Rutgers University
School of Engineering

Fall 2022

332:231 – Digital Logic Design

Sophocles J. Orfanidis
ECE Department
orfanidi@rutgers.edu

Unit 8 – Finite State Machines

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:

- Finite state machines (FSM)
- Mealy vs. Moore FSMs
- State diagrams
- State tables
- State assignment & encoding
- Gate-level implementation with D flip-flops
- Design Examples

Contents:

1. Finite state machines (FSM)
2. Mealy vs. Moore FSMs, design steps
3. State diagrams
4. State tables, compact and conventional forms
5. State assignment & encoding (plain binary, Gray, one-hot)
6. Gate-level implementation with D flip-flops
7. Examples: design of a car alarm system
   design of a 2-bit counter with input & output
   analyzing FSM block diagrams
   cruise control system, Moore & Mealy versions
   sequence recognizers
   state reduction
   electronic keypad lock
   vending machine
   elevator controller

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.

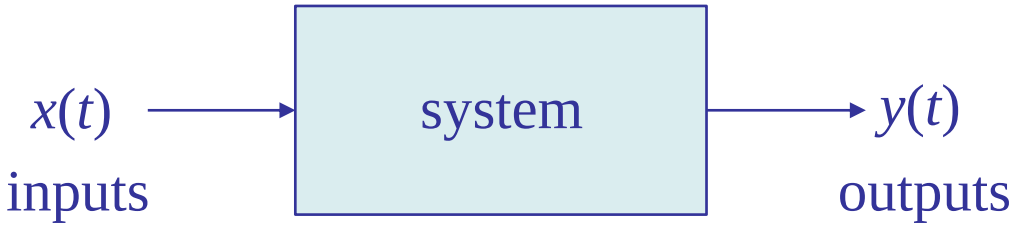
M. Mano, C. R. Kime, and T. Martin, *Logic and Computer Design Fundamentals*, 5/e, Pearson, 2016.

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.

sequential circuits and finite state-machines (FSM) are special cases of dynamic systems



In general, a system can be defined by specifying the I/O computational rule that determines the output signal $y(t)$ from the input signal $x(t)$.

The time variable t can be continuous or discrete. The system can be linear or non-linear, time-invariant or time-varying, and can be described by differential or difference equations.

State Machines

State-space realizations, also known as state-space models, have a very large number of applications in many diverse fields, such as,

- digital logic design
- differential equations for physical systems
- system theory
- electric circuits
- linear systems
- digital signal processing
- control systems
- communication systems
- predictive analytics
- biomedical signal processing
- geophysical signal processing
- aerospace engineering
- military systems
- statistics and time series analysis
- econometrics and financial engineering

State Machines

State-space realizations are very powerful representations of systems (linear or nonlinear, time-invariant or not).

The system is described by a set of internal states at each time instant t, denoted for example by $Q(t)$, and these states are used to compute the current output $y(t)$ in terms of the current input $x(t)$, and then update the states to their next values, $Q(t+1)$, so that they can be used at time $t+1$ (or, more generally at, $t+\Delta t$).

In other words, the system's time evolution is described iteratively by a computational algorithm of the form,

for each time instant t, do:

$$y(t) = G\big(x(t),\, Q(t)\big) \qquad \text{(compute output)}$$

$$Q(t+1) = F\big(x(t),\, Q(t)\big) \qquad \text{(update state)}$$

[to get started, one needs to know the initial state, $Q(0)$]

State Machines

For example, in going from time t to time t+2, one carries out the steps:

at time t,

$$y(t) = G(x(t), Q(t))$$

$$Q(t+1) = F(x(t), Q(t))$$

at time t+1,

$$y(t+1) = G(x(t+1), Q(t+1))$$

$$Q(t+2) = F(x(t+1), Q(t+1))$$

at time t+2,

$$y(t+2) = G(x(t+2), Q(t+2))$$

$$Q(t+3) = F(x+2), Q(t+2))$$

etc.

$F( )$ and $G( )$ depend on application

in DLD, F is referred to as next-state logic, or excitation logic

and, G is referred to as output logic

from here on, we'll use the simplified notation,

$$x_t\ ,\ y_t\ ,\ Q_t$$

for

$$x(t),\ y(t),\ Q(t)$$

State Machines

It should be emphasized that the updated state Q_{t+1} is being computed at time t, and becomes available at time t, replacing Q_t , but it is saved until it is used later at time t+1.

The computations can be cast as a repetitive algorithm, in which the present state is overwritten by the next state.

initialize state Q (typically at t=0), then,

at each time t, do,

$y_t = G(x_t,Q)$

$Q = F(x_t,Q)$

or,

at each time t, do,

$y_t = G(x_t,Q)$

$Q_{\text{next}} = F(x_t,Q)$

$Q = Q_{\text{next}}$

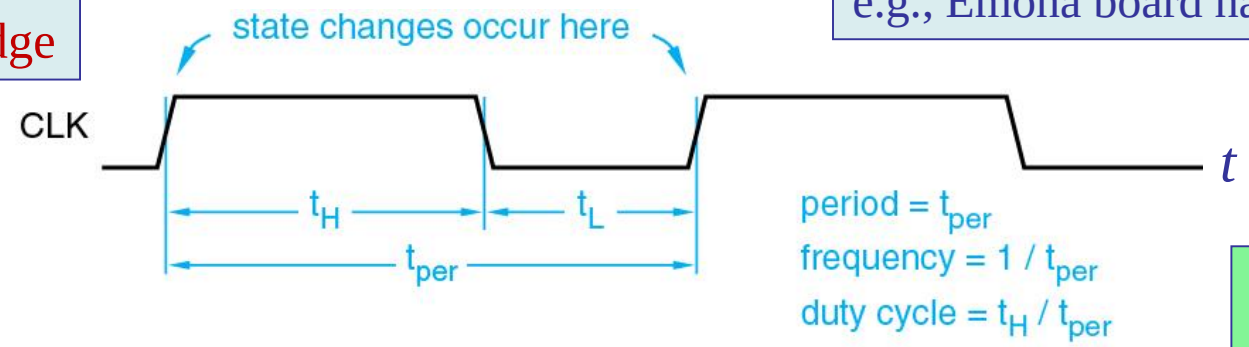
State Machines

We are assuming that time is discretized in units of 1, which means one clock period, so that t+1 means one clock period ahead of t.

In DLD (essentially all) sequential circuits are synchronously driven by a clock, and the state changes occur during the rising edge of the clock period (or, alternatively - but less commonly - during the falling edge.)

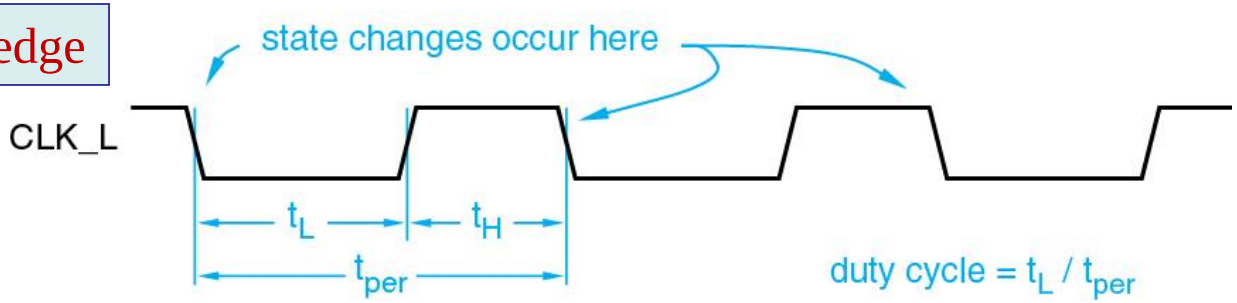
clock period depends on application, e.g., Emona board has 10 μsec period

rising edge



clock duty cycles are usually, 50%

falling edge



State Machines

State machines in DLD fall into two general families:

Moore type:

output equation depends only on Q, i.e., $y = G(Q)$

Mealy type:

output equation depends on both x and Q, i.e., $y = G(x, Q)$

for each time instant t, do:

$$y_t = G(x_t, Q_t)$$
$$Q_{t+1} = F(x_t, Q_t)$$

simplified notation →

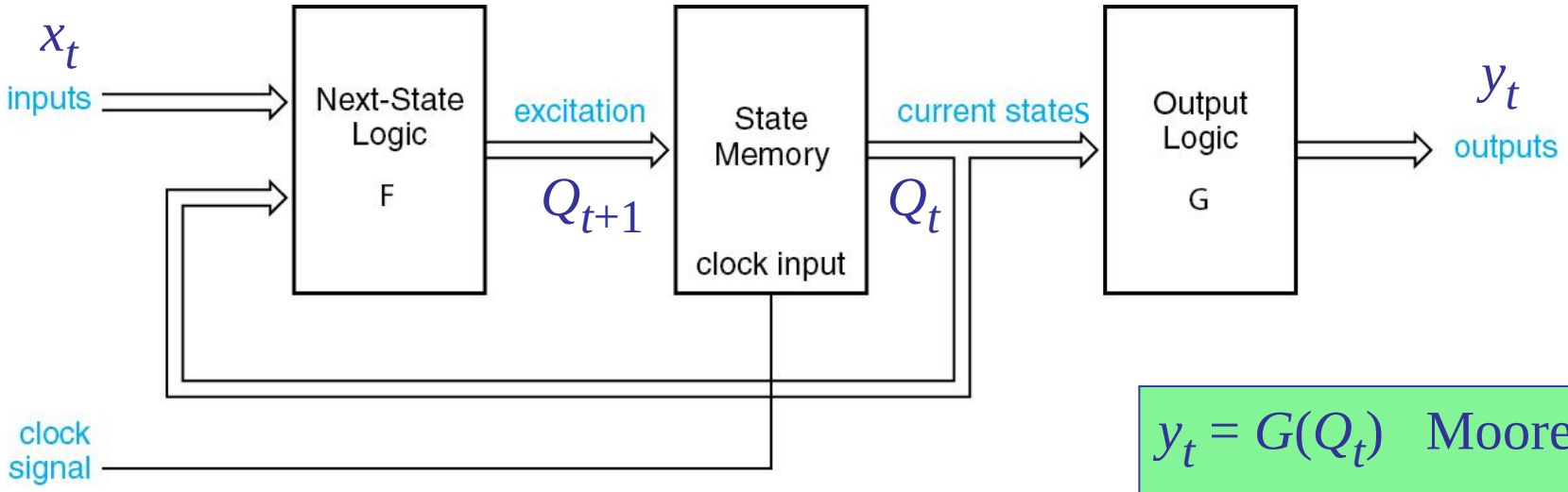
$$y = G(x, Q)$$
$$Q_{\text{next}} = F(x, Q)$$

changes occur at clock rising edges

in general, one can have multiple inputs, multiple outputs (MIMO systems), and multiple states.

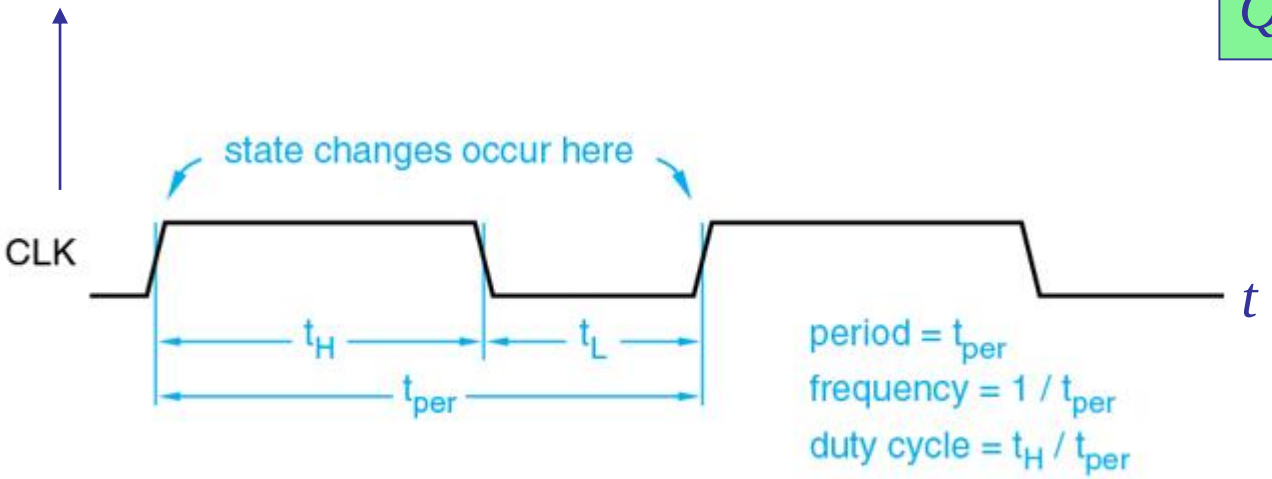
State Machines – Moore

e.g., edge-triggered D-flip-flops



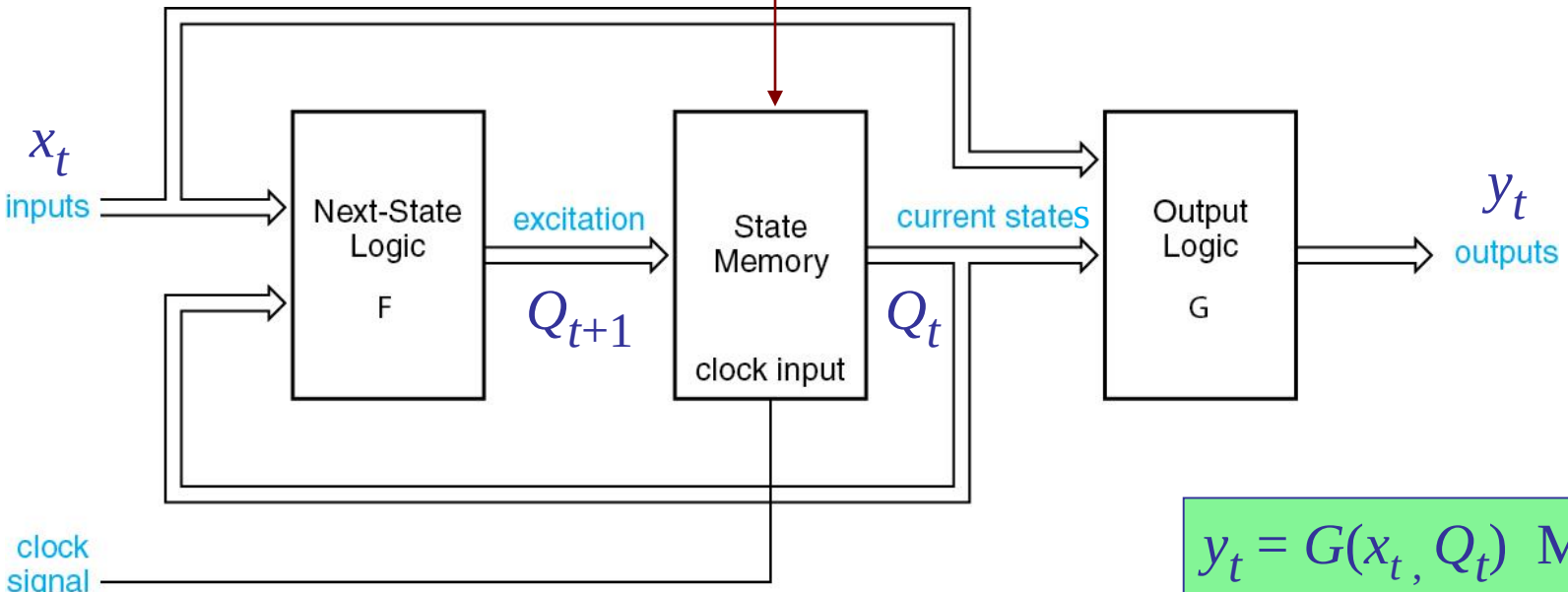
$y_t = G(Q_t)$ Moore

$Q_{t+1} = F(x_t, Q_t)$



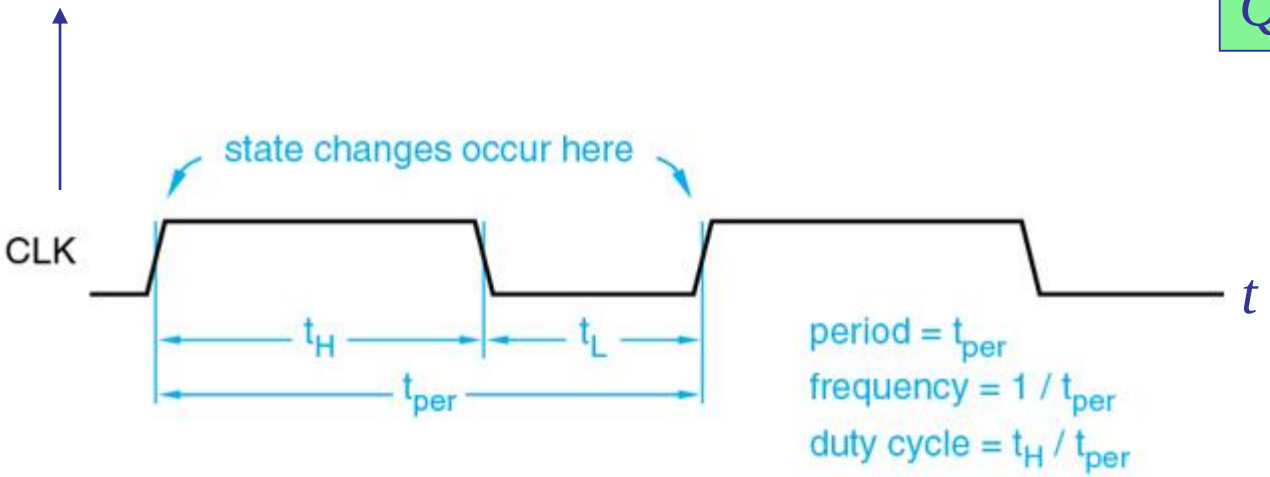
State Machines – Mealy

e.g., edge-triggered D-flip-flops

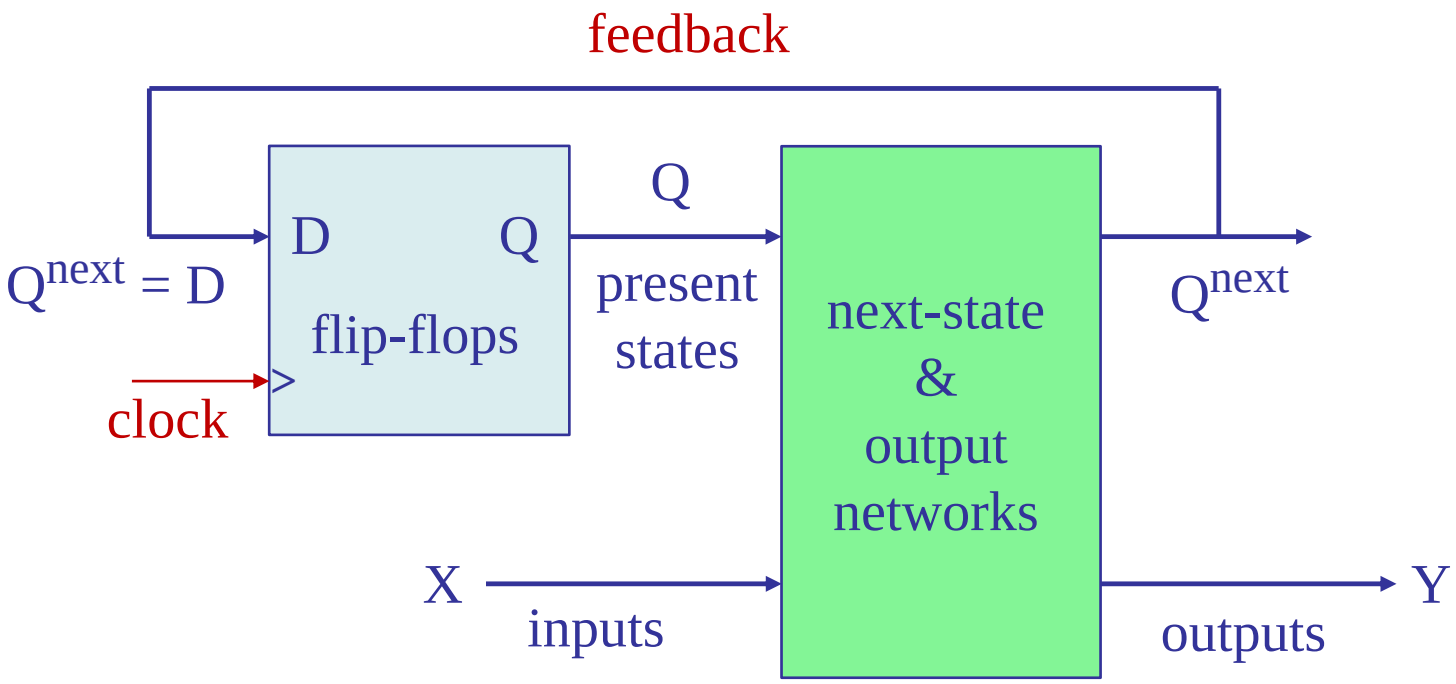


$y_t = G(x_t, Q_t)$ Mealy

$Q_{t+1} = F(x_t, Q_t)$

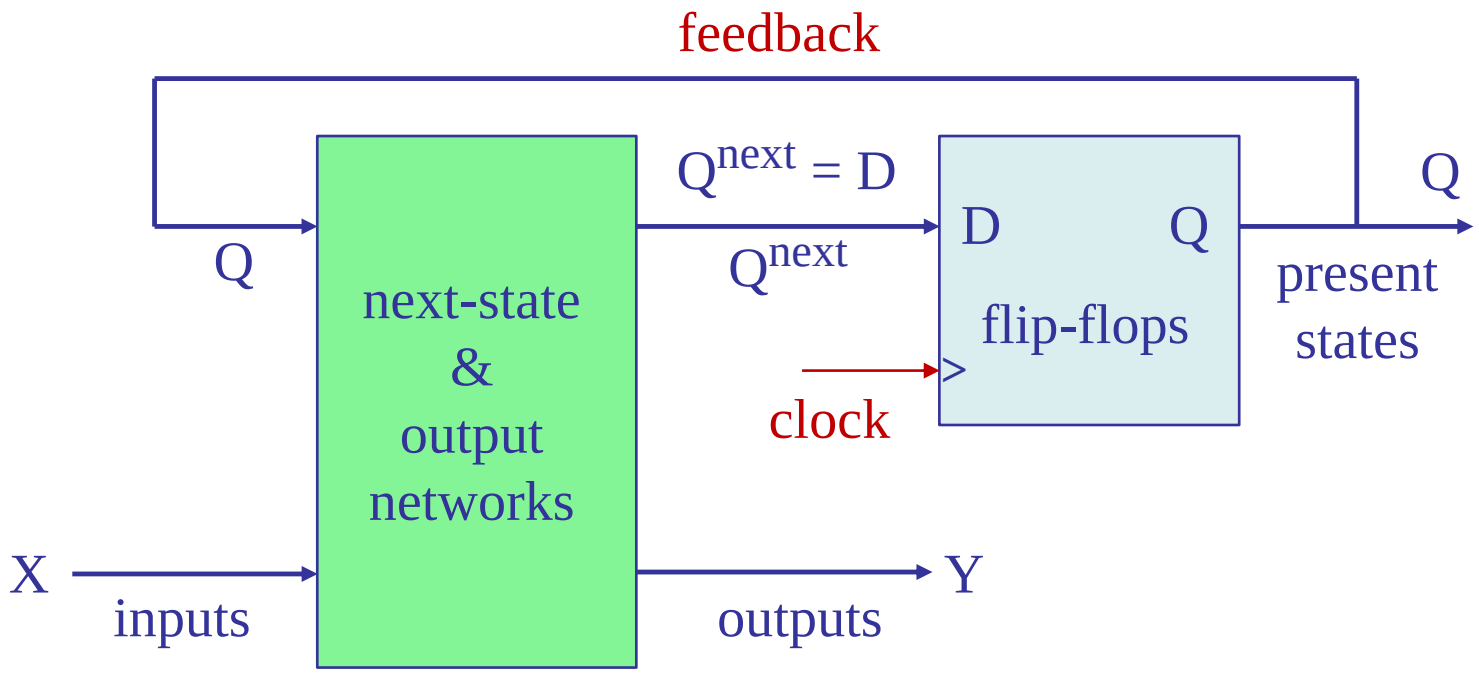


A more compact way of drawing an FSM is depicted below, assuming that D flip-flops are used for the memory/state-holding elements. The advantage is that the next states, Q^{next}, are the excitation inputs to the flip-flops, i.e., $D = Q^{next}$. See next page for an alternative drawing.

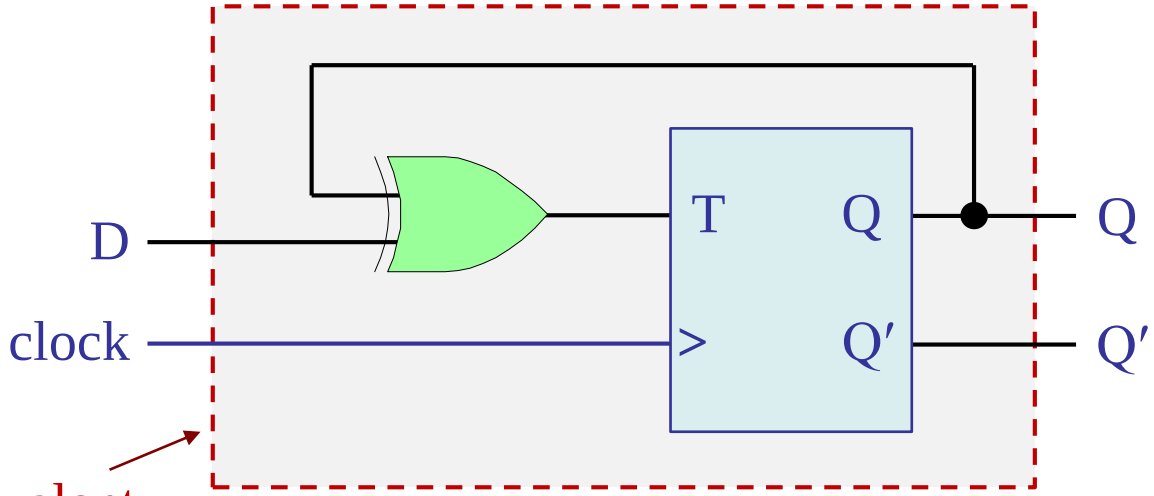


Alternative drawing

The next states, Q^{next}, are the excitation inputs to the flip-flops, i.e., $D = Q^{next}$. See next page on how to use other types of flip-flops.



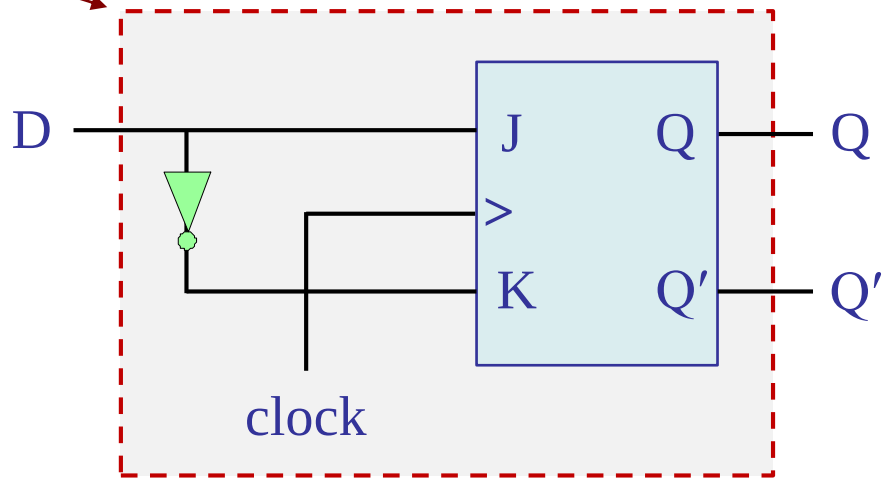
Using T or JK flip-flops instead of D flip-flops: (a) carry out the design using D flip-flops, and, (b) replace each D flip-flop by a T or a JK flip-flop as shown below (effectively converting them to D flip-flops).



$T = D \oplus Q$

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

equivalent D flip-flops



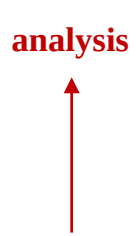
$J = D$

$K = D'$

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

FSM design steps with D-flip-flops:

1. Start with verbal description of the system, specifying its inputs, outputs, and the number of required states, including how the states transition from one to another during each clock edge time instant. You may choose one of the states as the starting or reset state.
2. Convert the verbal description into a state diagram.
3. Convert the state diagram into a state table, that is, a truth table that includes the inputs, the present states, the next states, and the outputs.
4. Encode the symbolic states in terms of state variables, by making a particular state-assignment or state-encoding choice (i.e., binary or Gray coding) and re-write the state table in terms of the encoded state variables.
5. Based on the state table, develop expressions for the next-state equations, $Q^{next} = F(X, Q)$, in terms of the present states Q and the inputs X, and also for output equations, $Y = G(X,Q)$.
6. Implement step-5 as a combinational circuit with logic gates.
7. Feed back the next states Q^{next} to the D inputs of the D-flip-flops.
8. Connect a common clock signal to all the flip-flops, and also set the flip-flop "clear" inputs so that the system starts from the reset state.



Conversely, the FSM analysis problem starts with a given block-diagram realization for the FSM, and proceeds backwards through the above steps, deriving the state table, and clarifying the operation of the FSM.

FSM synthesis/design steps – Summary

1. Start with verbal description and choose states and the reset state.
2. Convert the verbal description into a state diagram.
3. Convert the state diagram into a state table.
4. Encode the symbolic states with D-flip-flop state variables.
5. Derive the next-state and output equations in terms of state variables.
6. Implement the design using D-flip-flops and logic gates.
7. Simulate it with MATLAB/Simulink to verify proper operation.

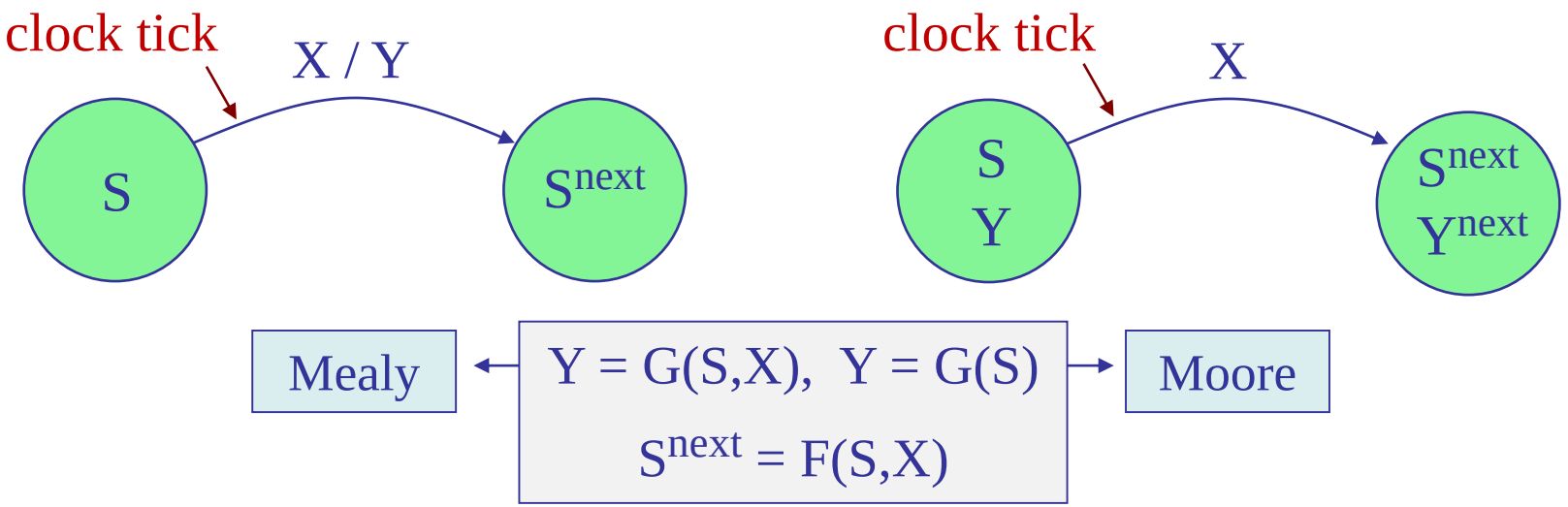
analysis ↑

synthesis ↓

state diagrams 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.

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. Moreover, 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. But, 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.



Example 1 – To clarify the above steps, we discuss the design of a Moore FSM for a car alarm system [cf. Hwang]. The alarm has an arming signal A, and it monitors several trigger inputs T, such as opening a door, breaking a glass, car vibration, opening the trunk, and so on. If the alarm is armed (A=1), then, if one or more of the trigger inputs is set, the alarm siren will go on.

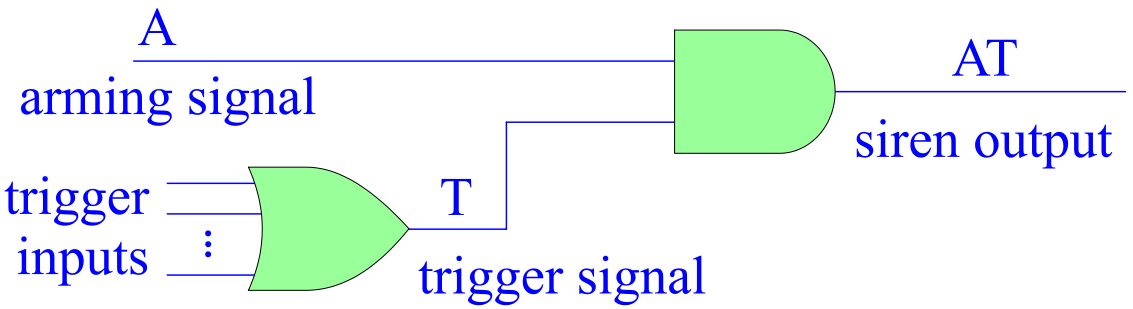
From this description we find that the siren signal will be,

$$\text{siren} = A \cdot T$$

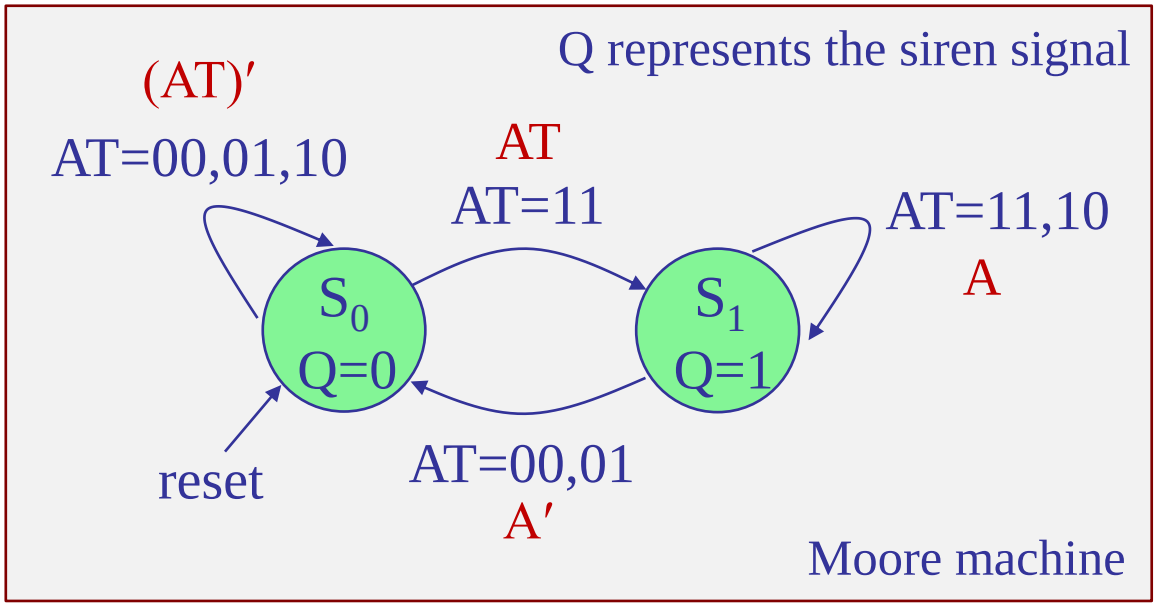
where the overall trigger signal T is obtained from the OR-ing,

$$T = \text{door} + \text{glass} + \text{vibration} + \text{trunk} + \cdots$$

A realization is shown below. However, simple as this is, it has a major limitation: if the siren is on (A=1 and T=1), then it will go off as soon as T=0 even though A=1 (e.g., someone may break into the car and close the door and the siren will stop). This can be fixed by introducing memory into the system.

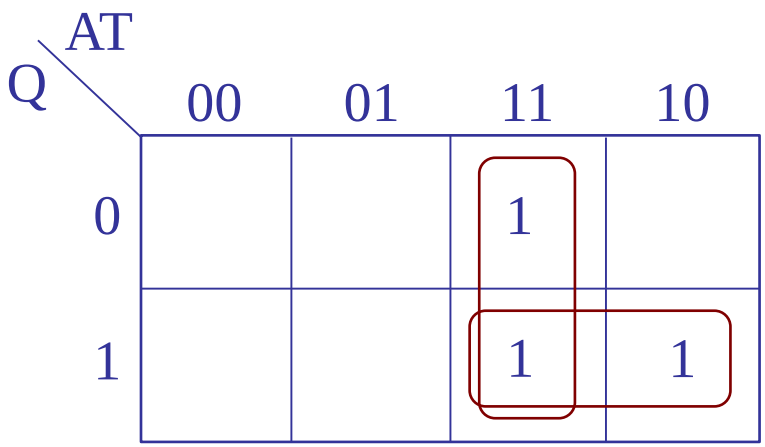


state diagram with state assignments Q = 0,1



characteristic table

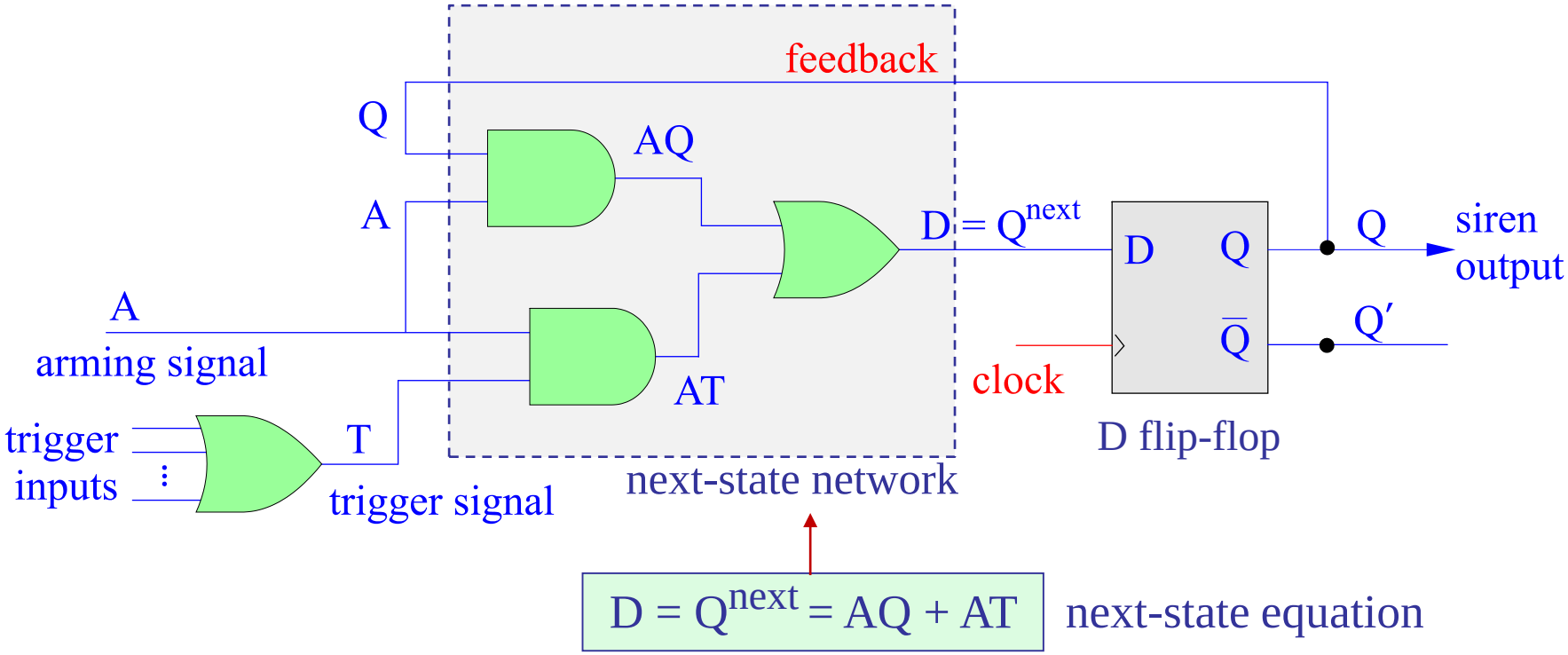
A	T	Q	Q^{next}	AT
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1



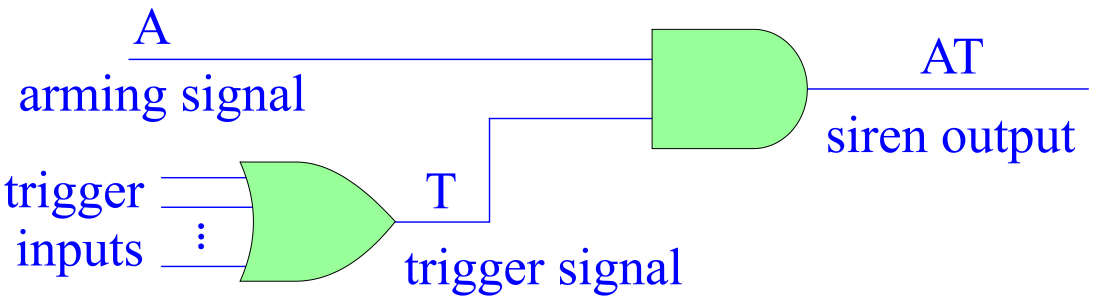
next-state equation

$$Q^{next} = AQ + AT$$

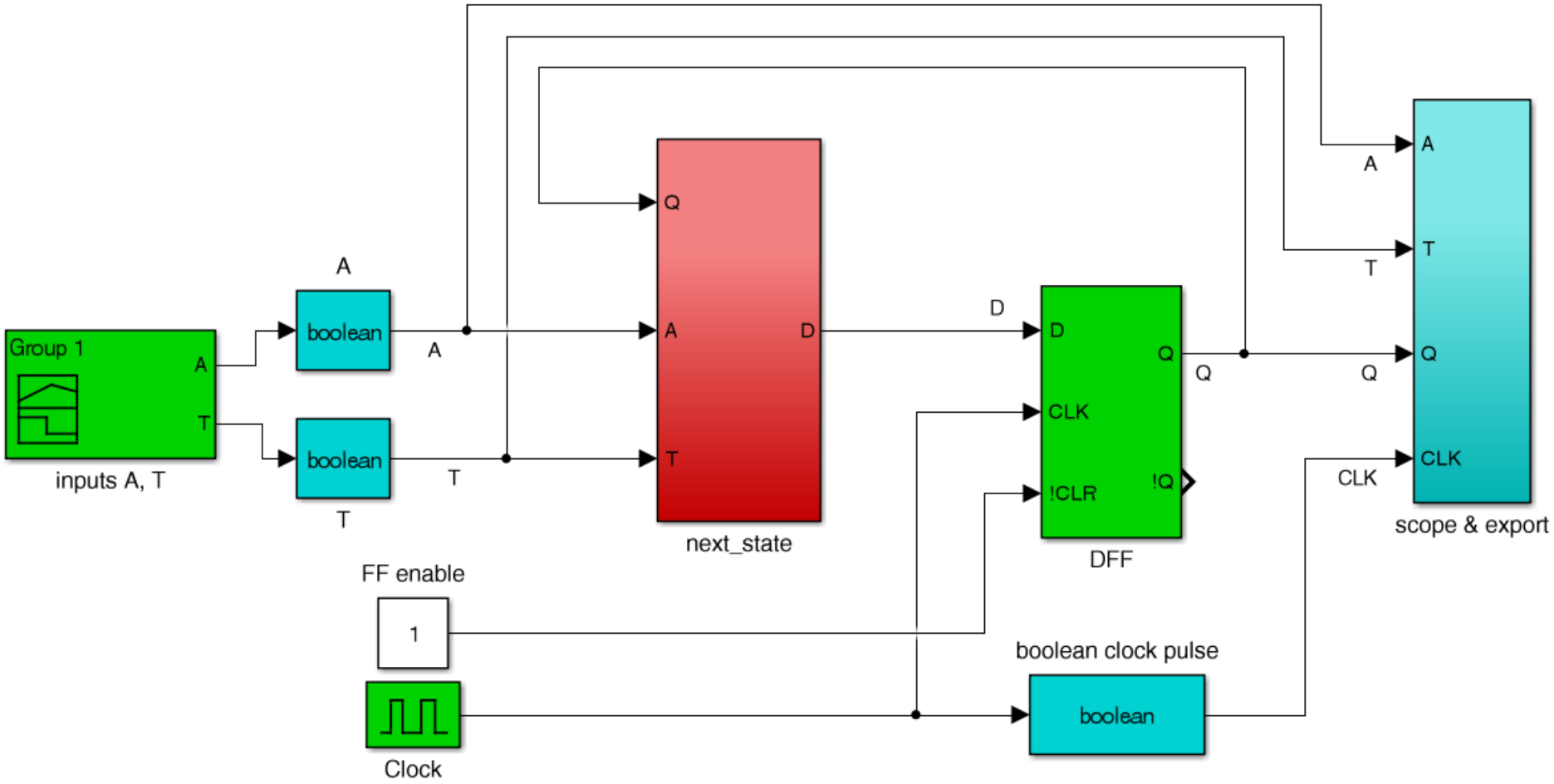
realization using D flip-flops →



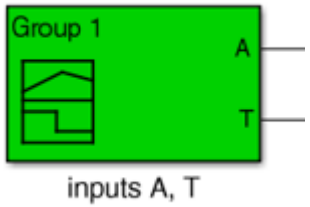
$D = Q^{next} = AQ + AT$ next-state equation



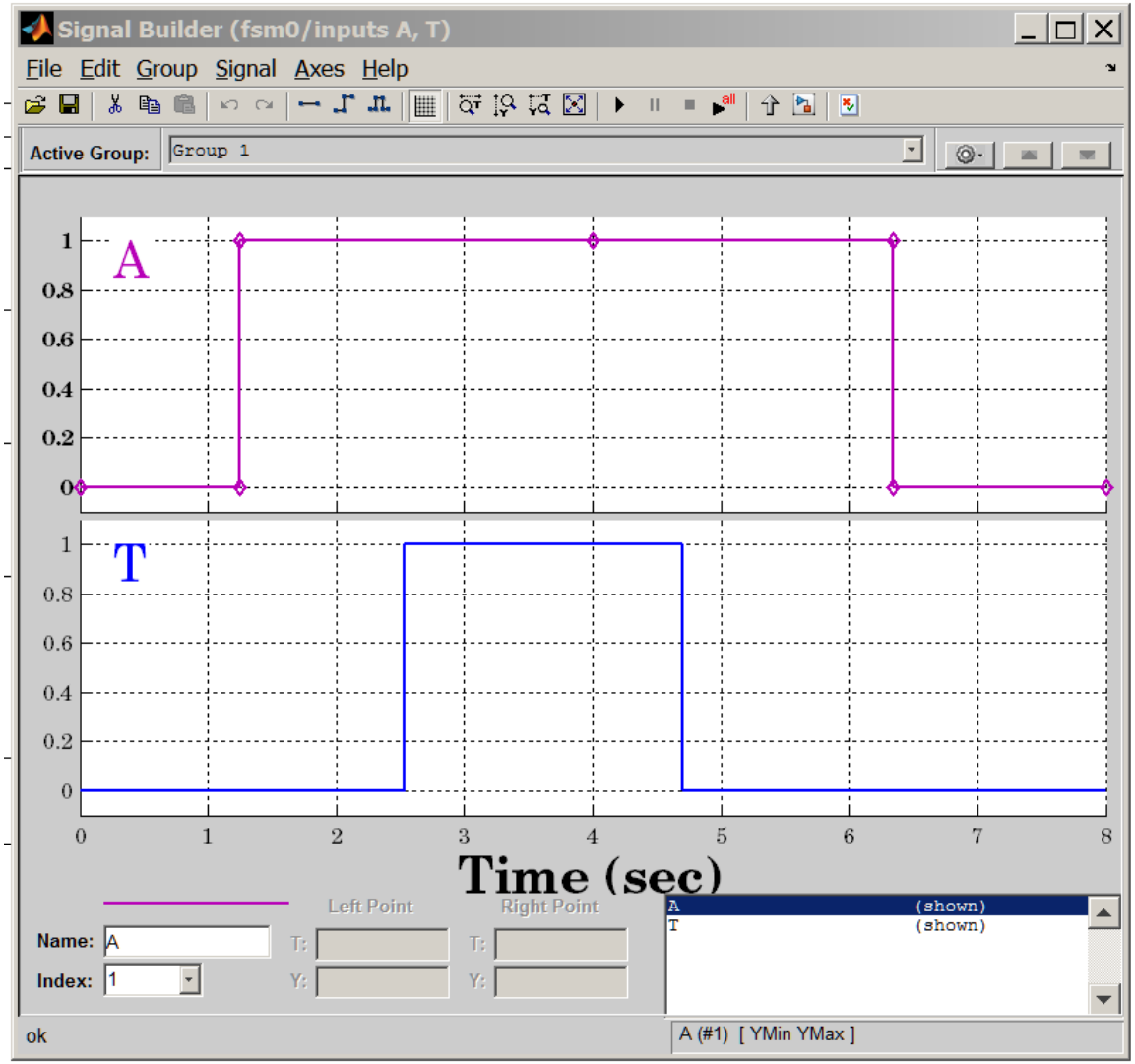
original realization with no memory

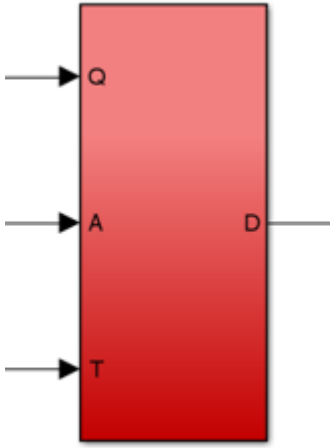


fsm0s.slx on Canvas

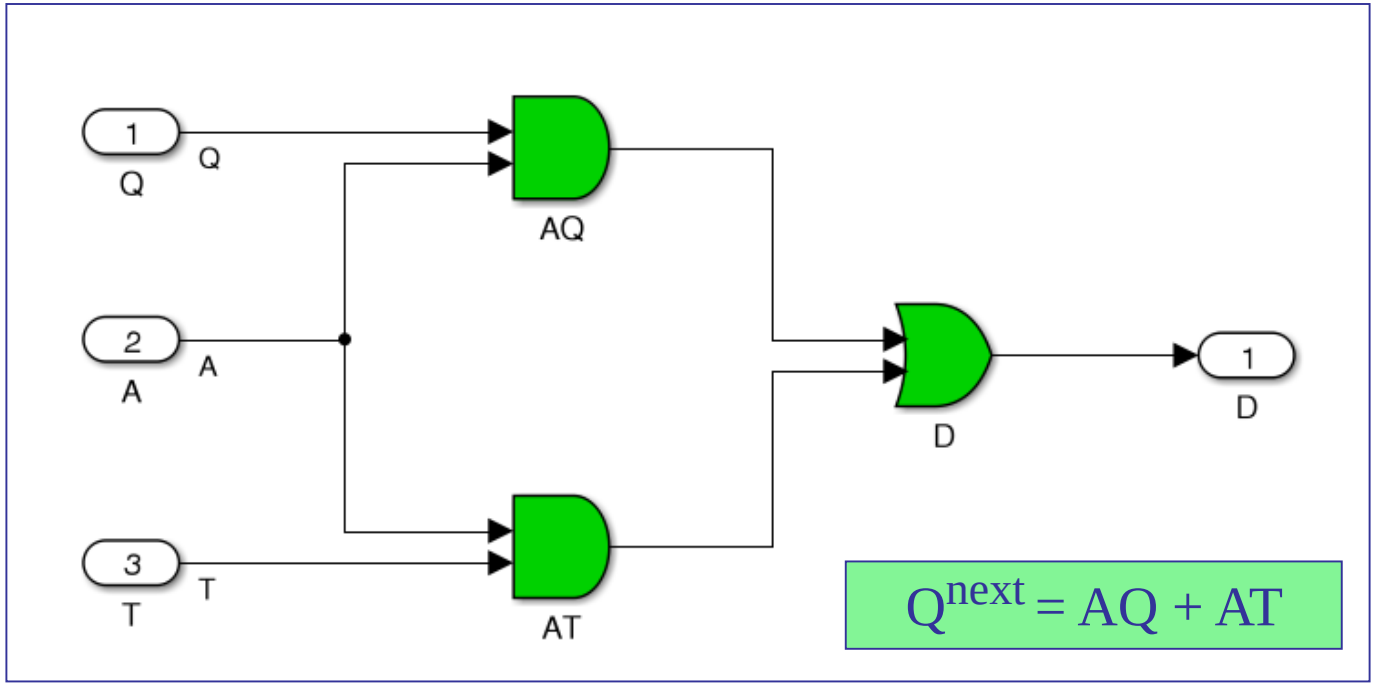


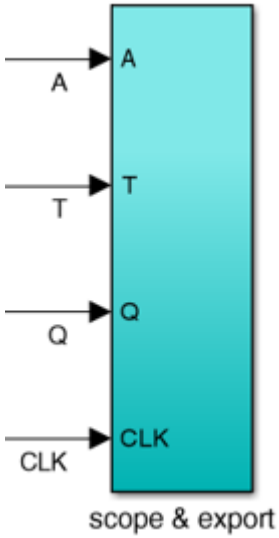
input signal generator



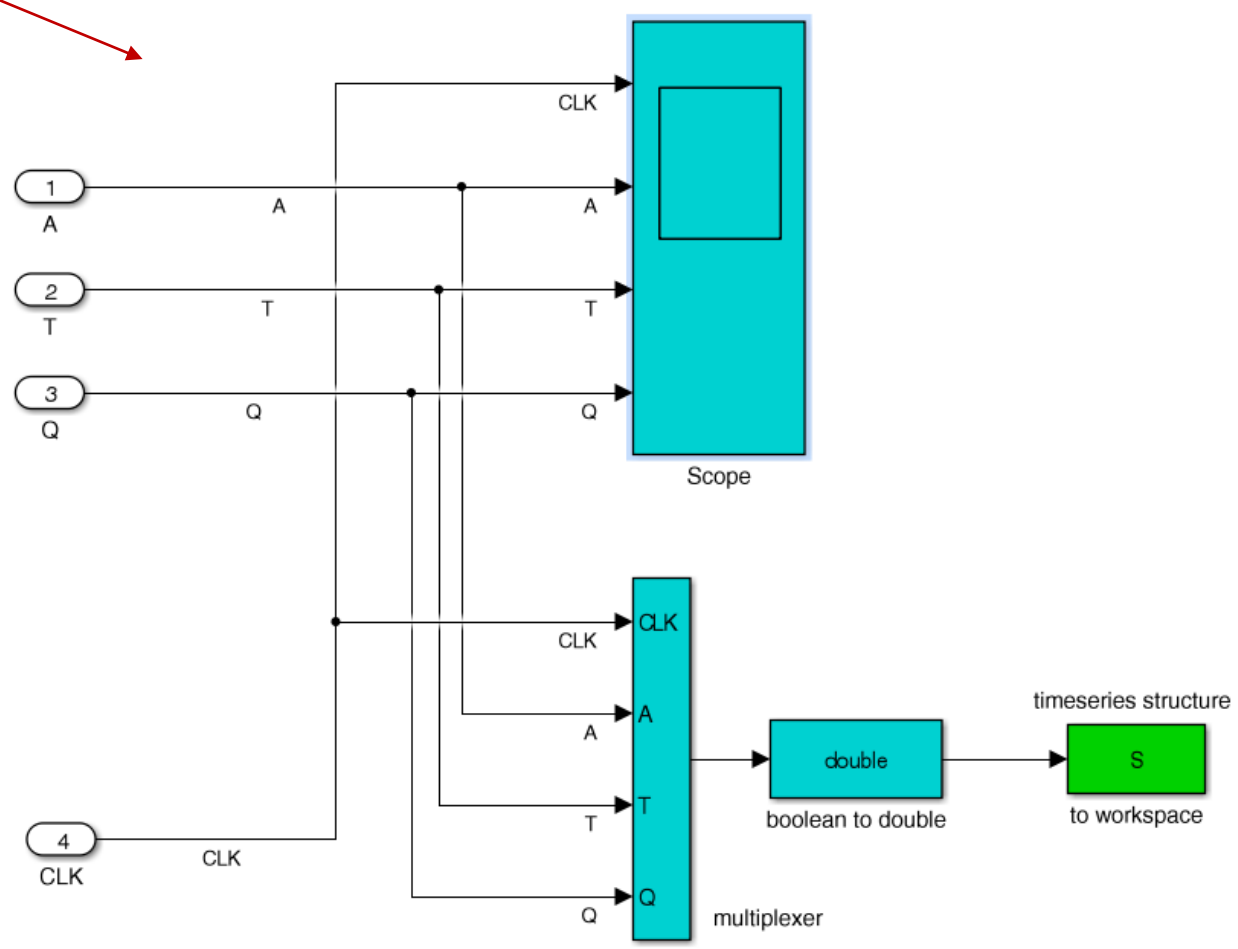


next-state network

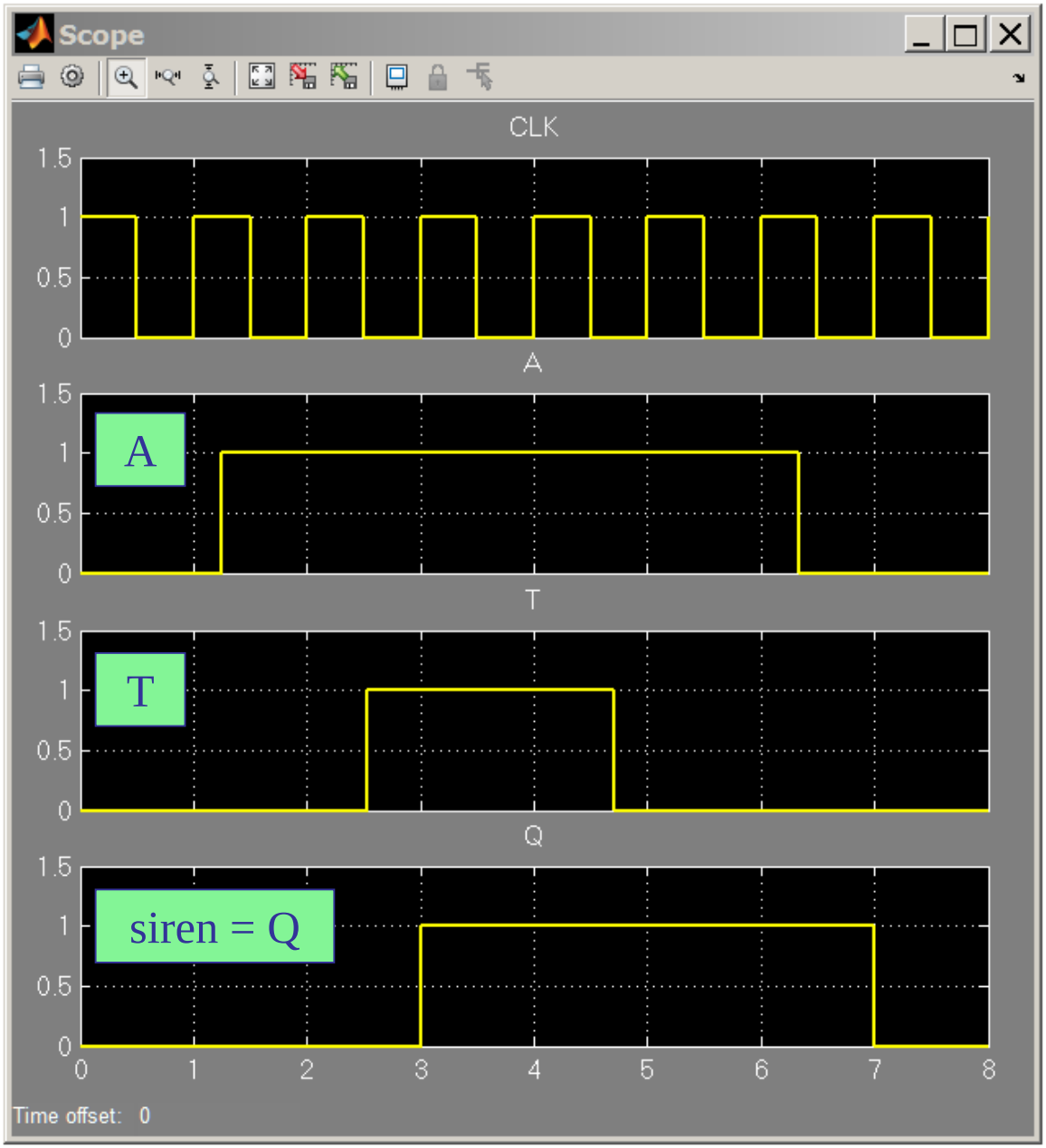




scope & data export subfunction

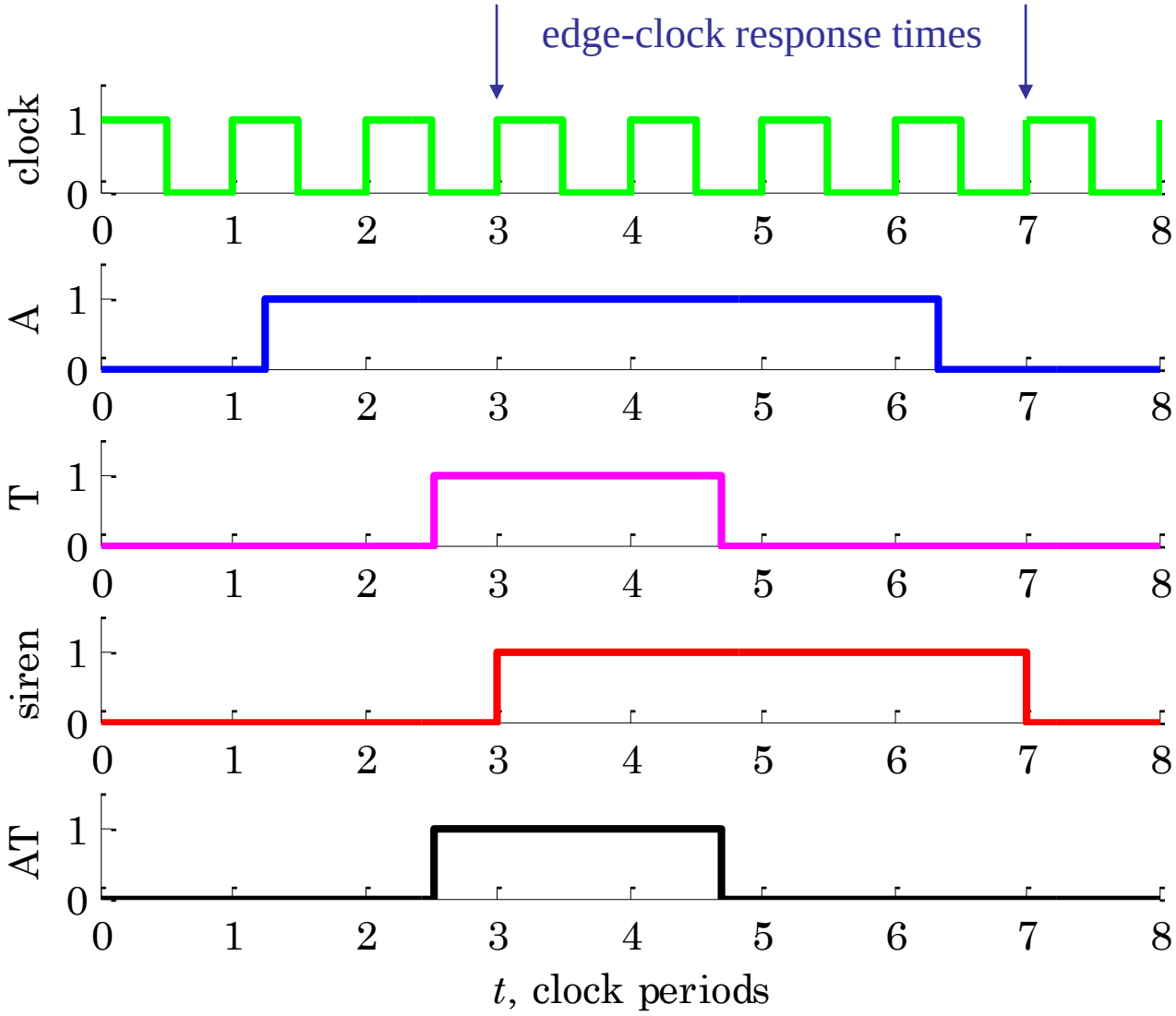


Example 1



scope output

timing diagram



MATLAB code →

siren output with memory

siren output without memory

fsm0m.m on Canvas

```
% fsm0m.m on Canvas - MATLAB code for generating timing diagram
% data imported from Simulink structure S from file fsm0s.slx

t   = S.time;           % time
CLK = S.data(:,1);      % clock pulses
A   = S.data(:,2);      % arming signal
T   = S.data(:,3);      % trigger signal
Q   = S.data(:,4);      % siren signal with memory
AT  = A & T;            % siren signal without memory

figure;
subplot(5,1,1); stairs(t,CLK, 'g-', 'linewidth',2);
       xaxis(0,8,0:8); yaxis(0,1.5,0:1); ylabel('clock');
subplot(5,1,2); stairs(t,A, 'b-', 'linewidth',2);
       xaxis(0,8,0:8); yaxis(0,1.5,0:1); ylabel('A');
subplot(5,1,3); stairs(t,T, 'm-', 'linewidth',2);
       xaxis(0,8,0:8); yaxis(0,1.5,0:1); ylabel('T');
subplot(5,1,4); stairs(t,Q, 'r-', 'linewidth',2);
       xaxis(0,8,0:8); yaxis(0,1.5,0:1); ylabel('siren');
subplot(5,1,5); stairs(t,AT, 'k-', 'linewidth',2);
       xaxis(0,8,0:8); yaxis(0,1.5,0:1); ylabel('AT');
xlabel('{\itt}, clock periods')
```

Example 2 – To clarify the above steps, we discuss the design of an FSM for a 2-bit binary counter that has an enable input, and an output.

