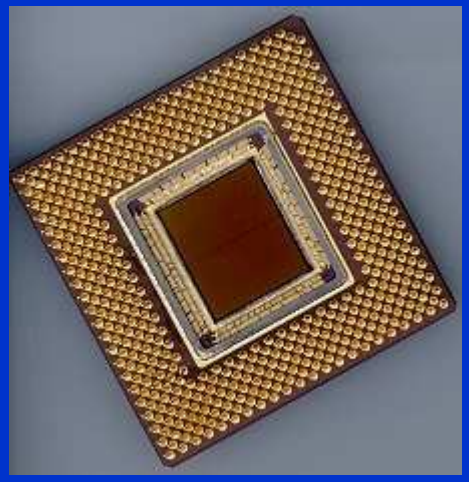


Introduction to Digital System Design

Module 3
Sequential Logic Circuits

Glossary of Common Terms

- **SEQUENTIAL LOGIC CIRCUIT – *next* output depends on its present inputs and its present state**
- **STATE – collection of state variables whose values at any one time contain all the information about the past necessary to account for the circuit's future behavior**
- **BI-STABLE – a logic device with two stable states**
- **LATCH – sequential circuit that watches all of its inputs *continuously* and changes its outputs *at any time* (independent of a clocking signal)**

Glossary of Common Terms

- **FLIP-FLOP – sequential circuit that samples its inputs and *changes its outputs* only at times determined by a *clocking signal***
- **FEEDBACK SEQUENTIAL CIRCUIT – uses ordinary gates and feedback loops to create sequential circuit building blocks such as latches and flip-flops**
- **CLOCKED SYNCHRONOUS STATE MACHINE – uses latches or flip-flops to create circuits whose inputs are examined and whose outputs change state in accordance with a controlling clock signal**

Glossary of Common Terms

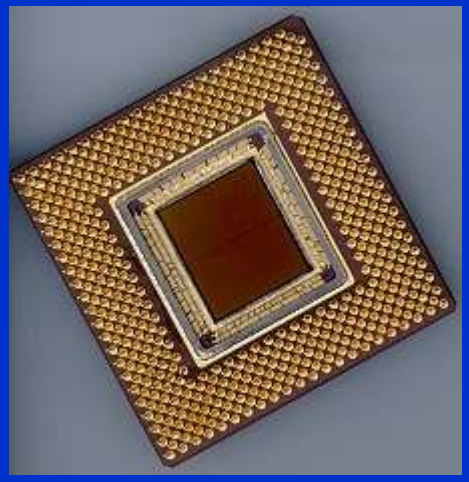
- **PRESENT STATE – NEXT STATE ("PS-NS" or "NEXT STATE") EQUATIONS – equations that describe the next state of a sequential circuit based on its present inputs and present state**
- **CHARACTERISTIC EQUATION – a next state equation that characterizes the behavior of a latch or flip-flop**
- **STATE TRANSITION DIAGRAM – a graph that depicts the state transition behavior of a sequential circuit**
- **TIMING CHART – a chart that depicts the timing behavior of a sequential circuit**

Glossary of Common Terms

- **EXCITATION EQUATIONS – equations that describe the inputs needed by sequential circuit memory elements (latches or flip-flops) to enable the circuit to transition from its present state to the desired next state**
- **SEQUENCE GENERATOR – a state machine that generates a (periodic) pre-defined output pattern of signal assertions**
- **COUNTER – a state machine that has a closed sequence of states**
- **SEQUENCE RECOGNIZER – a state machine that responds to a pre-defined input pattern of signal assertions and produces corresponding output signal assertions**

Module 3

- **Learning Outcome: "An ability to analyze and design sequential logic circuits"**
  - **A. Bi-stable Elements**
  - **B. Set-Reset (S-R) and Data (D) Latches**
  - **C. Data (D) and Toggle (T) Flip-Flops**
  - **D. State Machine Structure and Analysis**
  - **E. Clocked Synchronous State Machine Synthesis**
  - **F. State Machine Design Examples: Sequence Generators**
  - **G. State Machine Design Examples: Counters and Shift Registers**
  - **H. State Machine Design Examples: Sequence Recognizers**



Introduction to Digital System Design

Module 3-A
Bi-stable Elements

Reading Assignment:

DDPP 4th Ed., pp. 521-526

Learning Objectives:

- **Describe the difference between a combinational logic circuit and a sequential logic circuit**
- **Describe the difference between a feedback sequential circuit and a clocked synchronous state machine**
- **Define the state of a sequential circuit**
- **Define active high and active low as it pertains to clocking signals**
- **Define clock frequency and duty cycle**
- **Describe the operation of a bi-stable and analyze its behavior**
- **Define metastability and illustrate how the existence of a metastable equilibrium point can lead to a random next state**

Outline

- Overview
- Finite state machines
- Clock signal properties
- Types of sequential circuits
- Bi-stable elements
  - Digital analysis
  - Analog analysis
- Metastable behavior

Overview

- **Logic circuits are classified into two types:**
  - **a *combinational* logic circuit is one whose outputs depend only on its current inputs**
  - **a *sequential* logic circuit is one whose outputs depend not only on its current inputs, but also its *current state* (arrived at by its *past* sequence of inputs)**
- **The *state* of a sequential circuit is a collection of *state variables* whose values at any one time contain all the information about the past necessary to account for the circuit's future behavior**

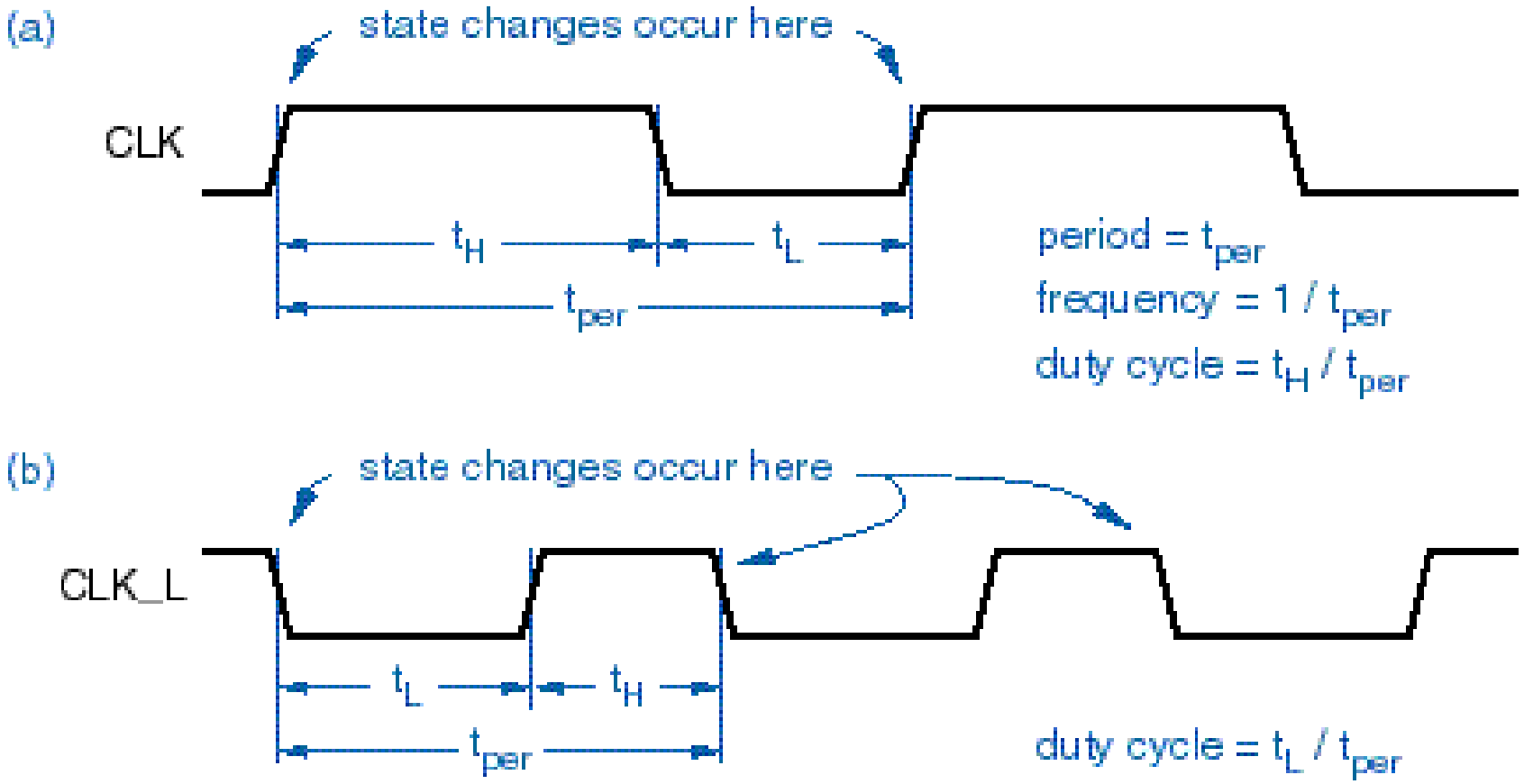
Finite State Machines

- In a digital logic circuit, state variables are binary values – a circuit with **n** binary state variables has 2^n possible states
- Since there are a only finite number of states possible, sequential circuits are sometimes called ***finite state machines***
- The state changes of most sequential circuits occur at times specified by a free-running **CLOCK** signal

Clock Signal Properties

- **By convention, a CLOCK signal is *active high* if state changes occur in response to the clock signal's *rising* edge (or when it is *high*)**
- **Similarly, a CLOCK signal is *active low* if state changes occur in response to the clock signal's *falling* edge (or when it is *low*)**
- **The *clock period* is the time between successive transitions in the same direction**
- **The *clock frequency* (measured in Hertz, or cycles-per-second) is the *reciprocal* of the clock period**
- **The *duty cycle* is the percentage of time that the clock signal is at its asserted level**

Clock Signal Properties

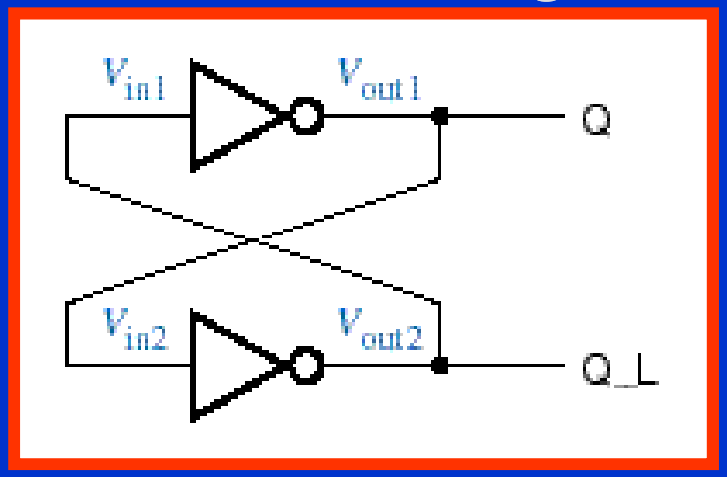


Types of Sequential Circuits

- **There are two basic types of sequential circuits that account for the majority of practical discrete designs:**
  - **a *feedback sequential circuit* uses ordinary gates and feedback loops to create sequential circuit building blocks such as latches and flip-flops**
  - **a *clocked synchronous state machine* uses latches and flip-flops (in particular, edge-triggered "D" flip-flops) to create circuits whose inputs are examined and whose outputs change state in accordance with a controlling clock signal**

Bi-stable Elements

- The “simplest” sequential circuit consists of a pair of inverters forming a feedback loop:



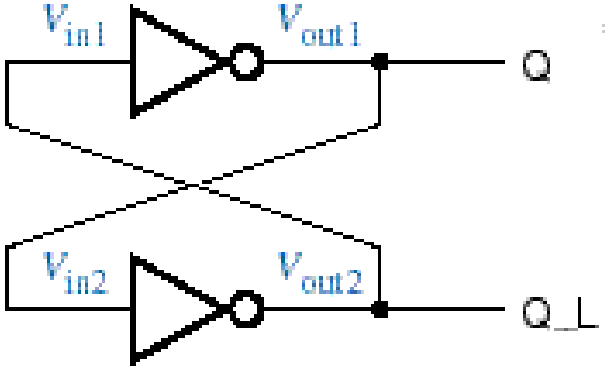
- This element has no inputs and therefore no way of controlling or changing its state
- When power is first applied, it randomly comes up in one state or the other and stays there forever (“not very useful”)

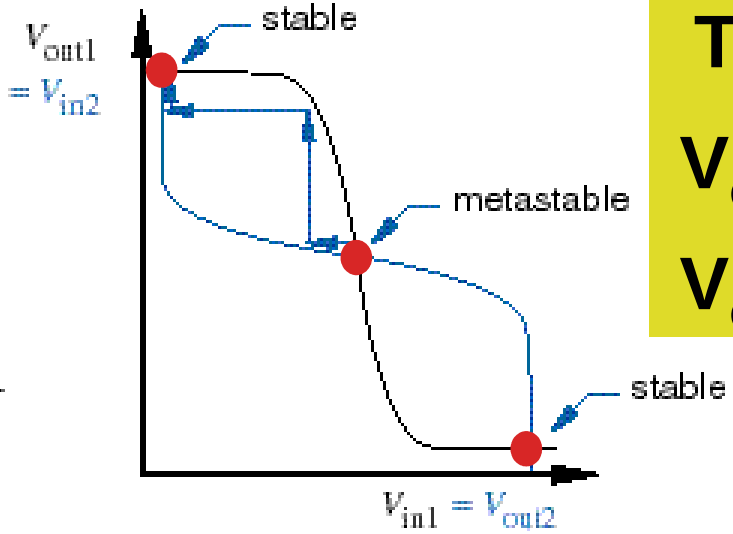
Digital Analysis of Bi-stable

- **This circuit is called a *bi-stable* because, based on (strictly) digital analysis, it has *two* stable states:**
  - **if Q is HIGH, then the bottom inverter has a high input and a LOW output, which forces the top inverter's output HIGH**
  - **if Q is LOW, then the bottom inverter has a LOW input and a HIGH output, which forces Q to go LOW**
- **Based on this analysis, a single state variable ("Q") could be used to describe the state of this circuit**

Analog Analysis of Bi-stable

- **Given the feedback connection, we know that $V_{in1} = V_{out2}$ and $V_{in2} = V_{out1}$**





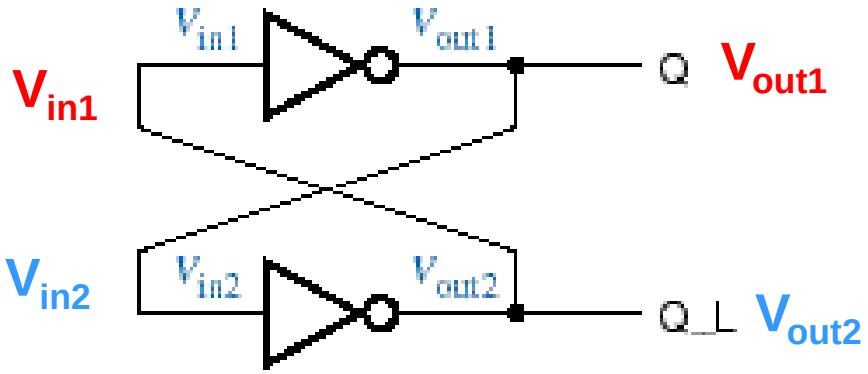
Transfer fns:

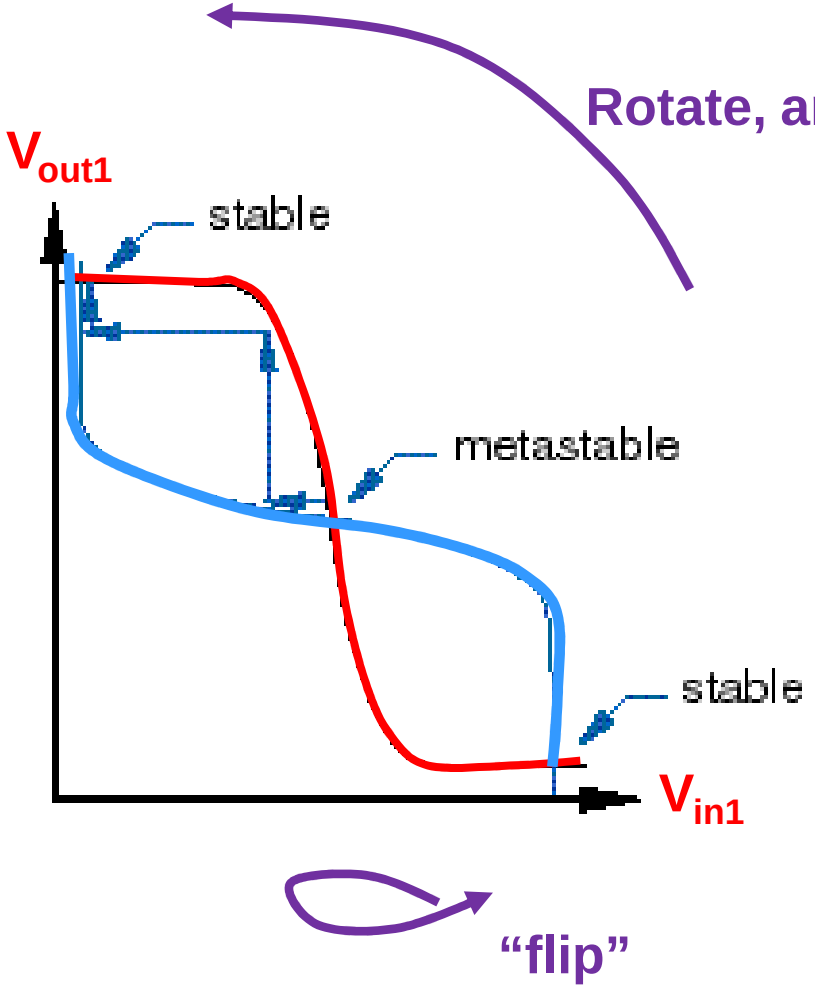
$V_{out1} = T(V_{in1})$

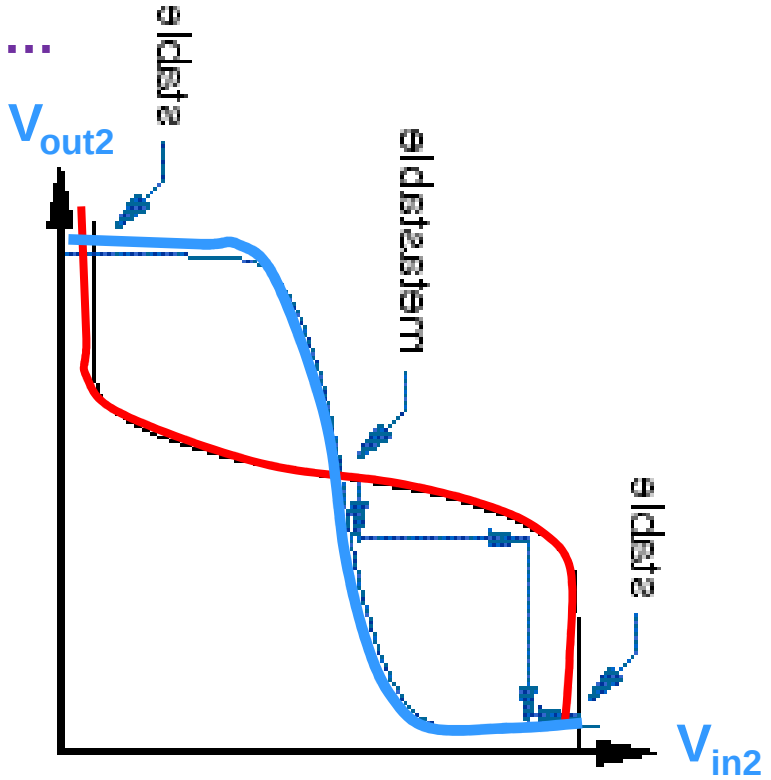
$V_{out2} = T(V_{in2})$

- **The feedback loop is in *equilibrium* if the input and output voltages of both inverters are constant DC values consistent with their transfer functions**

