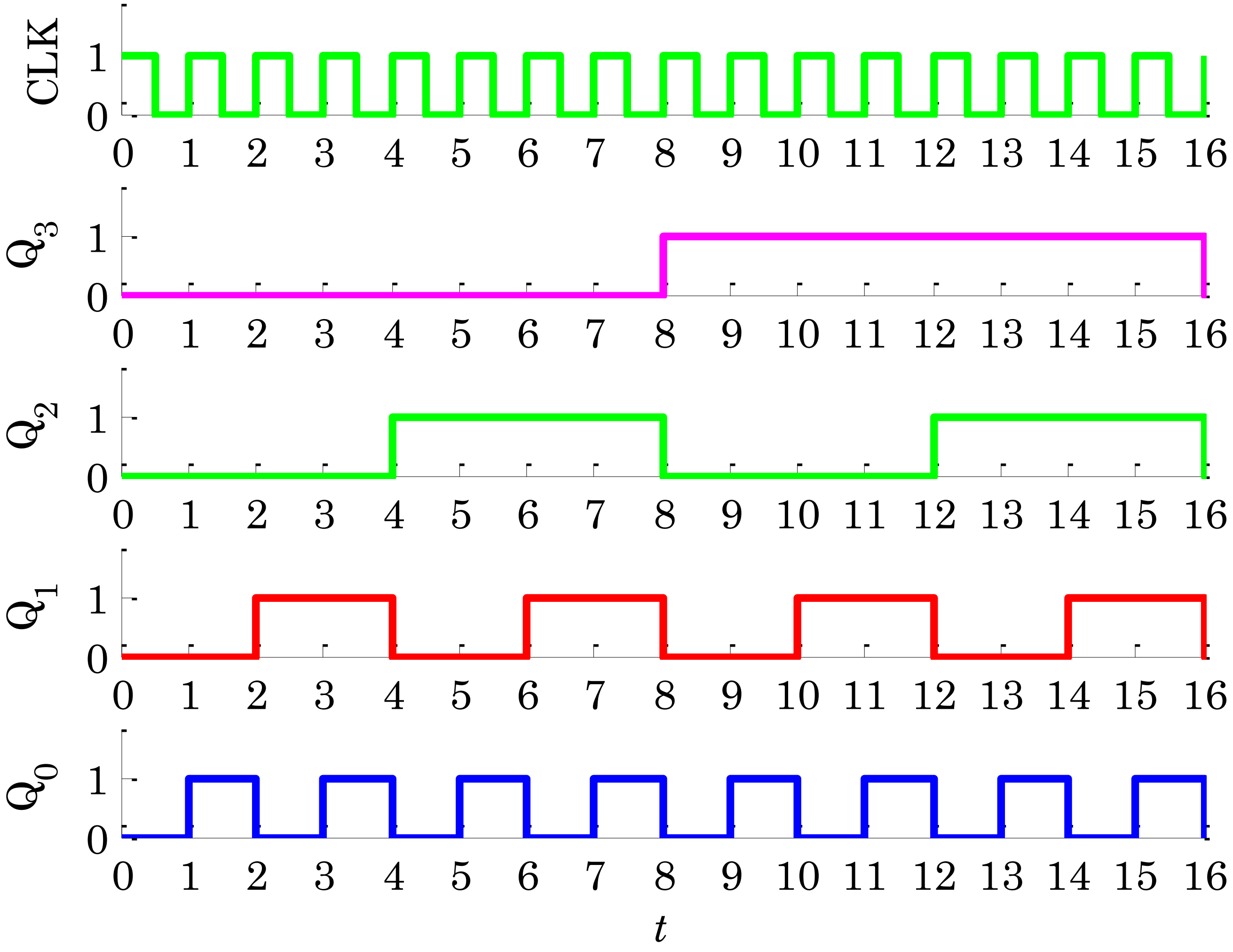
$$T_3 = Q_2\, Q_1\, Q_0$$

$$T_2 = Q_1\, Q_0$$

$$T_1 = Q_0$$

$$T_0 = 1$$

4-bit binary counter

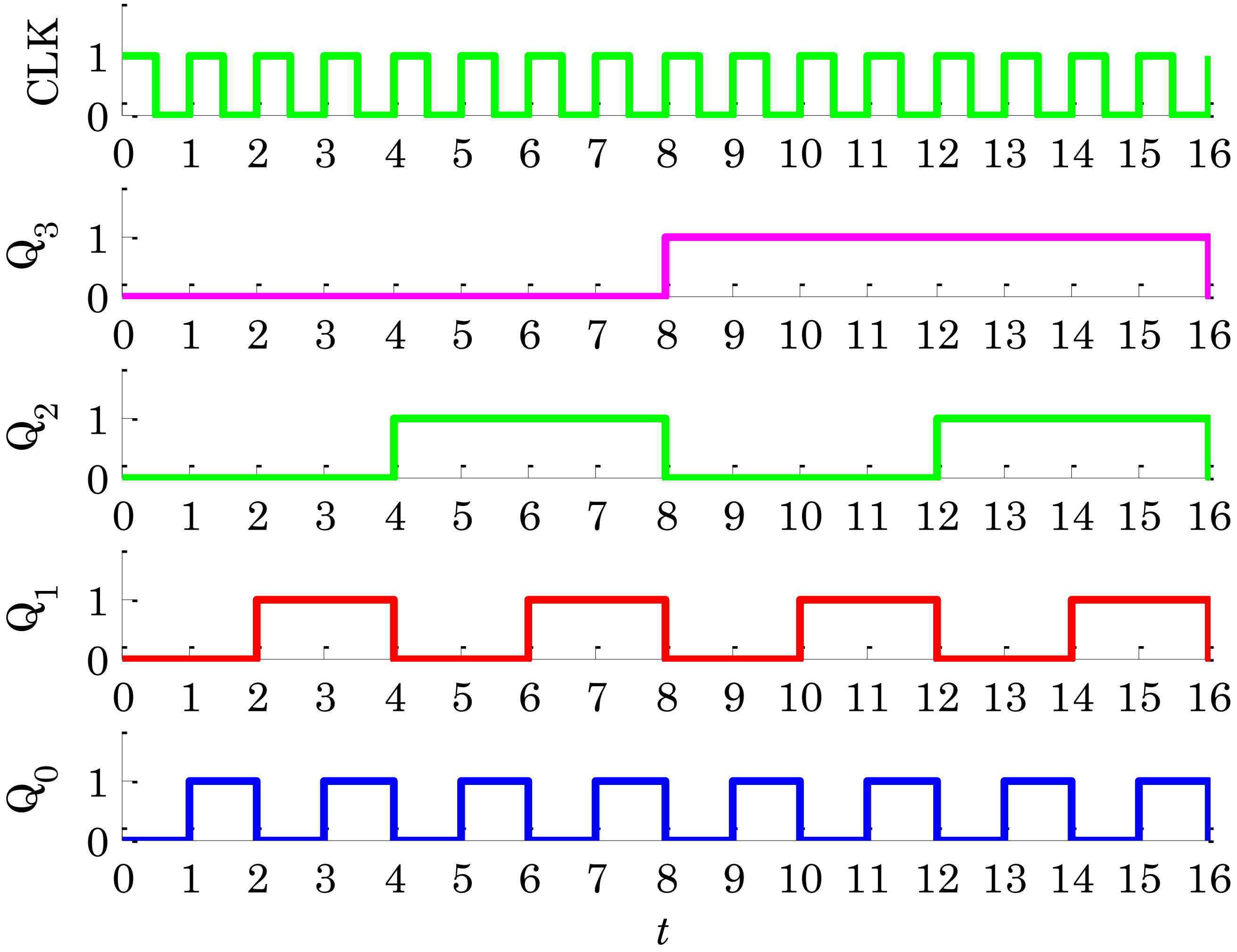


<span style="color:#c00000">Note on clock dividers:</span> the individual bit-outputs can be used as clock signals of lower frequencies. For an N-bit binary counter,

Q_n has $1/2^{n+1}$, the frequency of the clock, for n=0,1,…, N-1

Example 2

4-bit binary counter



Example: 100 kHz Emona clock divided down to 100 Hz.

use 10-bit counter and pick Q_9 bit as clock:

100 kHz / 2^{9+1} = = 97.6563 Hz

with, 102.4 kHz = = 1024 x 100 Hz

1024 x 100 / 2^{9+1} = = 100 Hz

Note on clock dividers: the individual bit-outputs can be used as clock signals of lower frequencies. For an N-bit binary counter,

Q_n has $1/2^{n+1}$, the frequency of the clock, for n=0,1,…, N-1

Example 3 – traffic light controller. Counters can be used to generate time intervals for task control. Here, we will use a 3-bit counter to generate the **red**, **green**, **yellow** signals of a traffic light. Since a 3-bit counter repeats every 8 time periods, we will assume for simplicity that the green light stays on for 4 periods, then, the yellow light comes on for 1 period, and then the red light comes on for 3 periods, and the whole process repeats.

We start by translating these design requirements into a truth table.

				next states			outputs		
t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	**G**	**Y**	**R**
0	0	0	0	0	0	1	**1**	0	0
1	0	0	1	0	1	0	**1**	0	0
2	0	1	0	0	1	1	**1**	0	0
3	0	1	1	1	0	0	**1**	0	0
4	1	0	0	1	0	1	0	**1**	0
5	1	0	1	1	1	0	0	0	**1**
6	1	1	0	1	1	1	0	0	**1**
7	1	1	1	0	0	0	0	0	**1**

3-bit counter

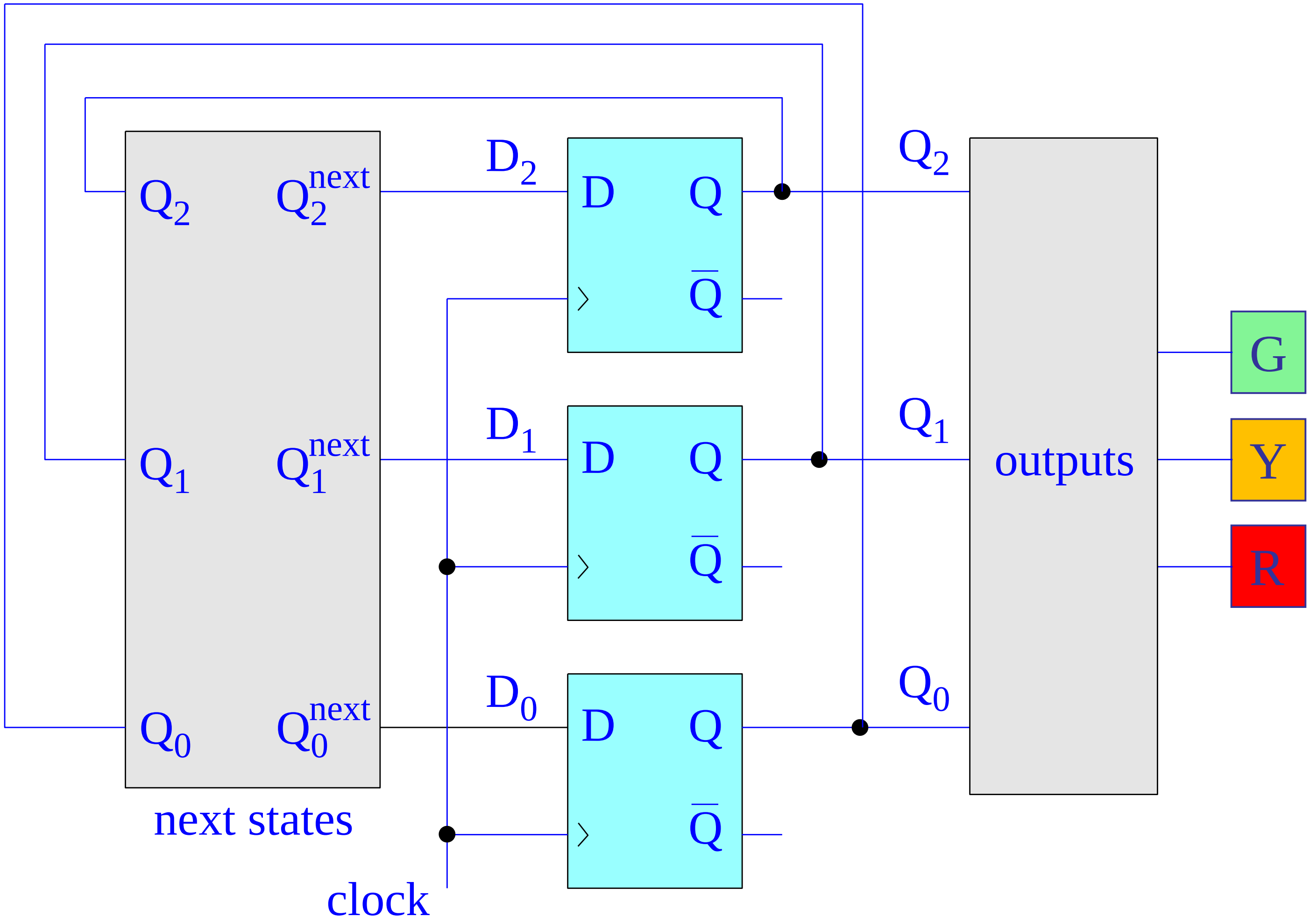
→

$$G = Q_2'$$

$$Y = Q_2 Q_1' Q_0'$$

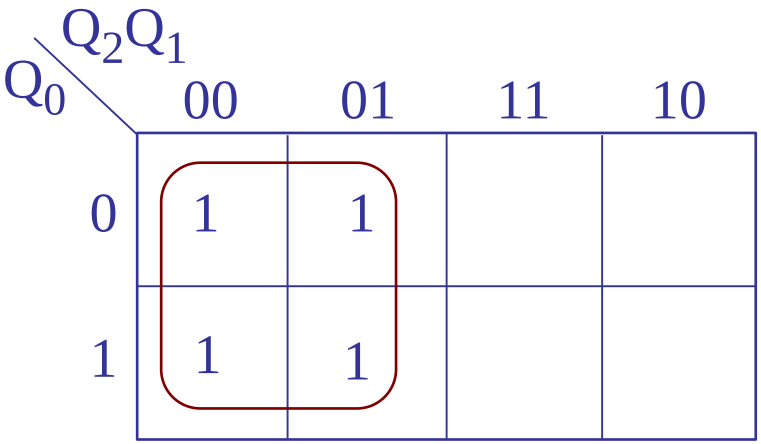
$$R = Q_2 Q_1 + Q_2 Q_0$$

the counter generates the sequence of states, $Q_2Q_1Q_0$, which, in turn, generate the green, yellow, red output signals

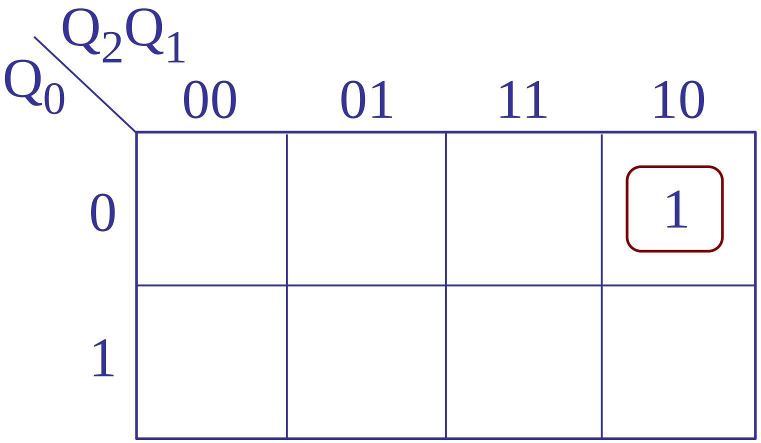


Example 3

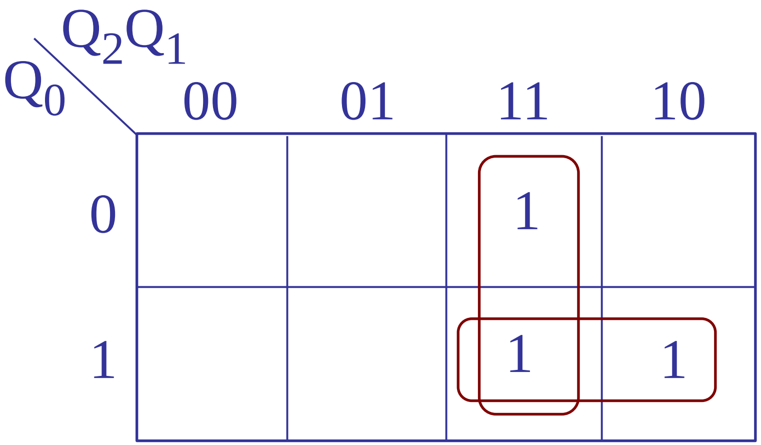
				next states			outputs		
t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	G	Y	R
0	0	0	0	0	0	1	1	0	0
1	0	0	1	0	1	0	1	0	0
2	0	1	0	0	1	1	1	0	0
3	0	1	1	1	0	0	1	0	0
4	1	0	0	1	0	1	0	1	0
5	1	0	1	1	1	0	0	0	1
6	1	1	0	1	1	1	0	0	1
7	1	1	1	0	0	0	0	0	1



$G = Q_2'$



$Y = Q_2\, Q_1'\, Q_0'$

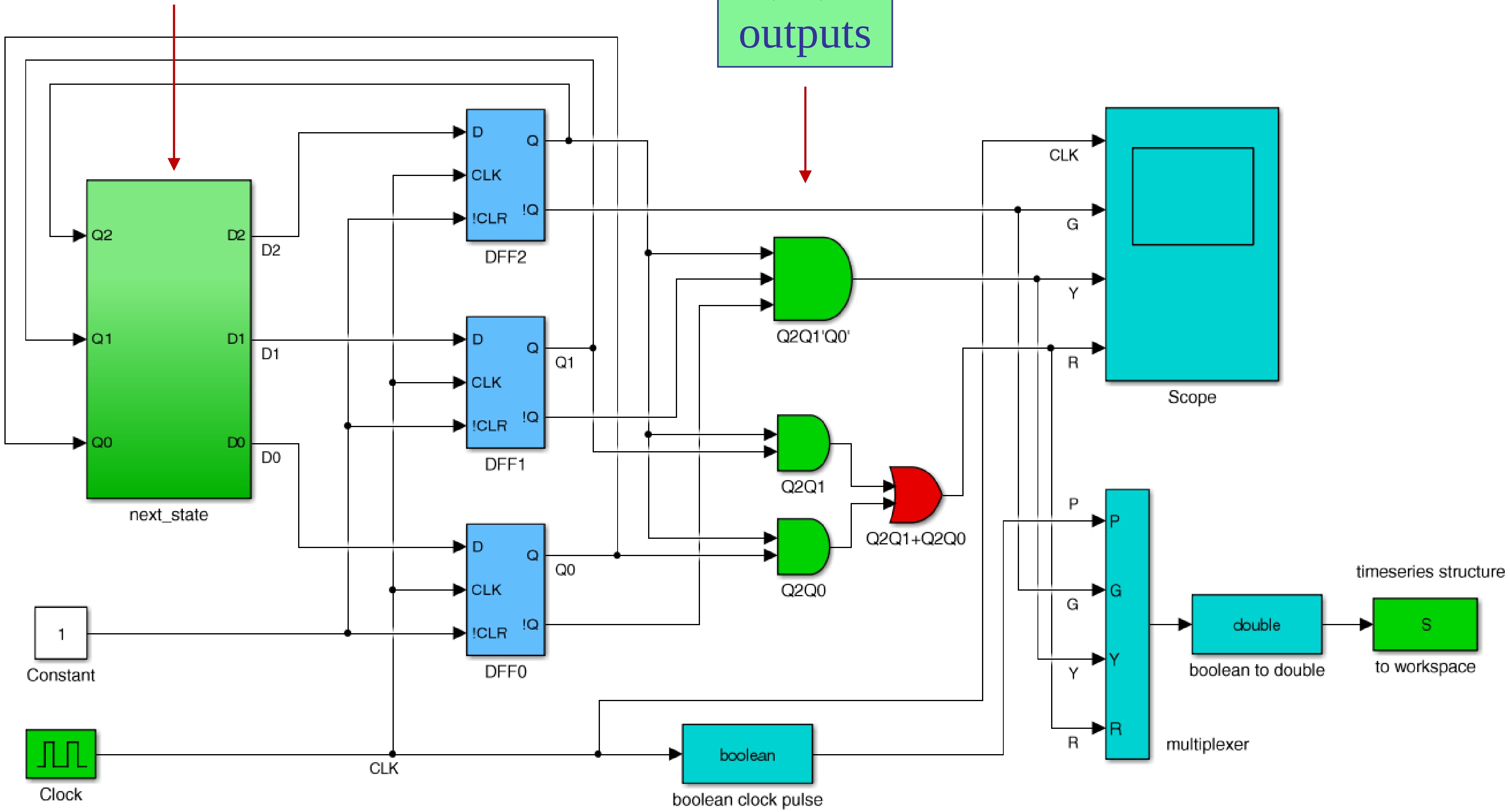


$R = Q_2\, Q_1 + Q_2\, Q_0$

Example 3

3-bit counter

G, Y, R outputs



Example 3

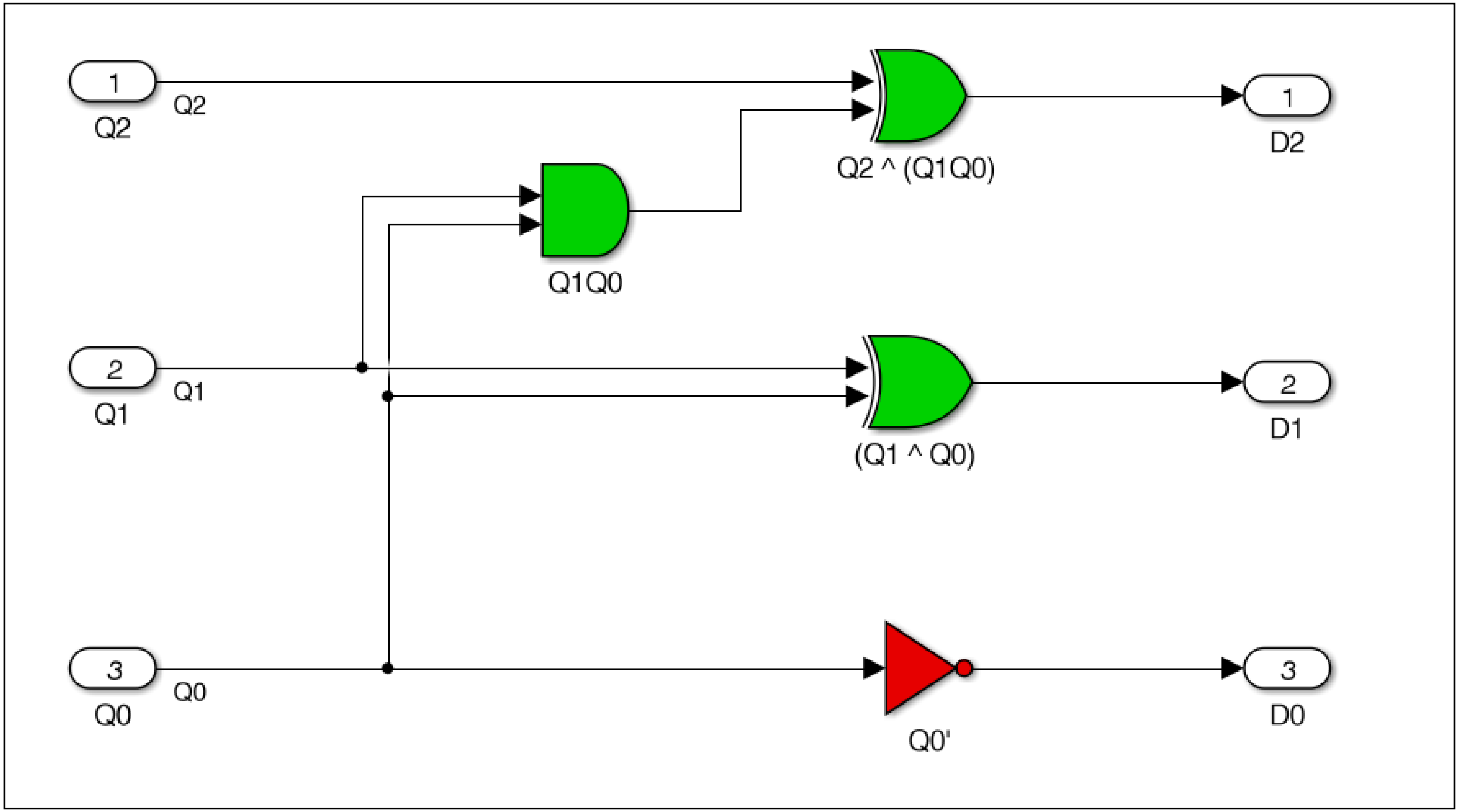
$$Q_2^{\text{next}} = Q_2 \oplus (Q_1\, Q_0)$$