This differs from a plain counter that we considered in unit-7 in that it is a sequential circuit with an additional input and an output.

This is the same example as that of Fig. 9-8 in Wakerly, but instead of starting with the block diagram, we will start with the verbal description and work our way through the design steps, eventually ending up with a block-diagram realization.

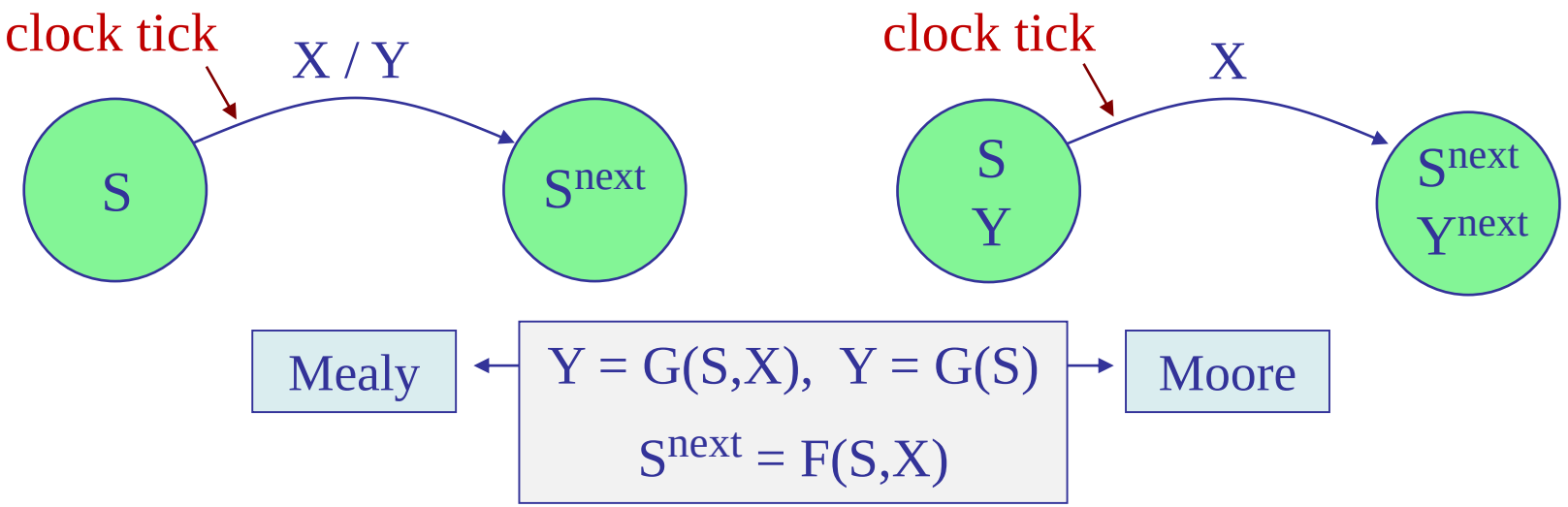
Step 1: Verbal description. Design a counter that counts through the sequence, 0, 1, 2, 3, of decimal numbers. The counter must have an enable input E, such that when E=1, the counting proceeds normally and repeating mod-4, but, it stops when E=0. Moreover, there should be an output Y that indicates the completion of the counting sequence every four counts, that is, Y=1, when the count reaches 3, and Y=0, otherwise. We may think of the successive numbers, 0,1,2,3, as the states of the counter, and we will denote them in the abstract as, S_0, S_1, S_2, S_3.

The reset state will be count = 0, or, state, S_0.

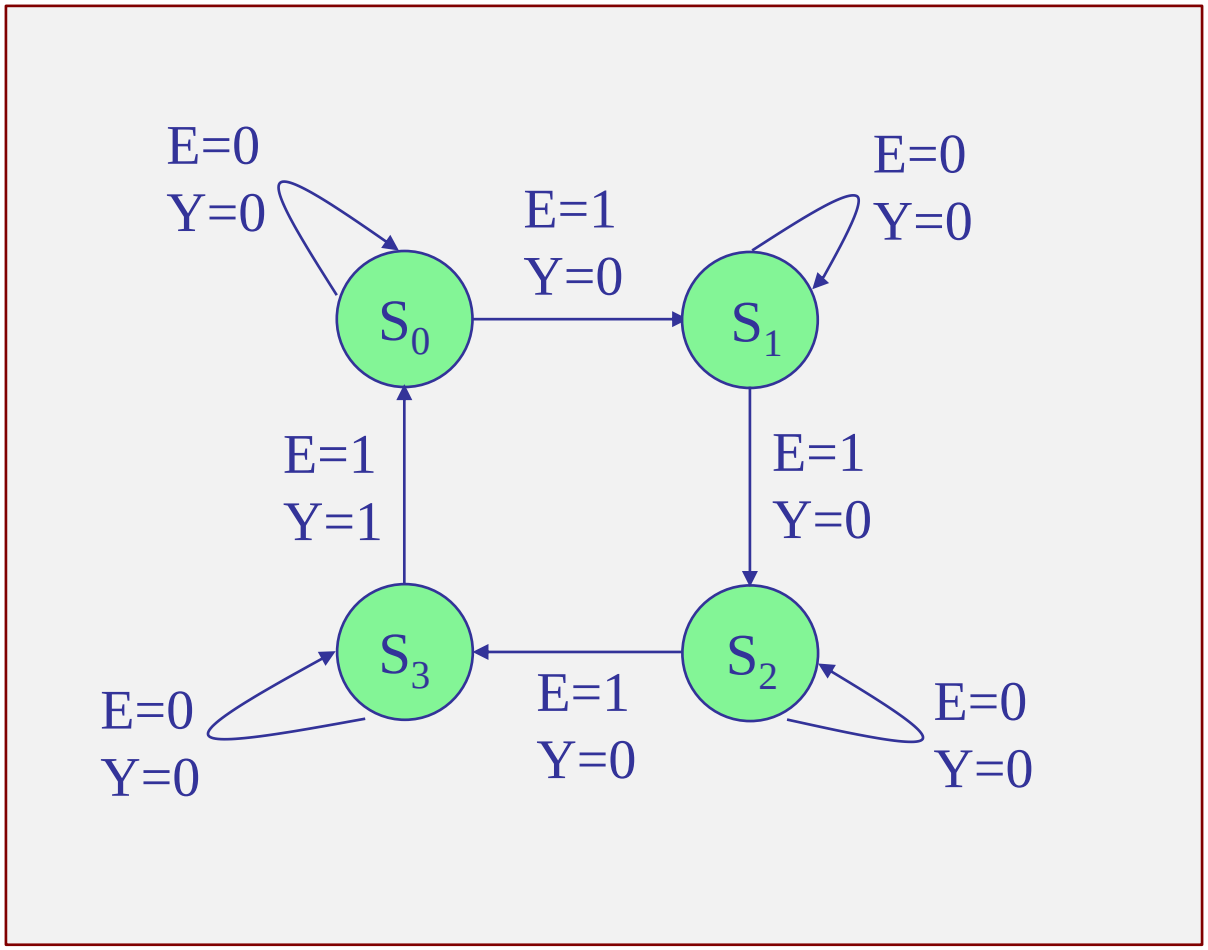
state diagrams are directed graphs that convey the same information as state tables, but in a graphical form, where the states are listed inside the circles and the arrows indicate the state transitions taking place at successive clock-edge time instants.

Conventions for drawing state diagrams:

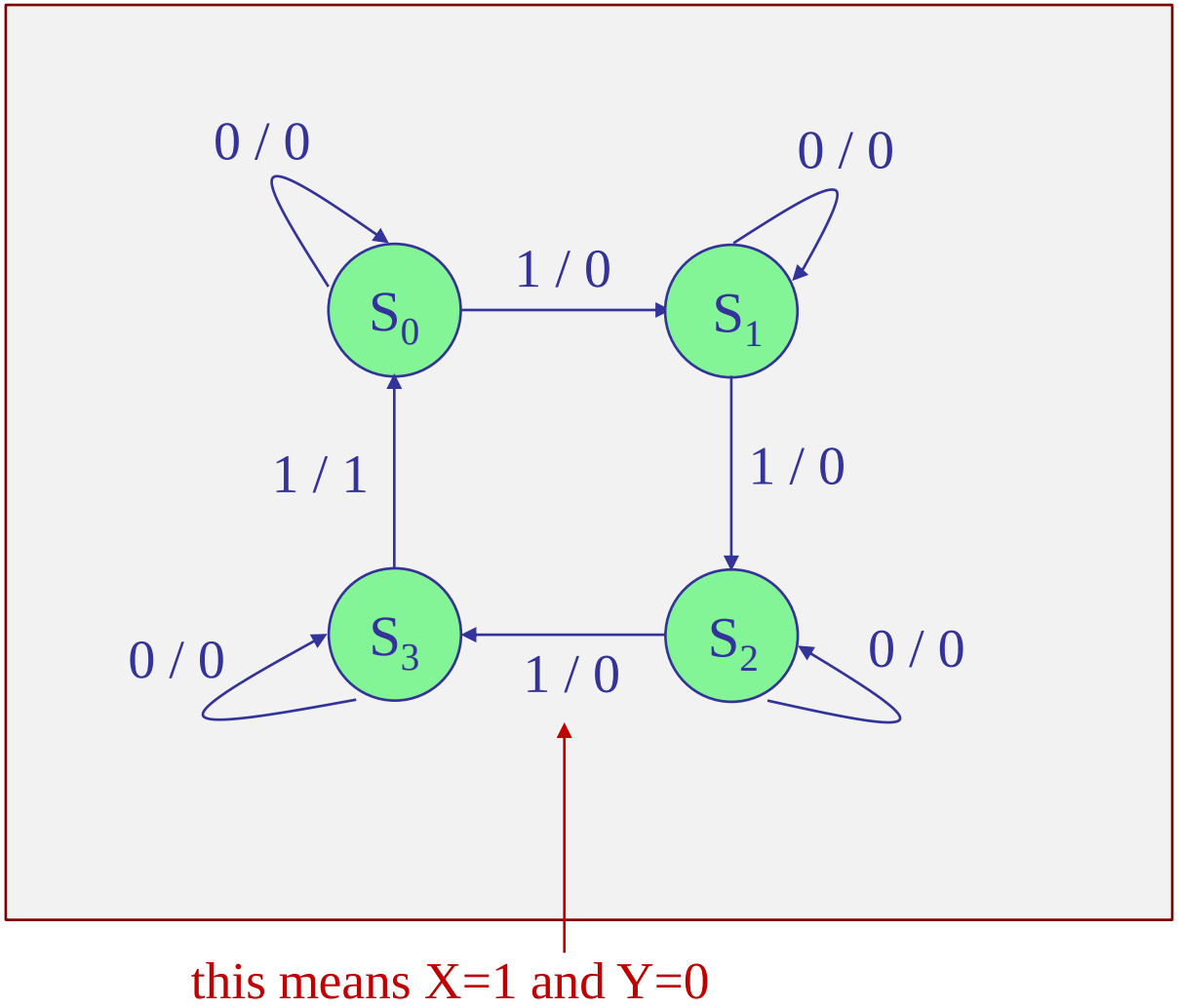
Along each arrow connecting two successive states S and S^{next}, indicate the values of the inputs X that caused this transition from the current state S. Moreover, for Mealy machines, indicate along the same arrow the values of the outputs Y that resulted from the values of X and the current state S. But, 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.



Step 2: State diagram. This is a Mealy FSM because Y depends on the input E, e.g., Y=0 when E=0, but Y=1 when E=1 at state S_3



Step 2: State diagram. Yet, another convention is to indicate the values of the input and output in the format X / Y, with the input written first and the output, second, along the transition arrows.



Step 3: State table. List all possible values of the input E and output Y, and all the required state transitions from present states to next states.

standard form

input E	present state	next state	output Y
0	S_0	S_0	0
0	S_1	S_1	0
0	S_2	S_2	0
0	S_3	S_3	0
1	S_0	S_1	0
1	S_1	S_2	0
1	S_2	S_3	0
1	S_3	S_0	1

← count completed then, repeat

compact form

present state	next state		output Y	
	E=0	E=1	E=0	E=1
S_0	S_0	S_1	0	0
S_1	S_1	S_2	0	0
S_2	S_2	S_3	0	0
S_3	S_3	S_0	0	1

Step 4: State assignment. In general, for an FSM with N states, one needs

$$n = \text{ceiling}(\log_2 N)$$

state variables, each to be stored in a separate flip-flop. Thus, here, we need, $n = \log_2(4) = 2$, state variables and two flip-flops, and we may choose the state variables to be binary bits denoted by Q_1, Q_0. The state table is now re-written in terms of the state variables.

input E	present state Q_1 Q_0	next state Q_1 Q_0	output Y
0	0 0	0 0	0
0	0 1	0 1	0
0	1 0	1 0	0
0	1 1	1 1	0
1	0 0	0 1	0
1	0 1	1 0	0
1	1 0	1 1	0
1	1 1	0 0	1

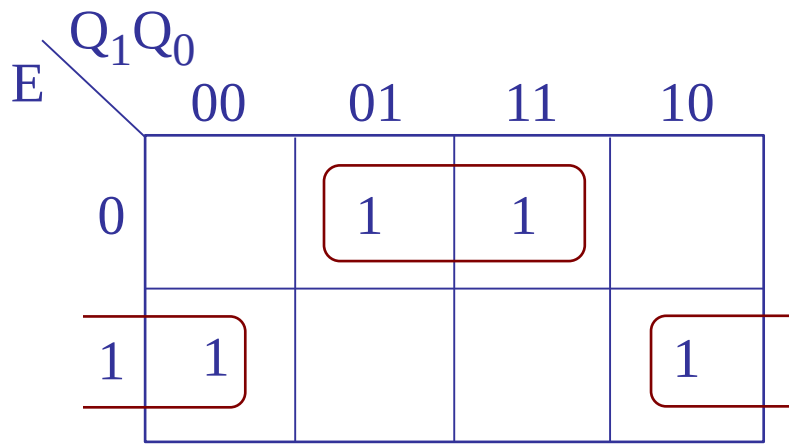
state encoding

	Q_1	Q_0
$S_0 \equiv$	0	0
$S_1 \equiv$	0	1
$S_2 \equiv$	1	0
$S_3 \equiv$	1	1

← count complete / repeat

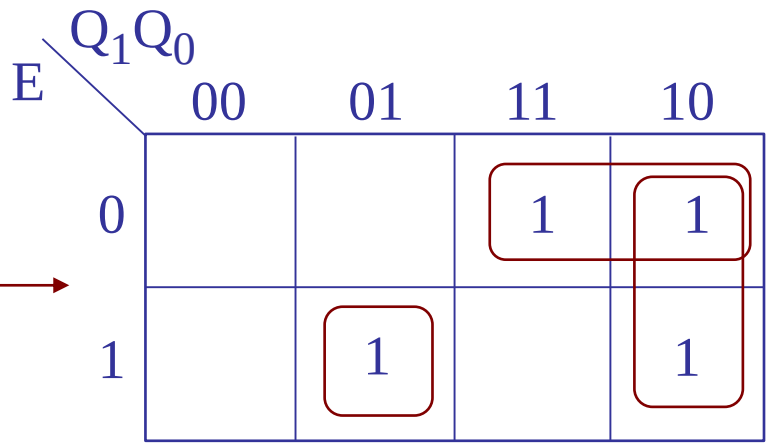
Step 5: Next-state and output equations. Use K-maps.

input E	present Q_1 Q_0	next Q_1 Q_0	output Y
0	0 0	0 0	0
0	0 1	0 1	0
0	1 0	1 0	0
0	1 1	1 1	0
1	0 0	0 1	0
1	0 1	1 0	0
1	1 0	1 1	0
1	1 1	0 0	1



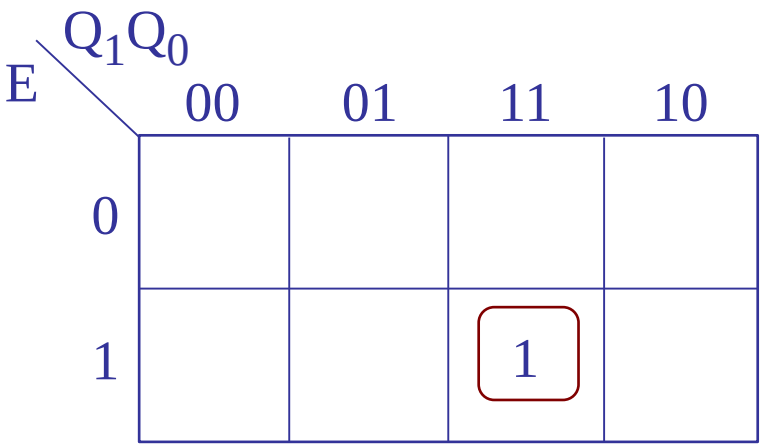
$$Q_0^{next} = E' Q_0 + E Q_0' = E \oplus Q_0$$

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



Step 5: Next-state and output equations. Use K-maps.

input E	present Q_1 Q_0	next Q_1 Q_0	output Y
0	0 0	0 0	0
0	0 1	0 1	0
0	1 0	1 0	0
0	1 1	1 1	0
1	0 0	0 1	0
1	0 1	1 0	0
1	1 0	1 1	0
1	1 1	0 0	1



$Y = E\,Q_1\,Q_0$

summary

equivalent form

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

$Q_1^{next} = Q_1 \oplus (EQ_0) = Q_1\,E' + (Q_1 \oplus Q_0)\;E$

Mealy FSM → $Y = E\,Q_1\,Q_0$

equivalent form

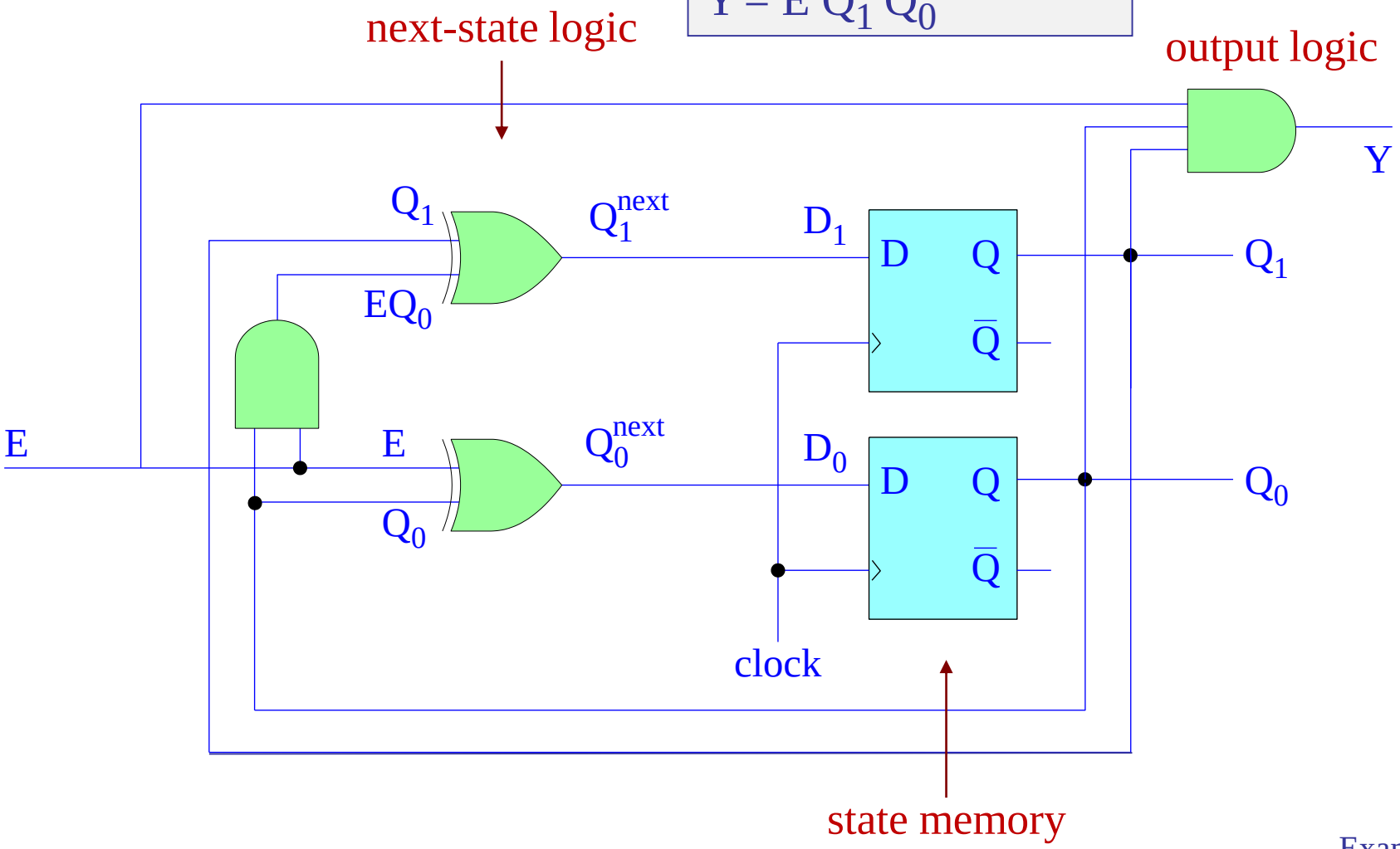
$$Q_1^{next} = Q_1 \oplus (EQ_0) = Q_1 E' + (Q_1 \oplus Q_0)\ E$$

proof:

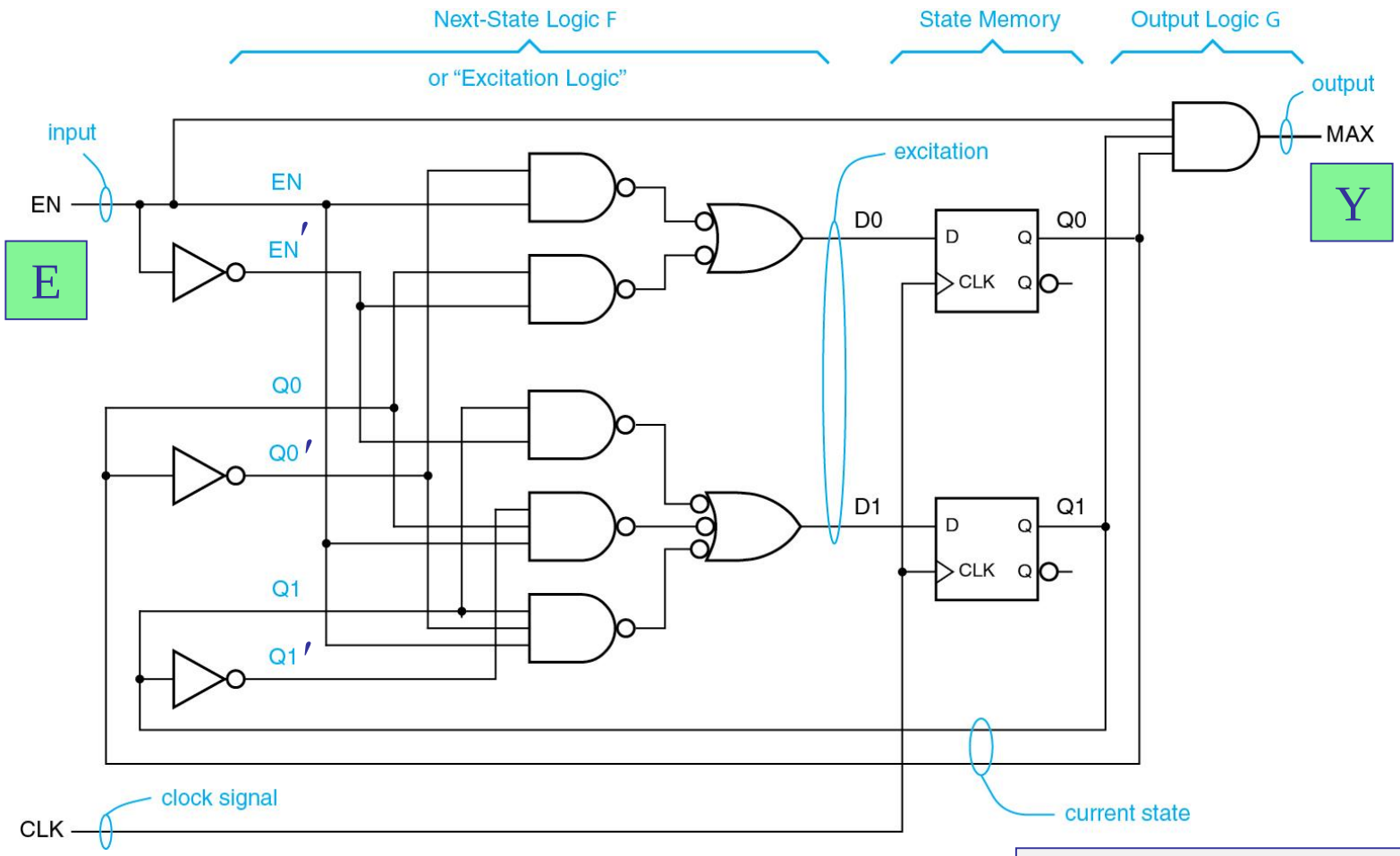
$$\begin{aligned}
Q_1^{next} &= Q_1 \oplus (EQ_0) = Q_1 (E\ Q_0)' + Q_1' (Q_0\ E) \\
&= Q_1 (E' + Q_0') + Q_1' Q_0 E \\
&= Q_1 E' + Q_1 Q_0' + Q_1' Q_0 E \\
&= Q_1 E' + Q_1 Q_0' (E + E') + Q_1' Q_0 E \\
&= (Q_1 + Q_1 Q_0') E' + (Q_1 Q_0' + Q_1' Q_0)E \\
&= Q_1 (1 + Q_0') E' + (Q_1 \oplus Q_0) E \\
&= Q_1 E' + (Q_1 \oplus Q_0)\ E
\end{aligned}$$

Steps 6-8: FSM block diagram. Use two flip flops for the state variables Q_1, Q_0

$$Q_0^{next} = E \oplus Q_0$$
$$Q_1^{next} = Q_1 \oplus (EQ_0)$$
$$Y = E\,Q_1\,Q_0$$



Equivalent realization from Wakerly, Fig. 9-8.

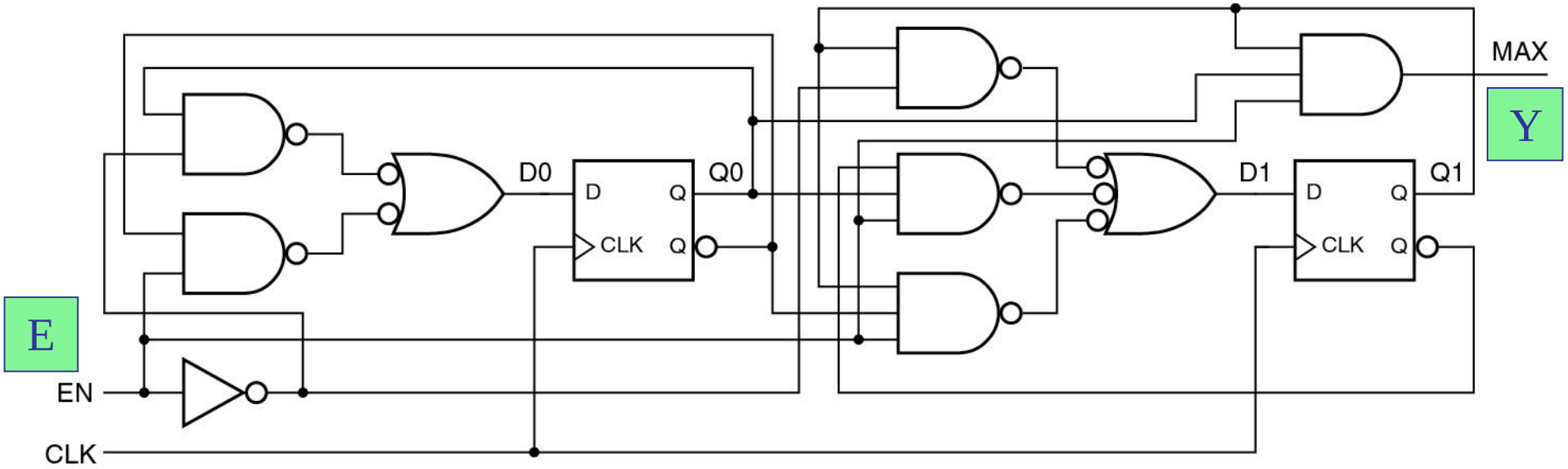


$$D_0 = E \oplus Q_0$$

$$D_1 = Q_1 E' + (Q_1 \oplus Q_0)\ E$$

$$Y = E\ Q_1\ Q_0$$

Another equivalent realization from Wakerly, Fig. 9-11.



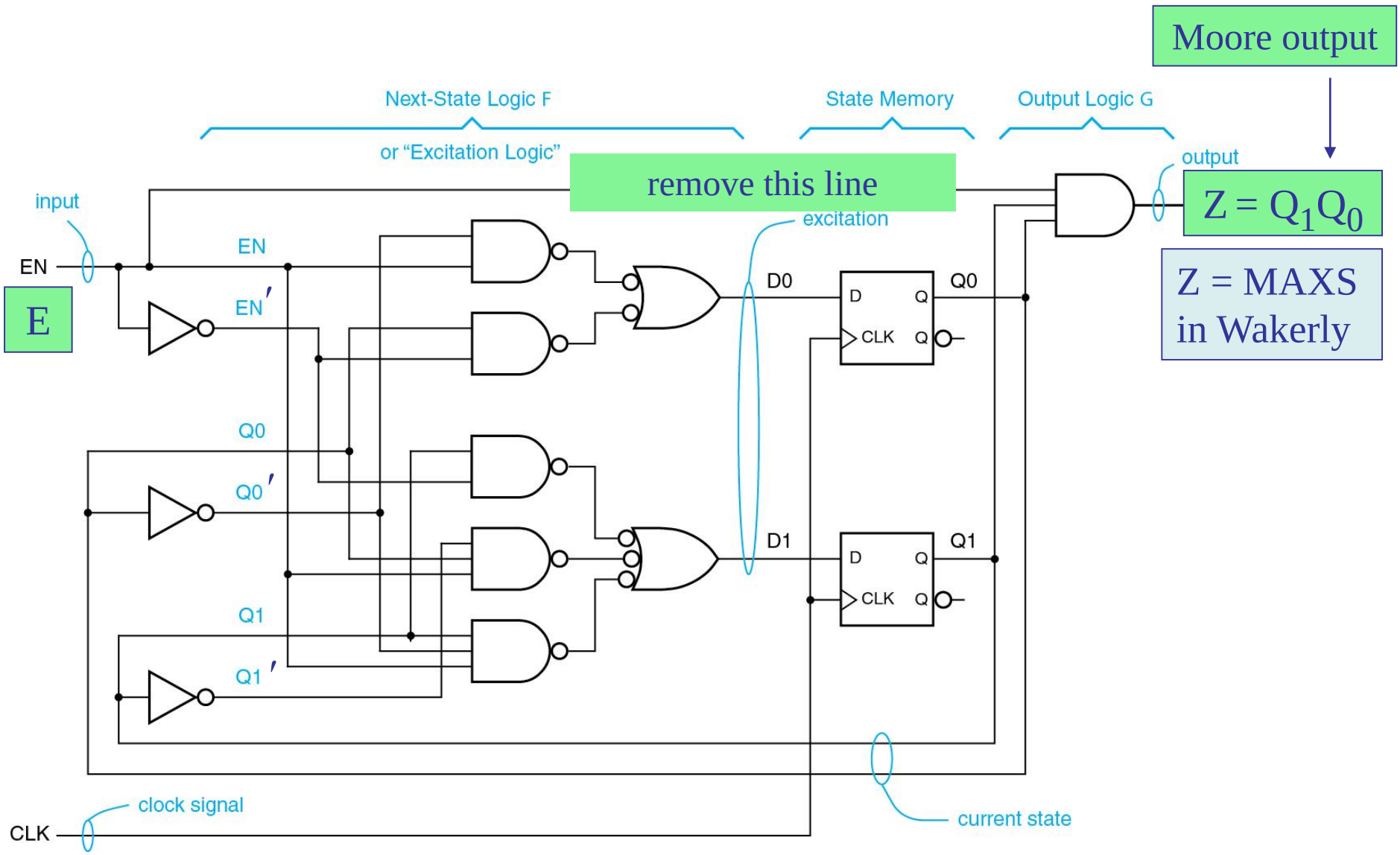
using also the Q′ outputs of the flip-flops

$D_0 = E \oplus Q_0 = E' Q_0 + E Q_0'$

$D_1 = Q_1 E' + (Q_1 \oplus Q_0)\ E = Q_1 E' + (Q_1 Q_0' + Q_1' Q_0)\ E$

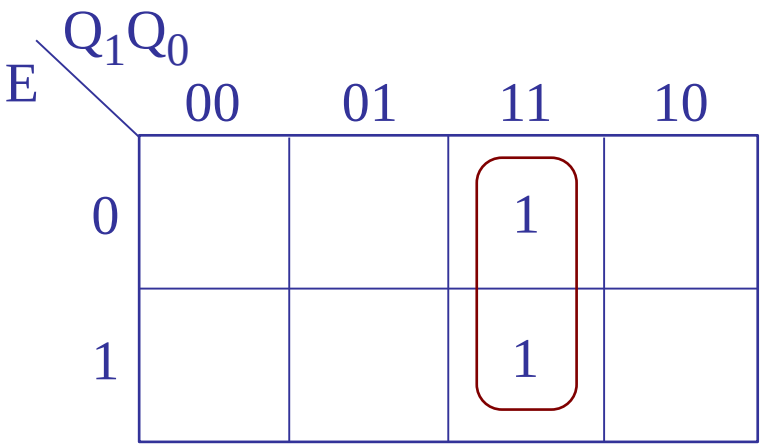
$Y = E Q_1 Q_0$

Example 2 - Moore FSM version of the 2-bit counter. Define an output, Z, that is independent of the input E and depends only on the current states.



Next-state and output equations for Moore version

input E	present Q_1 Q_0	next Q_1 Q_0	output Z
0	0 0	0 0	0
0	0 1	0 1	0
0	1 0	1 0	0
0	1 1	1 1	1
1	0 0	0 1	0
1	0 1	1 0	0
1	1 0	1 1	0
1	1 1	0 0	1



$Z = Q_1 Q_0$

Z is independent of E

summary

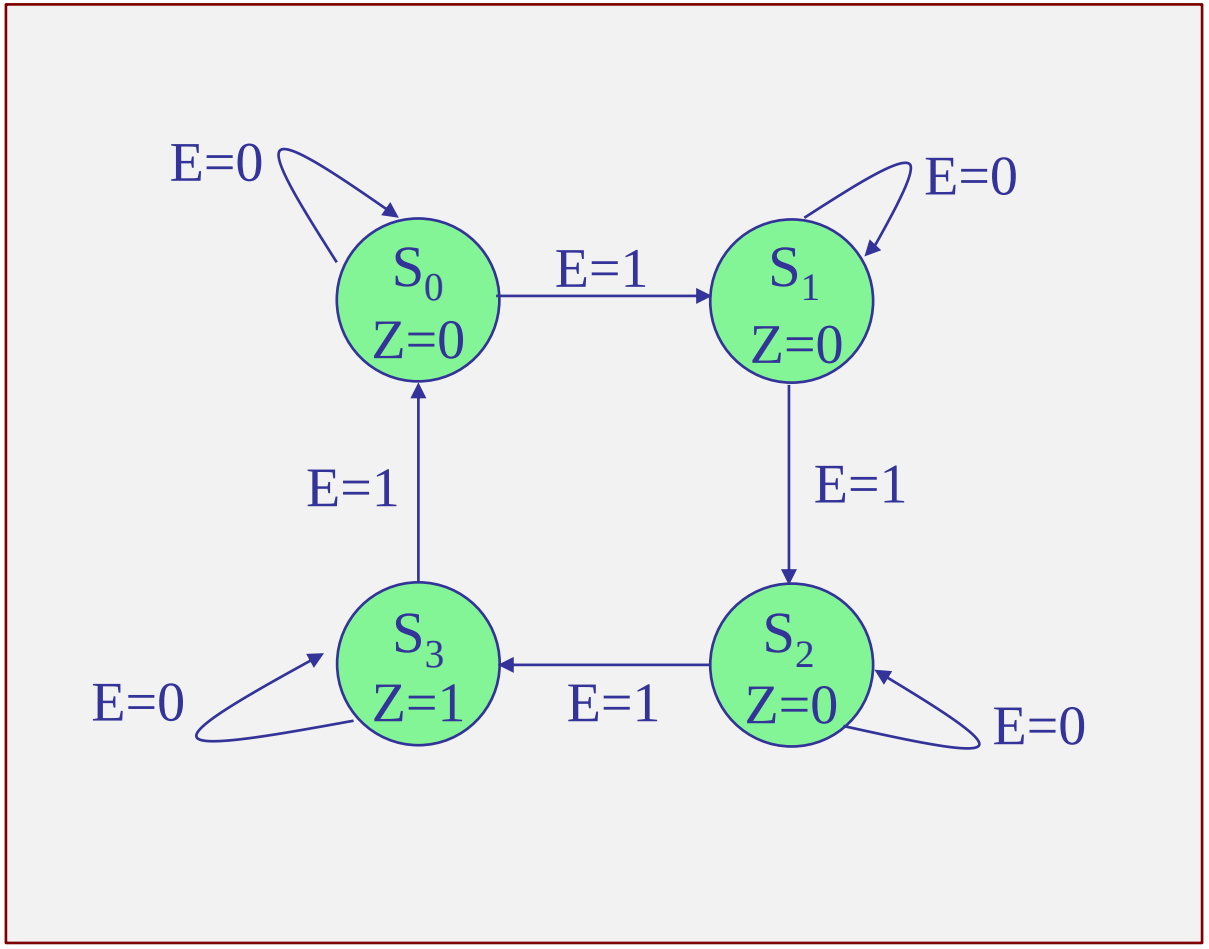
equivalent form

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

$Q_1^{next} = Q_1 \oplus (EQ_0) = Q_1 E' + (Q_1 \oplus Q_0)\ E$

Moore FSM → $Z = Q_1 Q_0$ (Moore output)

State diagram. This is a Moore FSM because Z depends only on the current states, Z is indicated inside each state circle, arrows are still labeled by the value of the input causing the transition.



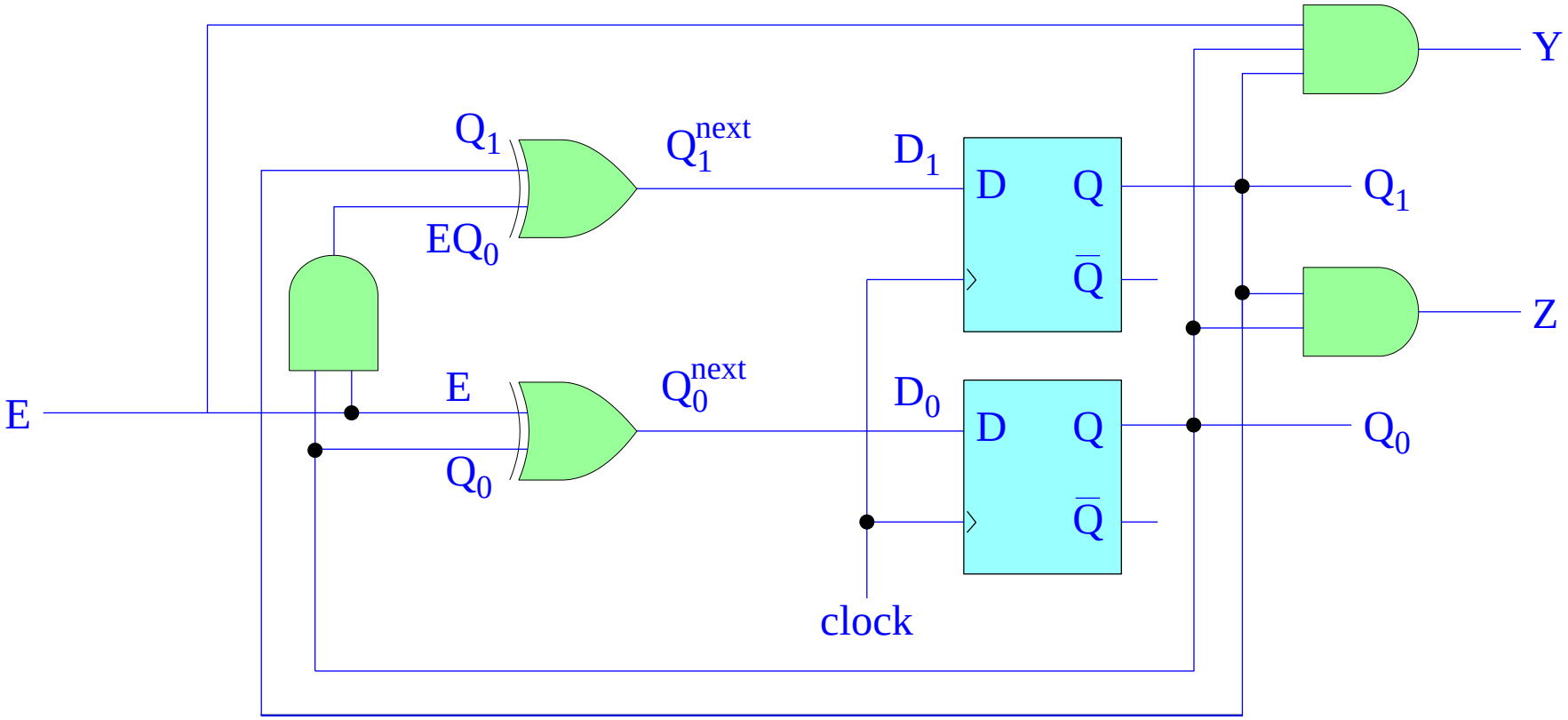
FSM block diagrams – Mealy and Moore versions

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

$Q_1^{next} = Q_1 \oplus (EQ_0)$

$Y = E\,Q_1\,Q_0$ (Mealy output)

$Z = Q_1\,Q_0$ (Moore output)



timing diagram

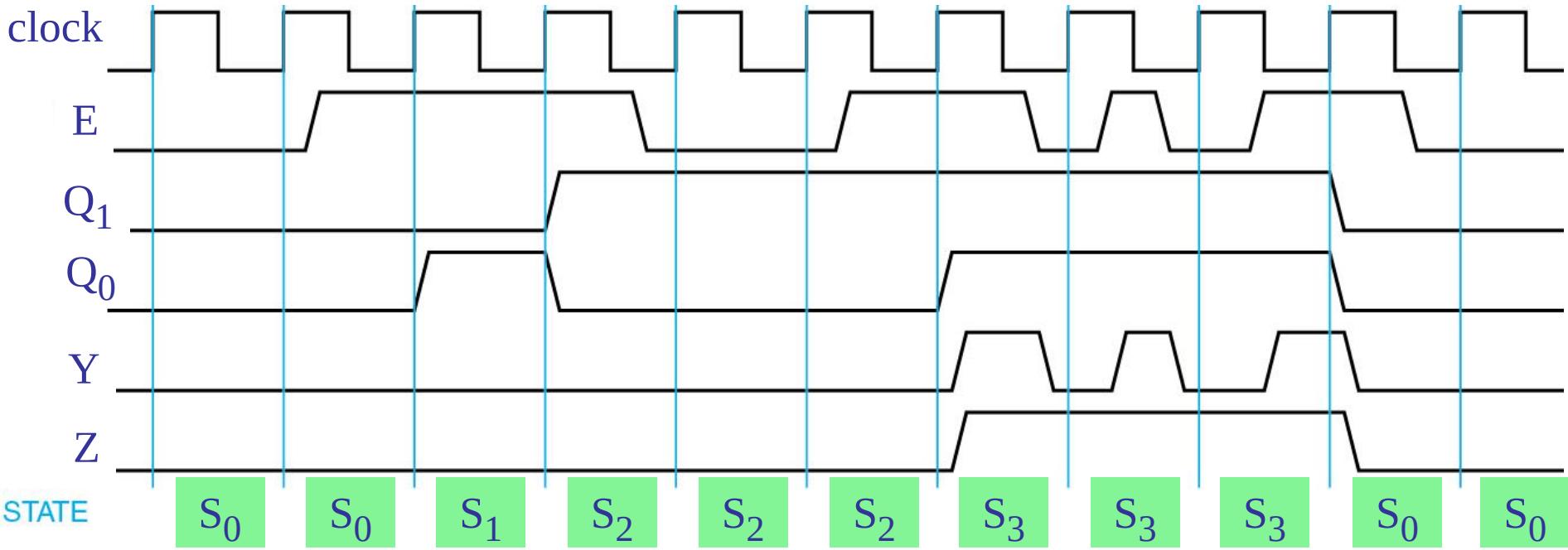
$Q_0^{next} = E \oplus Q_0$

$Q_1^{next} = Q_1 \oplus (EQ_0)$

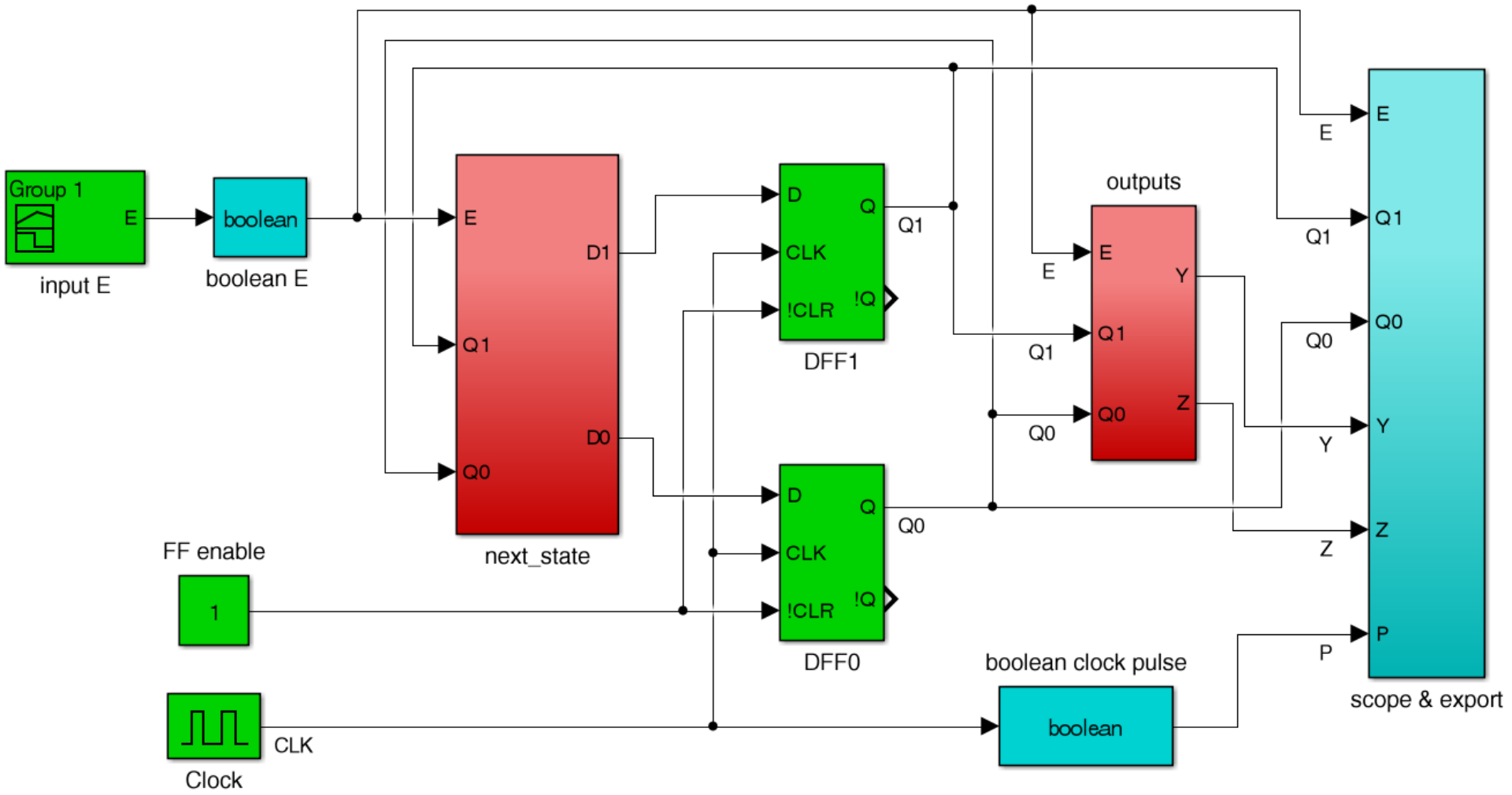
$Y = E\,Q_1\,Q_0$ (Mealy output)

$Z = Q_1\,Q_0$ (Moore output)

Wakerly – Fig. 9-12

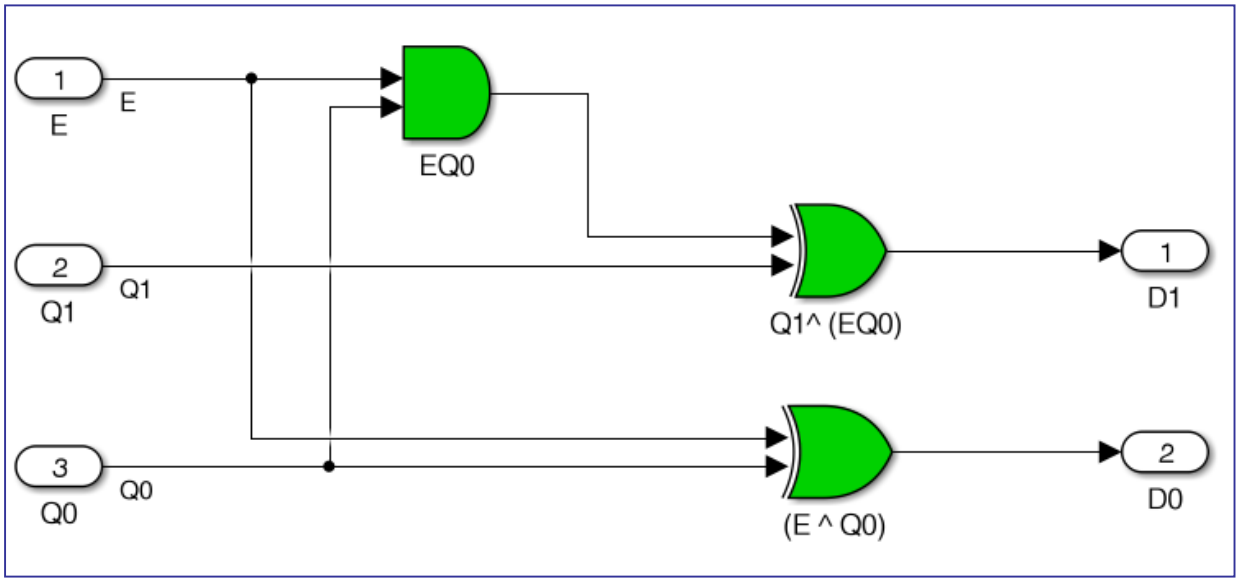


Simulink implementation and timing diagram.



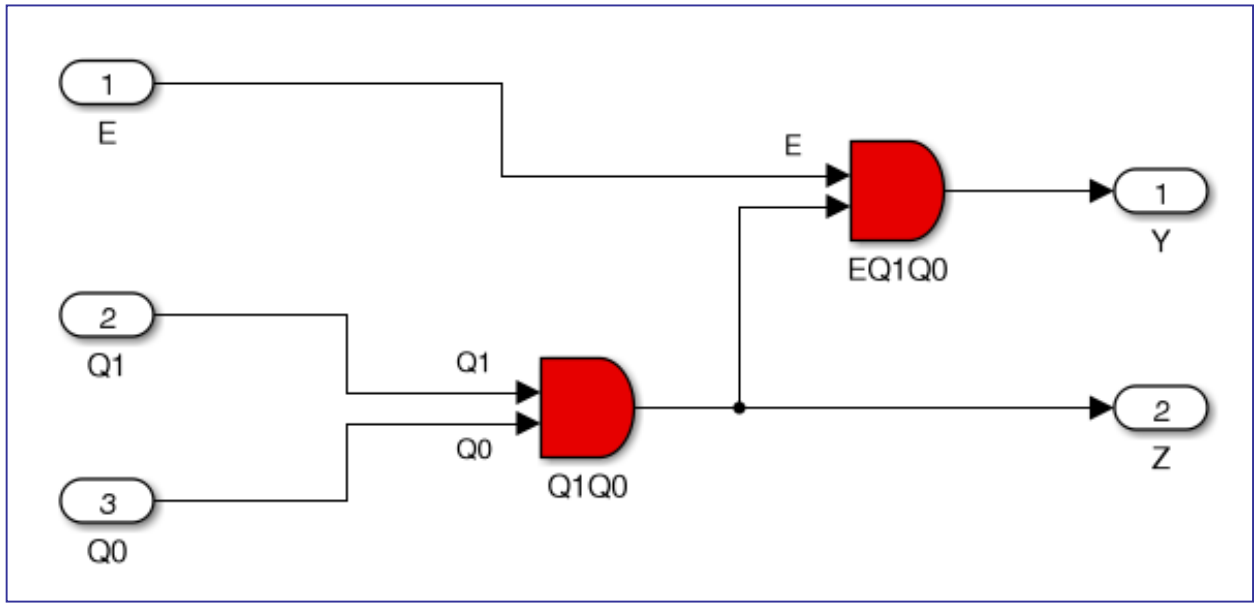
from Simulink file
fsm1eob.slx

Simulink implementation and timing diagram.



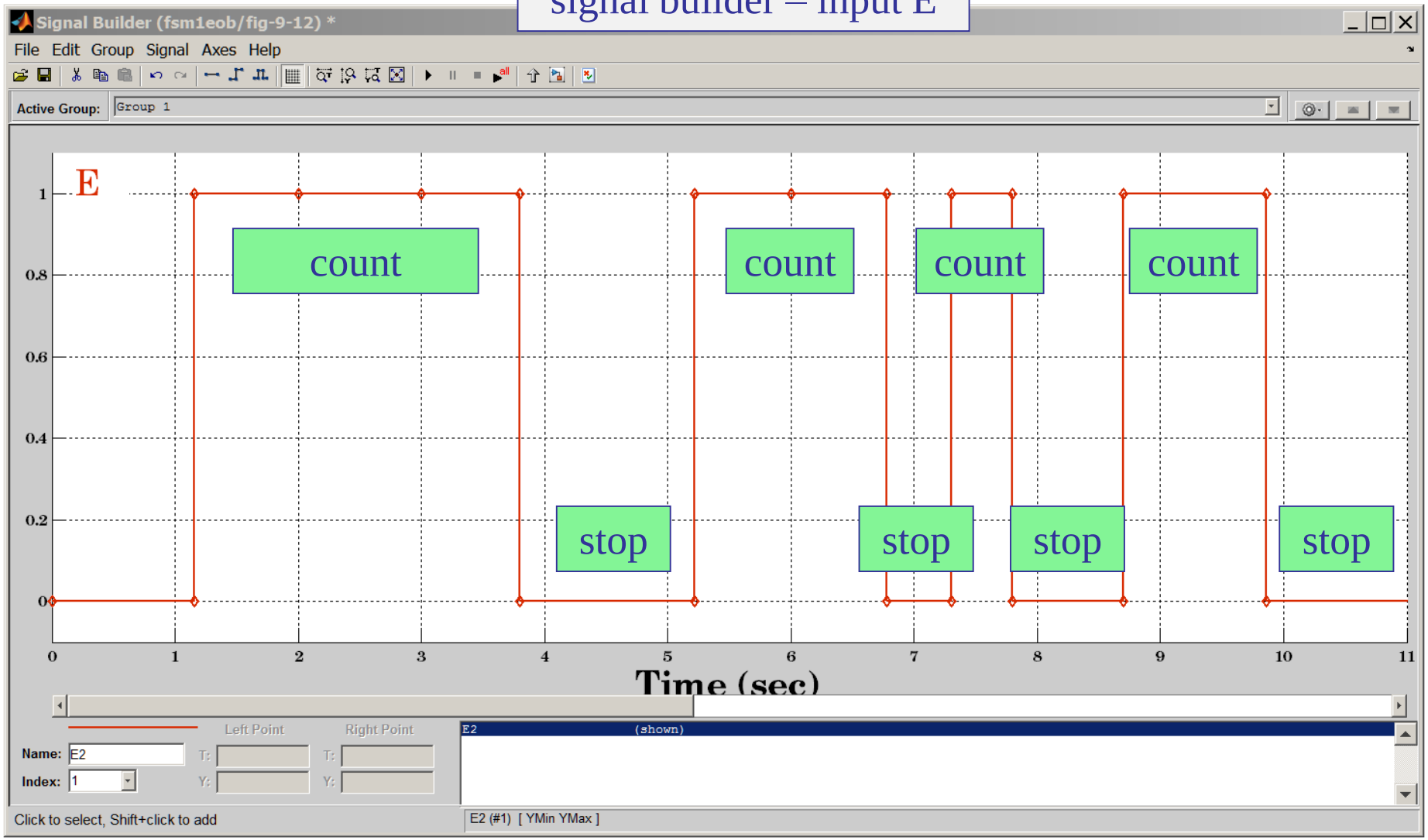
next-state sub-function

output sub-function

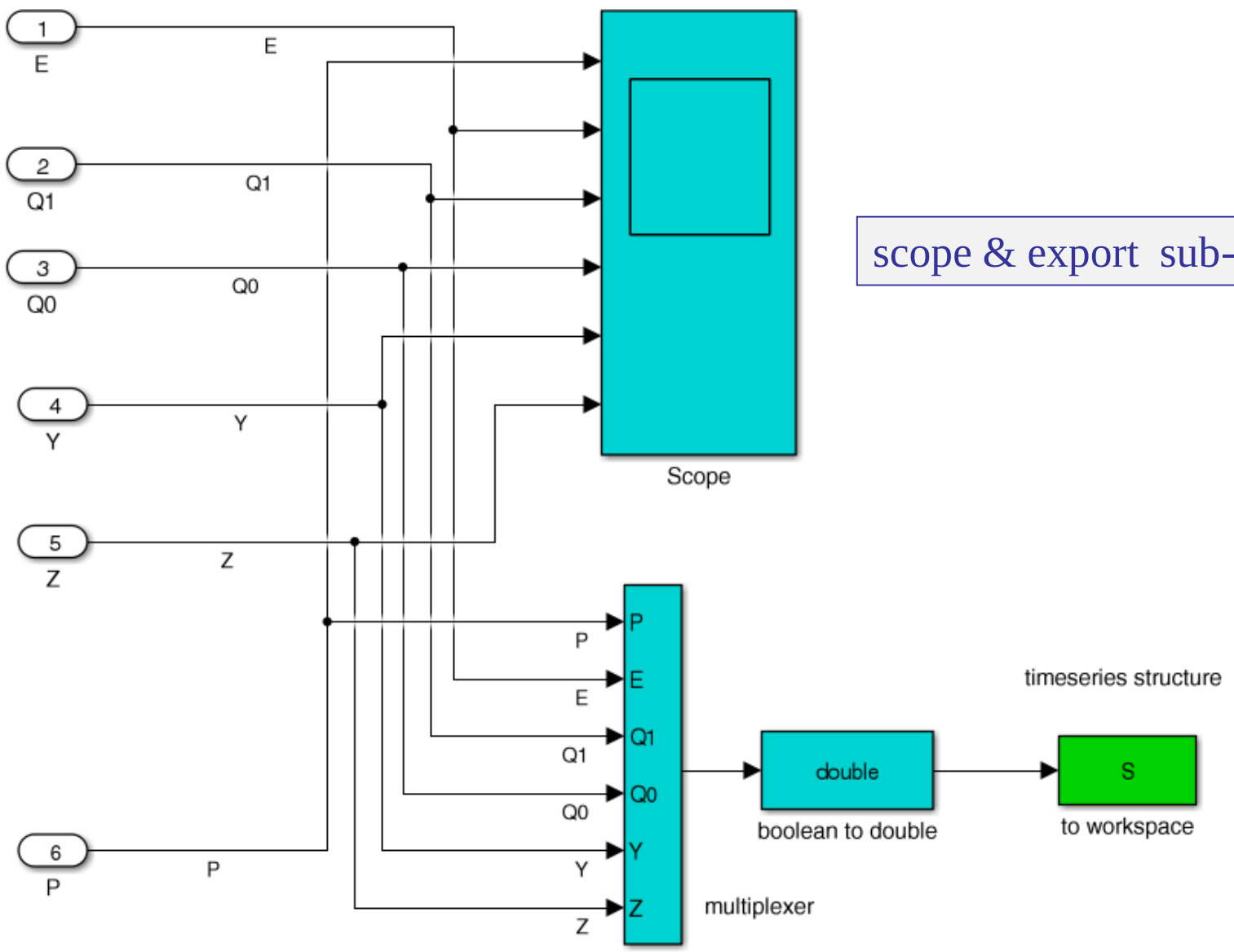


Simulink implementation and timing diagram.

signal builder – input E

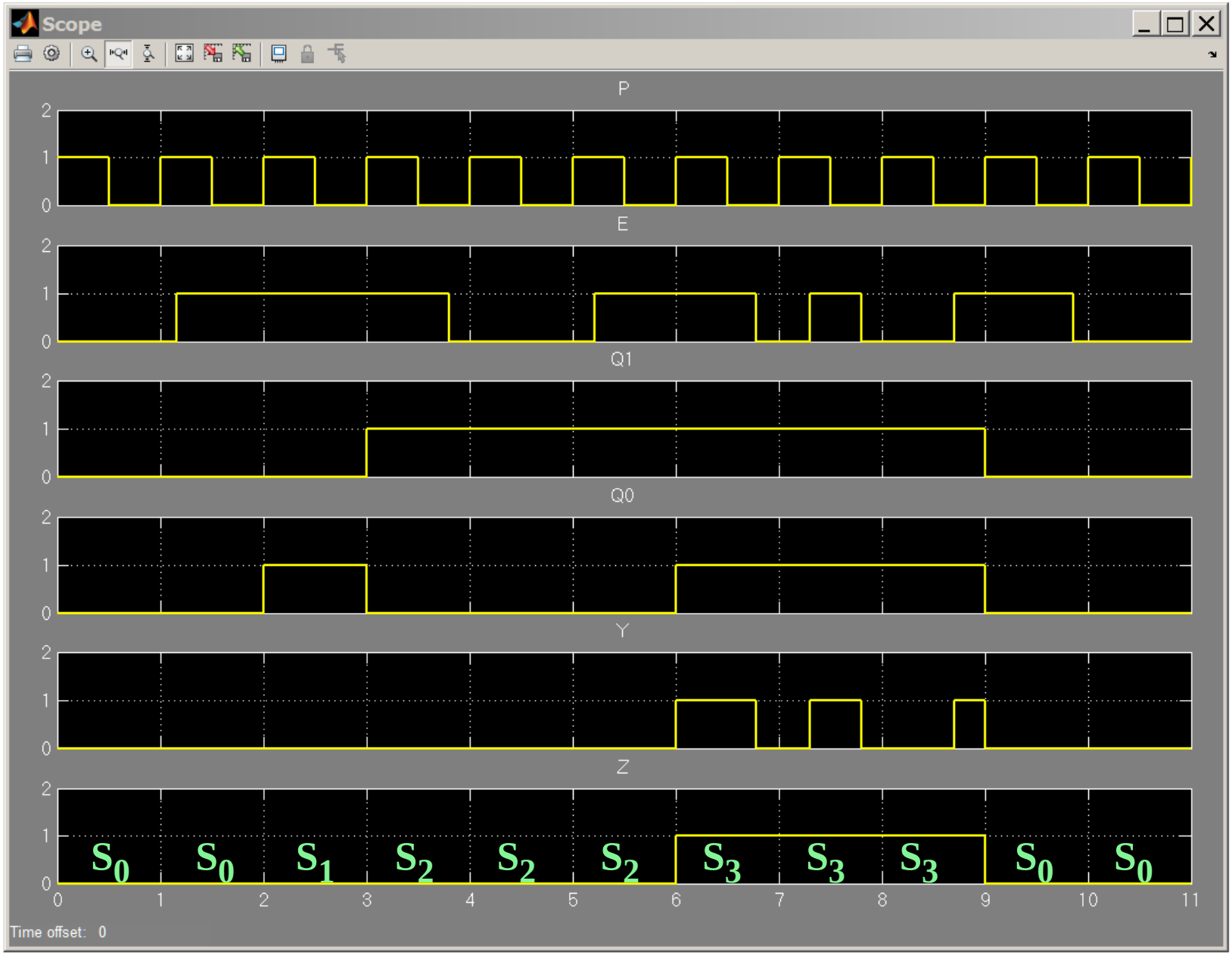


Simulink implementation and timing diagram.



scope & export sub-function

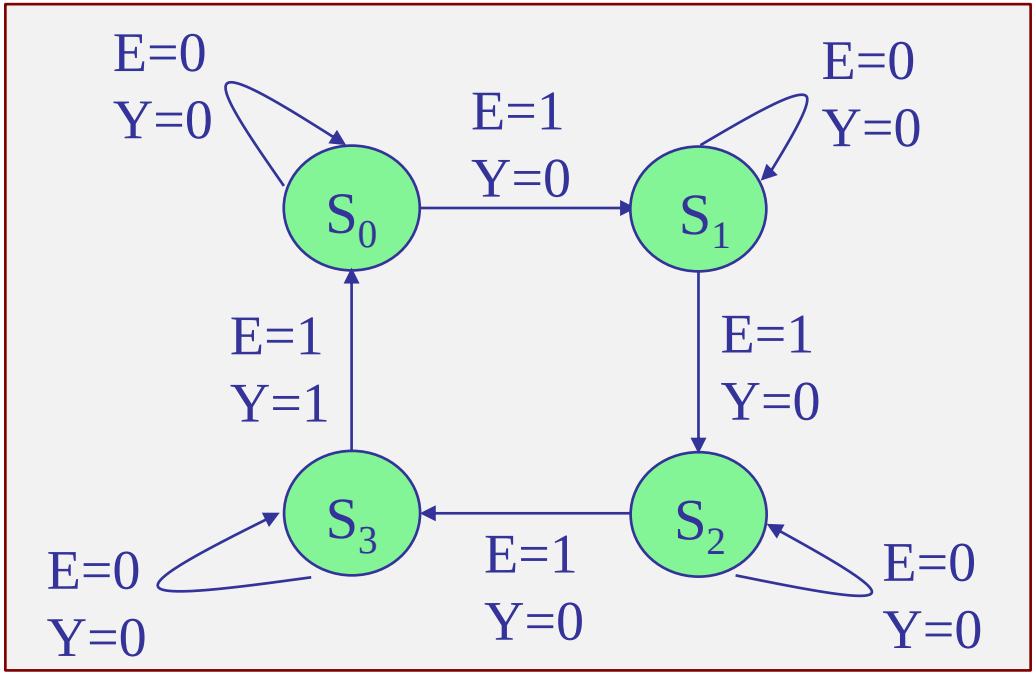
Example 2



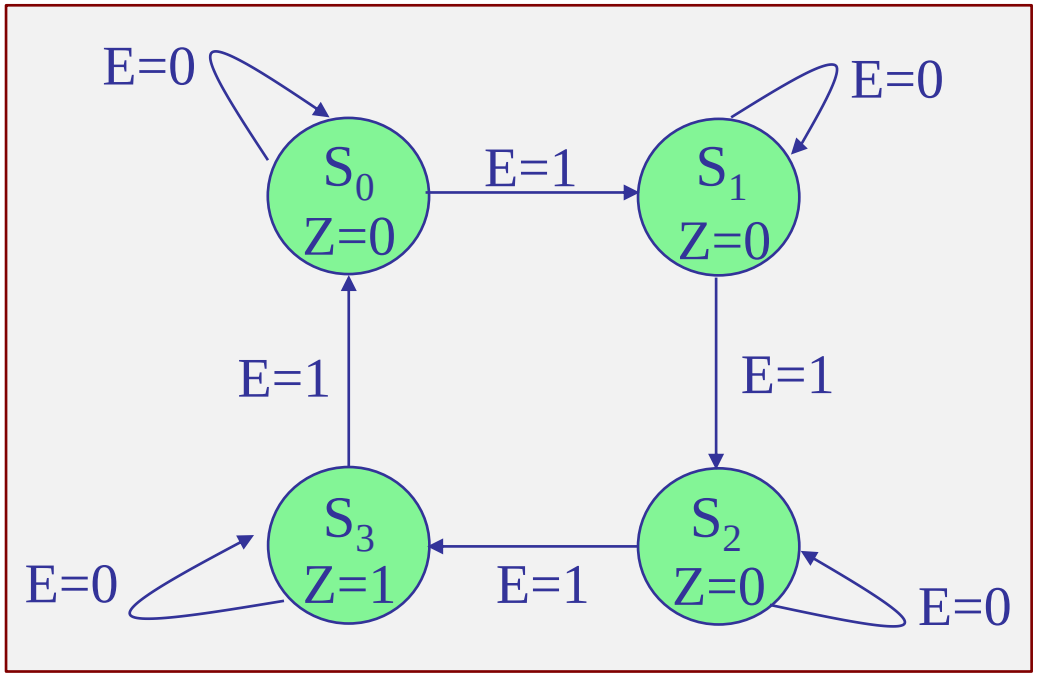
scope output

← **states**

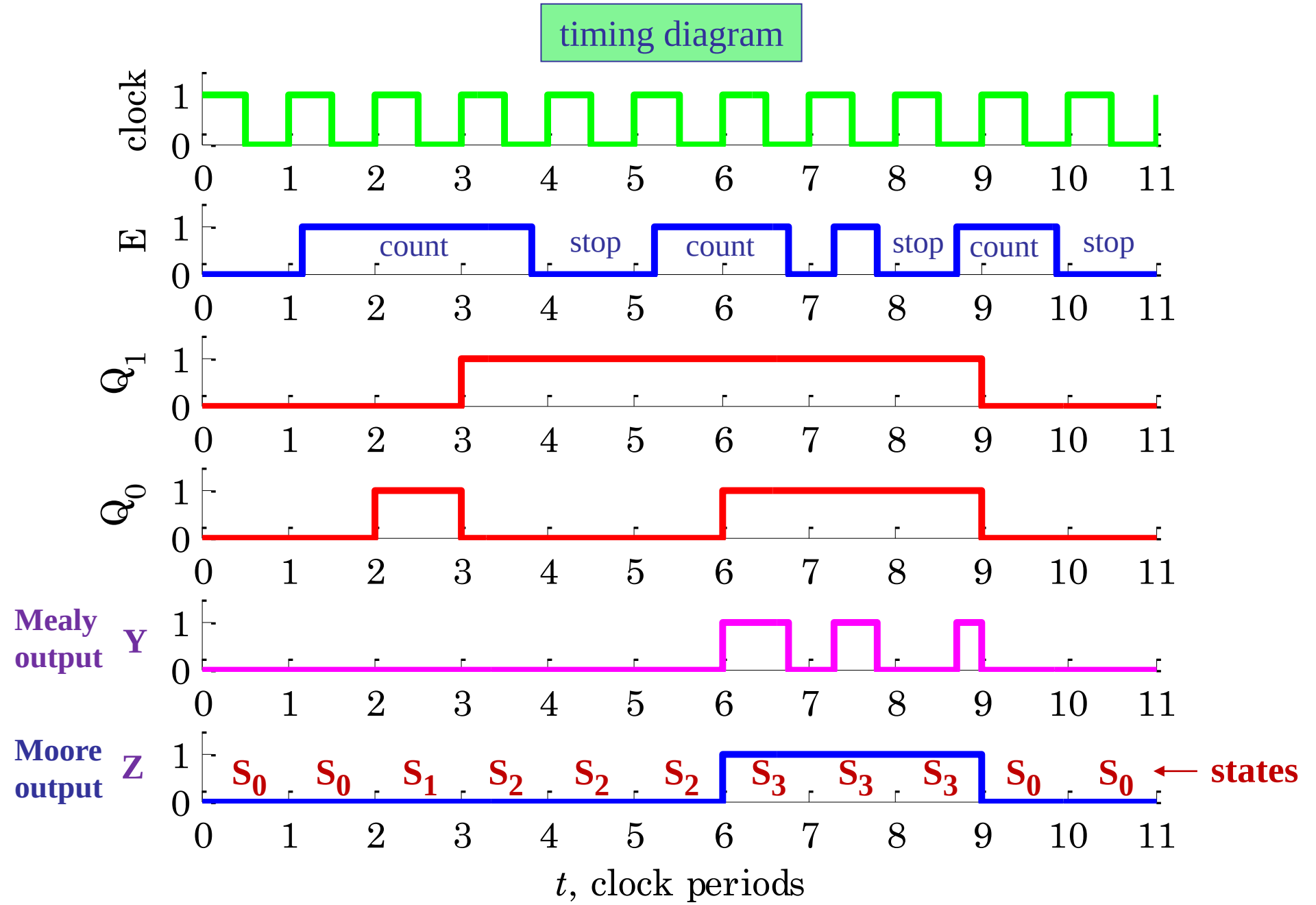
Example 2



Mealy state diagram



Moore state diagram


timing diagram
clock
E
count
stop
count
stop
count
stop
Q_1
Q_0
Mealy output
Y
Moore output
Z
S_0 S_0 S_1 S_2 S_2 S_2 S_3 S_3 S_3 S_0 S_0
states
0 1 2 3 4 5 6 7 8 9 10 11
t, clock periods

Example 3 – Another FSM analysis example from Wakerly, Fig. 9-13.