Analog Analysis





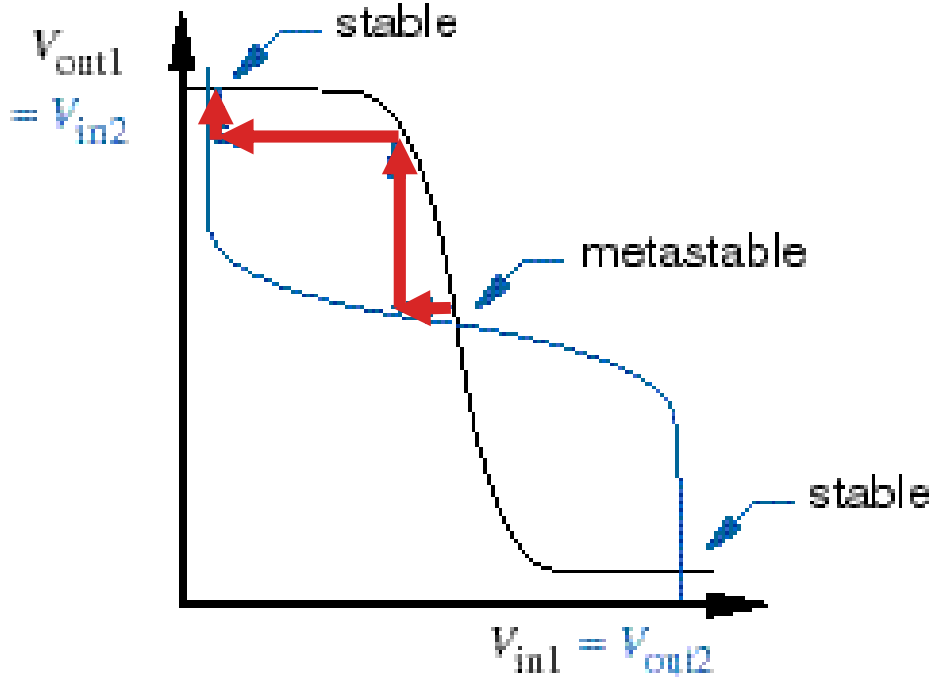


Analog Analysis of Bi-stable

- **The *equilibrium points* can be found graphically – they are the points at which the two transfer functions meet:**
  - **the two *stable* equilibrium points correspond to the two states identified in the "digital" analysis, with Q (Q_L) either "0" (LOW) or "1" (HIGH)**
  - **the *metastable* equilibrium point occurs with V_{out1} and V_{out2} about halfway between a valid logic "1" voltage and a valid logic "0" voltage – here, Q and Q_L are not valid logic signals but the loop equations are satisfied**

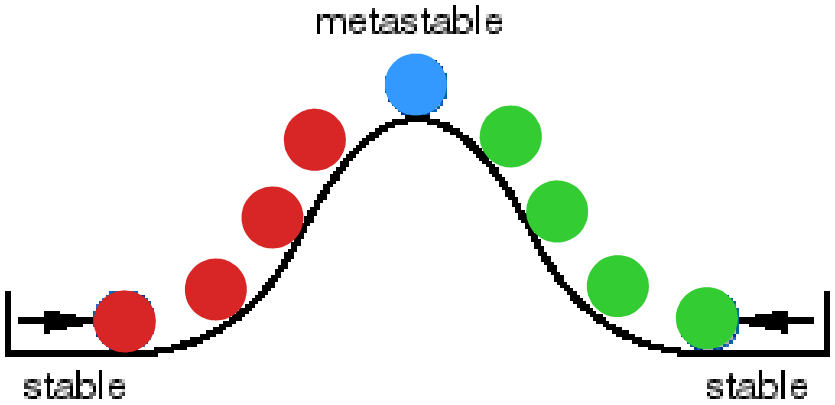
Metastable Behavior

- **The *metastable point* is *not truly stable*, because *random noise* will tend to drive a circuit operating at the metastable point toward one of the stable operating points**



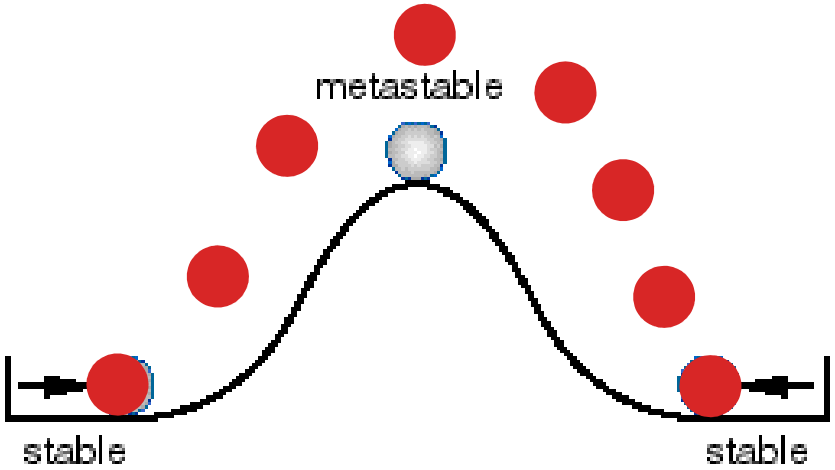
Metastable Behavior

- **Metastable behavior of a bistable can be compared to the behavior of a ball dropped onto a hill:**
  - **if ball is dropped from overhead, it will probably roll down immediately to one side of the hill or the other**
  - **if ball lands right at the top, it may sit there a while before random forces start it rolling**



Metastable Behavior

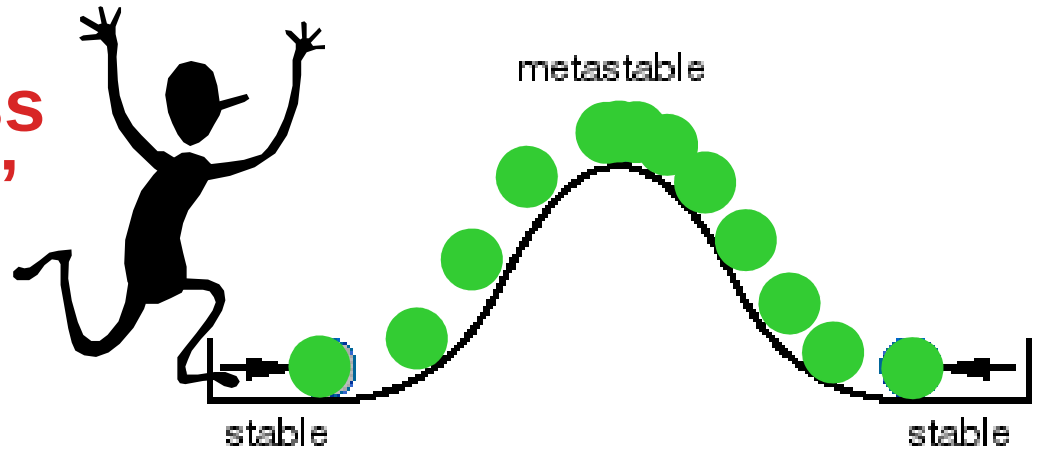
- **<u>Important</u>: If the "simplest" sequential circuit is susceptible to metastable behavior, you can be sure that *all* sequential circuits are susceptible (and it is not something that only occurs at power-up)**
- **Consider what happens if we try to "kick" the ball from side of the hill to the other:**



Metastable Behavior

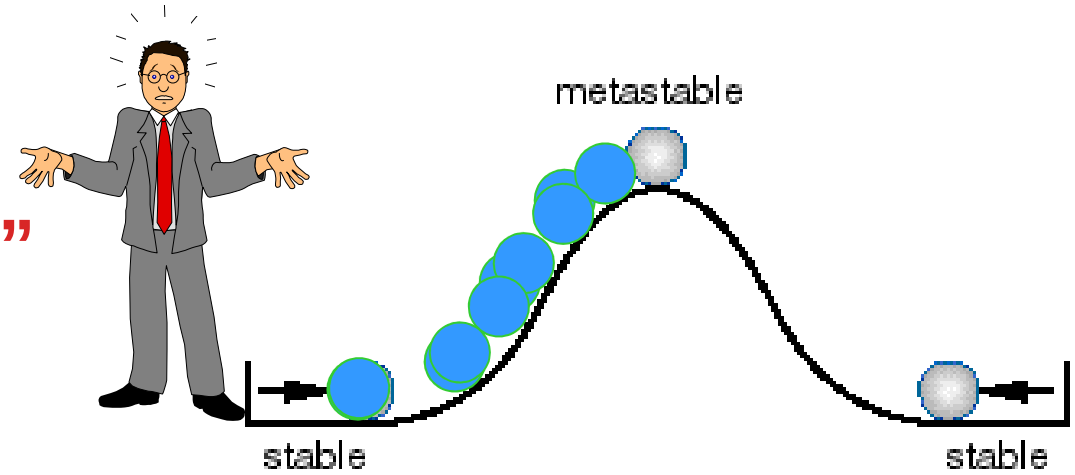
- **<u>Important</u>: If the "simplest" sequential circuit is susceptible to metastable behavior, you can be sure that *all* sequential circuits are susceptible (and it is not something that only occurs at power-up)**
- **Consider what happens if we try to "kick" the ball from side of the hill to the other:**

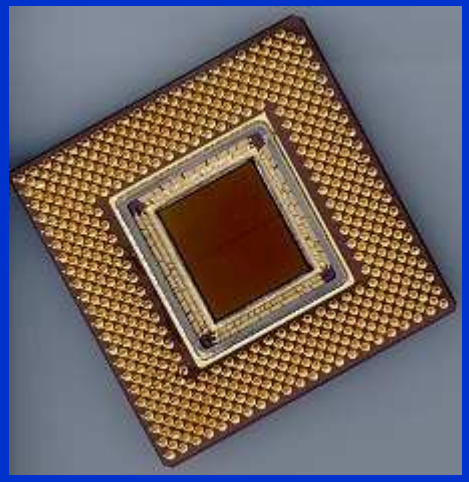
"clueless clip art"



Metastable Behavior

- **<u>Important</u>: If the "simplest" sequential circuit is susceptible to metastable behavior, you can be sure that *all* sequential circuits are susceptible (and it is not something that only occurs at power-up)**
- **Consider what happens if we try to "kick" the ball from side of the hill to the other:**





Introduction to Digital System Design

Module 3-B

Set-Reset (S-R) and Data (D) Latches

Reading Assignment:
DDPP 4th Ed., pp. 526-532

Learning Objectives:

- **Write present state – next state (PS-NS) equations that describes the behavior of a sequential circuit**
- **Draw a state transition diagram that depicts the behavior of a sequential circuit**
- **Construct a timing diagram that depicts the behavior of a sequential circuit**
- **Draw a circuit for a set-reset latch and analyze its behavior**
- **Discuss what is meant by "transparent" (or "data following") in reference to the response of a latch**

Outline

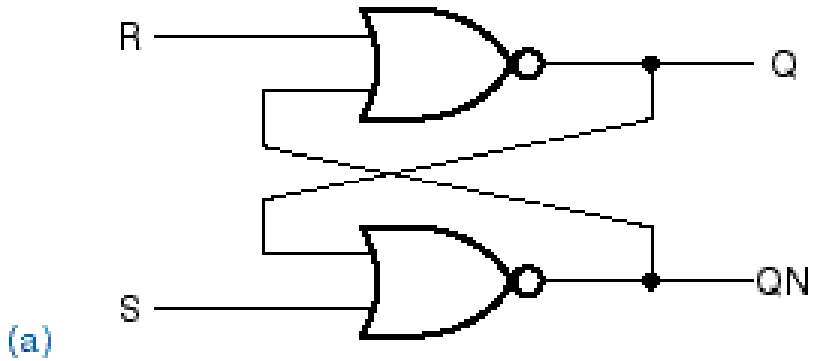
- Overview
- Set-Reset (S-R) latch
  - Basic operation
  - Timing charts
    - Normal operation
    - Response to 1-1 input combination
    - Response to a glitch/hazard
  - Propagation delays
  - Input pulse width
  - Variants
    - S′-R′ latch
    - S-R latch with enable
  - Characteristic equation
- Data (D) latch
  - Propagation delays
  - Setup and hold times

Overview

- **Definition: A *latch* is a sequential circuit that watches all of its inputs *continuously* and changes its outputs *at any time***
- **When a latch is *enabled*, it is "open" (i.e., its outputs "follow" its inputs)**
- **When a latch is *disabled* (its enable input is negated), it is "closed" (i.e., its outputs are "frozen" or "latched")**
- **This behavior lends itself to the names "data following" and "transparent"**
- **Note: Latches do not utilize a "clocking" signal; rather, they are "enabled" to open/close**

S-R Latch

- **An S-R (“set-reset”) latch based on NOR gates can be implemented as follows:**



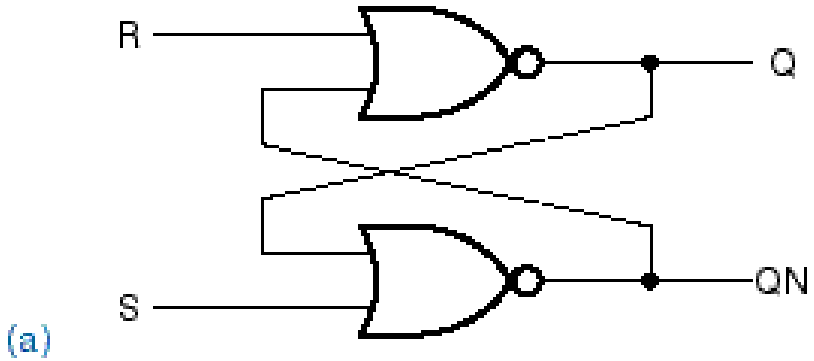
(a)

S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

(b)

- **It has a “set” (S) input and a “reset” (R) input and two outputs (Q and QN) that are normally complements of each other**

S-R Latch



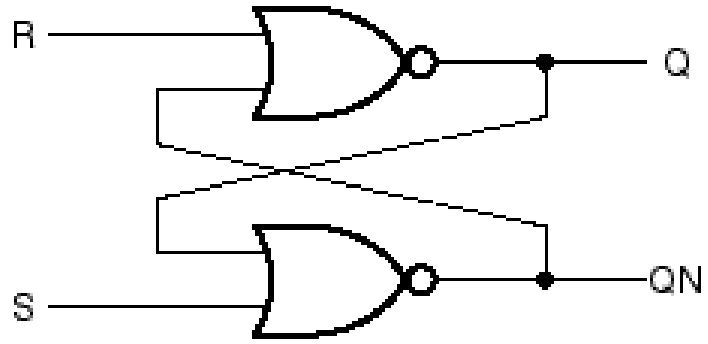
(a)

(b)

S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

- **Asserting S *sets* (presets) the Q output to "1"**
- **Asserting R *resets* (clears) the Q output to "0"**
- **If both S and R are "0", the circuit behaves like the bistable element – a feedback loop retains one of two logic states, Q = 0 or Q = 1**

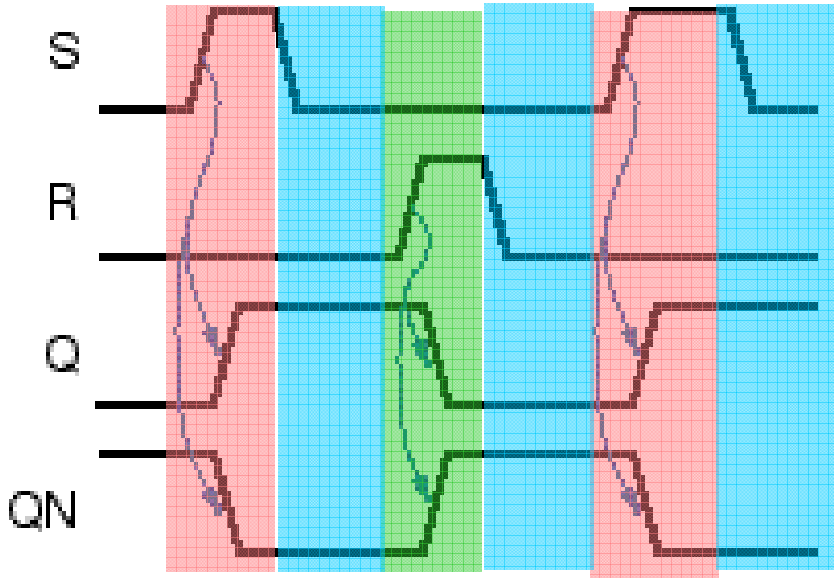
S-R Latch



(a)

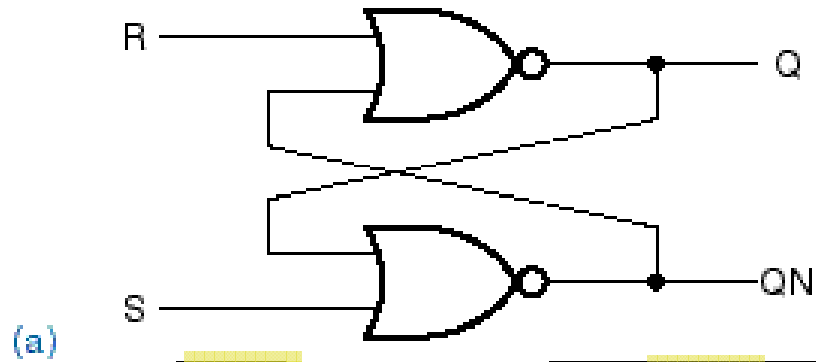
S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