$$Q_1^{\text{next}} = Q_1 \oplus Q_0$$

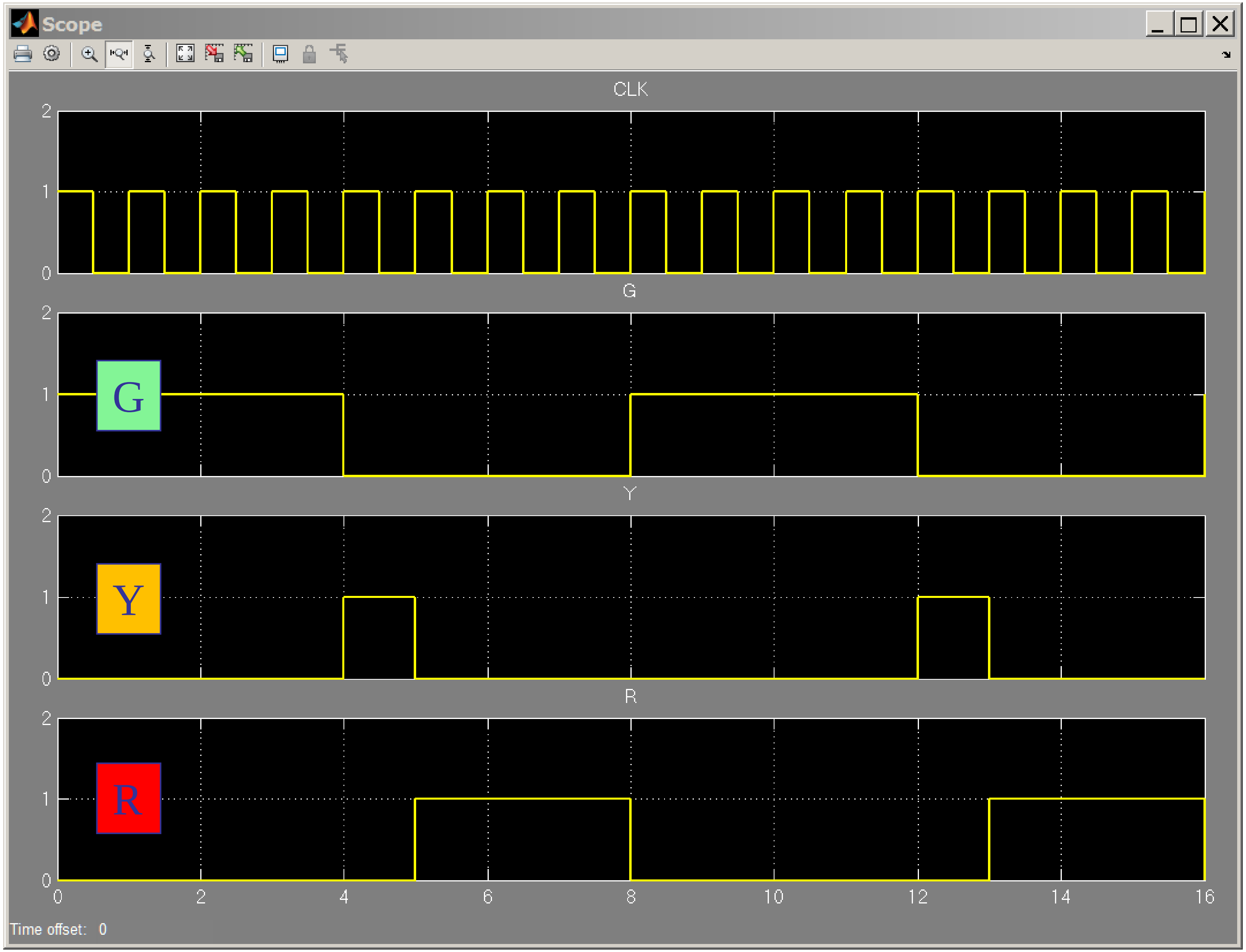
$$Q_0^{\text{next}} = Q_0 \oplus 1 = Q_0'$$

next-state sub-function

for a 3-bit counter

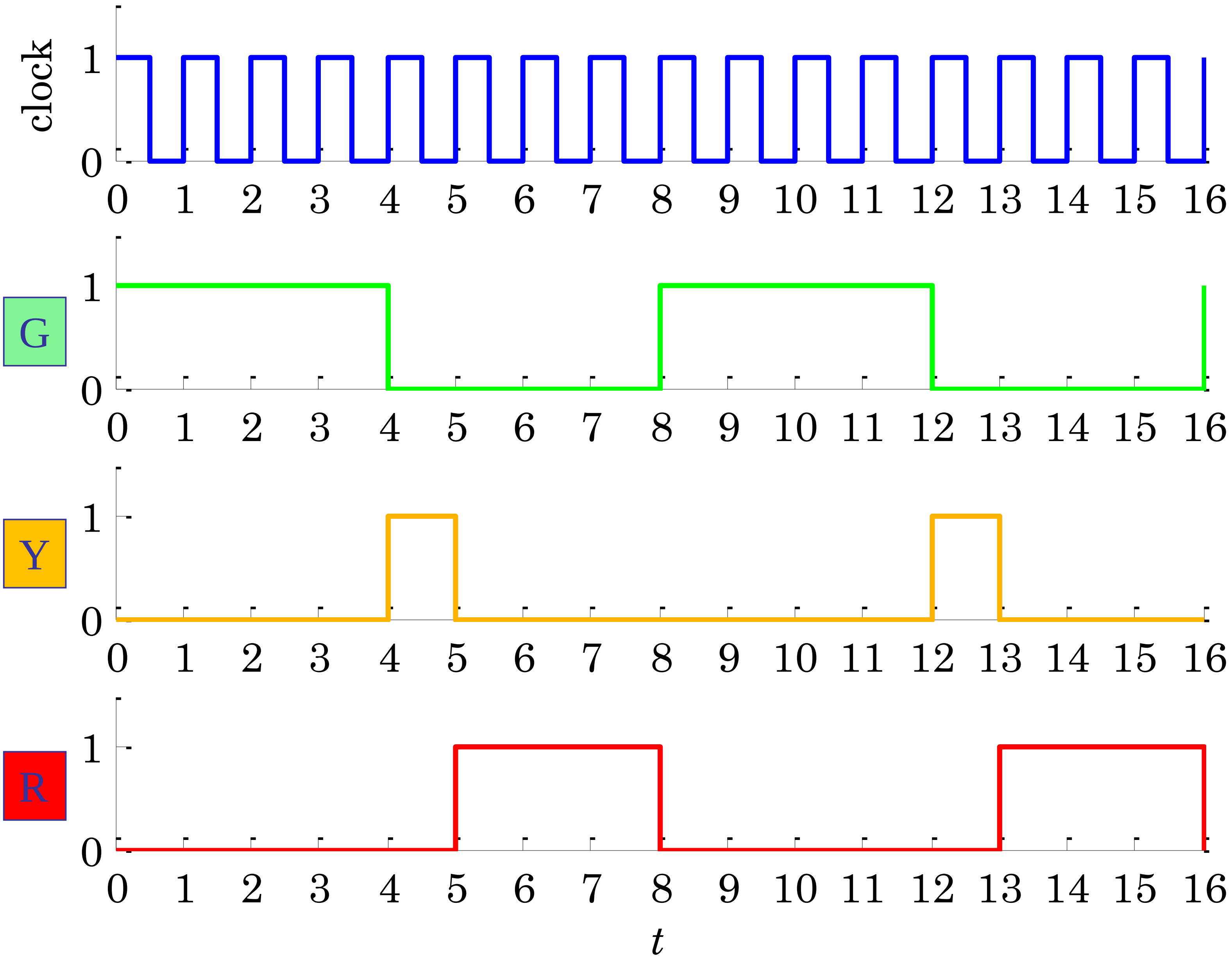


Example 1



scope output

Example 3


clock
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
G
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Y
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
R
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

Example 3

Example 4 – traffic light controller. In a variation of the previous example, design a traffic controller that is sequenced in the order **red-green-yellow,** as opposed to previous order of green-yellow-red.

The corresponding truth table and design equations are now:

				next states			outputs		
t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	R	G	Y
0	0	0	0	0	0	1	1	0	0
1	0	0	1	0	1	0	1	0	0
2	0	1	0	0	1	1	1	0	0
3	0	1	1	1	0	0	0	1	0
4	1	0	0	1	0	1	0	1	0
5	1	0	1	1	1	0	0	1	0
6	1	1	0	1	1	1	0	1	0
7	1	1	1	0	0	0	0	0	1

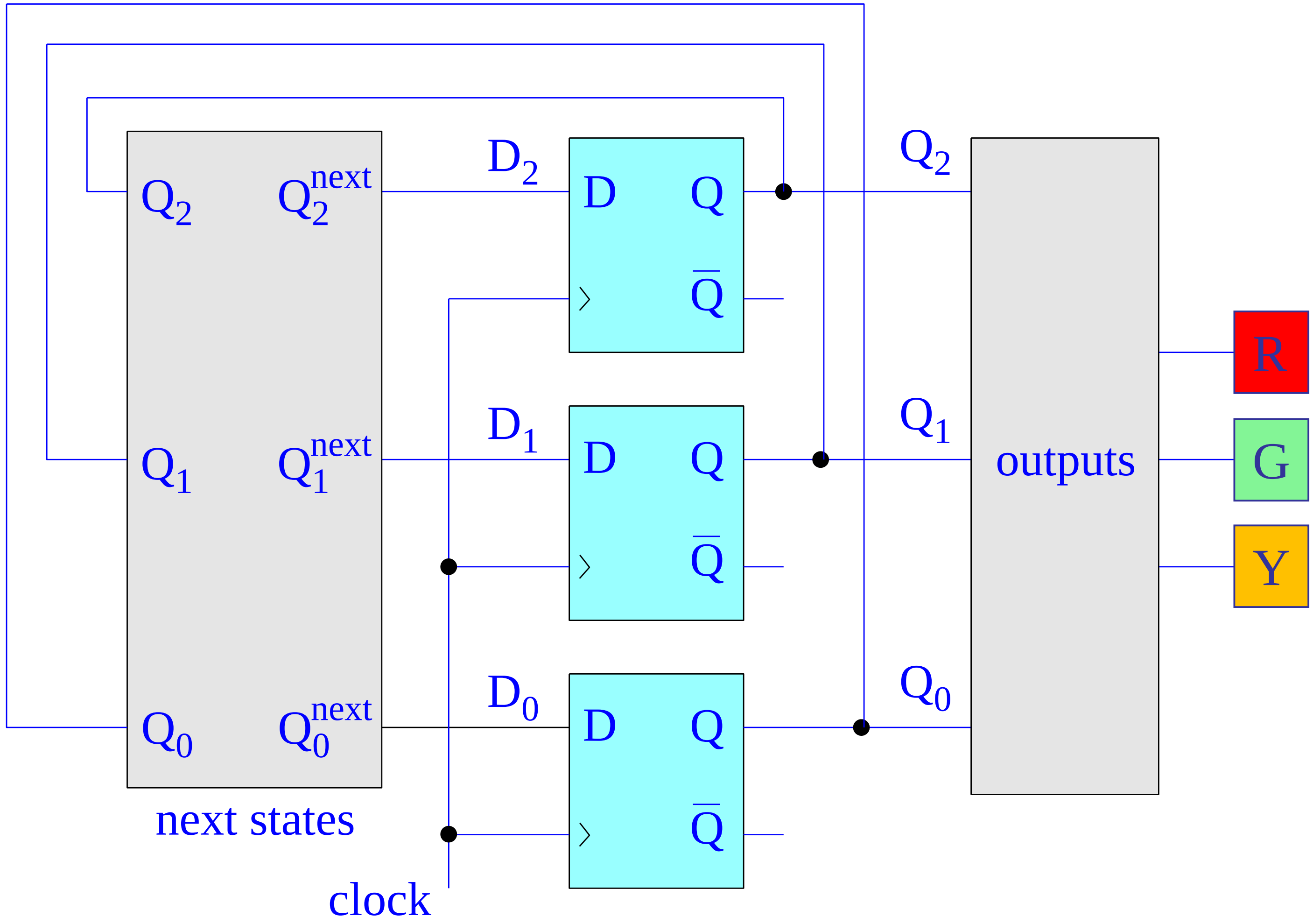
→

$$R = Q_2' Q_1' + Q_2' Q_0'$$

$$G = Q_2 \oplus (Q_1 Q_0)$$

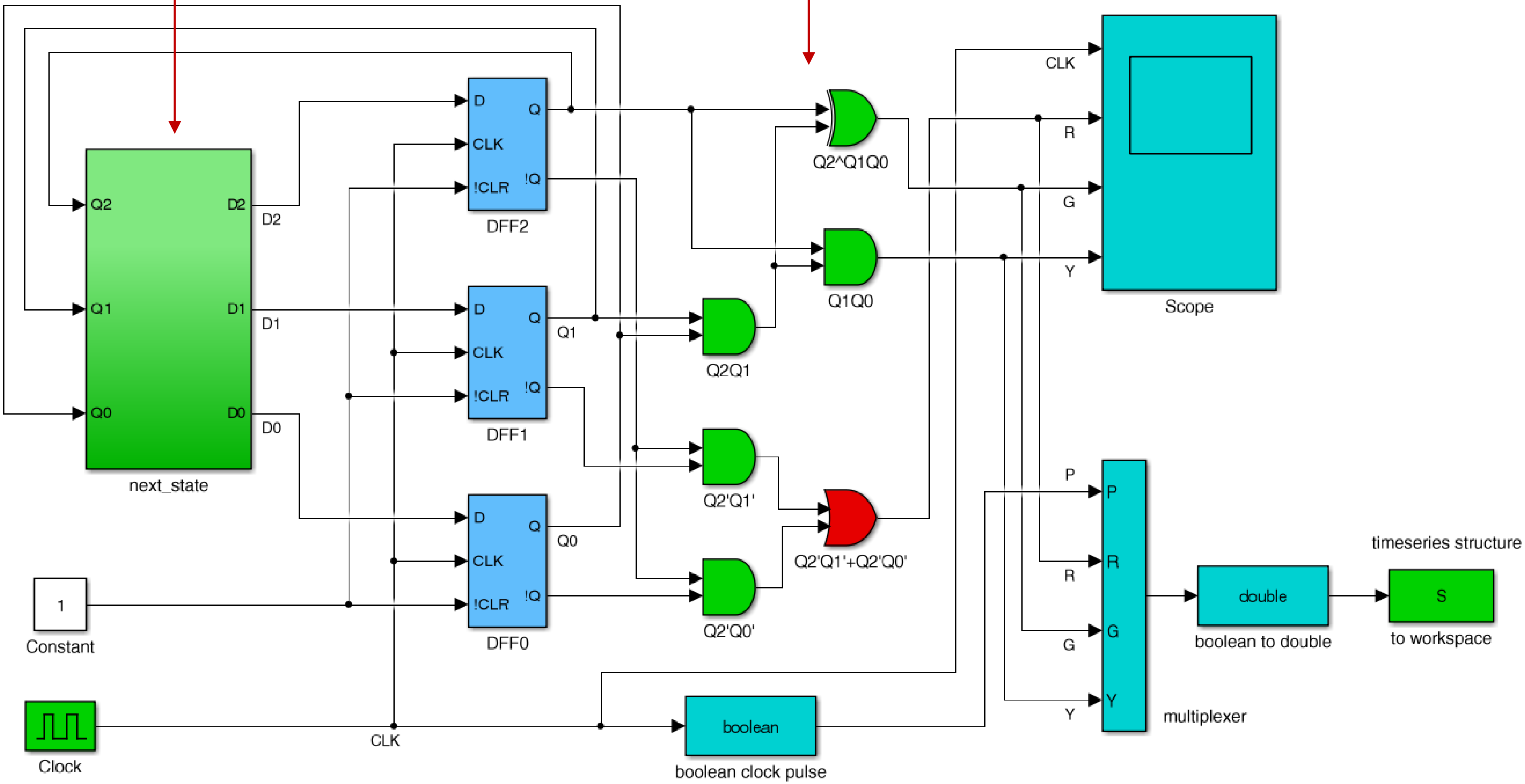
$$Y = Q_2 Q_1 Q_0$$

the counter generates the sequence of states, $Q_2Q_1Q_0$, which, in turn, generate the red, green, yellow output signals

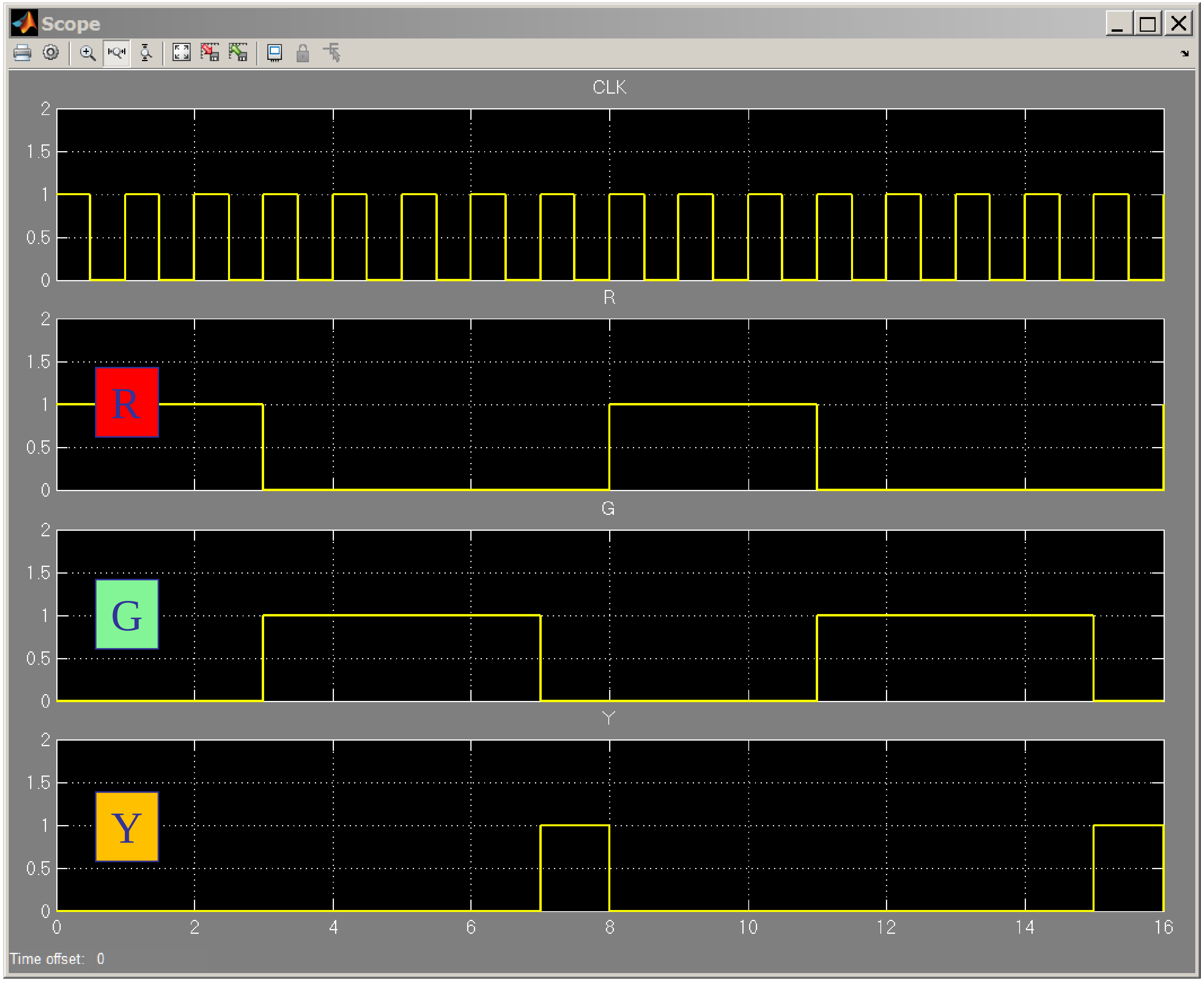


3-bit counter

R, G, Y outputs

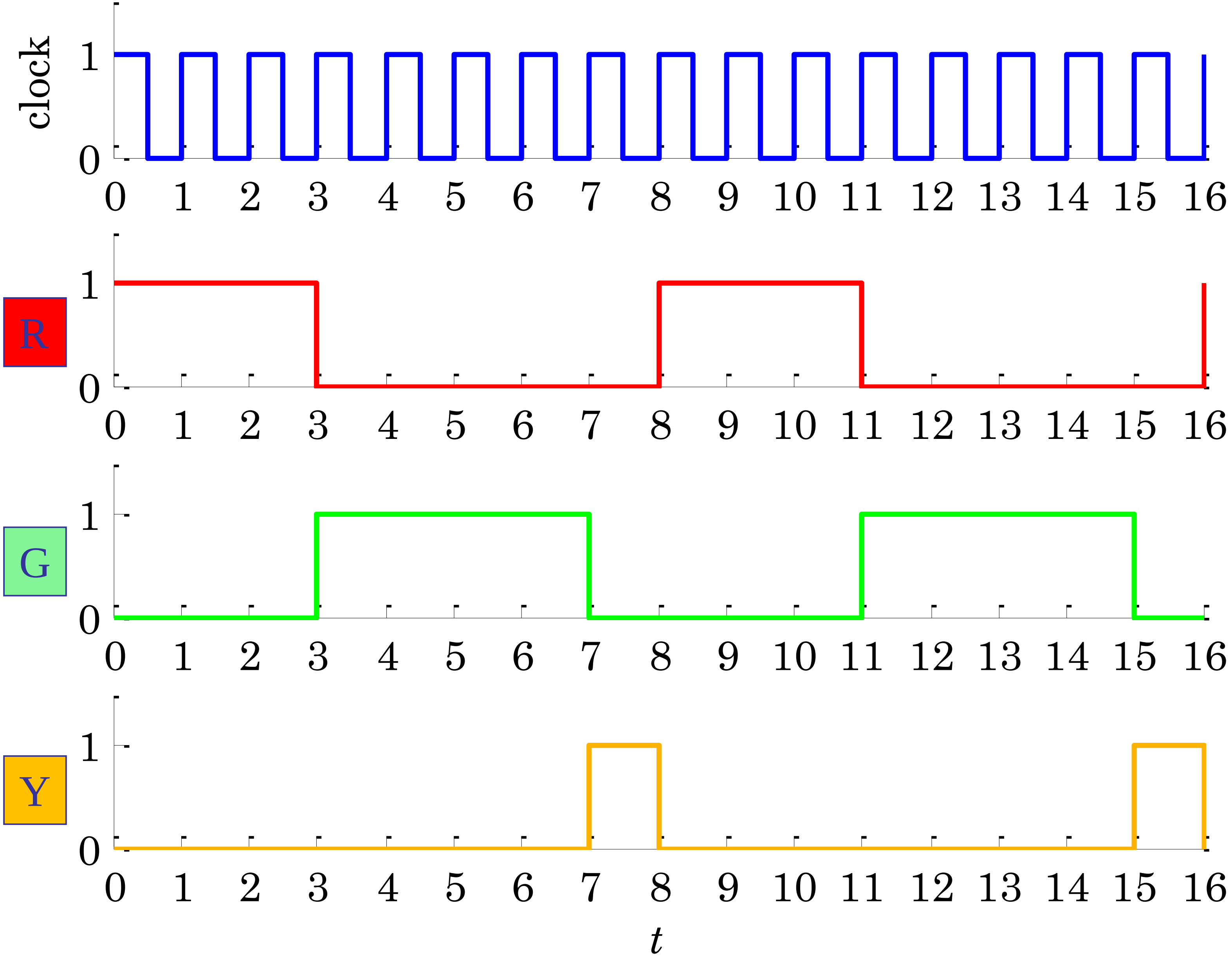


Example 4



scope output

Example 4



Example 4

Example 5 – generating arbitrary sequences. It is desired to design a counter that generates the following repeating length-8 sequence of numbers:

$$[0, 6, 2, 4, 1, 7, 3, 5]$$

Using three bits, $Q_2Q_1Q_0$, to represent these numbers, the corresponding characteristic table, counting in the above order, will be as follows.

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	1	0
6	1	1	0	0	1	0
2	0	1	0	1	0	0
4	1	0	0	0	0	1
1	0	0	1	1	1	1
7	1	1	1	0	1	1
3	0	1	1	1	0	1
5	1	0	1	0	0	0

→

$$Q_2^{\text{next}} = Q_2'$$

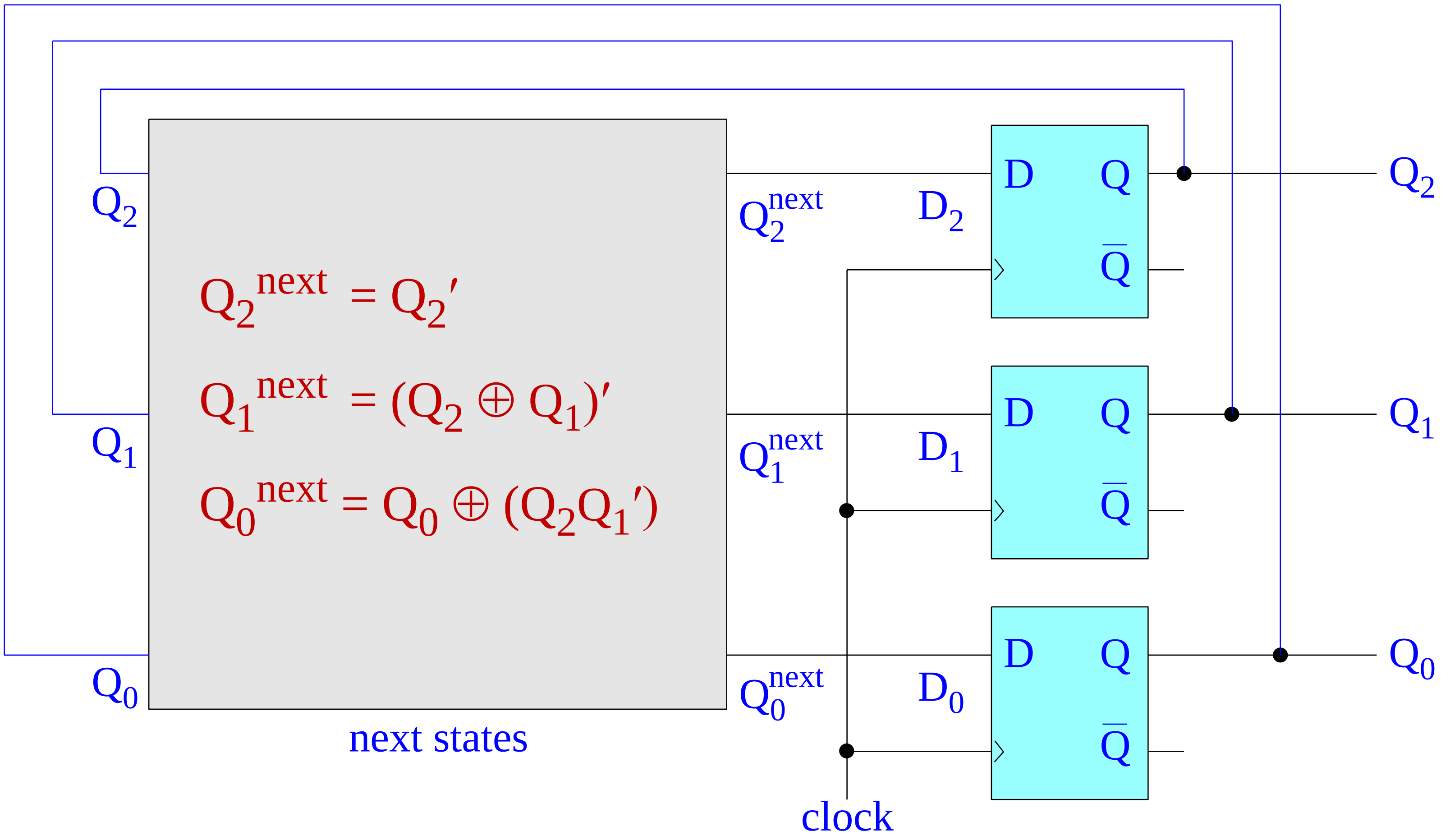
$$Q_1^{\text{next}} = (Q_2 \oplus Q_1)'$$

$$Q_0^{\text{next}} = Q_0 \oplus (Q_2Q_1')$$

← repeat

recall: $(a \oplus b)' = a\,b + a'\,b'$

realization using D flip-flops

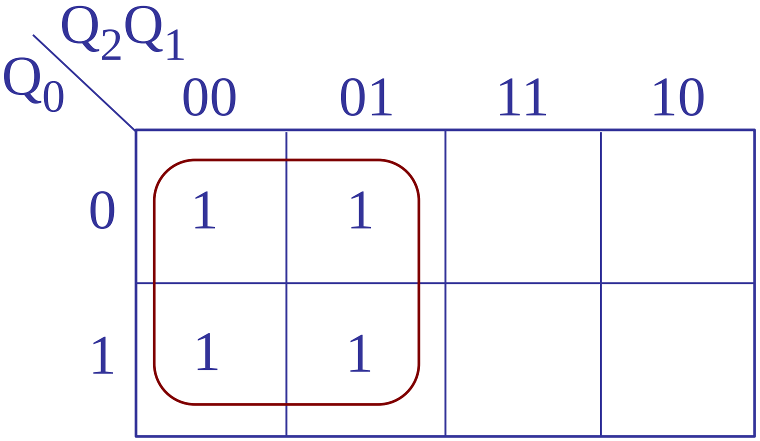


$$Q_2^{next} = Q_2'$$

$$Q_1^{next} = (Q_2 \oplus Q_1)'$$

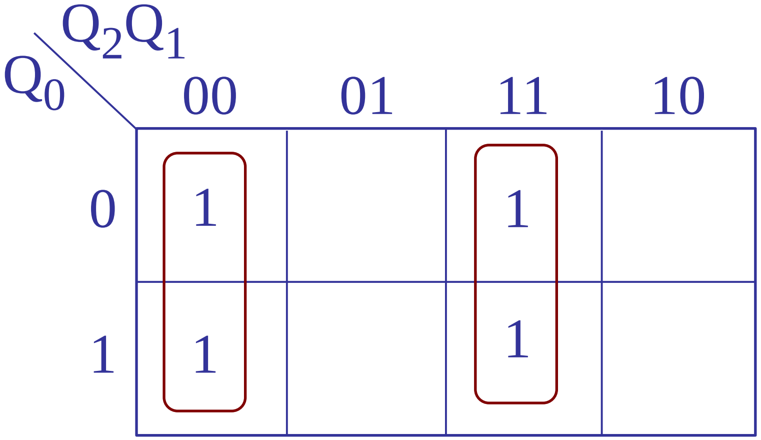
$$Q_0^{next} = Q_0 \oplus (Q_2 Q_1')$$

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	1	0
6	1	1	0	0	1	0
2	0	1	0	1	0	0
4	1	0	0	0	0	1
1	0	0	1	1	1	1
7	1	1	1	0	1	1
3	0	1	1	1	0	1
5	1	0	1	0	0	0