There are two inputs, X,Y, three flip-flops (states), Q_2, Q_1, Q_0, and two outputs Z_1, Z_2. There are 8 states corresponding to the 8 triplets, $Q_2Q_1Q_0$, and these will be denoted symbolically by the letters, A, B, C, D, E, F, G, H.

Analysis Procedure:

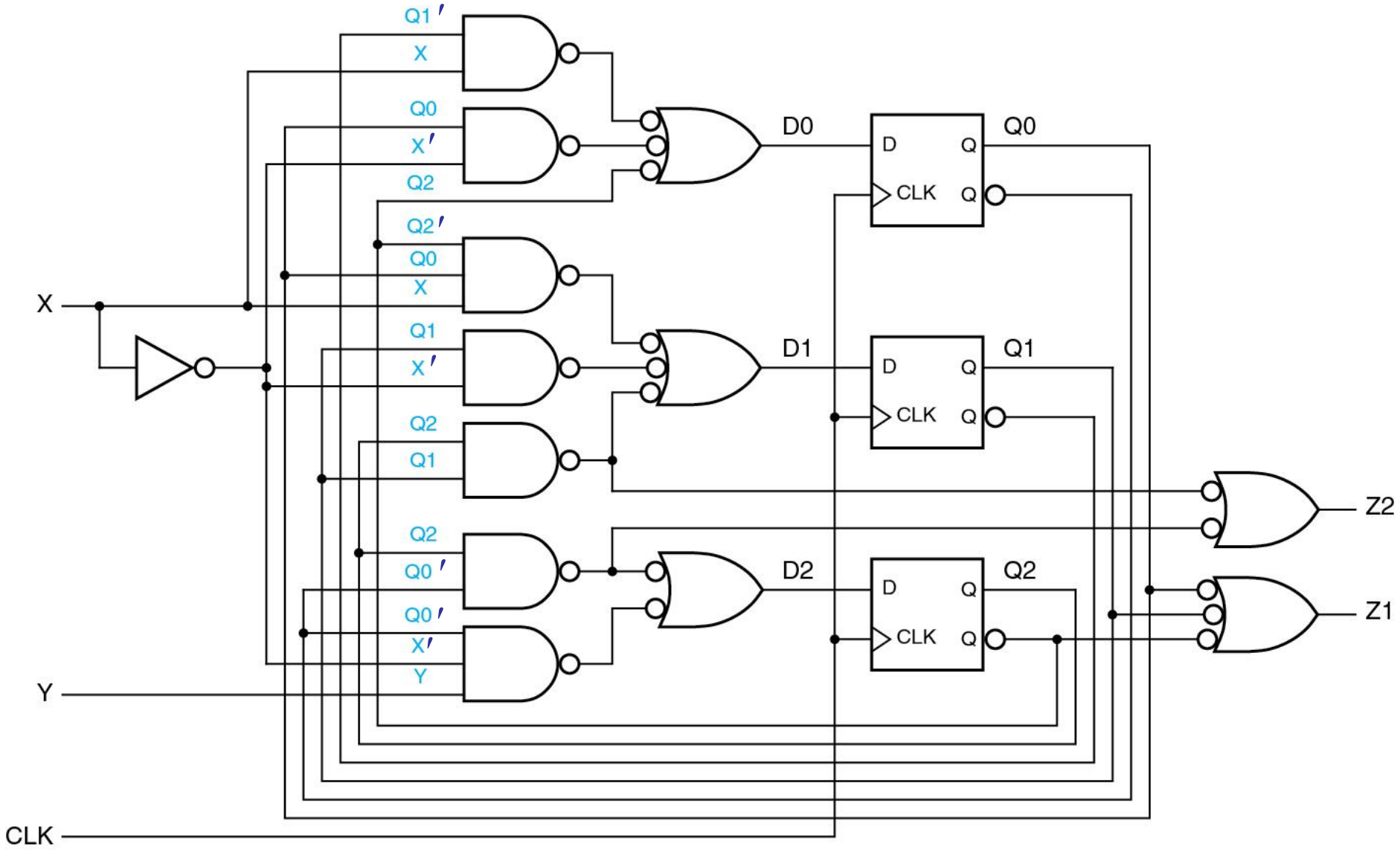
1. The output equations for Z_1, Z_2, and the flip-flop excitation equations for the flip-flop inputs, D_2, D_1, D_0, are read off from the logic diagram.

2. These become the transition equations for the next states, Q_2^{next},Q_1^{next},Q_0^{next}, denoted in Wakerly with the notation, Q_2^*, Q_1^*, Q_0^*.

3. The state-table can the be evaluated for all possible valuations of the inputs and the states, and then, converted into a more readable version using the symbolic state letter-names.

4. Finally, a state diagram is drawn, using a slightly different and more efficient drawing convention in this example.

Wakerly, Fig. 9-13. A State Machine with Three Flip-Flops and Eight States



Example 3

$$D_0 = Q_1' X + Q_0 X' + Q_2$$

$$D_1 = Q_2' Q_0 X + Q_1 X' + Q_2 Q_1$$

$$D_2 = Q_2 Q_0' + Q_0' X' Y$$

flip-flop excitation equations

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

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

$$Q_2^{next} = Q_2 Q_0' + Q_0' X' Y$$

next-state transition equations

$$Z_1 = Q_2 + Q_1' + Q_0'$$

$$Z_2 = Q_2 Q_1 + Q_2 Q_0'$$

output equations – Moore type

state table

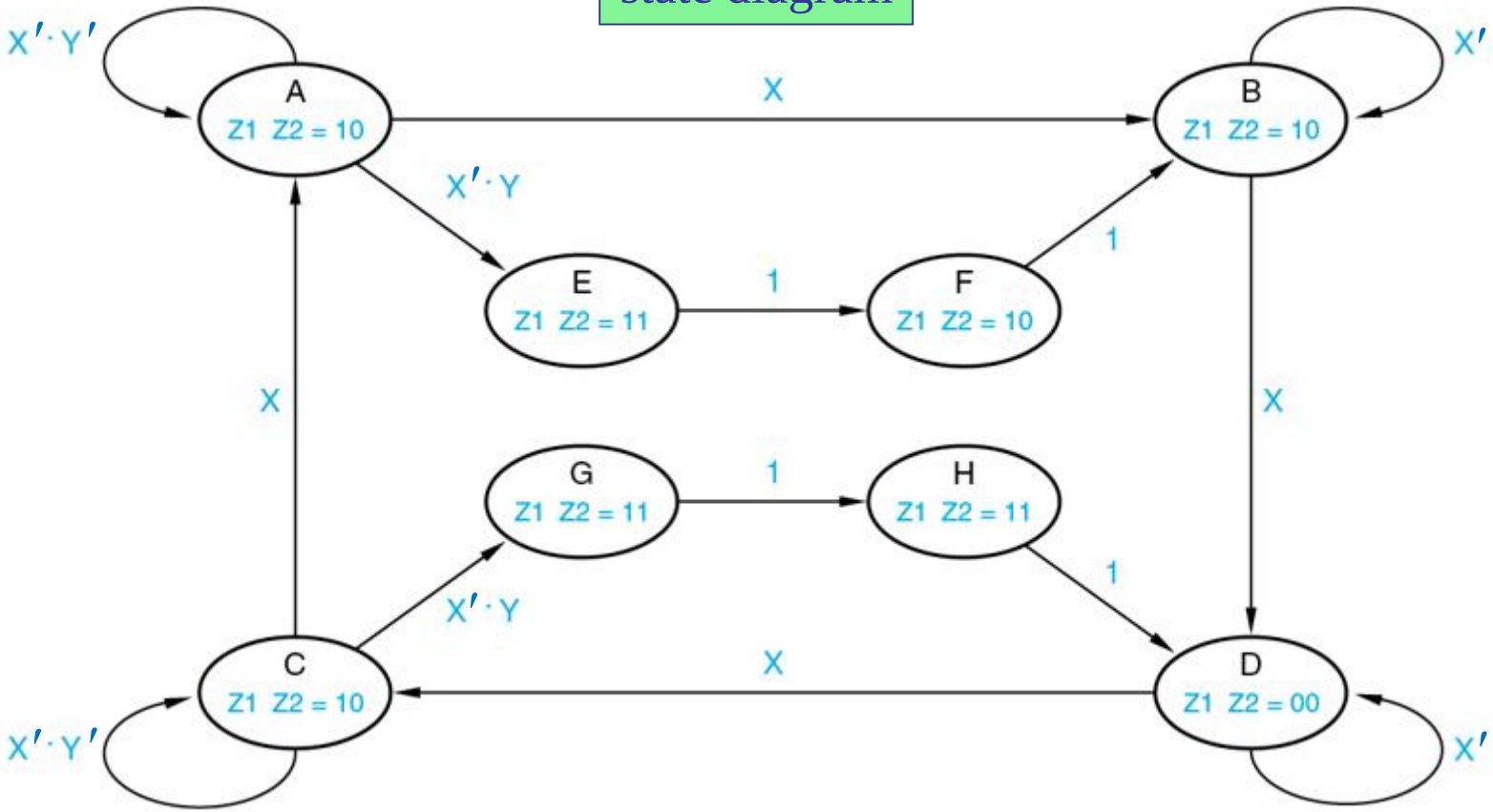
	XY				
Q2 Q1 Q0	00	01	10	11	Z1 Z2
000	000	100	001	001	10
001	001	001	011	011	10
010	010	110	000	000	10
011	011	011	010	010	00
100	101	101	101	101	11
101	001	001	001	001	10
110	111	111	111	111	11
111	011	011	011	011	11
	Q2* Q1* Q0*				

next states
for all valuations
of input pairs XY

symbolic state table

	XY				
S	00	01	10	11	Z1 Z2
A	A	E	B	B	10
B	B	B	D	D	10
C	C	G	A	A	10
D	D	D	C	C	00
E	F	F	F	F	11
F	B	B	B	B	10
G	H	H	H	H	11
H	D	D	D	D	11
	S*				

state diagram



new drawing convention:

use input expressions to label the transitions

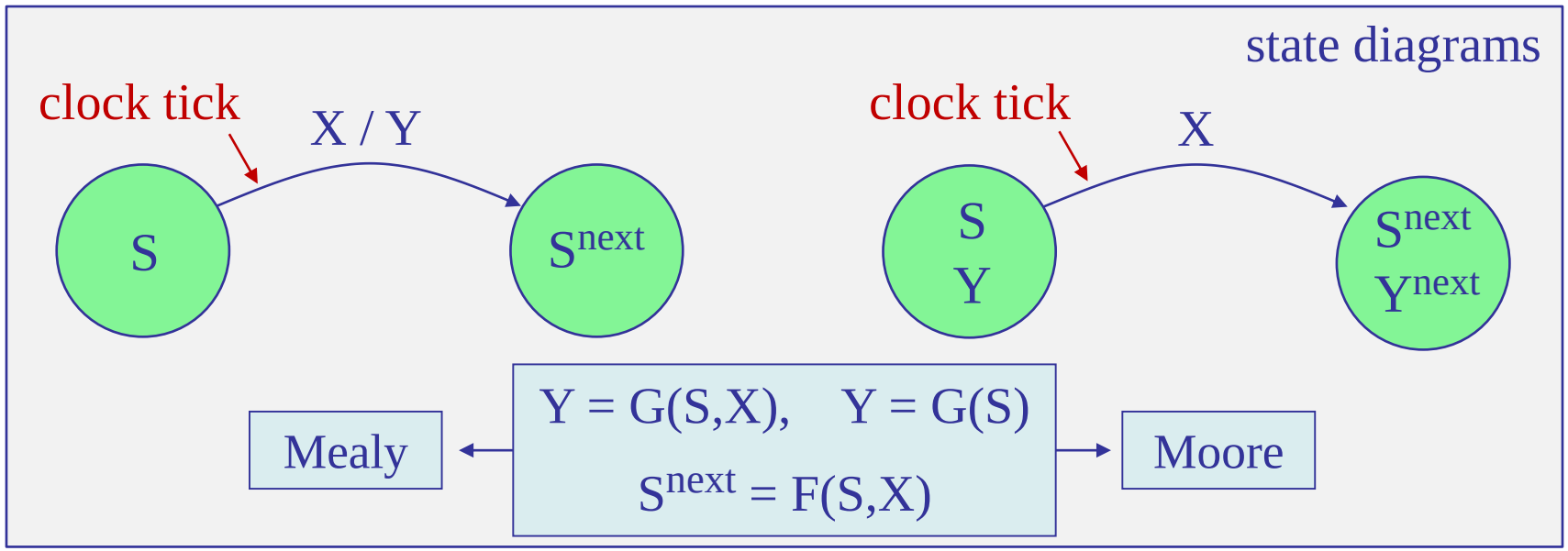
transition	inputs	new label
A → A	XY = 00	X′ Y′
A → E	XY = 01	X′ Y
A → B	XY = 10,11	X Y′ + XY = X

FSM – analysis / synthesis / design steps – Summary

analysis ↑

1. Start with verbal description and choose states and the reset state.
2. Convert the verbal description into a state diagram.
3. Convert the state diagram into a state table.
4. Encode the symbolic states with D-flip-flop state variables.
5. Derive the next-state and output equations in terms of state variables.
6. Implement the design using D-flip-flops and logic gates.
7. Simulate it with Simulink or HDL to verify proper operation.

↓ synthesis



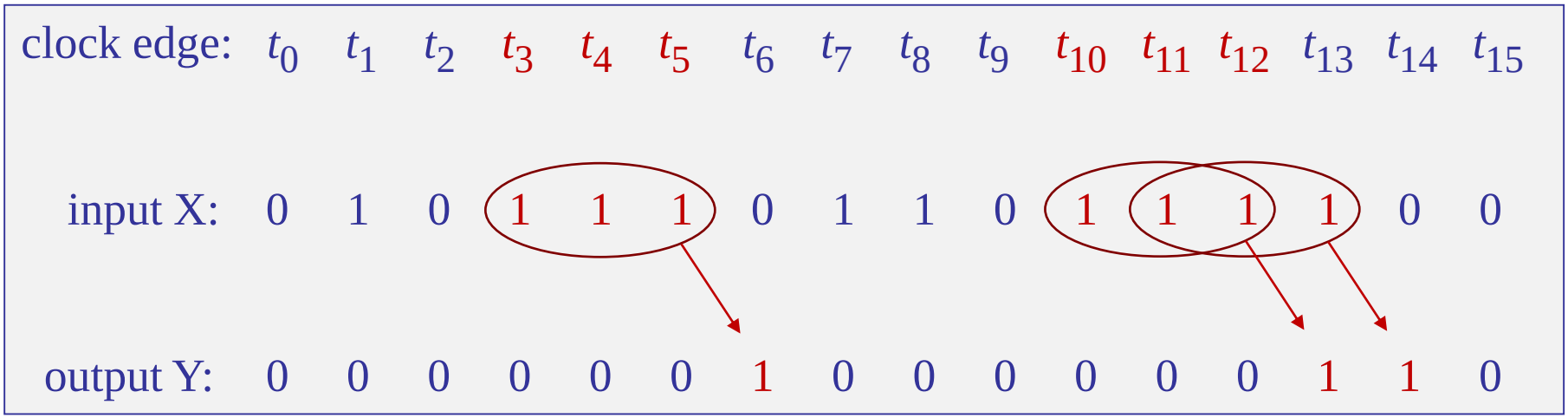
Example 4 – Sequence recognizer – Moore version. It is desired to design an FSM to detect the particular sequence of three or more consecutive ones, 111, in an incoming input stream X of 0s and 1s, measured at the positive edges of the clock signal. At each time instant, upon detecting three ones in the three immediately preceding time instants, the FSM should produce an output, Y = 1, otherwise, it should output, Y = 0. All changes should occur at the positive edges of the clock signal.

To put this in some context, one can think of a car cruise control system in which the speed is sampled at regular clock-edge time instants, and an indicator binary signal X indicates whether the speed is within acceptable limits (X=0), or, that the speed has become too excessive (X=1) , and in that case, a control signal, Y = 1, must be asserted to cause the car to slow down to the acceptable range. The control signal should remain at Y = 0 while the speed is within the acceptable range.

Thus, the design requirement is that when the speed is measured to be excessive (X=1) in three consecutive time instants, a control action must be taken to slow the car down, otherwise, no action is required.

[cf. Brown & Vranesic]

example input signal and output



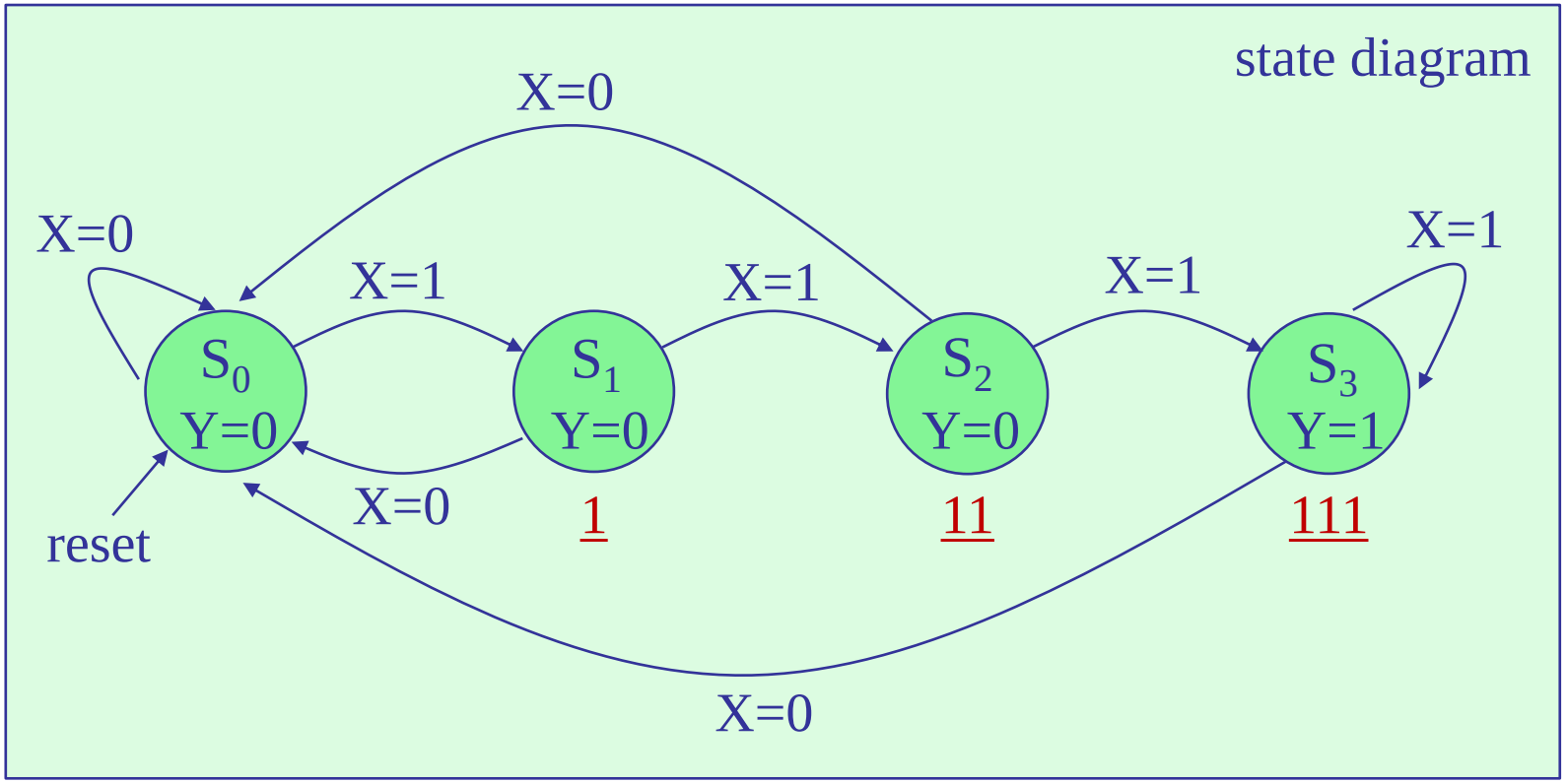
three consecutive 1s at t_3, t_4, t_5
will cause an output Y=1
at the next clock edge t_6

three consecutive 1s at t_{10}, t_{11}, t_{12}
will cause an output Y=1
at the next clock edge t_{13}

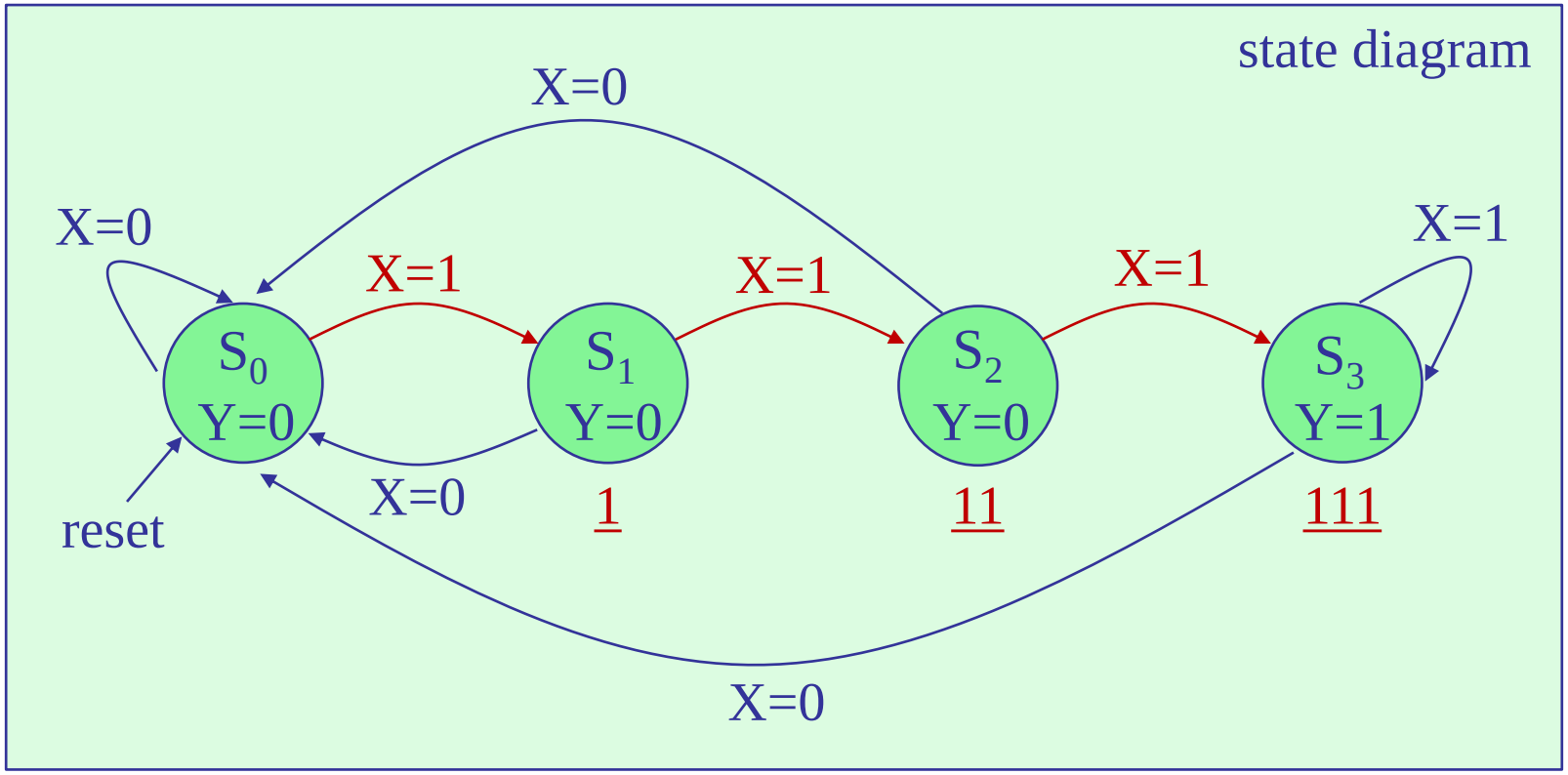
continuing with another
set of three 1s at t_{11}, t_{12}, t_{13}
will cause another output Y=1
at the next clock edge t_{14}

example input signal, state transition, and output

clock edge:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
input X:	0	1	0	1	1	1	0	1	1	0	1	1	1	1	0	0
state:	S_0	S_0	S_1	S_0	S_1	S_2	S_3	S_0	S_1	S_2	S_0	S_1	S_2	S_3	S_3	S_0
output Y:	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0



based on the current X, state transitions take place at the next active clock



state table

Moore FSM

present state	next state		output Y
	X=0	X=1	
S_0	S_0	S_1	0
S_1	S_0	S_2	0
S_2	S_0	S_3	0
S_3	S_0	S_3	1

Moore FSM – state table

compact form

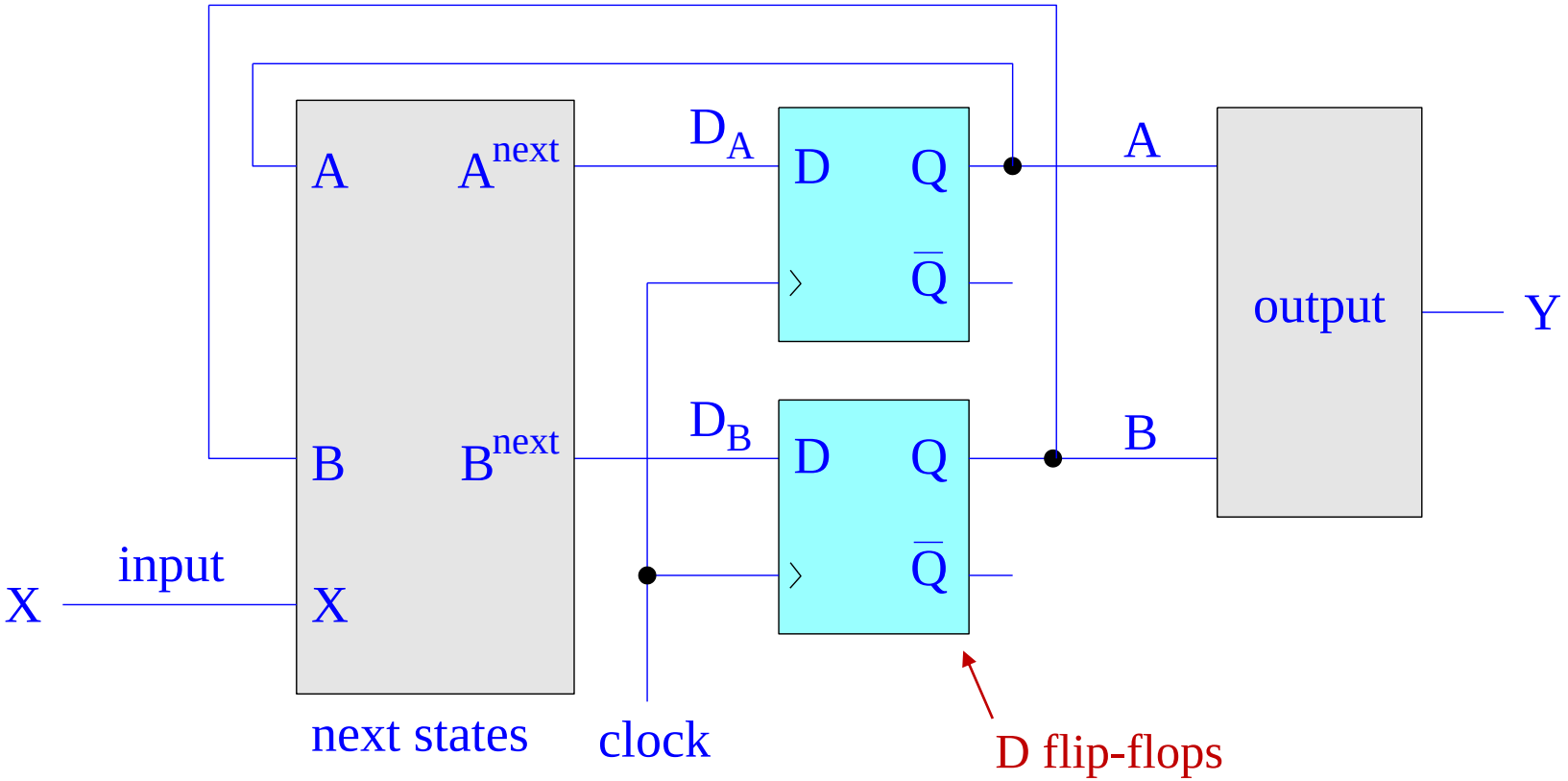
present state	next state X=0	next state X=1	output Y
S_0	S_0	S_1	0
S_1	S_0	S_2	0
S_2	S_0	S_3	0
S_3	S_0	S_3	1

conventional form

X	present state	next state	Y
0	S_0	S_0	0
0	S_1	S_0	0
0	S_2	S_0	0
0	S_3	S_0	1
1	S_0	S_1	0
1	S_1	S_2	0
1	S_2	S_3	0
1	S_3	S_3	1

State assignment. With N=4 states, we need, n = ceiling($\log_2 N$) = 2, state variables, say, A,B, and two D flip-flops, with inputs, say, D_A, D_B.

The overall system will be as shown below.



State encoding. With two state variables, say, A,B, we have two options for state encoding, i.e., associating A,B with the four states, S_0, S_1, S_2, S_3,

(1) plain binary, or,
(2) Gray coding

First, consider plain binary, and re-write the state table in conventional form.

Moore FSM – state table

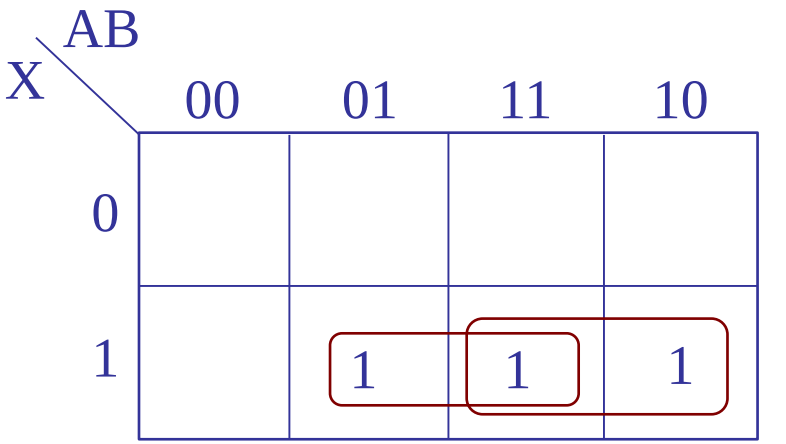
X	present state	A	B	next state	A	B	Y
0	S_0	0	0	S_0	0	0	0
0	S_1	0	1	S_0	0	0	0
0	S_2	1	0	S_0	0	0	0
0	S_3	1	1	S_0	0	0	1
1	S_0	0	0	S_1	0	1	0
1	S_1	0	1	S_2	1	0	0
1	S_2	1	0	S_3	1	1	0
1	S_3	1	1	S_3	1	1	1

binary encoding

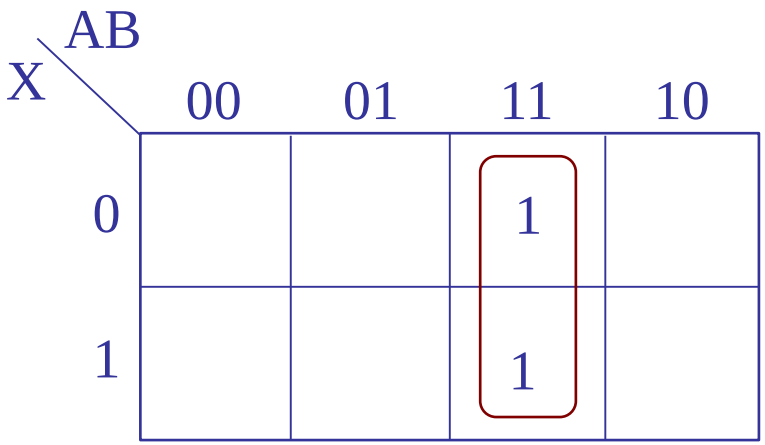
	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	0
S_3 ≡	1	1

Moore FSM – binary encoding

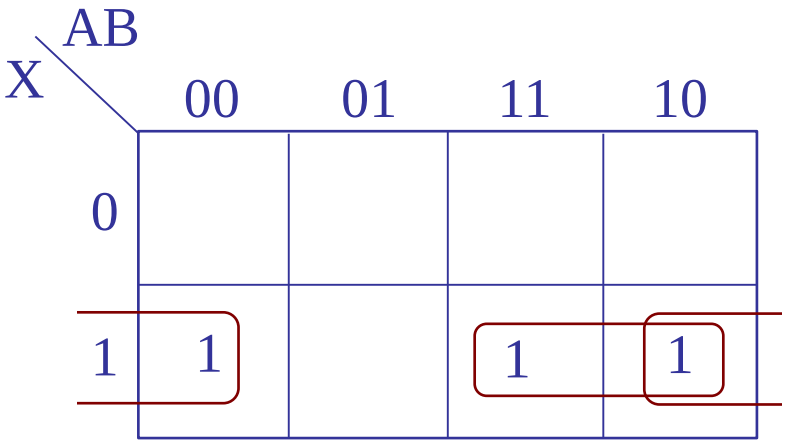
X	present A	present B	next A	next B	Y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	1	1



$D_A = A^{next} = X\,A + X\,B$



$Y = A\,B$ (Moore output)



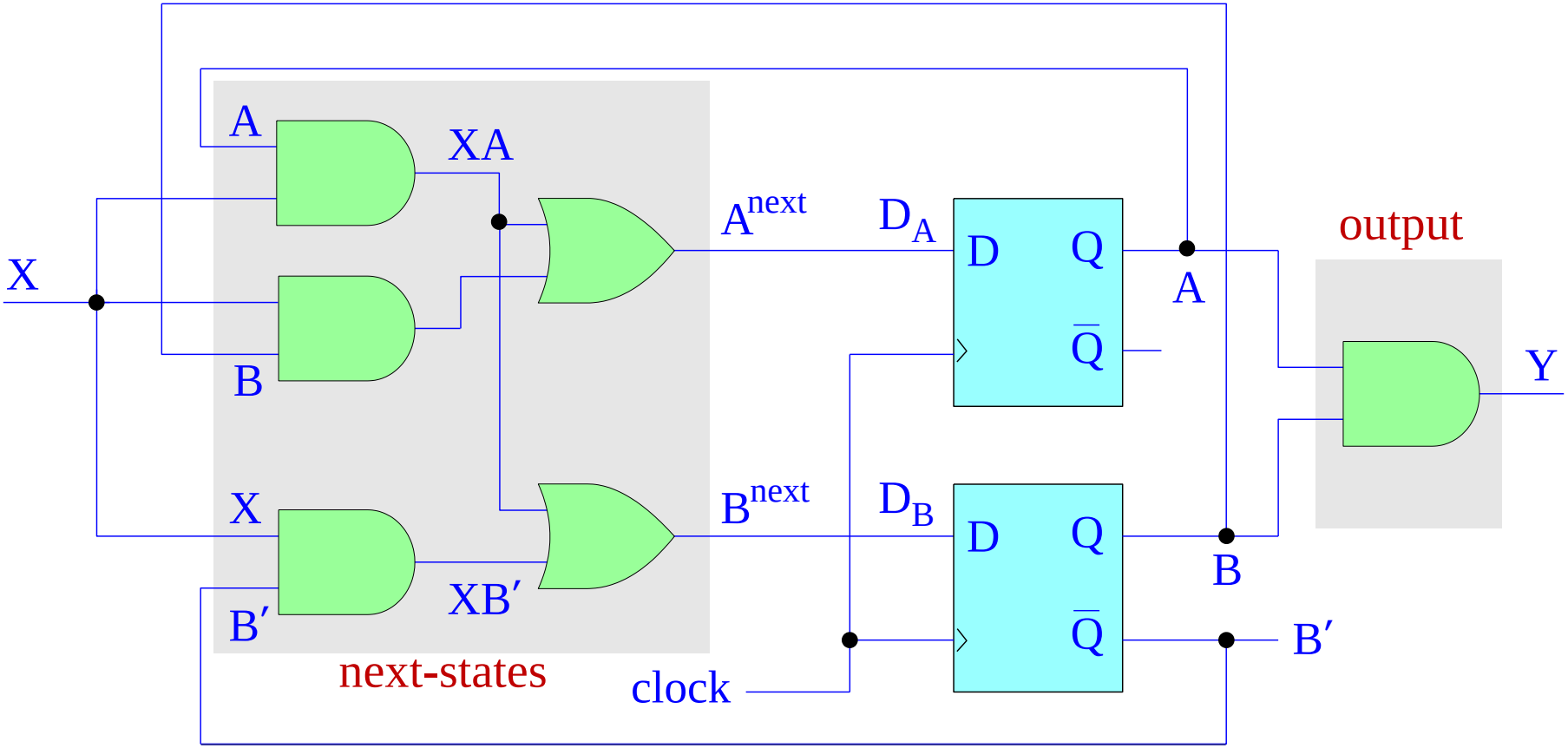
$D_B = B^{next} = X\,A + X\,B'$

Moore FSM realization binary encoding

$D_A = A^{next} = X(A + B)$

$D_B = B^{next} = X(A + B')$

$Y = AB$ (Moore output)

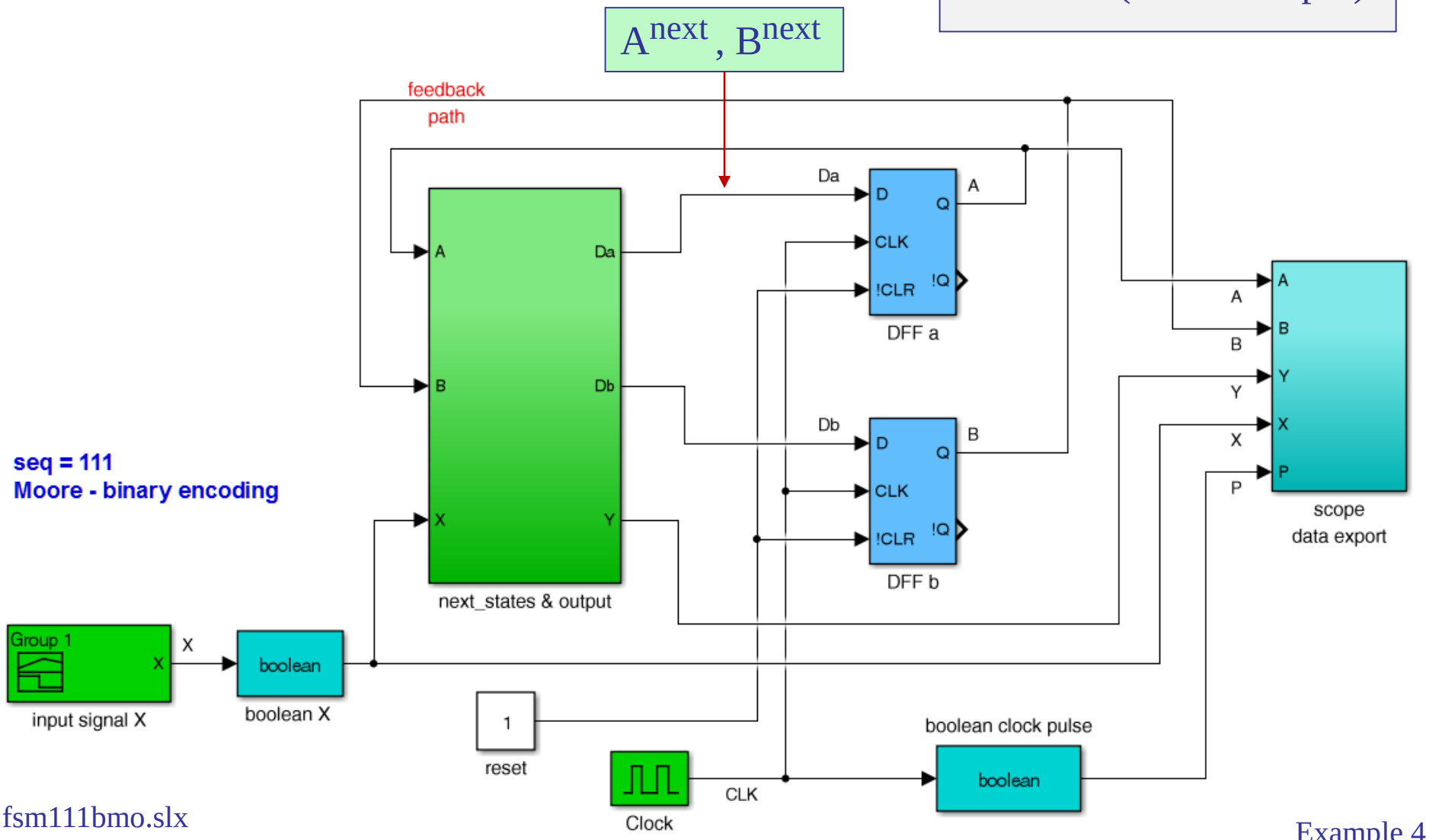


Simulink implementation and timing diagram

$$D_A = A^{next} = X(A + B)$$

$$D_B = B^{next} = X(A + B')$$

$$Y = AB \quad \text{(Moore output)}$$



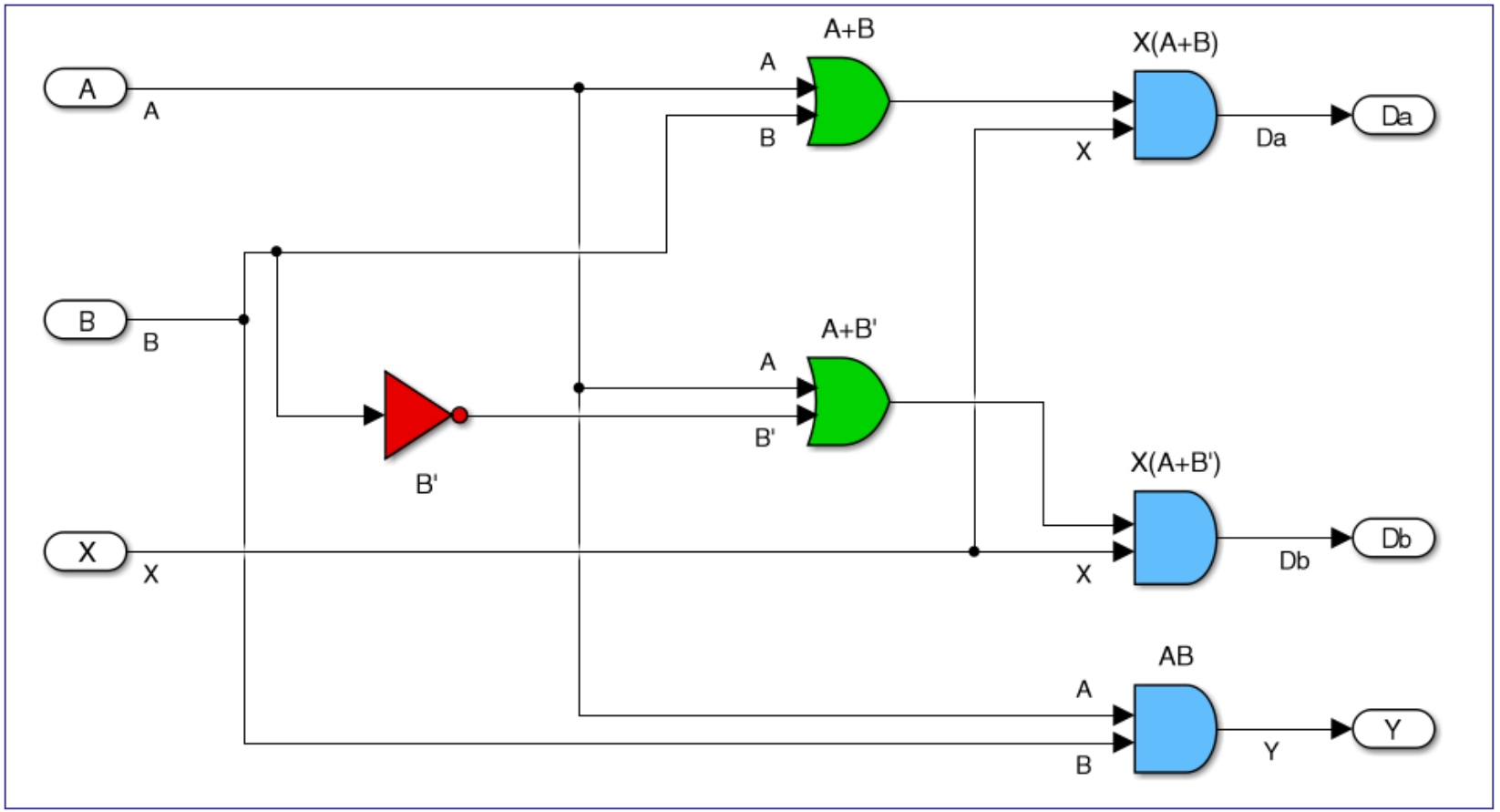
fsm111bmo.slx

next-states & output sub-function

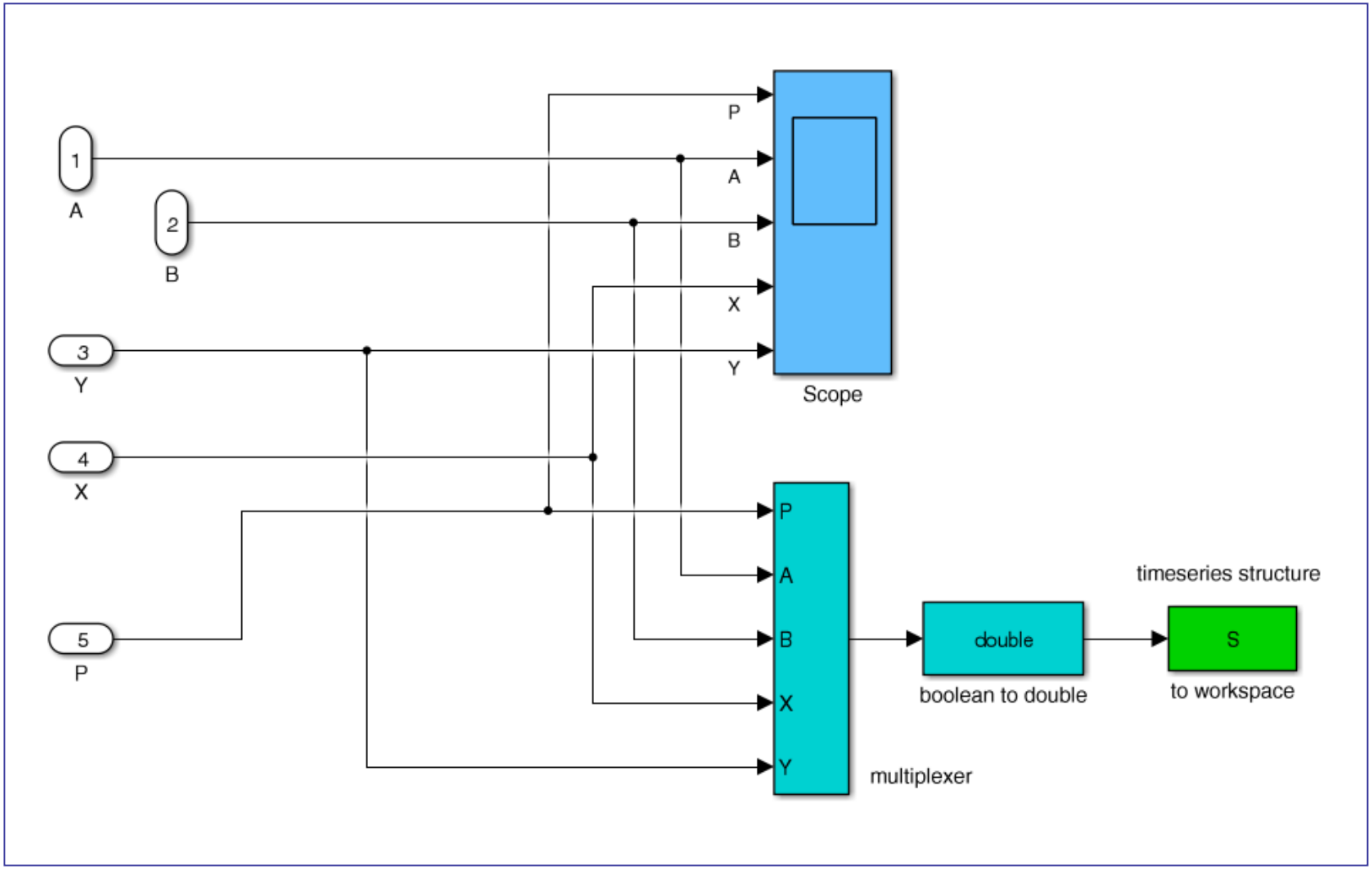
$D_A = A^{next} = X(A + B)$

$D_B = B^{next} = X(A + B')$

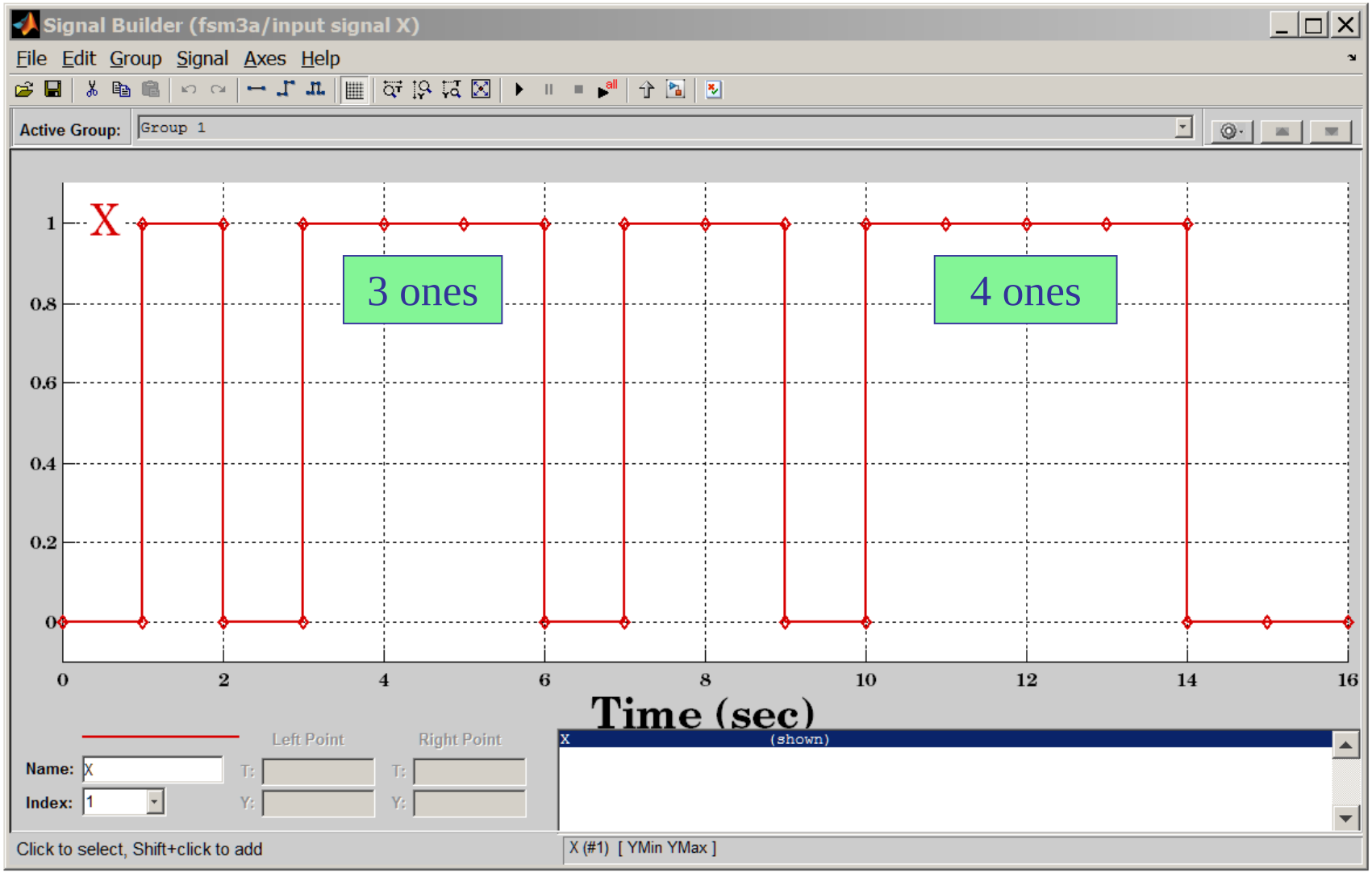
$Y = AB$ (Moore output)

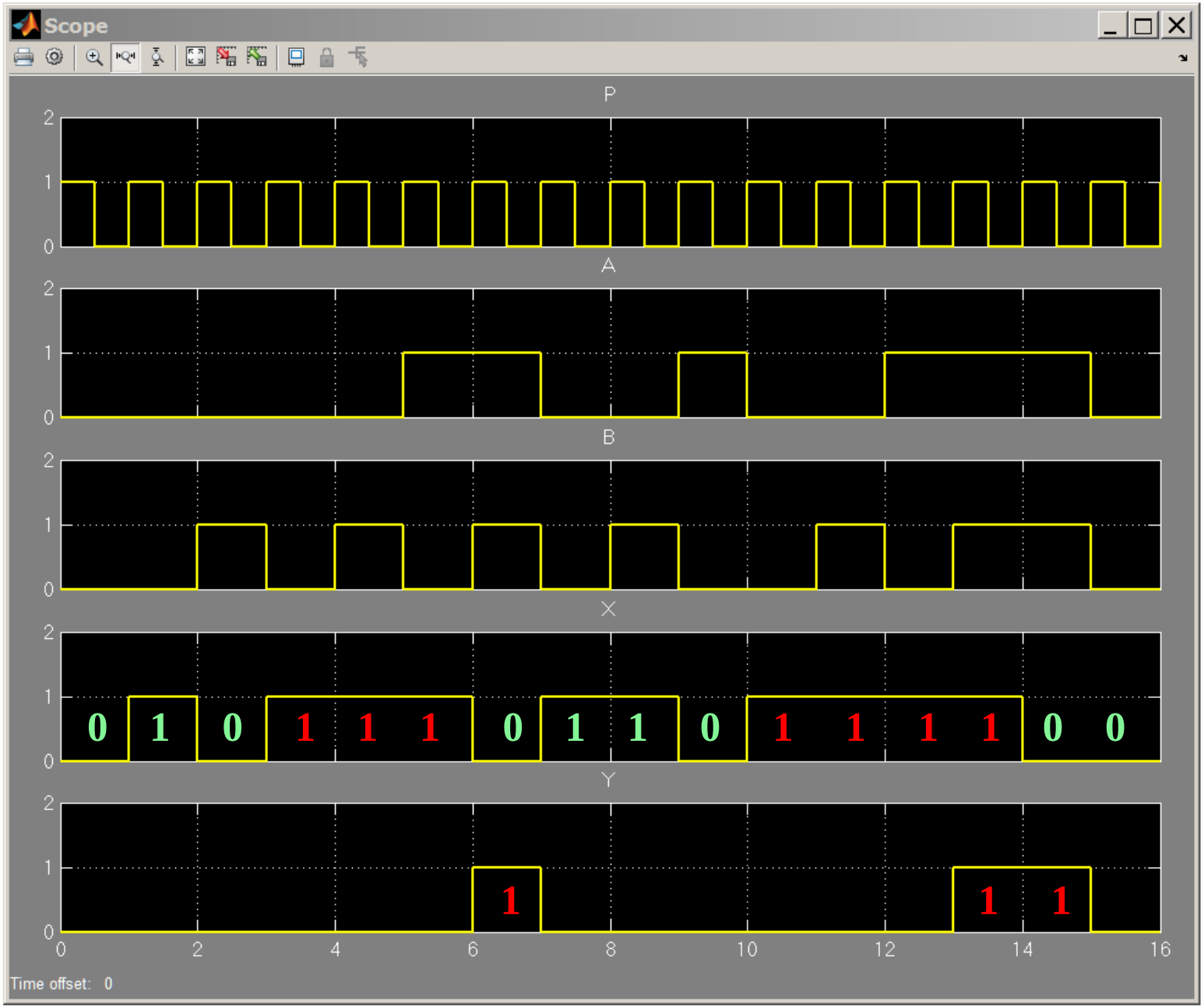


scope & data export sub-function



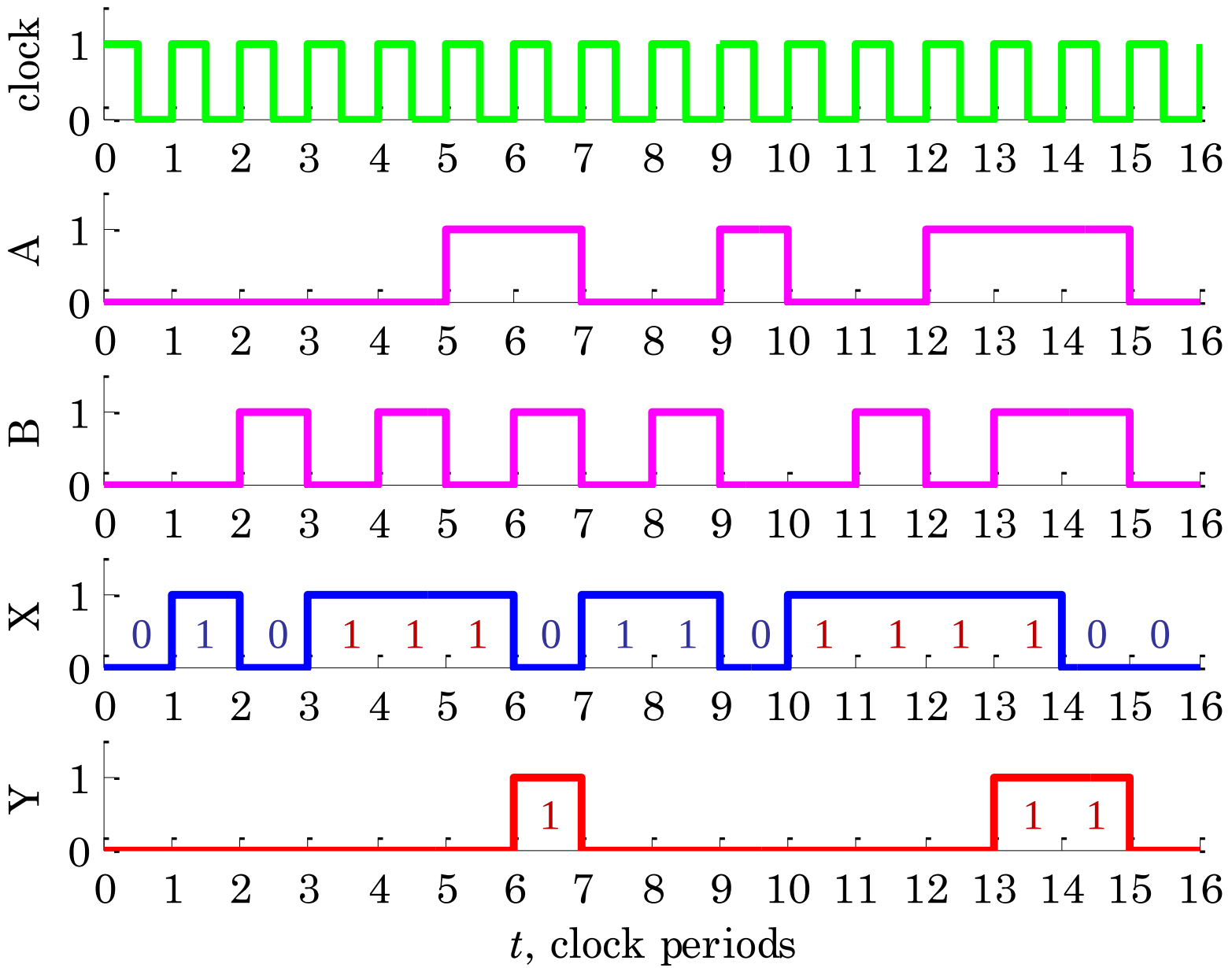
signal builder – input X





scope output

timing diagram



timing diagram

```
% MATLAB code for generating the timing diagram
% with data imported from the Simulink structure S

t = S.time;          % time
P = S.data(:,1);     % clock pulse
A = S.data(:,2);
B = S.data(:,3);
X = S.data(:,4);
Y = S.data(:,5);


figure;
subplot(5,1,1); stairs(t,P, 'g-','linewidth',2);
subplot(5,1,2); stairs(t,A, 'm-','linewidth',2);
subplot(5,1,3); stairs(t,B, 'm-','linewidth',2);
subplot(5,1,4); stairs(t,X, 'b-','linewidth',2);
subplot(5,1,5); stairs(t,Y, 'r-','linewidth',2);
xlabel('{\itt}, clock periods')
```

State encoding. Next, consider Gray encoding, and re-write the state table in conventional form.