(b)

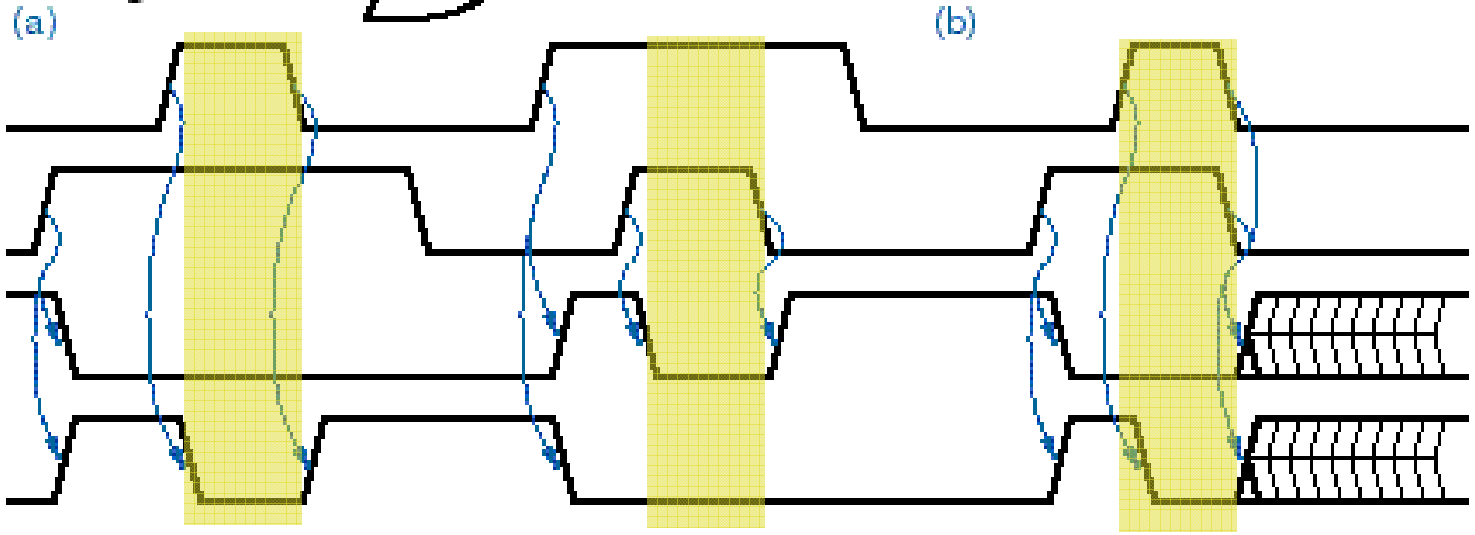


S-R Latch

- **If *both* S and R are "1", both outputs go LOW; if both S and R return to "0" simultaneously, the circuit goes to a random next state**

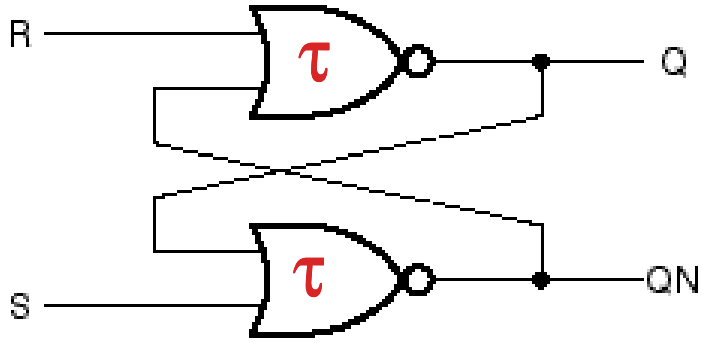


S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0



Exercise

- **Construct a timing chart for the S-R latch**

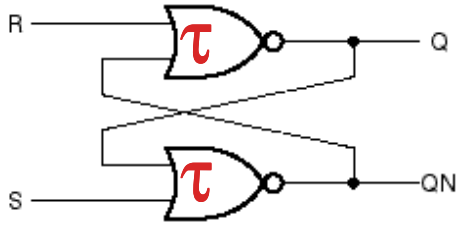


- **Solution: Start by writing *next state equations* that describe the circuit, and from them construct a *present state - next state* table**

$$Q(t+\tau) = R'(t) \cdot QN'(t)$$

$$QN(t+\tau) = S'(t) \cdot Q'(t)$$

Exercise



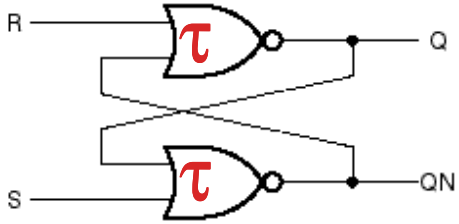
- **PS-NS table:**

$$Q(t+\tau) = R'(t) \cdot QN'(t) \qquad QN(t+\tau) = S'(t) \cdot Q'(t)$$

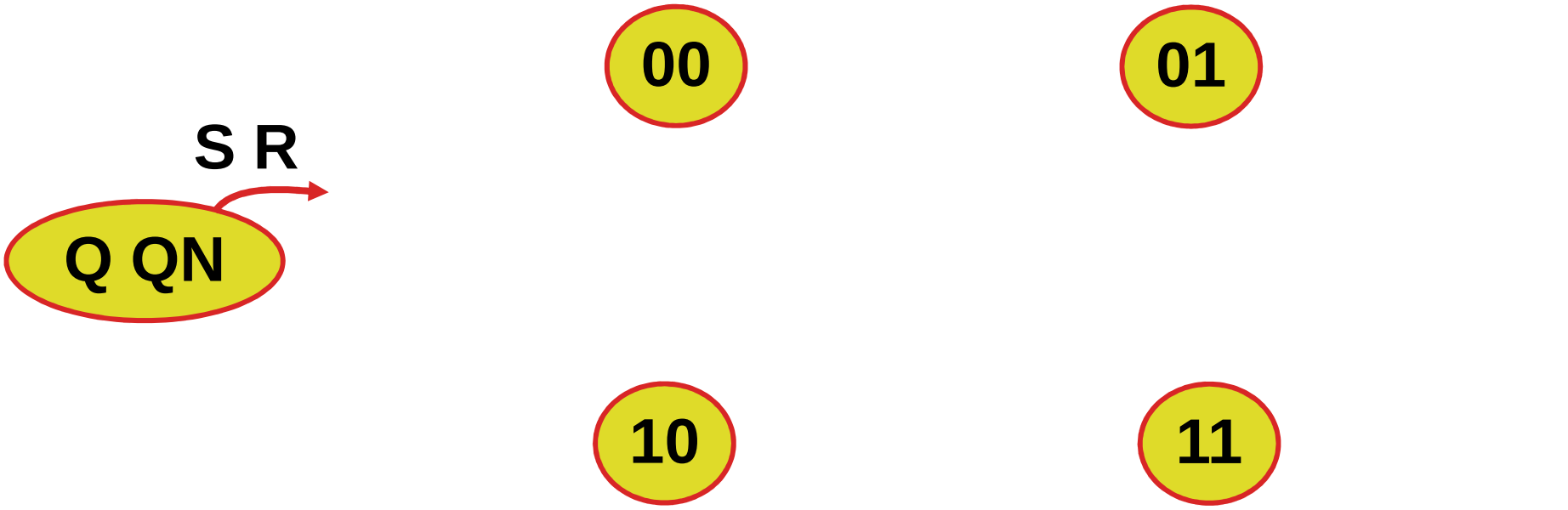
Present State Q(t) QN(t)	Present Inputs: S(t) R(t) 00	01	10	11
00	11	01	10	00
01	01	01	00	00
10	10	00	10	00
11	00	00	00	00

Next State

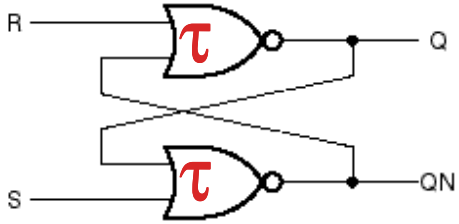
Exercise



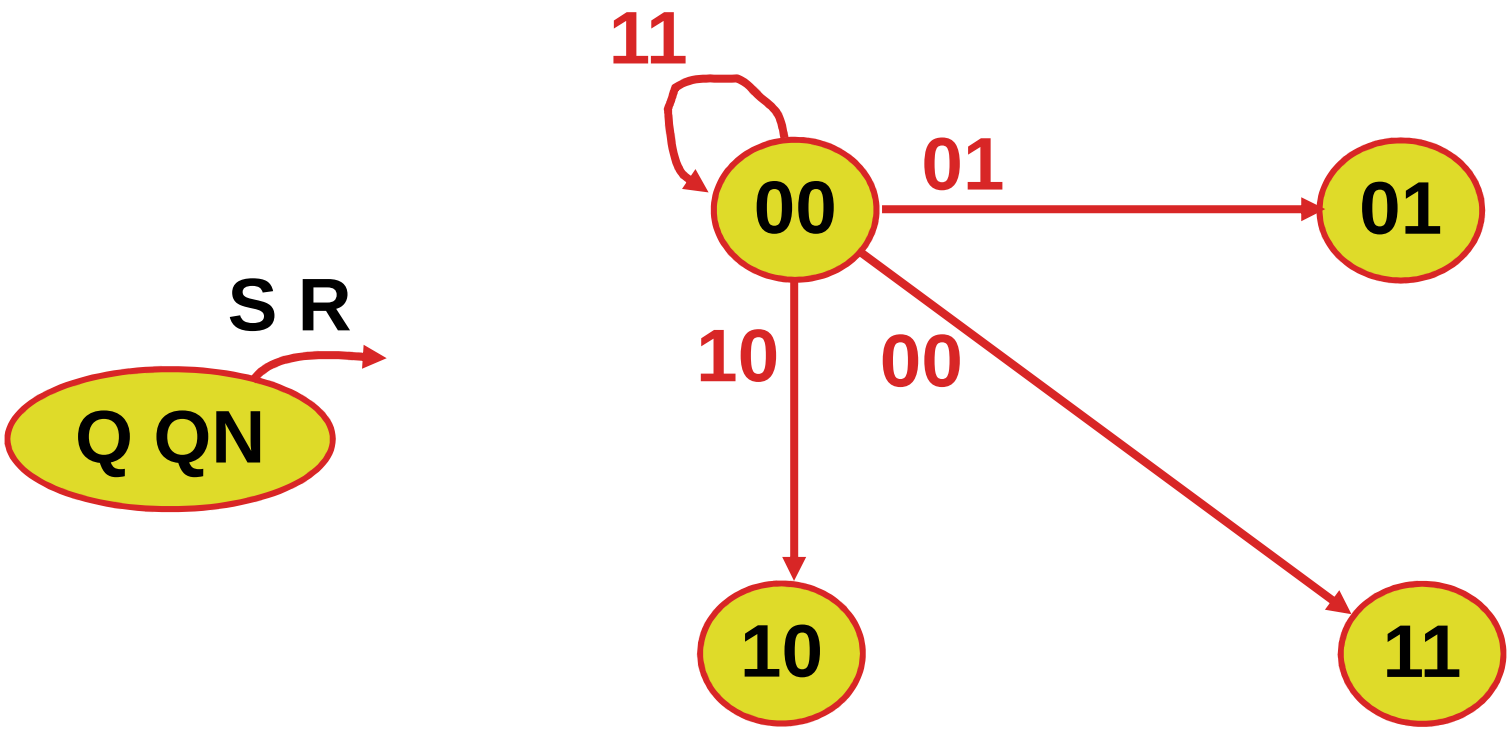
- **From the PS-NS table, construct a state transition diagram**



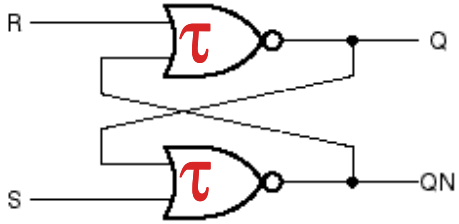
Exercise



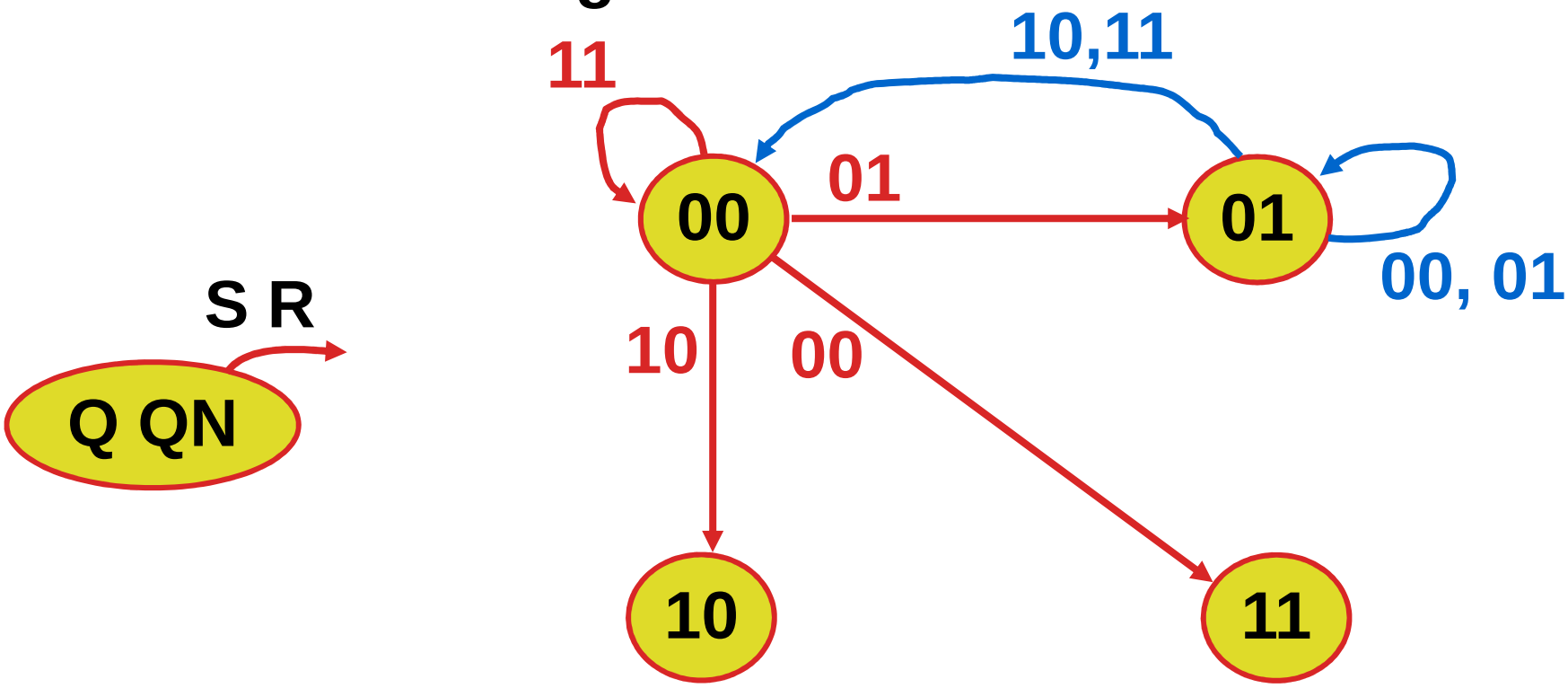
- **From the PS-NS table, construct a state transition diagram**



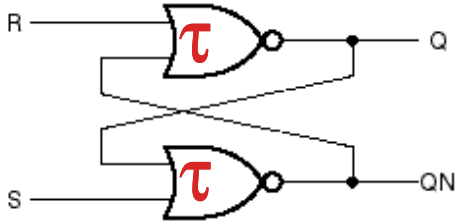
Exercise



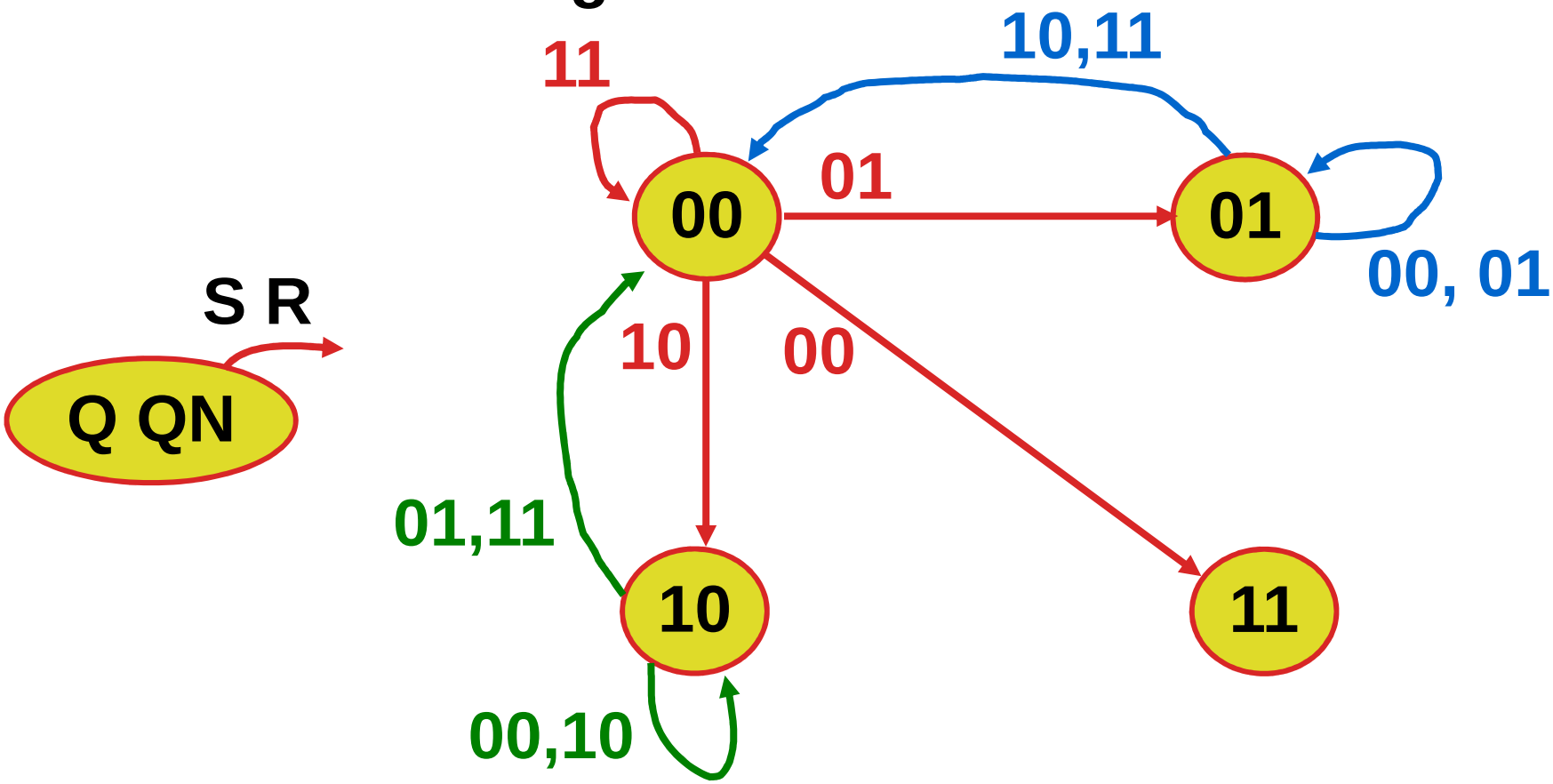
- **From the PS-NS table, construct a state transition diagram**