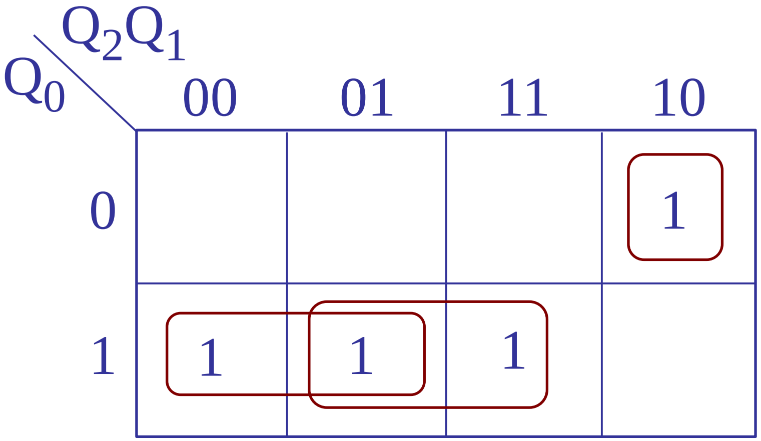
$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	1	1		
1	1	1		

$Q_2^{\text{next}} = Q_2'$



$Q_0 \backslash Q_2Q_1$	00	01	11	10
0	1		1	
1	1		1	

$Q_1^{\text{next}} = Q_2'\,Q_1' + Q_2\,Q_1 = (Q_2 \oplus Q_1)'$

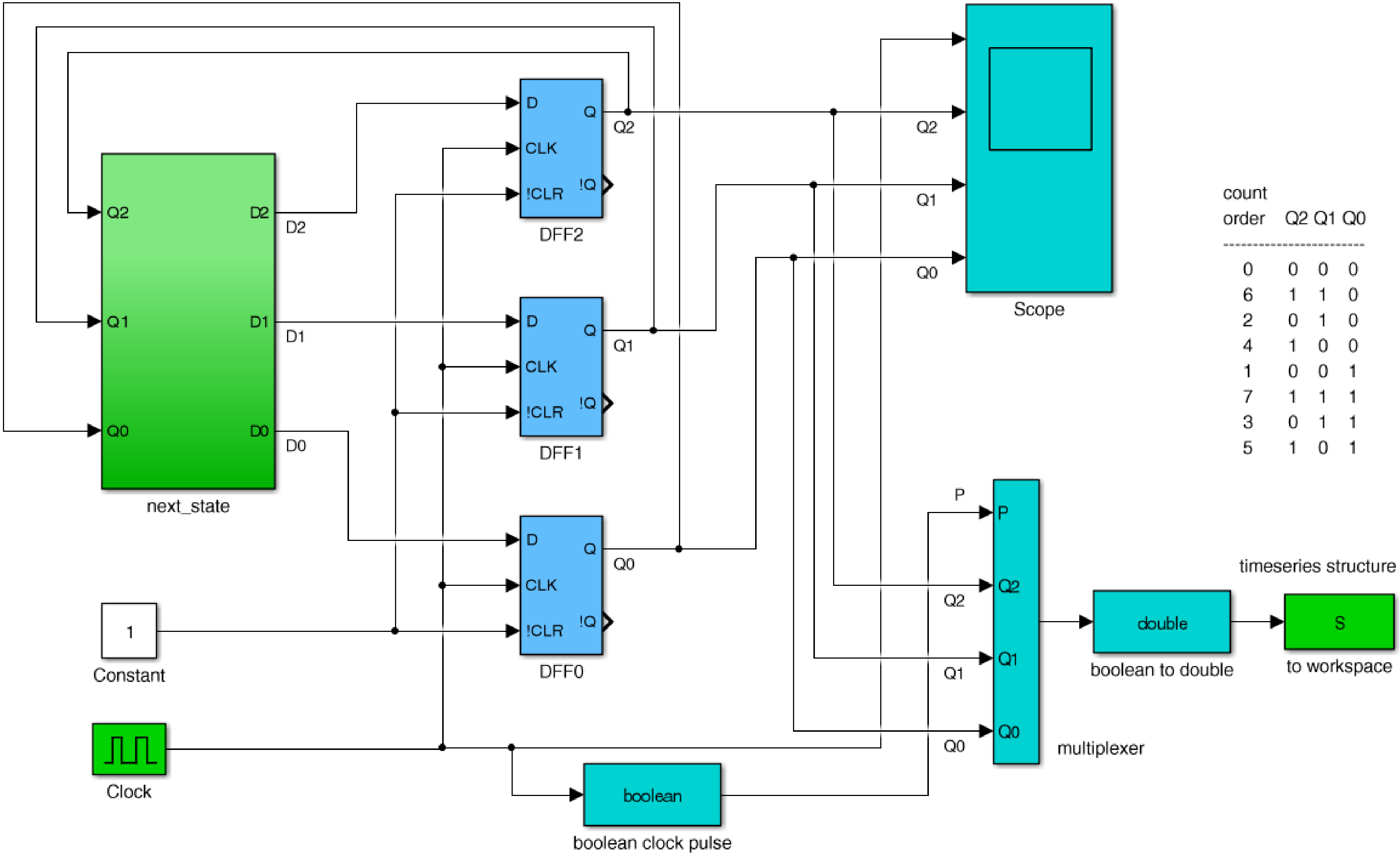


$Q_0 \backslash Q_2Q_1$	00	01	11	10
0				1
1	1	1	1	

$$Q_0^{\text{next}} = Q_2'\,Q_0 + Q_1\,Q_0 + Q_2\,Q_1'\,Q_0'$$
$$= Q_0 \oplus (Q_2Q_1')$$

Example 5

Simulink implementation



count order	Q2	Q1	Q0
0	0	0	0
6	1	1	0
2	0	1	0
4	1	0	0
1	0	0	1
7	1	1	1
3	0	1	1
5	1	0	1

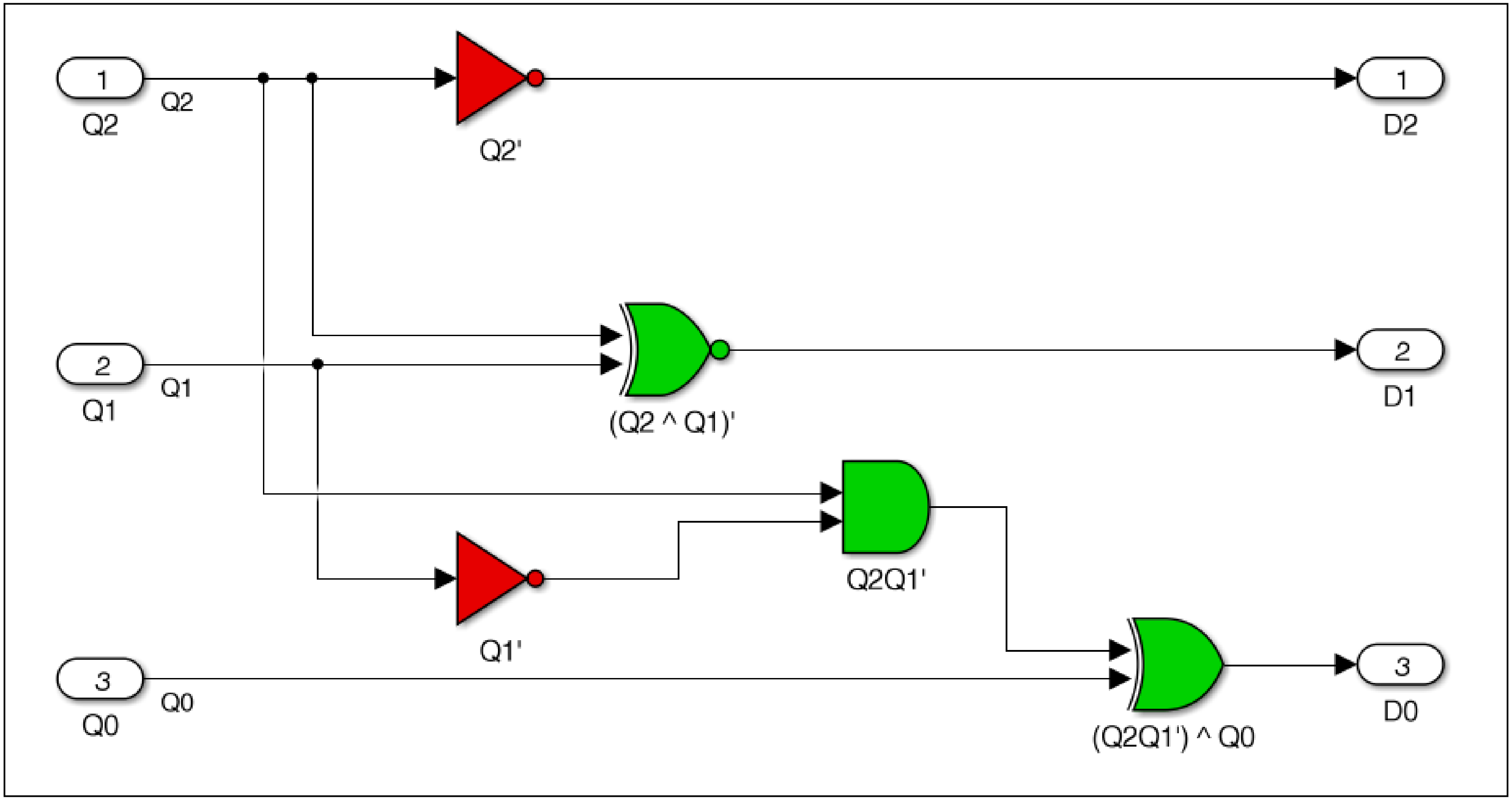
Example 5

$$Q_2^{\text{next}} = Q_2'$$

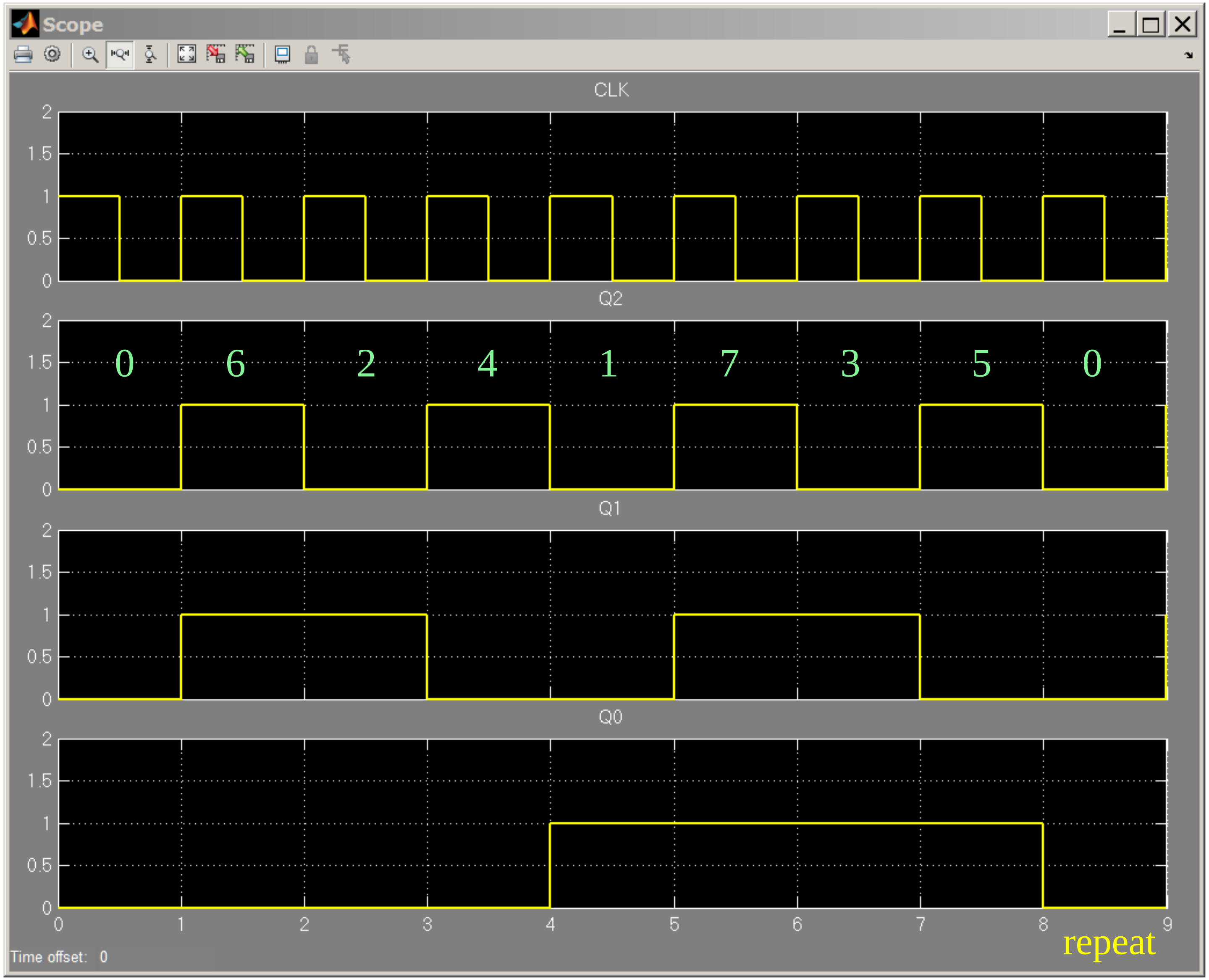
$$Q_1^{\text{next}} = (Q_2 \oplus Q_1)'$$

$$Q_0^{\text{next}} = Q_0 \oplus (Q_2 Q_1')$$

next-state subfunction



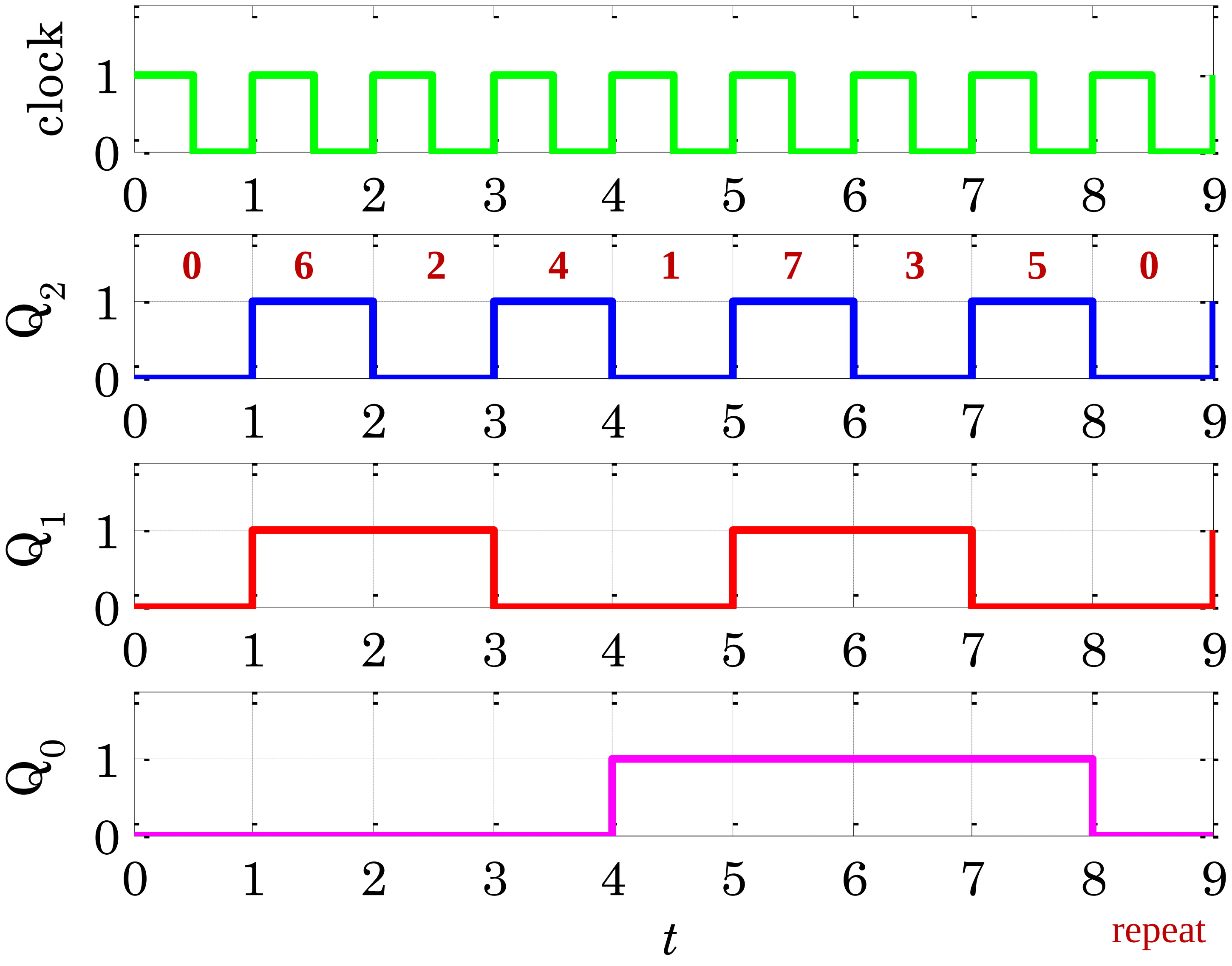
scope output



```
% plot timing diagram from exported Simulink data

t =  S.time;           % time
P =  S.data(:,1);      % clock pulse
Q2 = S.data(:,2);
Q1 = S.data(:,3);
Q0 = S.data(:,4);

figure;
subplot(4,1,1); stairs(t,P,'g-’);
subplot(4,1,2); stairs(t,Q2,'b-’);
subplot(4,1,3); stairs(t,Q1,'r-’);
subplot(4,1,4); stairs(t,Q0,'m-’);
xlabel('{\itt}')
```

Example 5

timing diagram



Example 5

Example 6 – generating a sub-sequence. Design a counter that generates the following repeating length-6 sub-sequence of the previous example,

[0, 6, 2, 4, 1, 7]

Using three bits, $Q_2Q_1Q_0$, to represent these numbers, the corresponding characteristic table, counting in the above order, will be as follows.

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	1	0
6	1	1	0	0	1	0
2	0	1	0	1	0	0
4	1	0	0	0	0	1
1	0	0	1	1	1	1
7	1	1	1	0	0	0
3	0	1	1	x	x	x
5	1	0	1	x	x	x

$$Q_2^{\text{next}} = Q_2'$$

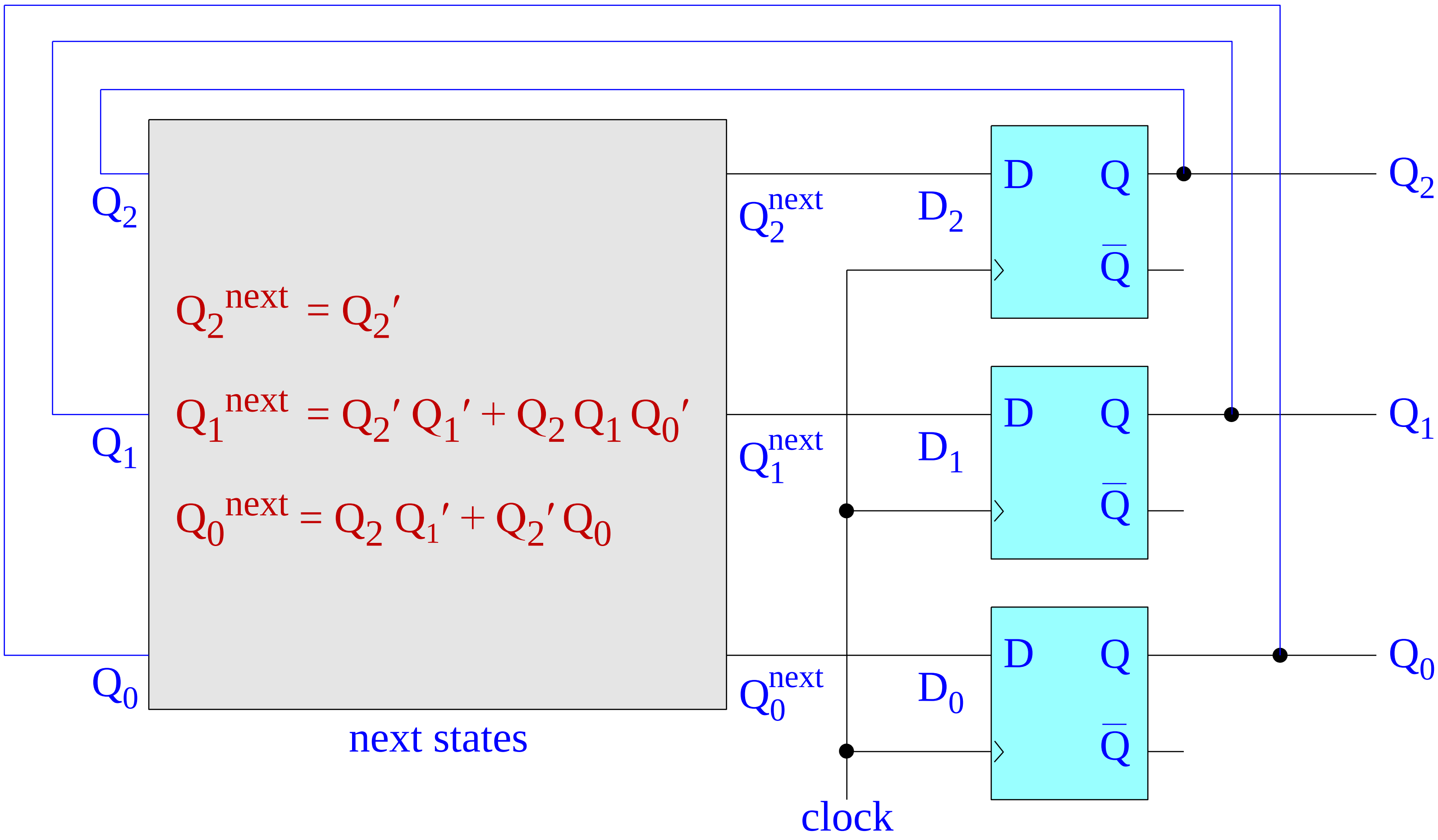
$$Q_1^{\text{next}} = Q_2'\,Q_1' + Q_2\,Q_1\,Q_0'$$

$$Q_0^{\text{next}} = Q_2\,Q_1' + Q_2'\,Q_0$$

repeat

don’t cares

realization using D flip-flops



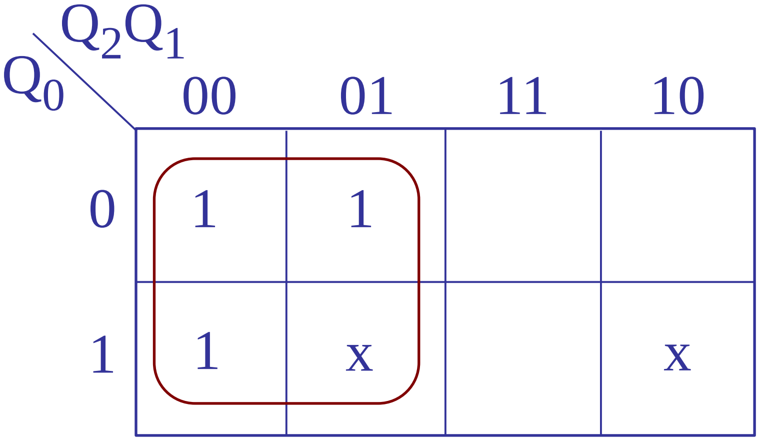
$$Q_2^{next} = Q_2'$$

$$Q_1^{next} = Q_2' Q_1' + Q_2 Q_1 Q_0'$$

$$Q_0^{next} = Q_2 Q_1' + Q_2' Q_0$$

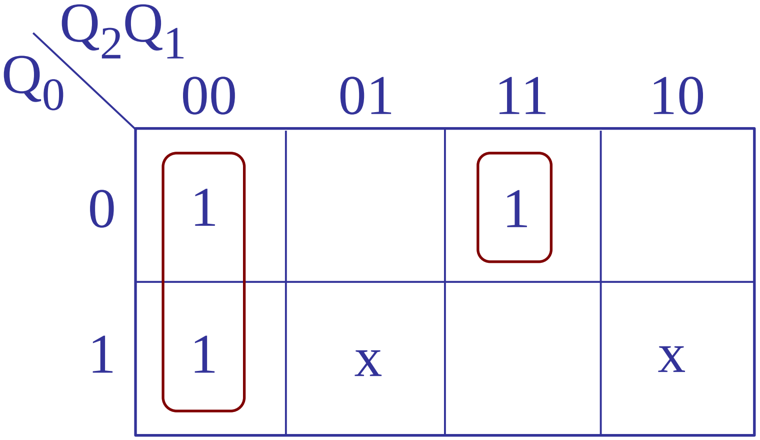
Example 6

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	1	0
6	1	1	0	0	1	0
2	0	1	0	1	0	0
4	1	0	0	0	0	1
1	0	0	1	1	1	1
7	1	1	1	0	0	0
3	0	1	1	x	x	x
5	1	0	1	x	x	x