Moore FSM – state table

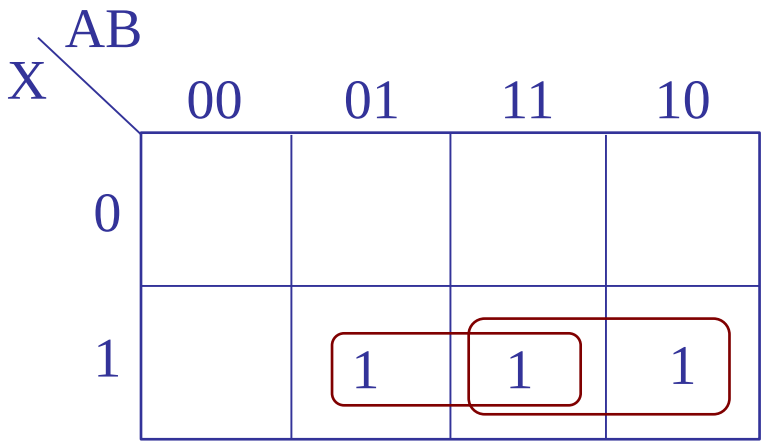
X	present state	A	B	next state	A	B	Y
0	S_0	0	0	S_0	0	0	0
0	S_1	0	1	S_0	0	0	0
0	S_2	1	1	S_0	0	0	0
0	S_3	1	0	S_0	0	0	1
1	S_0	0	0	S_1	0	1	0
1	S_1	0	1	S_2	1	1	0
1	S_2	1	1	S_3	1	0	0
1	S_3	1	0	S_3	1	0	1

Gray encoding

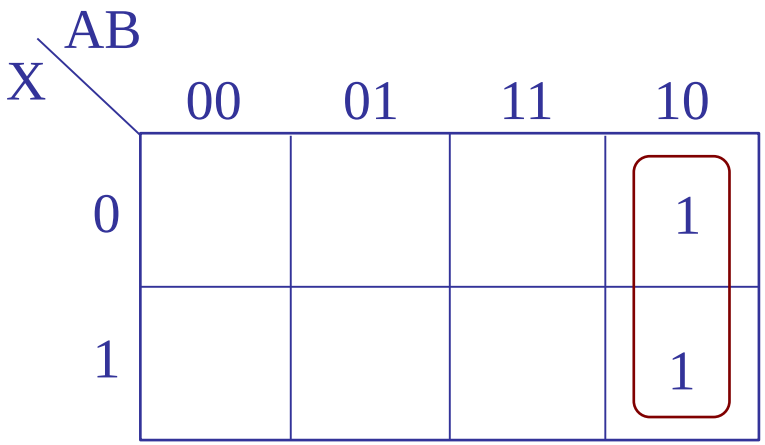
	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	1
S_3 ≡	1	0

Moore FSM – Gray encoding

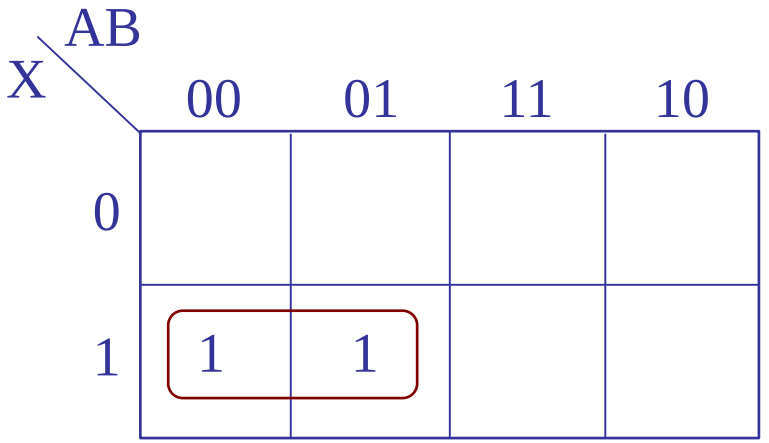
X	present A	present B	next A	next B	Y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	1	0	0	0
0	1	0	0	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	1	1	0	0
1	1	0	1	0	1



$D_A = A^{next} = X\,A + X\,B$



$Y = A\,B'$ (Moore output)



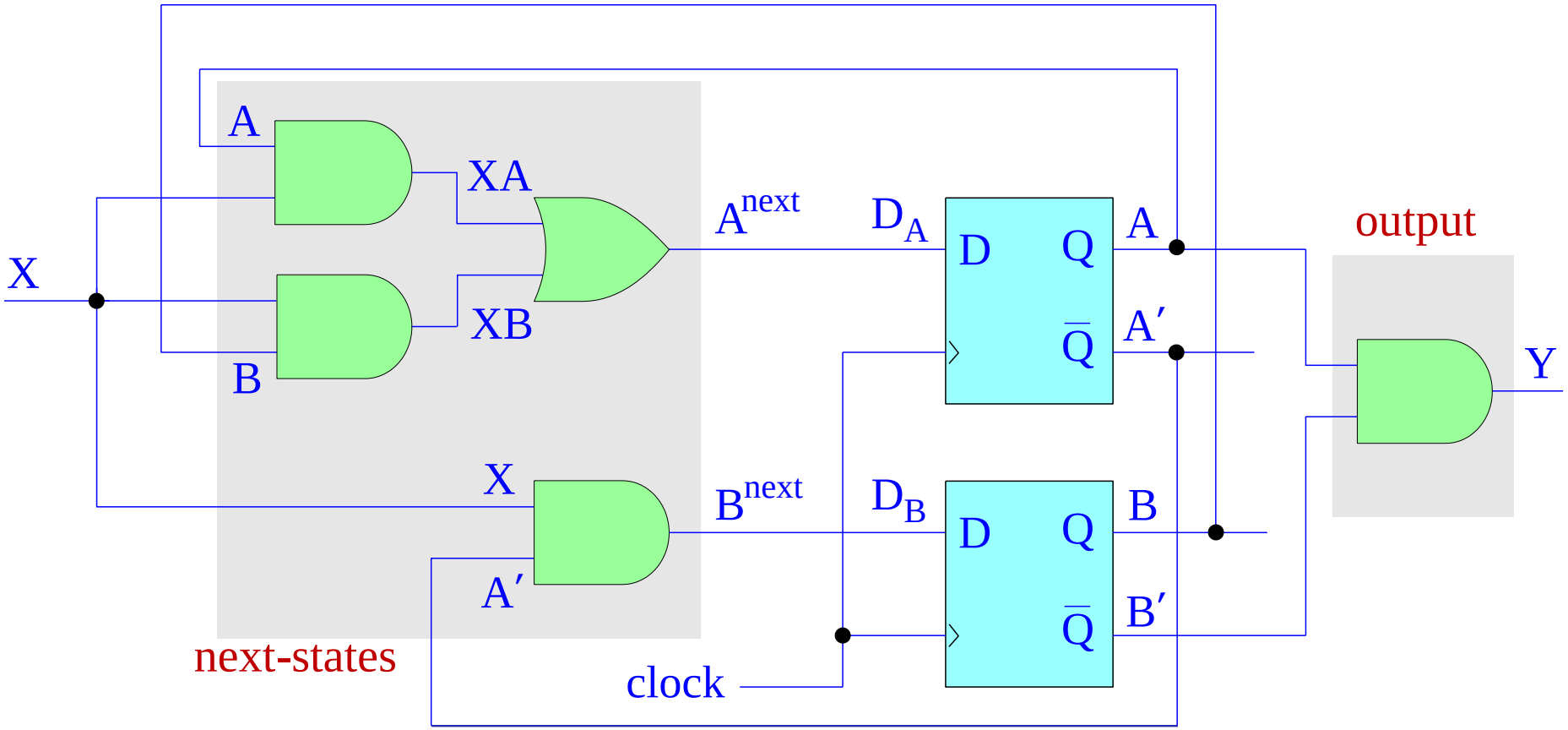
$D_B = B^{next} = X\,A'$

Moore FSM realization
Gray encoding

$D_A = A^{next} = X\,(A + B)$

$D_B = B^{next} = X\,A'$

$Y = A\,B'$ (Moore output)

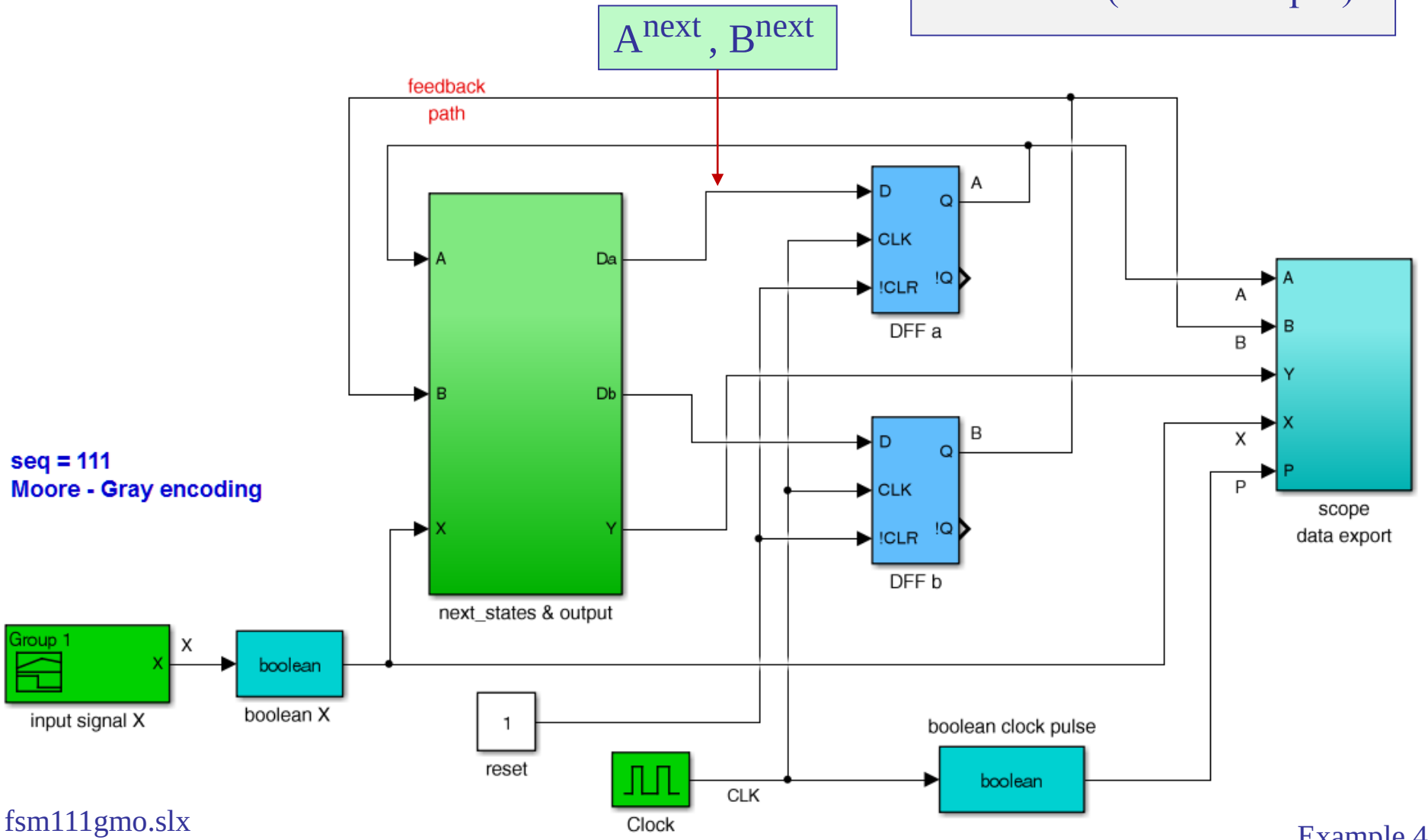


Simulink implementation and timing diagram

$$D_A = A^{next} = X\,(A + B)$$

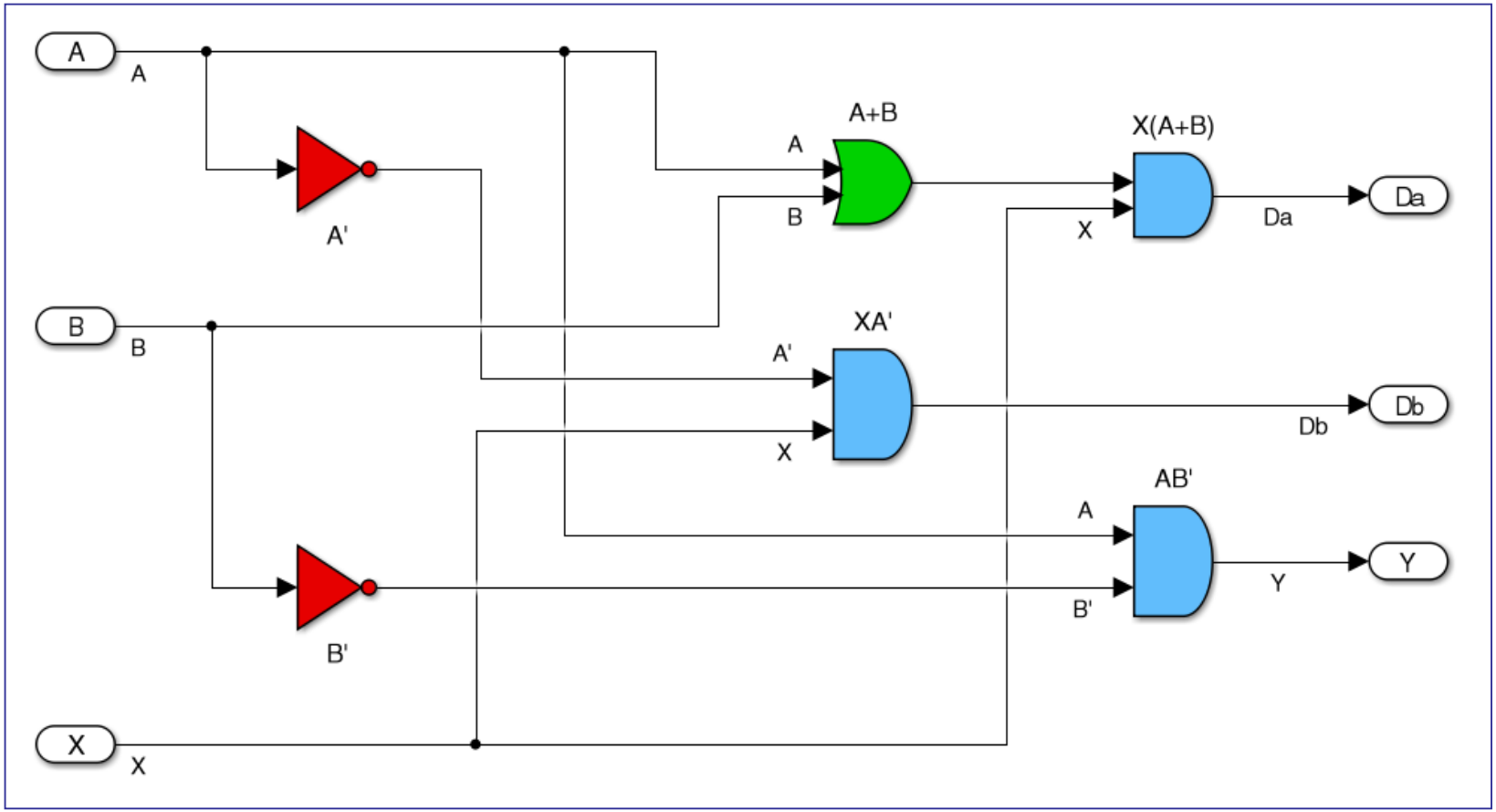
$$D_B = B^{next} = X\,A'$$

$$Y = A\,B' \quad \text{(Moore output)}$$



fsm111gmo.slx

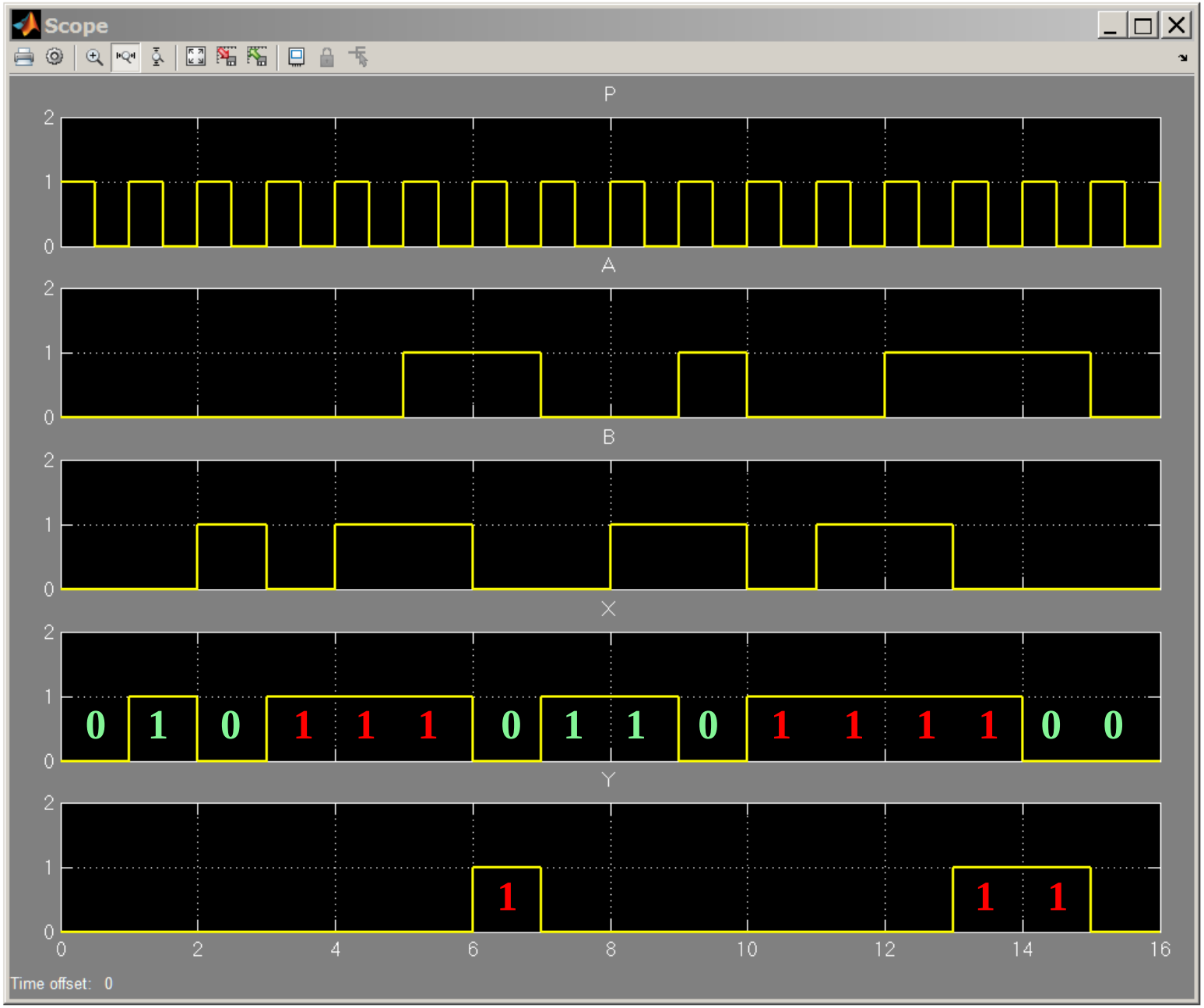
next-states & output sub-function



$$D_A = A^{next} = X\,(A + B)$$

$$D_B = B^{next} = X\,A'$$

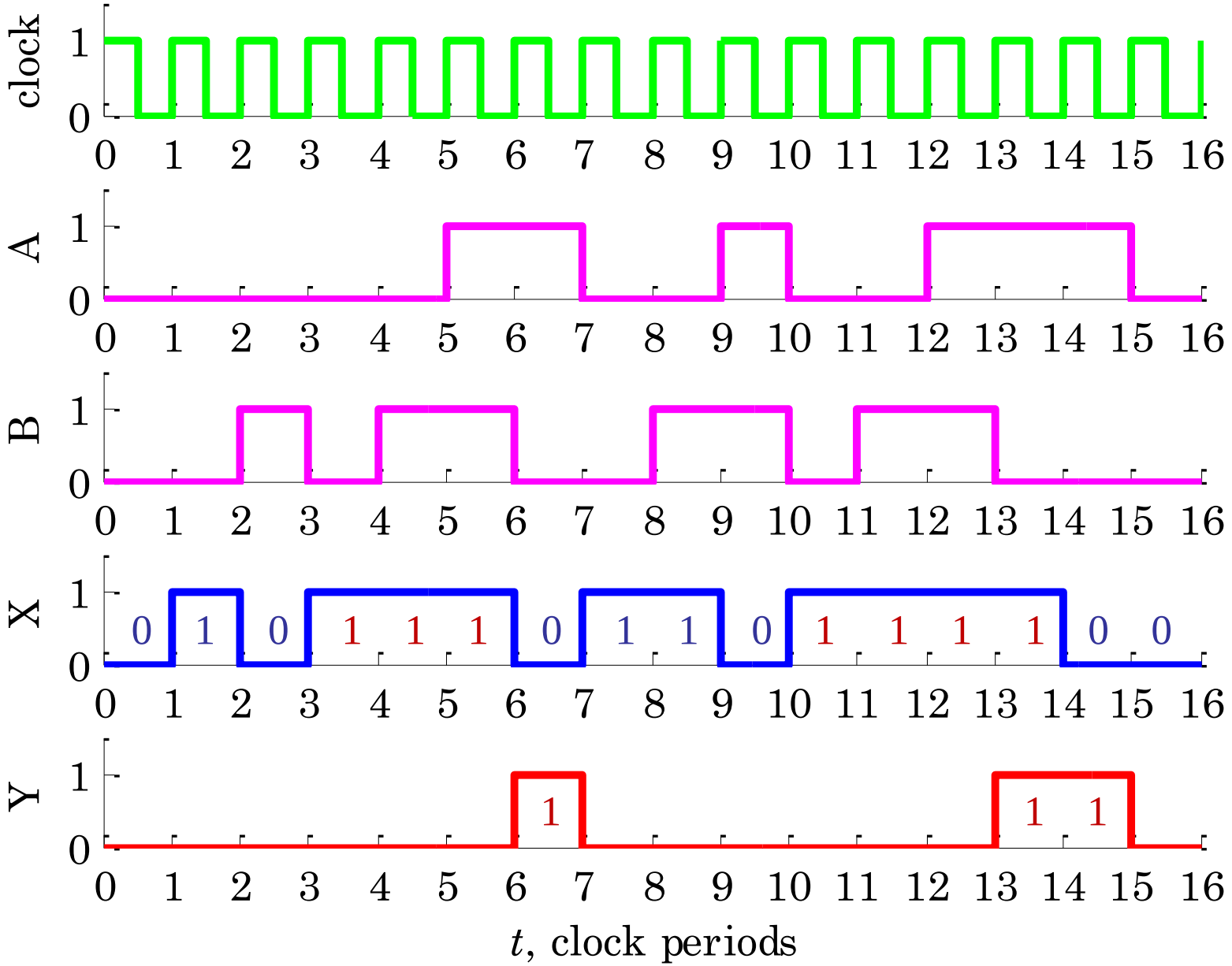
$$Y = A\,B' \quad \text{(Moore output)}$$



scope output

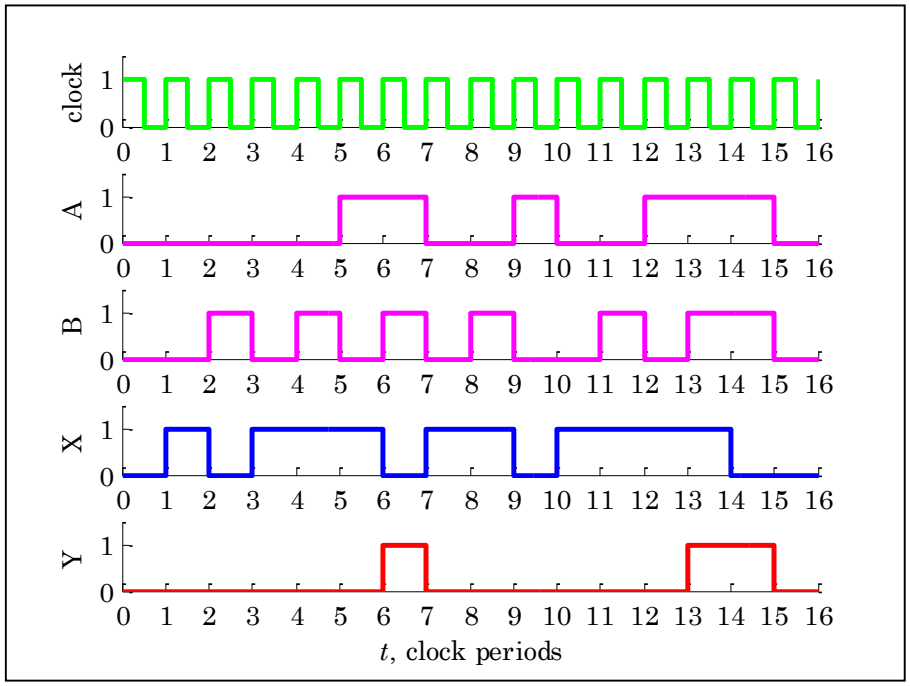
Example 4

Moore FSM – timing diagram



Moore FSM – timing diagram

binary encoding

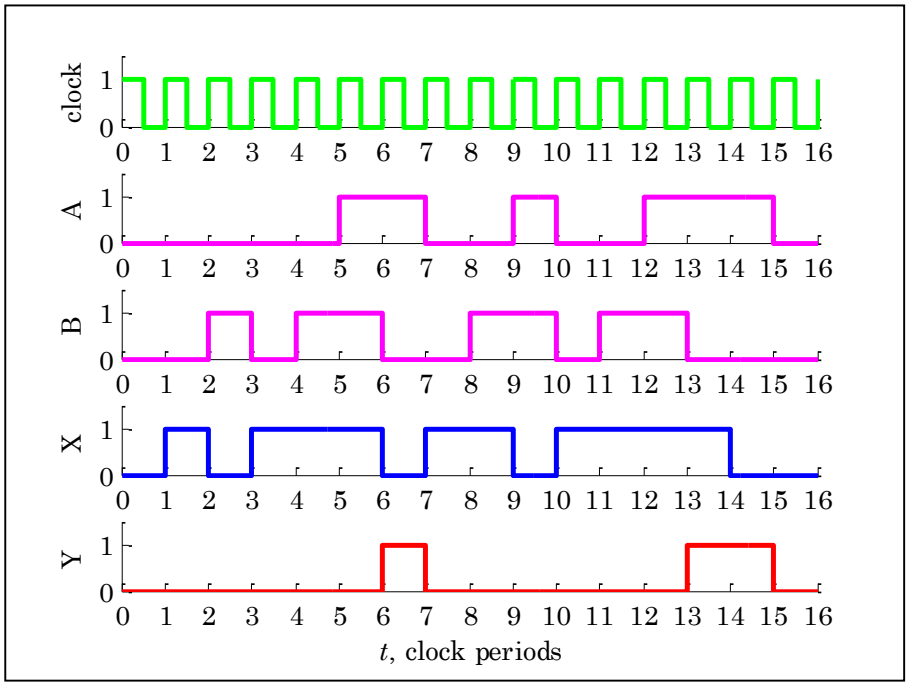


$D_A = A^{next} = X(A + B)$

$D_B = B^{next} = X(A + B')$

$Y = AB$ (Moore output)

Gray encoding



$D_A = A^{next} = X\,(A + B)$

$D_B = B^{next} = X\,A'$

$Y = AB'$ (Moore output)

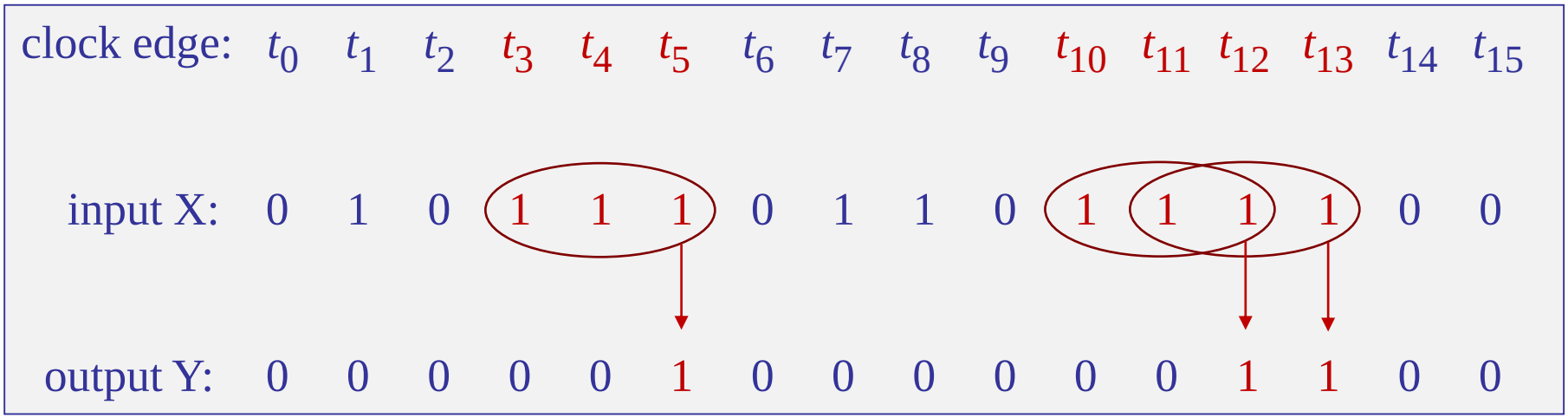
Example 5 – Sequence (111) recognizer – Mealy version. It is desired to design a Mealy FSM for the previous example. In the previous Moore version, the output was, Y=1, in the clock cycle that follows the observation of a third consecutive X=1.

In the present Mealy version, we require that Y=1 in the same clock cycle when the third occurrence of X=1 is observed.

This requirement makes Y dependent on both the input and the present state, hence, a Mealy machine.

(When a third X=1 is observed, the output Y is not instantaneously set equal to Y=1, but it requires a small delay, which we will ignore in the present discussion.)

example input signal and output



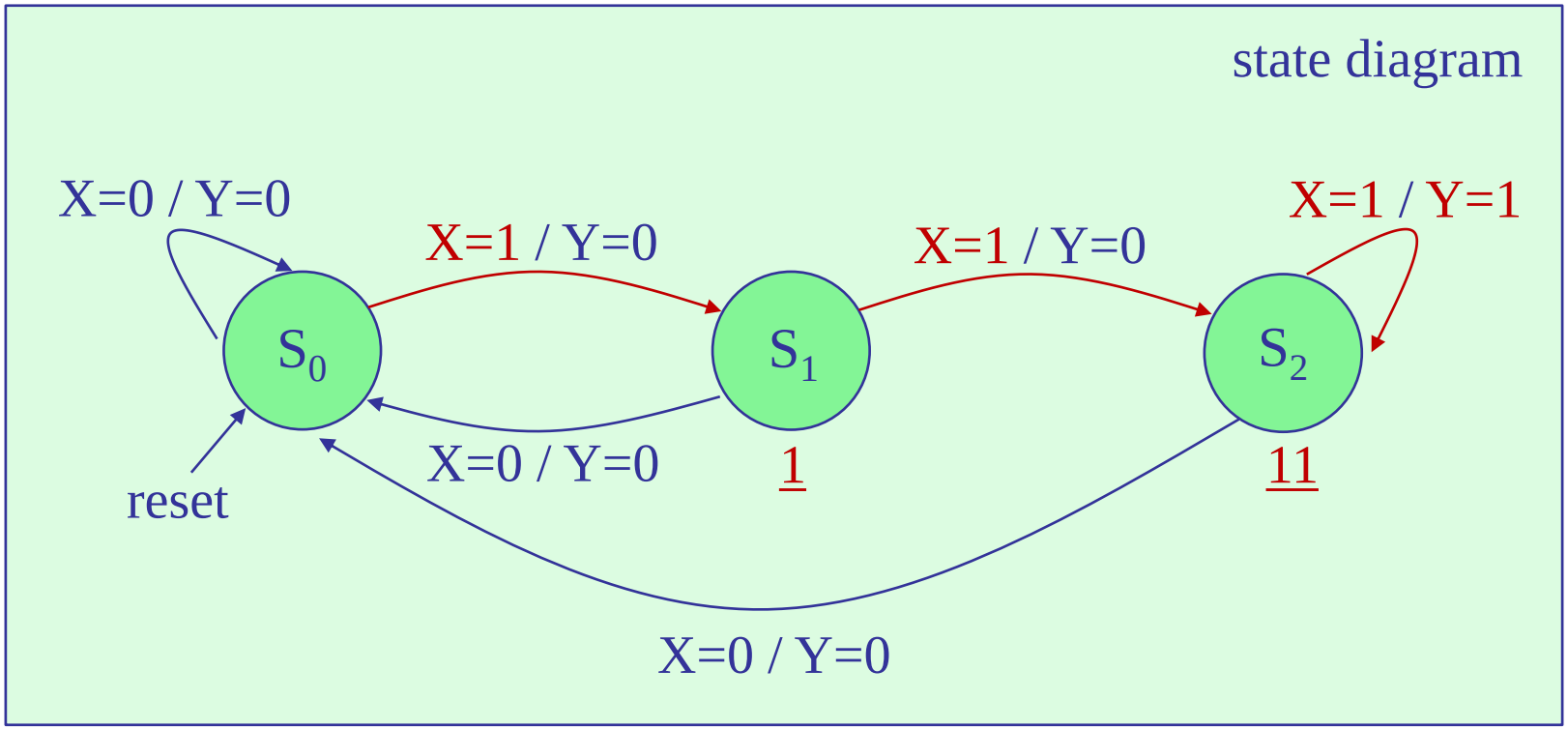
three consecutive 1s at t_3, t_4, t_5
will cause an output Y=1
at the present clock edge t_5

three consecutive 1s at t_{10}, t_{11}, t_{12}
will cause an output Y=1
at the present clock edge t_{12}

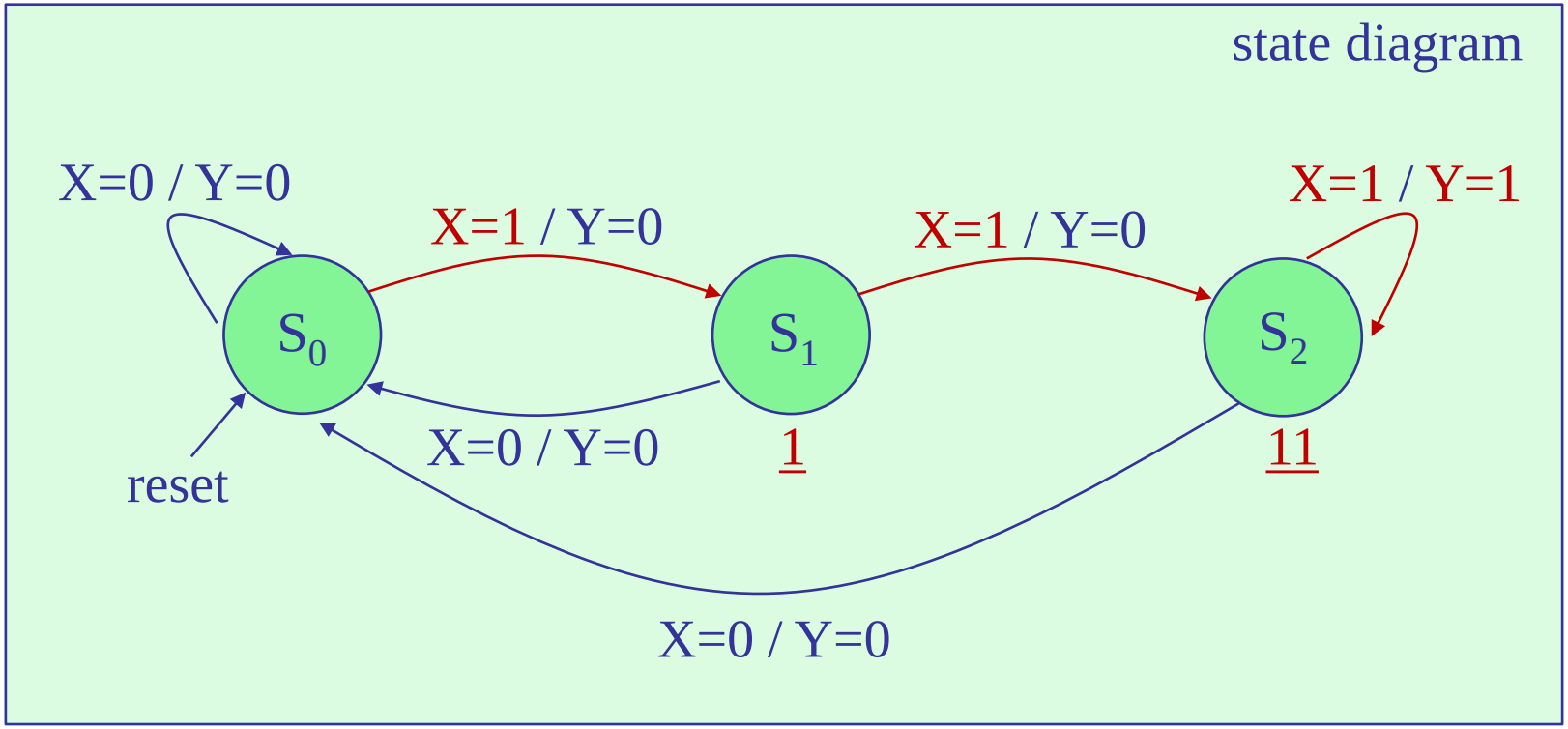
continuing with another
set of three 1s at t_{11}, t_{12}, t_{13}
will cause another output Y=1
at the last clock edge t_{13}

example input signal, state transition, and output

clock edge:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
input X:	0	1	0	1	1	1	0	1	1	0	1	1	1	1	0	0
state:	S_0	S_0	S_1	S_0	S_1	S_2	S_2	S_0	S_1	S_2	S_0	S_1	S_2	S_2	S_2	S_0
output Y:	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0



state transitions take place at the next active clock, but X,Y are at the present cycle



state table

Mealy FSM

present state	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_0	S_1	0	0
S_1	S_0	S_2	0	0
S_2	S_0	S_2	0	1

Mealy FSM – state table

compact form

present state	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_0	S_1	0	0
S_1	S_0	S_2	0	0
S_2	S_0	S_2	0	1

conventional form

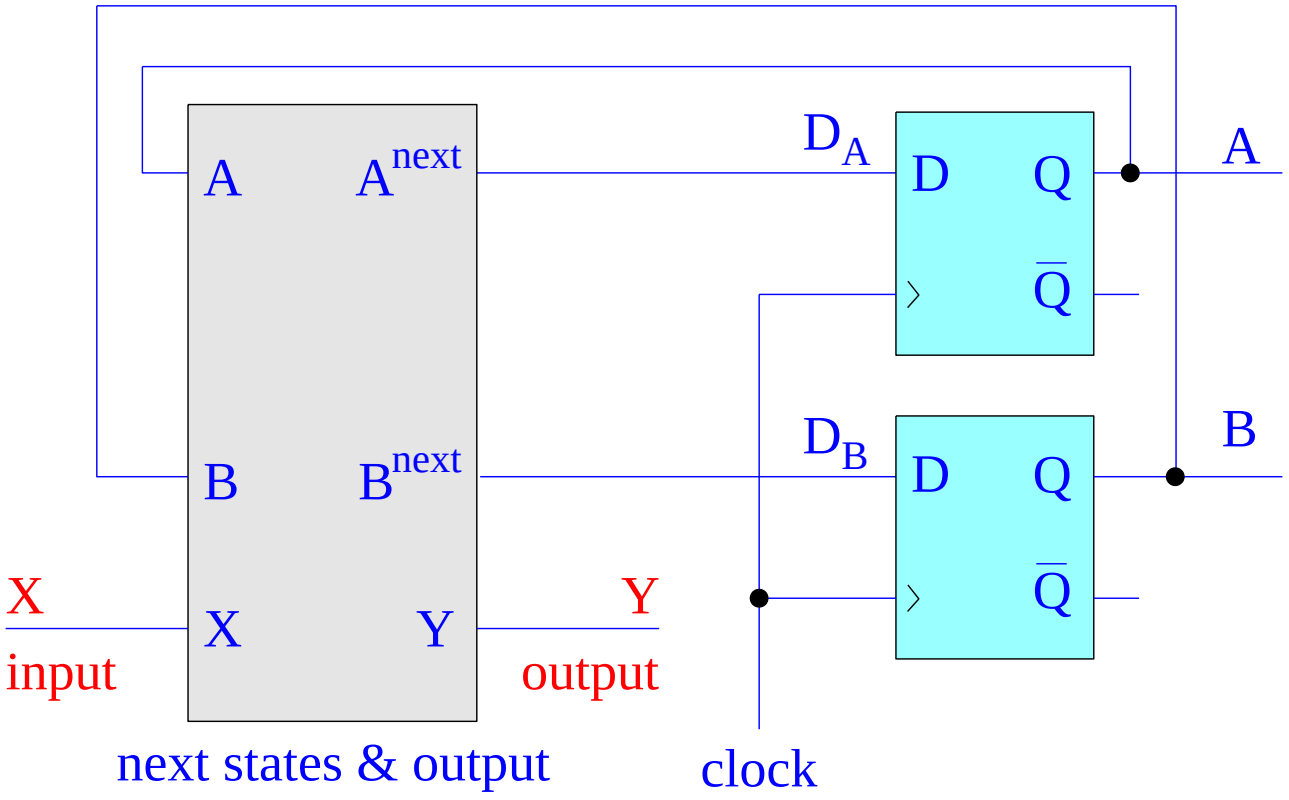
X	present state	next state	Y
0	S_0	S_0	0
0	S_1	S_0	0
0	S_2	S_0	0
1	S_0	S_1	0
1	S_1	S_2	0
1	S_2	S_2	1

State assignment. With N=3 states, we need,

n = ceiling($\log_2 N$) = ceiling($\log_2 3$) = ceiling(1.5850) = 2,

state variables, say, A,B, and two D flip-flops, with inputs, say, D_A, D_B.

The overall system will be as shown below.



State encoding. With two state variables, say, A,B, we have two options for state encoding, associating A,B with the three states, S_0, S_1, S_2,

(1) plain binary, or,
(2) Gray coding

We will use Gray coding with don't care entries for the unused pair, and re-write the state table in conventional form.

Mealy FSM - state table

X	present state	A	B	next state	A	B	Y
0	S_0	0	0	S_0	0	0	0
0	S_1	0	1	S_0	0	0	0
0	S_2	1	1	S_0	0	0	0
0	x	1	0	x	x	x	x
1	S_0	0	0	S_1	0	1	0
1	S_1	0	1	S_2	1	1	0
1	S_2	1	1	S_2	1	1	1
1	x	1	0	x	x	x	x

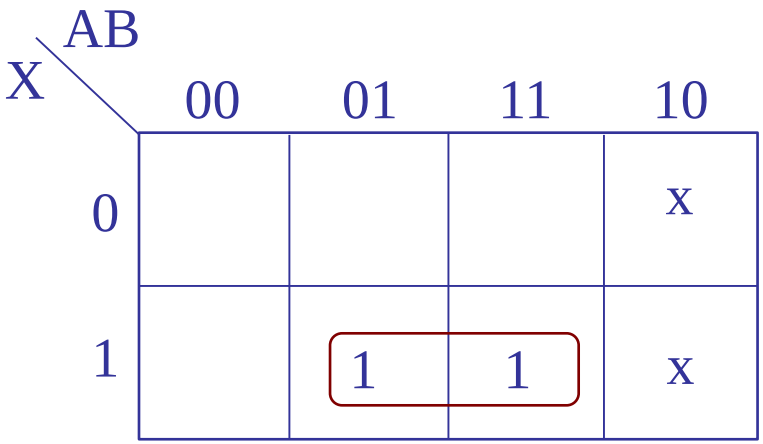
Gray encoding

	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	1

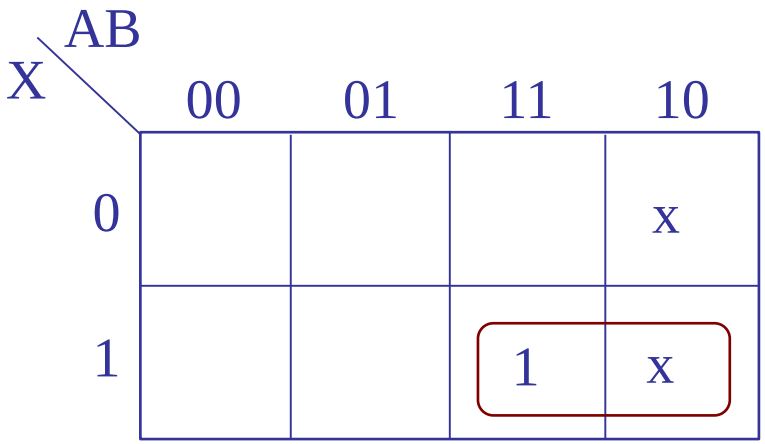
don't cares

Mealy FSM – state table

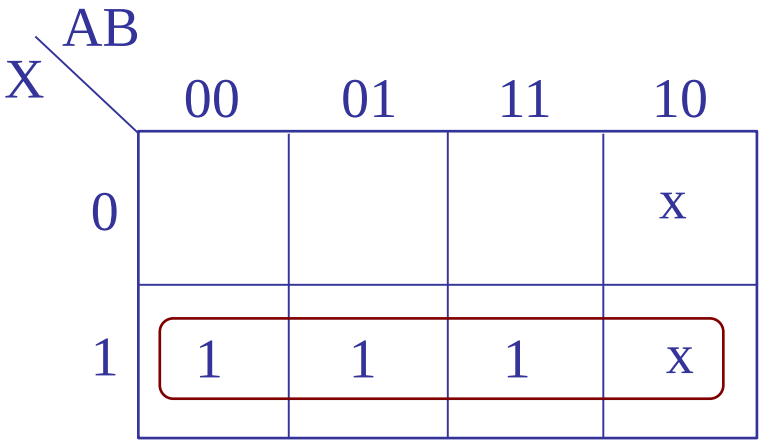
X	present A	present B	next A	next B	Y
0	0	0	0	0	0
0	0	1	0	0	0
0	1	1	0	0	0
0	1	0	x	x	x
1	0	0	0	1	0
1	0	1	1	1	0
1	1	1	1	1	1
1	1	0	x	x	x



$D_A = A^{next} = X\ B$



$Y = X\ A$ (Mealy output)



$D_B = B^{next} = X$

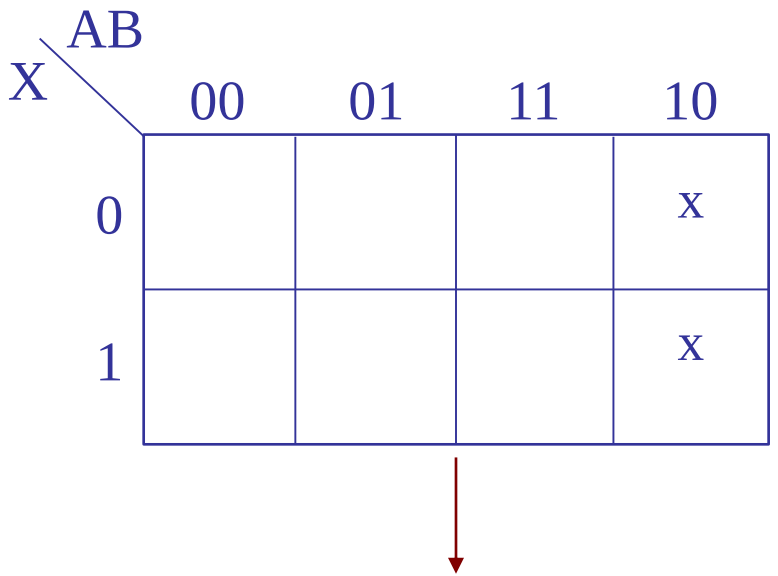
Mealy FSM – full state table

A B	X	next A B	Y
0 0	0	0 0	0
0 0	1	0 1	0
0 1	0	0 0	0
0 1	1	1 1	0
1 0	0	x0 x0	x0
1 0	1	x0 x1	x1
1 1	0	0 0	0
1 1	1	1 1	1

1,0 are unused states since they do not appear as next states, thus, they can be removed, or treated as don't cares

Gray encoding

	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	1



$D_A = A^{next} = XB$

$D_B = B^{next} = X$

$Y = XA$ (Mealy output)

Mealy FSM – full state table

A	B	X	next A	next B	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	1	0
1	0	0	x0	x0	x0
1	0	1	x0	x1	x1
1	1	0	0	0	0
1	1	1	1	1	1

```
[a,b,x] = a2d(0:7, 3);

da = x & b;
db = x;
y = x & a;

[a,b,x,da,db,y]

%   a    b    x    da   db   y
%   0    0    0    0    0    0
%   0    0    1    0    1    0

%   0    1    0    0    0    0
%   0    1    1    1    1    0

%   1    0    0    0    0    0
%   1    0    1    0    1    1

%   1    1    0    0    0    0
%   1    1    1    1    1    1
```

1,0 are unused states since they do not appear as next states, thus, they can be removed, or treated as don't care entries

$D_A = A^{next} = XB$

$D_B = B^{next} = X$

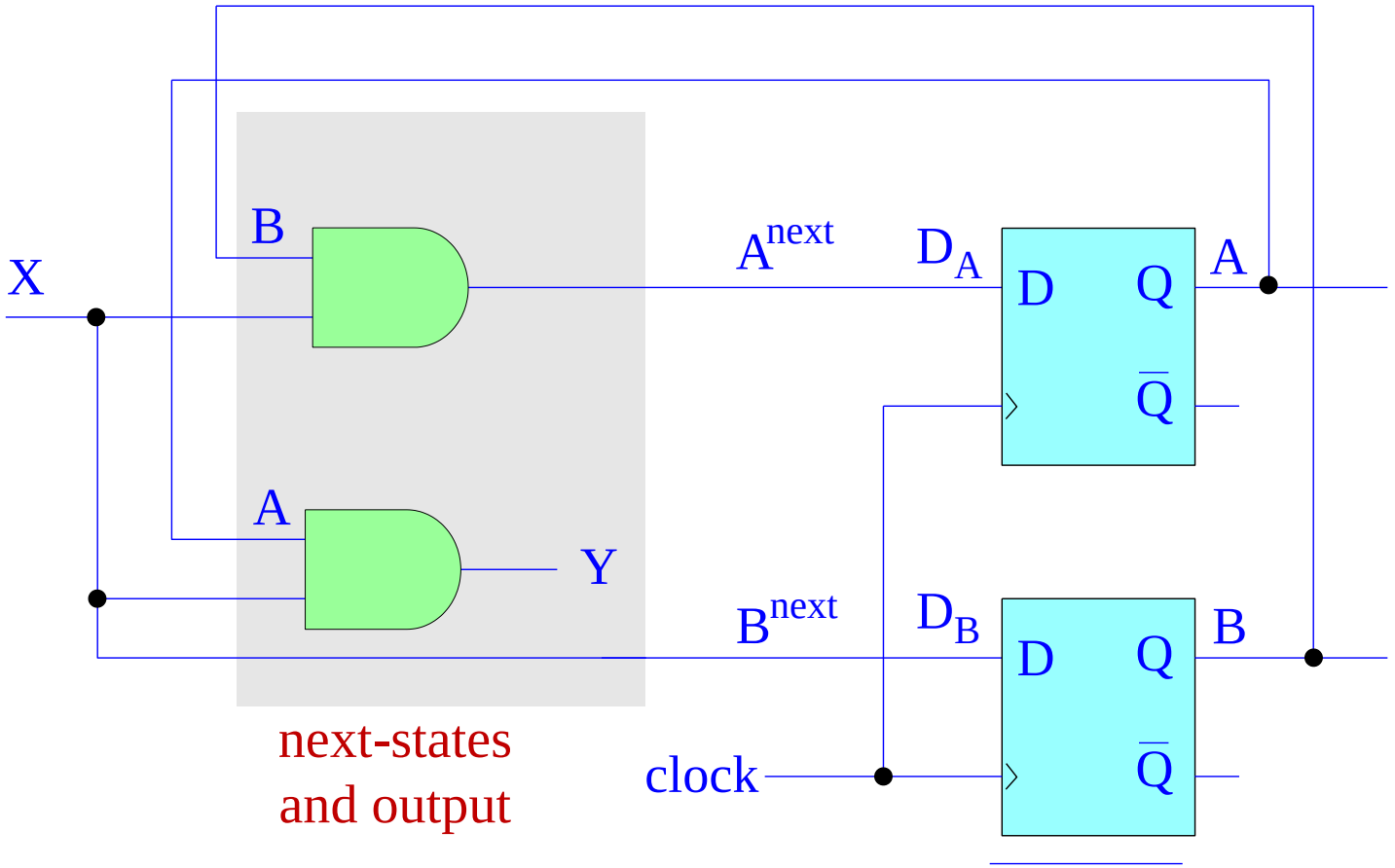
$Y = XA$ (Mealy output)

Mealy FSM realization
Gray encoding

$D_A = A^{next} = XB$

$D_B = B^{next} = X$

$Y = XA$ (Mealy output)

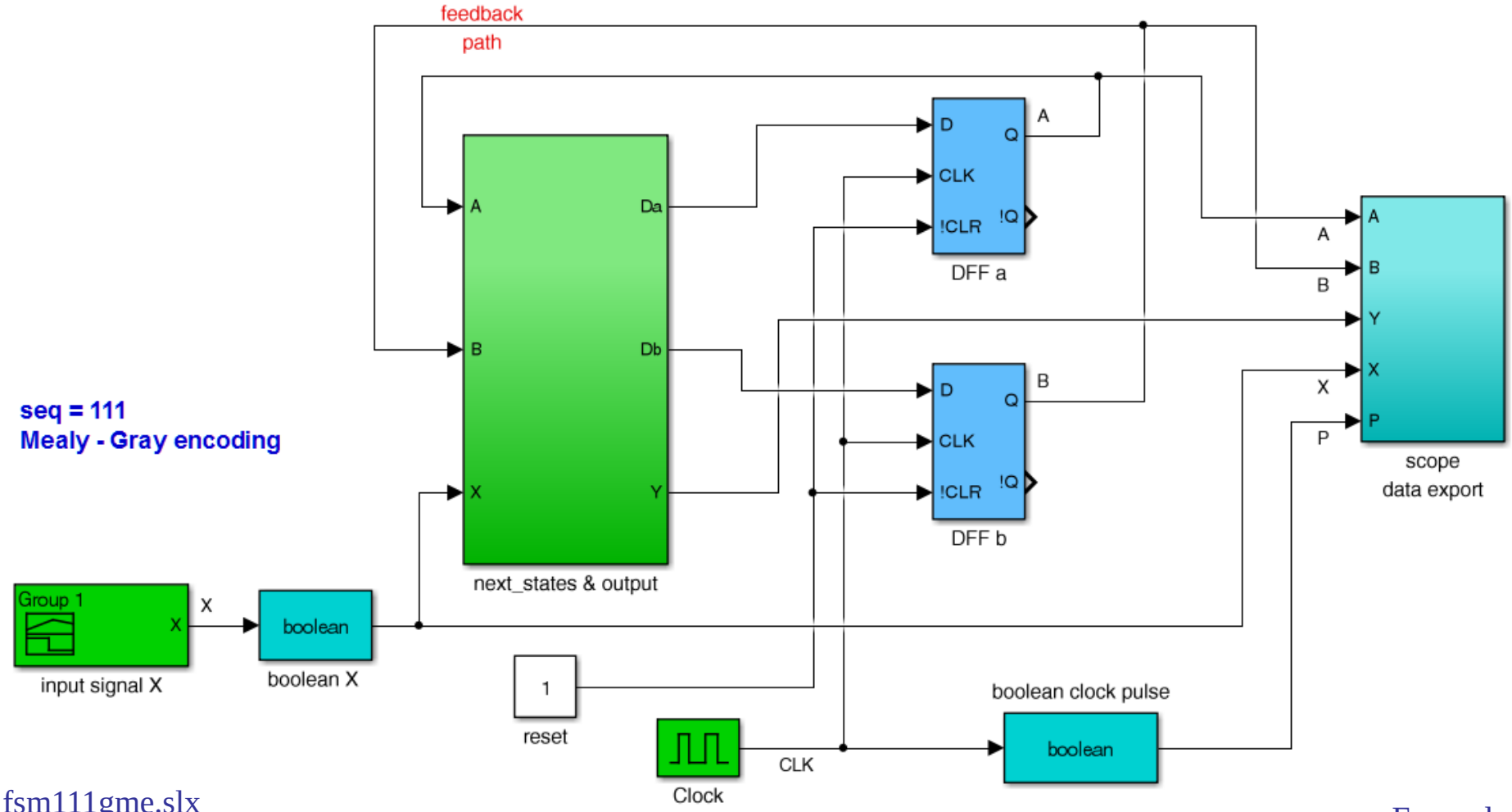


Simulink implementation and timing diagram

$D_A = A^{next} = XB$

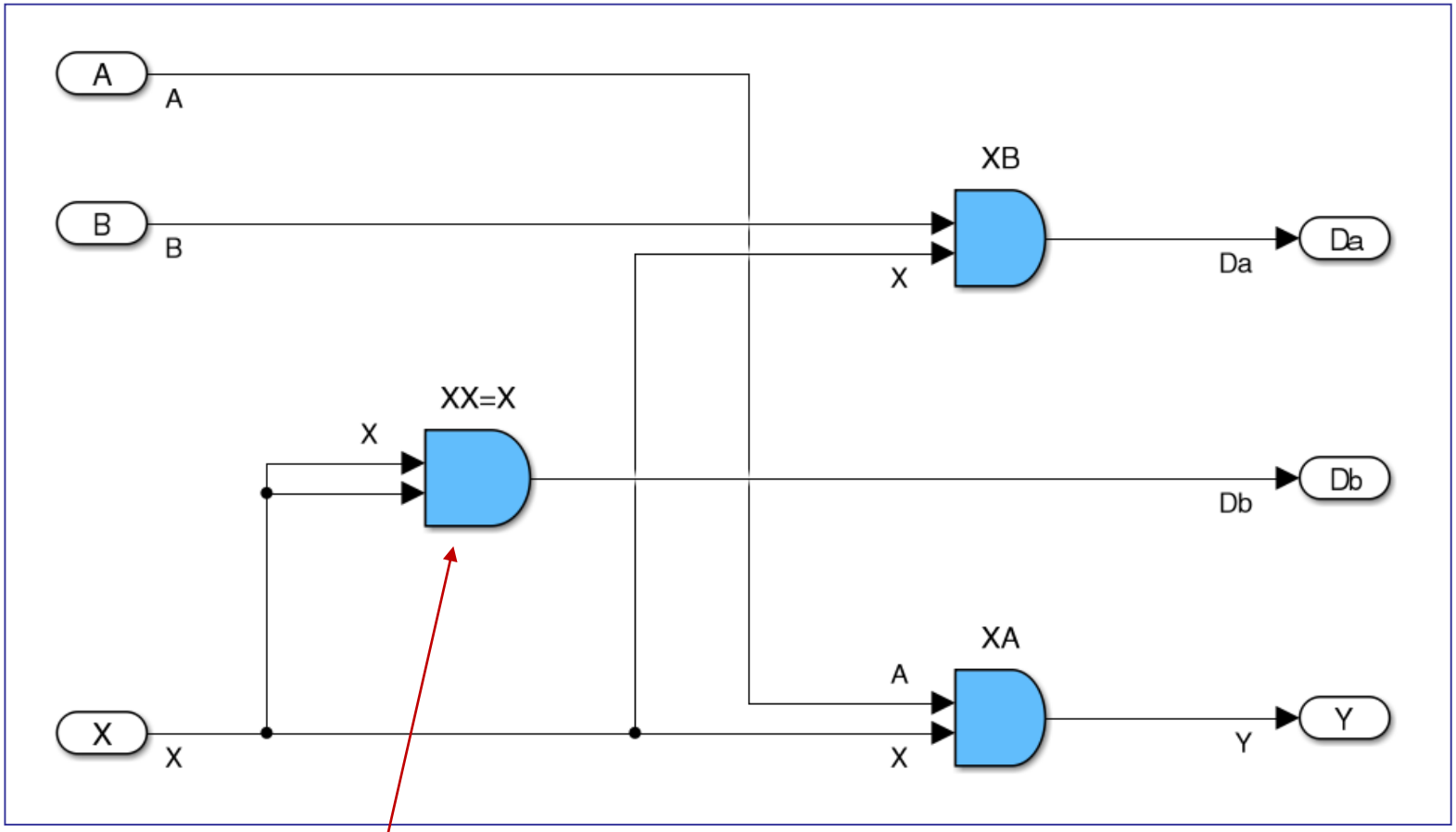
$D_B = B^{next} = X$

$Y = XA$ (Mealy output)



fsm111gme.slx

next-states & output sub-function

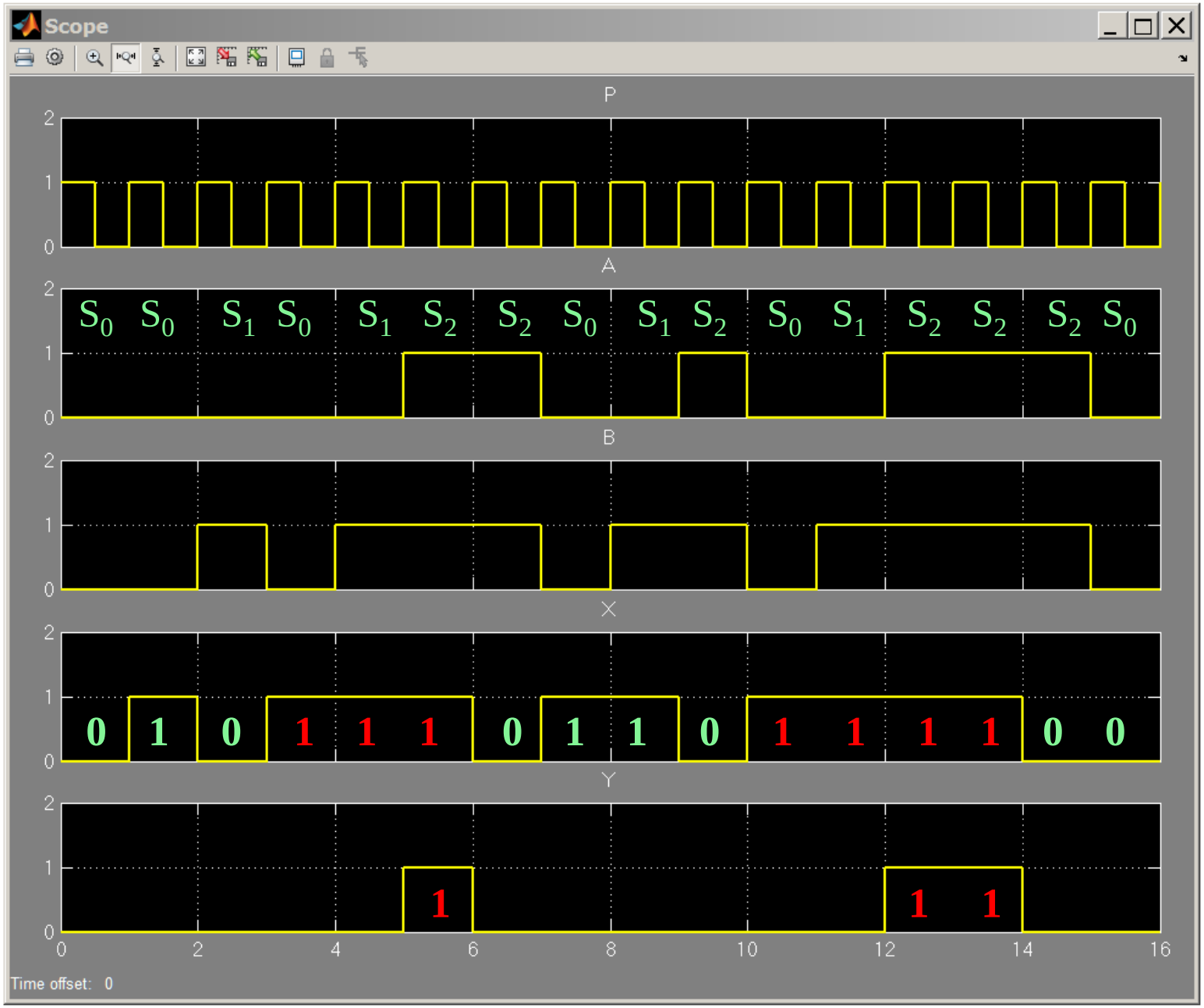


why is this necessary?