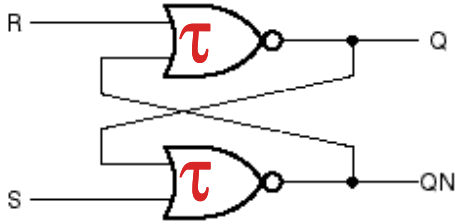
Exercise



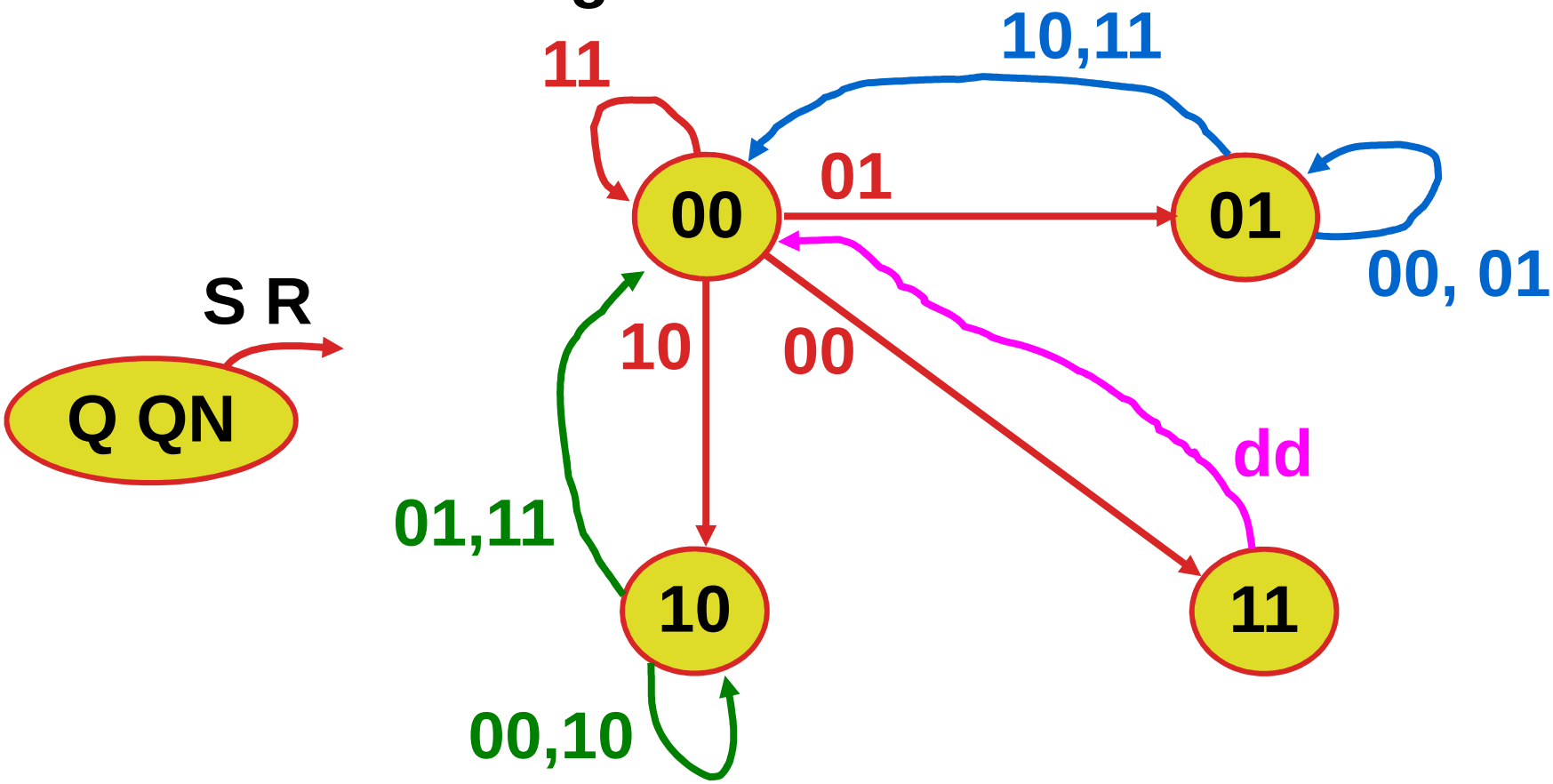
- **From the PS-NS table, construct a state transition diagram**