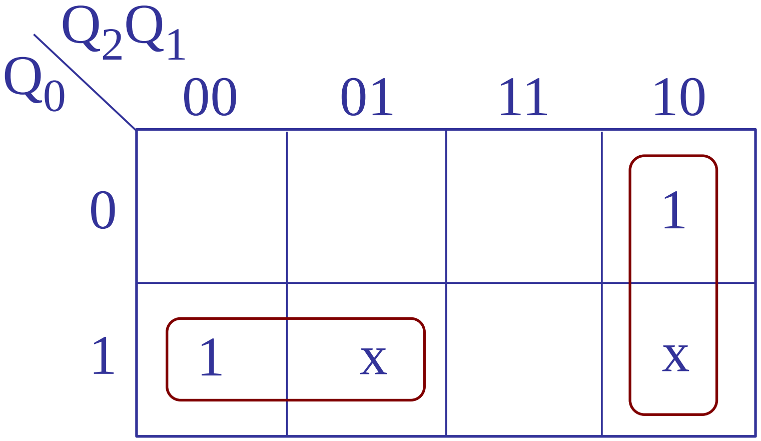
Q_0 \ Q_2Q_1	00	01	11	10
0	1	1		
1	1	x		x

$Q_2^{next} = Q_2'$



Q_0 \ Q_2Q_1	00	01	11	10
0	1		1	
1	1	x		x

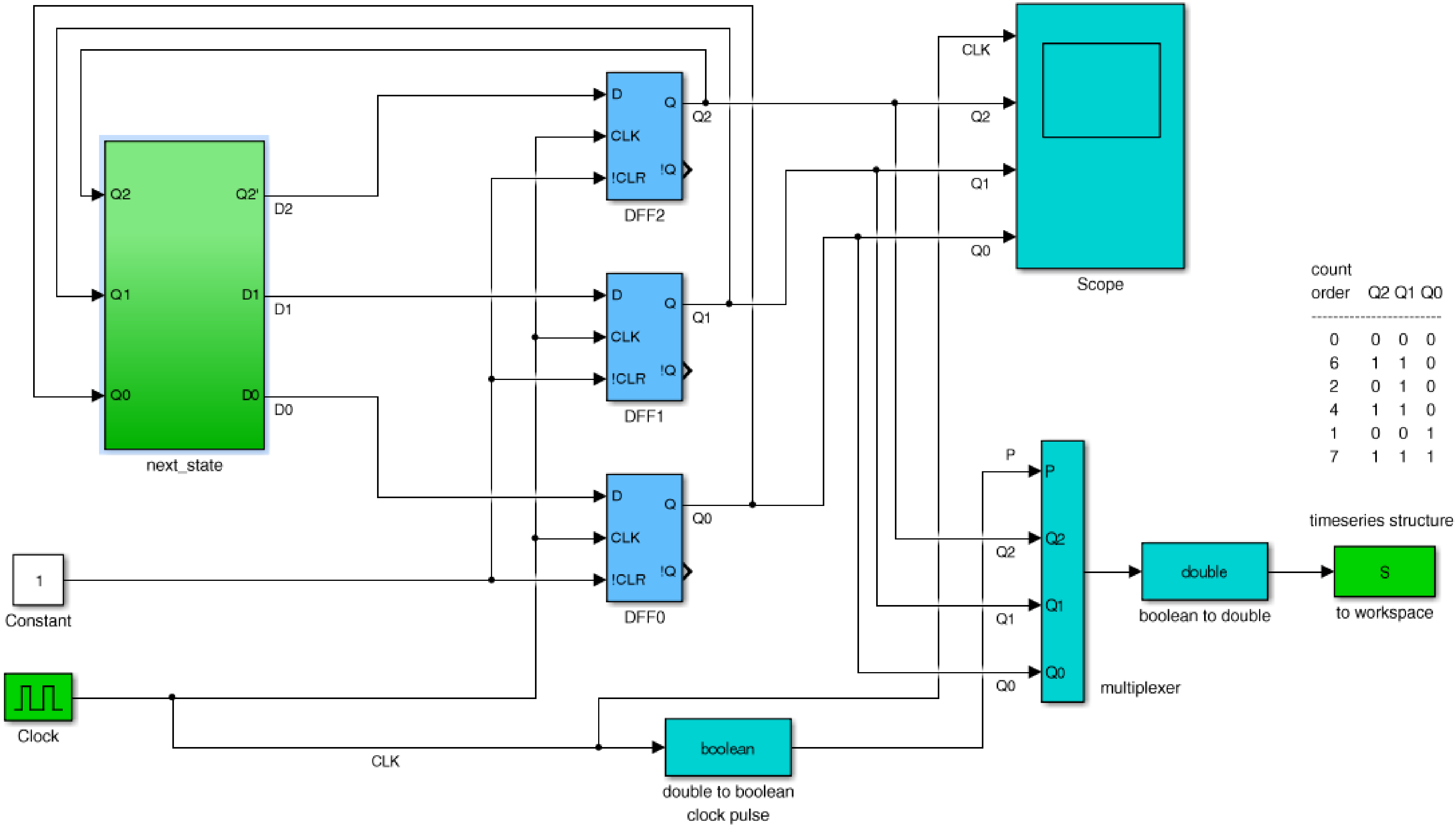
$Q_1^{next} = Q_2' \, Q_1' + Q_2 \, Q_1 \, Q_0'$



Q_0 \ Q_2Q_1	00	01	11	10
0				1
1	1	x		x

$Q_0^{next} = Q_2 \, Q_1' + Q_2' \, Q_0$

Simulink implementation



count order	Q2	Q1	Q0
0	0	0	0
6	1	1	0
2	0	1	0
4	1	1	0
1	0	0	1
7	1	1	1

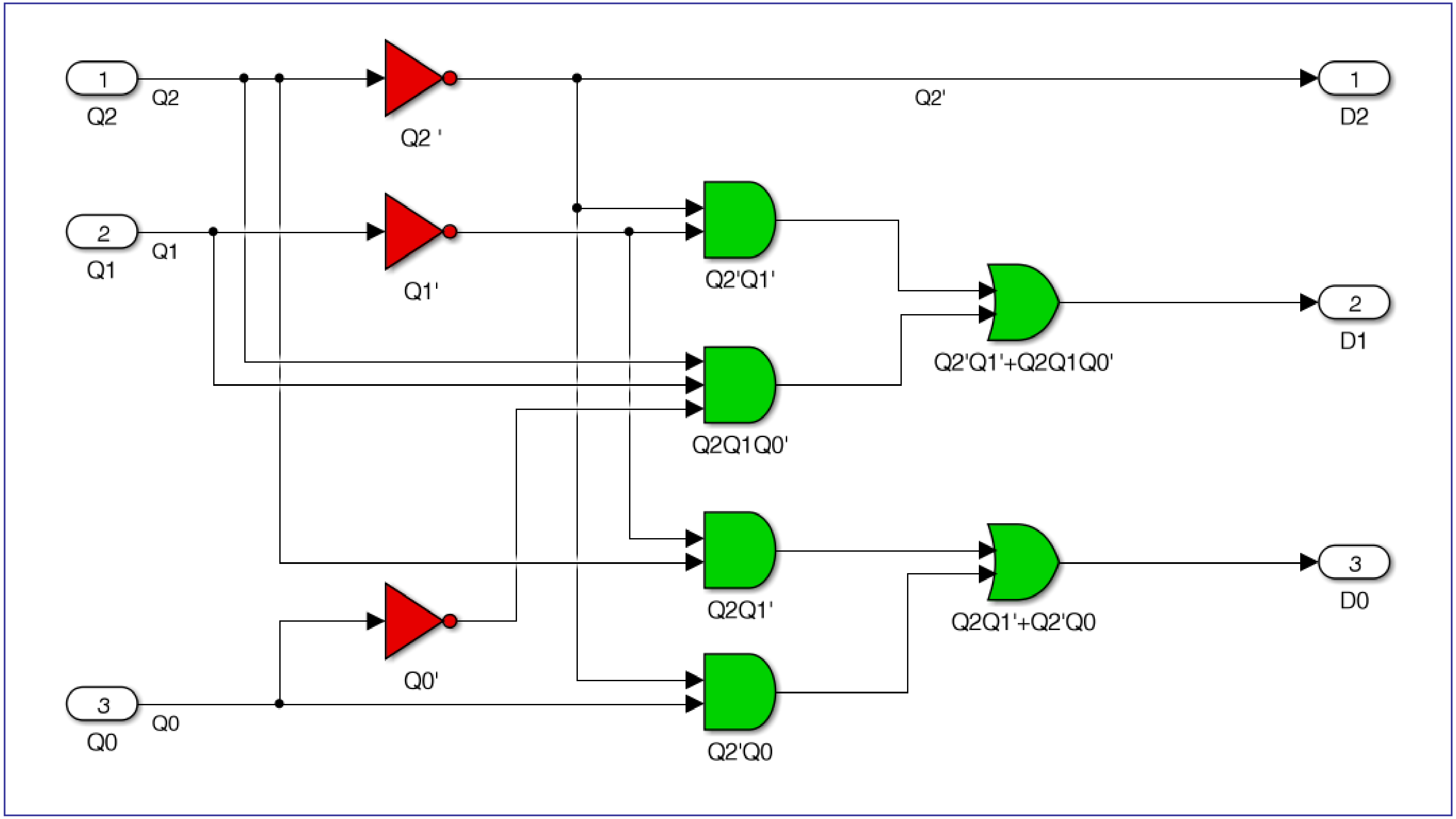
Example 6

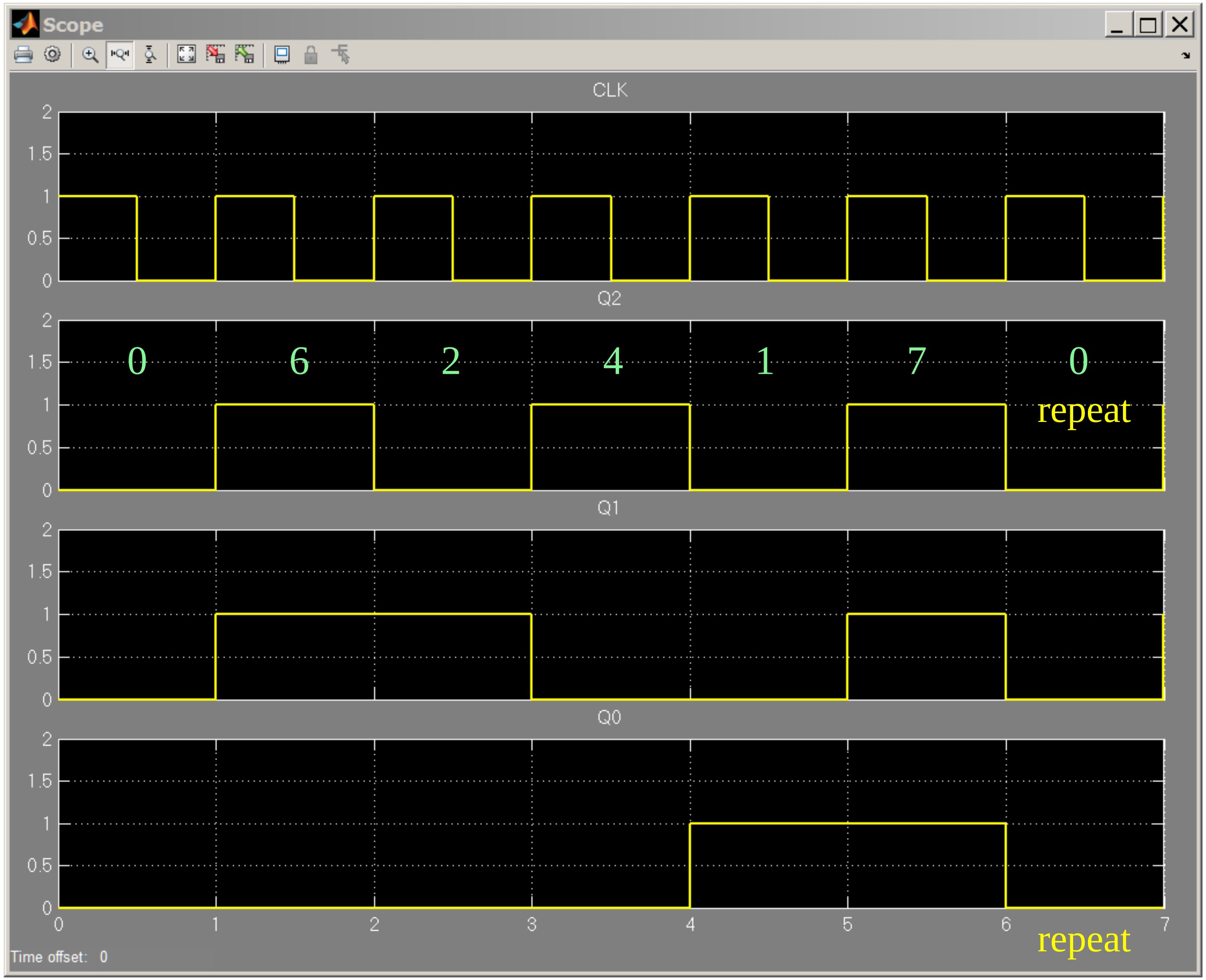
$$Q_2^{\text{next}} = Q_2'$$

$$Q_1^{\text{next}} = Q_2' Q_1' + Q_2 Q_1 Q_0'$$

$$Q_0^{\text{next}} = Q_2 Q_1' + Q_2' Q_0$$

next-state subfunction

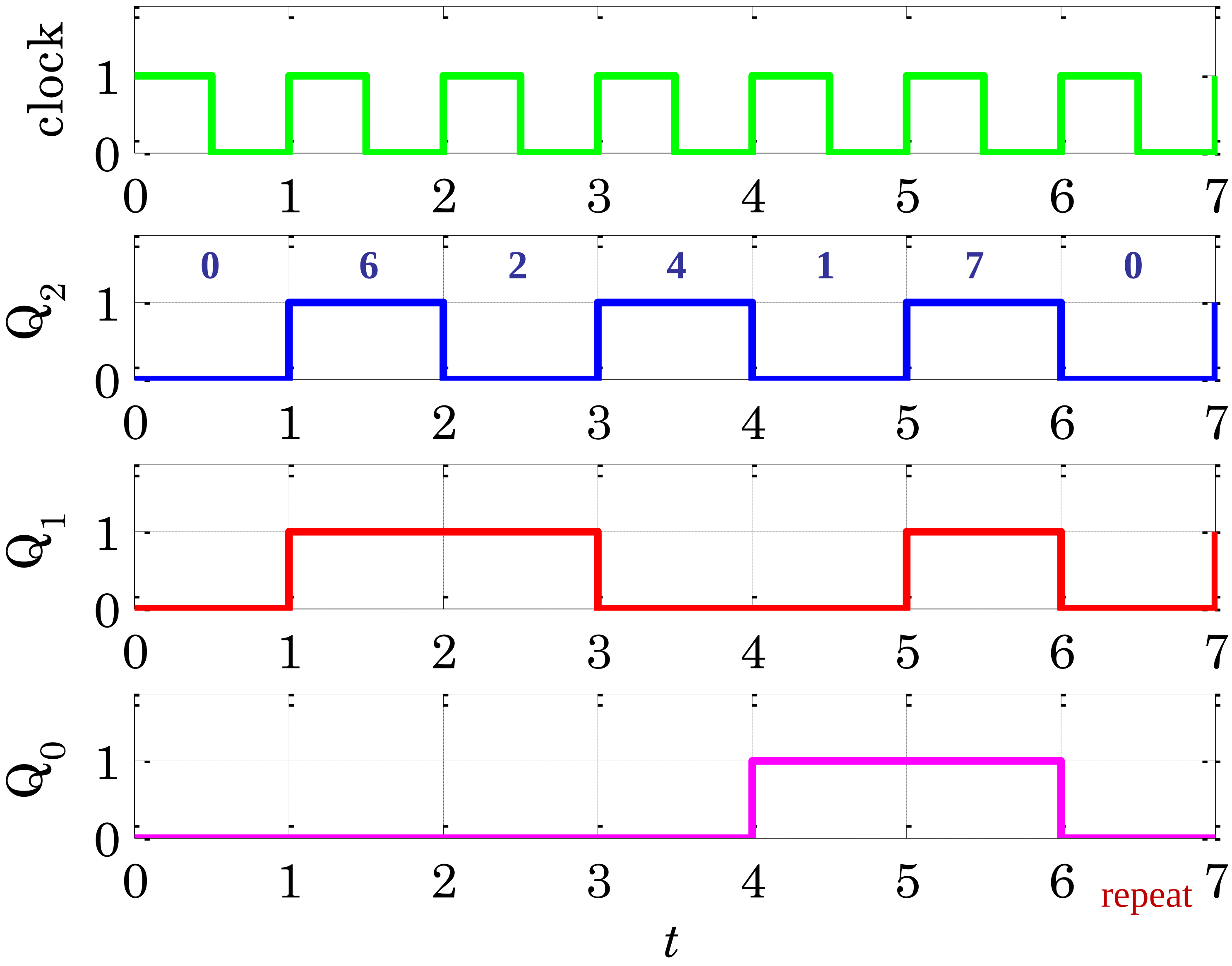




scope output

Example 6

timing diagram



Example 6

Example 7 – generating a sub-sequence. Design a counter that generates the following repeating length-5 sub-sequence of Example-5,

[0, 6, 2, 4, 1]

Using three bits, $Q_2Q_1Q_0$, to represent these numbers, the corresponding characteristic table, counting in the above order, will be as follows.

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	1	0
6	1	1	0	0	1	0
2	0	1	0	1	0	0
4	1	0	0	0	0	1
1	0	0	1	0	0	0
7	1	1	1	x	x	x
3	0	1	1	x	x	x
5	1	0	1	x	x	x

repeat

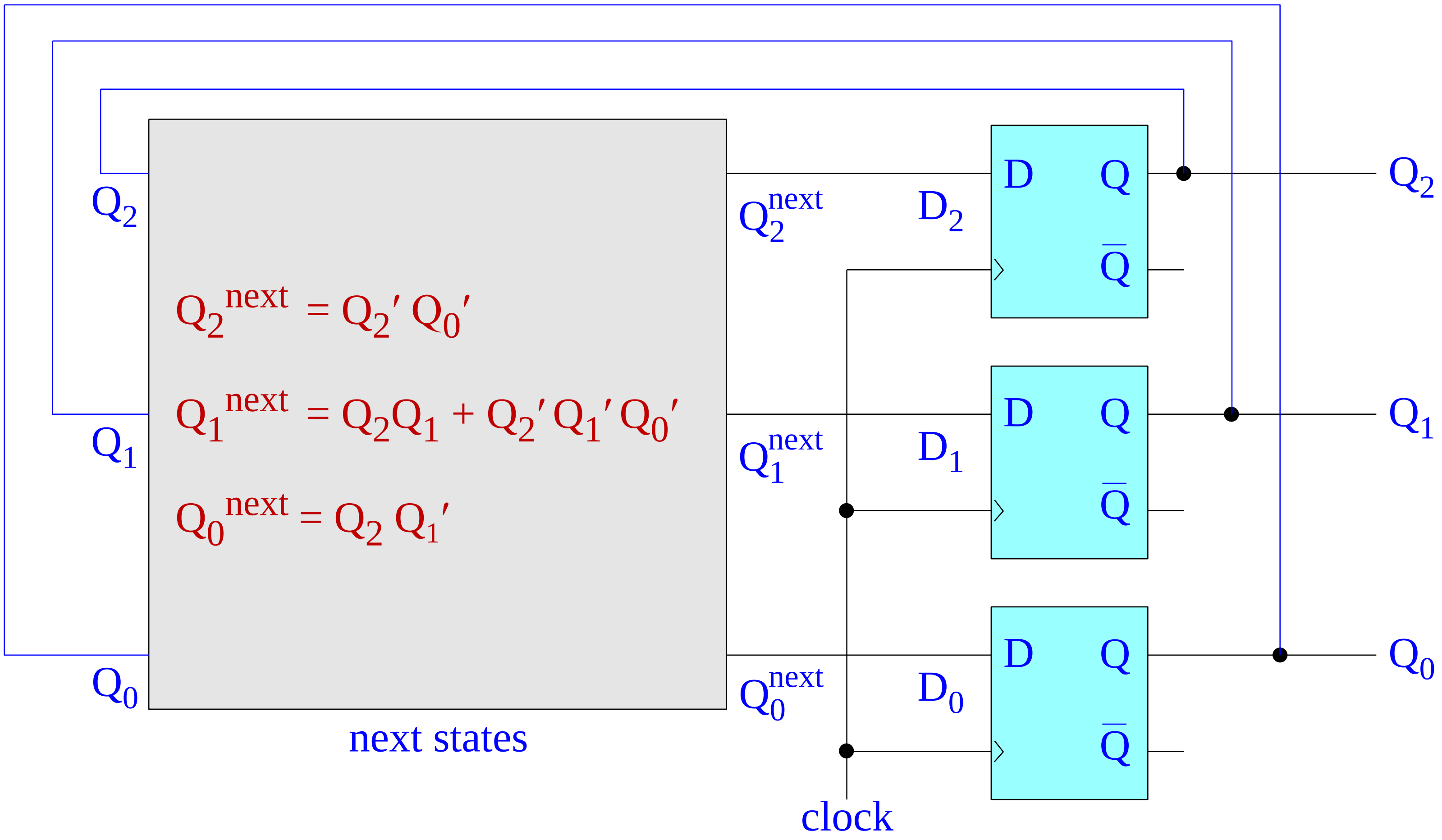
don’t cares

$$Q_2^{next} = Q_2' Q_0'$$

$$Q_1^{next} = Q_2 Q_1 + Q_2' Q_1' Q_0'$$

$$Q_0^{next} = Q_2 Q_1'$$

realization using D flip-flops



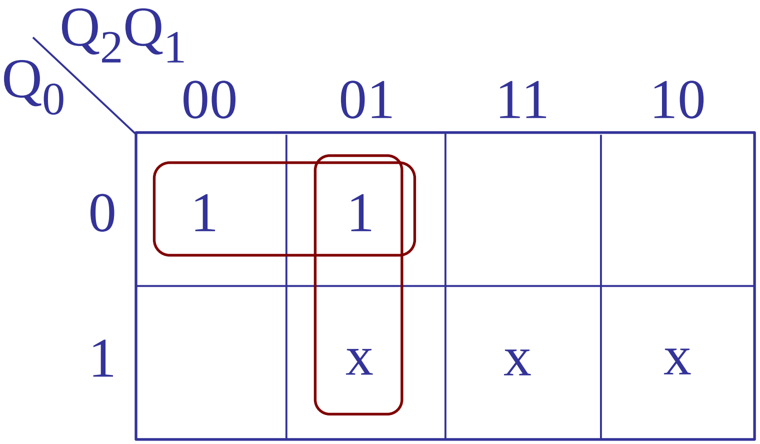
$$Q_2^{next} = Q_2' Q_0'$$

$$Q_1^{next} = Q_2Q_1 + Q_2' Q_1' Q_0'$$

$$Q_0^{next} = Q_2 Q_1'$$

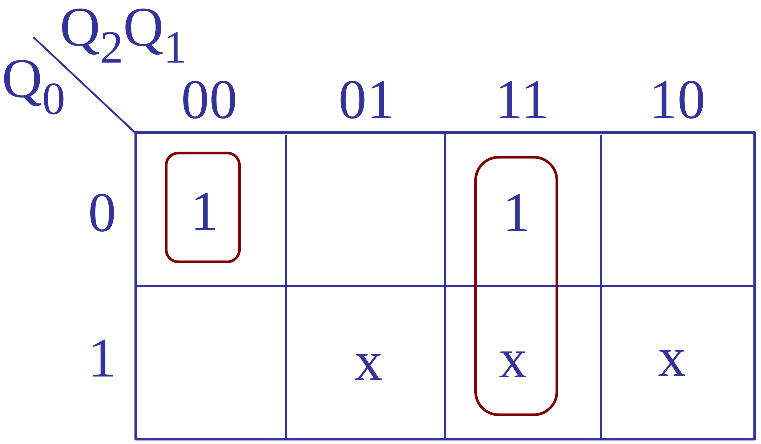
Example 7

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	1	0
6	1	1	0	0	1	0
2	0	1	0	1	0	0
4	1	0	0	0	0	1
1	0	0	1	0	0	0
7	1	1	1	x	x	x
3	0	1	1	x	x	x
5	1	0	1	x	x	x



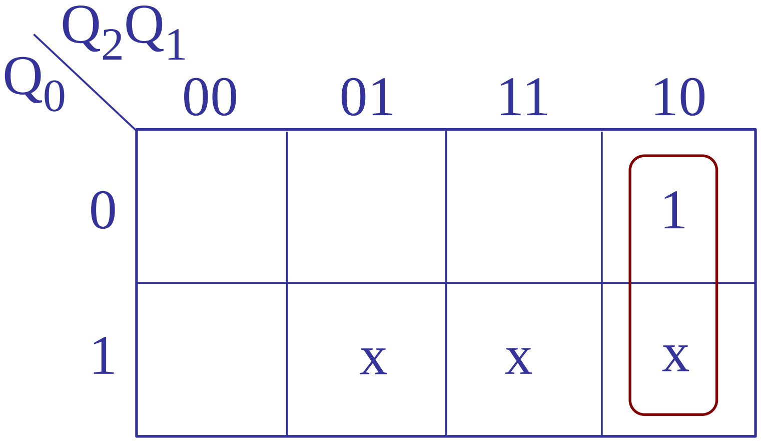
Q_0 \ Q_2Q_1	00	01	11	10
0	1	1		
1		x	x	x