$D_A = A^{next} = XB$

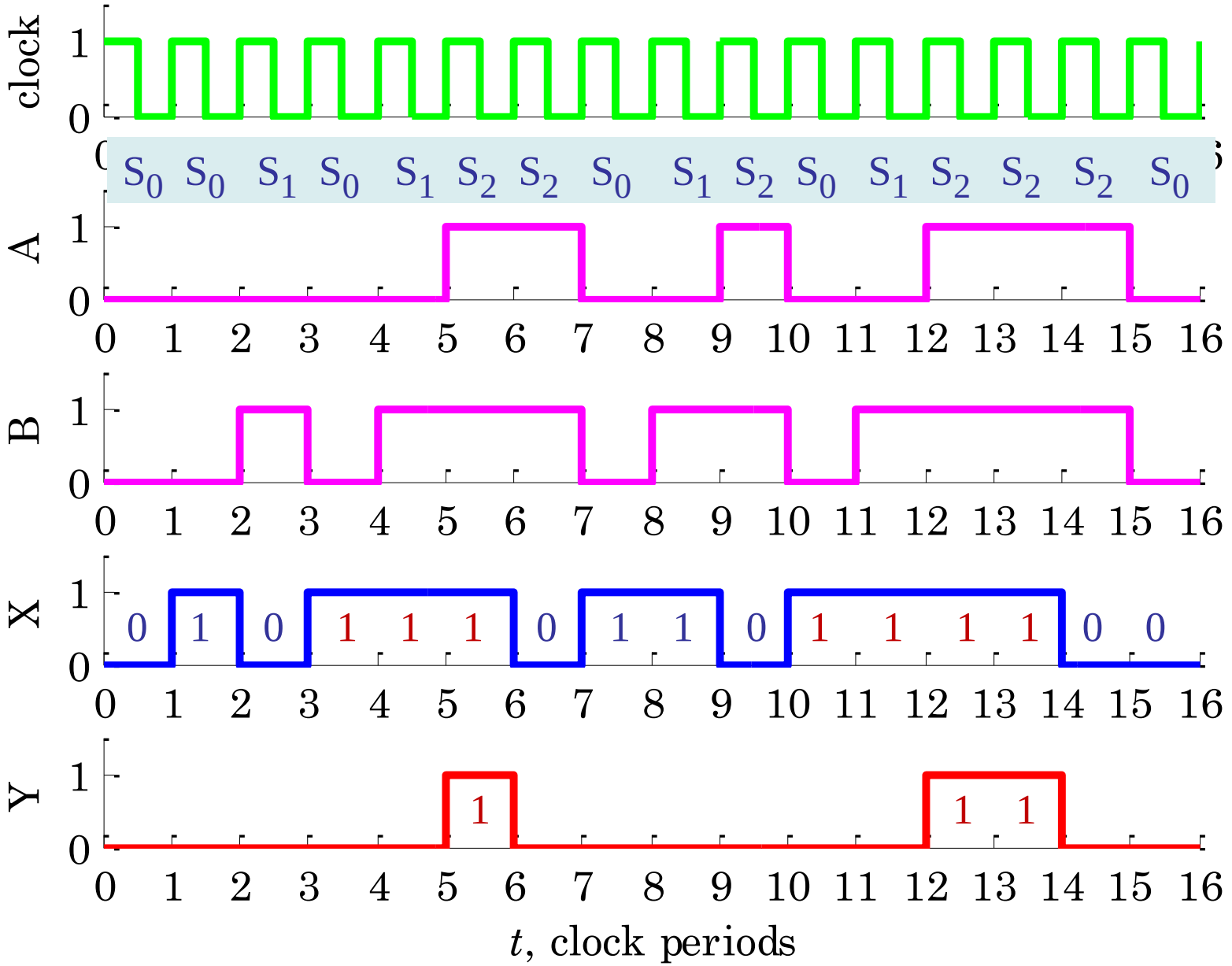
$D_B = B^{next} = X$

$Y = XA$ (Mealy output)



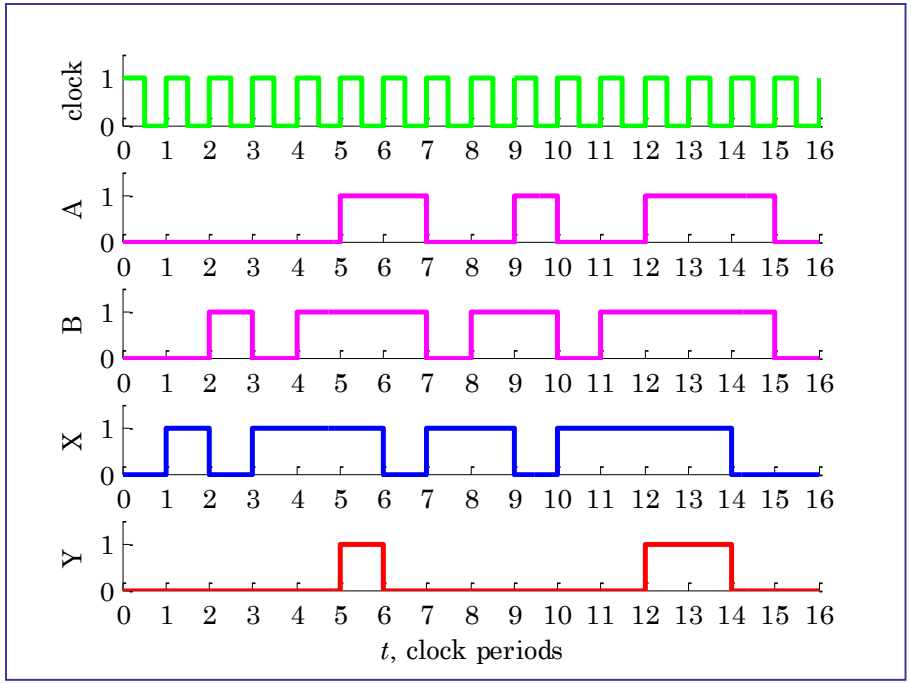
scope output

Mealy FSM – timing diagram



Mealy vs. Moore

Mealy - Gray encoding

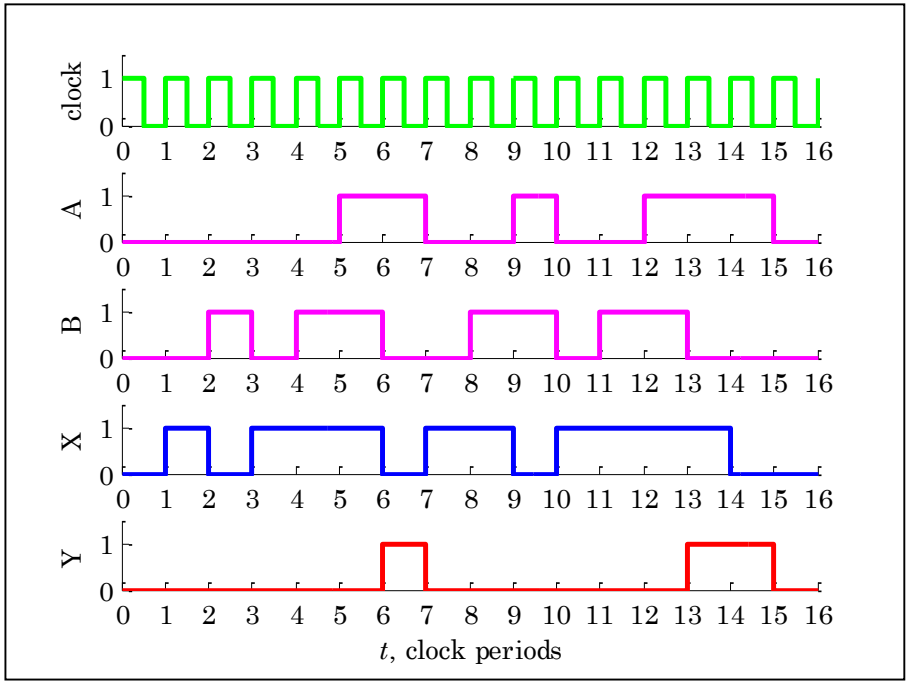


$D_A = A^{next} = XB$

$D_B = B^{next} = X$

$Y = XA$ (Mealy output)

Moore - Gray encoding



$D_A = A^{next} = X\,(A + B)$

$D_B = B^{next} = X\,A'$

$Y = A\,B'$ (Moore output)

Example 6 – Another sequence recognizer – Mealy version. It is desired to design a Mealy FSM circuit to detect the particular sequence of consecutive bits, 1101, in an incoming input stream X of zeros and ones, measured at the positive edges of the clock signal.

At each time instant, upon detecting the pattern 110 in the previous three inputs, and the current input is X=1, then, the FSM should produce an output, Y = 1, otherwise, it should produce, Y = 0.

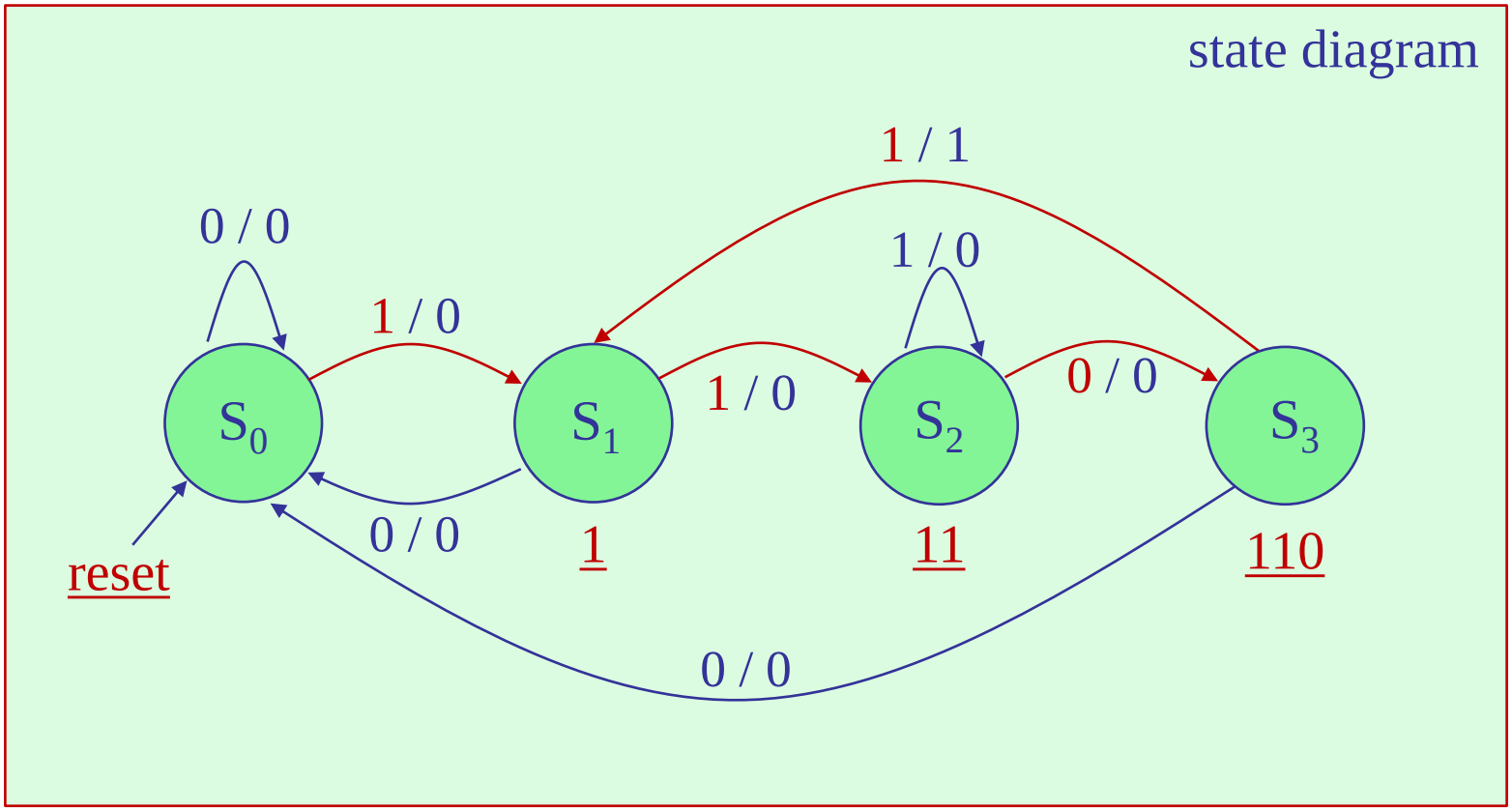
We may assume also that there is a Reset input that initially resets all states to zero.

[cf. Mano, Kime, Martin]

The number of required states can be determined by imagining the following successive sequence of occurrences of input bits:

$$\underline{\text{reset}} \rightarrow \underline{1} \rightarrow \underline{11} \rightarrow \underline{110}$$

Thus, we may use 4 states, denoted symbolically as, S_0, S_1, S_2. S_3.



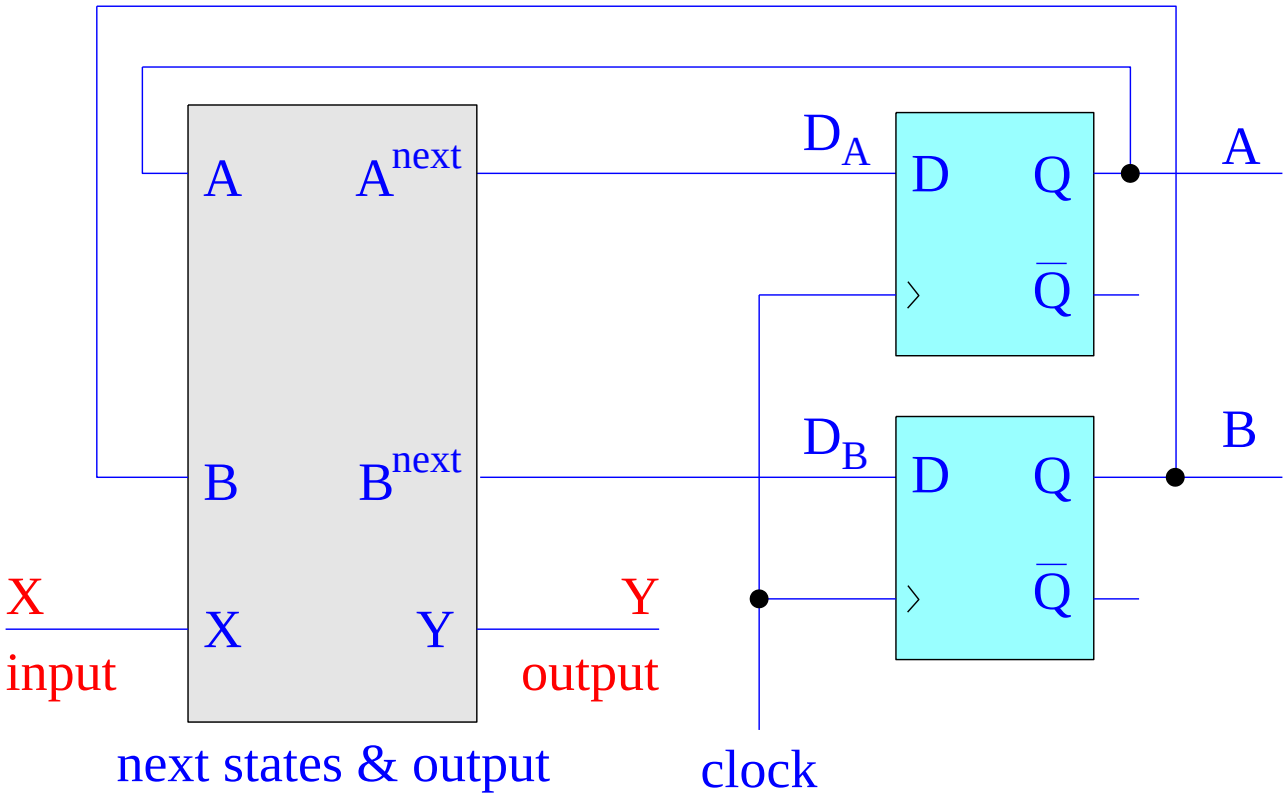
state table

Mealy FSM

present	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_0	S_1	0	0
S_1	S_0	S_2	0	0
S_2	S_3	S_2	0	0
S_3	S_0	S_1	0	1

State assignment. With N=4 states, we need, n = ceiling($\log_2 N$) = 2, state variables, say, A,B, and two D flip-flops, with inputs, say, D_A, D_B.

The overall system will be as shown below.



State encoding. With two state variables, say, A,B, we have two options for state encoding, associating A,B with the 4 states, S_0, S_1, S_2, S_3,

(1) plain binary, or,
(2) Gray encoding

We will start with Gray encoding and rewrite the state table in terms of the state variables A, B.

Mealy FSM - state table

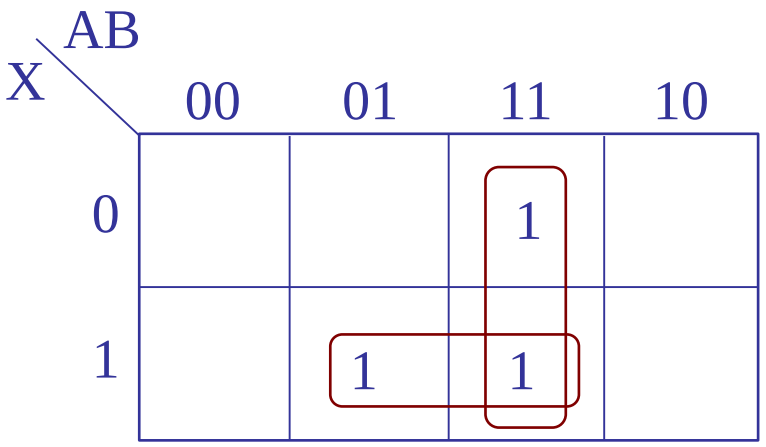
present states	next states A^{next} B^{next}		output Y	
A B	X=0	X=1	X=0	X=1
S_0 0 0	S_0 0 0	S_1 0 1	0	0
S_1 0 1	S_0 0 0	S_2 1 1	0	0
S_2 1 1	S_3 1 0	S_2 1 1	0	0
S_3 1 0	S_0 0 0	S_1 0 1	0	1

Gray encoding

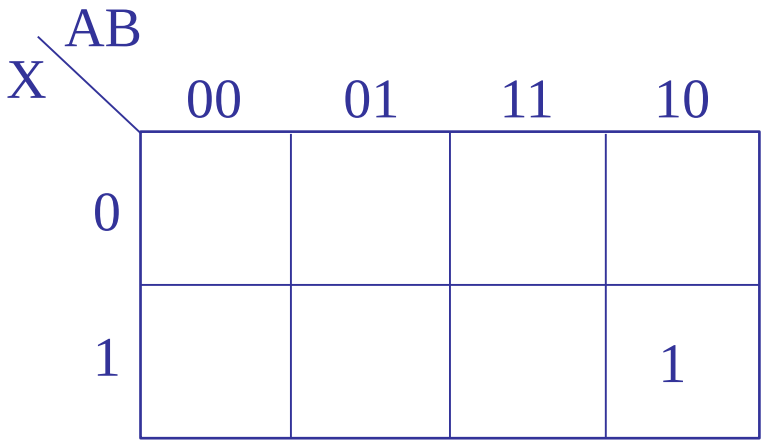
	A B
S_0 ≡	0 0
S_1 ≡	0 1
S_2 ≡	1 1
S_3 ≡	1 0

Mealy FSM – state table

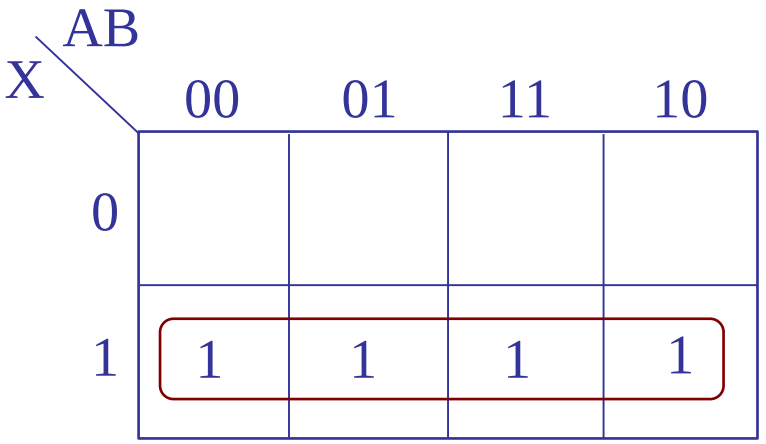
present states	next states A^{next} B^{next}		output Y	
A B	X=0	X=1	X=0	X=1
S_0 0 0	S_0 0 0	S_1 0 1	0	0
S_1 0 1	S_0 0 0	S_2 1 1	0	0
S_2 1 1	S_3 1 0	S_2 1 1	0	0
S_3 1 0	S_0 0 0	S_1 0 1	0	1



$D_A = A^{next} = X\,B + A\,B$



$Y = X\,A\,B'$



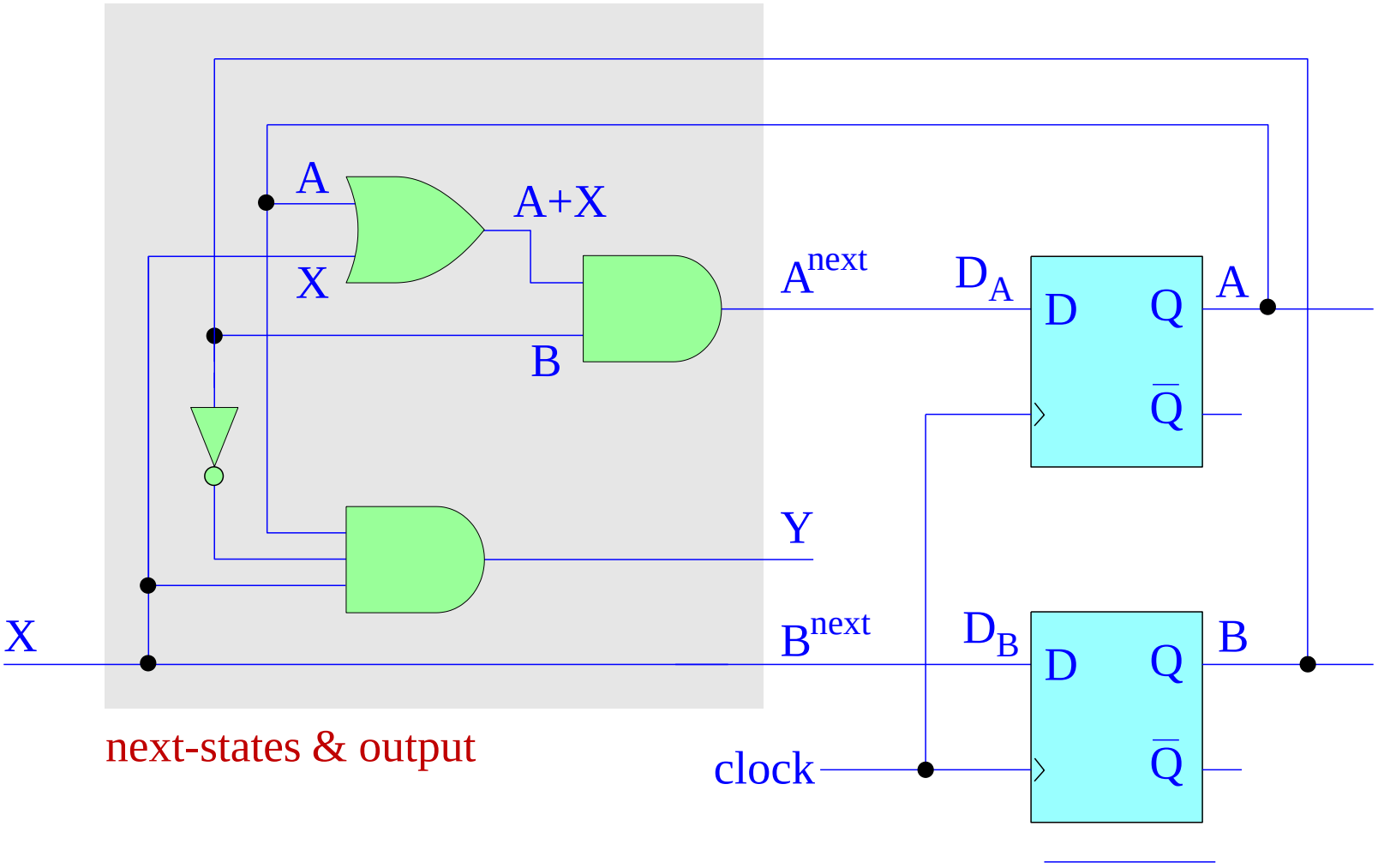
$D_B = B^{next} = X$

Block diagram realization

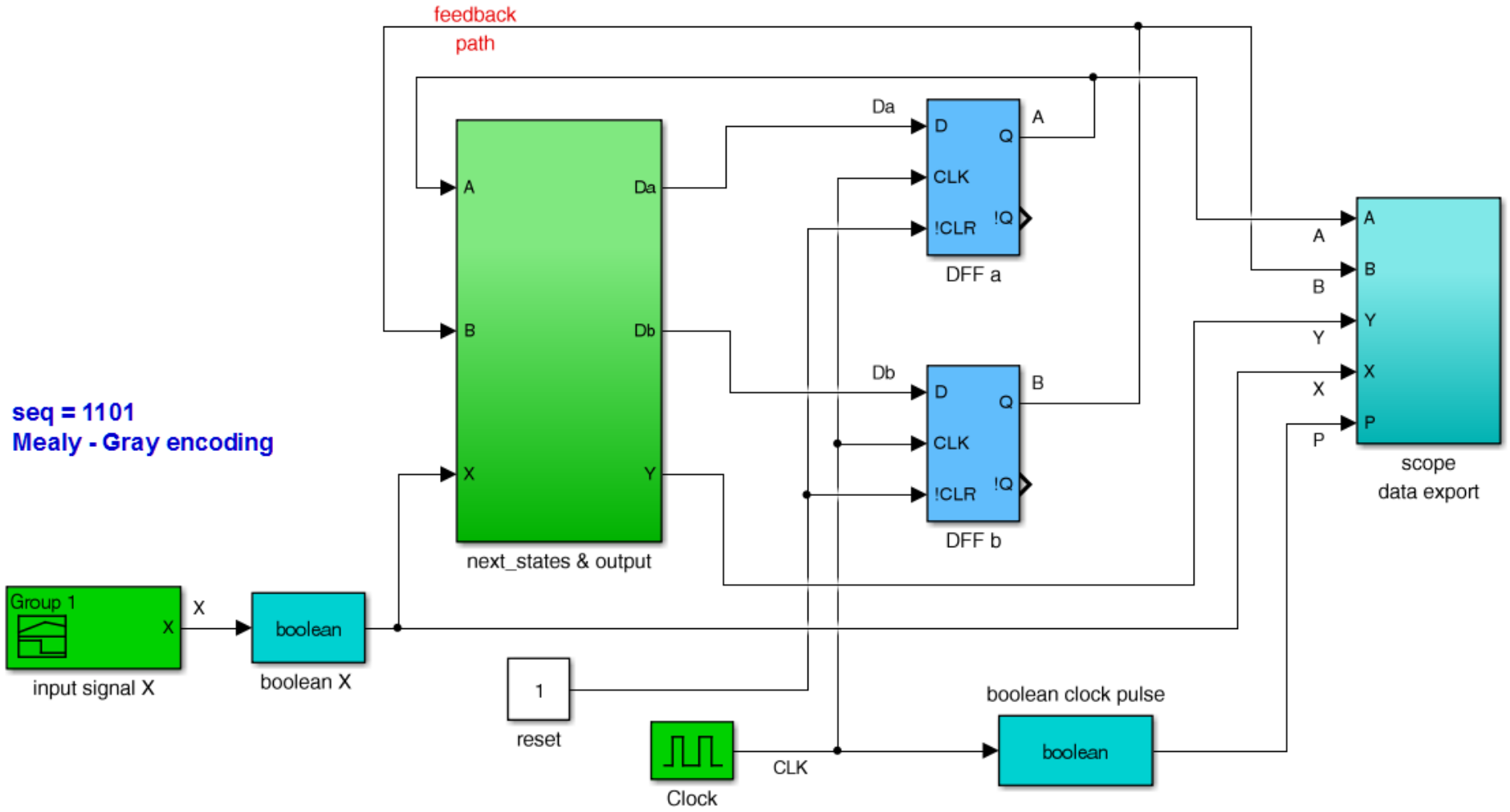
$D_A = A^{next} = (X + A)\,B$

$D_B = B^{next} = X$

$Y = X\,A\,B'$



Simulink implementation



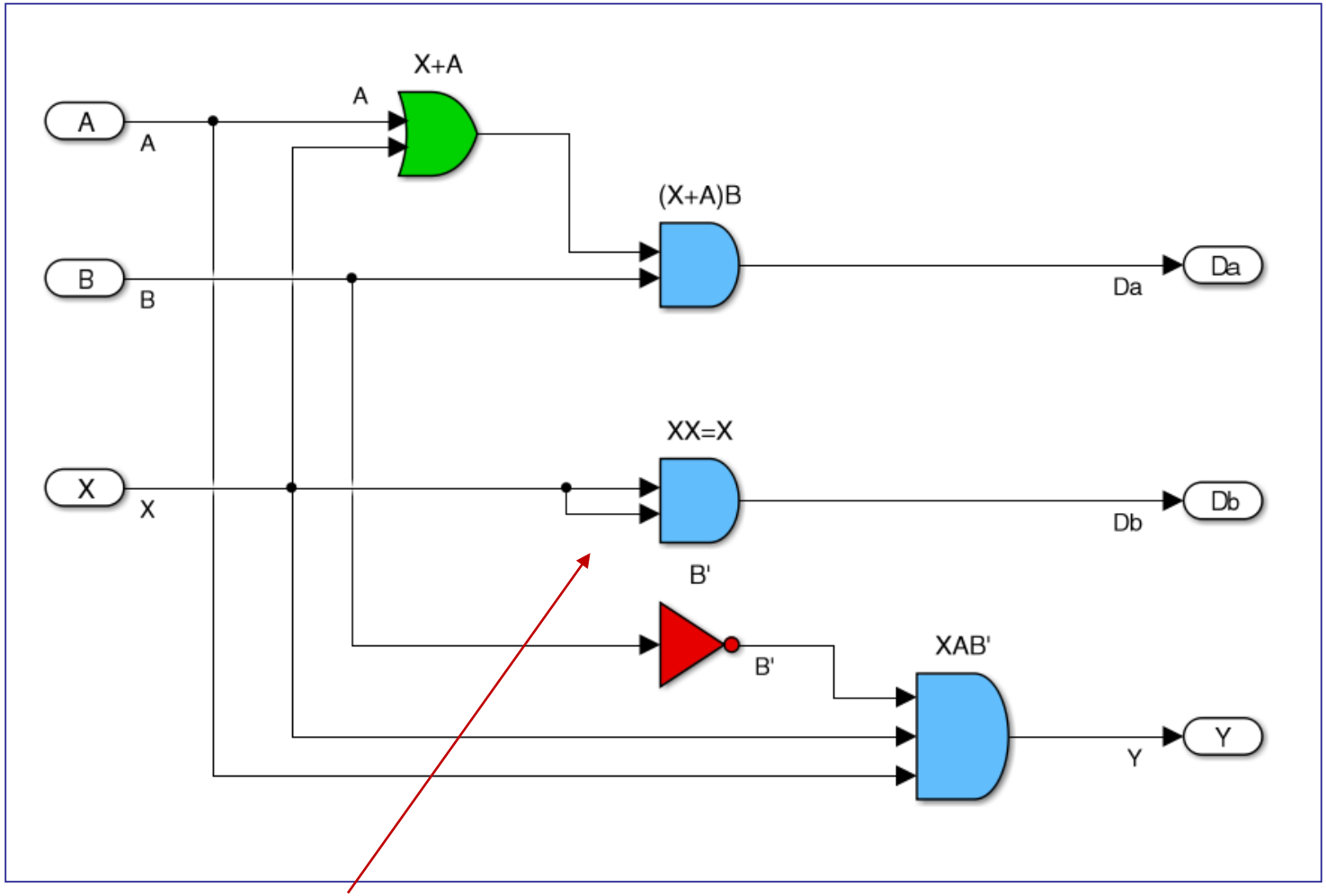
fsm1101gme.slx

next-state & output sub-function

$D_A = A^{next} = (X + A)\,B$

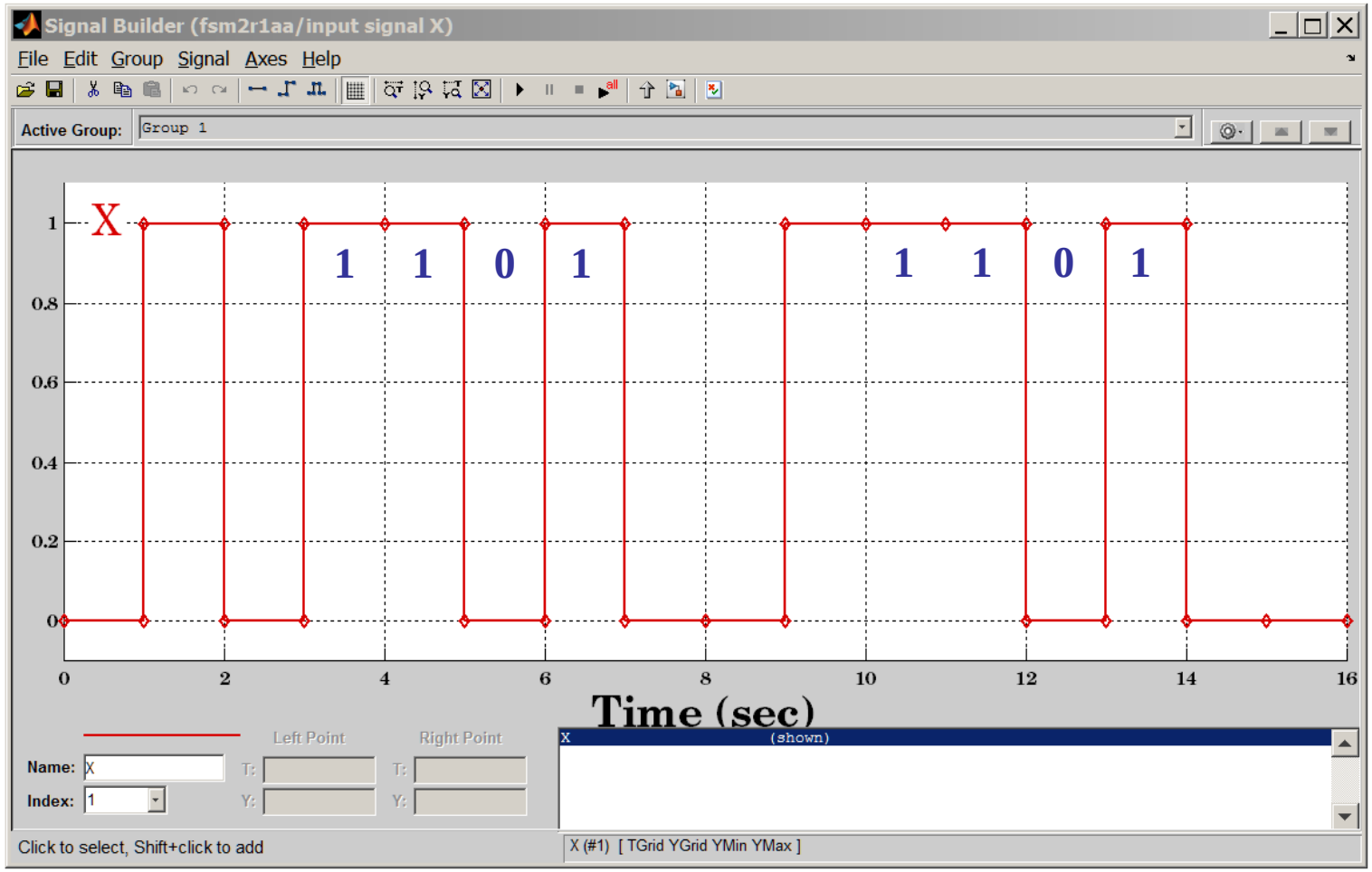
$D_B = B^{next} = X$

$Y = X\,A\,B'$



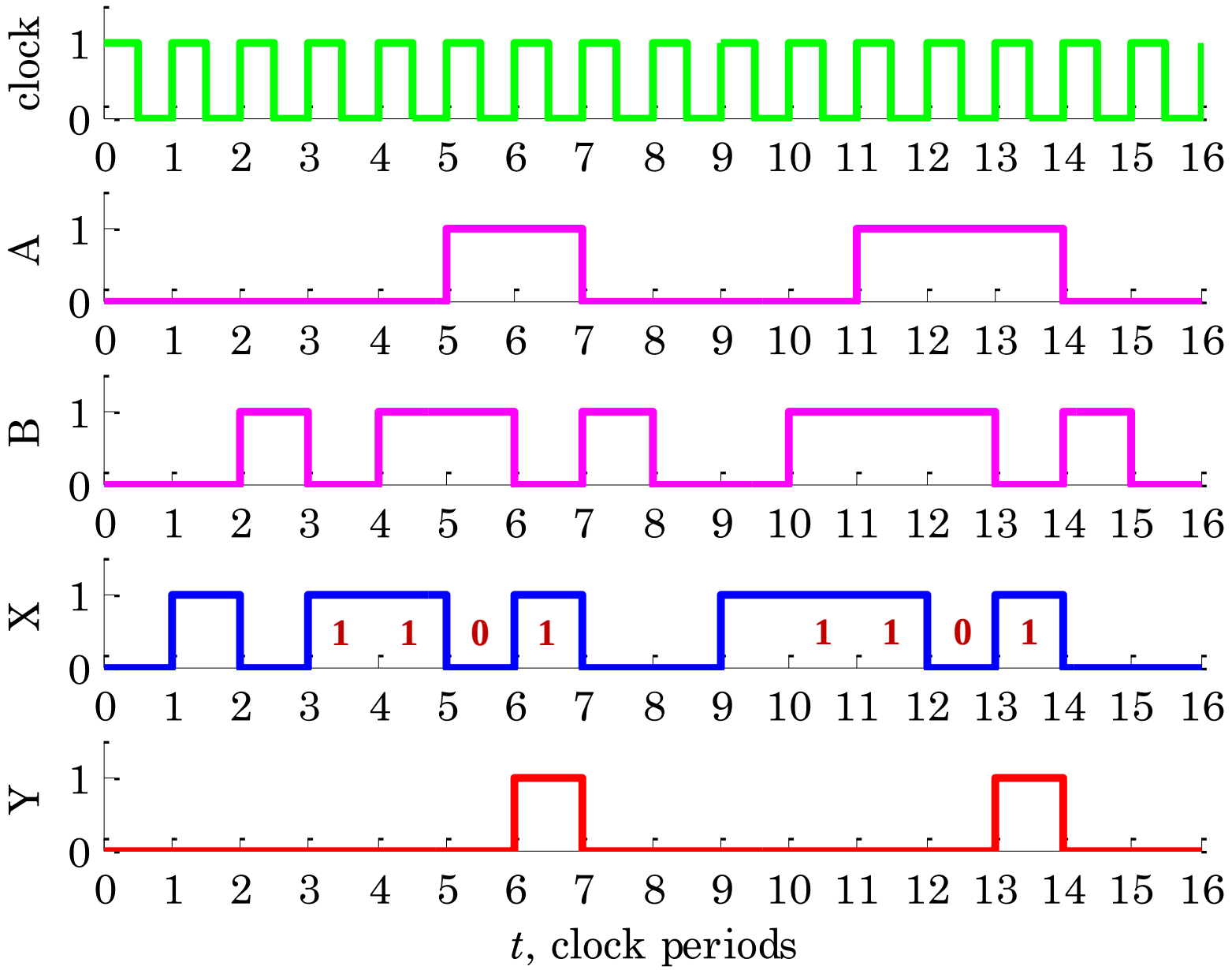
why is this necessary?

signal builder – input X



Example 6

sequence recognizer – 1101 – Mealy version – Gray encoding



State encoding. With plain binary encoding, we obtain a more complex realization, requiring more gates.

Mealy FSM – state table – binary encoding

present states		next states A^{next} B^{next}		output Y	
A B		X=0	X=1	X=0	X=1
S_0	0 0	S_0 0 0	S_1 0 1	0	0
S_1	0 1	S_0 0 0	S_2 1 0	0	0
S_2	1 0	S_3 1 1	S_2 1 0	0	0
S_3	1 1	S_0 0 0	S_1 0 1	0	1

$D_A = A^{next} = A\,B' + X\,A'\,B$

$D_B = B^{next} = X'\,A\,B' + X\,A\,B + X\,A'\,B'$

$Y = X\,A\,B$

binary encoding

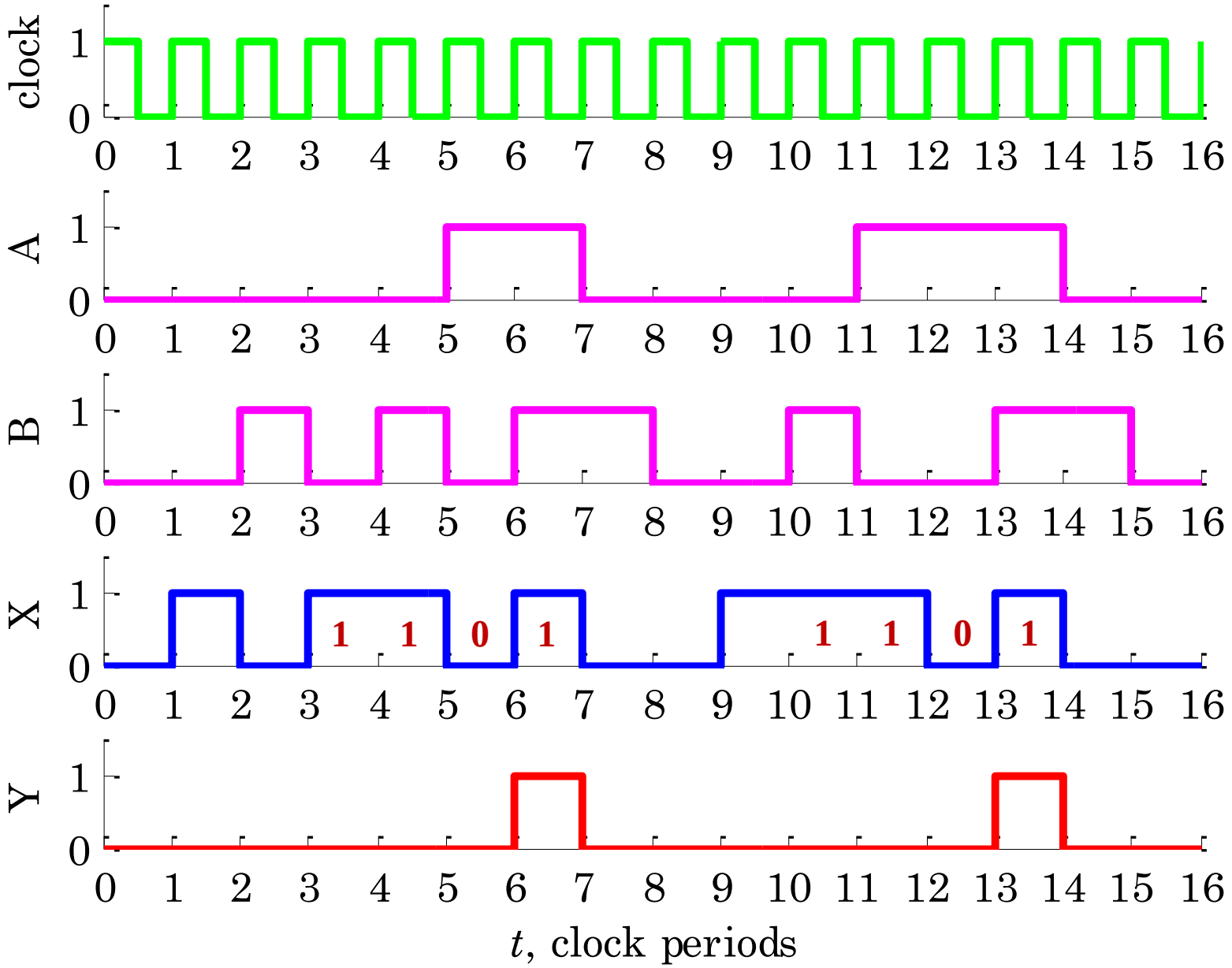
$D_A = A^{next} = (X + A)\,B$

$D_B = B^{next} = X$

$Y = X\,A\,B'$

Gray encoding

sequence recognizer – 1101 – Mealy version – binary encoding



Example 7 – Sequence recognizer – 1101 – Moore version. Here, we summarize without derivations the Moore version of the previous example.

The number of required states is five, corresponding to the sequence of occurrences of input bits:

$$\underline{\text{reset}} \rightarrow \underline{1} \rightarrow \underline{11} \rightarrow \underline{110} \rightarrow \underline{1101}$$

We may denote the states symbolically as, S_0, S_1, S_2, S_3, S_4. Thus, we will need three flip-flops, A, B, C, and may use (partial) Gray encoding.

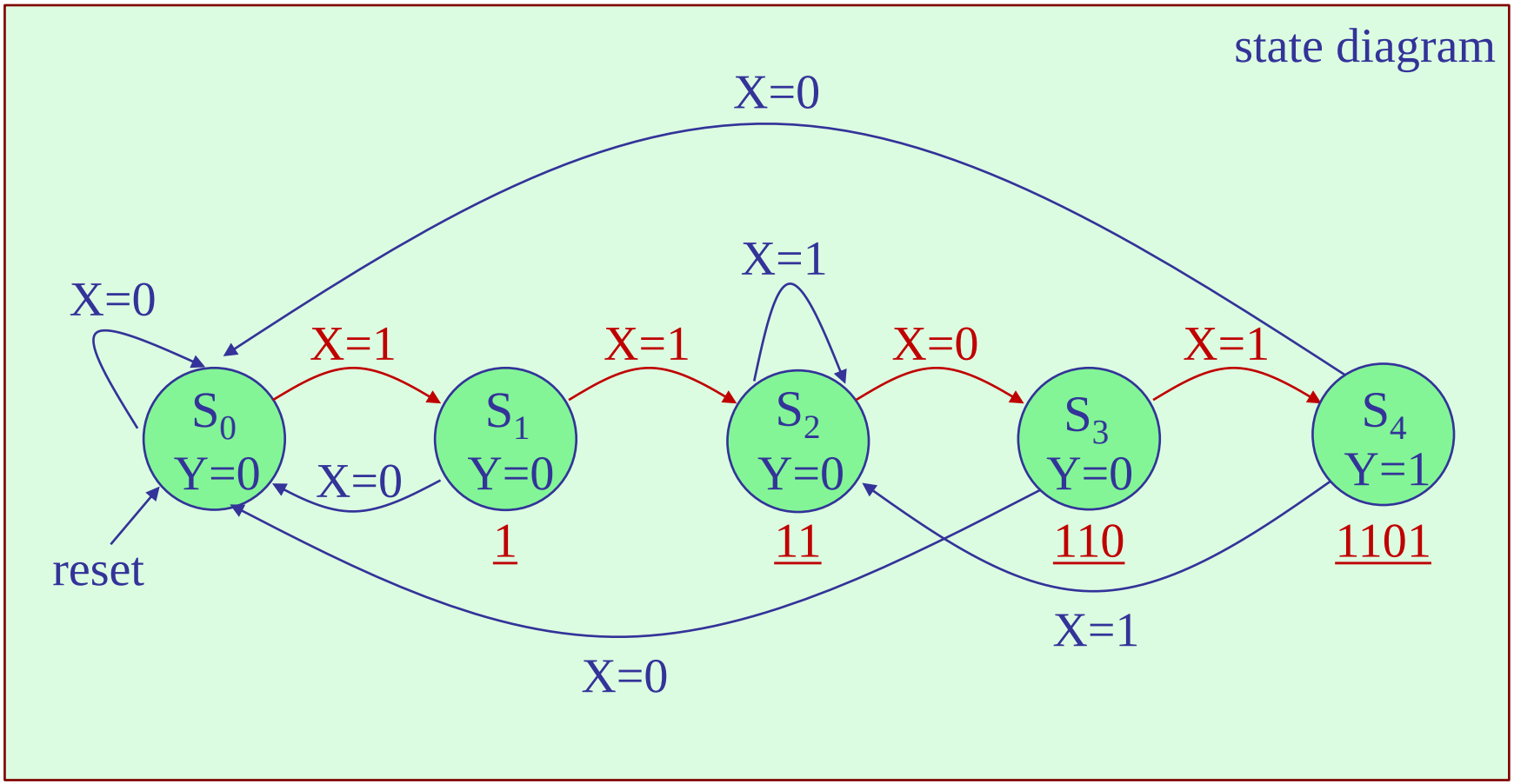
state table

Moore FSM

present state	next state X=0	next state X=1	output Y
S_0	S_0	S_1	0
S_1	S_0	S_2	0
S_2	S_3	S_2	0
S_3	S_0	S_4	0
S_4	S_0	S_2	1

state encoding

	A	B	C
S_0 ≡	0	0	0
S_1 ≡	0	0	1
S_2 ≡	0	1	1
S_3 ≡	0	1	0
S_4 ≡	1	0	0


state diagram
X=0
X=0
X=1
X=1
X=1
X=0
X=1
S0
Y=0
S1
Y=0
S2
Y=0
S3
Y=0
S4
Y=1
X=0
1
11
110
1101
reset
X=1
X=0

present	ABC	next ABC X=0	next ABC X=1	Y
S_0	0 0 0	S_0 0 0 0	S_1 0 0 1	0
S_1	0 0 1	S_0 0 0 0	S_2 0 1 1	0
S_2	0 1 1	S_3 0 1 0	S_2 0 1 1	0
S_3	0 1 0	S_0 0 0 0	S_4 1 0 0	0
S_4	1 0 0	S_0 0 0 0	S_2 0 1 1	1
	1 0 1	x x x	x x x	x
	1 1 1	x x x	x x x	x
	1 1 0	x x x	x x x	x

CX \ AB	00	01	11	10
00			x	
01			x	
11			x	x
10			x	x

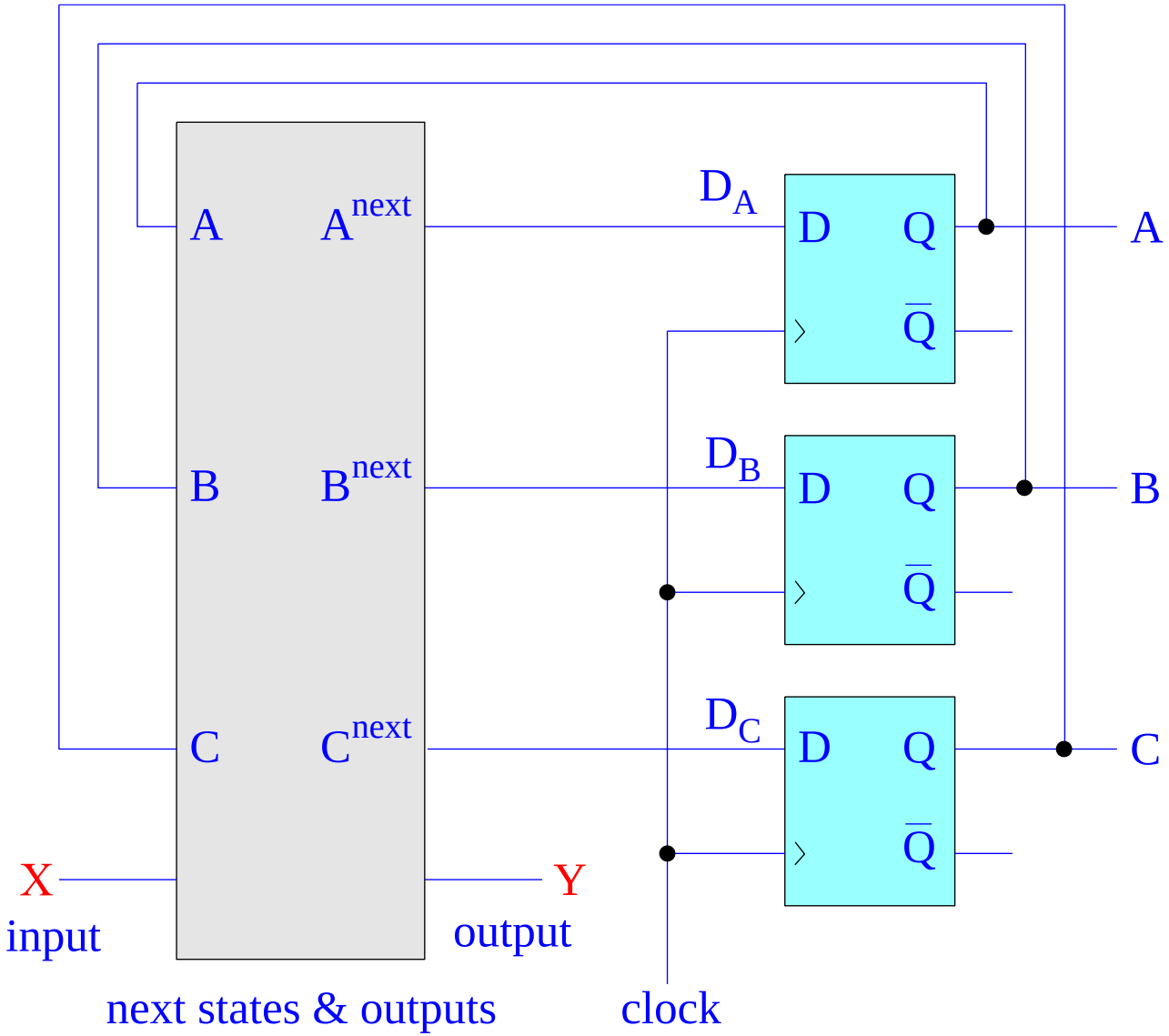
$D_A = A^{next} = B\ C'\ X$

$D_B = B^{next} = C\ X + A\ X + B\ C$

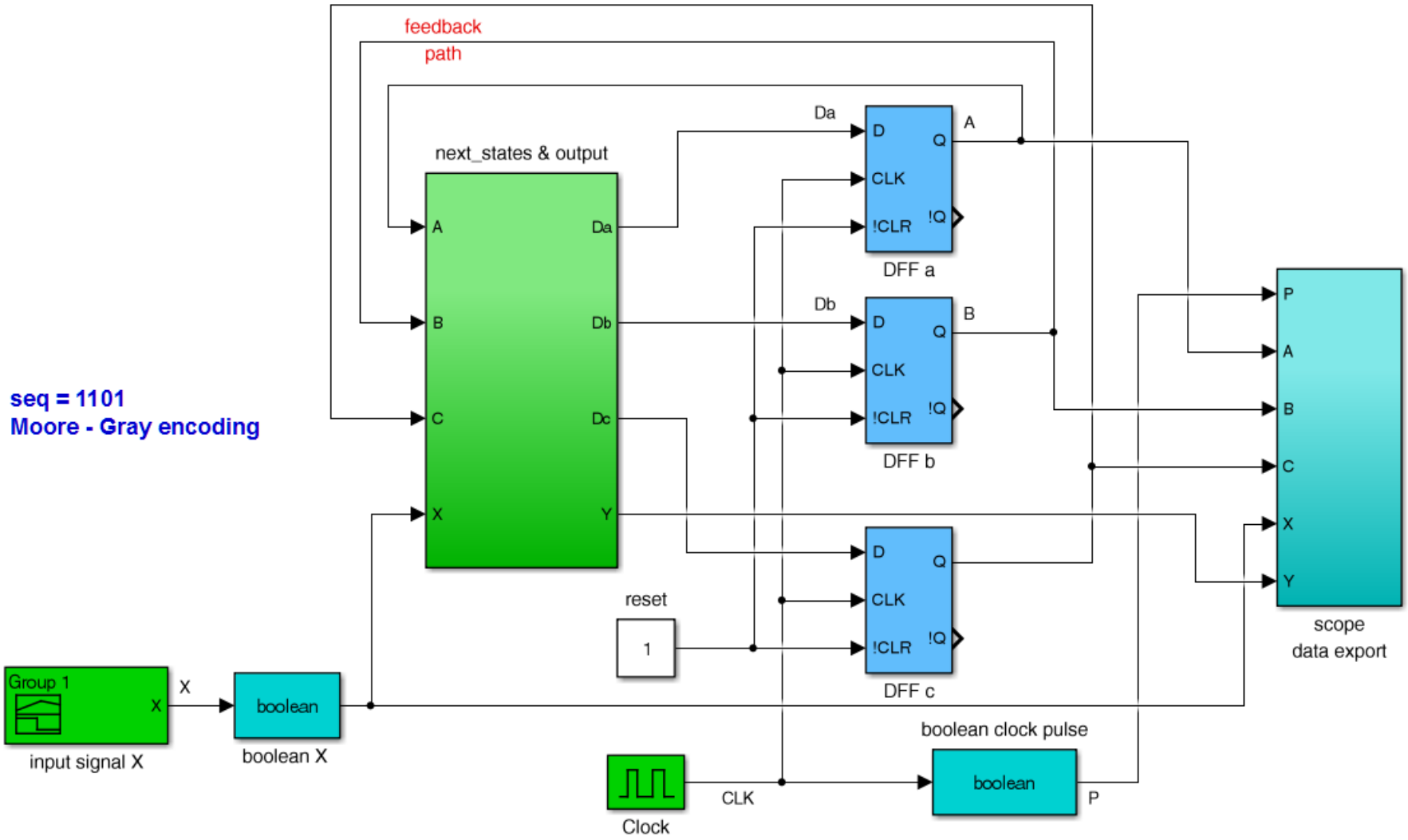
$D_C = C^{next} = C\ X + B'\ X$

$Y = A$

next states & output equations


A
B
C
A next
B next
C next
X
input
Y
output
next states & outputs
clock
D_A
D_B
D_C
D
Q
Q̄
A
B
C

Simulink implementation



fsm1101gmo.slx

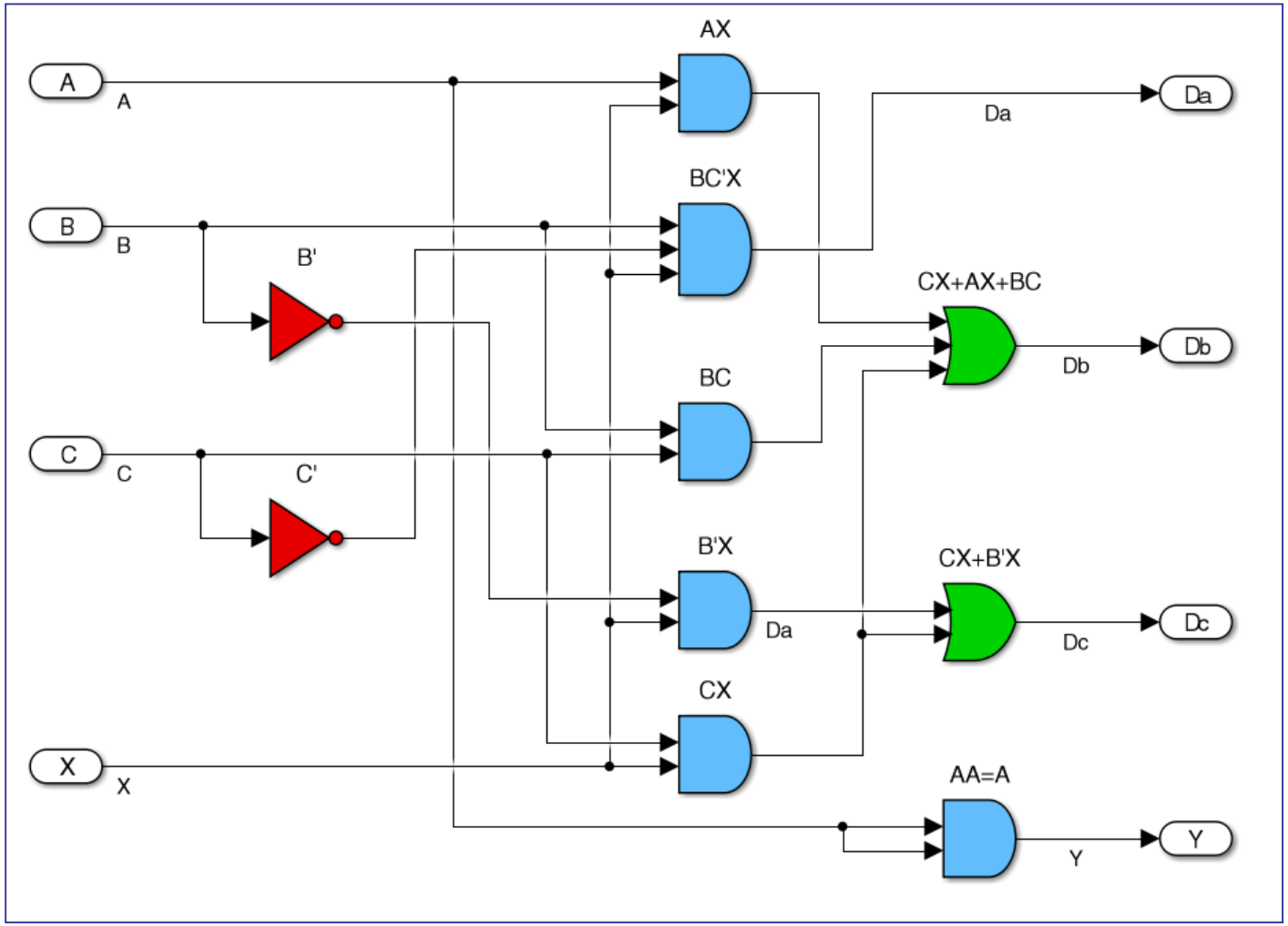
$$D_A = A^{next} = B\ C'\ X$$

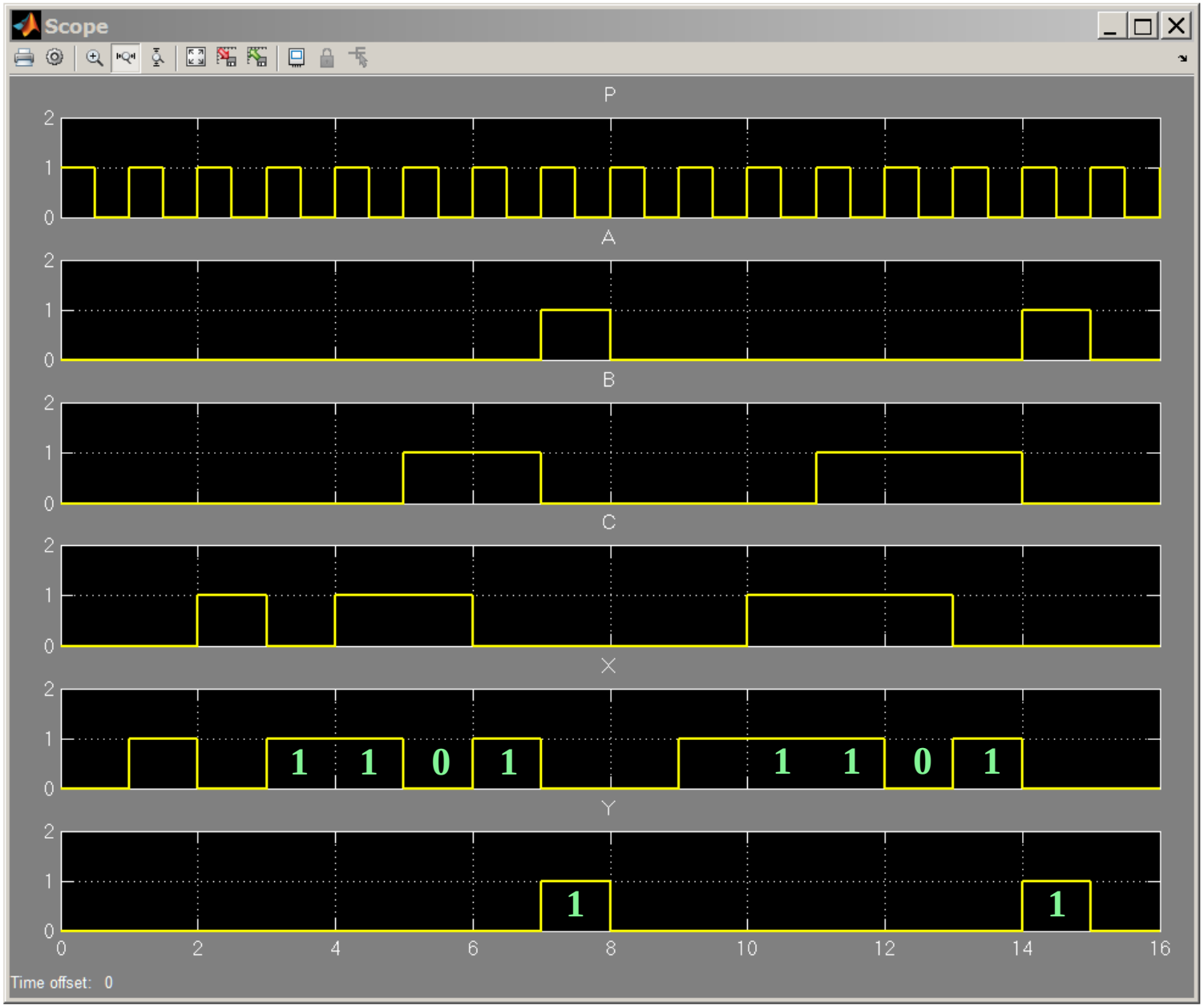
$$D_B = B^{next} = C\ X + A\ X + B\ C$$

$$D_C = C^{next} = C\ X + B'\ X$$

$$Y = A$$

next-states & output sub-function

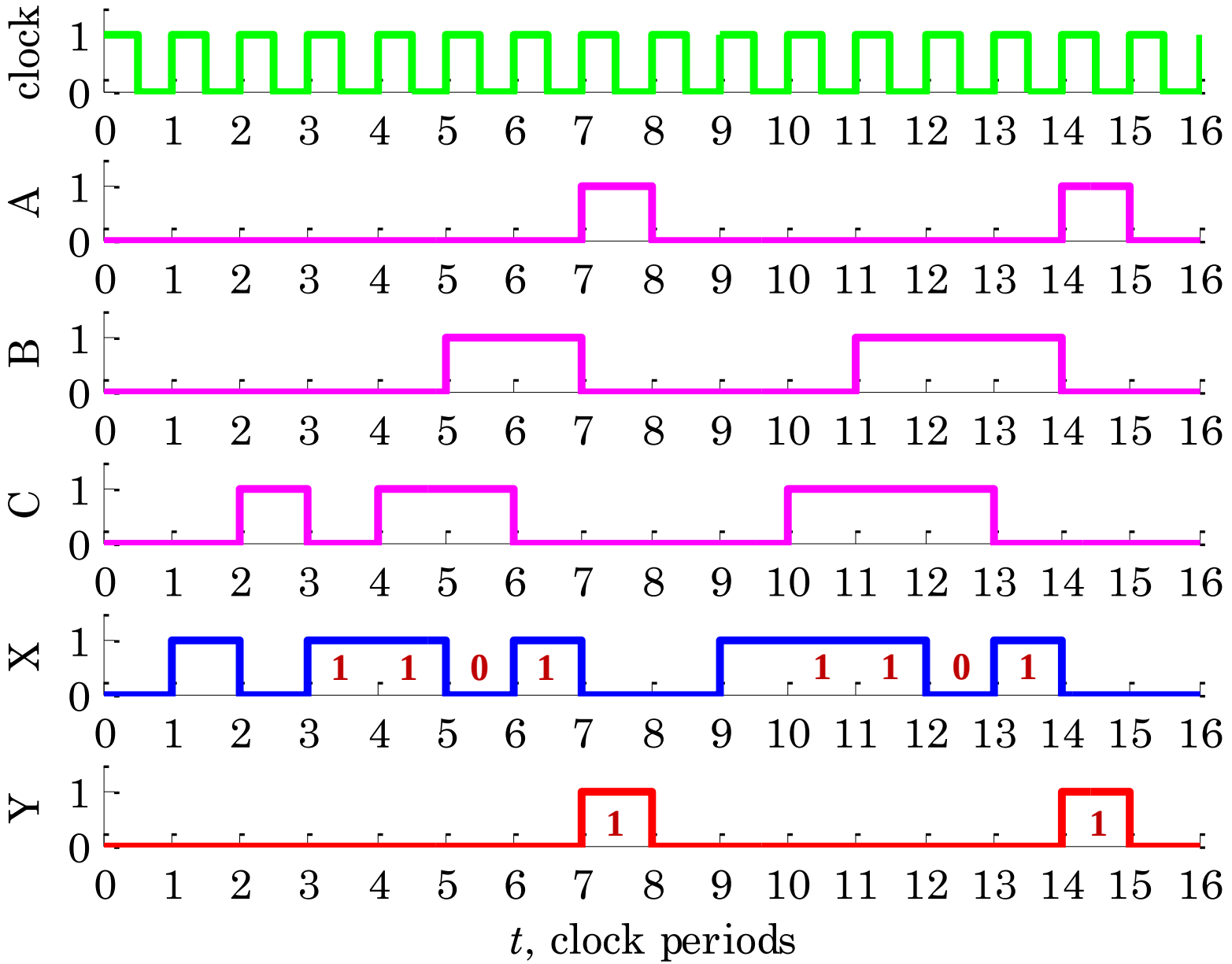




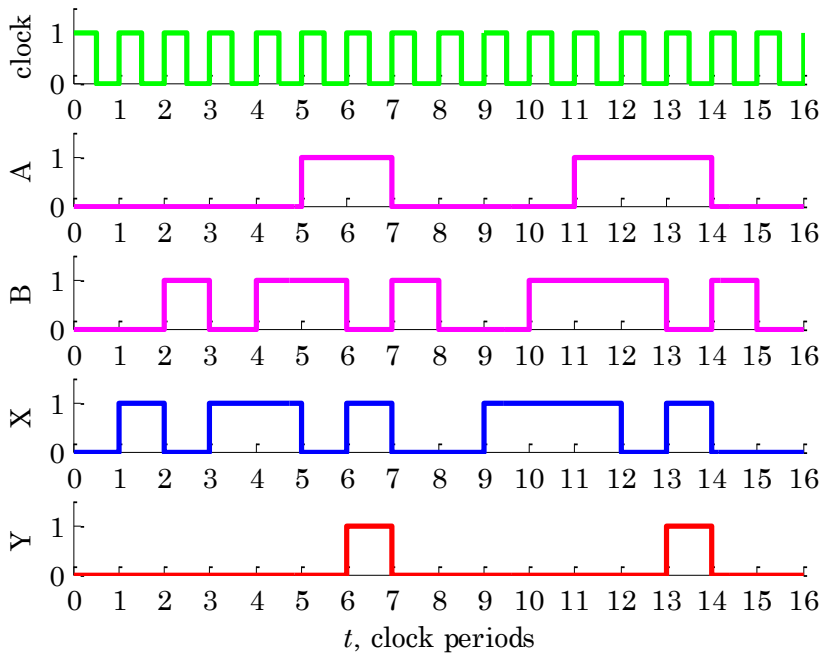
scope output

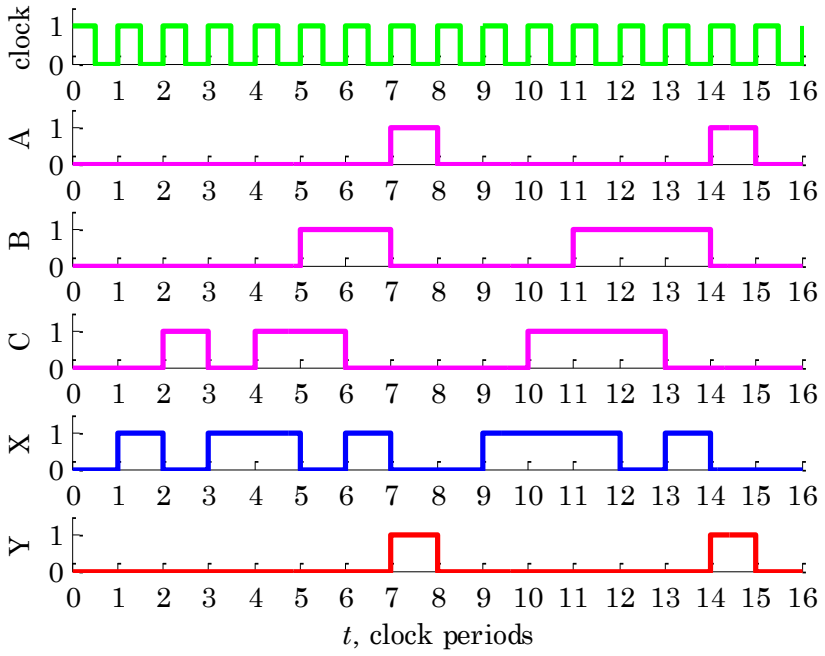
Example 7

sequence recognizer – 1101 – Moore version



Mealy vs. Moore





$D_A = A^{next} = (X + A)\, B$

$D_B = B^{next} = X$

$Y = X\, A\, B'$

$D_A = A^{next} = B\, C'\, X$

$D_B = B^{next} = C\, X + A\, X + B\, C$

$D_C = C^{next} = C\, X + B'\, X$

$Y = A$

Gray encoding for both cases

Note on State Assignment

With n flip-flops, there are 2^n possible n-bit patterns, or states. If our FSM only needs k coded states, (with $k \leq 2^n$), then, the number of different ways to choose such k coded states from the 2^n available ones is given by the binomial coefficient:

$$\binom{2^n}{k} = \frac{(2^n)!}{k!\,(2^n - k)!}$$

Moreover, there are $k!$ ways (permutations) to assign the chosen k coded n-bit states to k symbolic-named states, therefore, the total number of ways to assign k symbolic states to k, n-bit patterns is,

$$\binom{2^n}{k} \cdot k! = \frac{(2^n)!}{k!\,(2^n - k)!} \cdot k! = \frac{(2^n)!}{(2^n - k)!}$$

For our example, we have $n = 3$ and $k = 5$, which gives, $2^n = 8$, and,

$$\binom{8}{5} \cdot 5! = 56 \cdot 120 = 6720$$

Example 8 – Yet, another sequence recognizer – Mealy version. It is desired to design a Mealy FSM circuit to detect the particular sequence of consecutive bits, 0001, in an incoming input stream X of zeros and ones, measured at the positive edges of the clock signal.