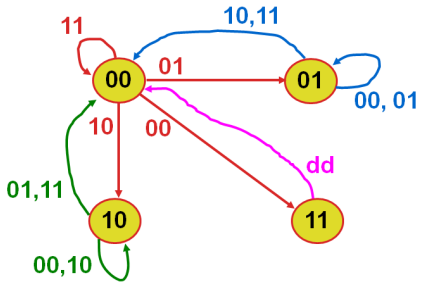
Exercise

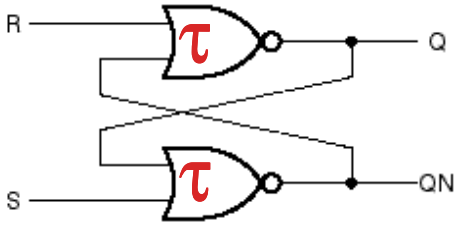


- **From the PS-NS table, construct a state transition diagram**

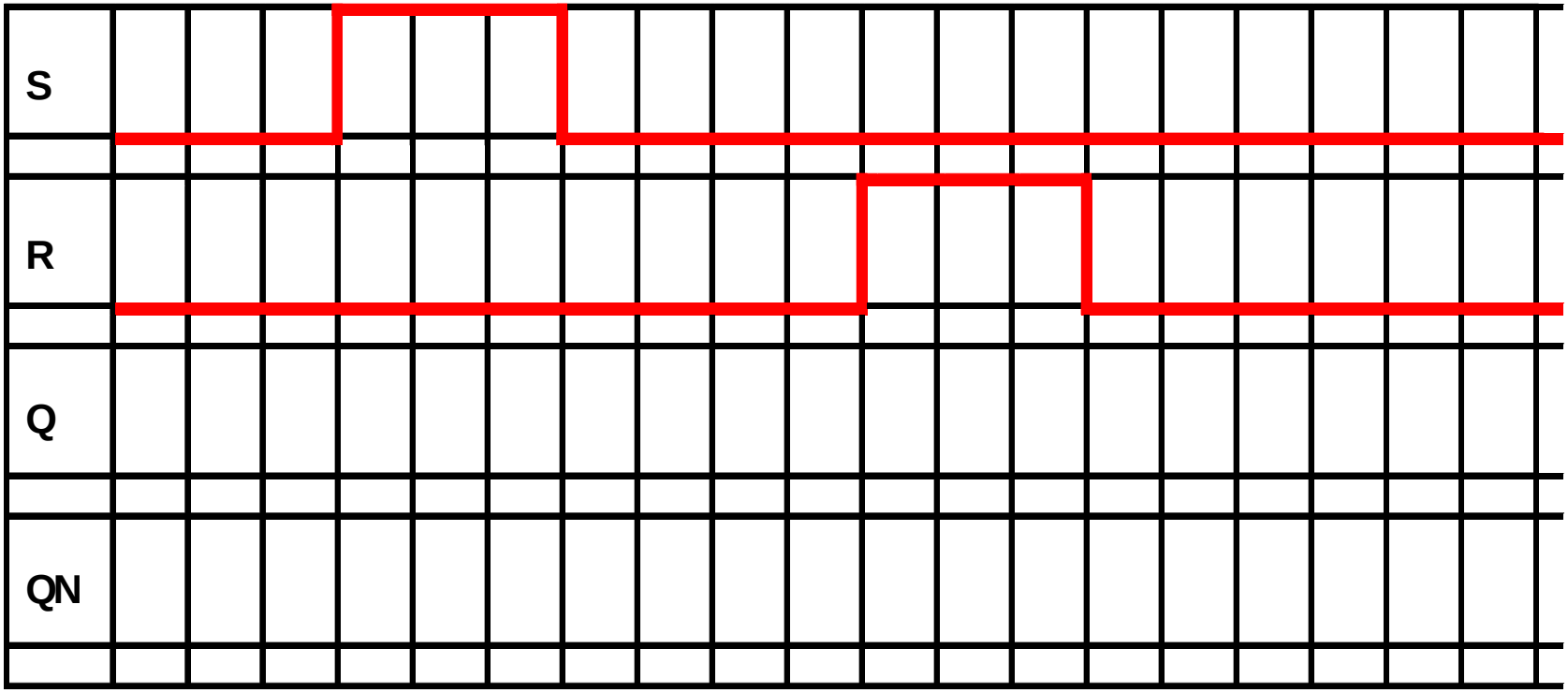


Exercise

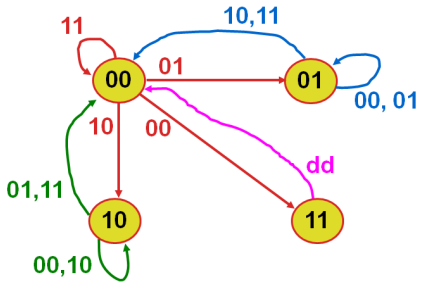


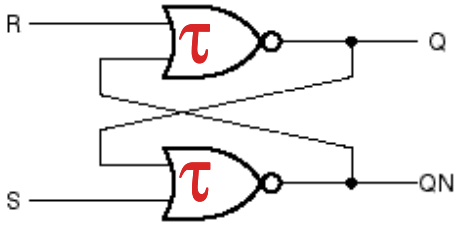


- **From the state transition diagram, construct a timing chart**

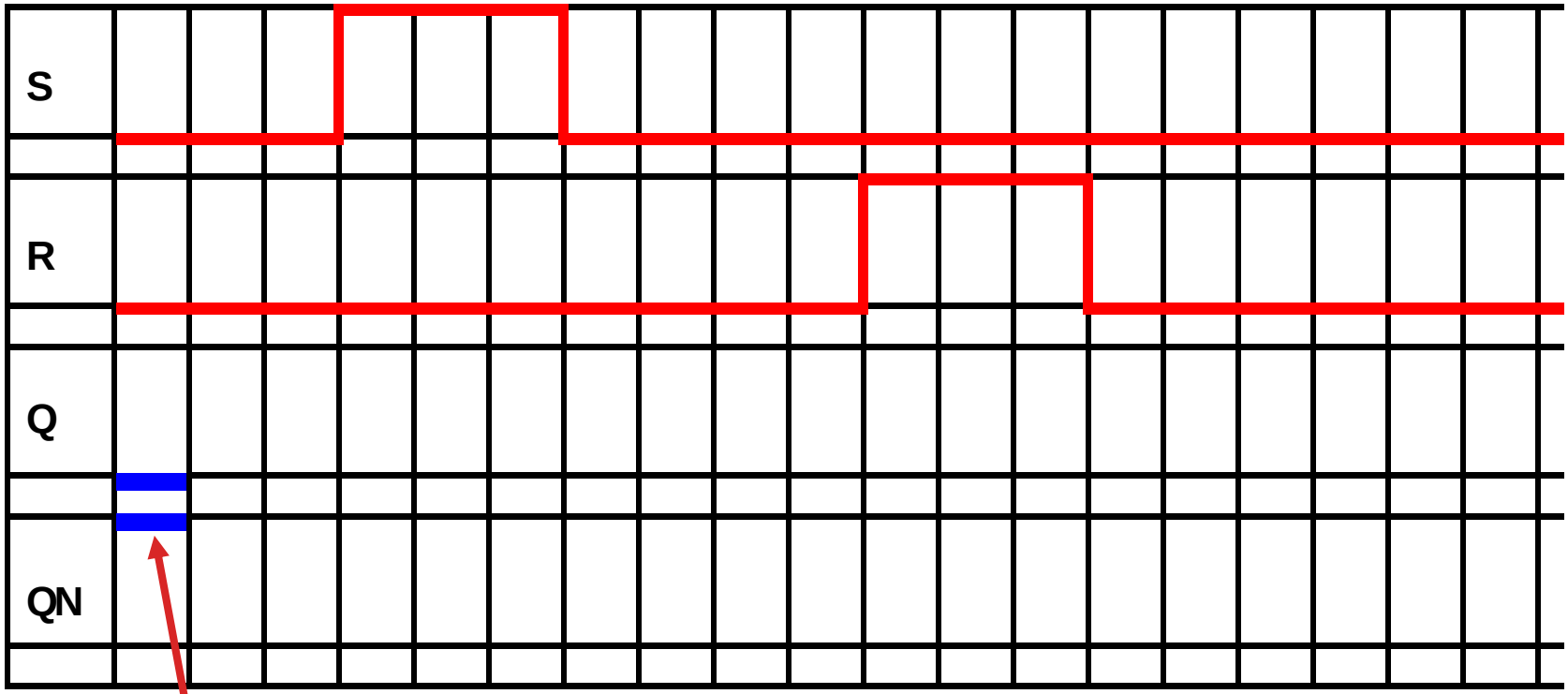


Exercise

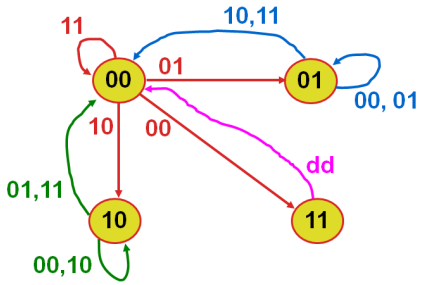


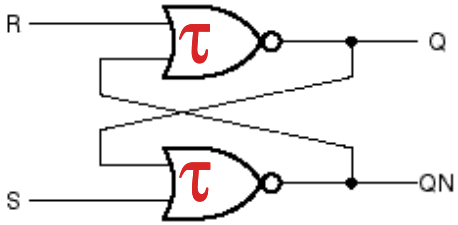


- **From the state transition diagram, construct a timing chart**

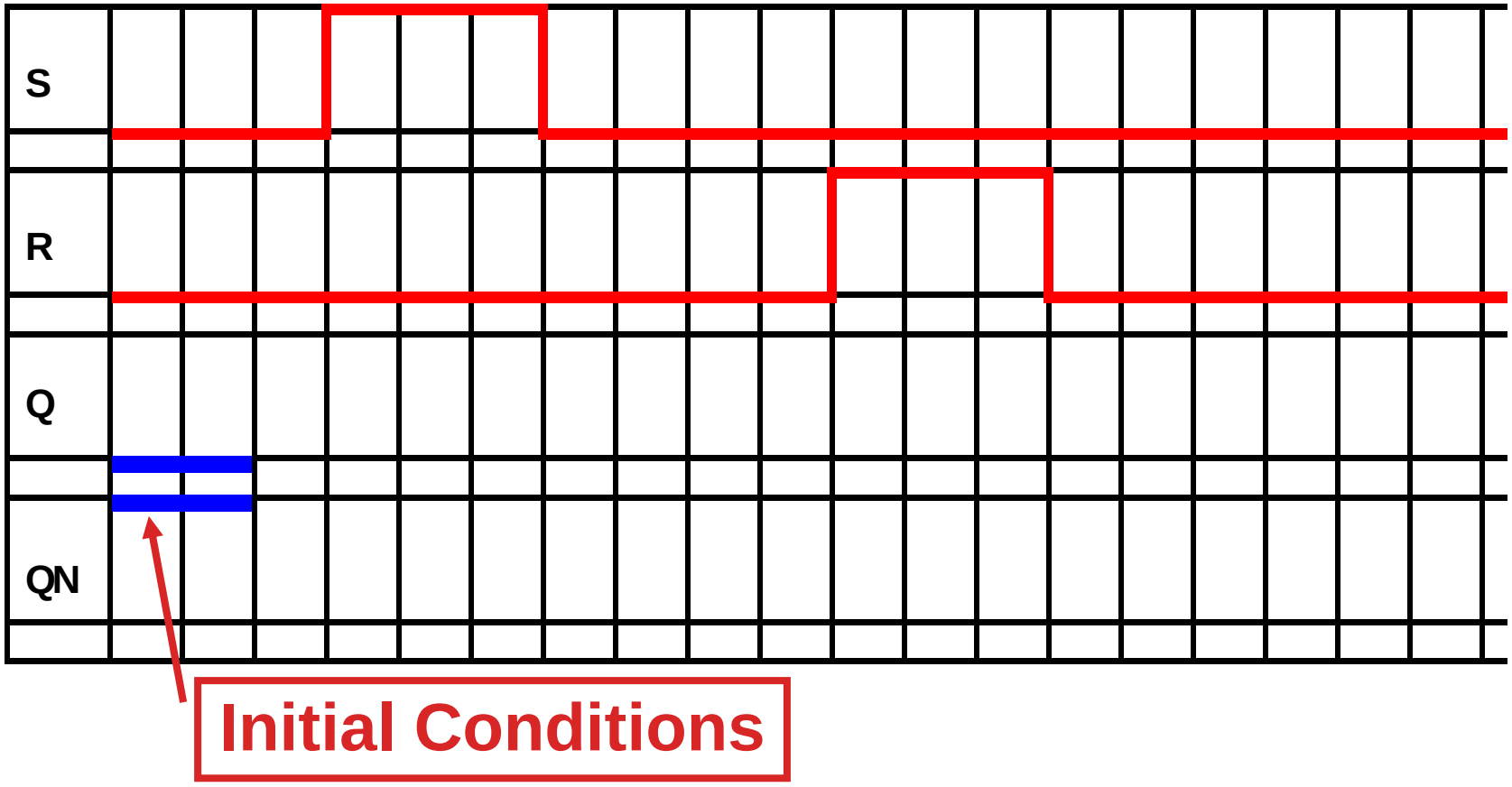


Exercise

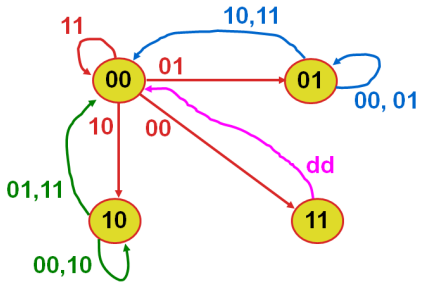


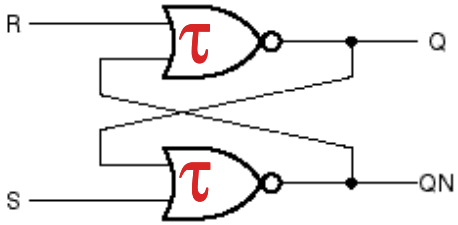


- **From the state transition diagram, construct a timing chart**

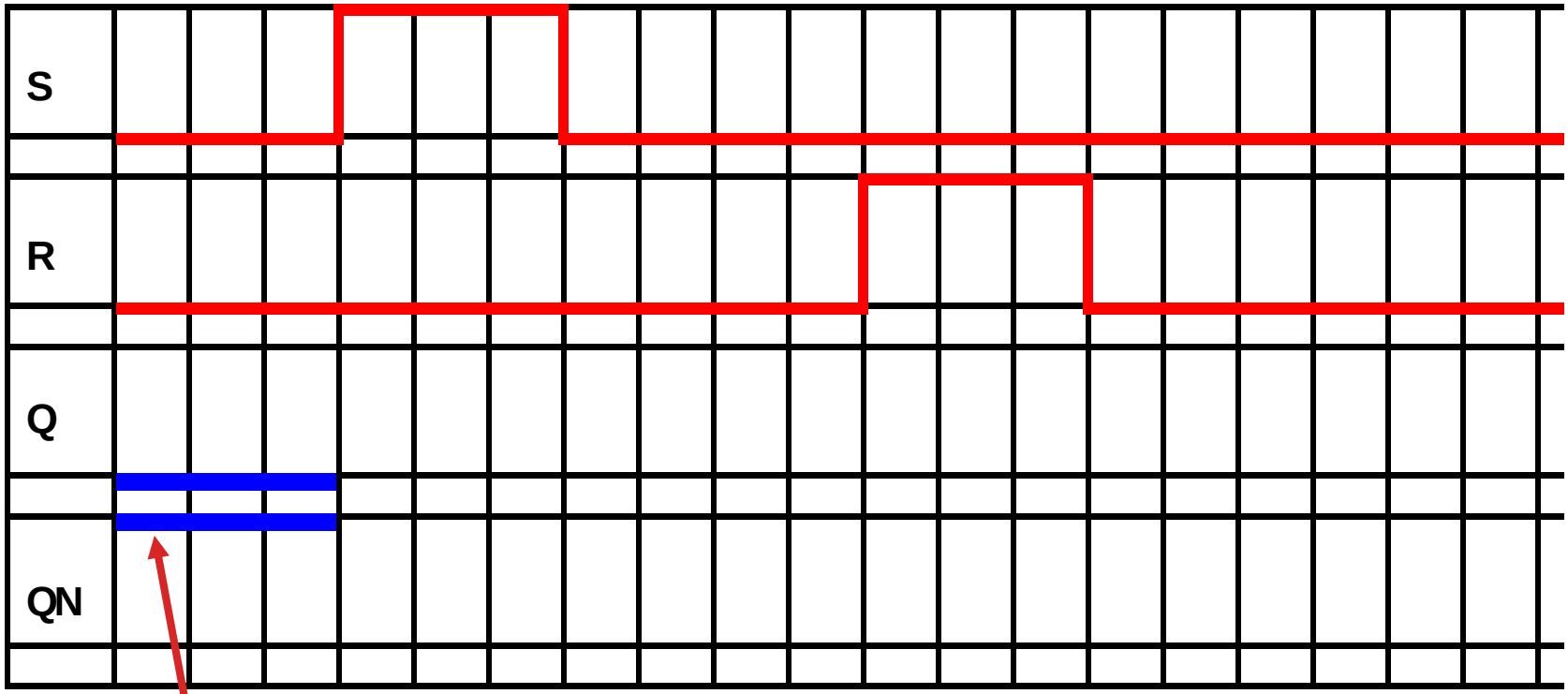


Exercise

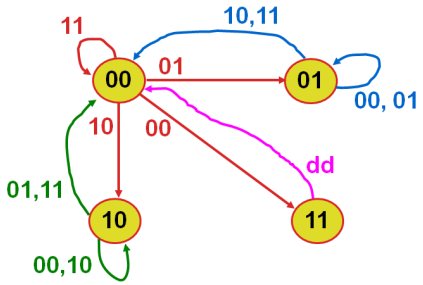


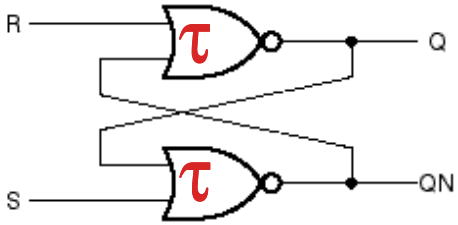


- **From the state transition diagram, construct a timing chart**

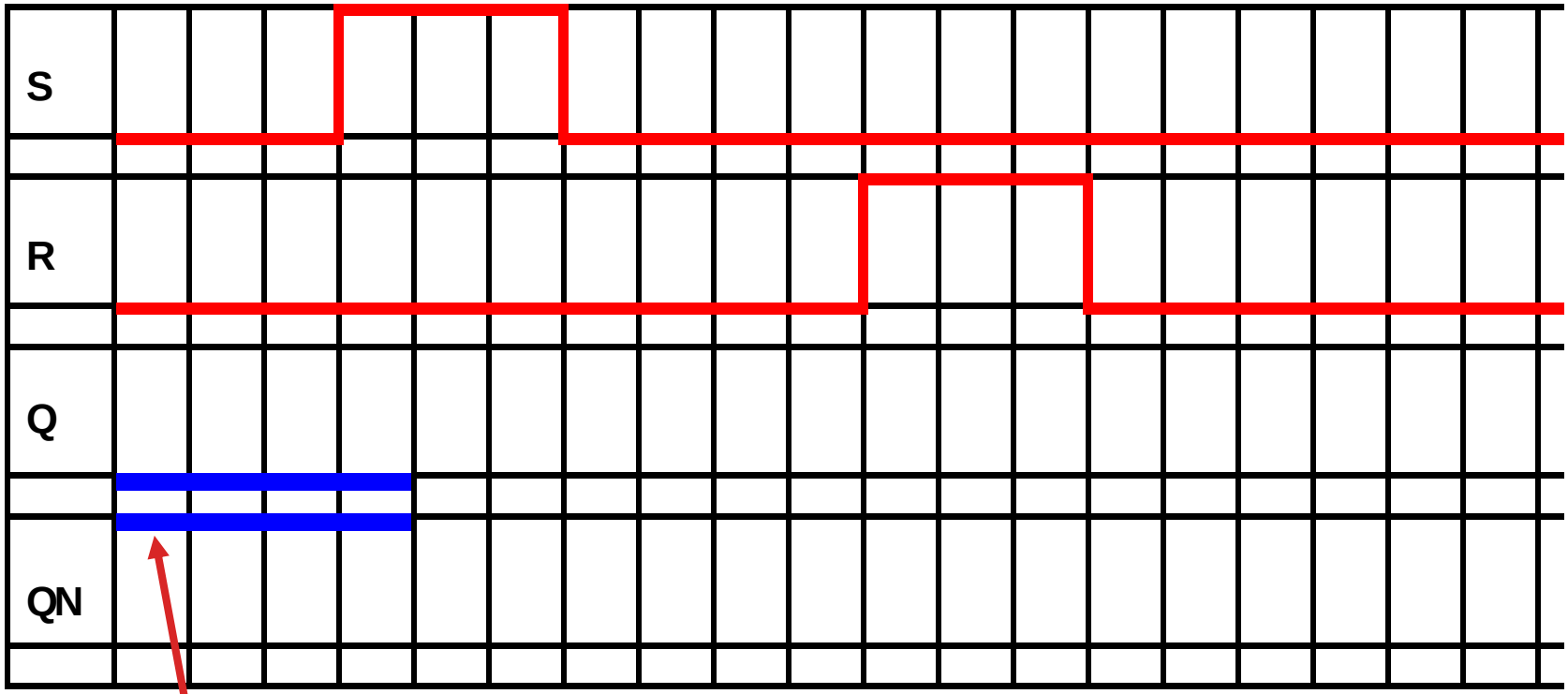


Exercise

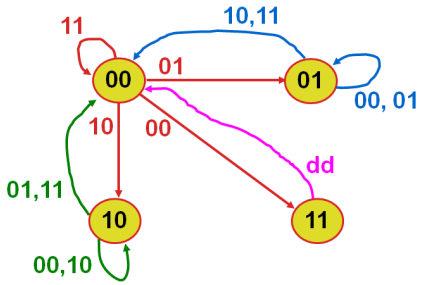