$Q_2^{next} = Q_2' \, Q_0'$



Q_0 \ Q_2Q_1	00	01	11	10
0	1		1	
1		x	x	x

$Q_1^{next} = Q_2 \, Q_1 + Q_2' Q_1' Q_0'$

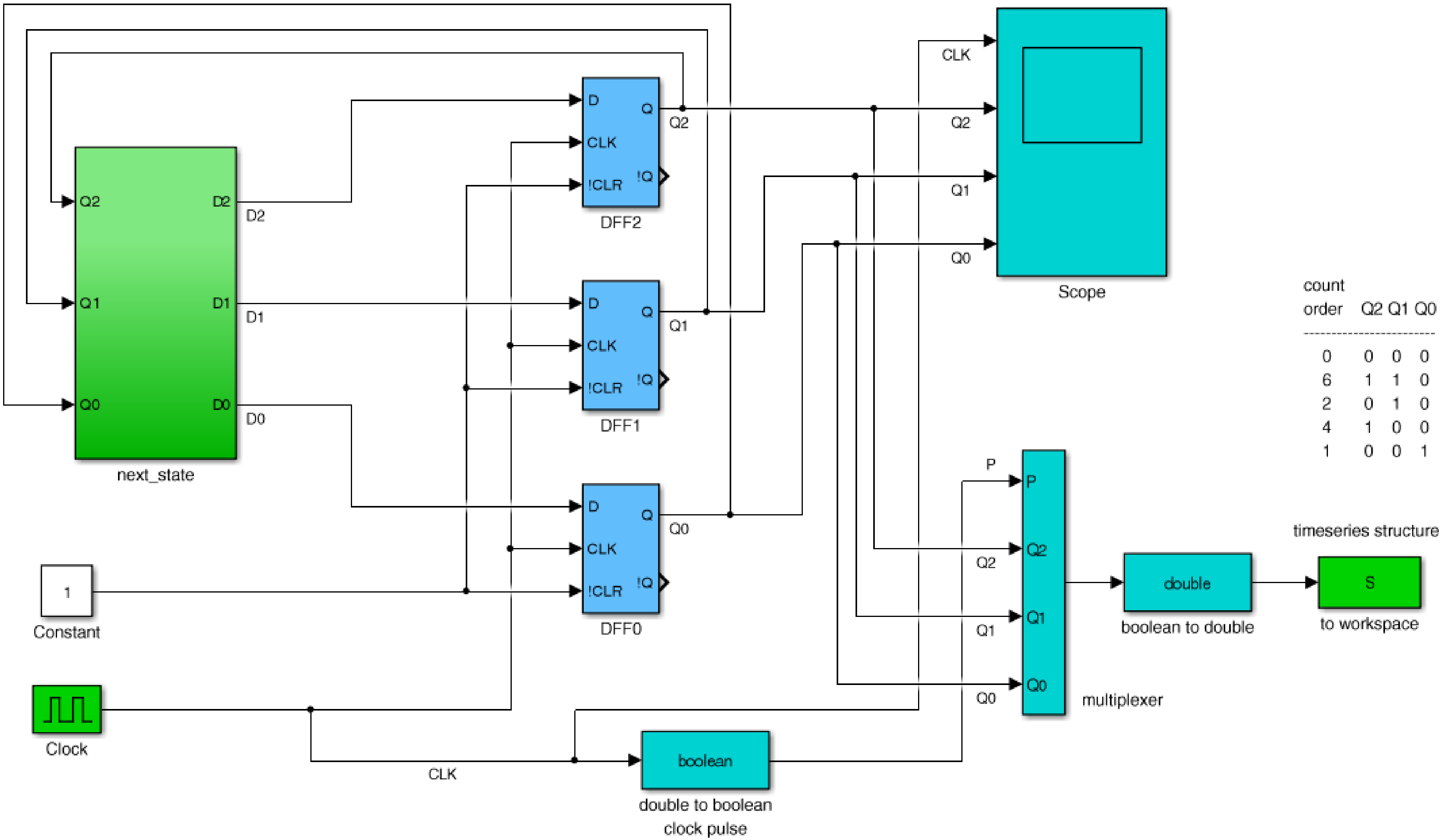


Q_0 \ Q_2Q_1	00	01	11	10
0				1
1		x	x	x

$Q_0^{next} = Q_2 \, Q_1'$

Example 7

Simulink implementation



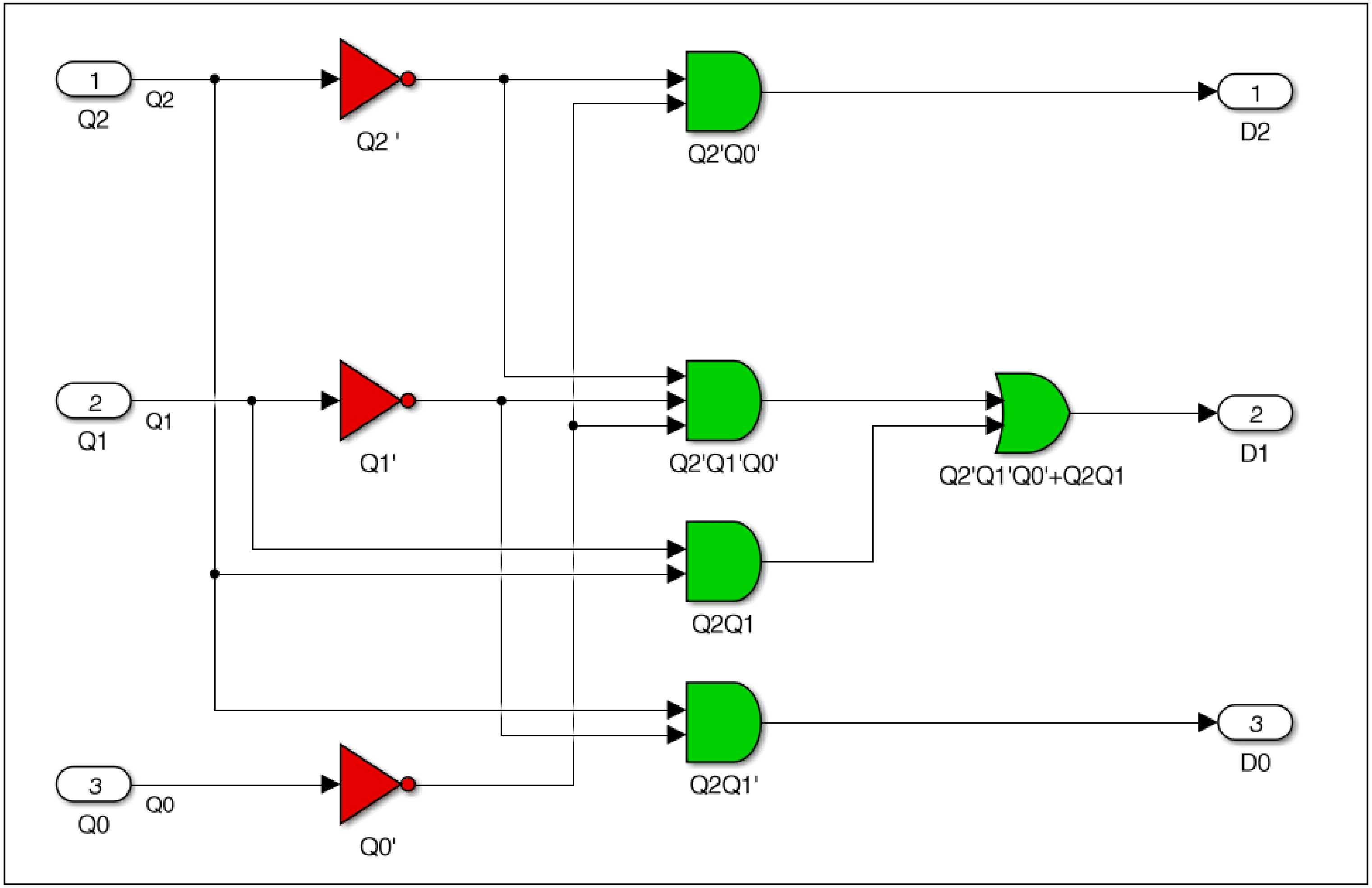
count order	Q2	Q1	Q0
0	0	0	0
6	1	1	0
2	0	1	0
4	1	0	0
1	0	0	1

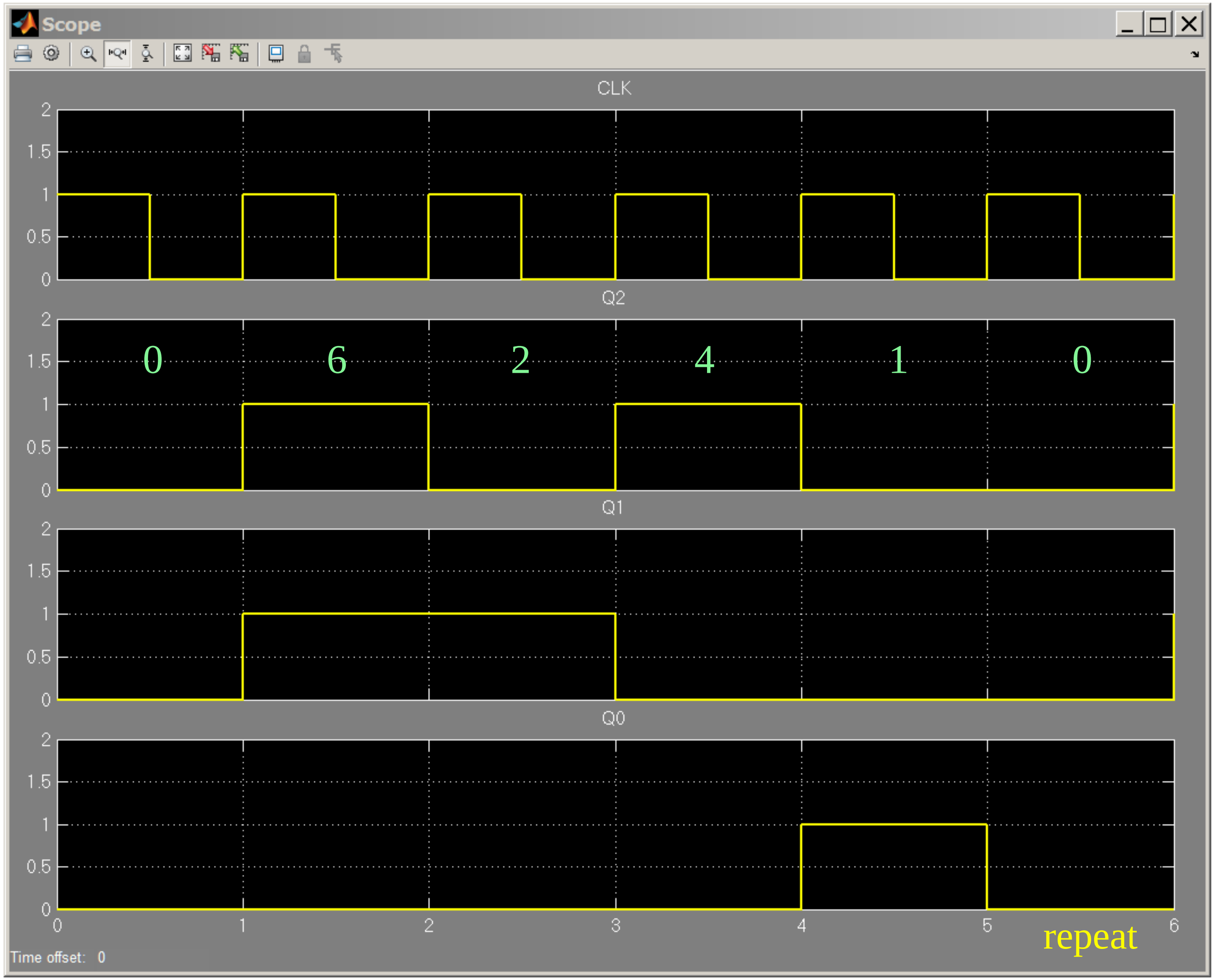
$$Q_2^{next} = Q_2' Q_0'$$

$$Q_1^{next} = Q_2 Q_1 + Q_2' Q_1' Q_0'$$

$$Q_0^{next} = Q_2 Q_1'$$

next-state subfunction

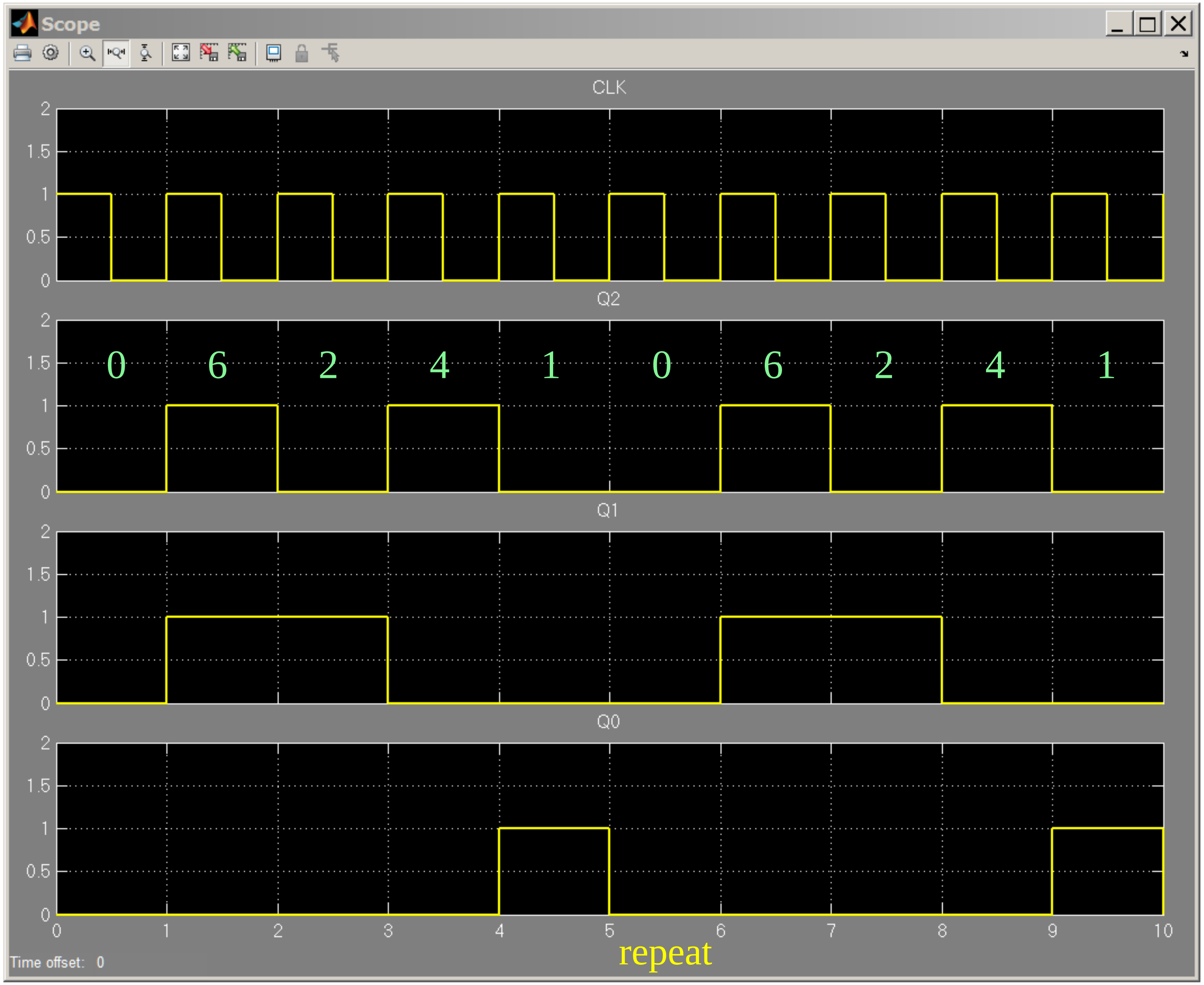




scope output

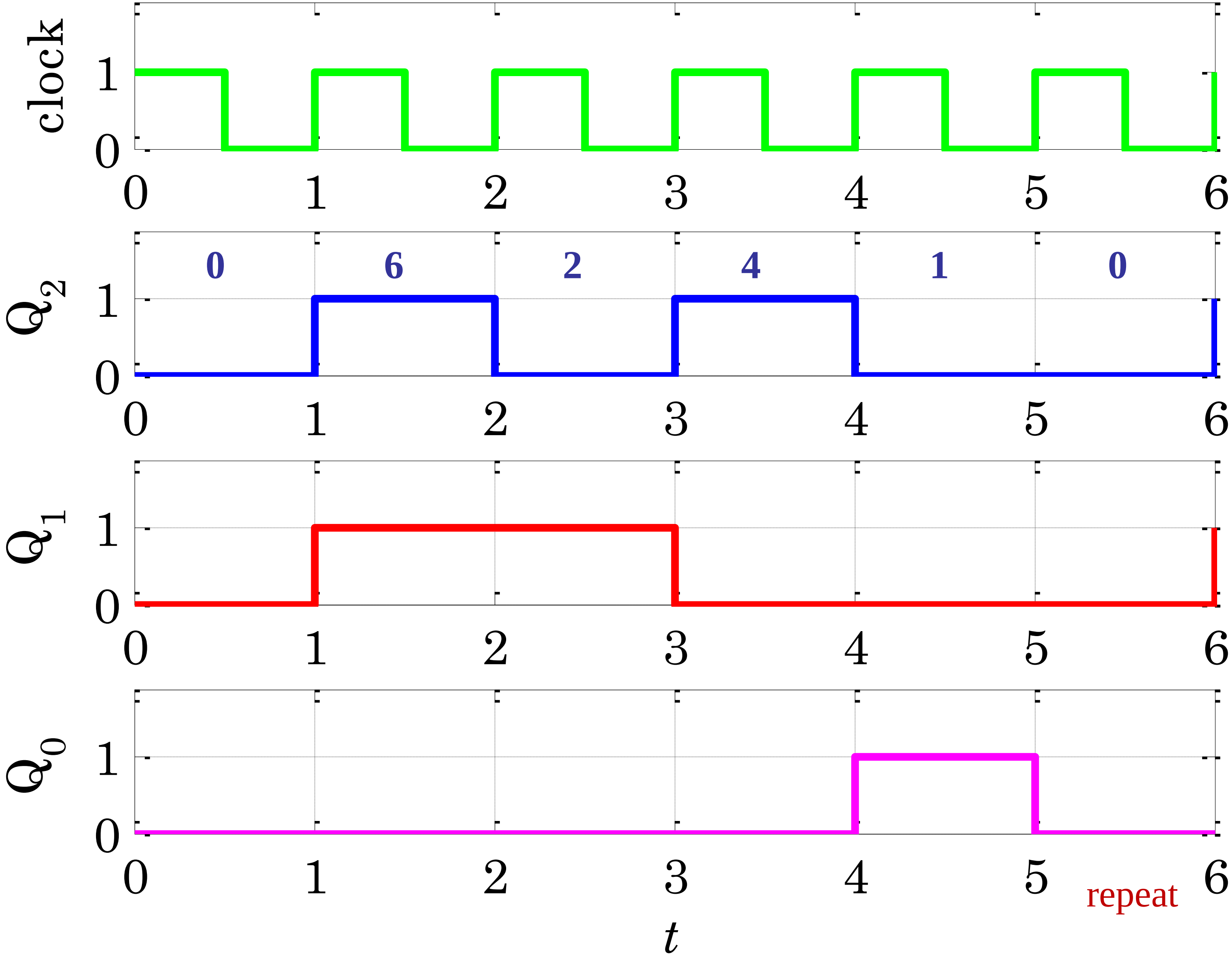
Example 7

showing two periods



scope output

timing diagram



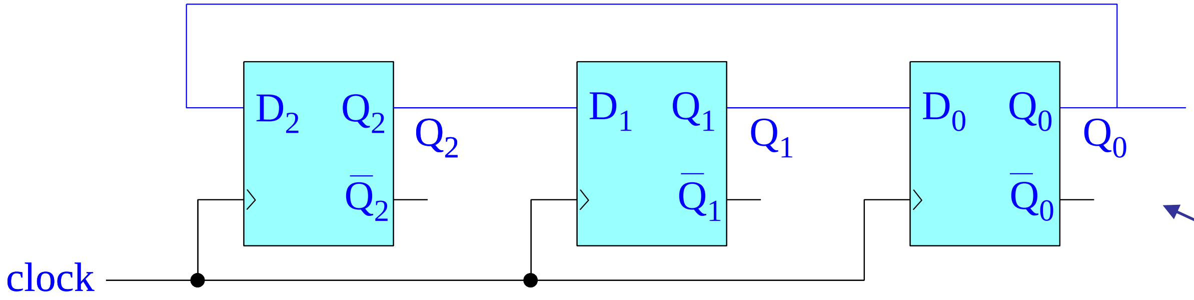
Example 7

Example 8 – Ring counters. Ring counters are specialized counters implemented by feedback shift registers in which the last bit is fed back to the beginning. The 3-bit and 4-bit cases are shown below, and they cycle through the following binary patterns:

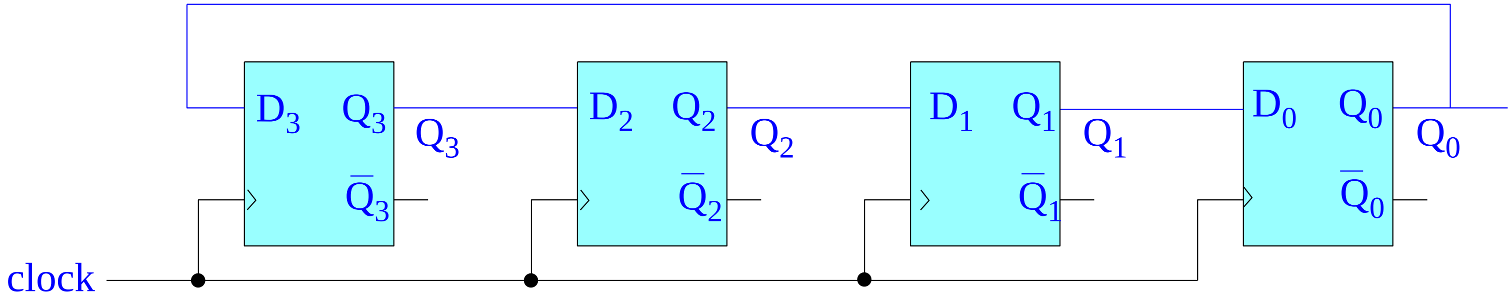
3-bit case: $Q_2Q_1Q_0$ = 100, 010, 001

4-bit case: $Q_3Q_2Q_1Q_0$ = 1000, 0100, 0010, 0001

similar to one-hot encoding



must be initialized to:
$Q_2Q_1Q_0$ = 100
$Q_3Q_2Q_1Q_0$ = 1000



The feedback structure of the counter can also be derived using K-maps assuming that the counter cycles through the patterns:

3-bit case: $Q_2Q_1Q_0$ = 100, 010, 001

4-bit case: $Q_3Q_2Q_1Q_0$ = 1000, 0100, 0010, 0001

for example, in the 3-bit case, we list the three desired patterns and the next ones, and treat all other patterns as “don’t’cares”,

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
4	1	0	0	0	1	0
2	0	1	0	0	0	1
1	0	0	1	1	0	0
0	0	0	0	x	x	x
3	0	1	1	x	x	x
5	1	0	1	x	x	x
6	1	1	0	x	x	x
7	1	1	1	x	x	x

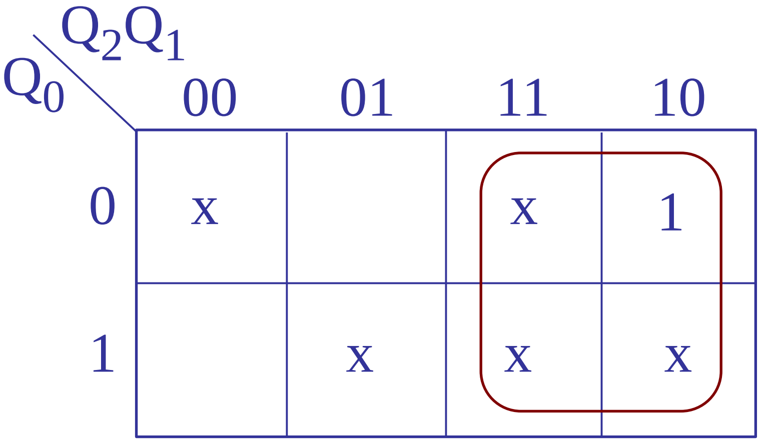
← repeat

← don’t cares

must be initialized to:
$Q_2Q_1Q_0$ = 100

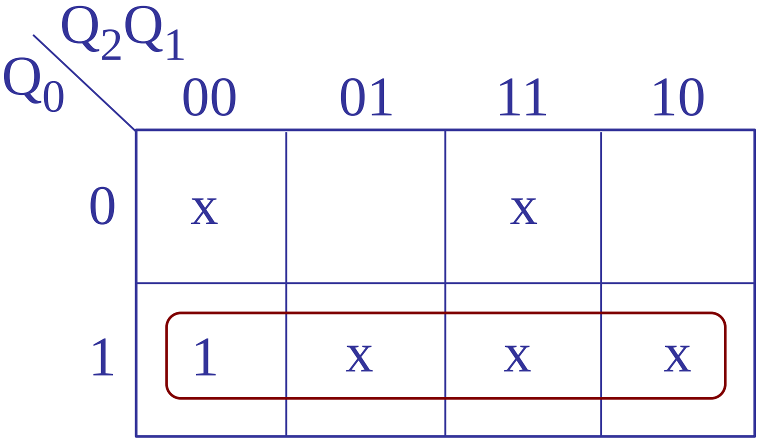
see p. 36 on how to initialize a counter

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
4	1	0	0	0	1	0
2	0	1	0	0	0	1
1	0	0	1	1	0	0
0	0	0	0	x	x	x
3	0	1	1	x	x	x
5	1	0	1	x	x	x
6	1	1	0	x	x	x
7	1	1	1	x	x	x



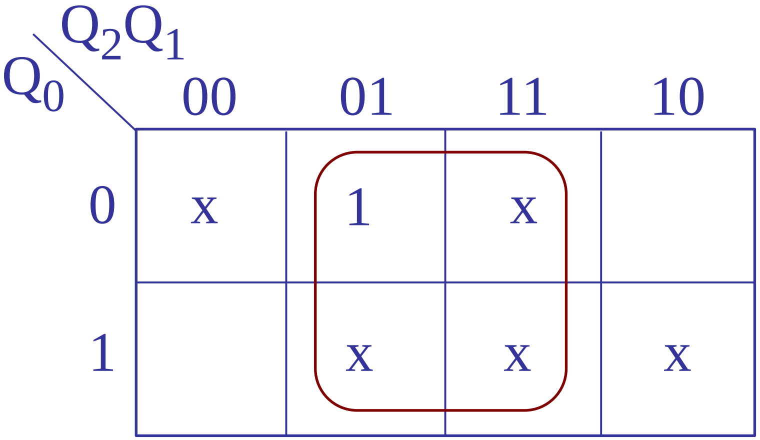
Q_0 \ Q_2Q_1	00	01	11	10
0	x		x	1
1		x	x	x