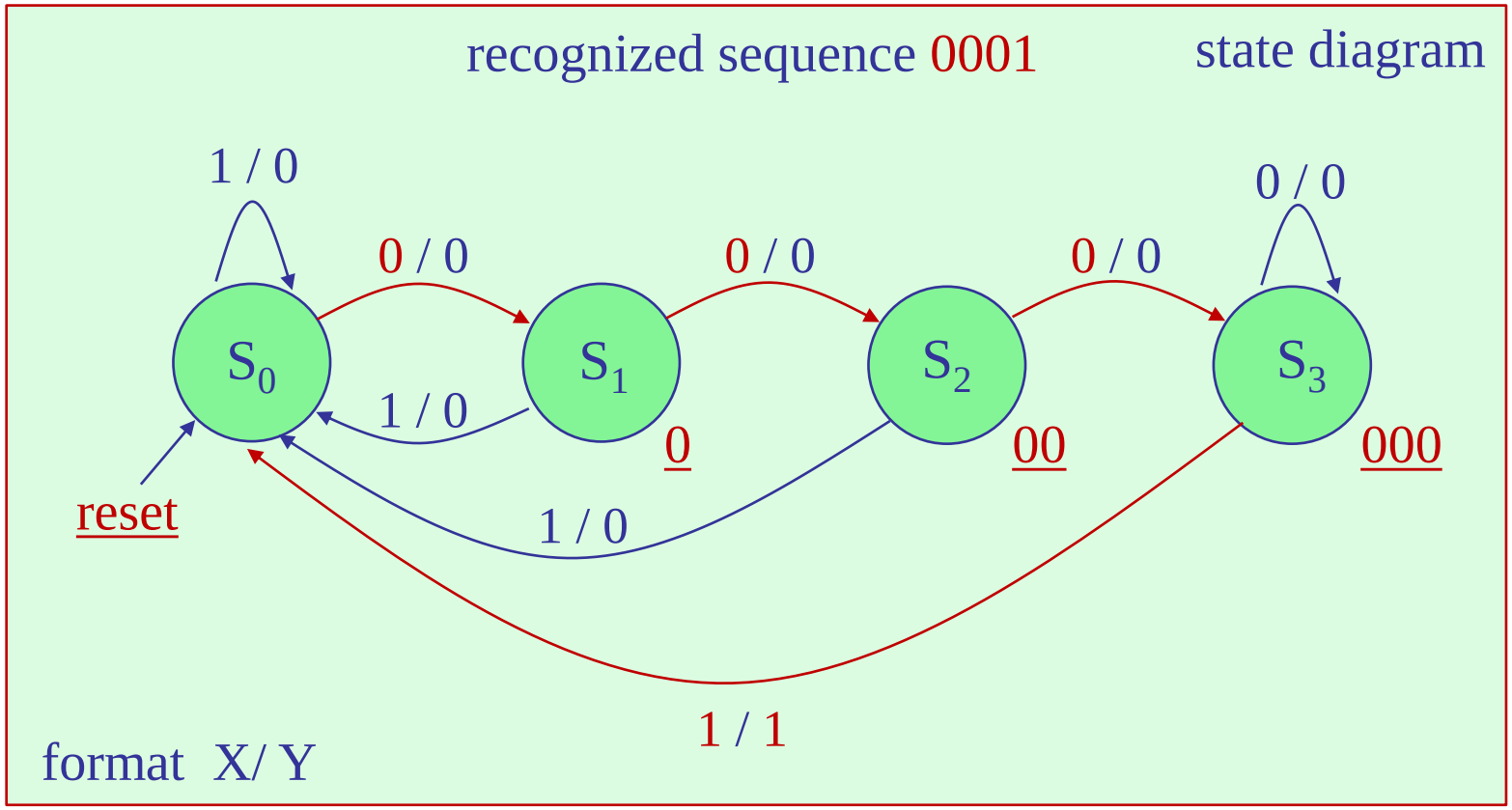
At each time instant, upon detecting the pattern 000 in the previous three inputs, and the current input is X=1, then, the FSM should produce an output, Y = 1, otherwise, it should produce, Y = 0.

Based on the above description, determine: (a) the state diagram, (b) state table, (c) state assignment with both Gray and binary encodings, (d) next-state and output equations, (e) build a realization using D flip-flops, and (f) simulate the operation of the system by sending in the following test sequence of binary inputs and verifying the expected outputs:

X = [0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0]

Y = [0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0]

only incomplete answers are given below.



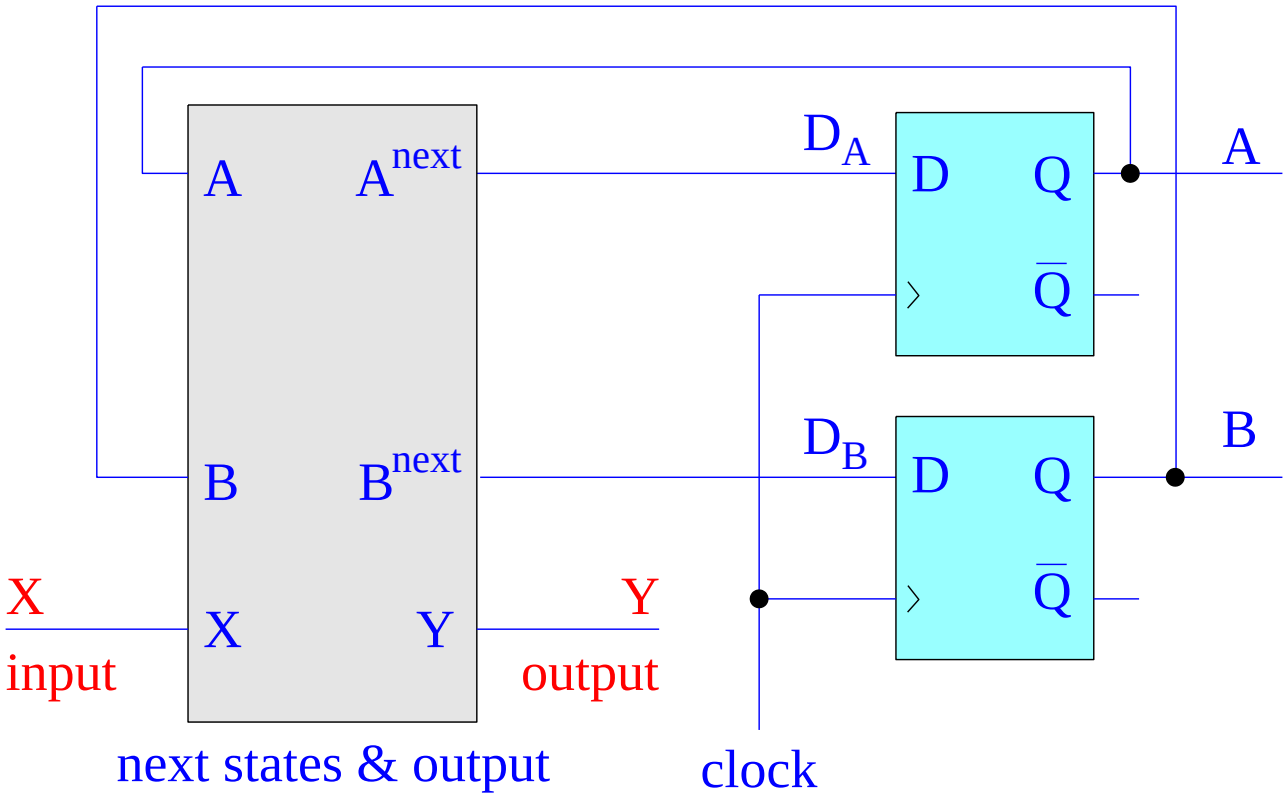
state table

Mealy FSM

present	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_1	S_0	0	0
S_1	S_2	S_0	0	0
S_2	S_3	S_0	0	0
S_3	S_3	S_0	0	1

State assignment. With N=4 states, we need, n = ceiling($\log_2 N$) = 2, state variables, say, A,B, and two D flip-flops, with inputs, say, D_A, D_B.

The overall system will be as shown below.



State encoding. Using K-maps, show the following next-state and output equations for the cases of plain binary and Gray encodings.

$D_A = A^{next} = X'(A + B)$

$D_B = B^{next} = X'(A + B')$

$Y = X\,A\,B$

binary encoding

binary encoding

	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	0
S_3 ≡	1	1

$D_A = A^{next} = X'(A + B)$

$D_B = B^{next} = X'A'$

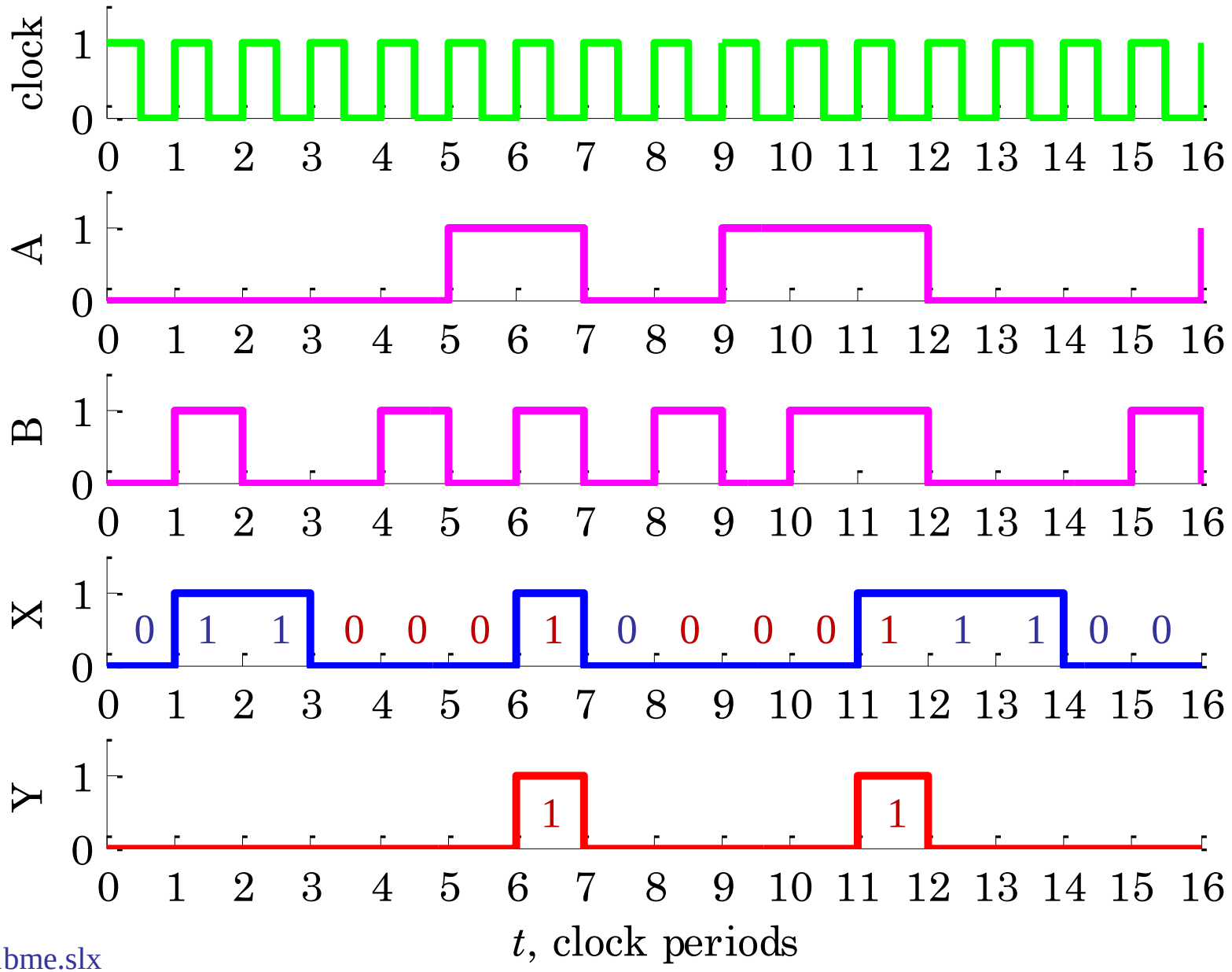
$Y = X\,A\,B'$

Gray encoding

Gray encoding

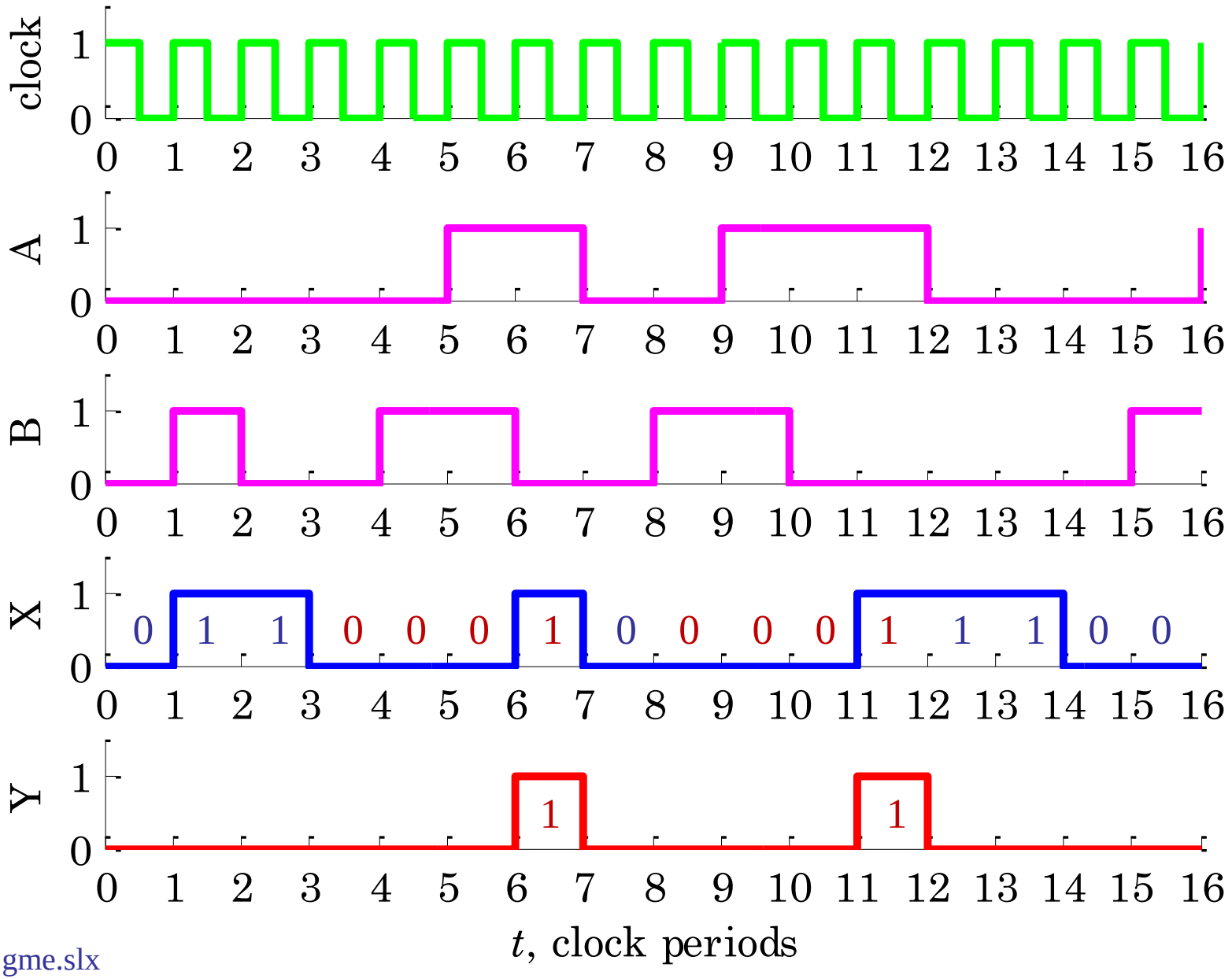
	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	1
S_3 ≡	1	0

sequence recognizer – 0001 – Mealy version – binary encoding



fsm0001bme.slx

sequence recognizer – 0001 – Mealy version – Gray encoding



fsm0001gme.slx

Example 9 – State reduction. It is often possible to reduce the number of states by identifying redundant states. Consider the following state table of a Mealy FSM, with one input X and one output Y [cf. Kana],

present	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_2	S_3	1	0
S_1	S_4	S_2	0	0
S_2	S_2	S_1	1	1
S_3	S_2	S_5	0	1
S_4	S_2	S_3	1	0
S_5	S_0	S_2	0	0

S_0 and S_4 are equivalent states, can keep S_0, and eliminate S_4, replacing references to S_4 by S_0

two states are equivalent if they produce the same outputs for the same inputs

note also that if a state does not appear as a next state, then it can be eliminated from the state table and state diagram

present	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_2	S_3	1	0
S_1	S_0	S_2	0	0
S_2	S_2	S_1	1	1
S_3	S_2	S_5	0	1
S_5	S_0	S_2	0	0

kept S_0, eliminated S_4,

replaced S_4 by S_0

now S_1 and S_5 are equivalent,

so, keep S_1, and eliminate S_5,

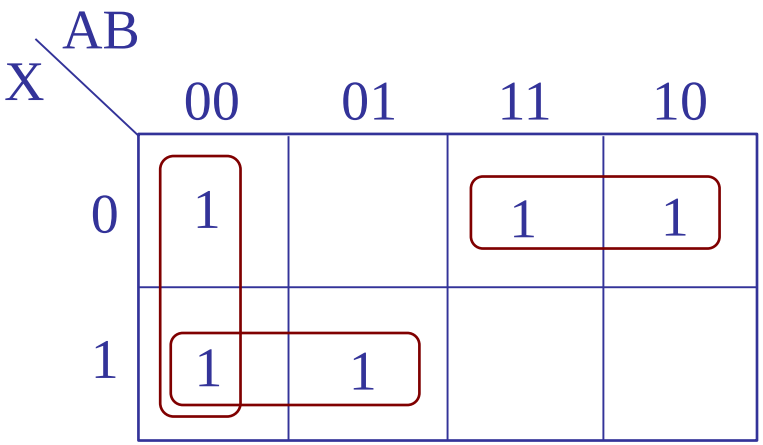
replace references to S_5 by S_1

present	next state		output Y	
	X=0	X=1	X=0	X=1
S_0	S_2	S_3	1	0
S_1	S_0	S_2	0	0
S_2	S_2	S_1	1	1
S_3	S_2	S_1	0	1

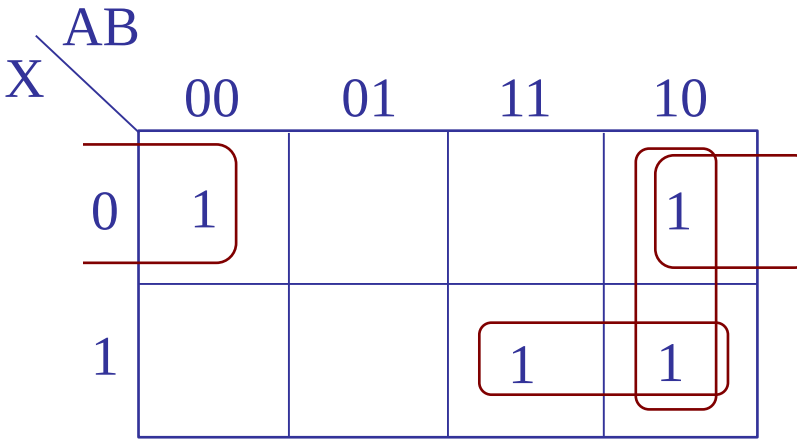
simplified state table,

next, assign flip-flop state variables, A,B, and derive the next-state and output equations

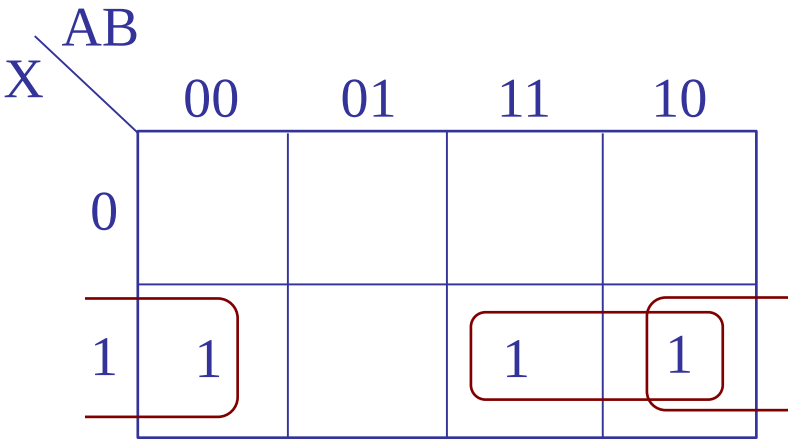
present states		next states A^{next}, B^{next}		output Y	
	A B	X=0	X=1	X=0	X=1
S_0	0 0	S_2 1 0	S_3 1 1	1	0
S_1	0 1	S_0 0 0	S_2 1 0	0	0
S_2	1 0	S_2 1 0	S_1 0 1	1	1
S_3	1 1	S_2 1 0	S_1 0 1	0	1



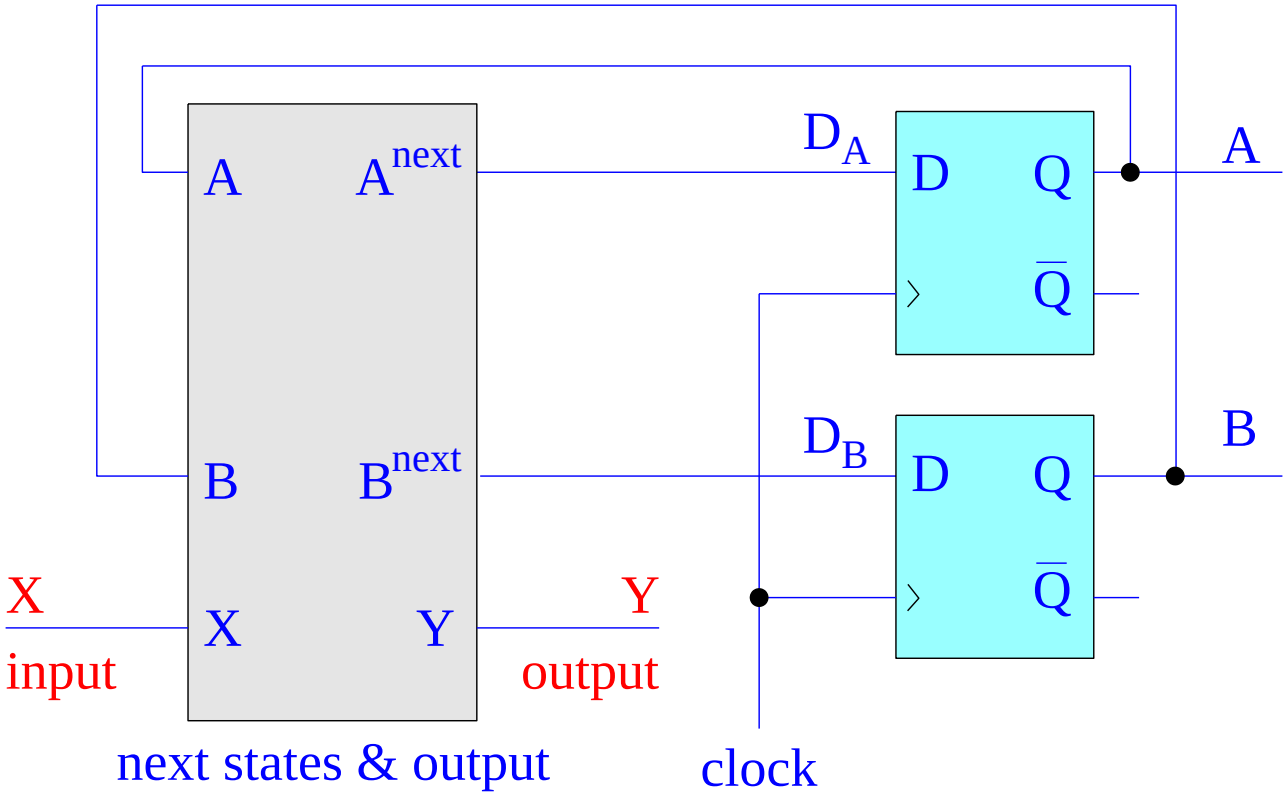
$A^{next} = A' B' + X A' + X' A$



$Y = A B' + X A + X' B'$



$B^{next} = X B' + X A$

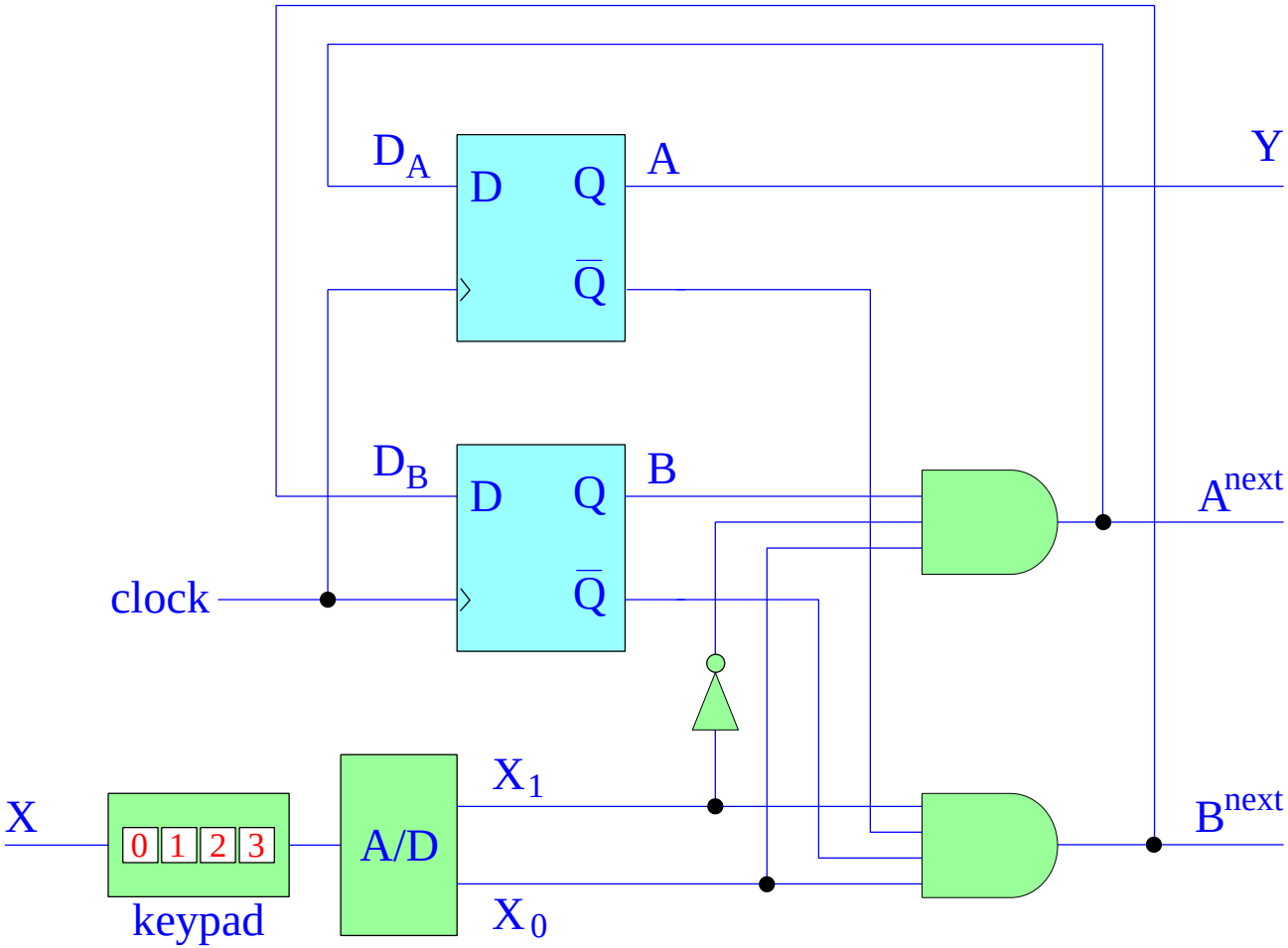


$$D_A = A^{next} = A' B' + X A' + X' A$$

$$D_B = B^{next} = X B' + X A$$

$$Y = A B' + X A + X' B'$$

Example 10 – Electronic keypad lock [cf. Harris & Harris].
Your DLD classmates give you a two-digit keypad lock as a birthday present, but when you open the box, you discover that instead of user instructions, they have included the following schematic.



You recognize this as a Moore FSM, and you will attempt to analyze it and eventually arrive at a state diagram, which will help you determine the unlocking combination.

The keypad accepts as input X the decimal digits, 0,1,2,3, and converts them to binary through an A/D converter, that is, each decimal X = 0,1,2,3 is represented by the bits, X_1X_0 = 00, 01, 10, 11, so that there are two binary inputs to the FSM. There is also an output, Y=A, but you don't know its purpose yet, and you suspect it might be the "unlock" signal.

From the block diagram, you determine the next-state and flip-flop excitation equations and output equations:

$$D_A = A^{next} = BX_1' X_0$$
$$D_B = B^{next} = A' B' X_1 X_0$$
$$Y = A$$

Using these equations, you generate a possible state table that includes all possible evaluations of the flip-flop variables A, B for all inputs X_1, X_0, including also the above next states and output.

A	B	X_1	X_0	A^{next}	B^{next}	Y
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	0	1
1	0	1	1	0	0	1
1	1	0	0	0	0	1
1	1	0	1	1	0	1
1	1	1	0	0	0	1
1	1	1	1	0	0	1

$$D_A = A^{next} = BX_1' X_0$$

$$D_B = B^{next} = A' B' X_1 X_0$$

$$Y = A$$

```
% MATLAB code

[A,B,X1,X0] = a2d(0:15,4);

Anext = B & (~X1) & X0;

Bnext = (~A) & (~B) & X1 & X0;

Y = A;

[A,B,X1,X0, Anext,Bnext,Y]
```

Next, carry out state reduction by removing unused states, noticing that the state, **AB = 11**, does not appear as a next state, so it can be removed,

also noticing that the state, **AB=10**, always results in the same next-state, **00**, and the same, output **Y=1**, regardless of the input, so these rows can be combined replacing the inputs by "don't cares".

reduced state table

A	B	X_1	X_0	A^{next}	B^{next}	Y
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	x	x	0	0	1

Next, carry out state assignment, replacing the state variables A,B using symbolic names, say, S_0, S_1, S_2,

	A B
S_0	0 0
S_1	0 1
S_2	1 0

and re-write the reduced state table using the symbolic names, and also replace the binary inputs X_1,X_0 by their decimal values X

$D_A = A^{next} = BX_1' X_0$

$D_B = B^{next} = A' B' X_1 X_0$

$Y = A$

	A B
S_0	0 0
S_1	0 1
S_2	1 0

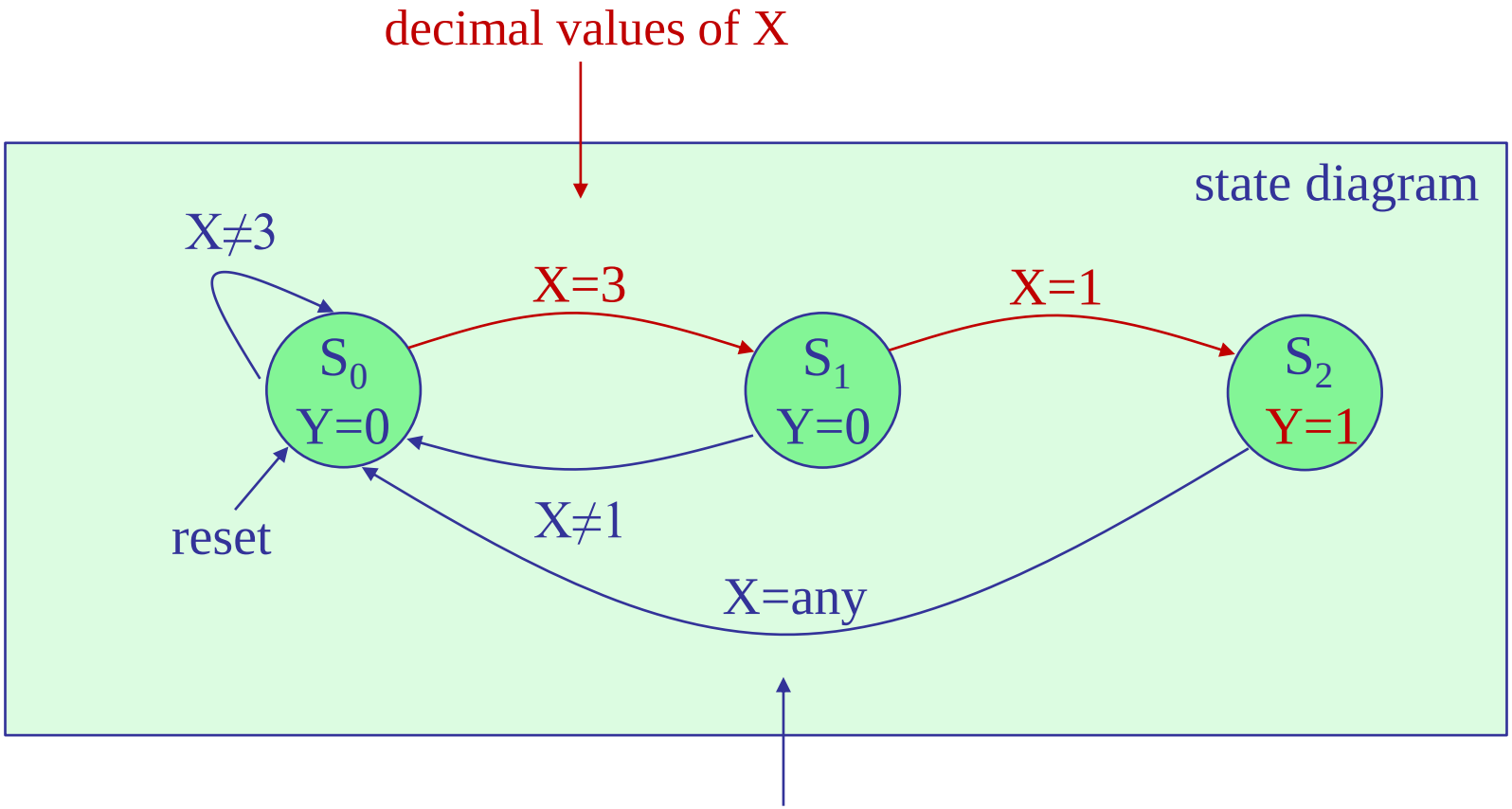
reduced state table

A	B	X_1	X_0	A^{next}	B^{next}	Y
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	1	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	x	x	0	0	1

symbolic state table

state	X	next	Y
S_0	0	S_0	0
S_0	1	S_0	0
S_0	2	S_0	0
S_0	3	S_1	0
S_1	0	S_0	0
S_1	1	S_2	0
S_1	2	S_0	0
S_1	3	S_0	0
S_2	x	S0	1

next, convert the state table into an equivalent state diagram



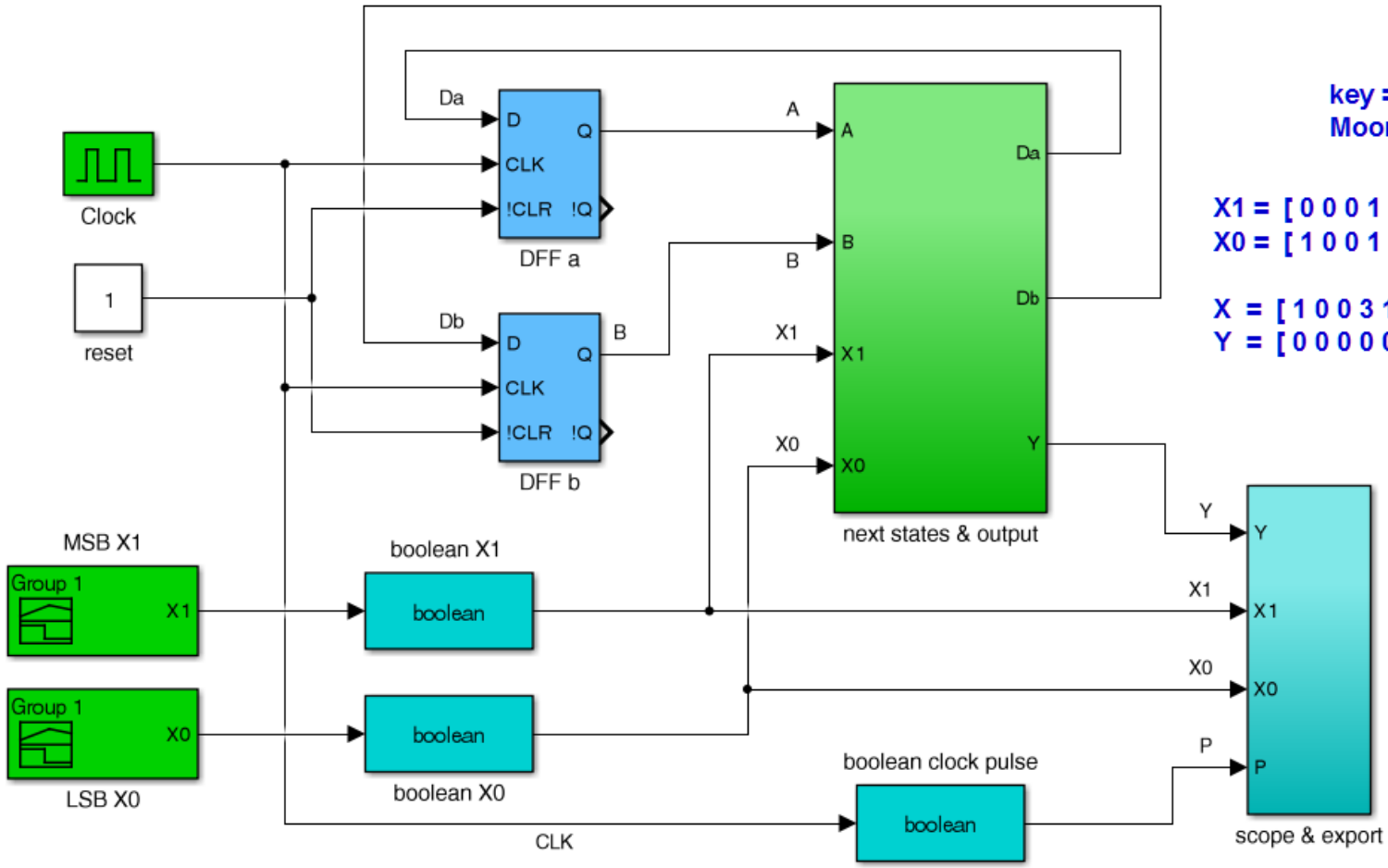
unlock combination is 31,

moving you to state S_2, and issuing the "unlock" signal Y=1,

and the FSM, then, moves to the reset/lock state S_0

next, test the FSM operation with Simulink

Simulink implementation



key = 31
Moore version

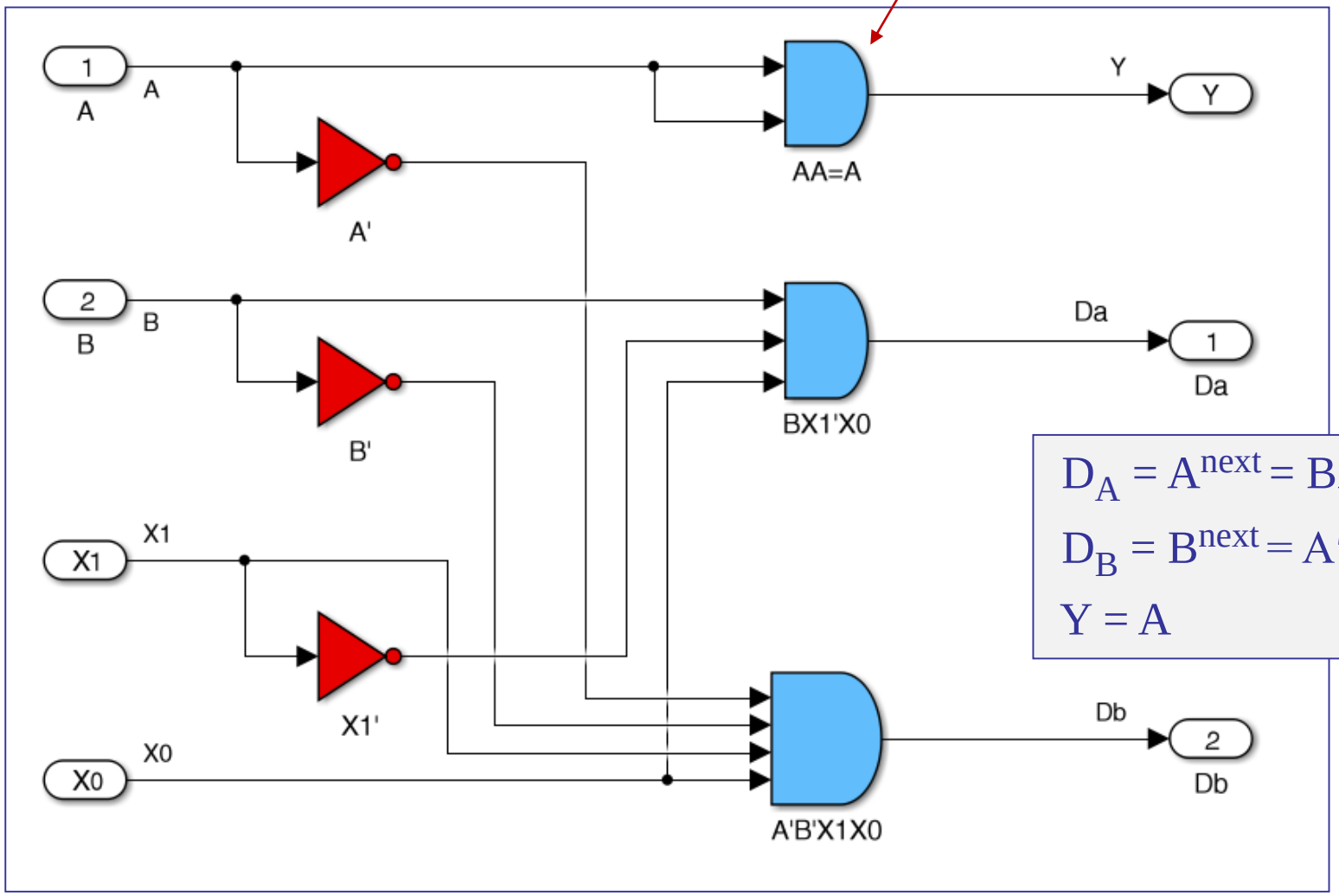
X1 = [0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0]
X0 = [1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 0]

X = [1 0 0 3 1 0 1 1 1 3 1 0 0 3 1 0]
Y = [0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1]

key31a.slx

next-states & output sub-function

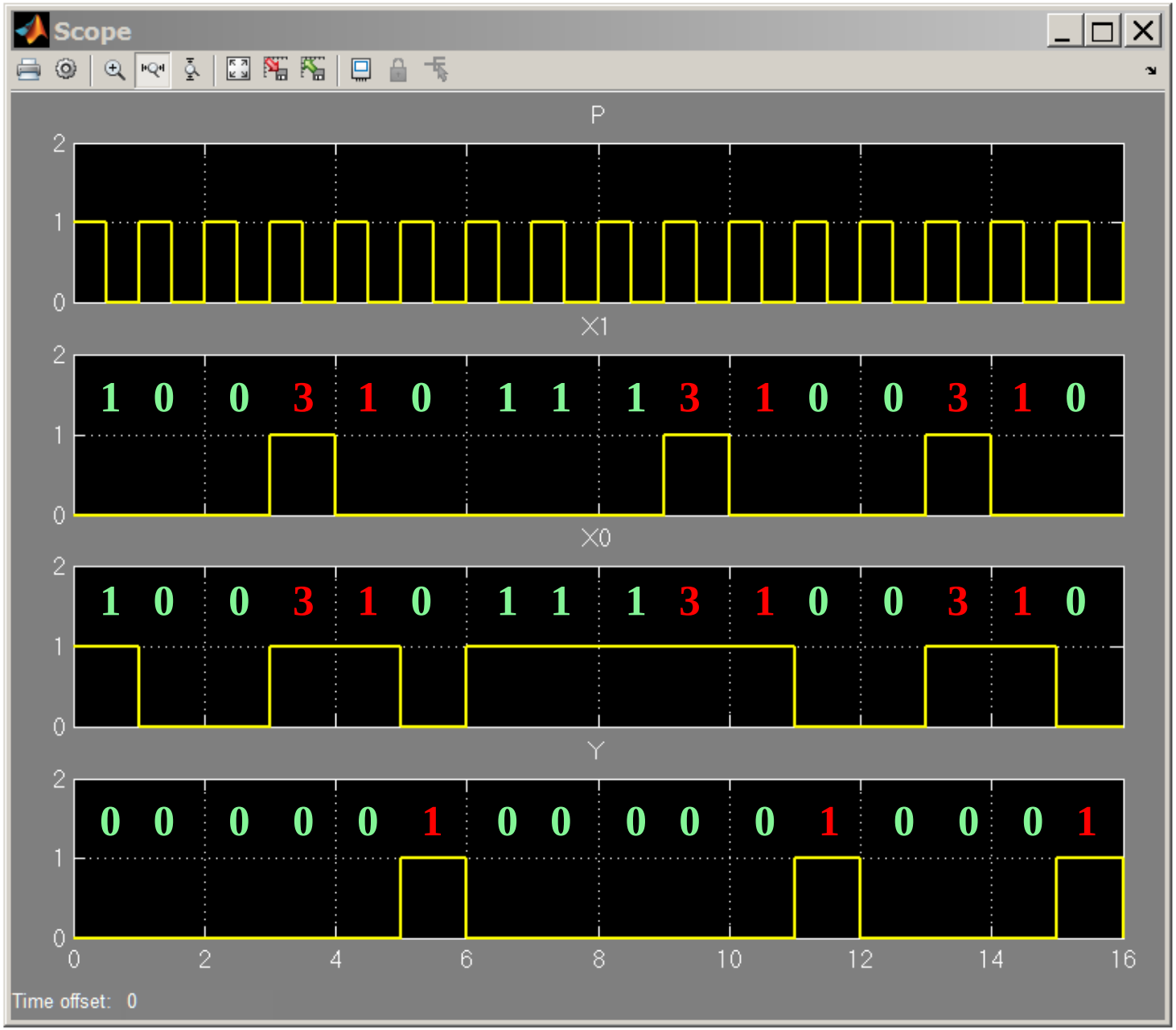
why is this necessary?



$$D_A = A^{next} = BX_1' X_0$$

$$D_B = B^{next} = A' B' X_1 X_0$$

$$Y = A$$

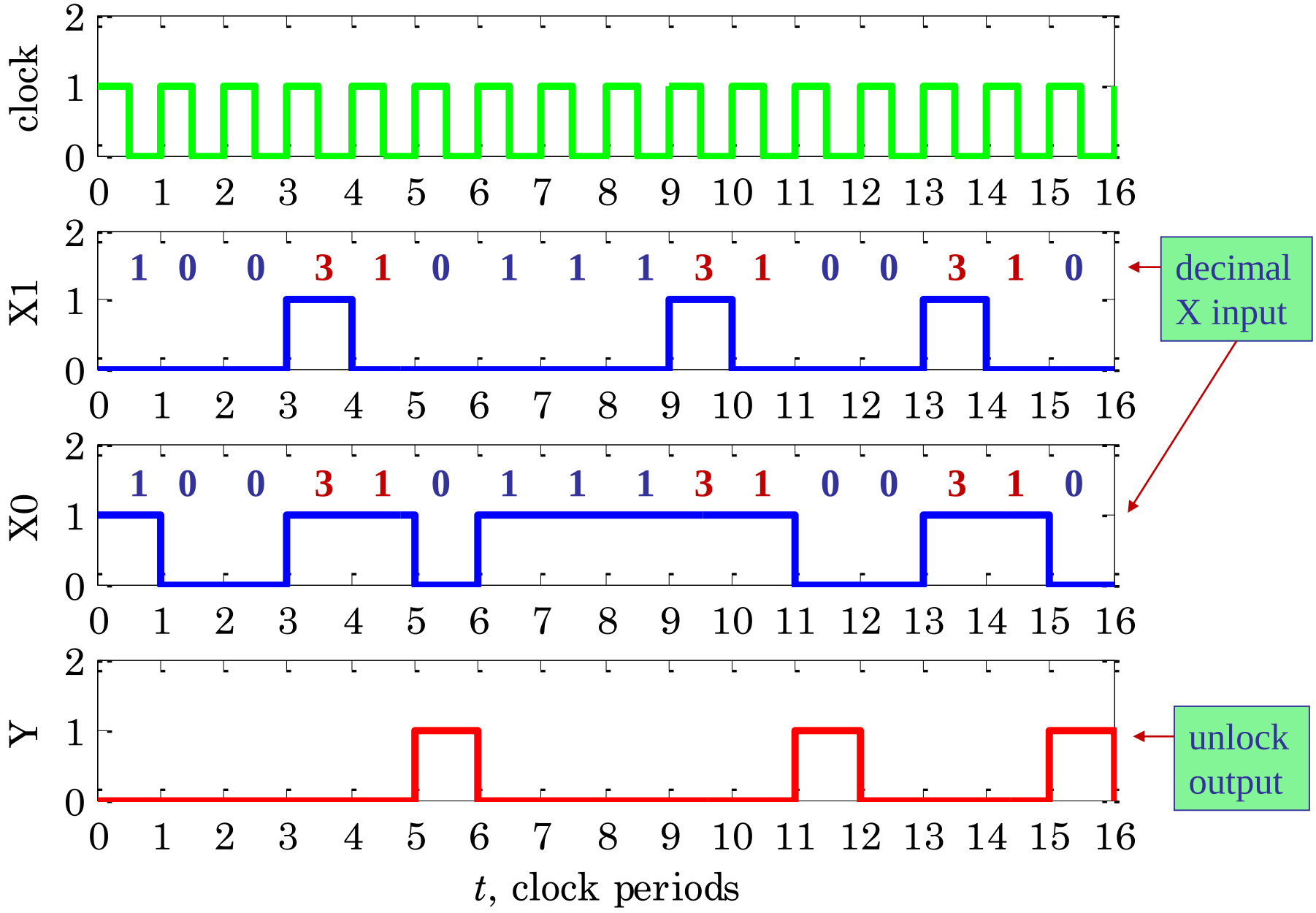


scope display

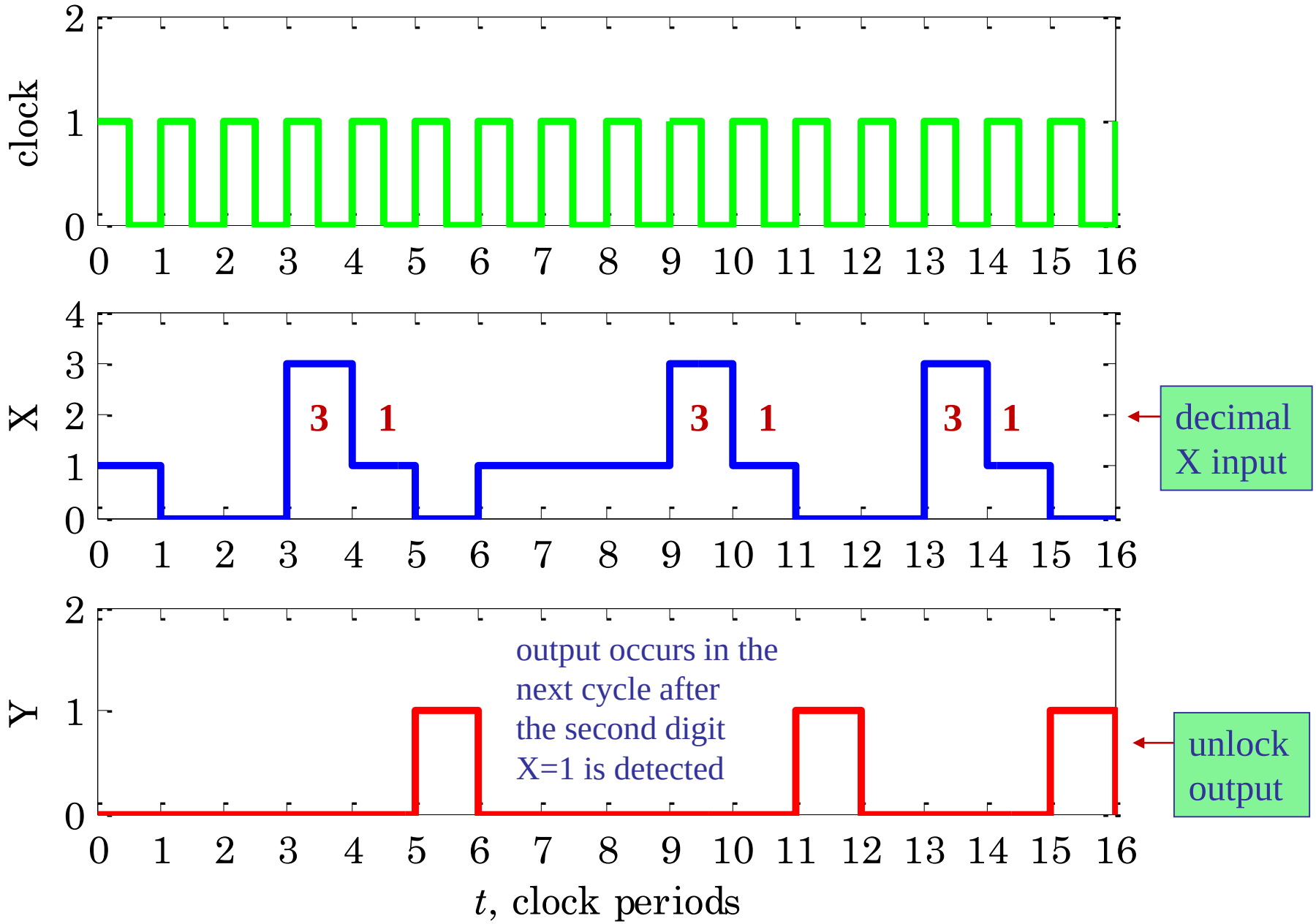
decimal X input

unlock output

Example 10

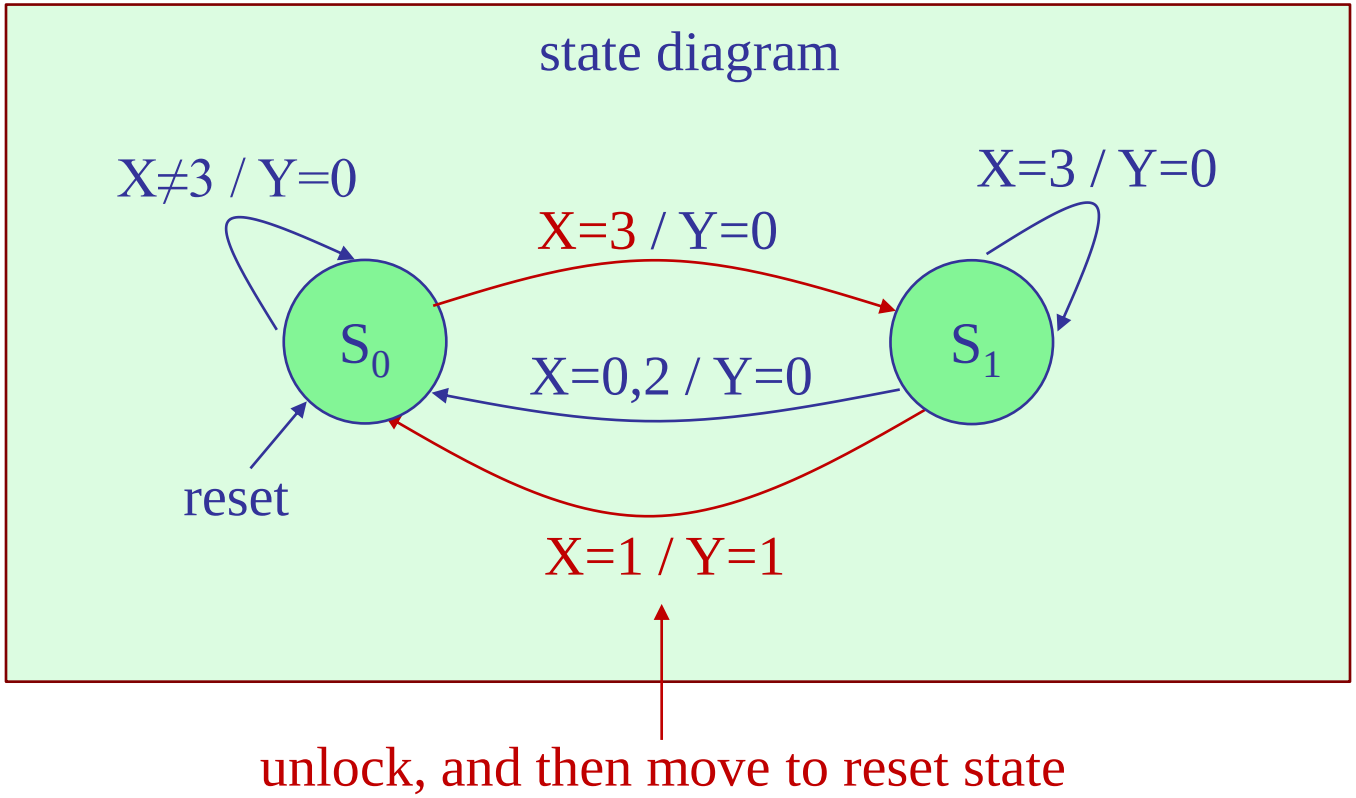

clock
X1
X0
Y
1 0 0 3 1 0 1 1 1 3 1 0 0 3 1 0
decimal
X input
unlock
output
t, clock periods

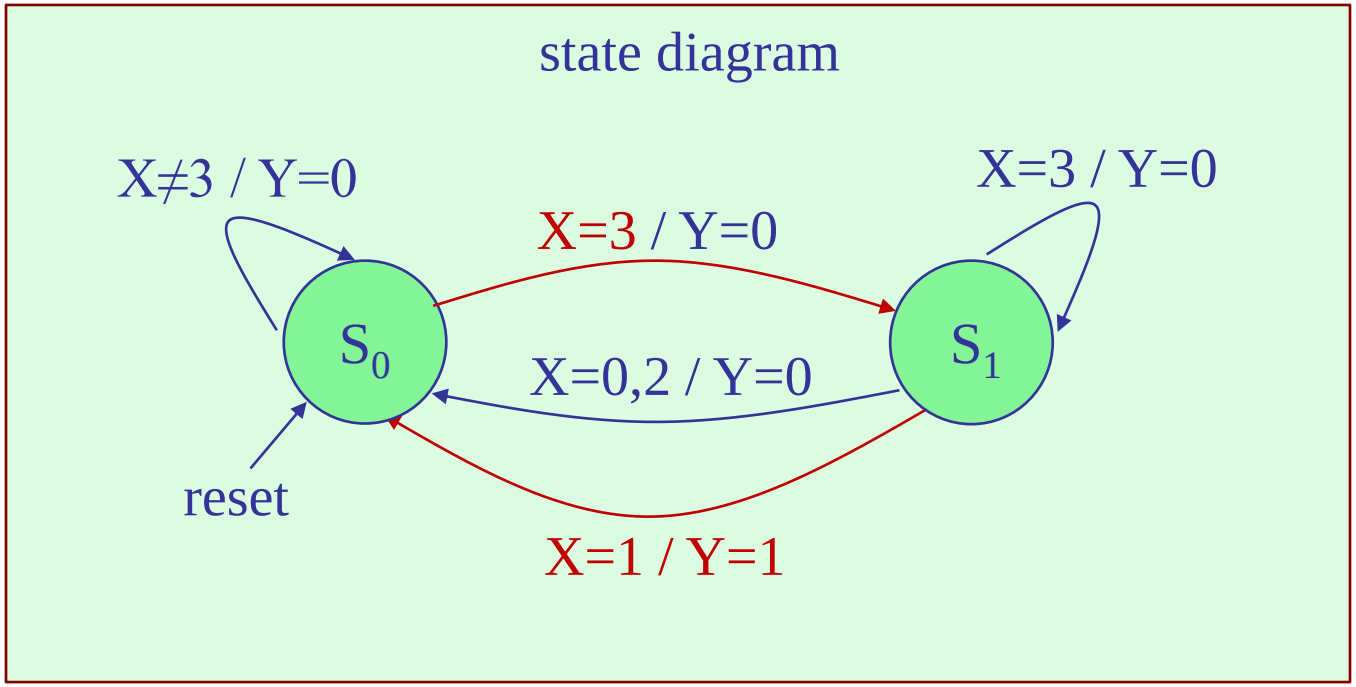
Example 10


clock
X
Y
3
1
3
1
3
1
decimal
X input
output occurs in the
next cycle after
the second digit
X=1 is detected
unlock
output
t, clock periods

Example 10

Example 11 – Electronic keypad lock. Having figured out the operation of your Moore FSM padlock, you now wish to derive a Mealy version. Since you know that the unlock combination is 31, you may start with a Mealy state diagram. Only two states are needed now because the system issues the unlock output, Y=1, as soon as it has recognized the second digit, X=1. The system unlocks, and then moves to the reset state.