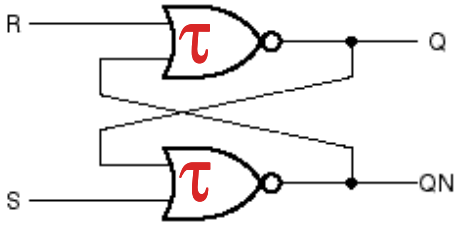


- **From the state transition diagram, construct a timing chart**

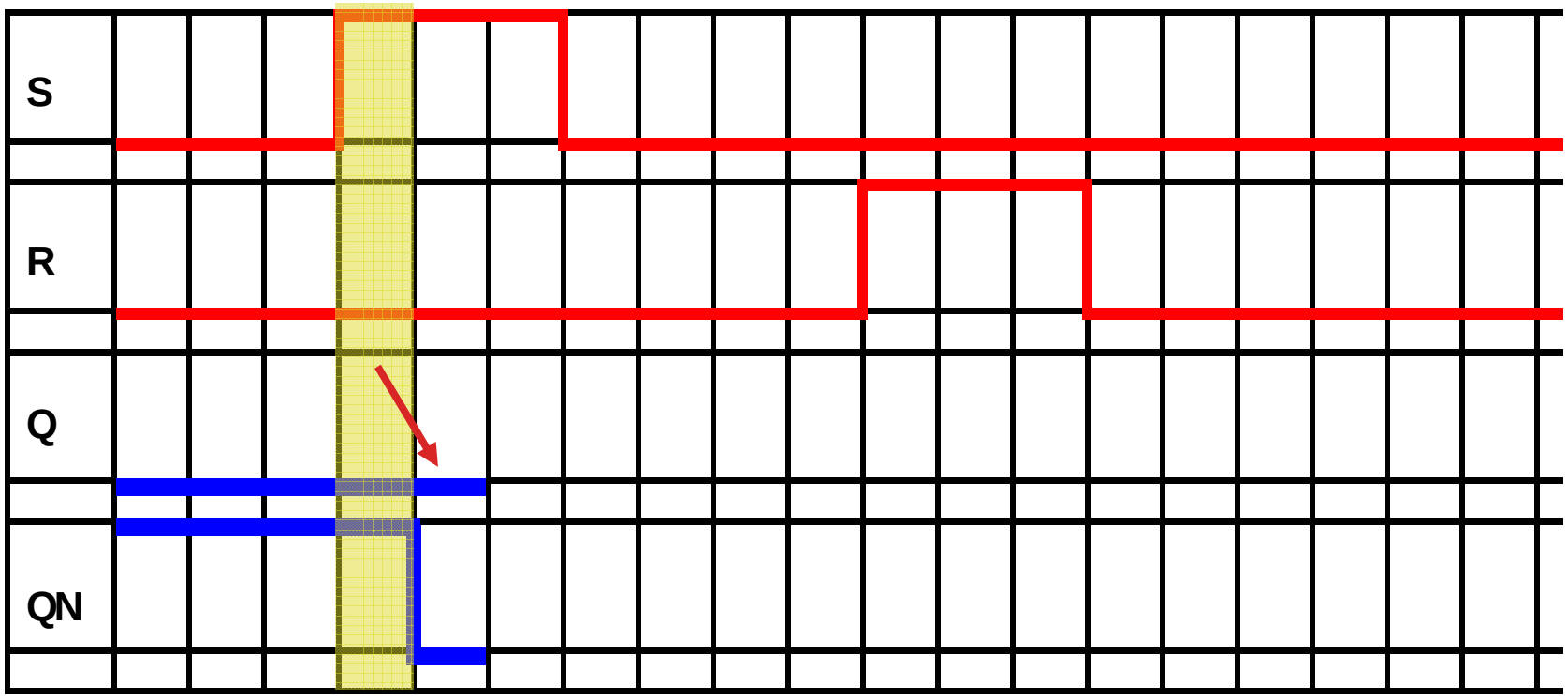


Exercise

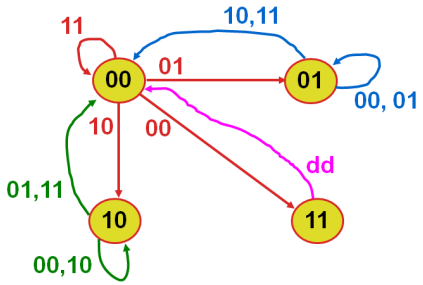


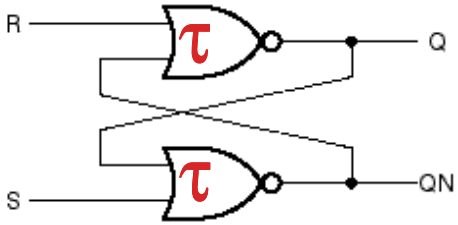


- **From the state transition diagram, construct a timing chart**

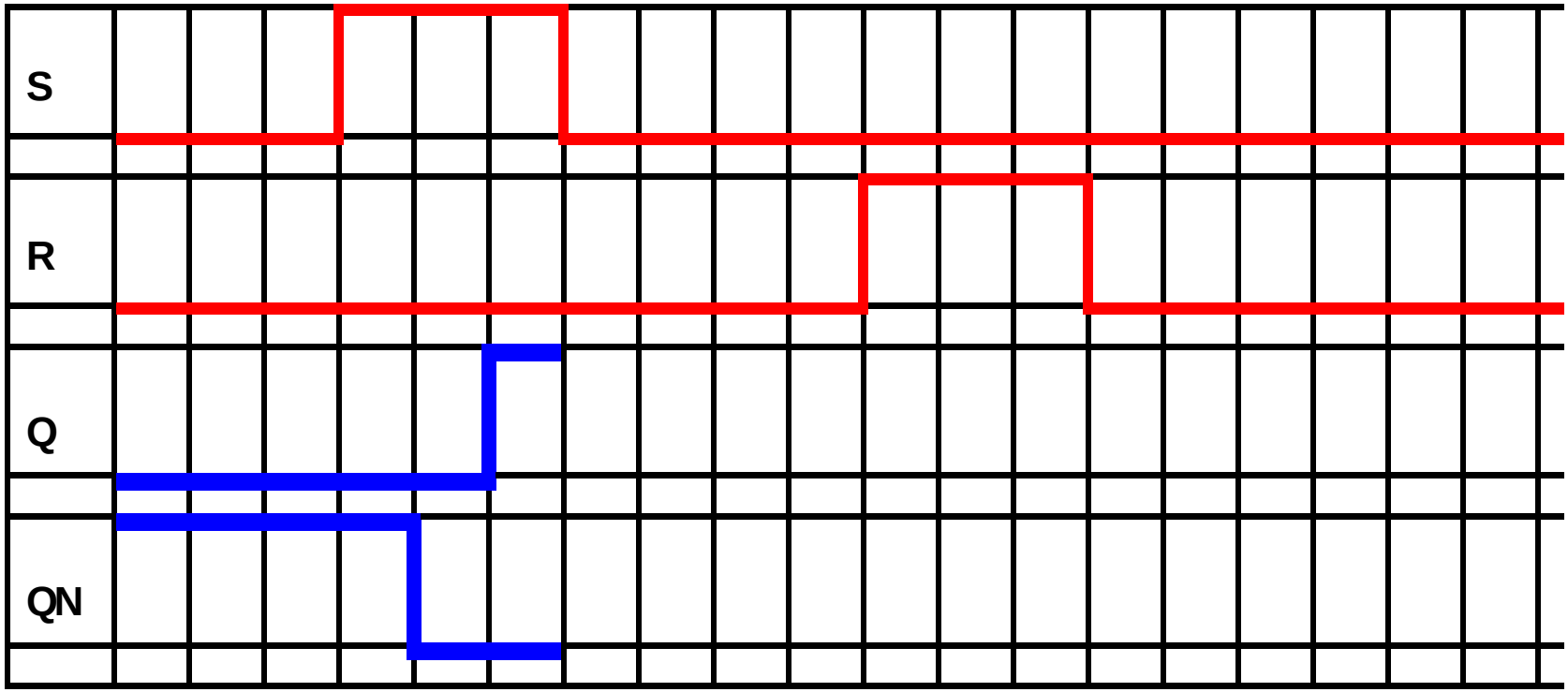


Exercise

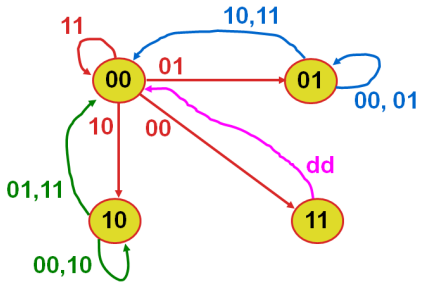


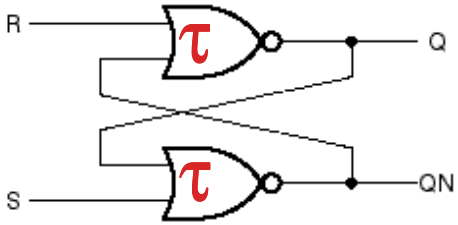


- **From the state transition diagram, construct a timing chart**

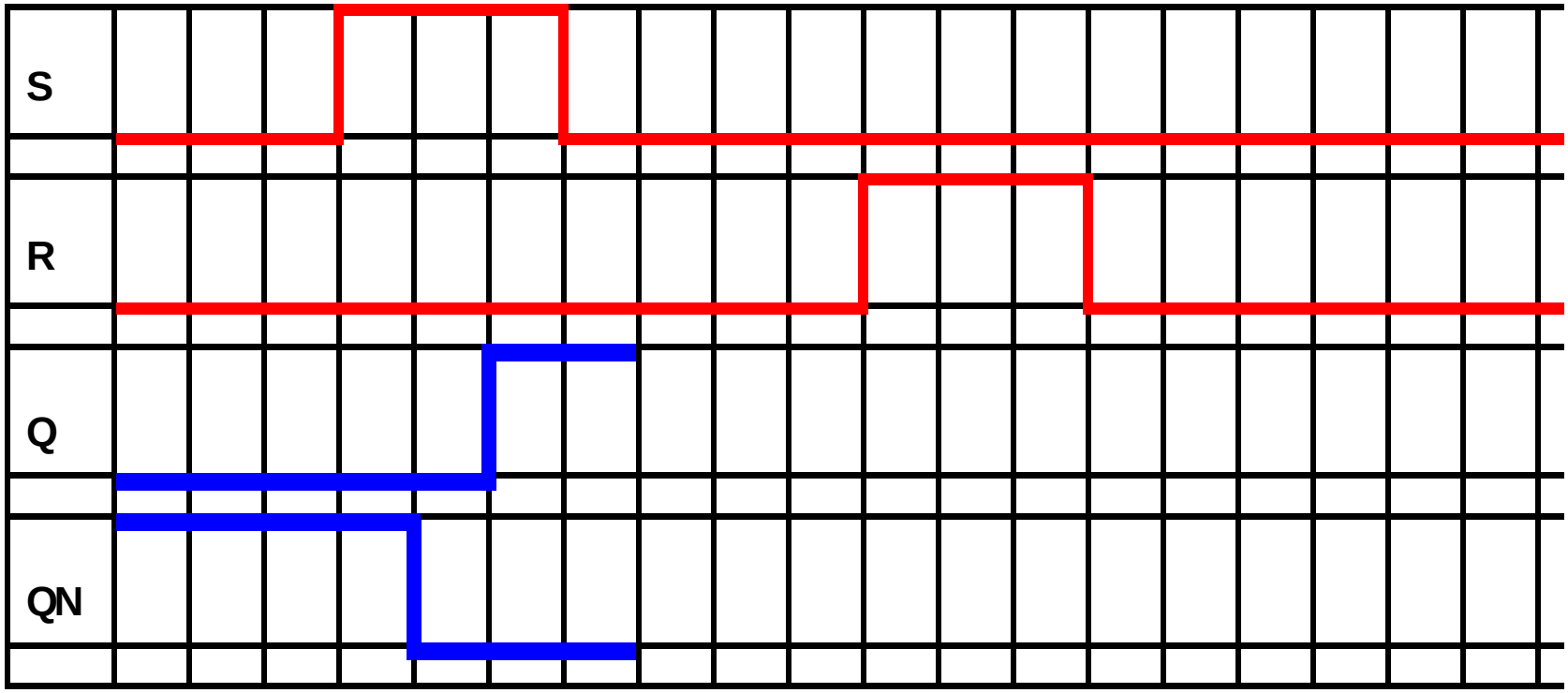


Exercise

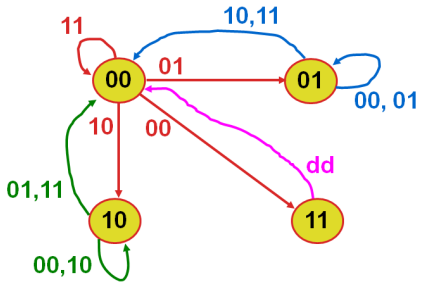


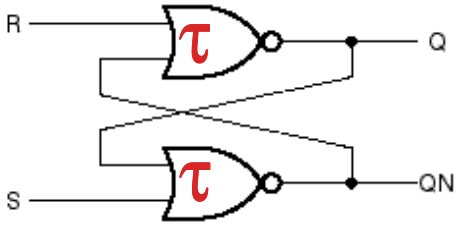


- **From the state transition diagram, construct a timing chart**

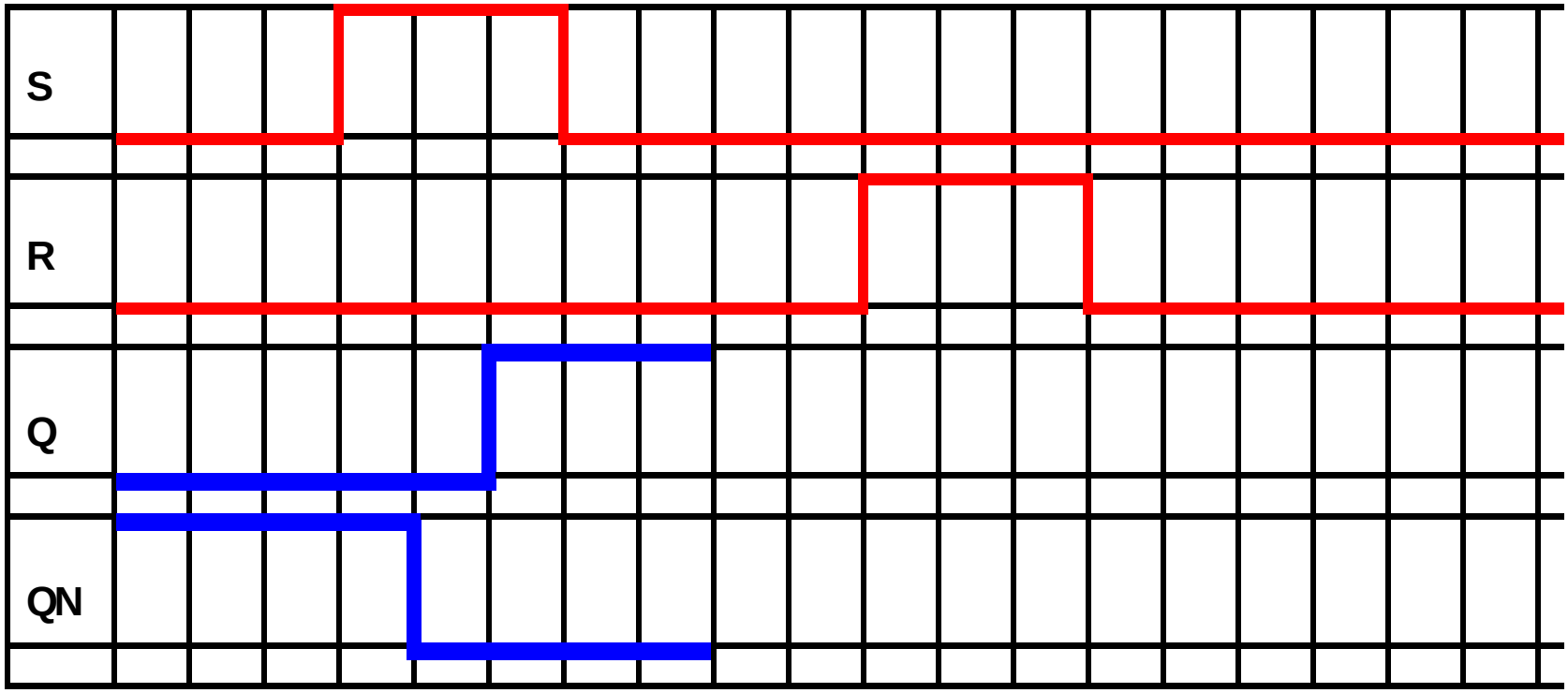


Exercise

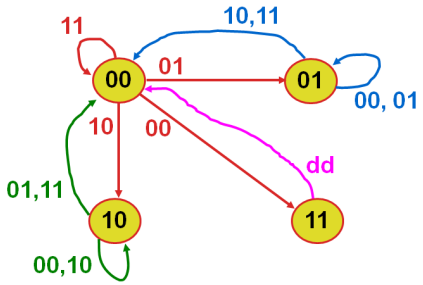


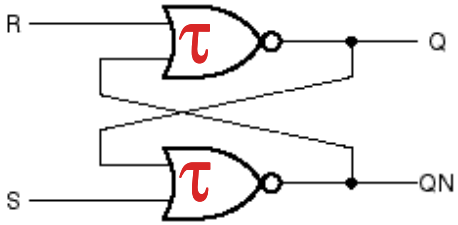


- **From the state transition diagram, construct a timing chart**

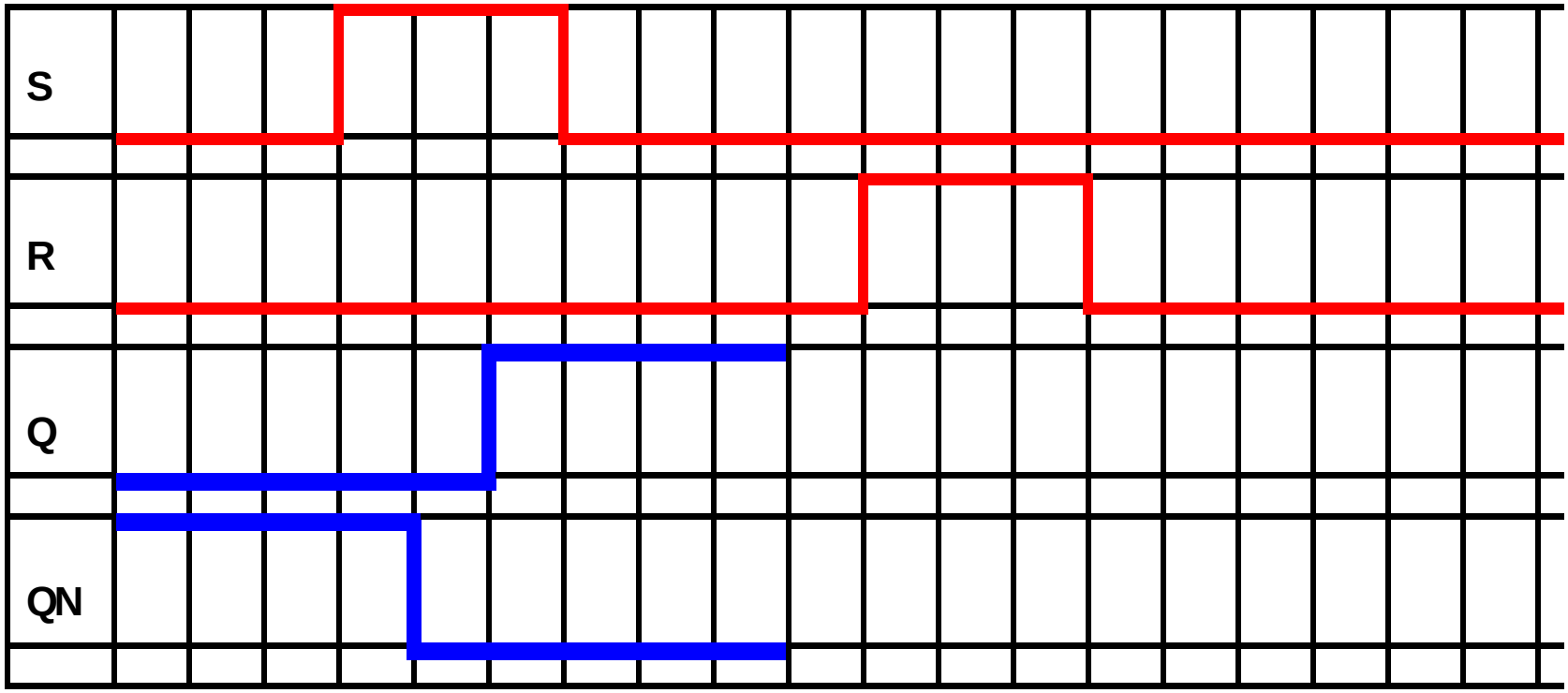


Exercise

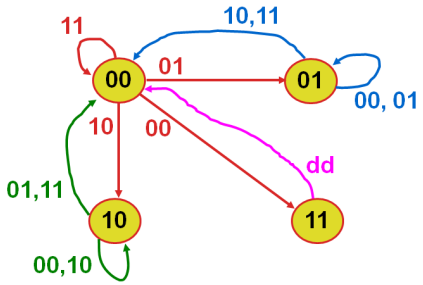


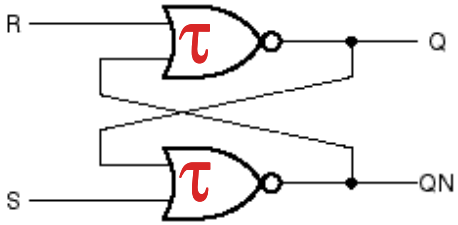