$Q_1{}^{\text{next}} = Q_2$



Q_0 \ Q_2Q_1	00	01	11	10
0	x		x	
1	1	x	x	x

$Q_2{}^{\text{next}} = Q_0$



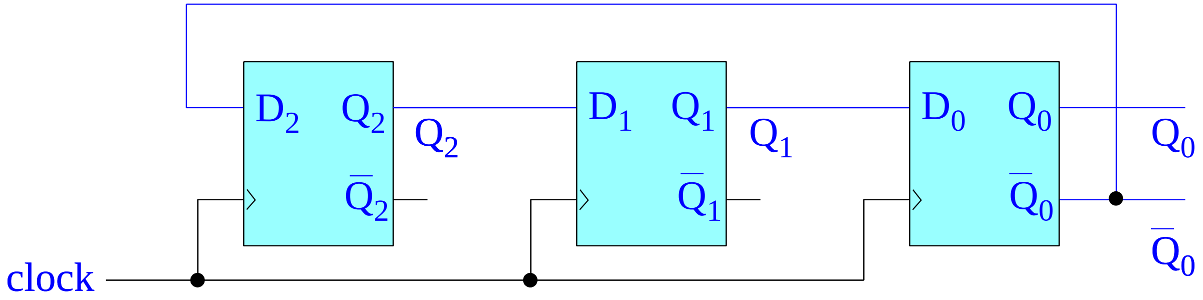
Q_0 \ Q_2Q_1	00	01	11	10
0	x	1	x	
1		x	x	x

$Q_0{}^{\text{next}} = Q_1$

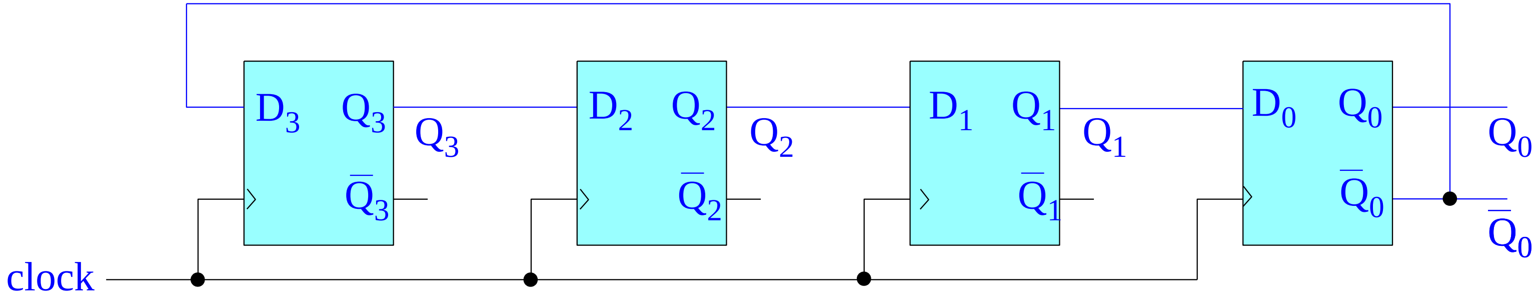
Example 9 – Johnson counters. Also known as twisted ring counters, they are specialized counters implemented by feedback shift registers in which the last bit is complemented and then fed back to the beginning. The 3-bit and 4-bit cases are shown below, and they cycle through the binary patterns:

3-bit case: $Q_2Q_1Q_0$ = 000, 100, 110, 111, 011, 001

4-bit case: $Q_3Q_2Q_1Q_0$ = 0000, 1000, 1100, 1110, 1111, 0111, 0011, 0001



must be initialized to:
$Q_2Q_1Q_0$ = 000
$Q_3Q_2Q_1Q_0$ = 0000



The feedback structure of these counters can be derived using K-maps assuming that the counter cycles through the specified patterns.

example, in the 3-bit case, we list the six desired patterns and the next ones, and treat all other patterns as “don’t’cares”,

$$Q_2Q_1Q_0 = 000,\ 100,\ 110,\ 111,\ 011,\ 001$$

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0	0
4	1	0	0	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	1	1
3	0	1	1	0	0	1
1	0	0	1	0	0	0
2	0	1	0	x	x	x
5	1	0	1	x	x	x

← repeat

← don’t cares

Example 9

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	1	0	0
4	1	0	0	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	1	1
3	0	1	1	0	0	1
1	0	0	1	0	0	0
2	0	1	0	x	x	x
5	1	0	1	x	x	x

Q_0 \ Q_2Q_1	00	01	11	10
0	1	x	1	1
1				x

$Q_2^{\text{next}} = Q_0'$

Q_0 \ Q_2Q_1	00	01	11	10
0		x	1	1
1			1	x

$Q_1^{\text{next}} = Q_2$

Q_0 \ Q_2Q_1	00	01	11	10
0		x	1	
1		1	1	x

$Q_0^{\text{next}} = Q_1$

Example 9

Example 10 – Linear feedback shift registers (LFSR). LFSRs have similar structure as ring counters but with more complicated feedback connections using XOR gates. They can be designed to generate maximal-length pseudo-random sequences (i.e., maximal-length, 2^n-1, for n-bit registers) and have many uses in error-control coding, cryptography, testing of digital circuits, and digital communications.

Consider the following repeating length-7 “pseudo-random” sequence, mentioned in Wakerly, Sect. 11.2.5, Table 11-5:

$$s_t = [\ 4, 2, 5, 6, 7, 3, 1, \ldots\ ]$$

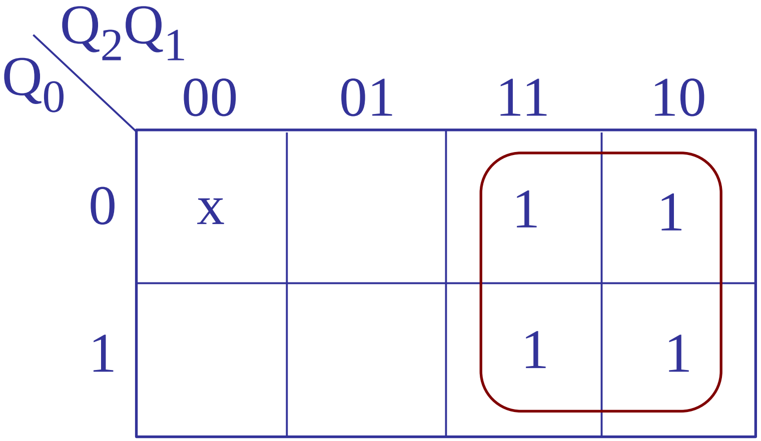
It turns out that this is a 3-bit maximal-length (2^3-1 = 7) LFSR sequence and, in this example, we will start with the sequence and derive its realization and structure as a linear feedback shift register.

Three D flip flops will be required which must be initialized to the first values of that sequence, that is,

$$Q_2Q_1Q_0 = 100 \quad \text{(representing the decimal 4)}$$

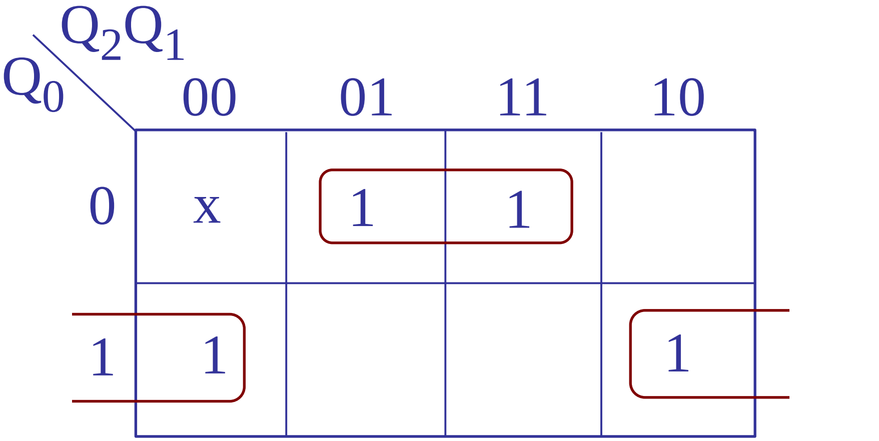
We begin by writing the truth table with 000 as a “don’t care entry”.

s_t	Q_2	Q_1	Q_0	next states Q_2	Q_1	Q_0	
4	1	0	0	0	1	0	
2	0	1	0	1	0	1	
5	1	0	1	1	1	0	
6	1	1	0	1	1	1	
7	1	1	1	0	1	1	
3	0	1	1	0	0	1	
1	0	0	1	1	0	0	← repeat
0	0	0	0	x	x	x	



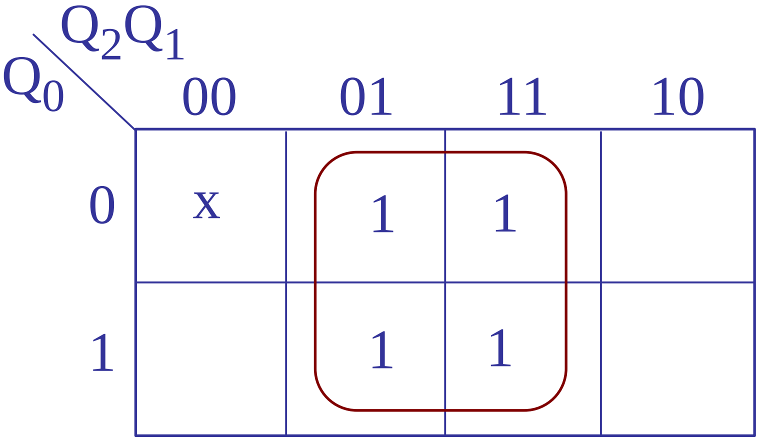
Q_0 Q_2Q_1	00	01	11	10
0	x		1	1
1			1	1

$Q_1^{next} = Q_2$



Q_0 Q_2Q_1	00	01	11	10
0	x	1	1	
1	1			1

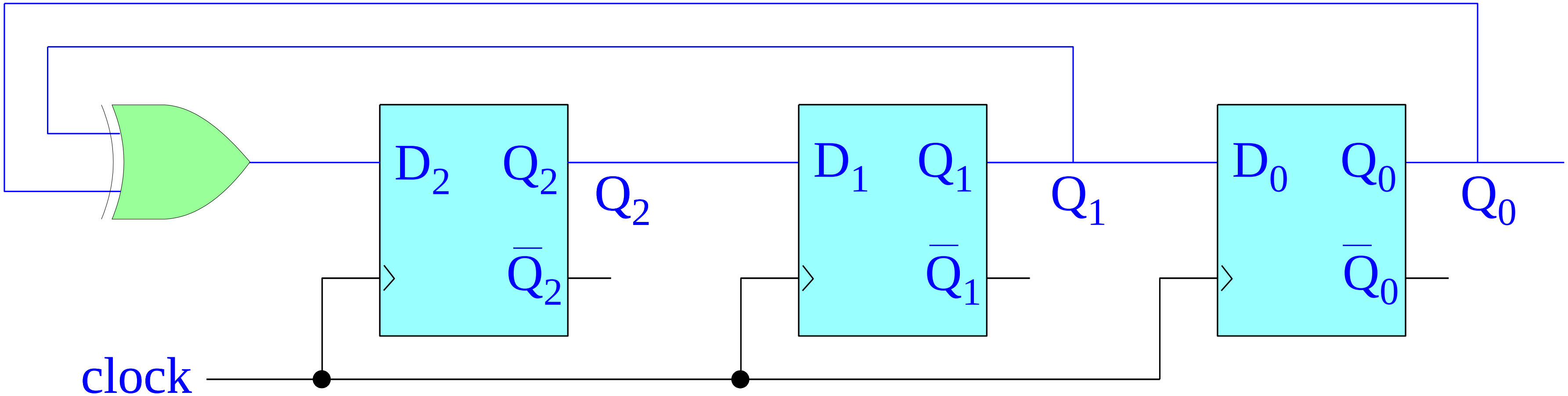
$Q_2^{next} = Q_1 Q_0' + Q_1' Q_0 = Q_1 \oplus Q_0$



Q_0 Q_2Q_1	00	01	11	10
0	x	1	1	
1		1	1	

$Q_0^{next} = Q_1$

LFSR realization



initialize:

$D_2 = 1$

$D_1 = 0$

$D_0 = 0$

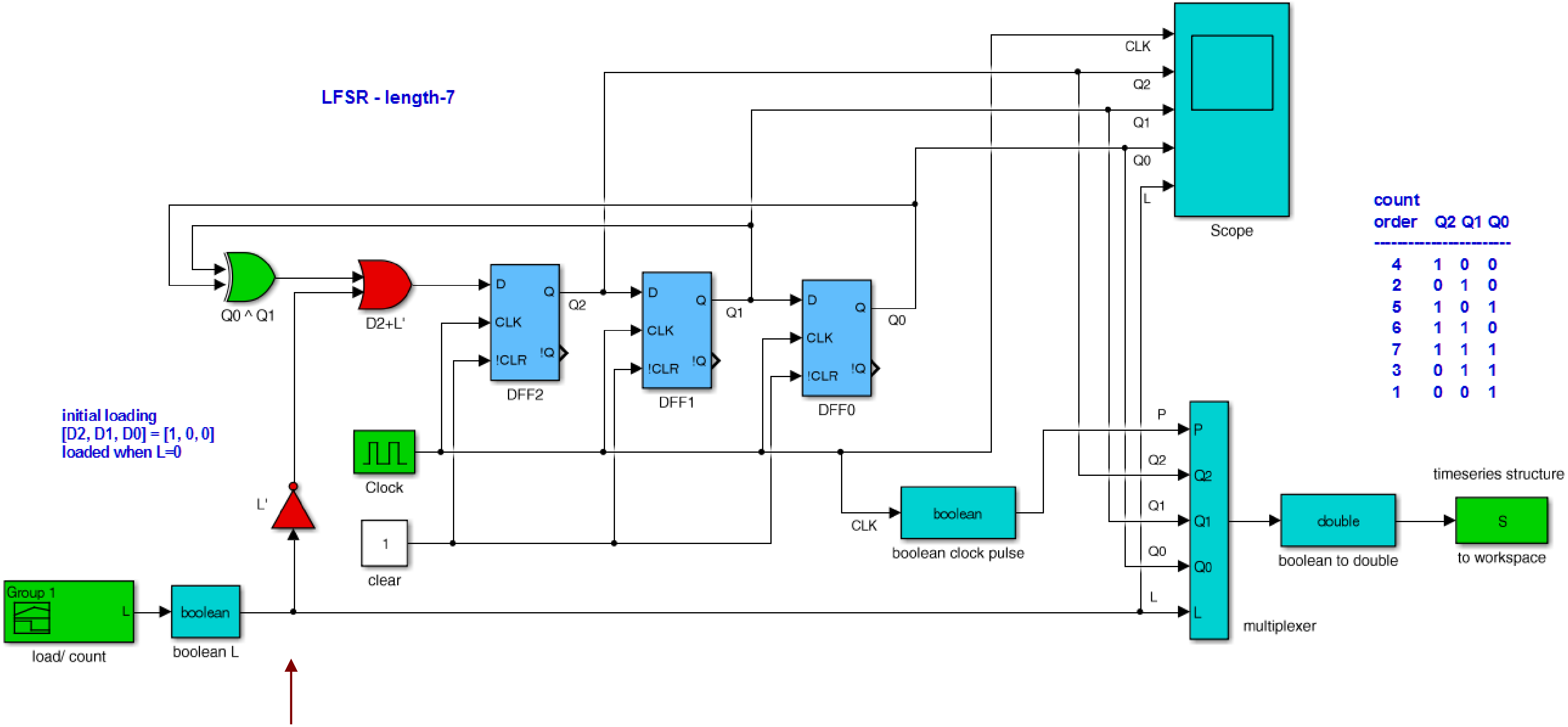
→

$D_2 = Q_2^{\text{next}} = Q_1 \oplus Q_0$

$D_1 = Q_1^{\text{next}} = Q_2$

$D_0 = Q_0^{\text{next}} = Q_1$

Simulink implementation



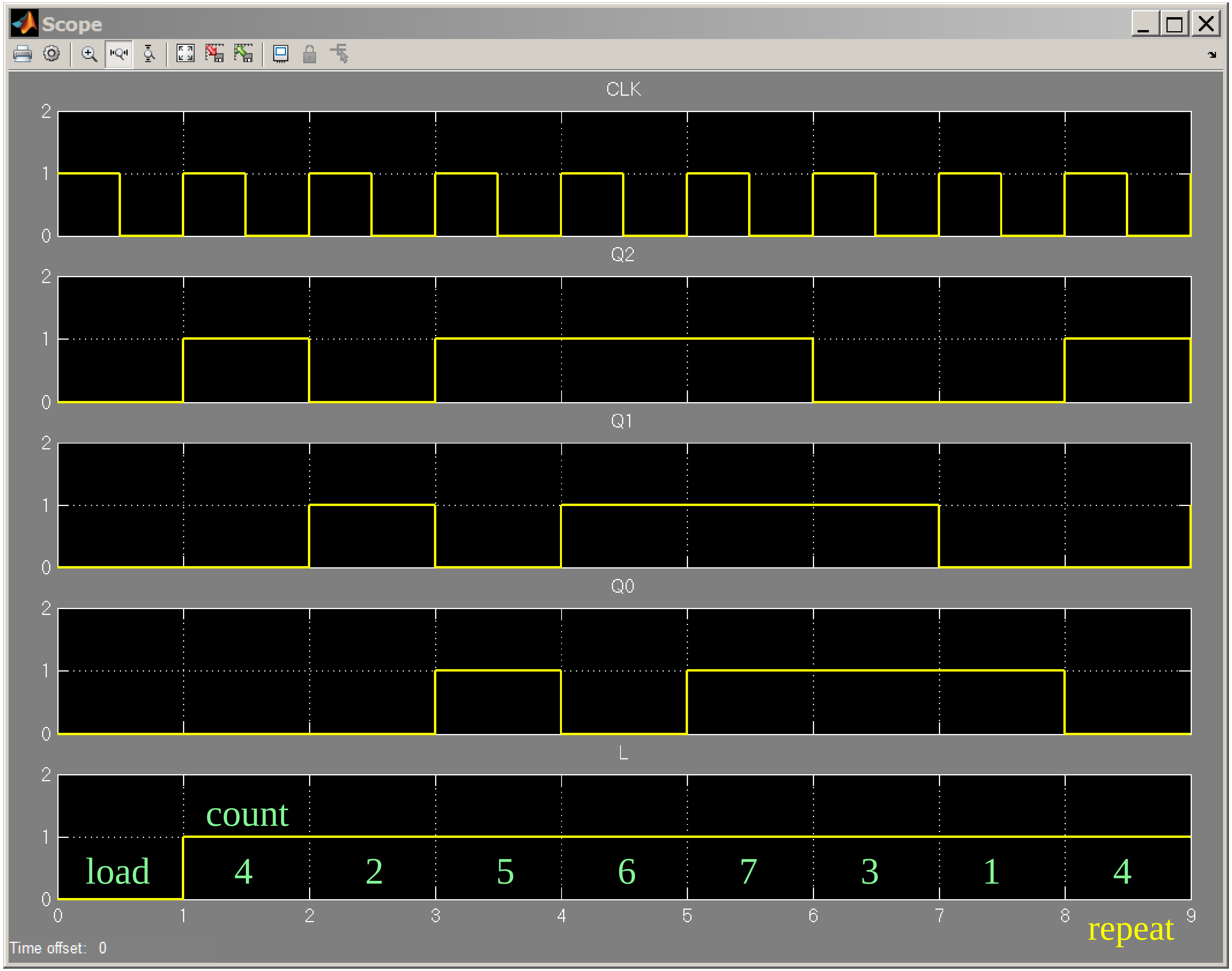
count order	Q2	Q1	Q0
4	1	0	0
2	0	1	0
5	1	0	1
6	1	1	0
7	1	1	1
3	0	1	1
1	0	0	1

initialization method is specific to this case,

$$D_2 = L' + Q_1 \oplus Q_0$$

load, $L = 0$
count, $L = 1$

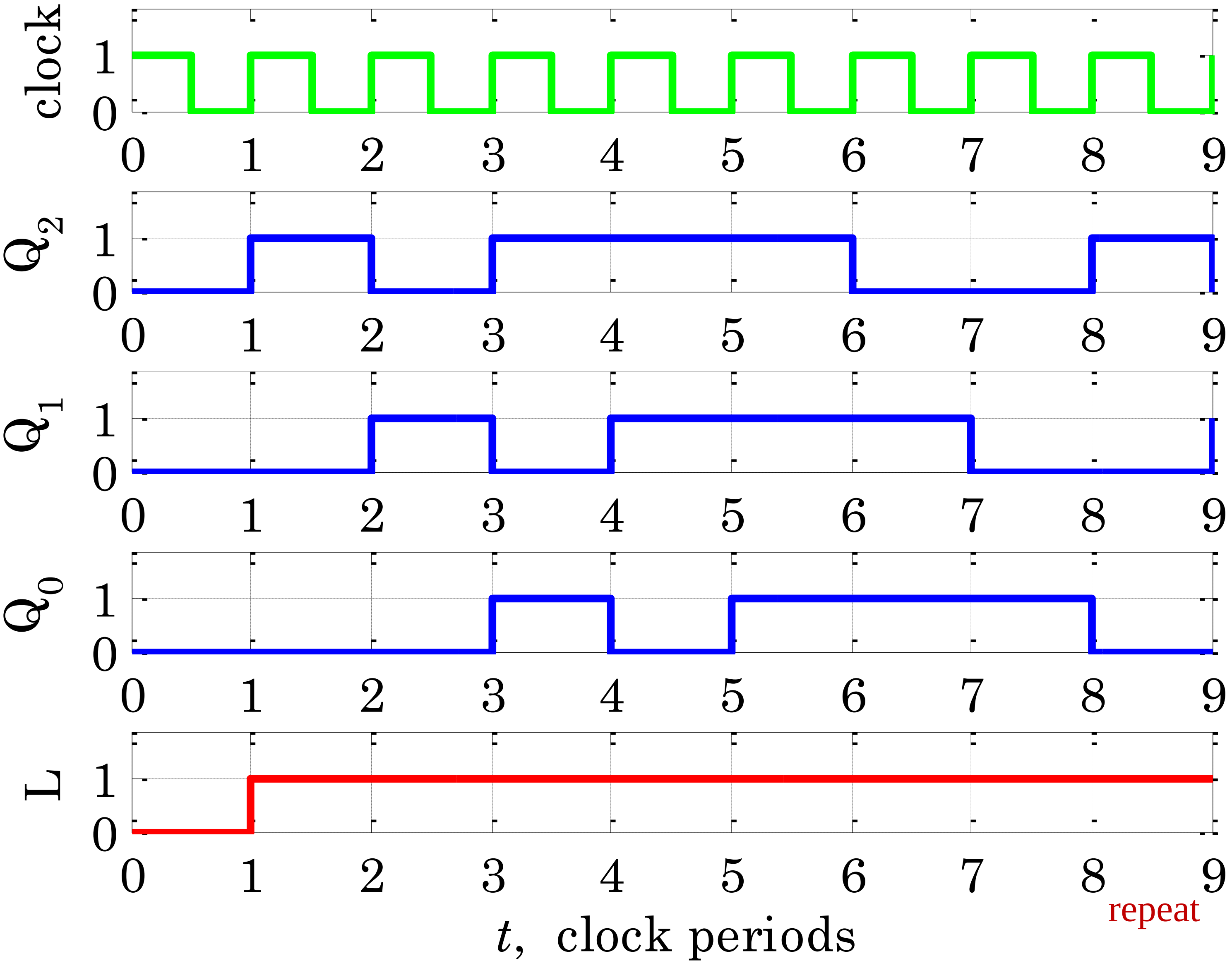
Example 10



scope output

Example 10

s_t = **load 4 2 5 6 7 3 1 4 . . .**



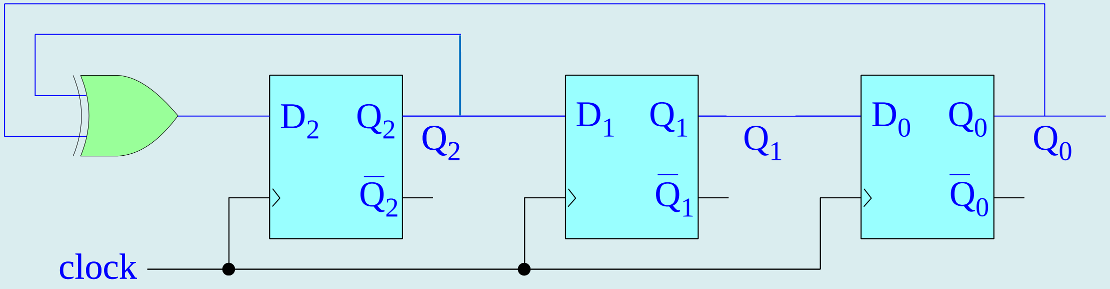
load, L=0
count, L=1

Example 10 - continued. Suppose we initialize at $Q_2Q_1Q_0$ = 011, and replace the next-state equations by,

$$Q_2^{next} = Q_2 \oplus Q_0$$

$$Q_1^{next} = Q_2$$

$$Q_0^{next} = Q_1$$



then, what sequence will be generated? (practice exam initializes to $Q_2Q_1Q_0$ = 101)

					next states	
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
0	0	0	0	x	x	x

initialize

					next states	
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
0	0	0	0	x	x	x

					next states	
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	0	1	0
2	0	1	0	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
0	0	0	0	x	x	x

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	0	1	0
2	0	1	0	0	0	1
1	0	0	1	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
0	0	0	0	x	x	x

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	0	1	0
2	0	1	0	0	0	1
1	0	0	1	1	0	0
4	1	0	0	*	*	*
*	*	*	*	*	*	*
*	*	*	*	*	*	*
0	0	0	0	x	x	x

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	0	1	0
2	0	1	0	0	0	1
1	0	0	1	1	0	0
4	1	0	0	1	1	0
6	1	1	0	*	*	*
*	*	*	*	*	*	*
0	0	0	0	x	x	x

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	0	1	0
2	0	1	0	0	0	1
1	0	0	1	1	0	0
4	1	0	0	1	1	0
6	1	1	0	1	1	1
7	1	1	1	*	*	*
0	0	0	0	x	x	x

				next states		
s_t	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
3	0	1	1	1	0	1
5	1	0	1	0	1	0
2	0	1	0	0	0	1
1	0	0	1	1	0	0
4	1	0	0	1	1	0
6	1	1	0	1	1	1
7	1	1	1	0	1	1
0	0	0	0	x	x	x

← repeat

$$Q_2^{next} = Q_2 \oplus Q_0$$

$$Q_1^{next} = Q_2$$

$$Q_0^{next} = Q_1$$

Example 10

Example 11 – Linear feedback shift registers (LFSR). Here, we consider a 4-bit LFSR with maximal length of $2^4 - 1 = 15$.