symbolic state table

present state	next state				output Y			
	X=0	X=1	X=2	X=3	X=0	X=1	X=2	X=3
S_0	S_0	S_0	S_0	S_1	0	0	0	0
S_1	S_0	S_0	S_0	S_1	0	1	0	0

State encoding. With two states, we need one state variables, say, A, and one D flip-flop, and represent the states as follows:

	A
S_0	0
S_1	1

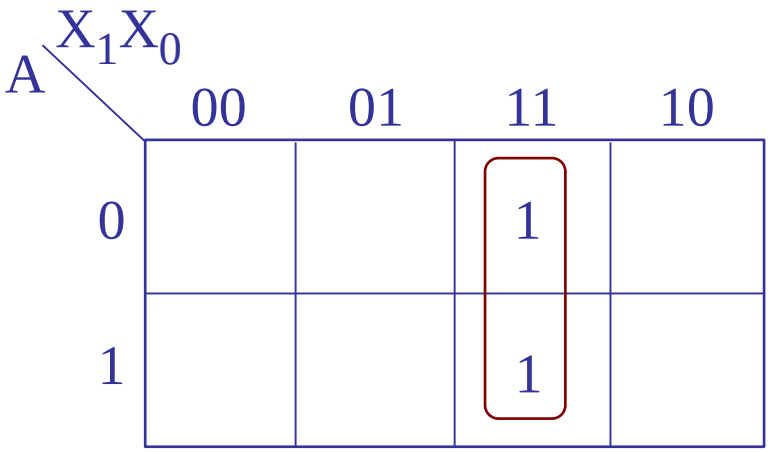
and also, replace the decimal X with its binary 2-bit representation, X_1X_0

encoded state table

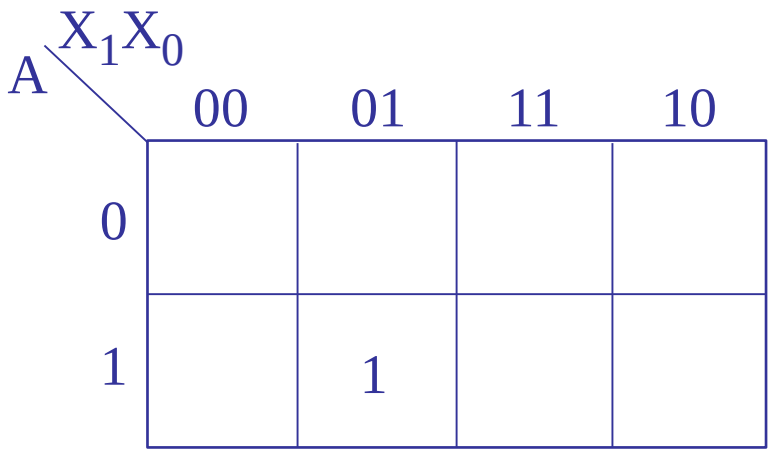
present state	next state A^{next}				output Y			
A	X=00	X=01	X=10	X=11	X=00	X=01	X=10	X=11
0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	1	0	0

next-state & output equations

present state	next state A^{next}				output Y			
A	X=00	X=01	X=10	X=11	X=00	X=01	X=10	X=11
0	0	0	0	1	0	0	0	0
1	0	0	0	1	0	1	0	0



$A^{next} = X_1 X_0$

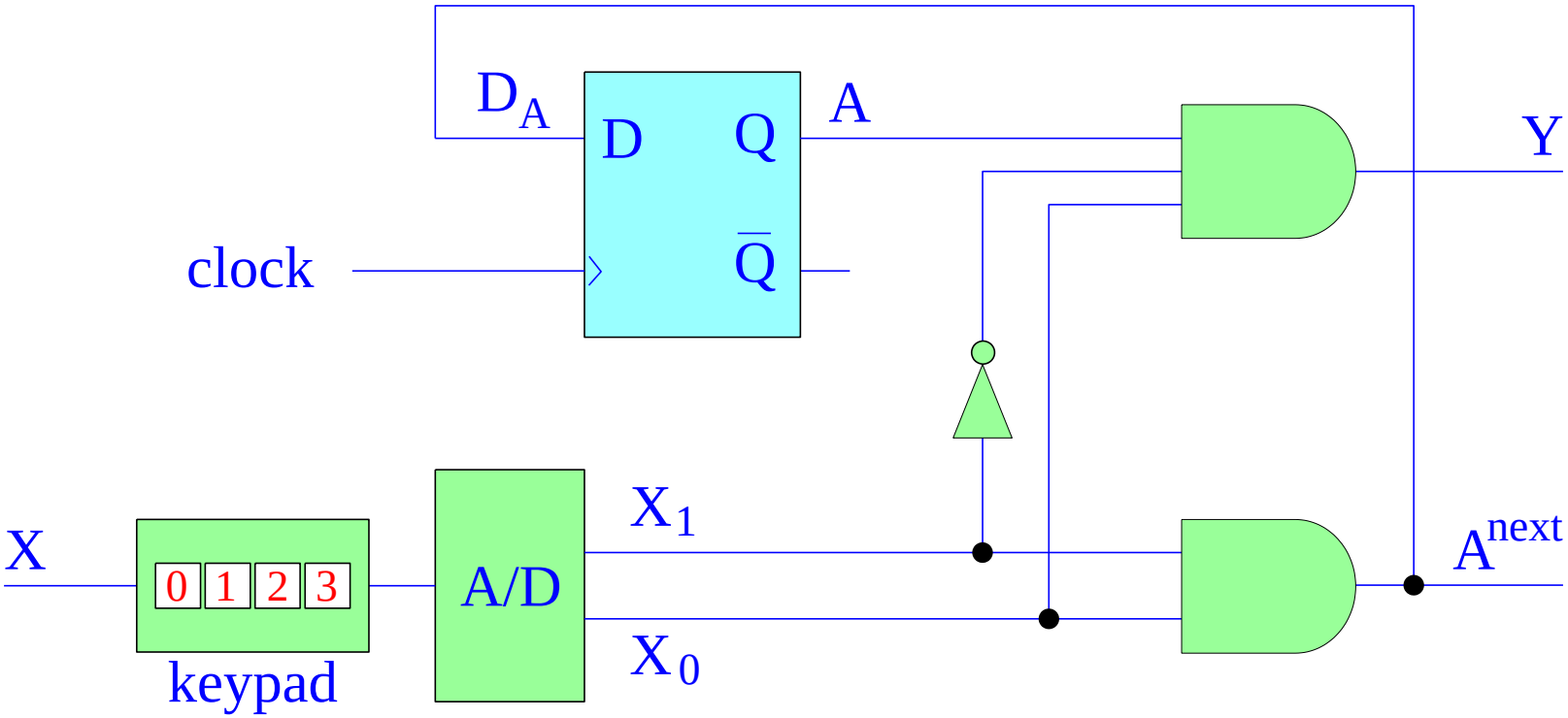


$Y = A X_1' X_0$

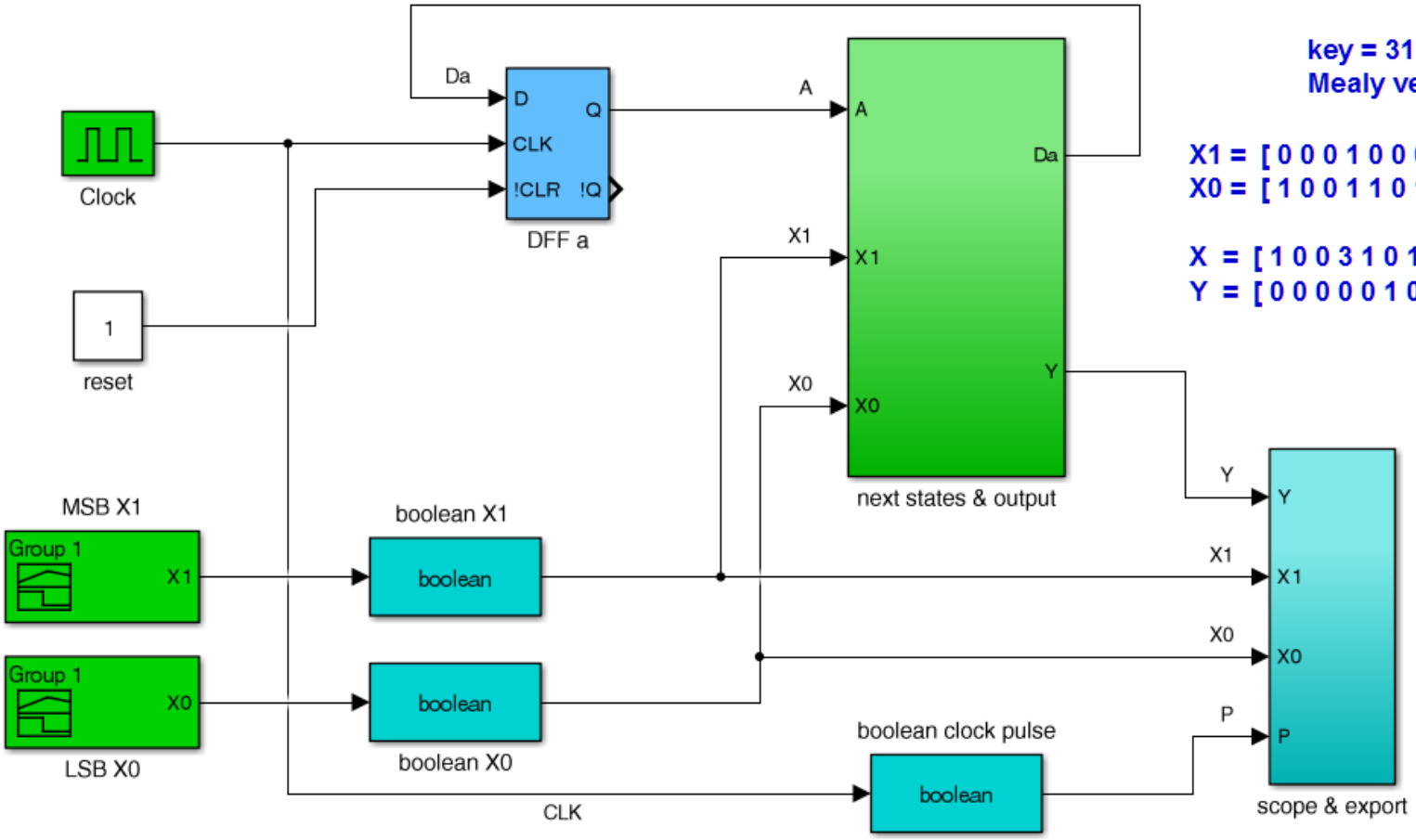
block diagram realization

$A^{next} = X_1 X_0$

$Y = A X_1' X_0$



Simulink implementation



key = 31
Mealy version

X1 = [0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0]
X0 = [1 0 0 1 1 0 1 1 1 1 1 0 0 1 1 0]

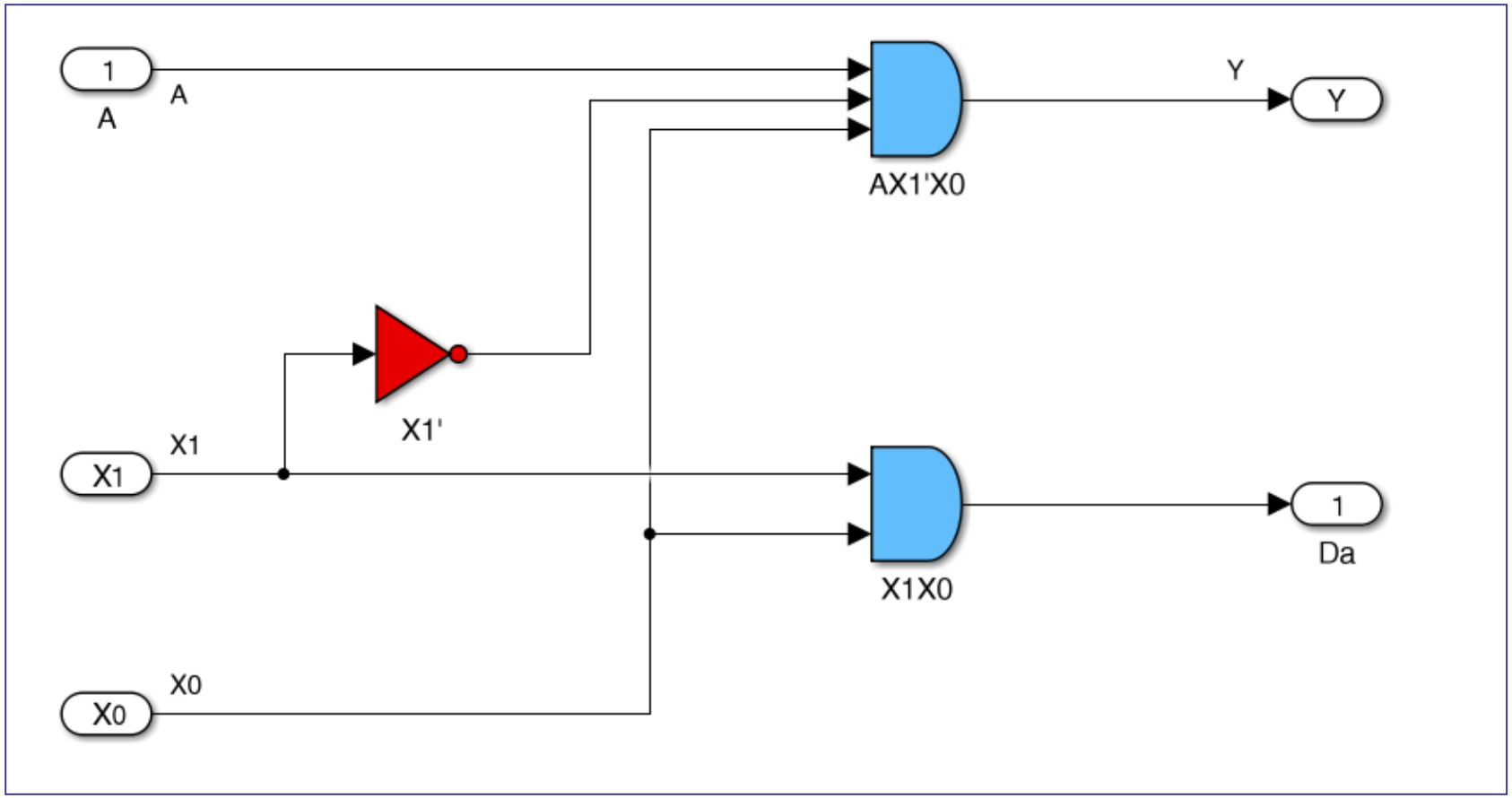
X = [1 0 0 3 1 0 1 1 1 3 1 0 0 3 1 0]
Y = [0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1]

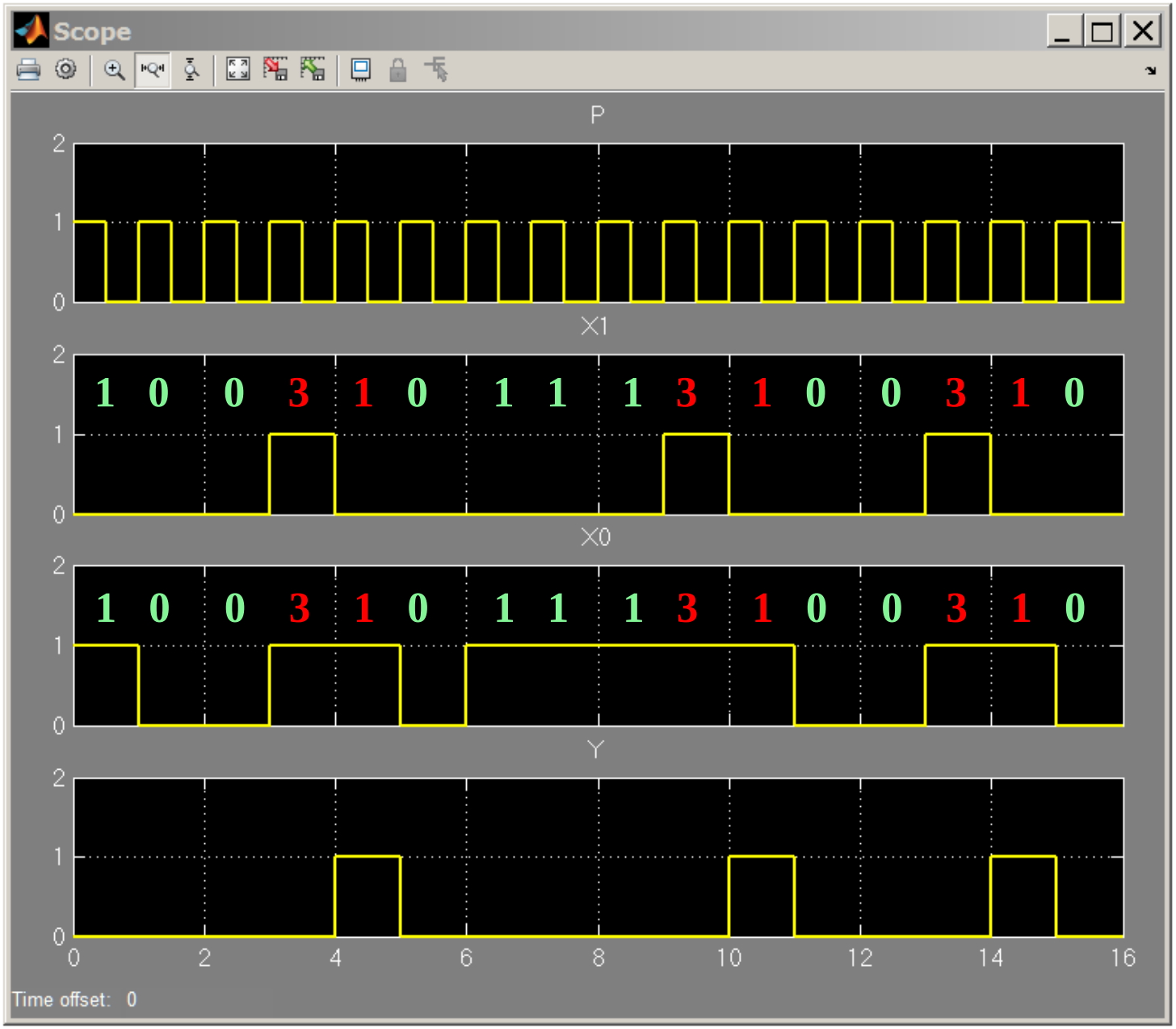
key31b.slx

next-state & output sub-function

$A^{next} = X_1 X_0$

$Y = A X_1' X_0$



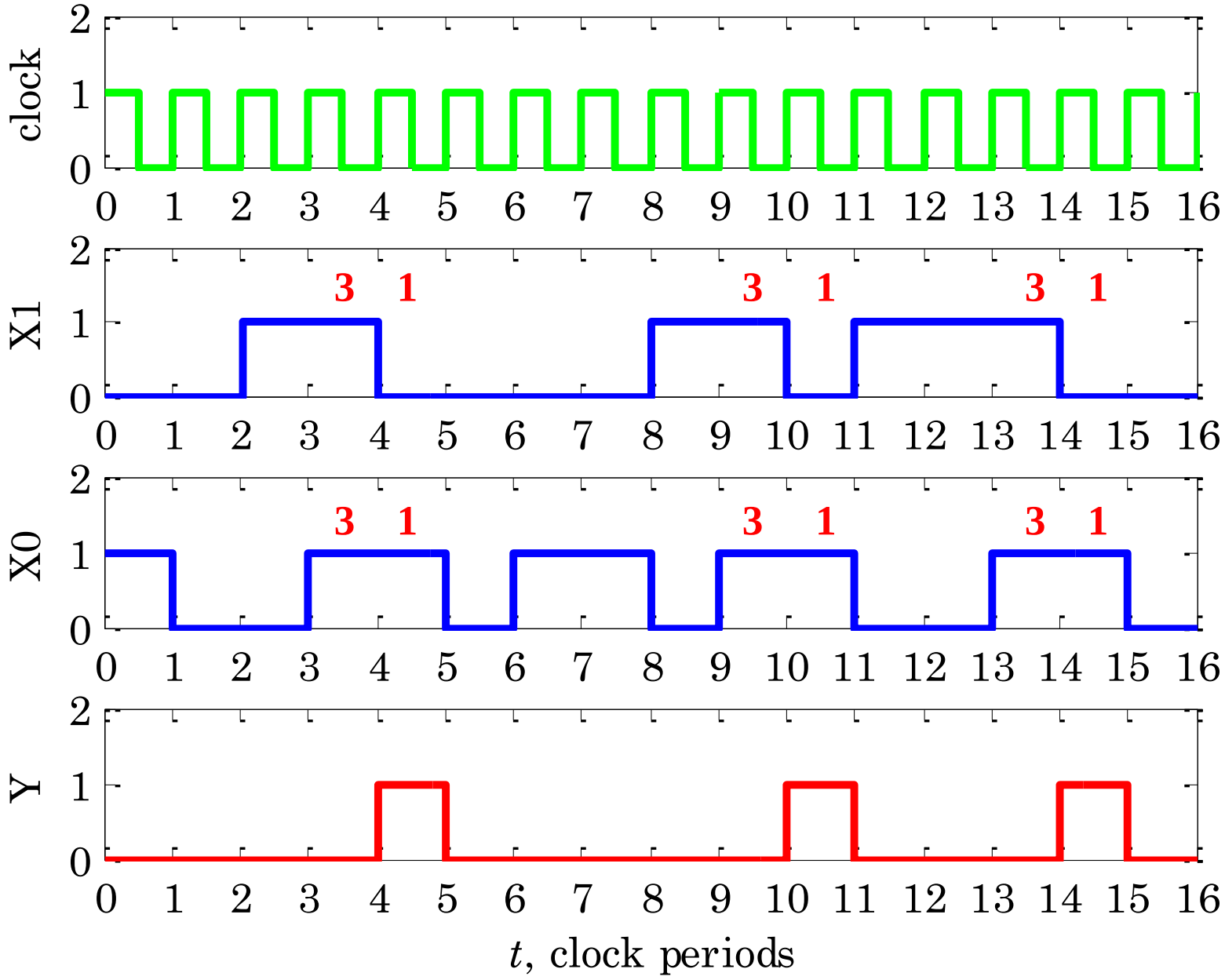


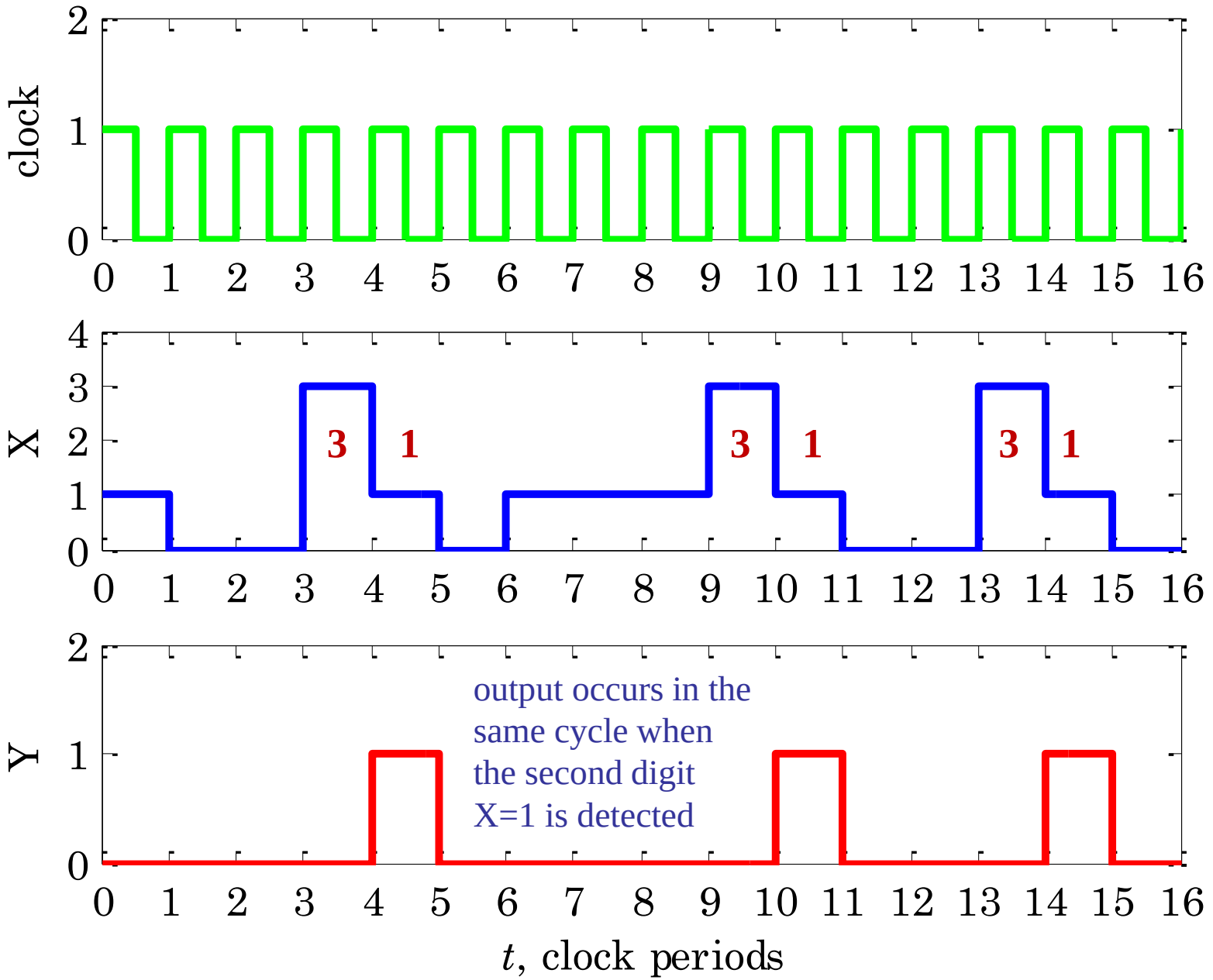
scope display

decimal X input

unlock output

Example 11


clock
X1
X0
Y
3 1
3 1
3 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t, clock periods


clock
X
Y
3
1
3
1
3
1
output occurs in the
same cycle when
the second digit
X=1 is detected
t, clock periods

Example 12 – Electronic keypad lock. Having enjoyed analyzing the 2-digit padlock, you now wish to build your own electronic keypad lock with a 3-digit combination and program the unlocking combination to be the course number of your favorite ECE course, that is, 231!

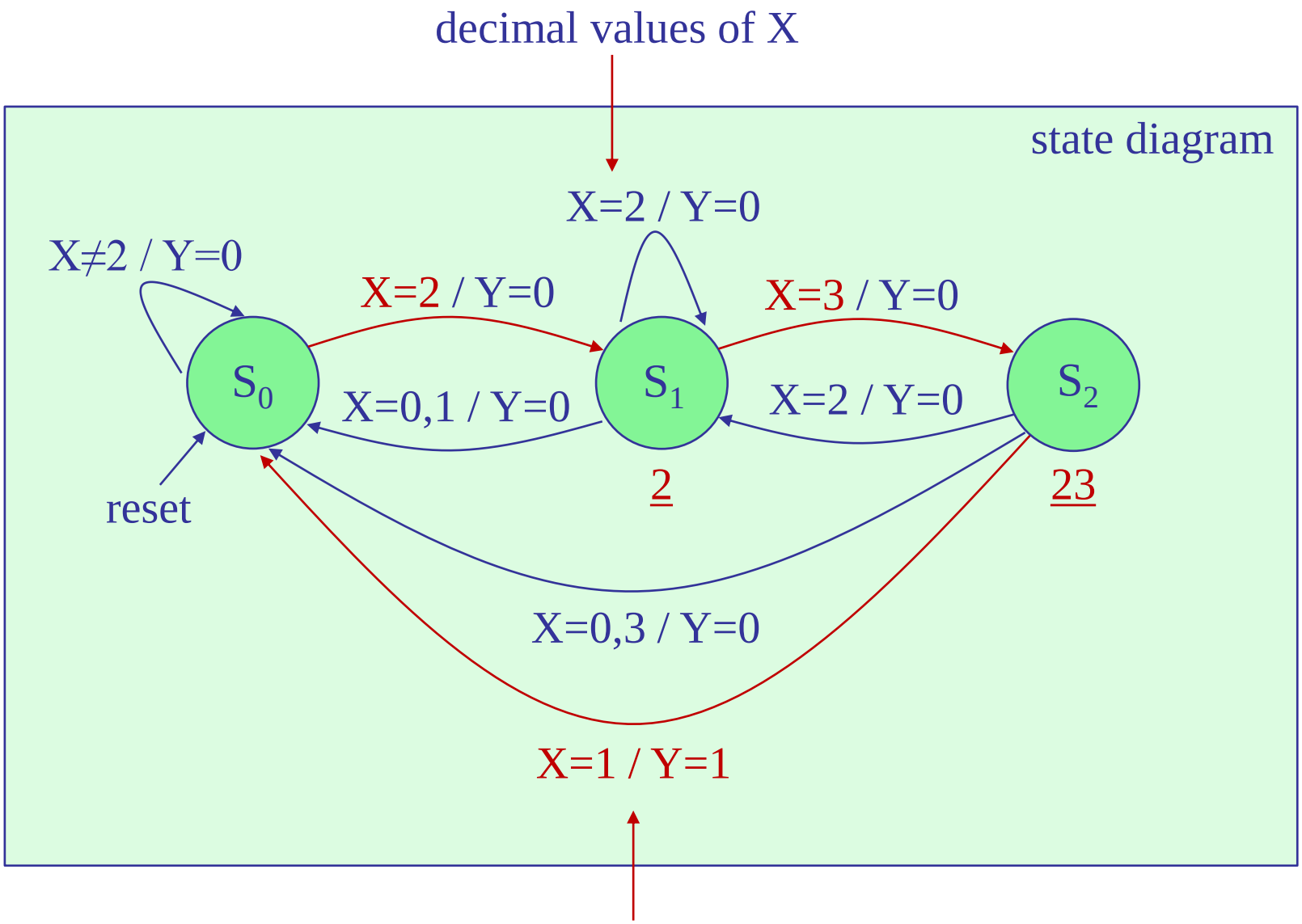
As before, the keypad accepts as input X only the decimal digits, 0,1,2,3, and converts them to binary through an A/D converter, that is, each decimal X = 0,1,2,3 is represented by the bits, X_1X_0 = 00, 01, 10, 11, so that there are two binary inputs to the FSM.

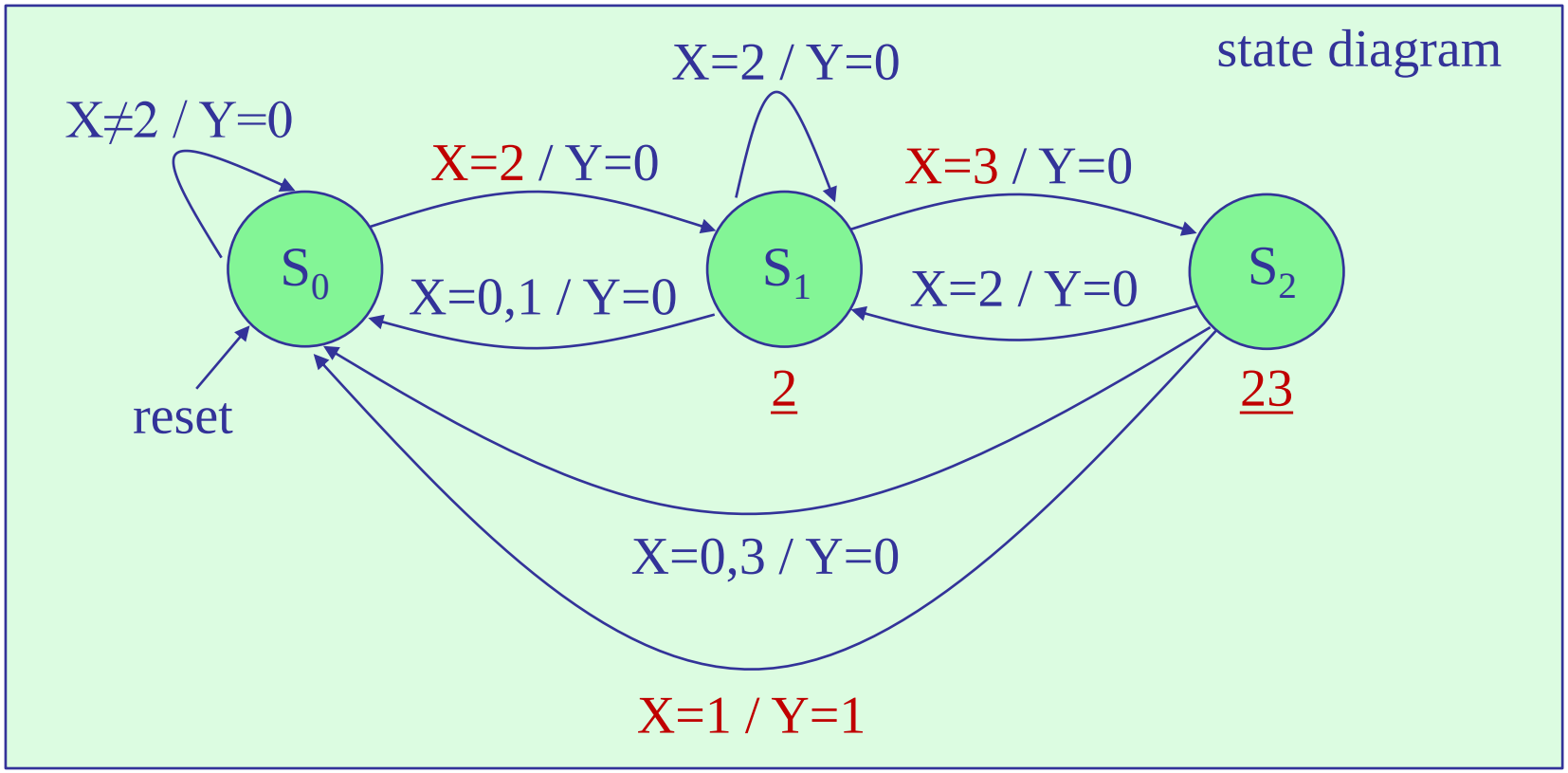
As the user enters a sequence of decimal numbers X into the keypad, the FSM should generate an "unlock" signal, Y=1, whenever the previous two decimal inputs were, X=2,3, and the current input is, X=1. Otherwise, the system should remain locked at, Y=0. Because the output Y depends on the input X, this will be a Mealy FSM.

Based on the above description, determine: (a) the state diagram, (b) state table, (c) next-state and output equations, (d) build a realization using D flip-flops, and (e) simulate the operation of the system by sending in the following test sequence of decimal inputs and verifying the expected outputs:

X = [1 0 2 3 1 0 1 1 2 3 1 2 2 3 1 0]

Y = [0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0]


decimal values of X
state diagram
X=2 / Y=0
X≠2 / Y=0
X=2 / Y=0
X=3 / Y=0
S0
S1
S2
X=0,1 / Y=0
X=2 / Y=0
reset
2
23
X=0,3 / Y=0
X=1 / Y=1
got 231, which unlocks, and then moves to reset state



state table

present state	next state				output Y			
	X=0	X=1	X=2	X=3	X=0	X=1	X=2	X=3
S_0	S_0	S_0	S_1	S_0	0	0	0	0
S_1	S_0	S_0	S_1	S_2	0	0	0	0
S_2	S_0	S_0	S_1	S_0	0	1	0	0

State encoding. With three states, we need two state variables, say, A,B, and we will use plain binary encoding to represent the states:

	A	B
S_0 ≡	0	0
S_1 ≡	0	1
S_2 ≡	1	0

encoded state table

present state	next states A^{next}, B^{next}				output Y			
A B	X=0	X=1	X=2	X=3	X=0	X=1	X=2	X=3
0 0	0 0	0 0	0 1	0 0	0	0	0	0
0 1	0 0	0 0	0 1	1 0	0	0	0	0
1 0	0 0	0 0	0 1	0 0	0	1	0	0

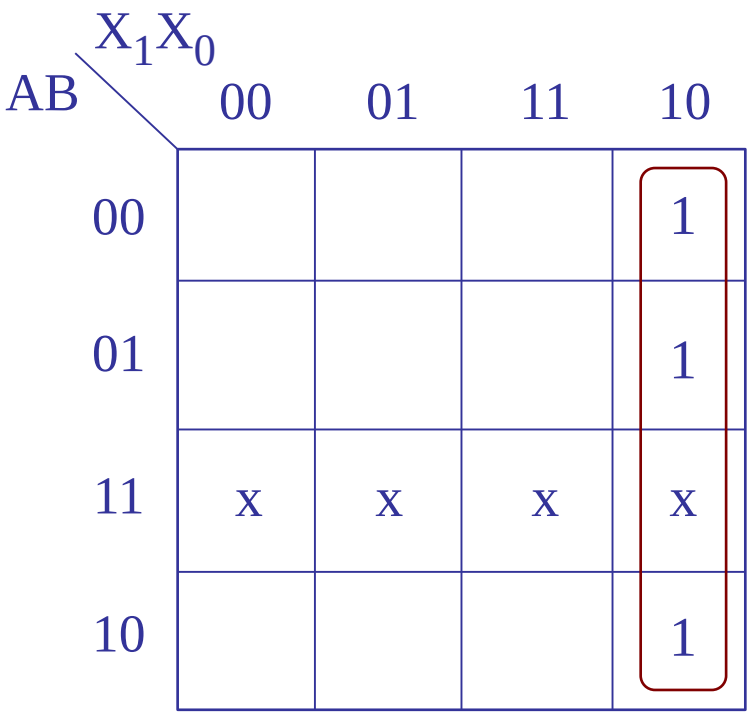
binary X

	A	B	X_1	X_0	A^{next}	B^{next}	Y
S_0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0
	0	0	1	0	0	1	0
	0	0	1	1	0	0	0
S_1	0	1	0	0	0	0	0
	0	1	0	1	0	0	0
	0	1	1	0	0	1	0
	0	1	1	1	1	0	0
S_2	1	0	0	0	0	0	0
	1	0	0	1	0	0	1
	1	0	1	0	0	1	0
	1	0	1	1	0	0	0
	1	1	0	0	x	x	x
	1	1	0	1	x	x	x
	1	1	1	0	x	x	x
	1	1	1	1	x	x	x

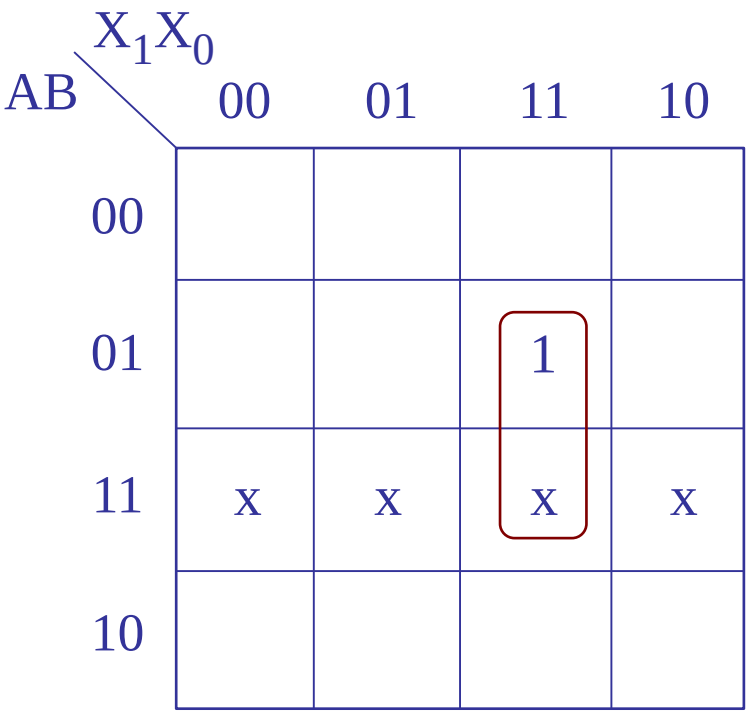
conventional state table

← don't cares

K-map simplifications



$D_B = B^{next} = X_1 X_0'$



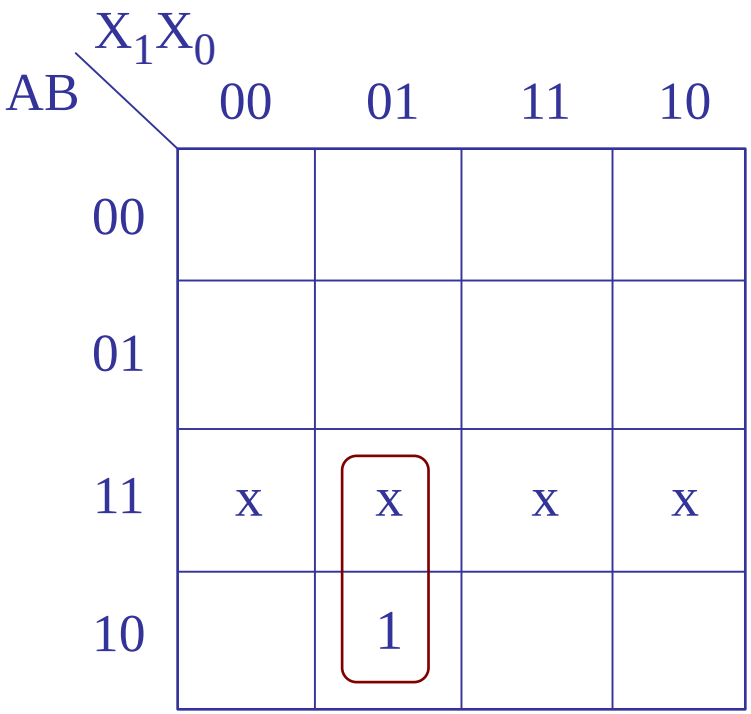
$D_A = A^{next} = B X_1 X_0$

K-map simplifications

$D_A = A^{next} = B\ X_1\ X_0$

$D_B = B^{next} = X_1\ X_0'$

$Y = A\ X_1'\ X_0$



$Y = A\ X_1'\ X_0$

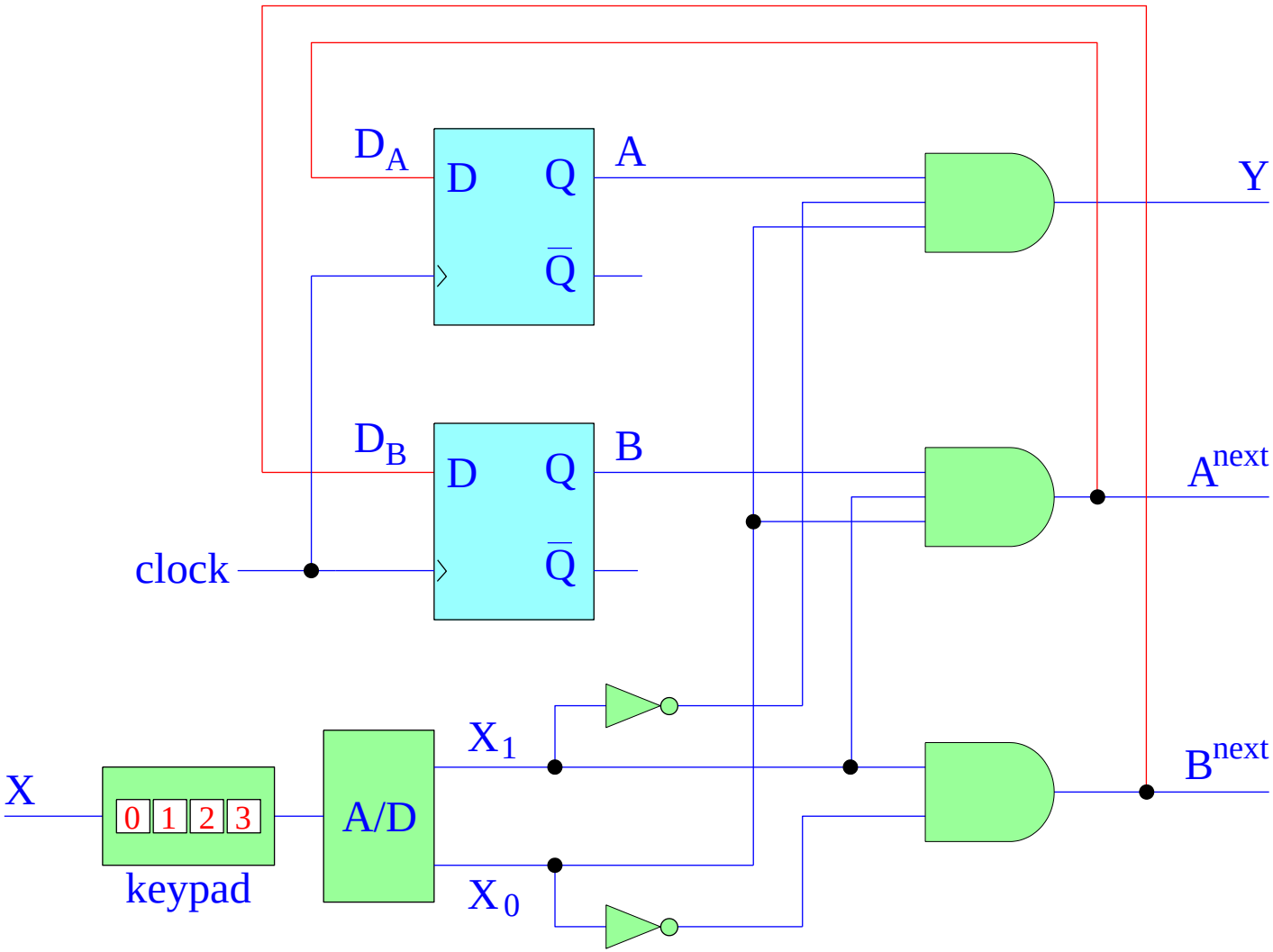
```
[A,B,X1,X0] = a2d(0:15,4);

Anext = B & X1 & X0;

Bnext = X1 & ~X0;

Y = A & ~X1 & X0;

[A,B,X1,X0,Anext,Bnext,Y]
```

$D_A = A^{next} = B\ X_1\ X_0$

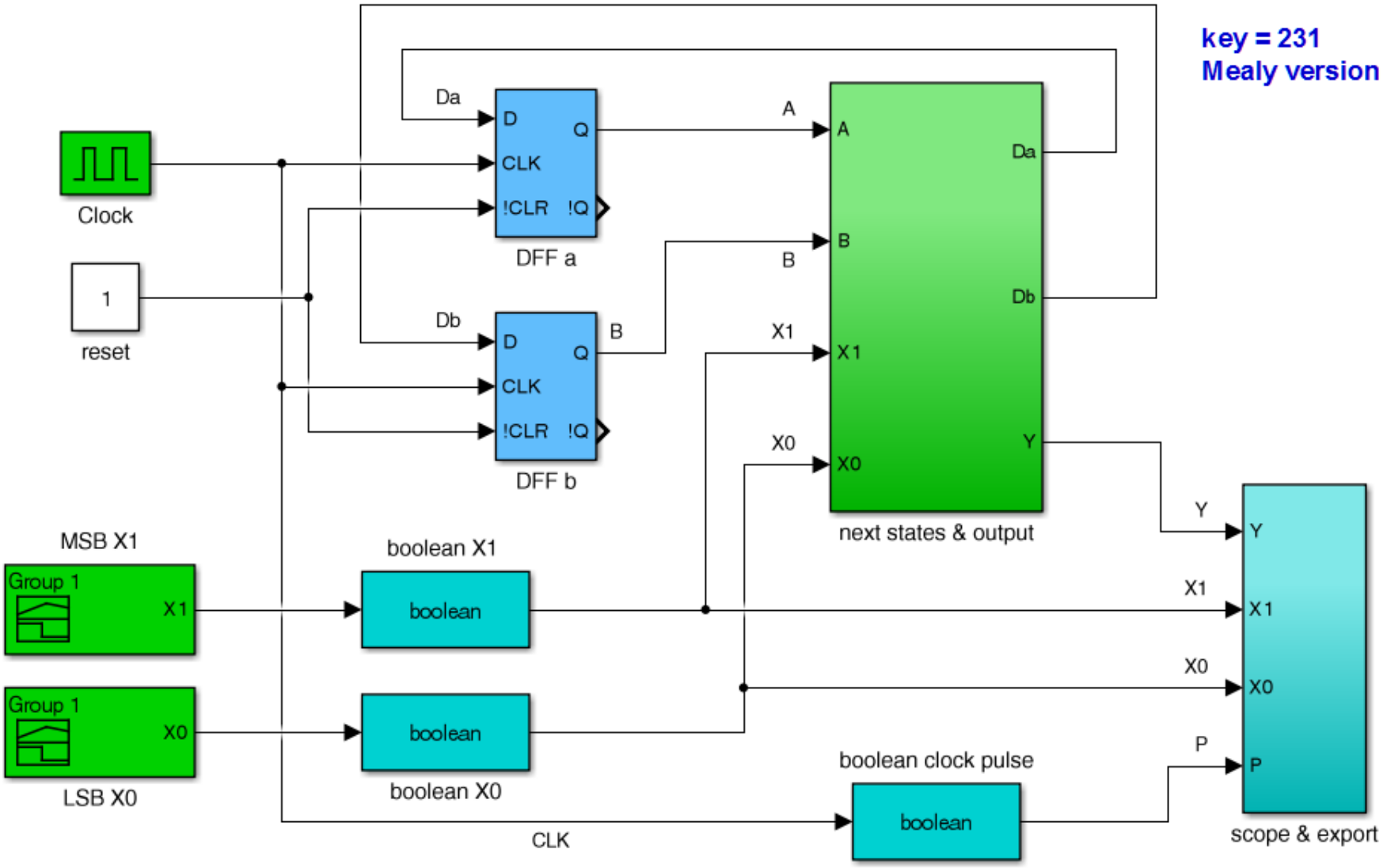
$D_B = B^{next} = X_1\ X_0'$

$Y = A\ X_1'\ X_0$

block diagram realization

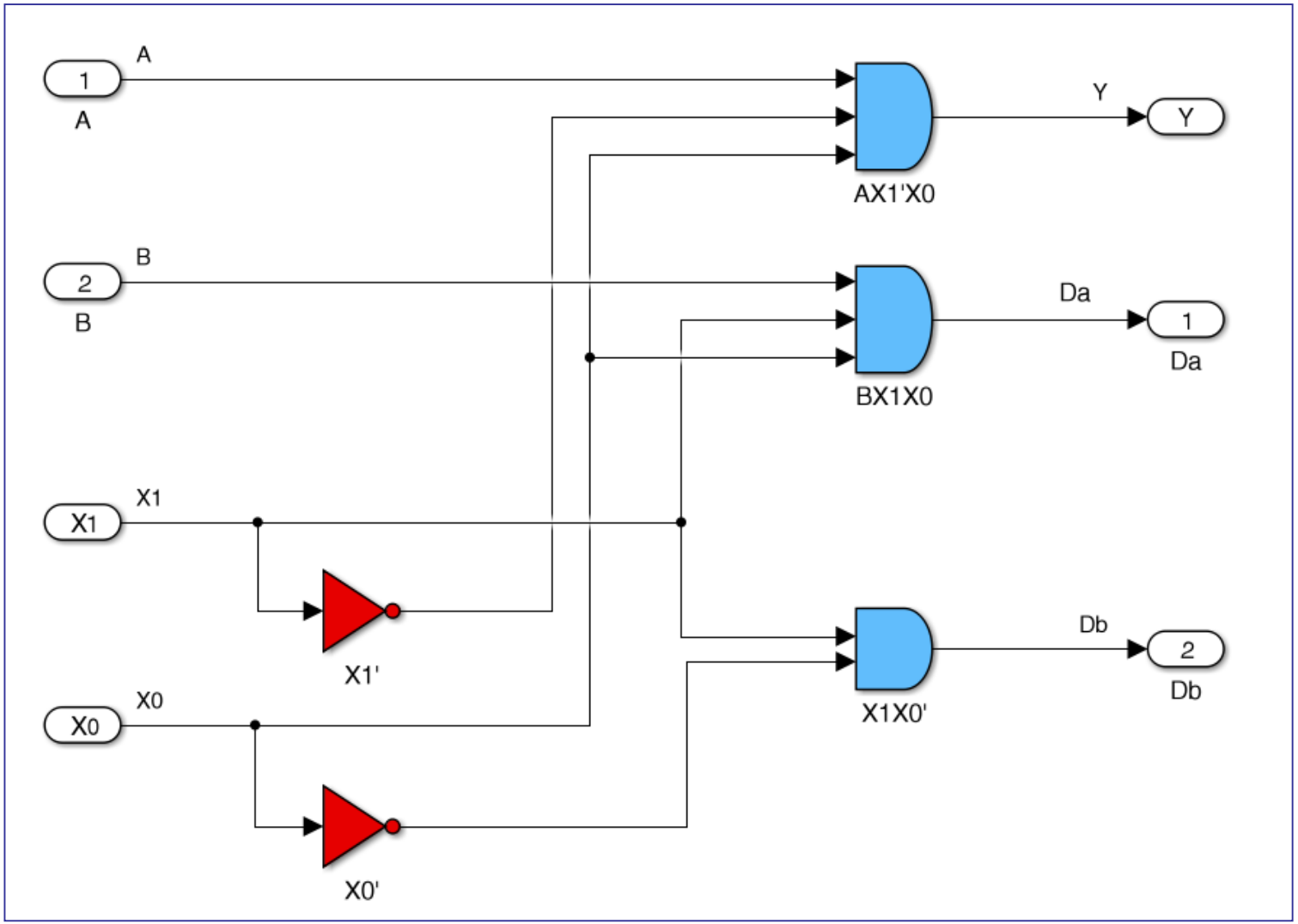


Simulink implementation

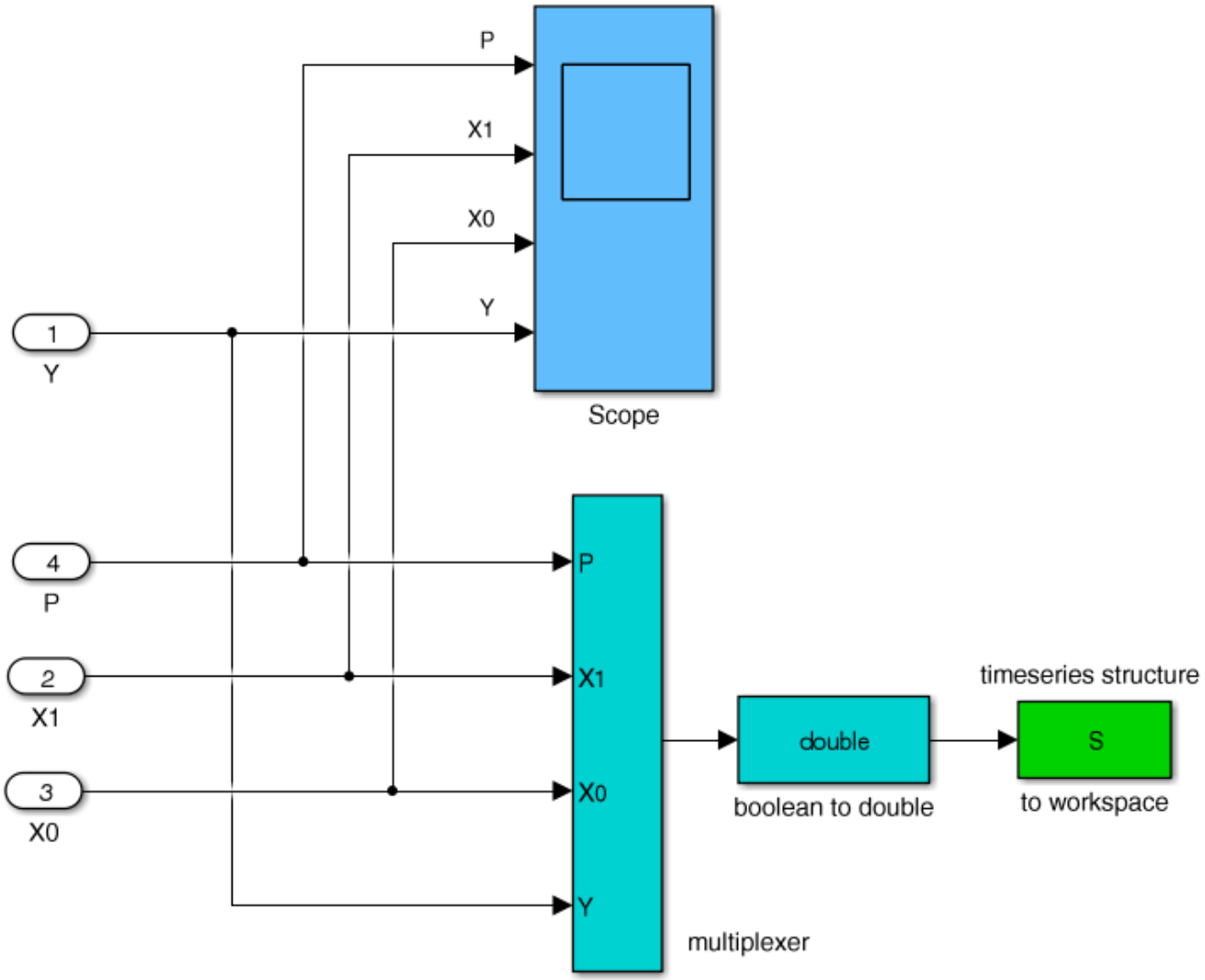


key231bbb.slx

next states & output sub-function



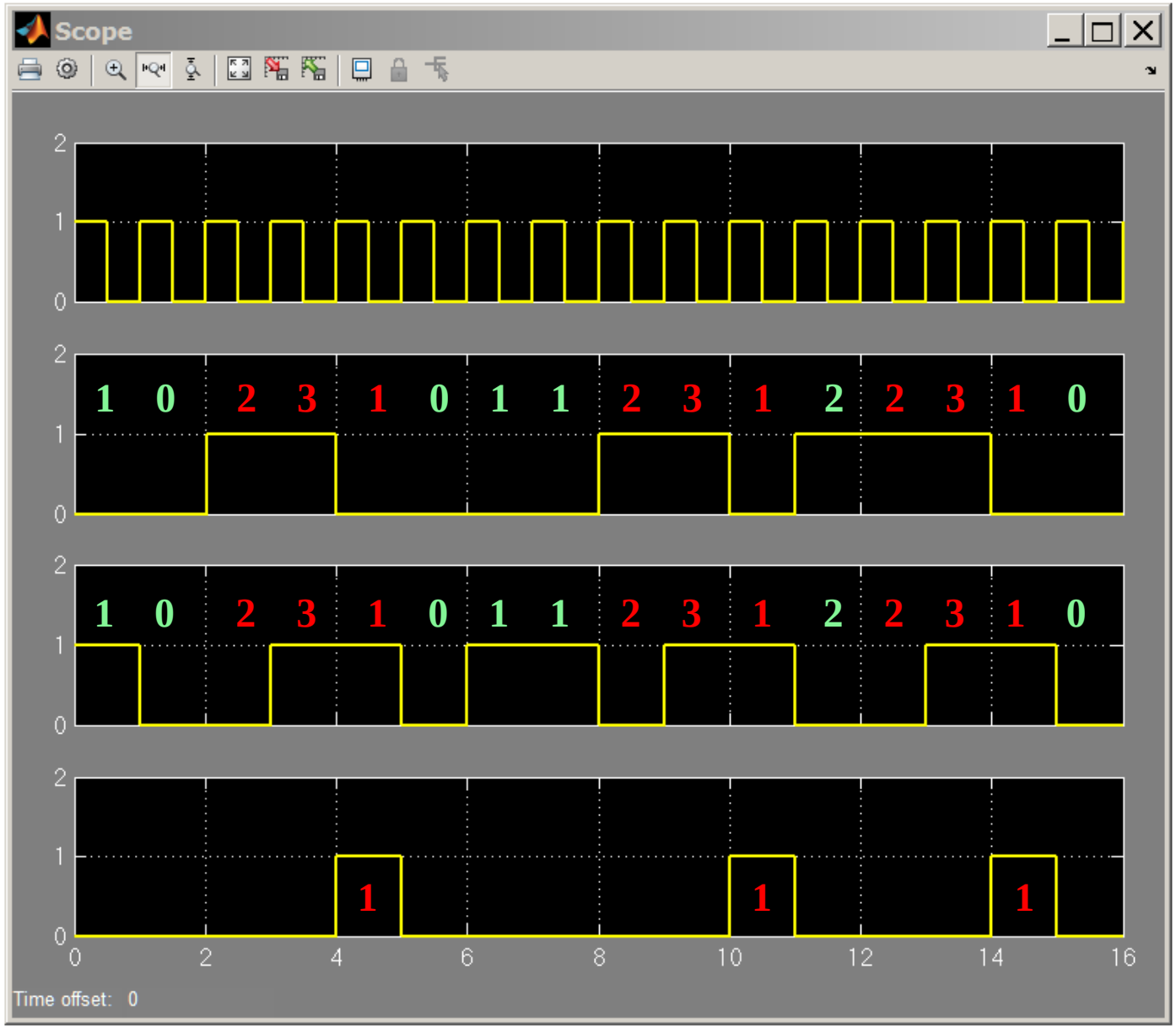
scope & date export sub-function



Example 12

input X = 1 0 2 3 1 0 1 1 2 3 1 2 2 3 1 0

scope display



clock

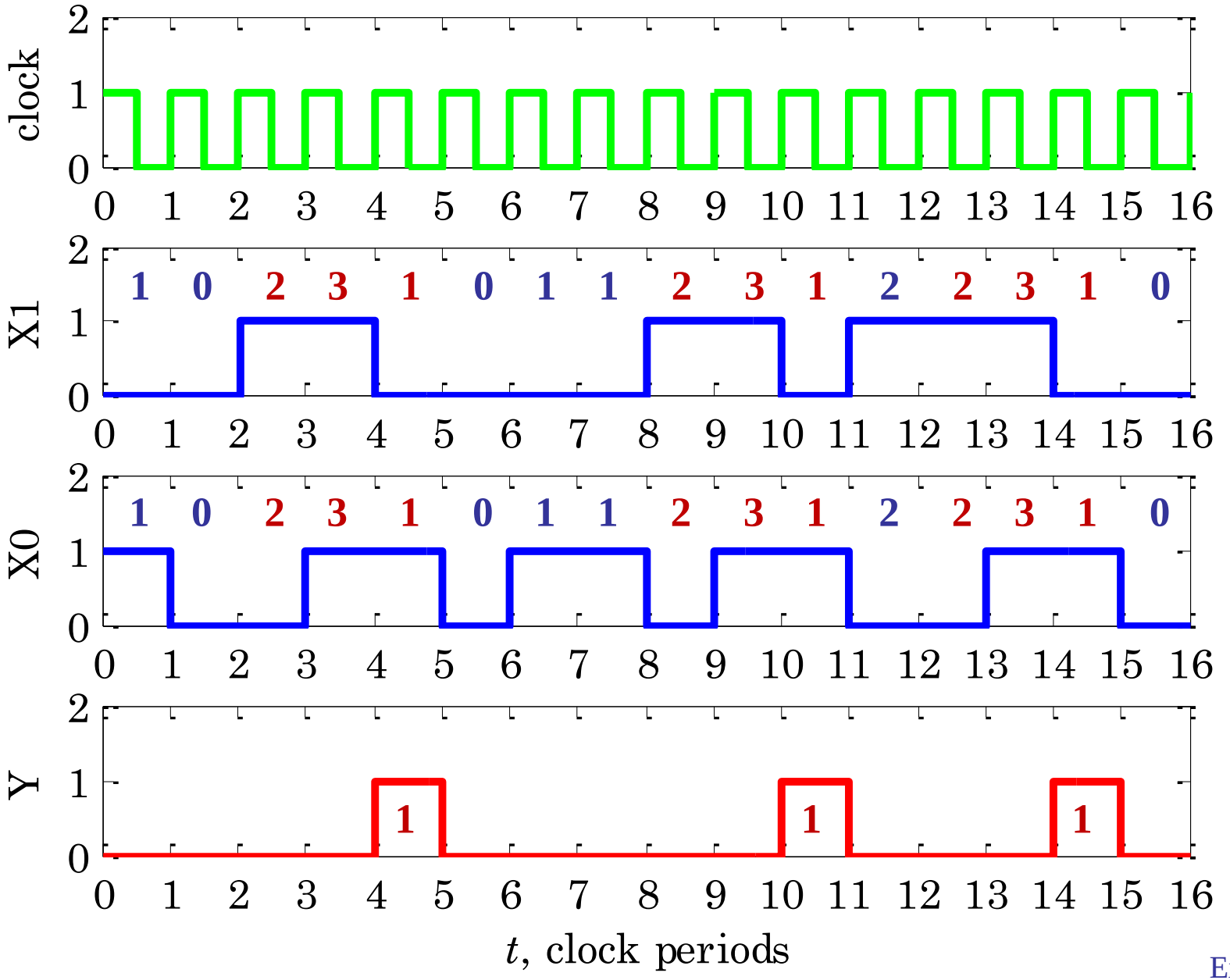
X_1

X_0

Y

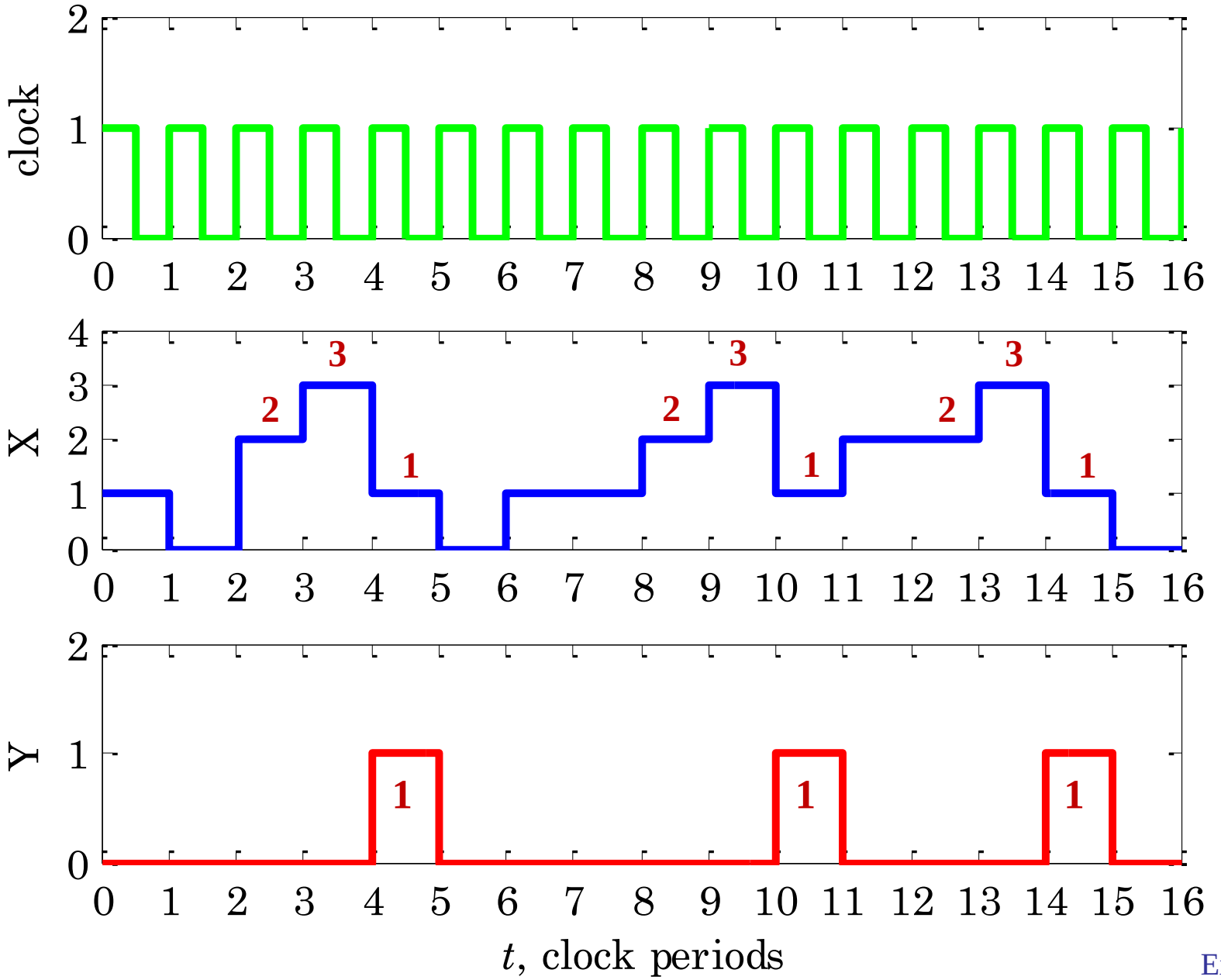
Example 12

input X = 1 0 2 3 1 0 1 1 2 3 1 2 2 3 1 0



Example 12

input X = 1 0 2 3 1 0 1 1 2 3 1 2 2 3 1 0



```
t =  S.time;
P =  S.data(:,1);
X1 = S.data(:,2);
X0 = S.data(:,3);
Y  = S.data(:,4);

set(0,'DefaultAxesFontSize',10);
figure;
subplot(4,1,1); stairs(t,P, 'g-'); yaxis(0,2); ylabel('clock');
subplot(4,1,2); stairs(t,X1,'b-'); yaxis(0,2); ylabel('X1');
subplot(4,1,3); stairs(t,X0,'b-'); yaxis(0,2); ylabel('X0');
subplot(4,1,4); stairs(t,Y, 'r-'); yaxis(0,2); ylabel('Y');
xlabel('{\itt}, clock periods')

X = d2a([X1,X0],+1);    % decimal version of [X1,X0]

set(0,'DefaultAxesFontSize',10);
figure;
subplot(3,1,1); stairs(t,P, 'g-'); yaxis(0,2); ylabel('clock');
subplot(3,1,2); stairs(t,X,'b-'); yaxis(0,4); ylabel('X');
subplot(3,1,3); stairs(t,Y,'r-'); yaxis(0,2); ylabel('Y');
xlabel('{\itt}, clock periods')
```

Example 13 – Vending machine. We consider the design of a Moore FSM for a simple vending machine that accepts nickels or dimes as inputs, deposited one at a time, and dispenses some candy that costs 15 cents, and also returns some change, if necessary [cf. Maxfield].