- **From the state transition diagram, construct a timing chart**

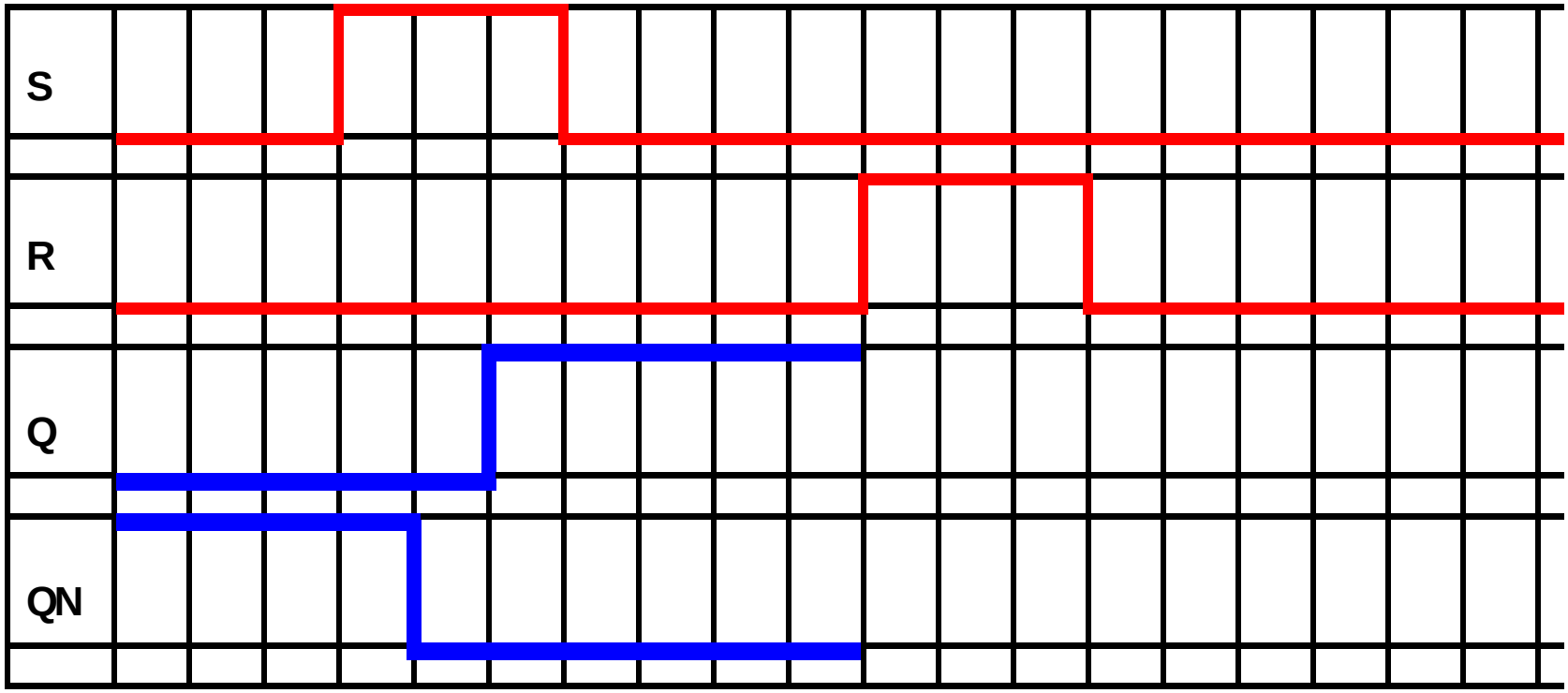


Exercise

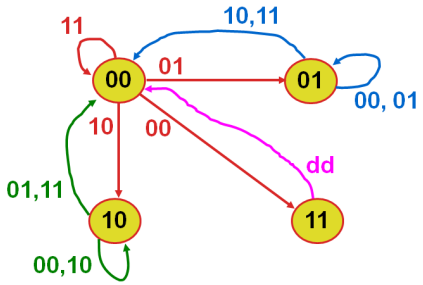


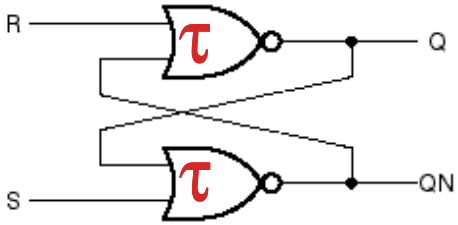


- **From the state transition diagram, construct a timing chart**

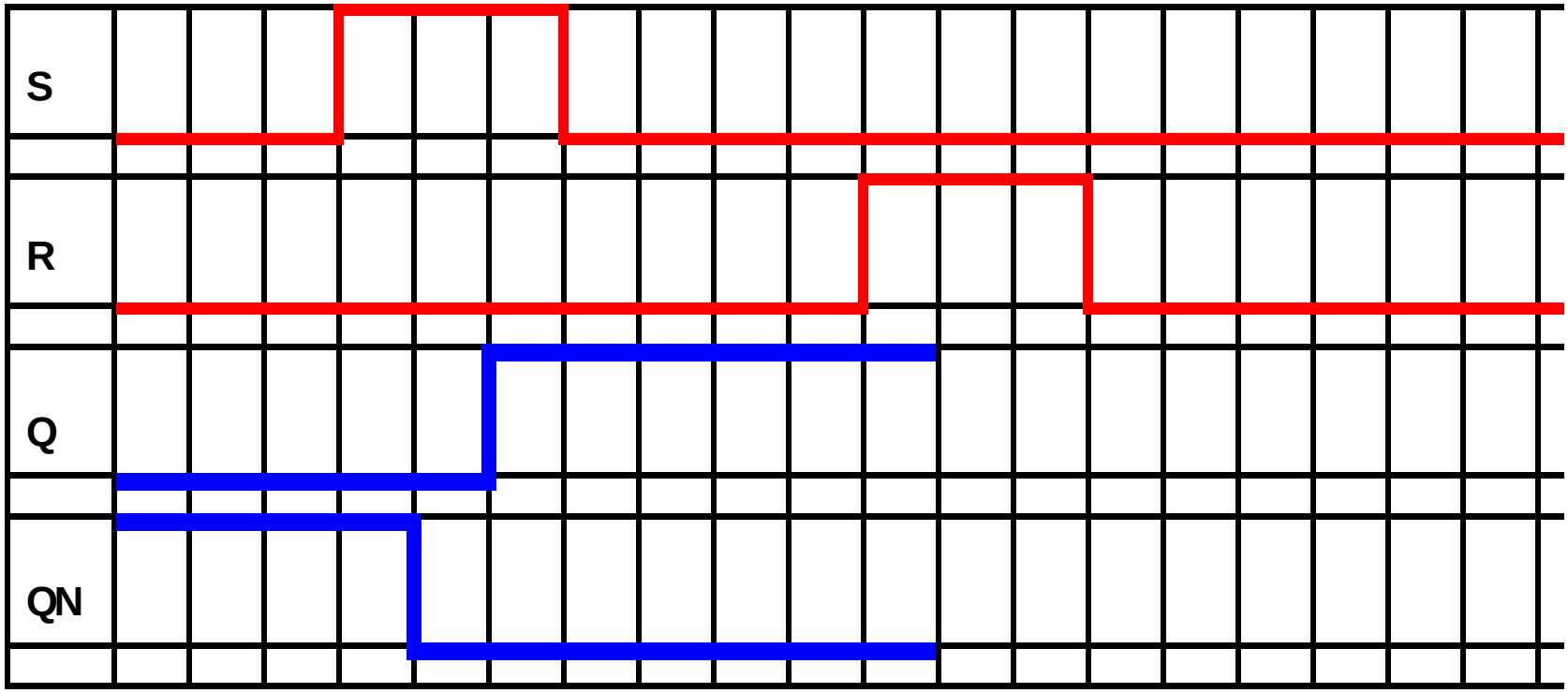


Exercise

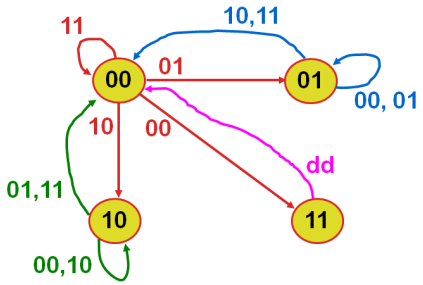


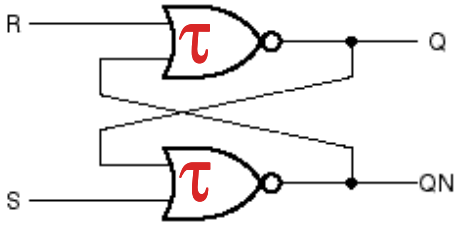


- **From the state transition diagram, construct a timing chart**

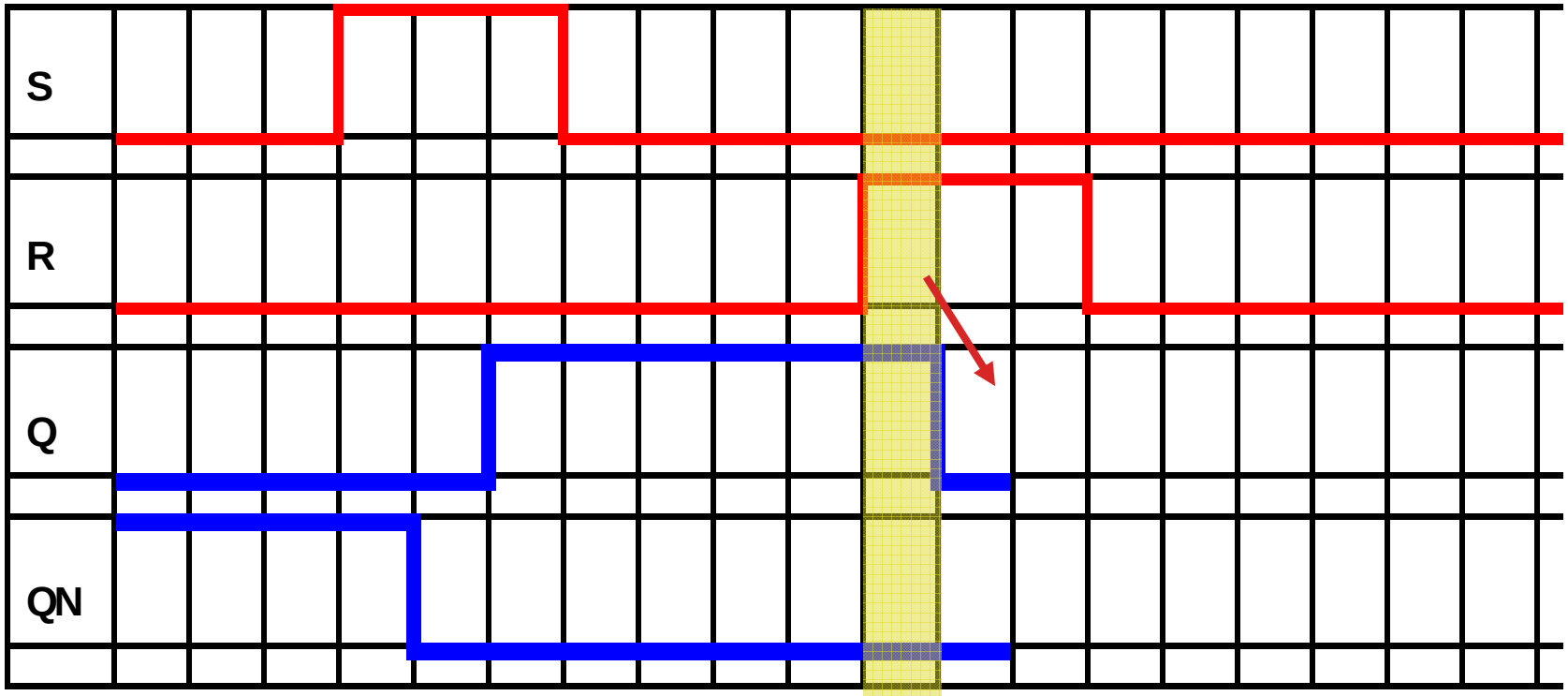


Exercise

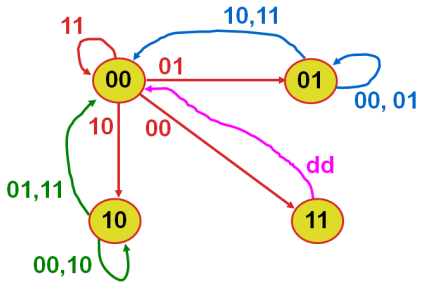


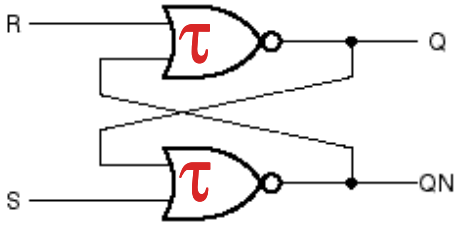


- **From the state transition diagram, construct a timing chart**

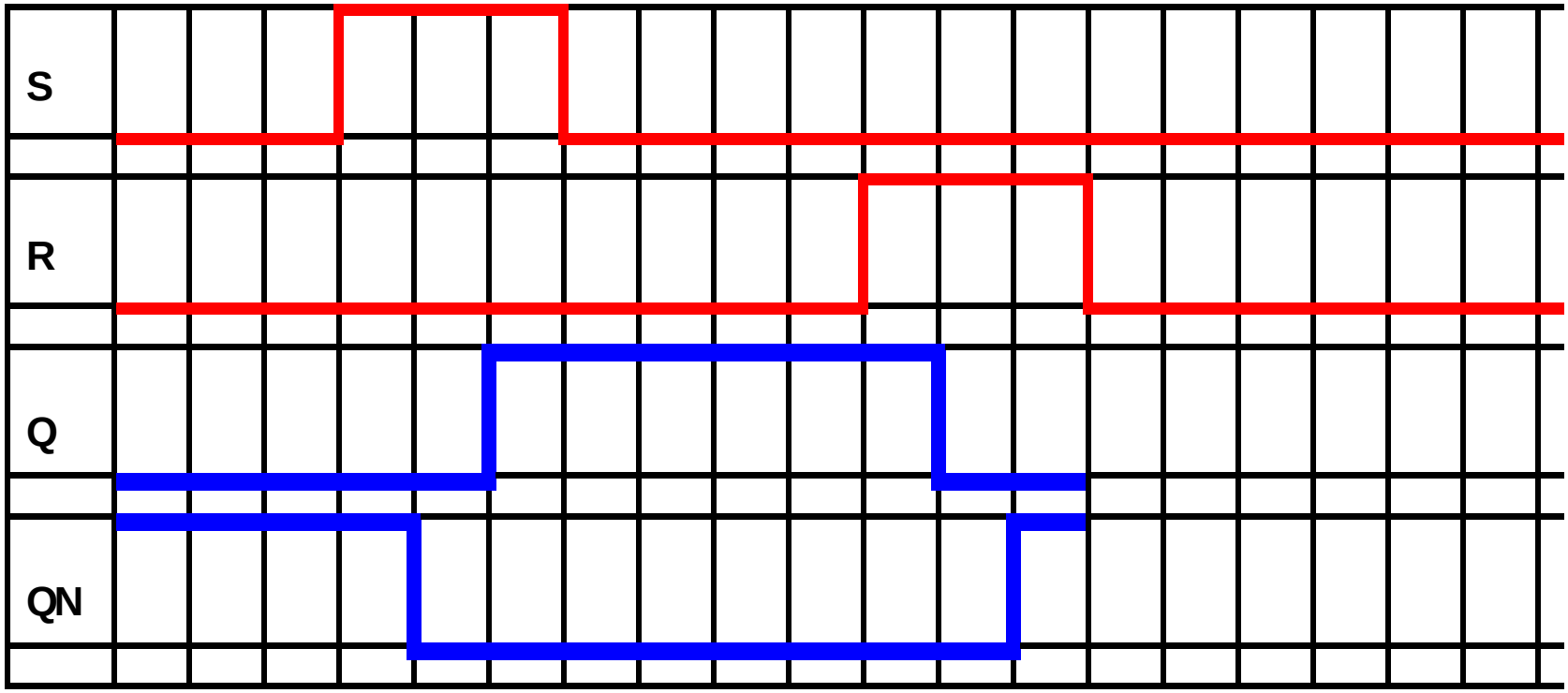


Exercise

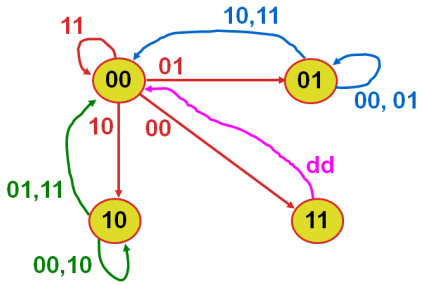


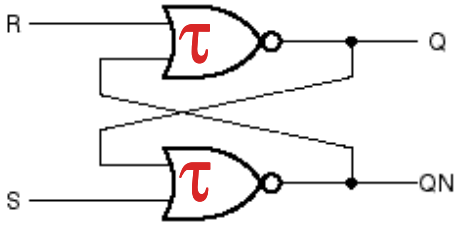


- **From the state transition diagram, construct a timing chart**

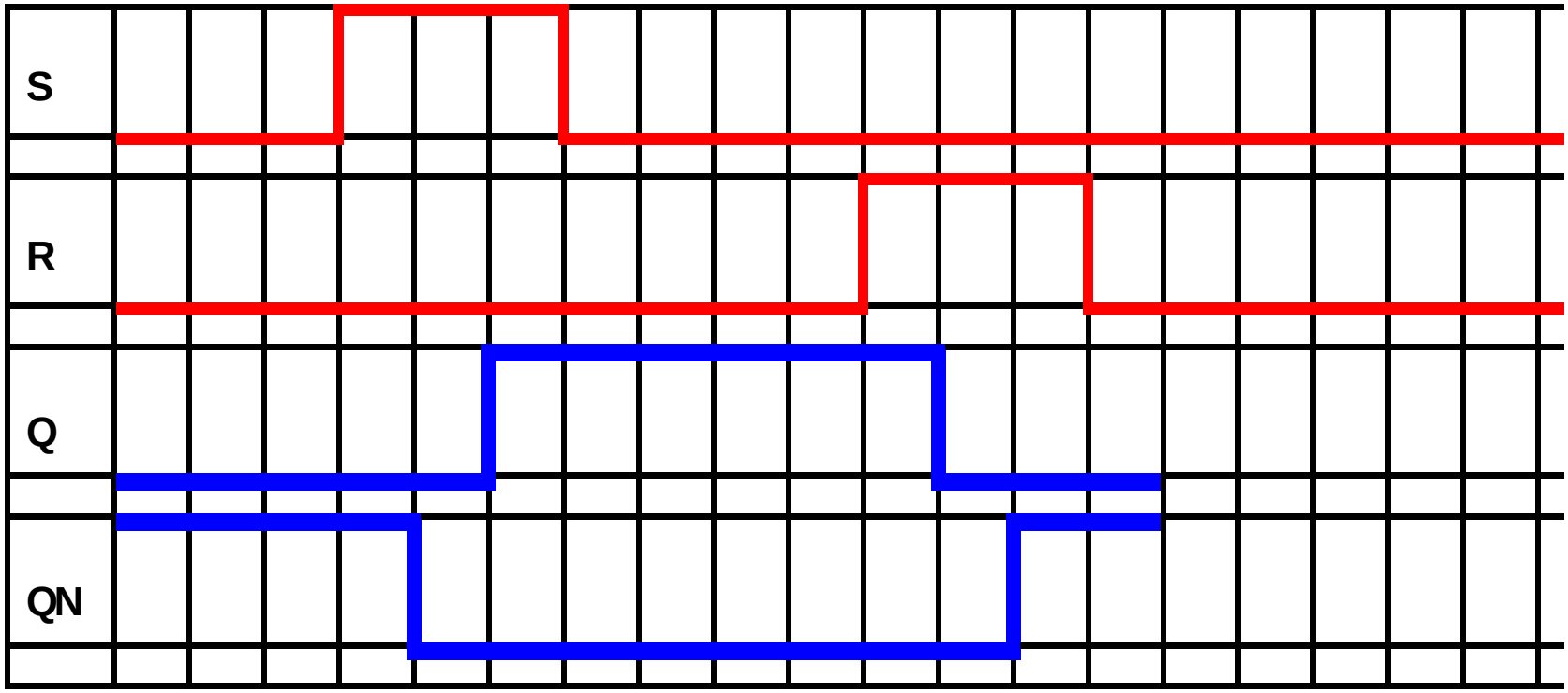


Exercise

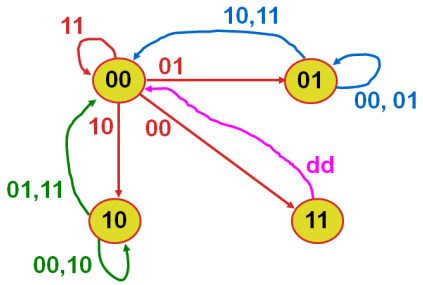


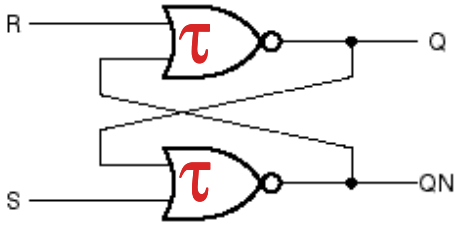