Again, we will start with the pseudo-random sequence itself and derive the proper feedback structure of the LFSR.

$$s_t = [\ 8, 12, 14, 15, 7, 11, 5, 10, 13, 6, 3, 9, 4, 2, 1\ ]$$

Four D flip flops will be required which must be initialized to the first values of that sequence, that is,

$$Q_3Q_2Q_1Q_0 = 1000 \quad \text{(representing the decimal 8)}$$

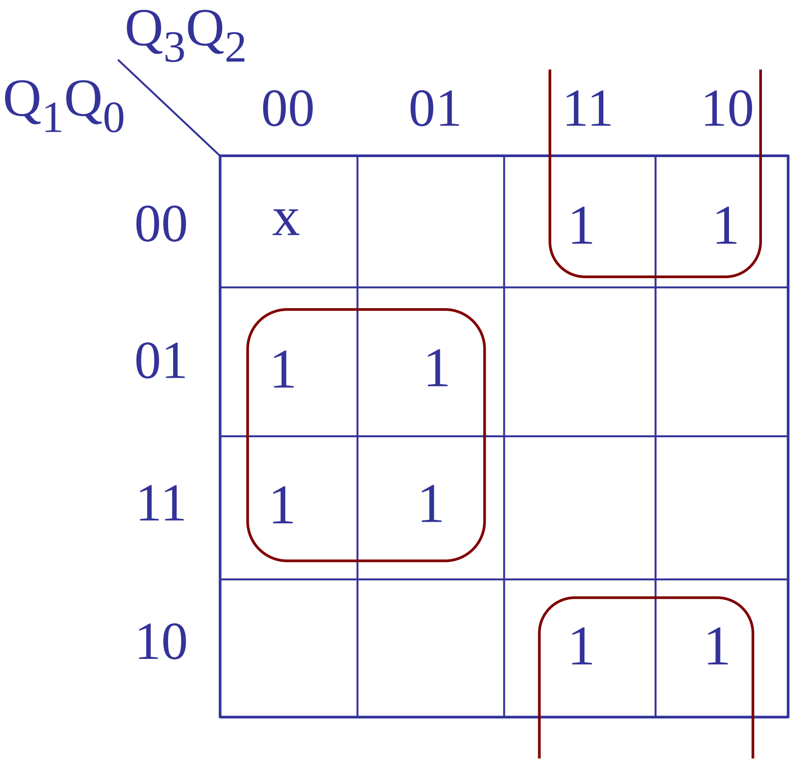
In general, the zero-pattern 0000 is not allowed in LFSRs because the particular feedback structure of the LFSR would cause the sequence to get stuck at zero – we will treat 0000 as a “don’t care entry”.

Brown & Vranesic, *Fundamentals of Digital Logic With Verilog Design*, 3/e, McGraw-Hill, 2014.

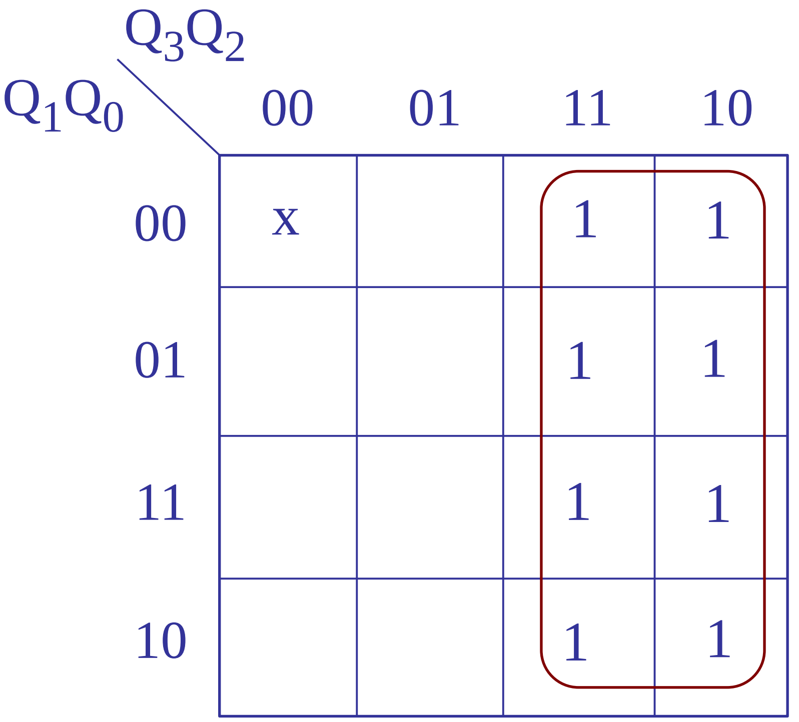
characteristic table

					next states				next
s_t	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	s_t
8	1	0	0	0	1	1	0	0	12
12	1	1	0	0	1	1	1	0	14
14	1	1	1	0	1	1	1	1	15
15	1	1	1	1	0	1	1	1	7
7	0	1	1	1	1	0	1	1	11
11	1	0	1	1	0	1	0	1	5
5	0	1	0	1	1	0	1	0	10
10	1	0	1	0	1	1	0	1	13
13	1	1	0	1	0	1	1	0	6
6	0	1	1	0	0	0	1	1	3
3	0	0	1	1	1	0	0	1	9
9	1	0	0	1	0	1	0	0	4
4	0	1	0	0	0	0	1	0	2
2	0	0	1	0	0	0	0	1	1
1	0	0	0	1	1	0	0	0	8
0	0	0	0	0	x	x	x	x	x

Example 11



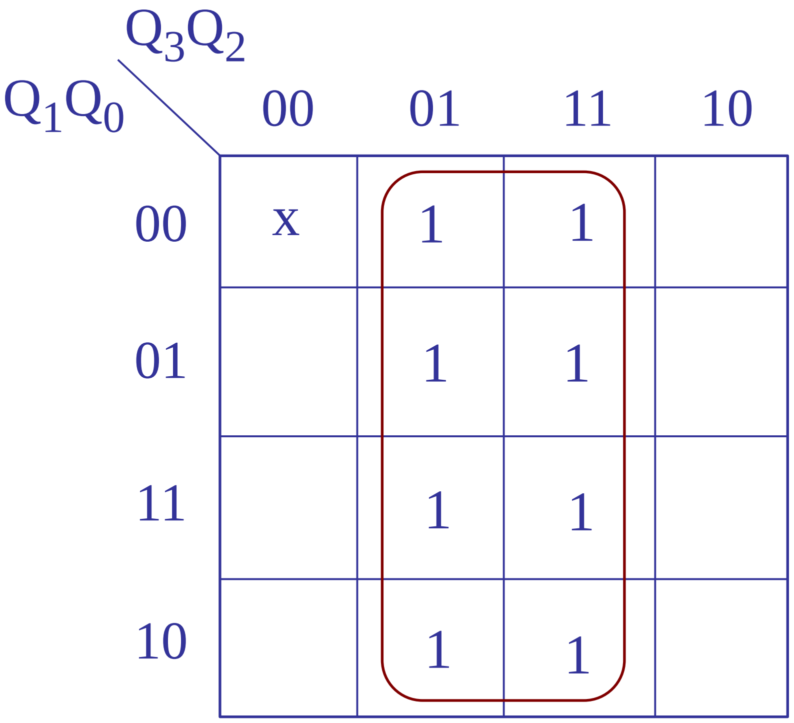
Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	x		1	1
01	1	1		
11	1	1		
10			1	1



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	x		1	1
01			1	1
11			1	1
10			1	1

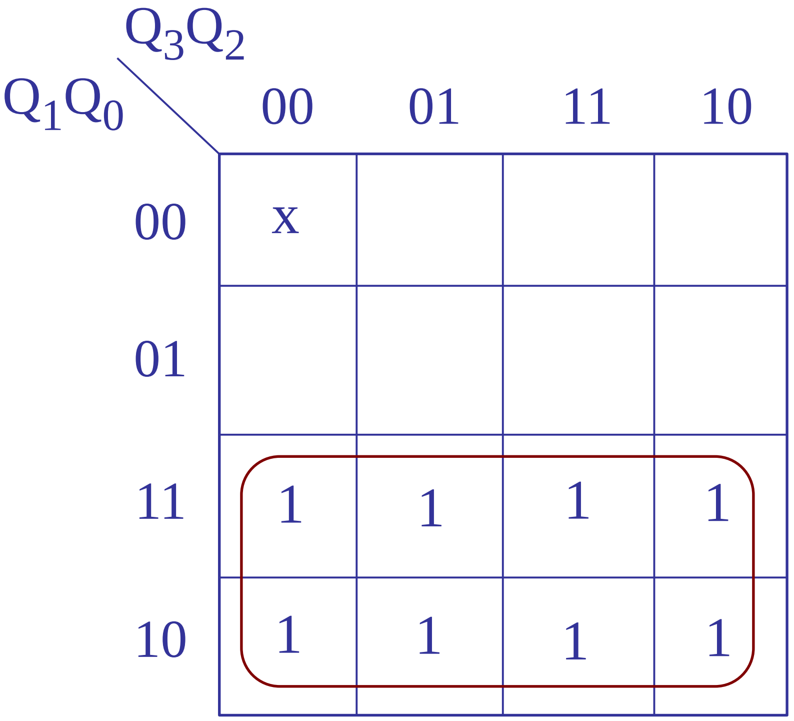
$$Q_3^{next} = Q_3\, Q_0' + Q_3'\, Q_0 = Q_3 \oplus Q_0$$

$$Q_2^{next} = Q_3$$



Q_3Q_2 Q_1Q_0	00	01	11	10
00	x	1	1	
01		1	1	
11		1	1	
10		1	1	

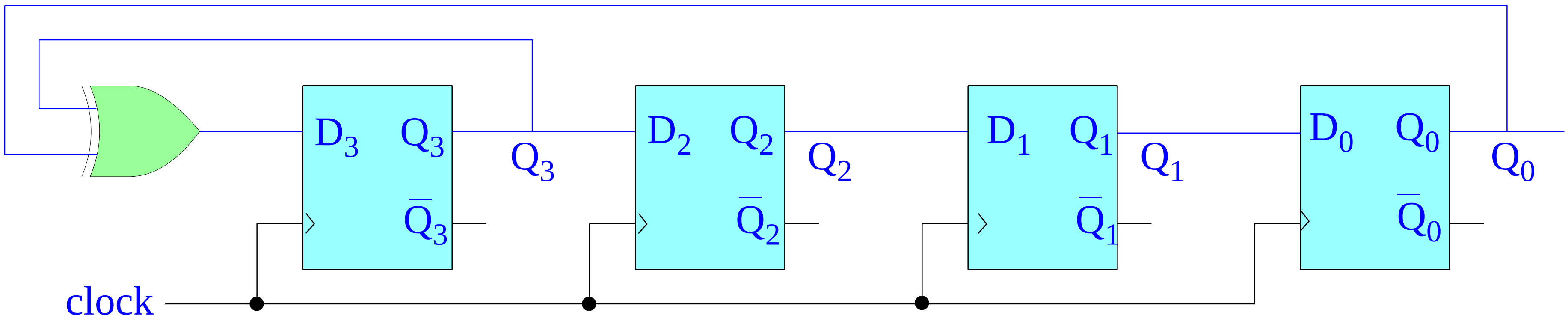
$Q_1^{next} = Q_2$



Q_3Q_2 Q_1Q_0	00	01	11	10
00	x			
01				
11	1	1	1	1
10	1	1	1	1

$Q_0^{next} = Q_1$

LFSR realization



initialize:

$D_3 = 1$

$D_2 = 0$

$D_1 = 0$

$D_0 = 0$

→

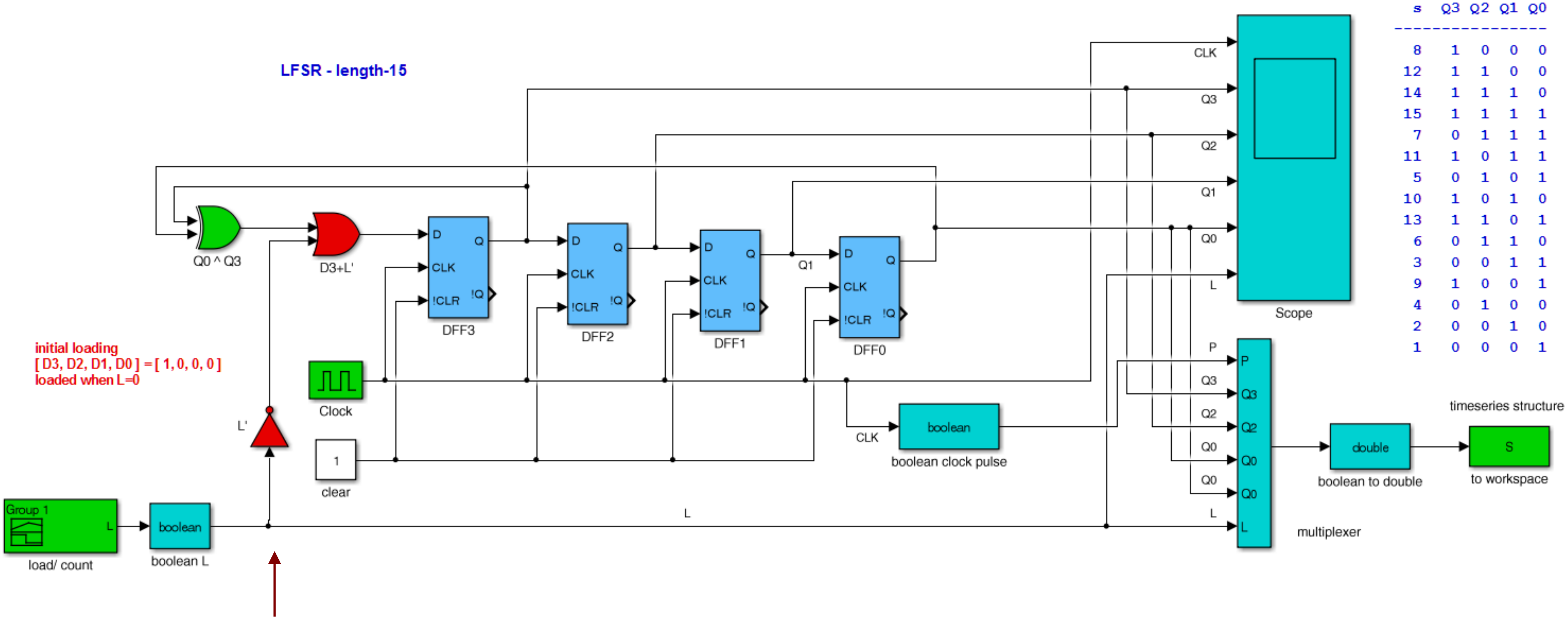
$D_3 = Q_3^{\text{next}} = Q_3 \oplus Q_0$

$D_2 = Q_2^{\text{next}} = Q_3$

$D_1 = Q_1^{\text{next}} = Q_2$

$D_0 = Q_0^{\text{next}} = Q_1$

Simulink implementation



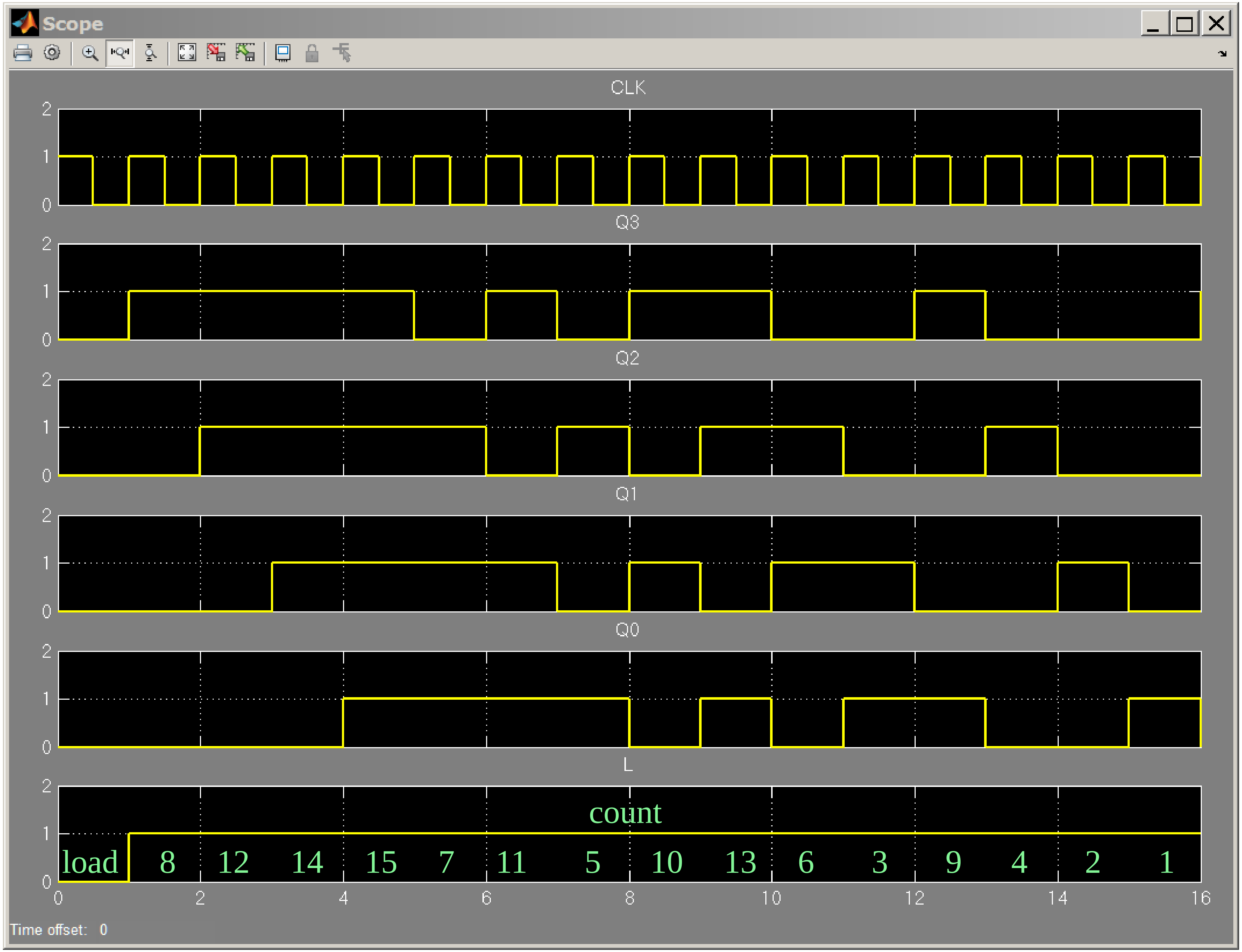
initial loading
[D3, D2, D1, D0] = [1, 0, 0, 0]
loaded when L=0

s	Q3	Q2	Q1	Q0
8	1	0	0	0
12	1	1	0	0
14	1	1	1	0
15	1	1	1	1
7	0	1	1	1
11	1	0	1	1
5	0	1	0	1
10	1	0	1	0
13	1	1	0	1
6	0	1	1	0
3	0	0	1	1
9	1	0	0	1
4	0	1	0	0
2	0	0	1	0
1	0	0	0	1

initialization method is specific to this case,

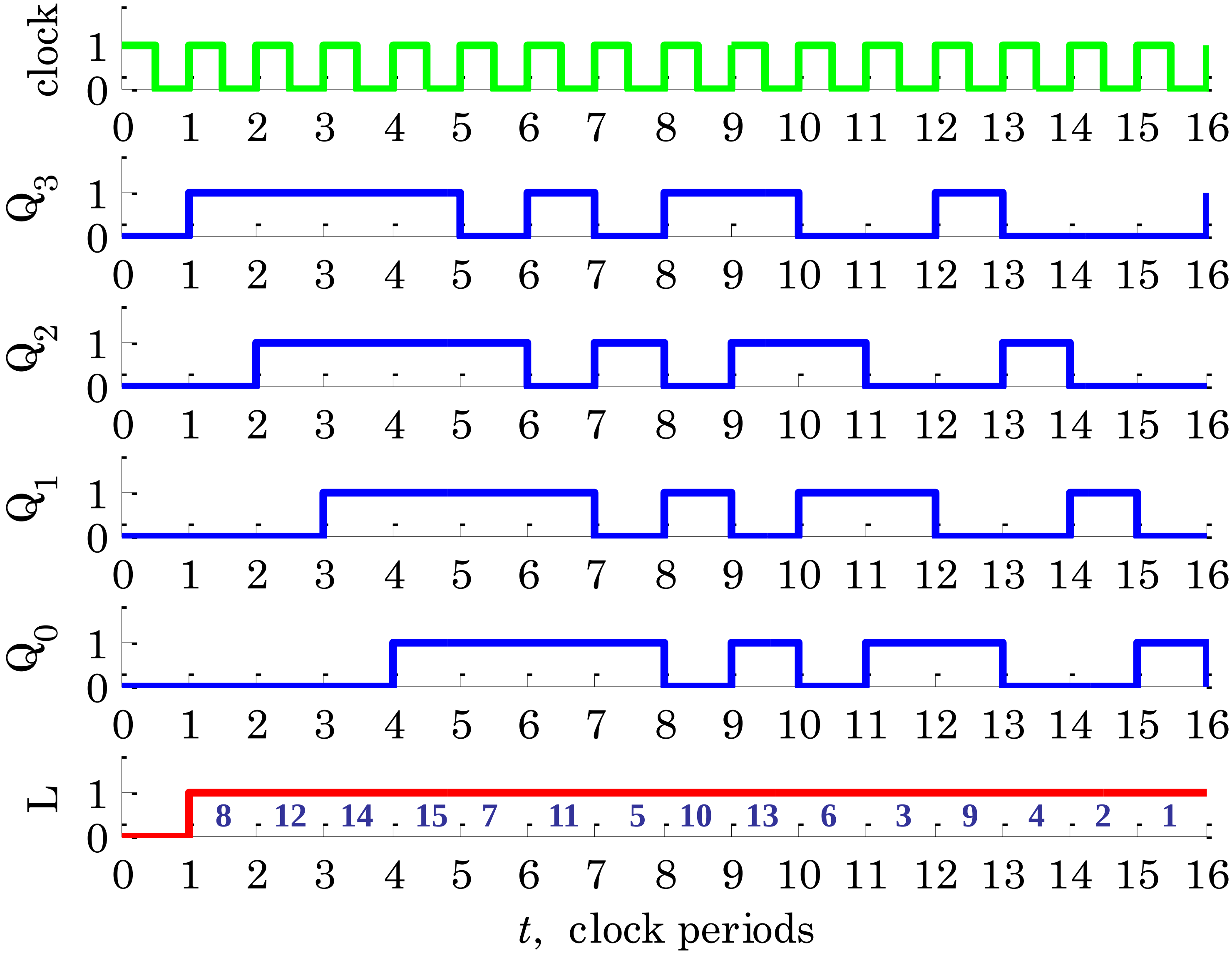
$$D_3 = L' + Q_3 \oplus Q_0$$

load, $L = 0$
count, $L = 1$


Scope
CLK
Q3
Q2
Q1
Q0
L
count
load
8
12
14
15
7
11
5
10
13
6
3
9
4
2
1
0
2
4
6
8
10
12
14
16
Time offset: 0

Example 11

s_t = 8 12 14 15 7 11 5 10 13 6 3 9 4 2 1



load, L=0
count, L=1

Example 11

Example 12 – BCD counter. A BCD counter can be thought of as generating a sub-sequence of the full 4-bit binary counter, that is, the sequence,

$$s_t = [\ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\ ]$$

Using K-maps, with the unused entries of the 4-bit counter treated as "don't cares", show that the next-state equations for the BCD counter are,

$$\boxed{\begin{aligned} Q_0^{\text{next}} &= Q_0' \\ Q_1^{\text{next}} &= Q_1 Q_0' + Q_3' Q_1' Q_0 \\ Q_2^{\text{next}} &= Q_2 \oplus (Q_1\, Q_0) \\ Q_3^{\text{next}} &= Q_3 Q_0' + Q_2 Q_1 Q_0 \end{aligned}}$$

Implement the counter in Simulink and generate a timing diagram. The required characteristic table is shown on the next page. Moreover, compute the actual characteristic table based on these equations, noting that some of the "don't care" entries were not used.