Five states will be needed:

S_{00} - 0-cent state, also the reset state (entered upon power-up)

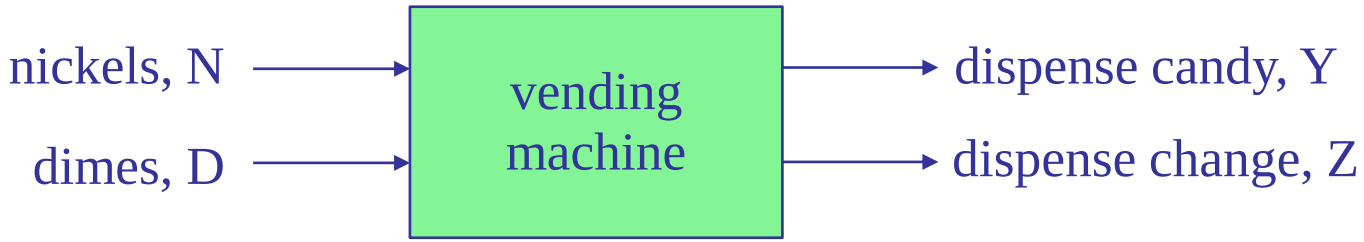
S_{05} - 5-cent state

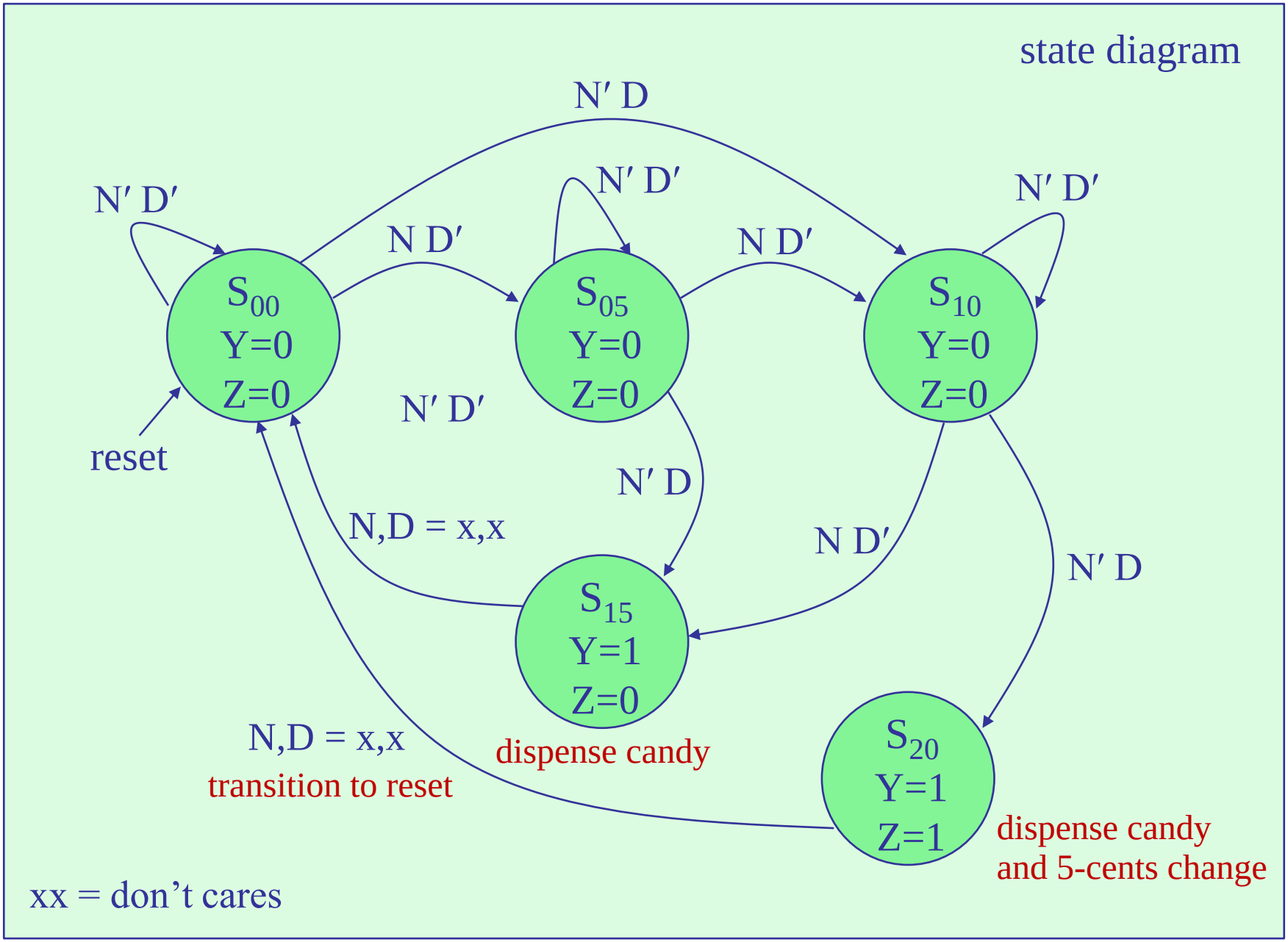
S_{10} - 10-cent state

S_{15} - 15-cent state

S_{20} - 20-cent state

The state diagram is shown on the next page using the arrow convention explained on p. 59 that displays input expressions instead of values.





note: the possible input combinations are N D′, N′ D, N′ D′

main part of state table – with don't care inputs at S_{15} and S_{20}

present	A	B	C	N	D	next	A	B	C	Y	Z
S_{00}	1	0	0	0	0	S_{00}	1	0	0	0	0
S_{00}	1	0	0	1	0	S_{05}	1	0	1	0	0
S_{00}	1	0	0	0	1	S_{10}	0	0	0	0	0
S_{05}	1	0	1	0	0	S_{05}	1	0	1	0	0
S_{05}	1	0	1	1	0	S_{10}	0	0	0	0	0
S_{05}	1	0	1	0	1	S_{15}	0	0	1	0	0
S_{10}	0	0	0	0	0	S_{10}	0	0	0	0	0
S_{10}	0	0	0	1	0	S_{15}	0	0	1	0	0
S_{10}	0	0	0	0	1	S_{20}	0	1	1	0	0
S_{15}	0	0	1	0	0	S_{00}	1	0	0	1	0
S_{15}	0	0	1	1	0	S_{00}	1	0	0	1	0
S_{15}	0	0	1	0	1	S_{00}	1	0	0	1	0
S_{20}	0	1	1	0	0	S_{00}	1	0	0	1	1
S_{20}	0	1	1	1	0	S_{00}	1	0	0	1	1
S_{20}	0	1	1	0	1	S_{00}	1	0	0	1	1

state encoding

	A	B	C
S_{00}	1	0	0
S_{05}	1	0	1
S_{10}	0	0	0
S_{15}	0	0	1
S_{20}	0	1	1

rest of state table with unsused or don't care states

A	B	C	N	D	A	B	C	Y	Z
present					next				
1	0	0	1	1	0	0	1	0	0
1	0	1	1	1	0	0	0	0	0
0	0	0	1	1	0	1	1	0	0
0	0	1	1	1	1	0	0	1	0
0	1	1	1	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1
1	1	0	1	1	0	0	1	0	1
1	1	1	1	1	0	0	0	0	1
0	1	0	0	0	0	0	0	0	1
0	1	0	0	1	0	1	1	0	1
0	1	0	1	0	0	0	1	0	1
1	1	0	0	0	1	0	0	0	1
1	1	0	0	1	0	0	0	0	1
1	1	0	1	0	1	0	1	0	1
1	1	1	0	0	1	0	1	0	1
1	1	1	0	1	0	0	1	0	1
1	1	1	1	0	0	0	0	0	1

17 rows

some used states,
some unused states,
but, the inputs, N=1, D=1,
are not allowed

all are unused states

next-state & output equations are obtained from the state table after taking advantage of all unused don't care entries.

next states & output equations

$$A^{next} = A' C + A N' D' + A C' D'$$

$$B^{next} = A' C' D$$

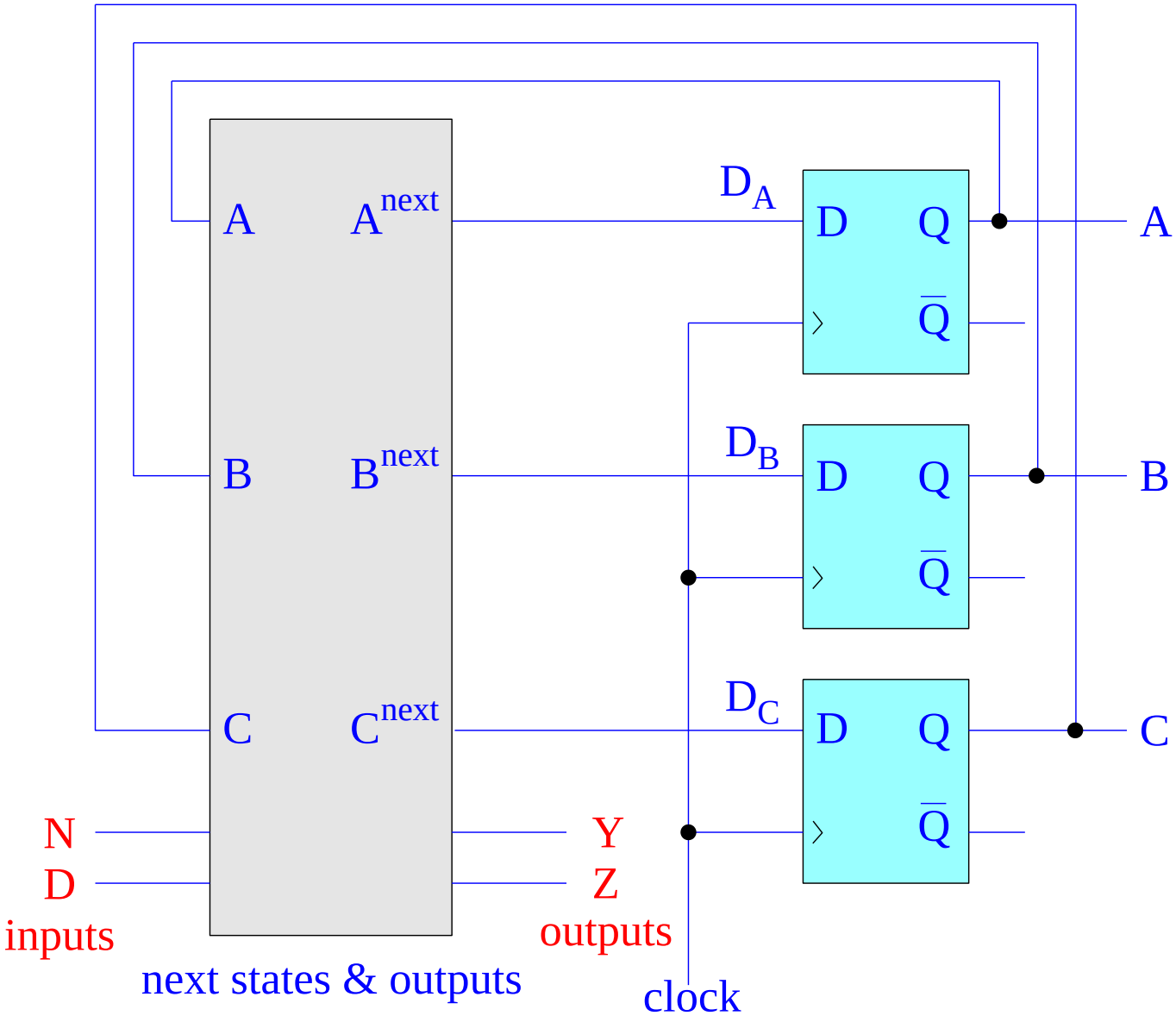
$$C^{next} = A' C' D + A C N' + C' N$$

$$Y = A' C$$

$$Z = B$$

MATLAB code
for producing the main part of the stable on p.171 ⟶
and the rest of the state table on p.172

```
% MATLAB code used to generate the state table

% generate main part of state table
n = [16 18 17 20 22 21 0 2 1 4 6 5 12 14 13];

% generate rest of state table
% n = [19 23 3 7 15 11 27 31 8 9 10 24 25 26 28 29 30];

[A,B,C,N,D] = a2d(n,5);          % generate states & inputs

Anext = (~A & C) | (A & ~N & ~D) | (A & ~C & ~D);

Bnext = ~A & ~C & D;

Cnext = (~A & ~C & D) | (A & C & ~N) | (~C & N);

Y = ~A & C;

Z = B;

[A,B,C,N,D,Anext,Bnext,Cnext,Y,Z]    % print state table
```


A
B
C
A^next
B^next
C^next
D_A
D_B
D_C
D
Q
Q̄
A
B
C
N
D
inputs
Y
Z
outputs
next states & outputs
clock

Example 14 – Elevator controller. [this is a simplified version of a DLD lab from earlier years.] Consider the design of a simple elevator controller for a three-story building. There are three states F_1, F_2, F_3 defined so that,

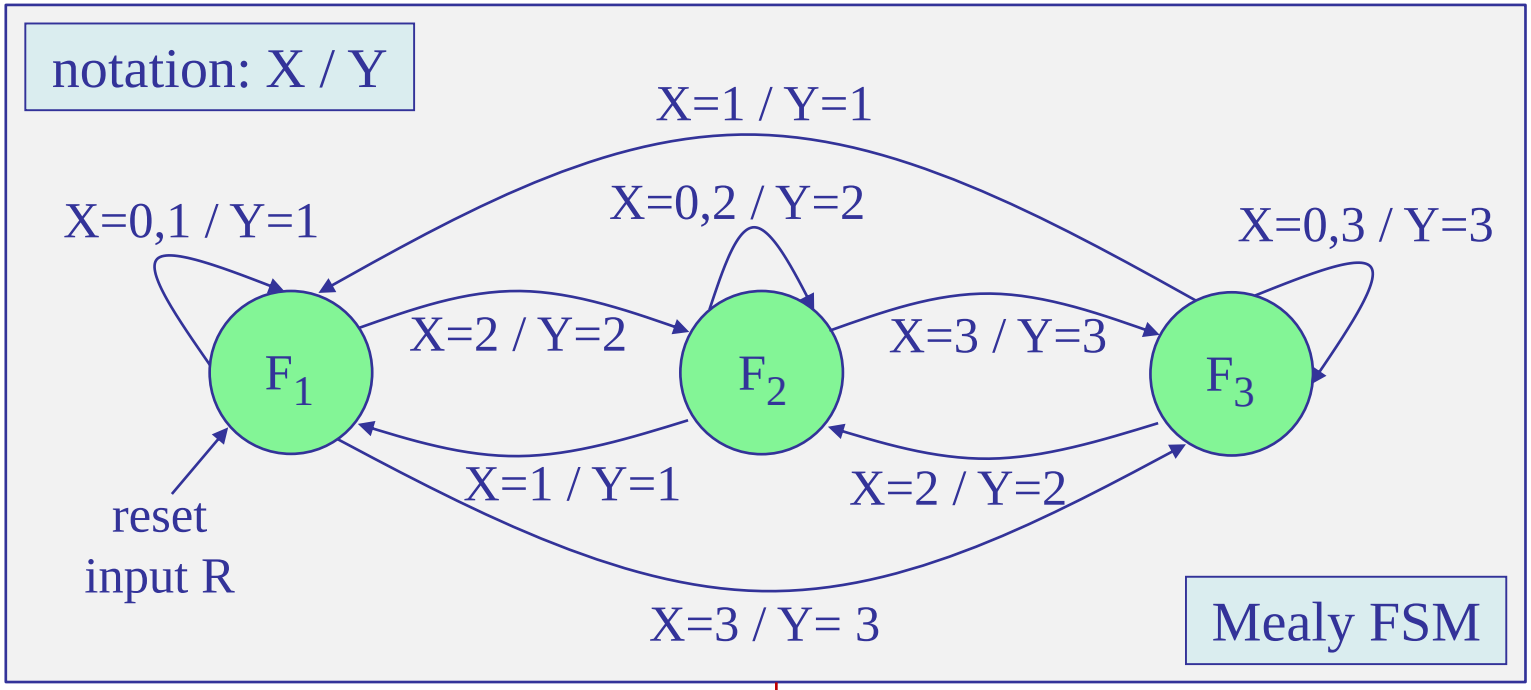
F_1 = elevator is at floor 1
F_2 = elevator is at floor 2
F_3 = elevator is at floor 3

The input to the controller is the (decimal) variable X, defined so that,

X = 0, no request (i.e., stay on the current floor)
X = 1, move to floor 1
X = 2, move to floor 2
X = 3, move to floor 3

There is also a reset input R, such that R=1 will cause the elevator to move to floor 1 from any floor, regardless of X. Otherwise, R=0 has no effect, and the input X will determine the action.

There is also an output Y taking the (decimal) values Y=1, 2, 3, that are generated when the elevator is moving to floors 1, 2, 3, respectively.



state diagram

present state	next state					output Y			
	R=0	X=0	X=1	X=2	X=3	X=0	X=1	X=2	X=3
F_1		F_1	F_1	F_2	F_3	1	1	2	3
F_2		F_2	F_1	F_2	F_3	2	1	2	3
F_3		F_3	F_1	F_2	F_3	3	1	2	3
	R=1	X=0	X=1	X=2	X=3	X=0	X=1	X=2	X=3
F_1, F_2, F_3		F_1	F_1	F_1	F_1	1	1	1	1

state table

input & output encoding	
X = X_1X_0	Y = Y_1Y_0
0 = 0 0	
1 = 0 1	1 = 0 1
2 = 1 0	2 = 1 0
3 = 1 1	3 = 1 1

state encoding		
F	A B	
	0 0	unused
F1	0 1	
F2	1 0	
F3	1 1	

three states require two D flip-flops D_A & D_B

binary ordering

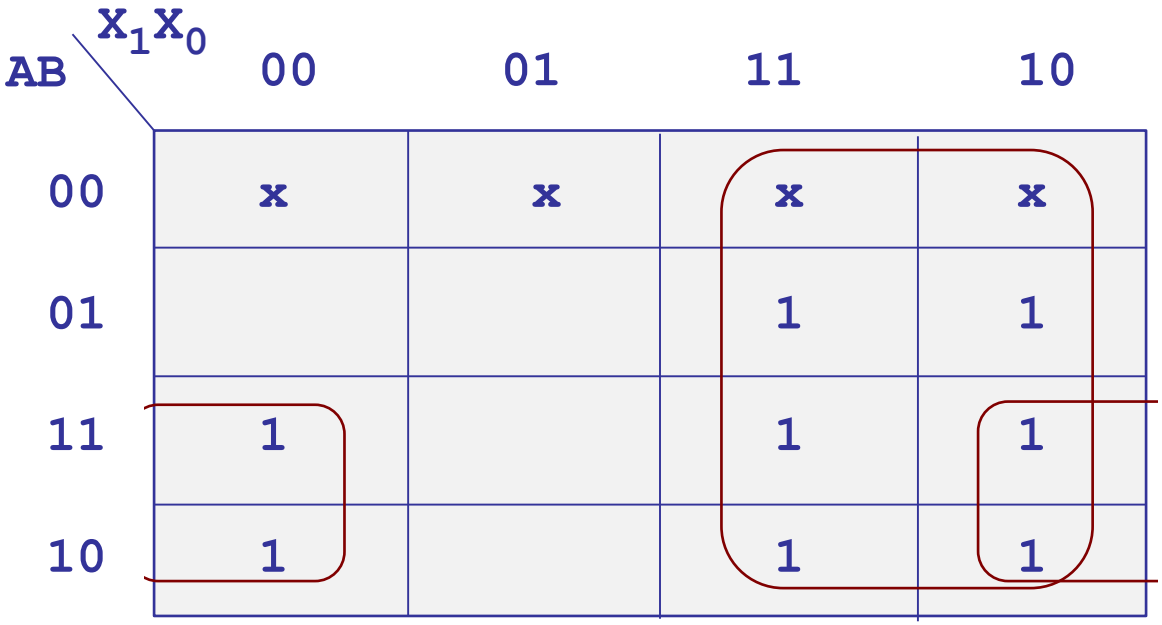
K-maps for A^{next}, B^{next}

K-maps for Y_1, Y_0 are similar

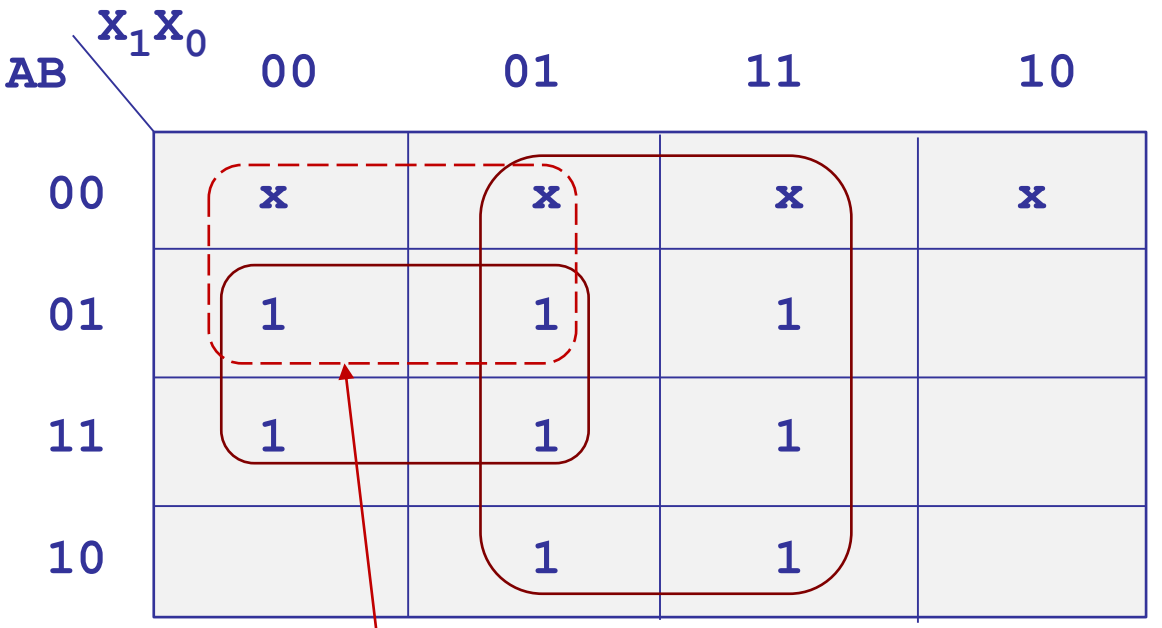
AB \ X_1X_0	00	01	11	10
00	xx	xx	xx	xx
F_1 01	F_1 01	F_1 01	F_3 11	F_2 10
F_3 11	F_3 11	F_1 01	F_3 11	F_2 10
F_2 10	F_2 10	F_1 01	F_3 11	F_2 10

don't cares

K-maps



$A^{next} = X_1 + X_0'A$



$Y_1 = A^{next}$

$Y_0 = B^{next}$

$Y = 2Y_1 + Y_0$ (decimal)

$X = 2X_1 + X_0$ (decimal)

$B^{next} = X_0 + X_1' (A + B')$

adding the reset input R and the output Y (in decimal)

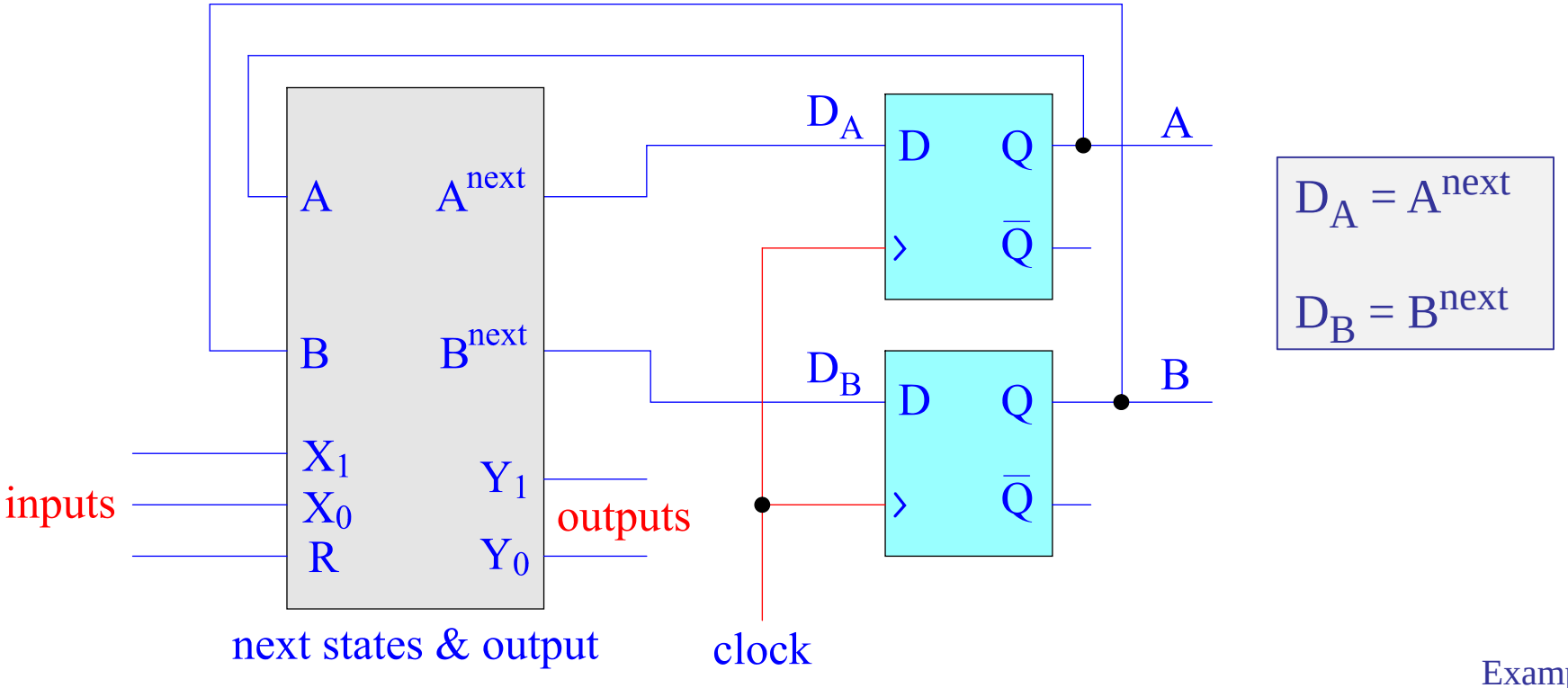
$A^{next} = R'(X_1 + X_0'A)$

$B^{next} = R + X_0 + X_1'(A + B')$

$Y = 2A^{next} + B^{next}$ (decimal)

resets to state $F_1 = AB = 01$, if R=1, and normal operation, if R=0

requires two D flip-flops



$D_A = A^{next}$

$D_B = B^{next}$

computed state table

$A^{next} = R'(X_1 + X_0'A)$

$B^{next} = R + X_0 + X_1'(A + B')$

$Y = 2A^{next} + B^{next}$

$X = 2X_1 + X_0$

decimal

R	X_1	X_0	A	B	D_A	D_B	X	Y
0	0	0	0	0	0	1	0	1
0	0	0	0	1	0	1	0	1
0	0	0	1	0	1	0	0	2
0	0	0	1	1	1	1	0	3
0	0	1	0	0	0	1	1	1
0	0	1	0	1	0	1	1	1
0	0	1	1	0	0	1	1	1
0	0	1	1	1	0	1	1	1
0	1	0	0	0	1	0	2	2
0	1	0	0	1	1	0	2	2
0	1	0	1	0	1	0	2	2
0	1	0	1	1	1	0	2	2
0	1	1	0	0	1	1	3	3
0	1	1	0	1	1	1	3	3
0	1	1	1	0	1	1	3	3
0	1	1	1	1	1	1	3	3

decimal

R	X_1	X_0	A	B	D_A	D_B	X	Y
1	0	0	0	0	0	1	0	1
1	0	0	0	1	0	1	0	1
1	0	0	1	0	0	1	0	1
1	0	0	1	1	0	1	0	1
1	0	1	0	0	0	1	1	1
1	0	1	0	1	0	1	1	1
1	0	1	1	0	0	1	1	1
1	0	1	1	1	0	1	1	1
1	1	0	0	0	0	1	2	1
1	1	0	0	1	0	1	2	1
1	1	0	1	0	0	1	2	1
1	1	0	1	1	0	1	2	1
1	1	1	0	0	0	1	3	1
1	1	1	0	1	0	1	3	1
1	1	1	1	0	0	1	3	1
1	1	1	1	1	0	1	3	1

state A,B=0,0 does not appear as a next state, so it can be removed

computed state table

$A^{next} = R'(X_1 + X_0'A)$

$B^{next} = R + X_0 + X_1'(A + B')$

$Y = 2A^{next} + B^{next}$

$X = 2X_1 + X_0$

reduced state table

R	X_1	X_0	A	B	D_A	D_B	X	Y
0	0	0	0	1	0	1	0	1
0	0	0	1	0	1	0	0	2
0	0	0	1	1	1	1	0	3
0	0	1	0	1	0	1	1	1
0	0	1	1	0	0	1	1	1
0	0	1	1	1	0	1	1	1
0	1	0	0	1	1	0	2	2
0	1	0	1	0	1	0	2	2
0	1	0	1	1	1	0	2	2
0	1	1	0	1	1	1	3	3
0	1	1	1	0	1	1	3	3
0	1	1	1	1	1	1	3	3

R	X_1	X_0	A	B	D_A	D_B	X	Y
1	0	0	0	1	0	1	0	1
1	0	0	1	0	0	1	0	1
1	0	0	1	1	0	1	0	1
1	0	1	0	1	0	1	1	1
1	0	1	1	0	0	1	1	1
1	0	1	1	1	0	1	1	1
1	1	0	0	1	0	1	2	1
1	1	0	1	0	0	1	2	1
1	1	0	1	1	0	1	2	1
1	1	1	0	1	0	1	3	1
1	1	1	1	0	0	1	3	1
1	1	1	1	1	0	1	3	1

next, replace states by their symbolic names →

symbolic state table

$$A^{next} = R'(X_1 + X_0'A)$$

$$B^{next} = R + X_0 + X_1'(A + B')$$

$$Y = 2A^{next} + B^{next}$$

$$X = 2X_1 + X_0$$

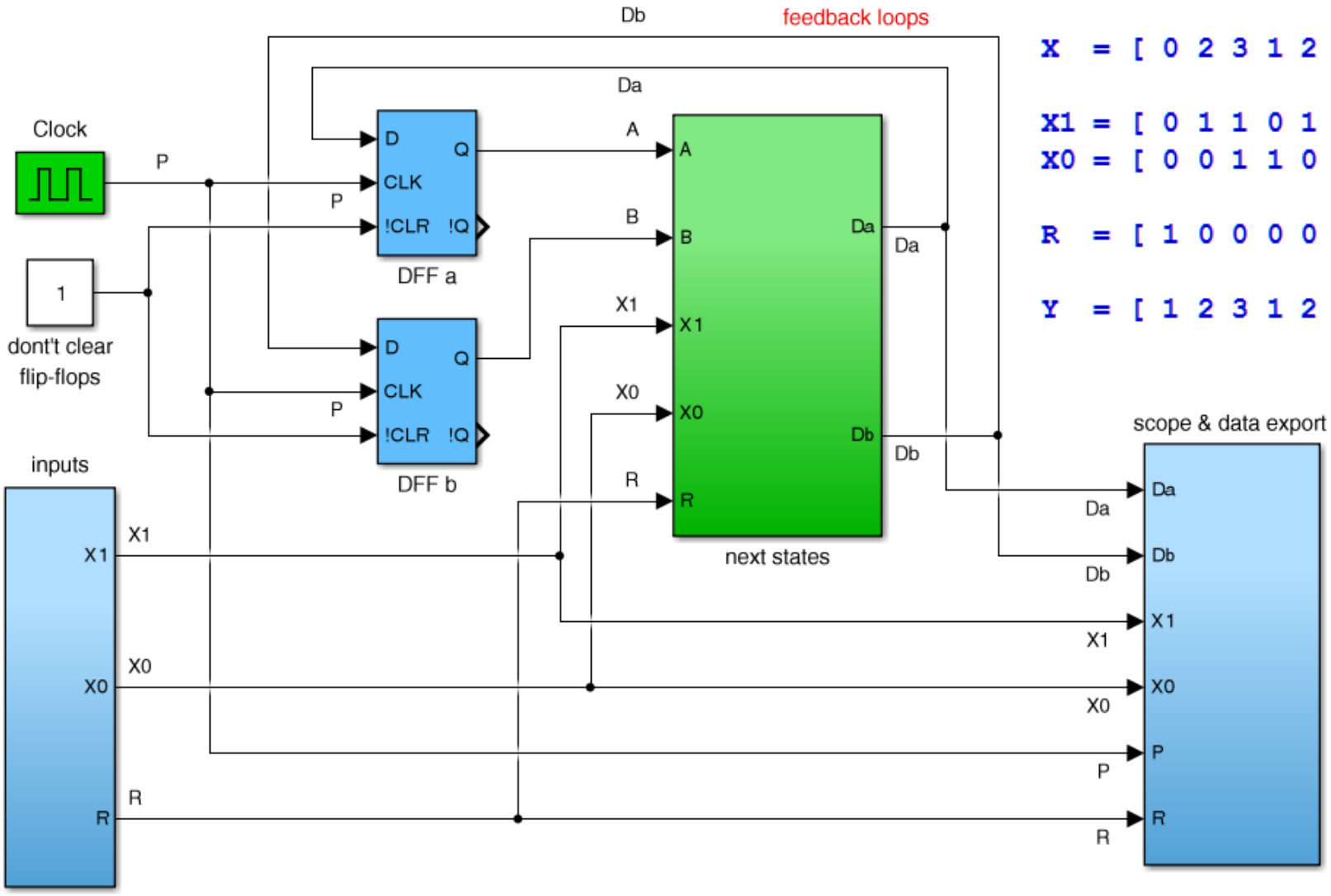
R	X_1	X_0	A	B	D_A	D_B	X	Y
0	0	0	F1		F1		0	1
0	0	0	F2		F2		0	2
0	0	0	F3		F3		0	3
0	0	1	F1		F1		1	1
0	0	1	F2		F1		1	1
0	0	1	F3		F1		1	1
0	1	0	F1		F2		2	2
0	1	0	F2		F2		2	2
0	1	0	F3		F2		2	2
0	1	1	F1		F3		3	3
0	1	1	F2		F3		3	3
0	1	1	F3		F3		3	3

equivalent to the table on p. 177

R	X_1	X_0	A	B	D_A	D_B	X	Y
1	0	0	F1		F1		0	1
1	0	0	F2		F1		0	1
1	0	0	F3		F1		0	1
1	0	1	F1		F1		1	1
1	0	1	F2		F1		1	1
1	0	1	F3		F1		1	1
1	1	0	F1		F1		2	1
1	1	0	F2		F1		2	1
1	1	0	F3		F1		2	1
1	1	1	F1		F1		3	1
1	1	1	F2		F1		3	1
1	1	1	F3		F1		3	1
1	x	x	Fx		F1		x	1

compact form

Simulink implementation

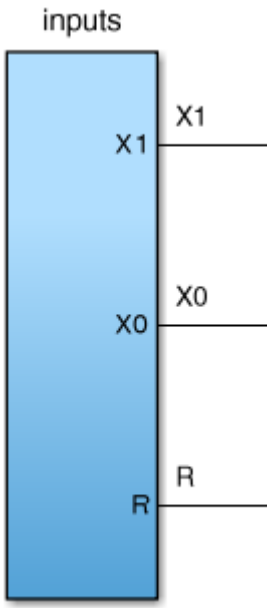


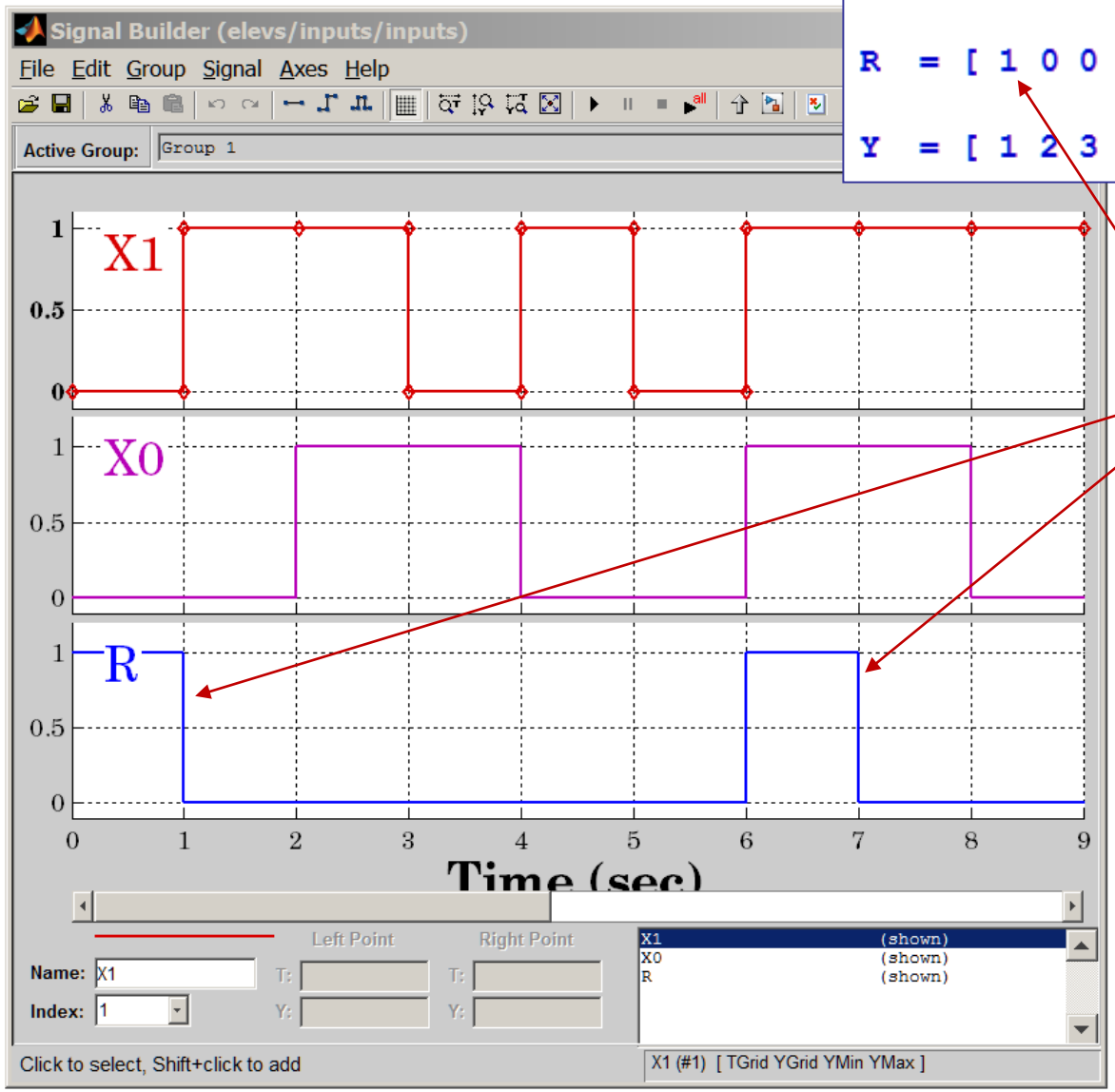

```
X  = [ 0 2 3 1 2 0 3 3 2 ]

X1 = [ 0 1 1 0 1 0 1 1 1 ]
X0 = [ 0 0 1 1 0 0 1 1 0 ]

R  = [ 1 0 0 0 0 0 1 0 0 ]

Y  = [ 1 2 3 1 2 2 1 3 2 ]
```

file: elevs.slx

Simulink implementation input subfunction

```
X  = [ 0 2 3 1 2 0 3 3 2 ]

X1 = [ 0 1 1 0 1 0 1 1 1 ]
X0 = [ 0 0 1 1 0 0 1 1 0 ]

R  = [ 1 0 0 0 0 0 1 0 0 ]

Y  = [ 1 2 3 1 2 2 1 3 2 ]
```



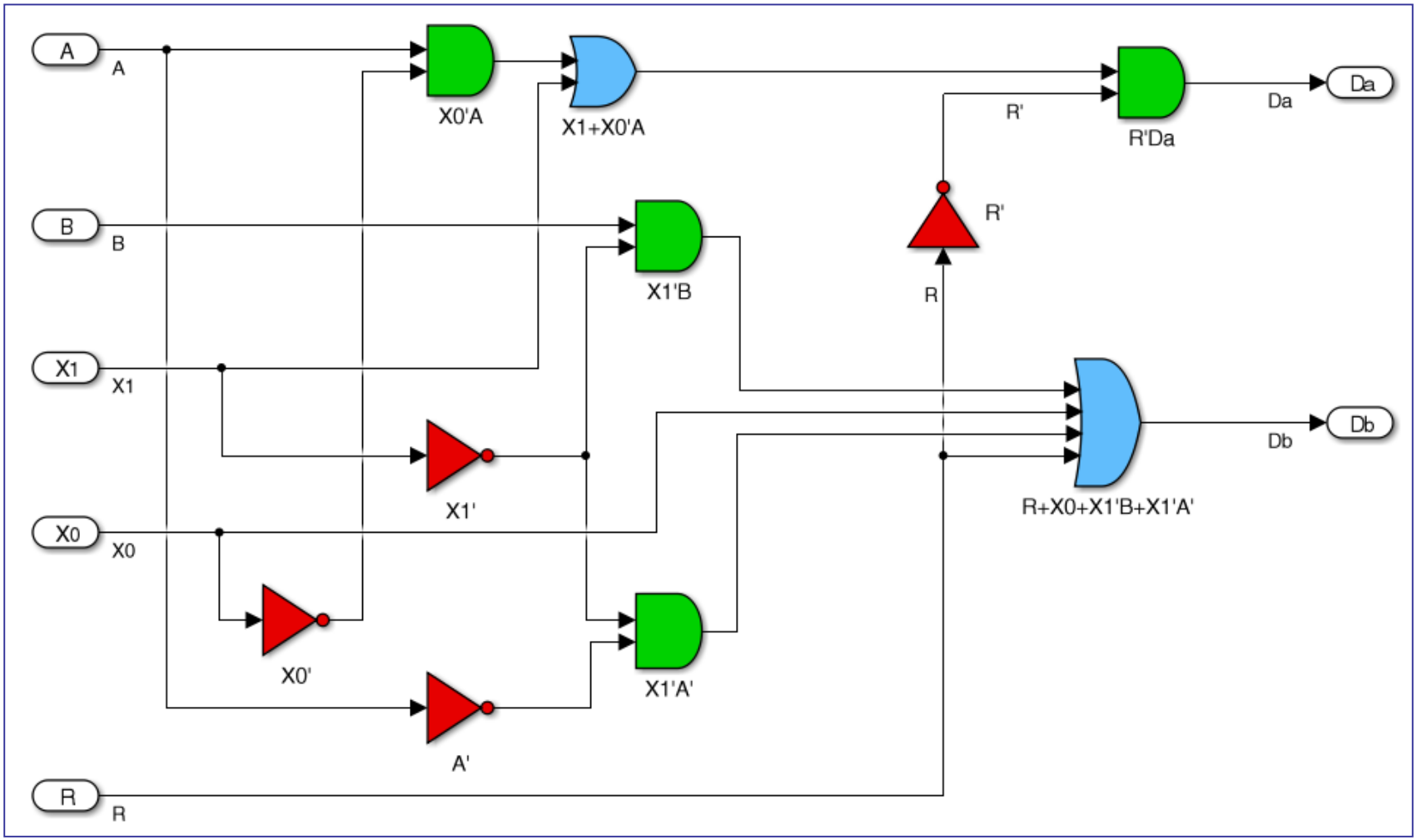


reset

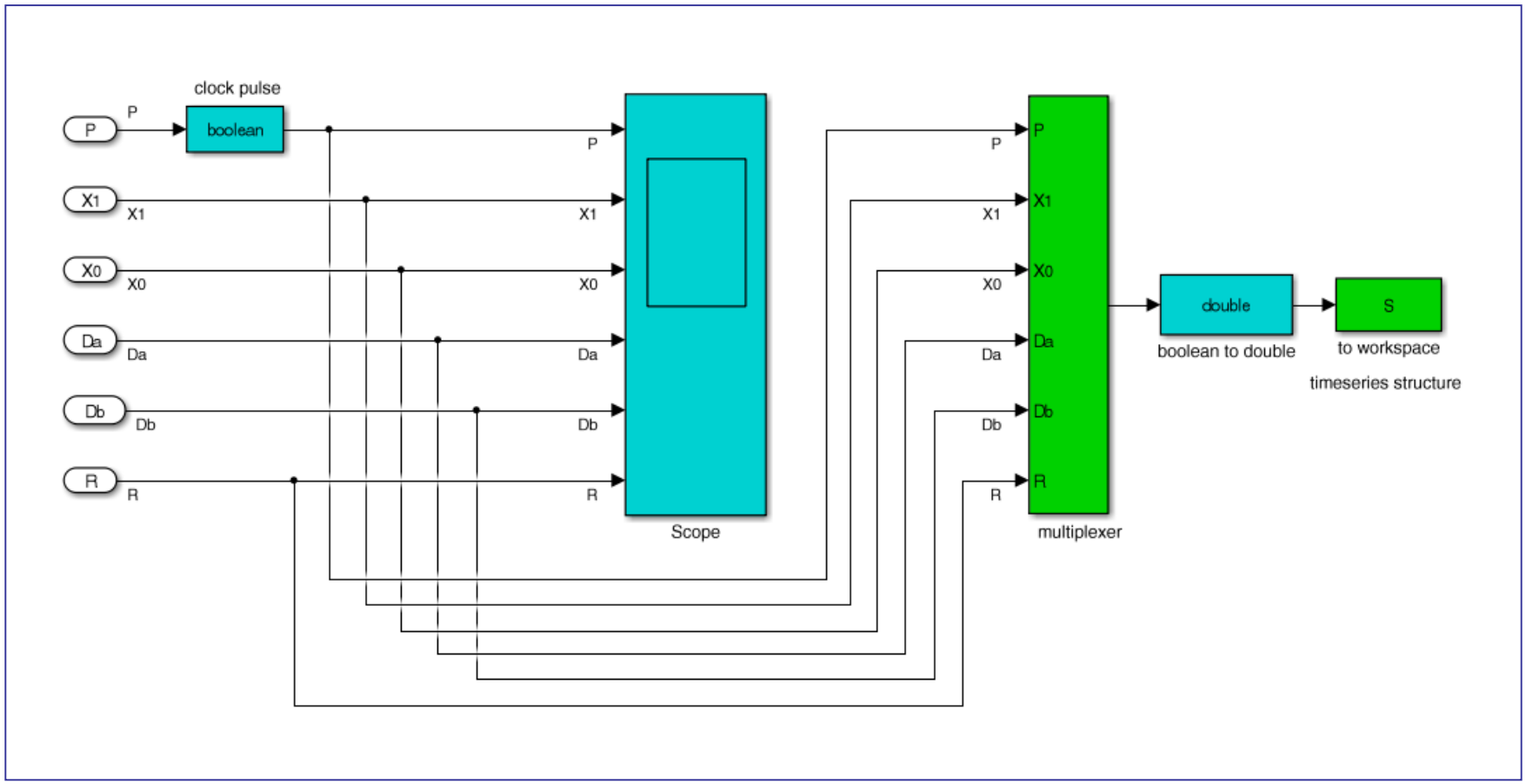
Simulink implementation next-states subfunction

$$D_A = A^{next} = R'(X_1 + X_0'A)$$

$$D_B = B^{next} = R + X_0 + X_1'(A+B')$$



Simulink implementation
scope & data export subfunction



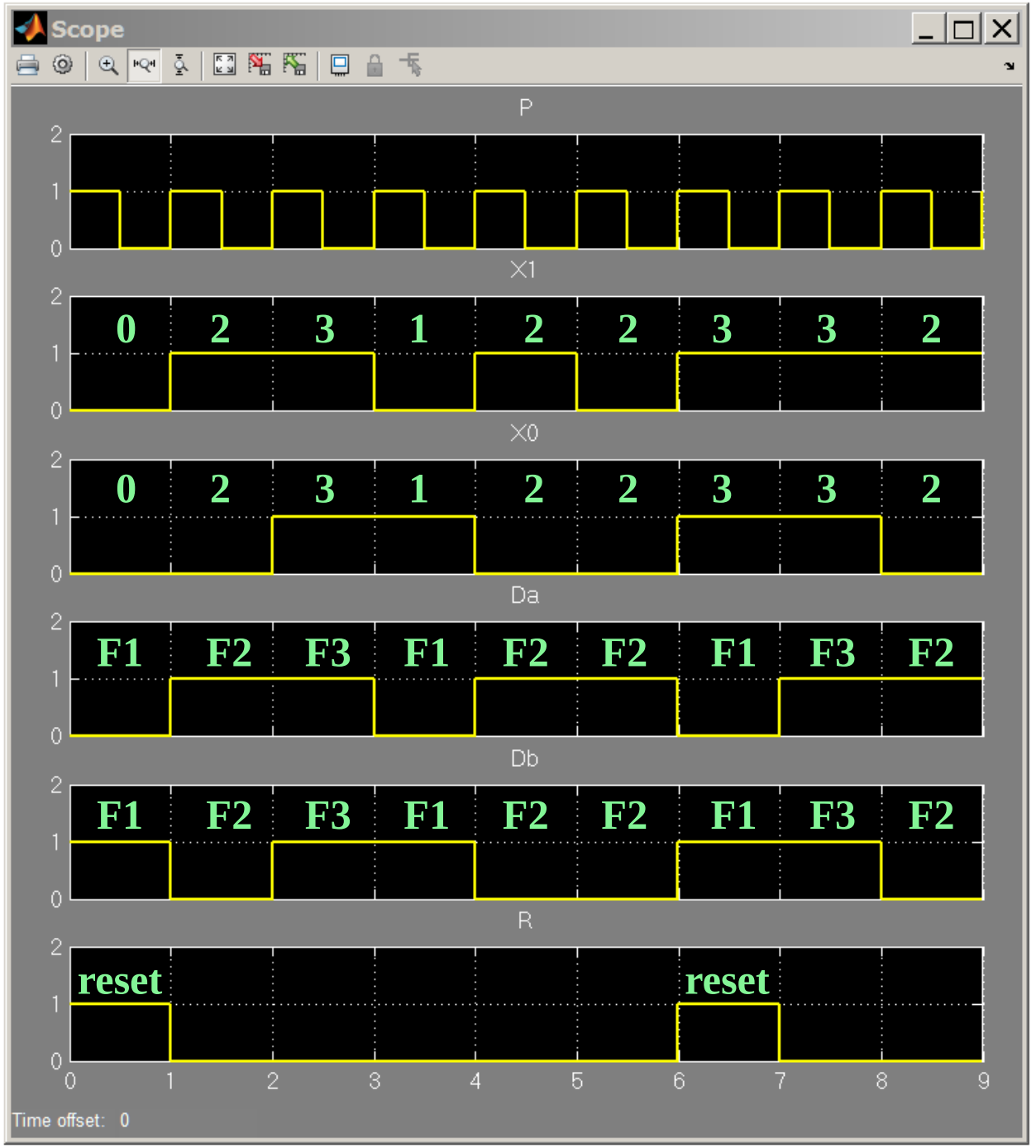
Simulink implementation

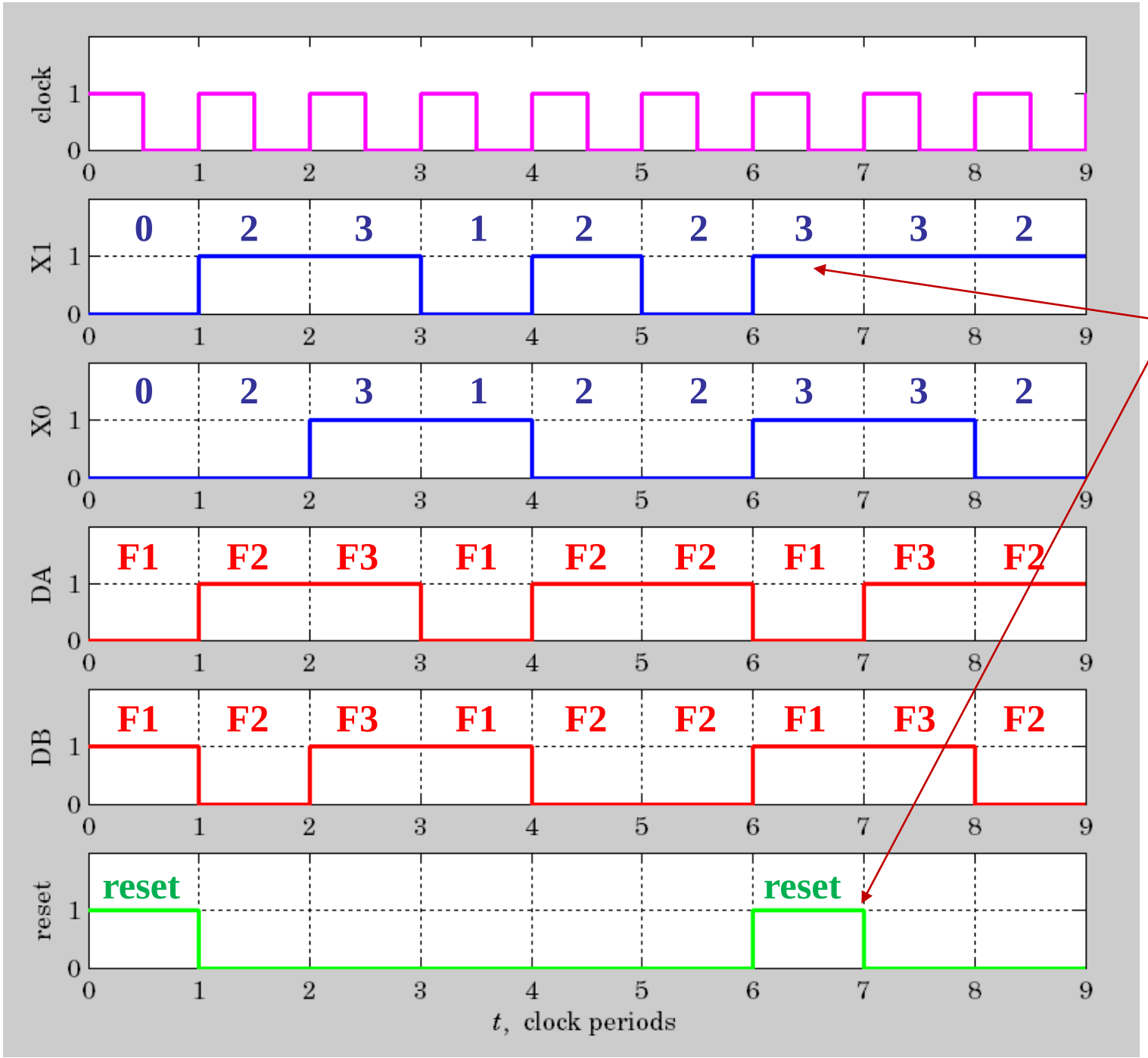
scope output

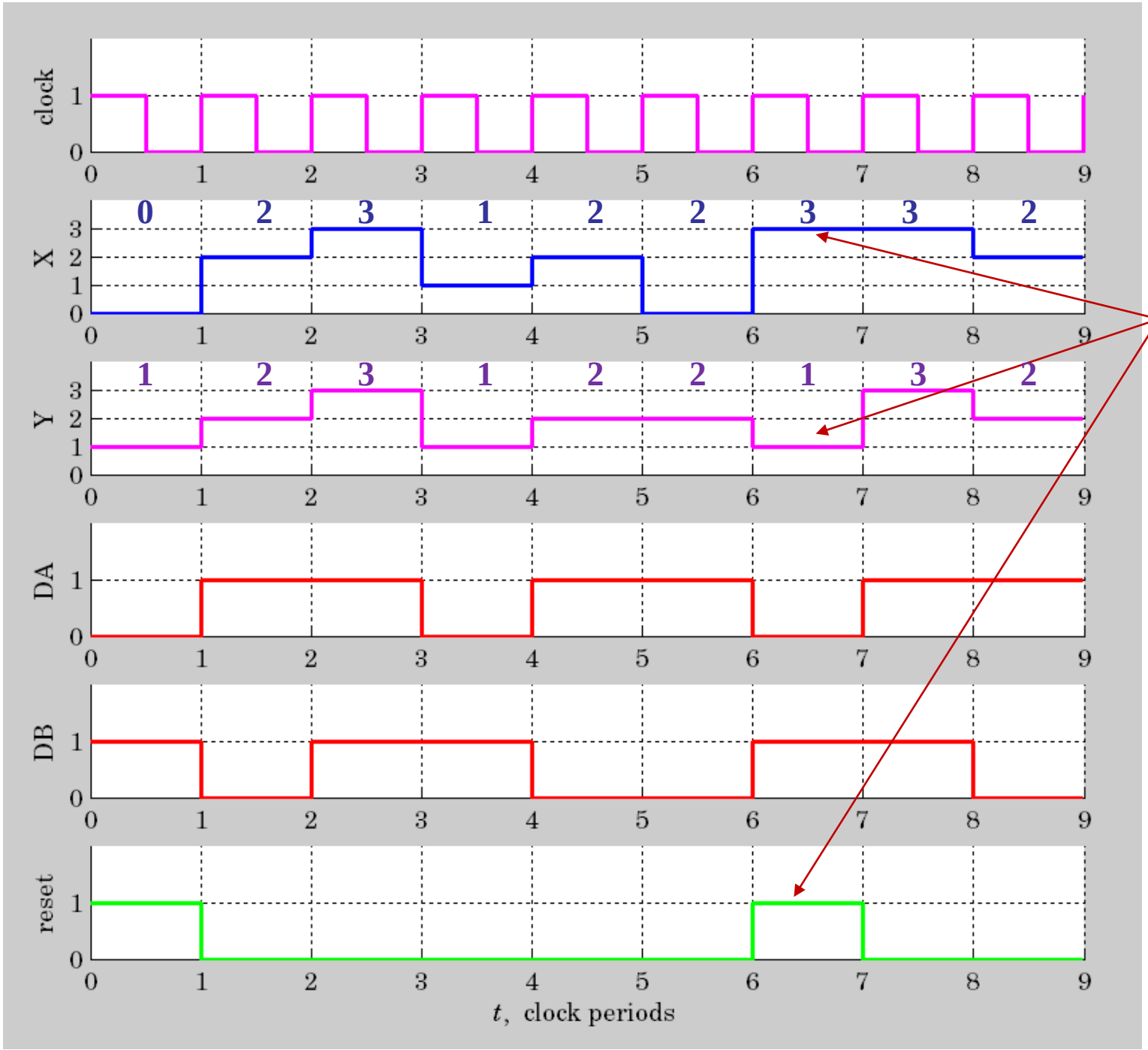
state encoding		
F	A	B
	0	0 unused
F_1	0	1
F_2	1	0
F_3	1	1

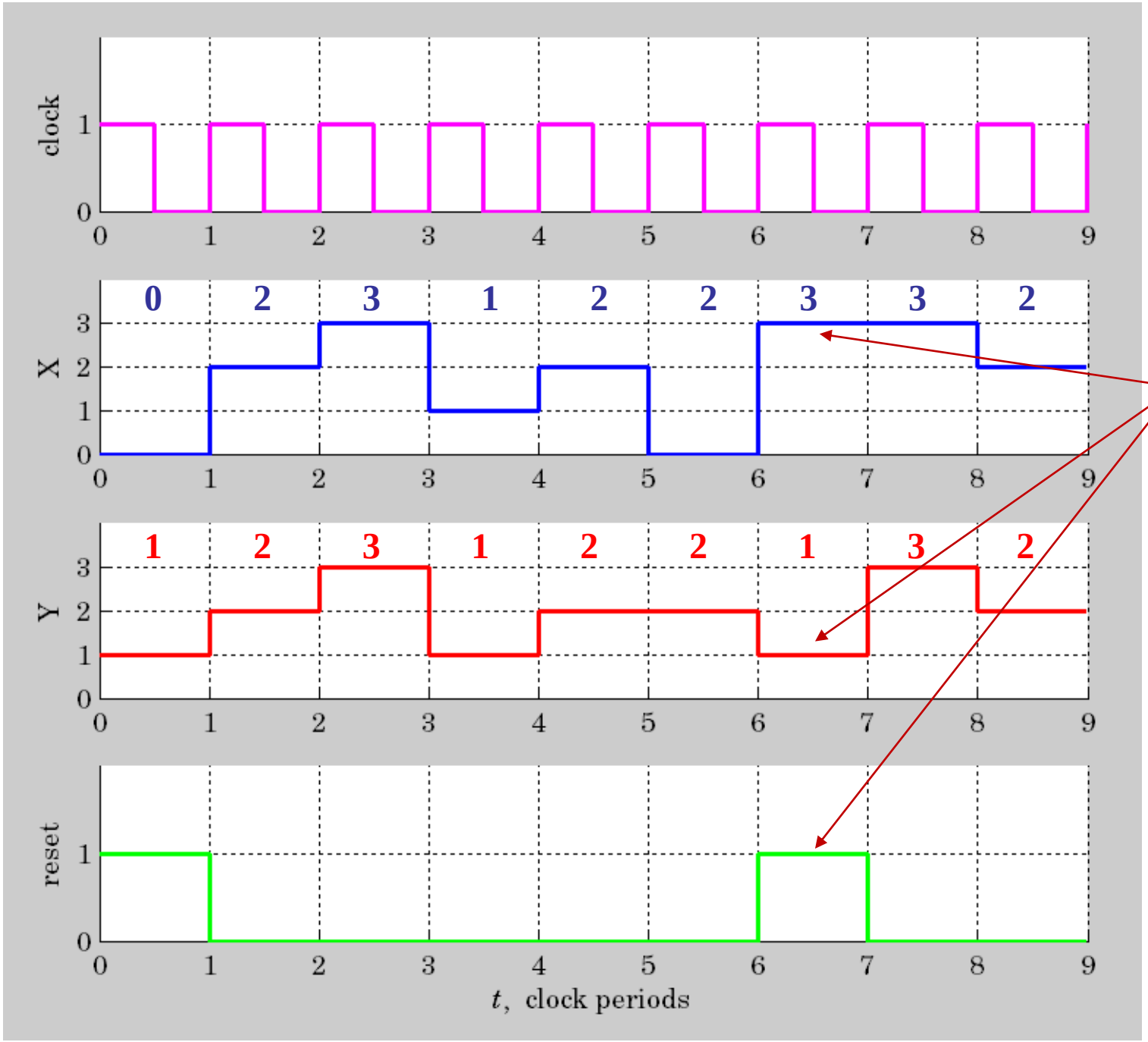
```
X  = [ 0 2 3 1 2 0 3 3 2 ]

X1 = [ 0 1 1 0 1 0 1 1 1 ]
X0 = [ 0 0 1 1 0 0 1 1 0 ]

R  = [ 1 0 0 0 0 0 1 0 0 ]

Y  = [ 1 2 3 1 2 2 1 3 2 ]
```

reset




clock
X1
X0
DA
DB
reset
0 2 3 1 2 2 3 3 2
F1 F2 F3 F1 F2 F2 F1 F3 F2
reset
reset
resets
to F1
t, clock periods


clock
X
Y
DA
DB
reset
0 2 3 1 2 2 3 3 2
1 2 3 1 2 2 1 3 2
resets
to F1
t, clock periods


clock
X
Y
reset
0 2 3 1 2 2 3 3 2
1 2 3 1 2 2 1 3 2
resets
to F1
t, clock periods