BCD counter

characteristic table

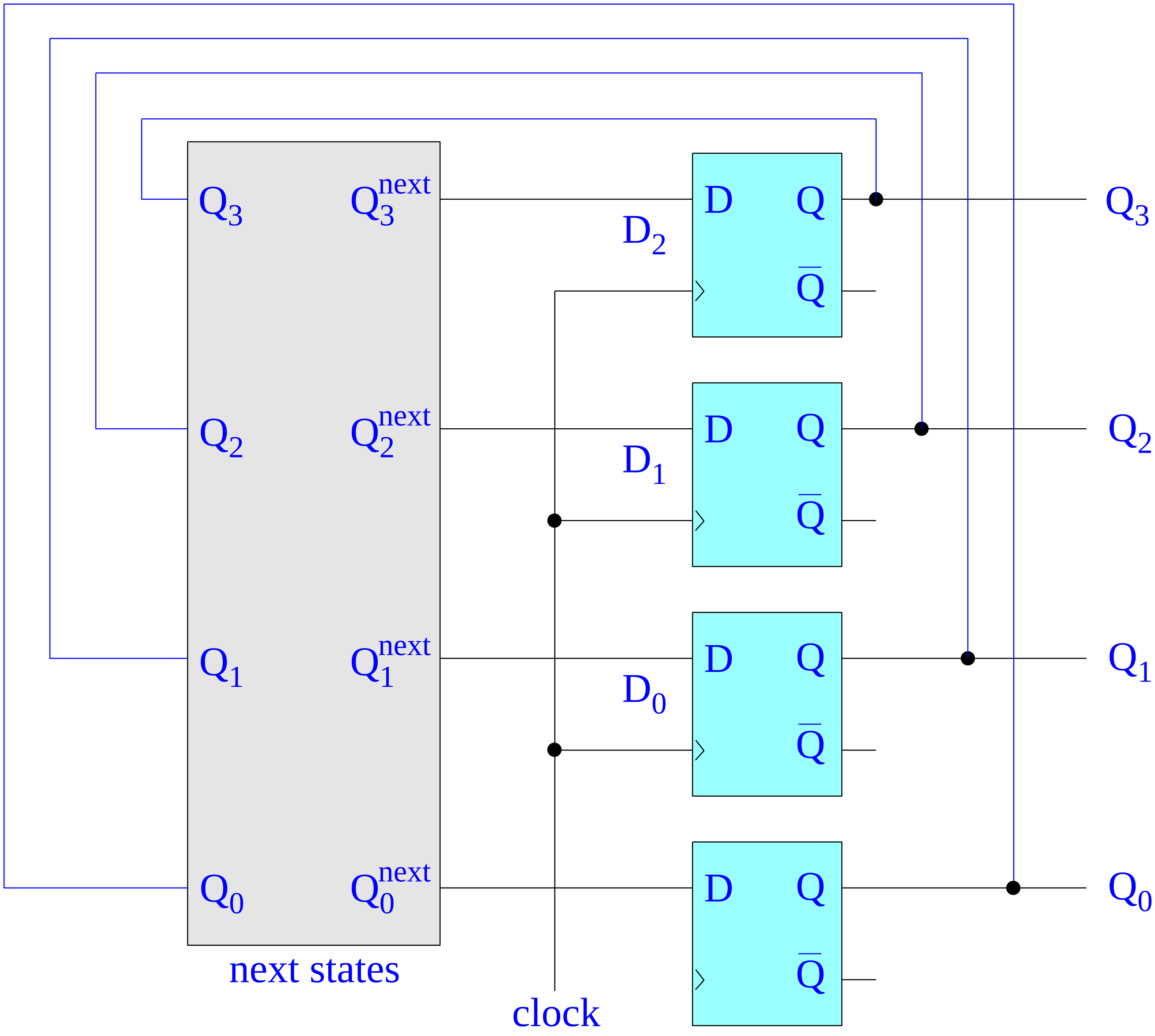
					next states			
s_t	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	0	0	0	0
10	1	0	1	0	x	x	x	x
11	1	0	1	1	x	x	x	x
12	1	1	0	0	x	x	x	x
13	1	1	0	1	x	x	x	x
14	1	1	1	0	x	x	x	x
15	1	1	1	1	x	x	x	x

← repeat

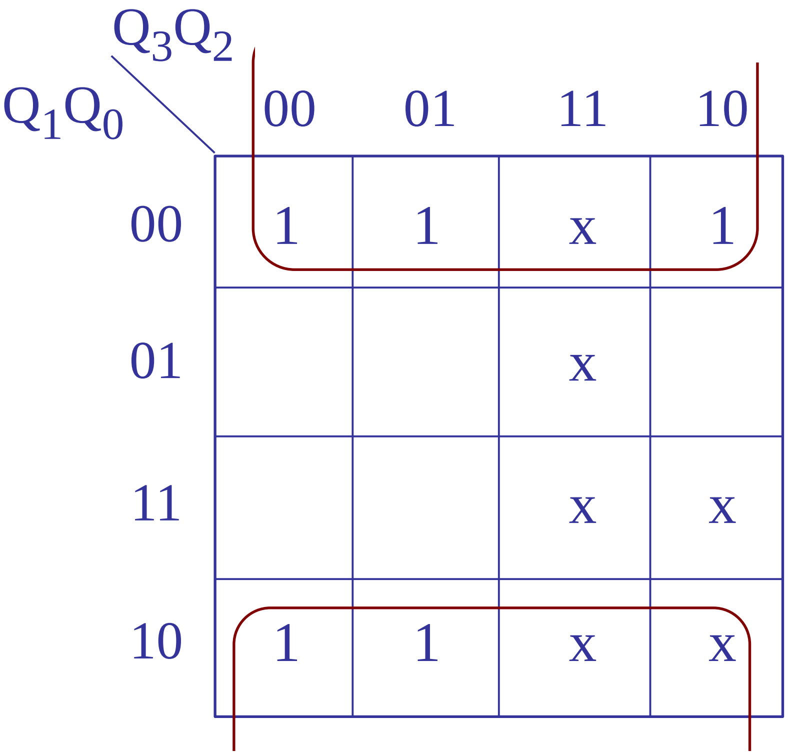
← don't cares

Example 12

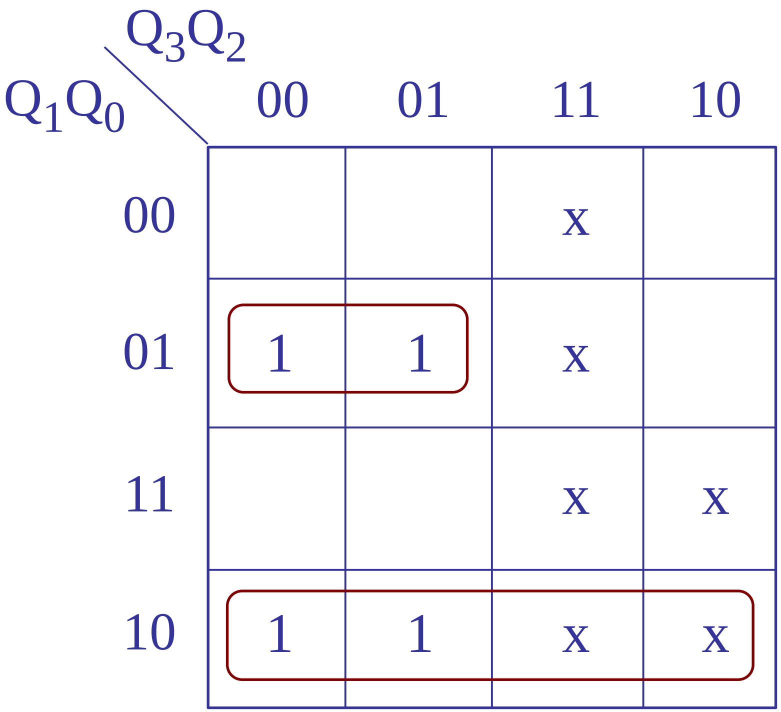
BCD counter



Example 12



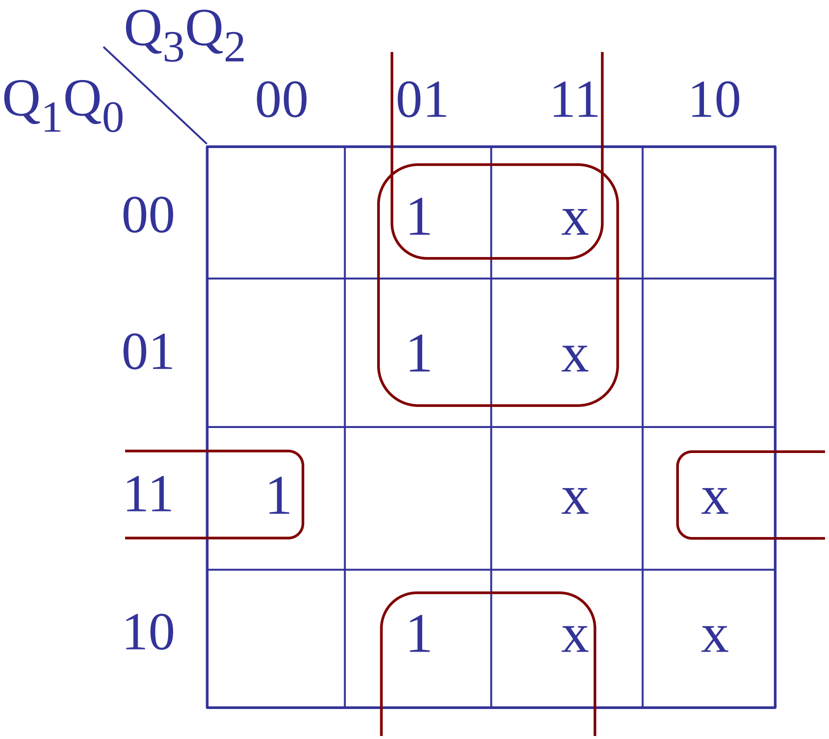
Q_1Q_0 \ Q_3Q_2	00	01	11	10
00	1	1	x	1
01			x	
11			x	x
10	1	1	x	x



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00			x	
01	1	1	x	
11			x	x
10	1	1	x	x

$$Q_0^{next} = Q_0'$$

$$Q_1^{next} = Q_1Q_0' + Q_3'Q_1'Q_0$$

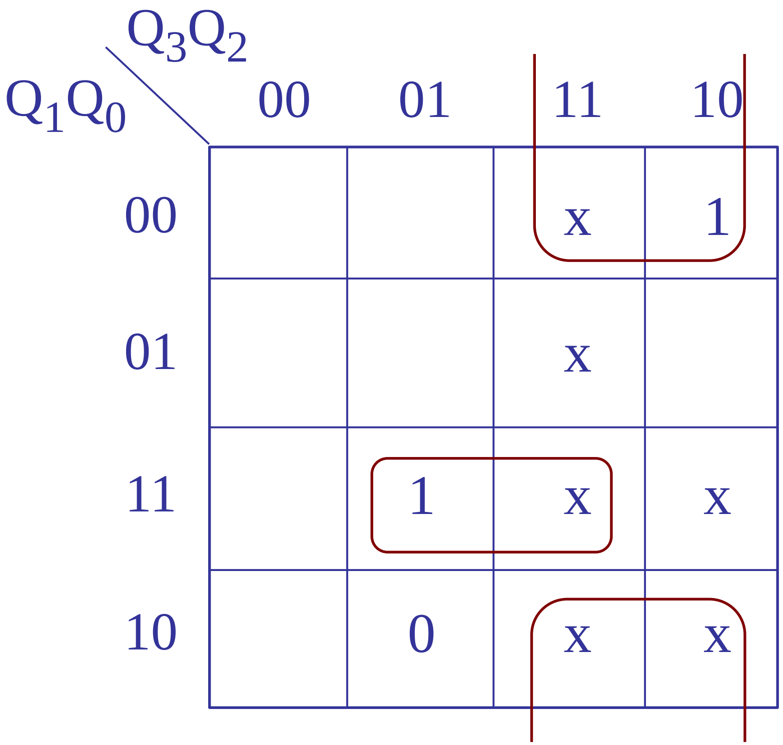


Q_1Q_0 \ Q_3Q_2	00	01	11	10
00		1	x	
01		1	x	
11	1		x	x
10		1	x	x

$$Q_2^{next} = Q_2\, Q_1' + Q_2\, Q_0' + Q_2'Q_1Q_0$$

$$= Q_2\, (Q_1' + Q_0') + Q_2'Q_1Q_0$$

$$= Q_2 \oplus (Q_1Q_0)$$



Q_1Q_0 \ Q_3Q_2	00	01	11	10
00			x	1
01			x	
11		1	x	x
10		0	x	x

$$Q_3^{next} = Q_3Q_0' + Q_2Q_1Q_0$$

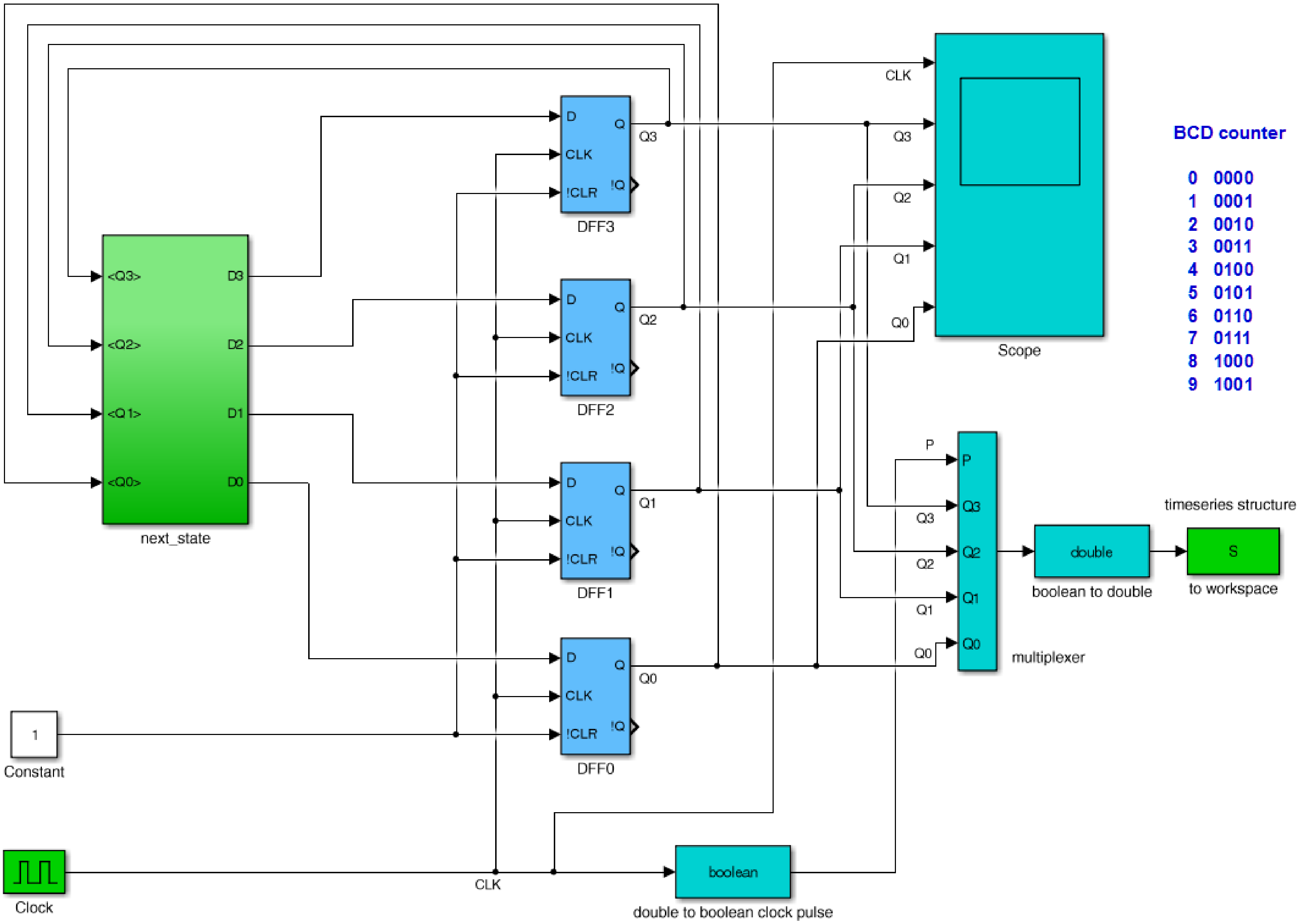
BCD counter

actual truth table

					next states			
s_t	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	0	0	0	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	0	1	0	0
12	1	1	0	0	1	1	0	1
13	1	1	0	1	0	1	0	0
14	1	1	1	0	1	1	1	1
15	1	1	1	1	1	0	0	0

← repeat

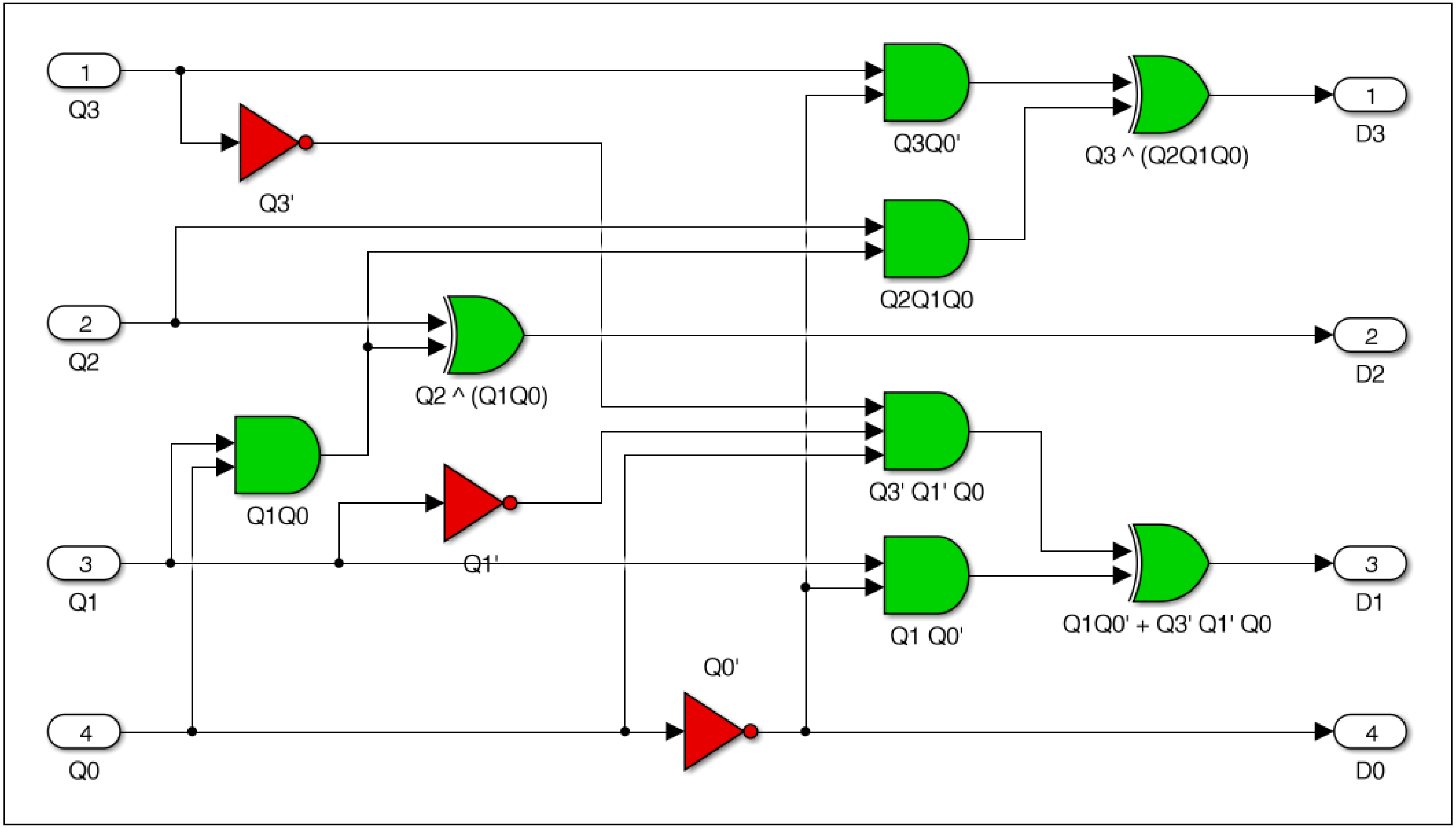
← some don’t cares were not used



BCD counter

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Example 12



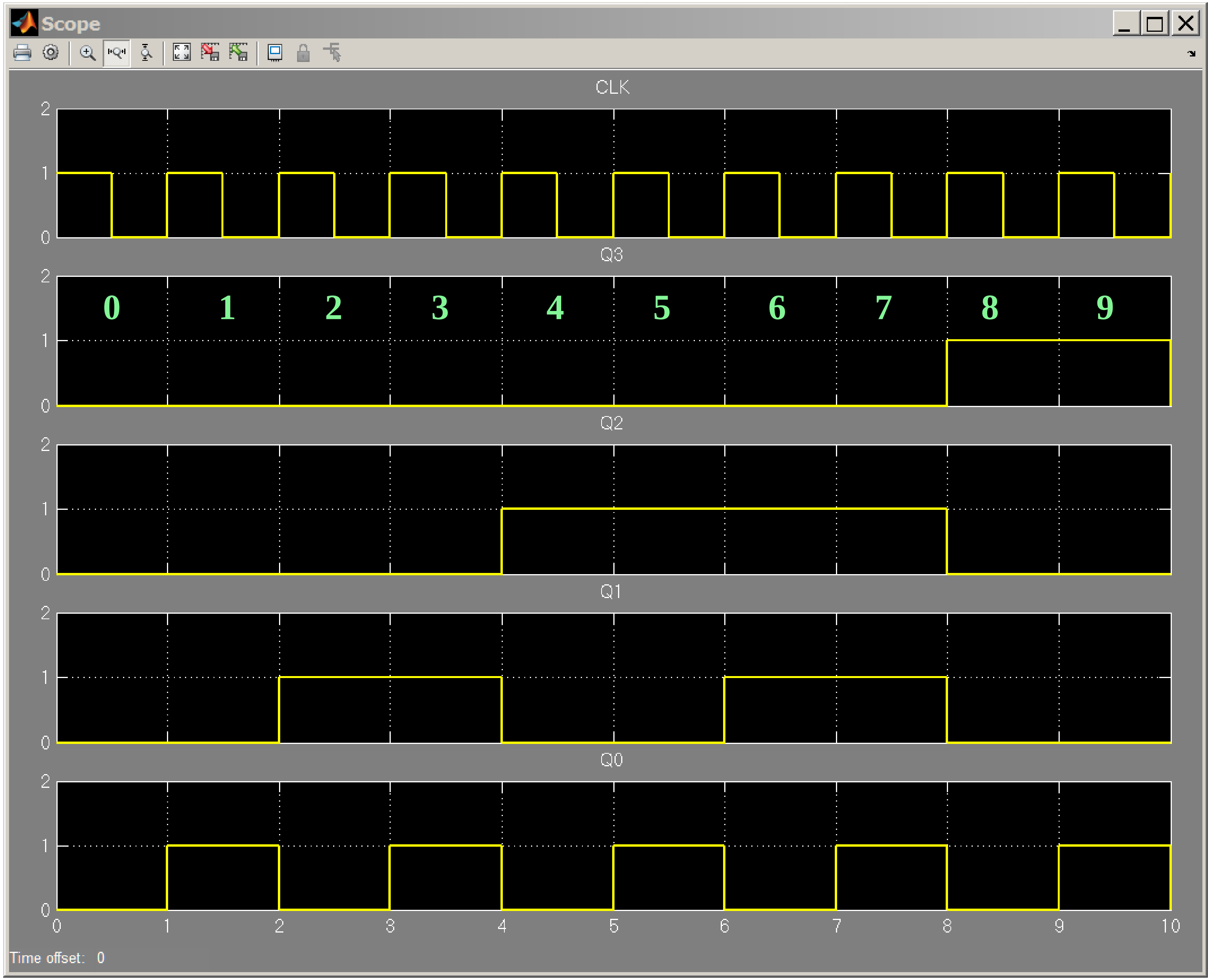
next-state subfunction

$$Q_0^{next} = Q_0'$$

$$Q_1^{next} = Q_1Q_0' + Q_3'Q_1'Q_0$$

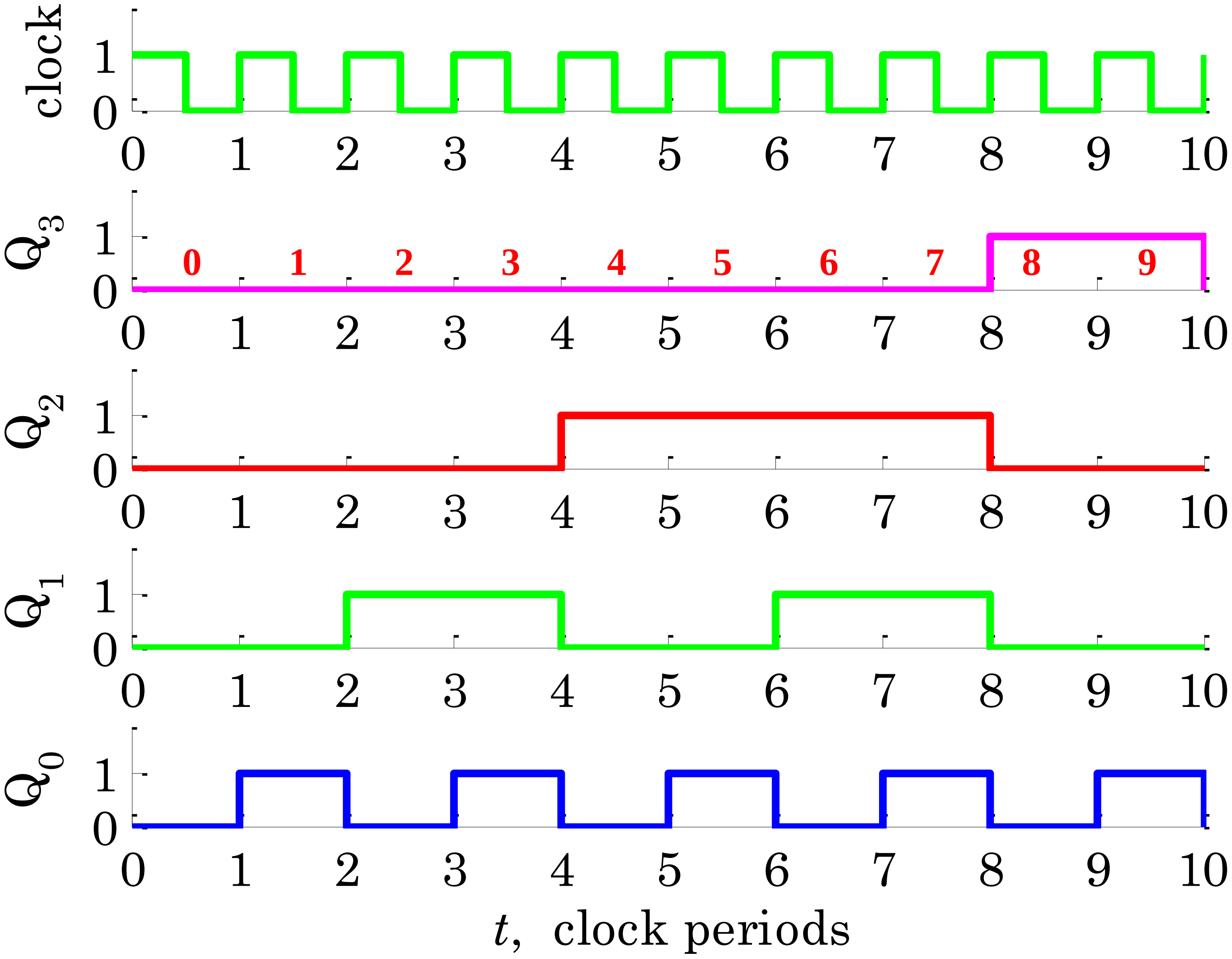
$$Q_2^{next} = Q_2 \oplus (Q_1 Q_0)$$

$$Q_3^{next} = Q_3Q_0' + Q_2Q_1Q_0$$



scope output

timing diagram


clock
Q_3
Q_2
Q_1
Q_0
t, clock periods