- **From the state transition diagram, construct a timing chart**

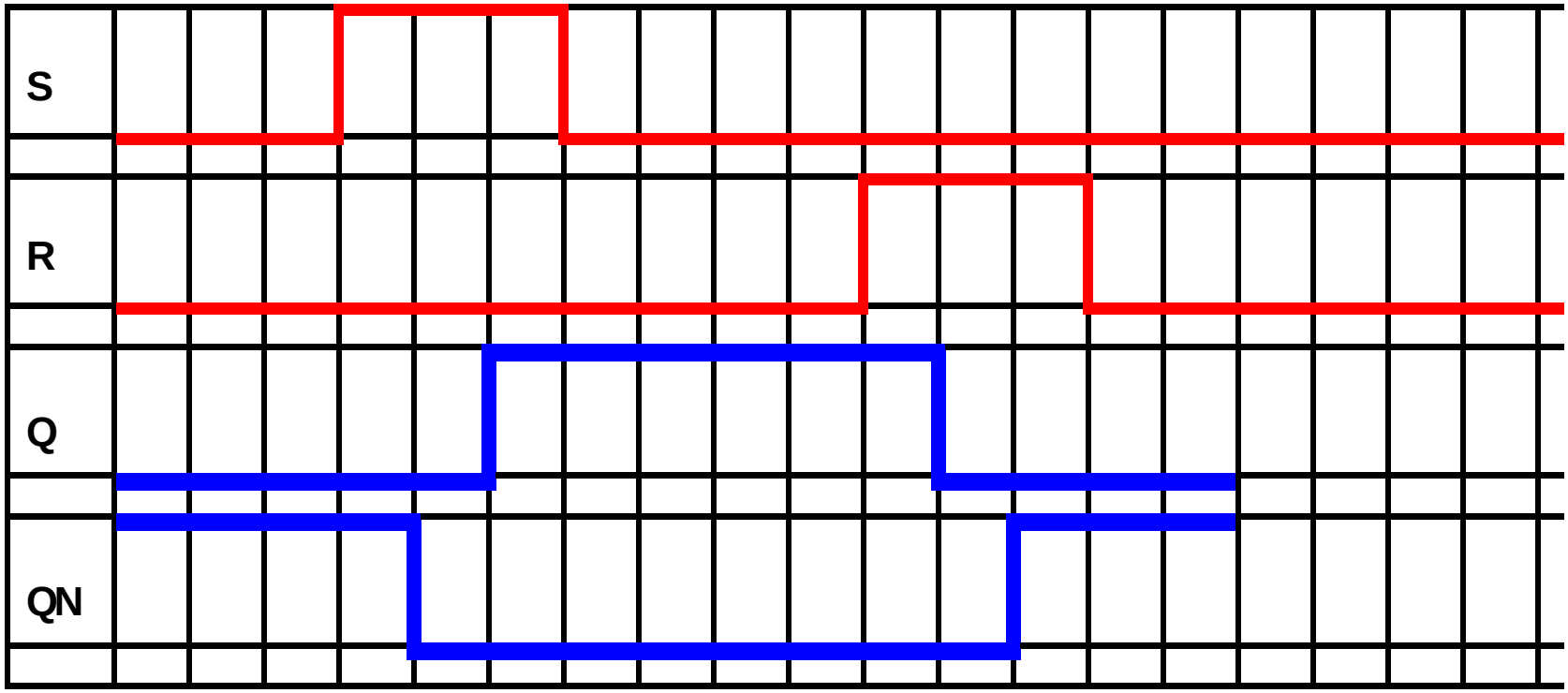


Exercise

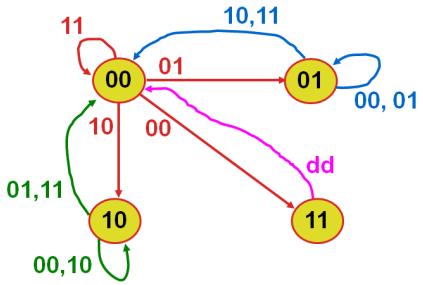


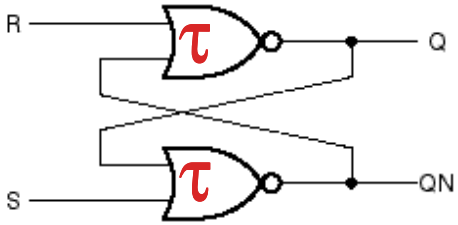


- **From the state transition diagram, construct a timing chart**

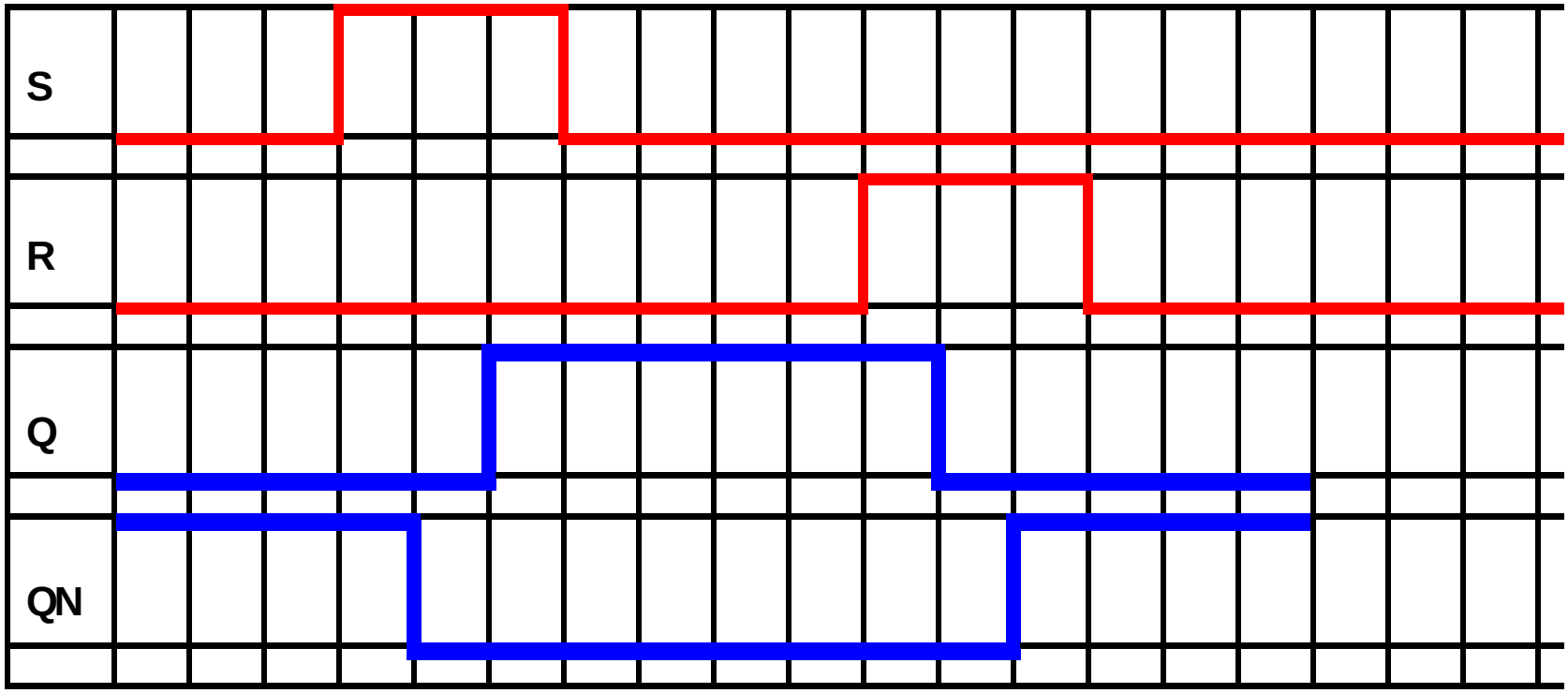


Exercise

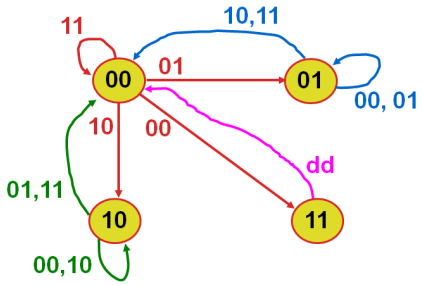


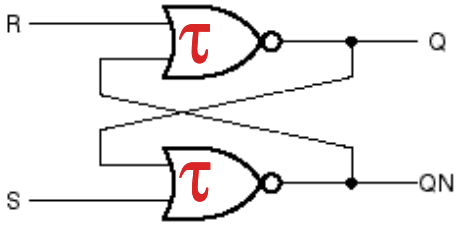


- **From the state transition diagram, construct a timing chart**

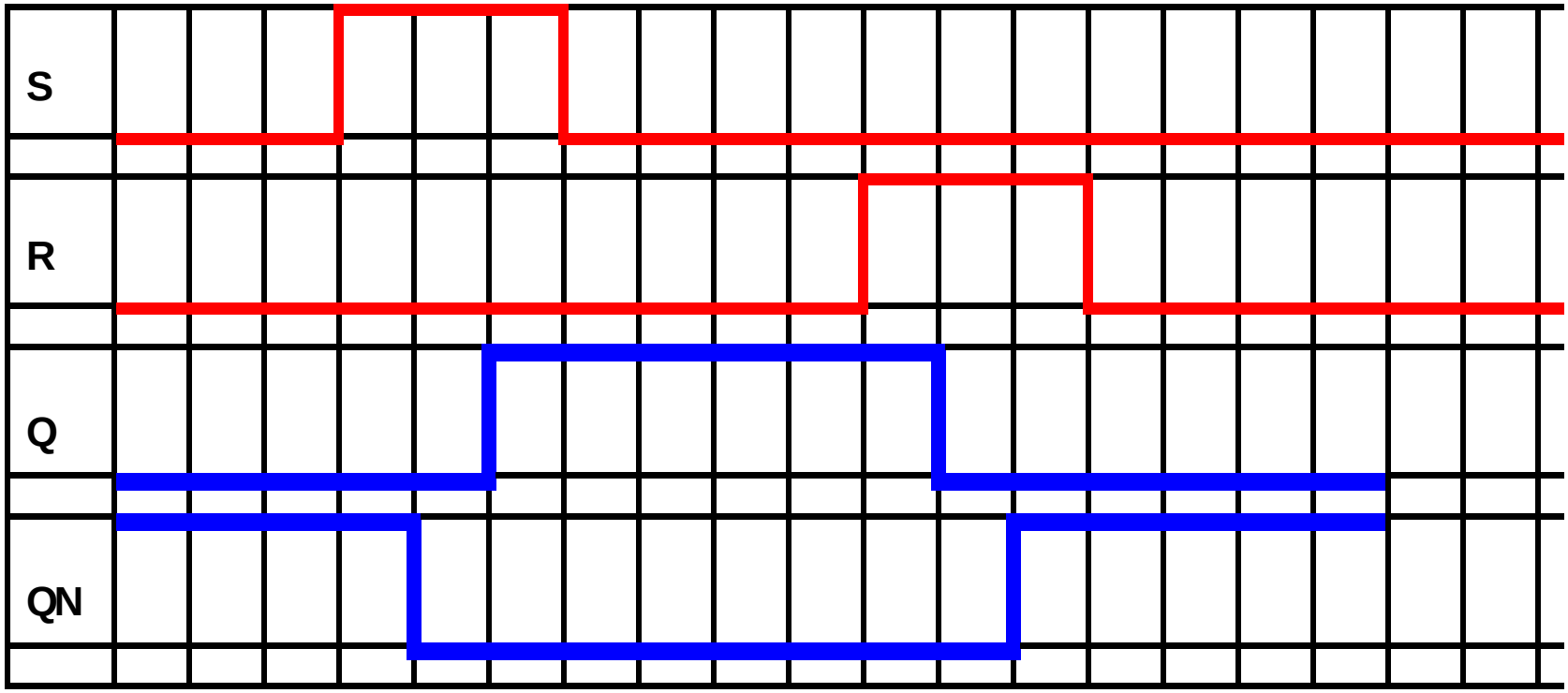


Exercise

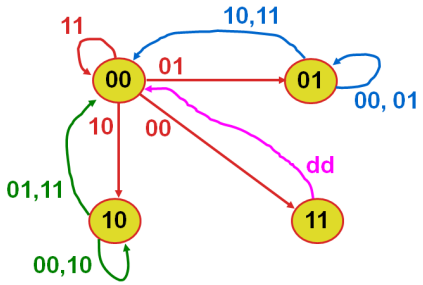


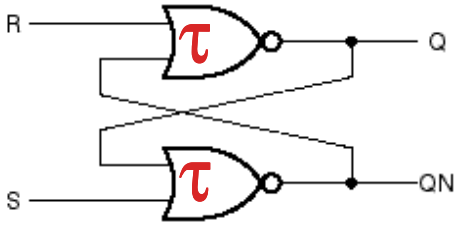


- **From the state transition diagram, construct a timing chart**

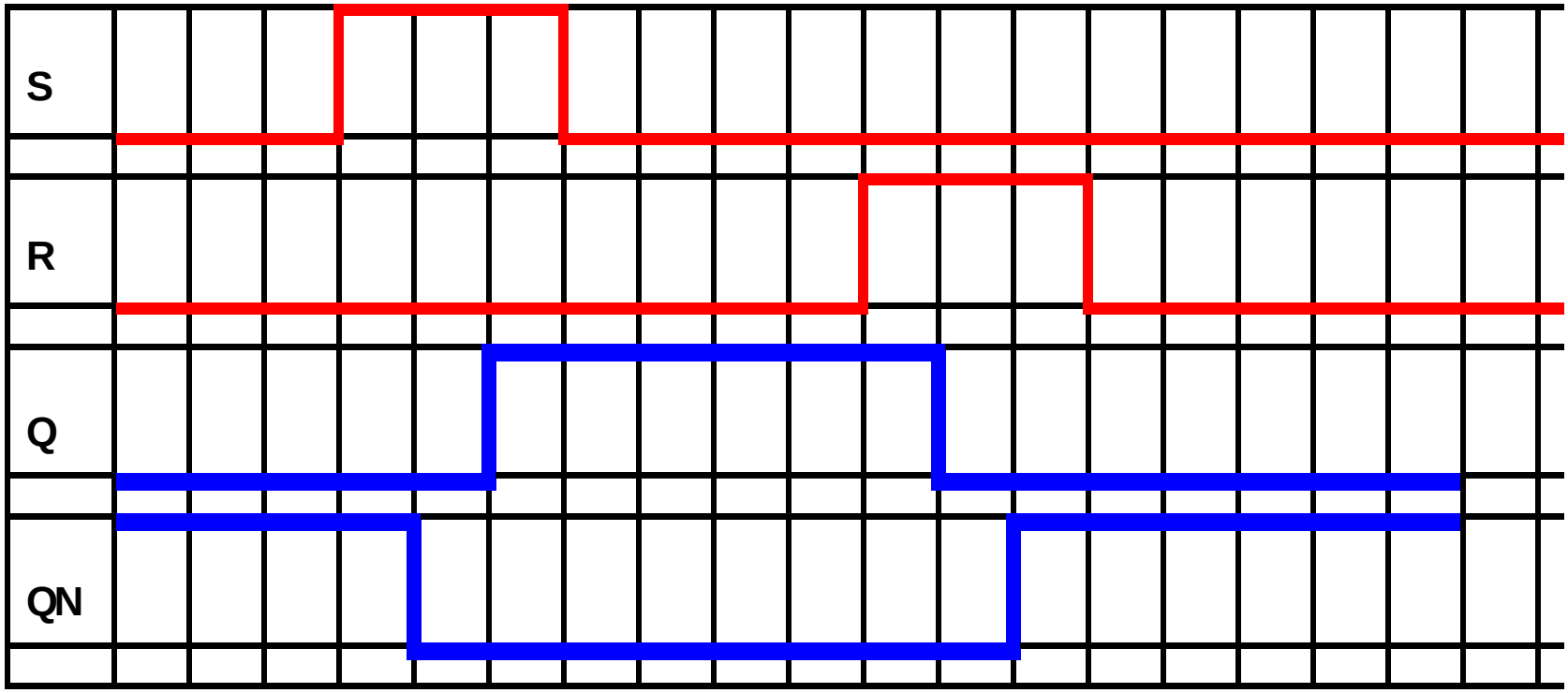


Exercise

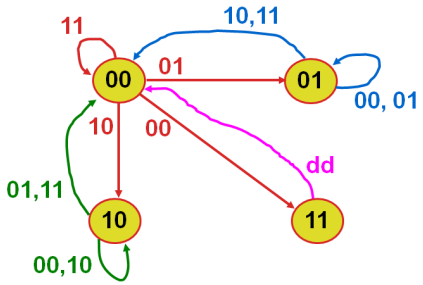


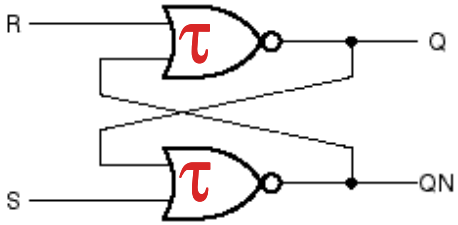


- **From the state transition diagram, construct a timing chart**

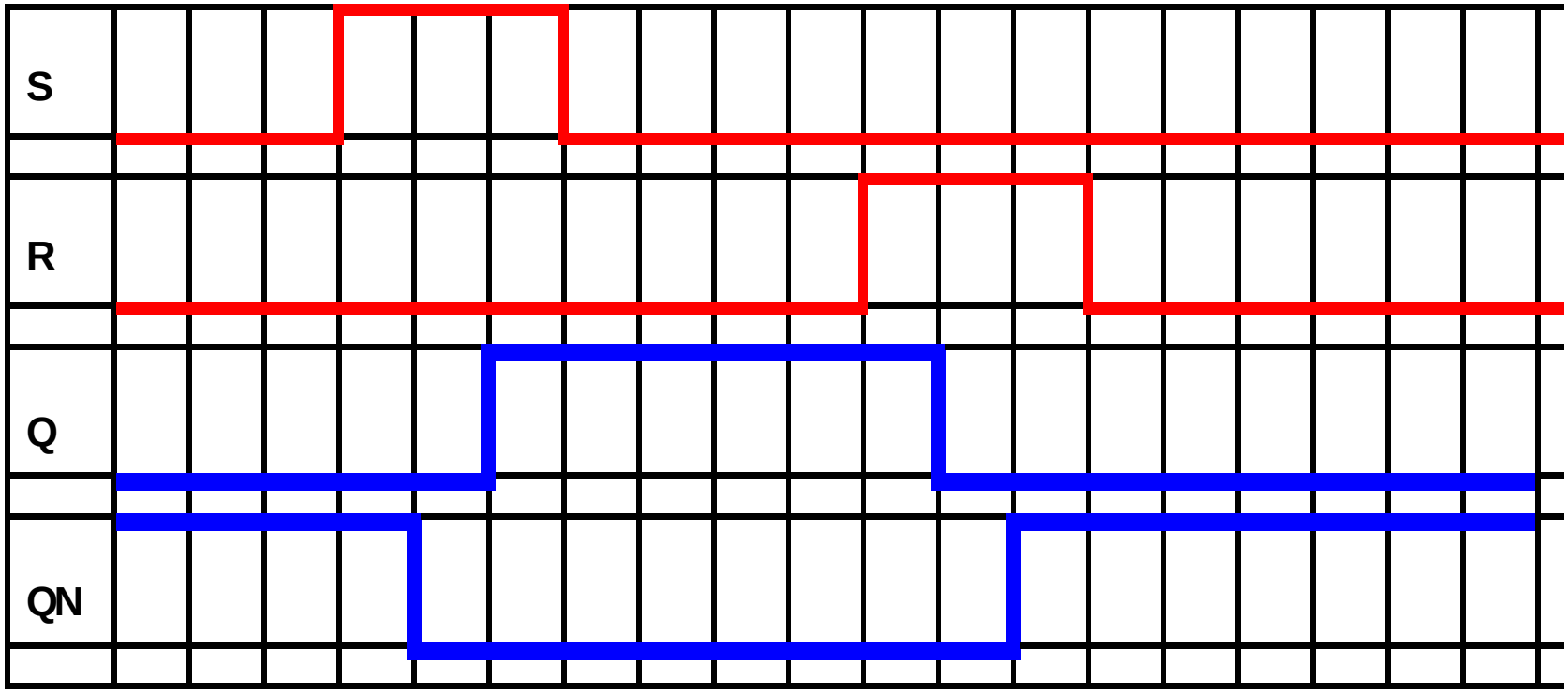


Exercise

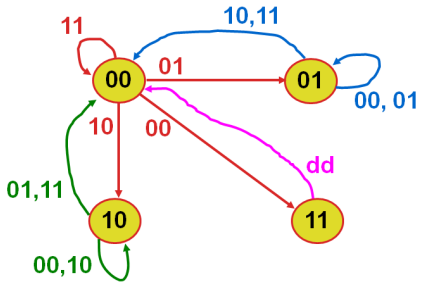


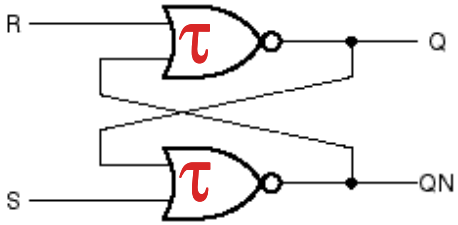


- **From the state transition diagram, construct a timing chart**

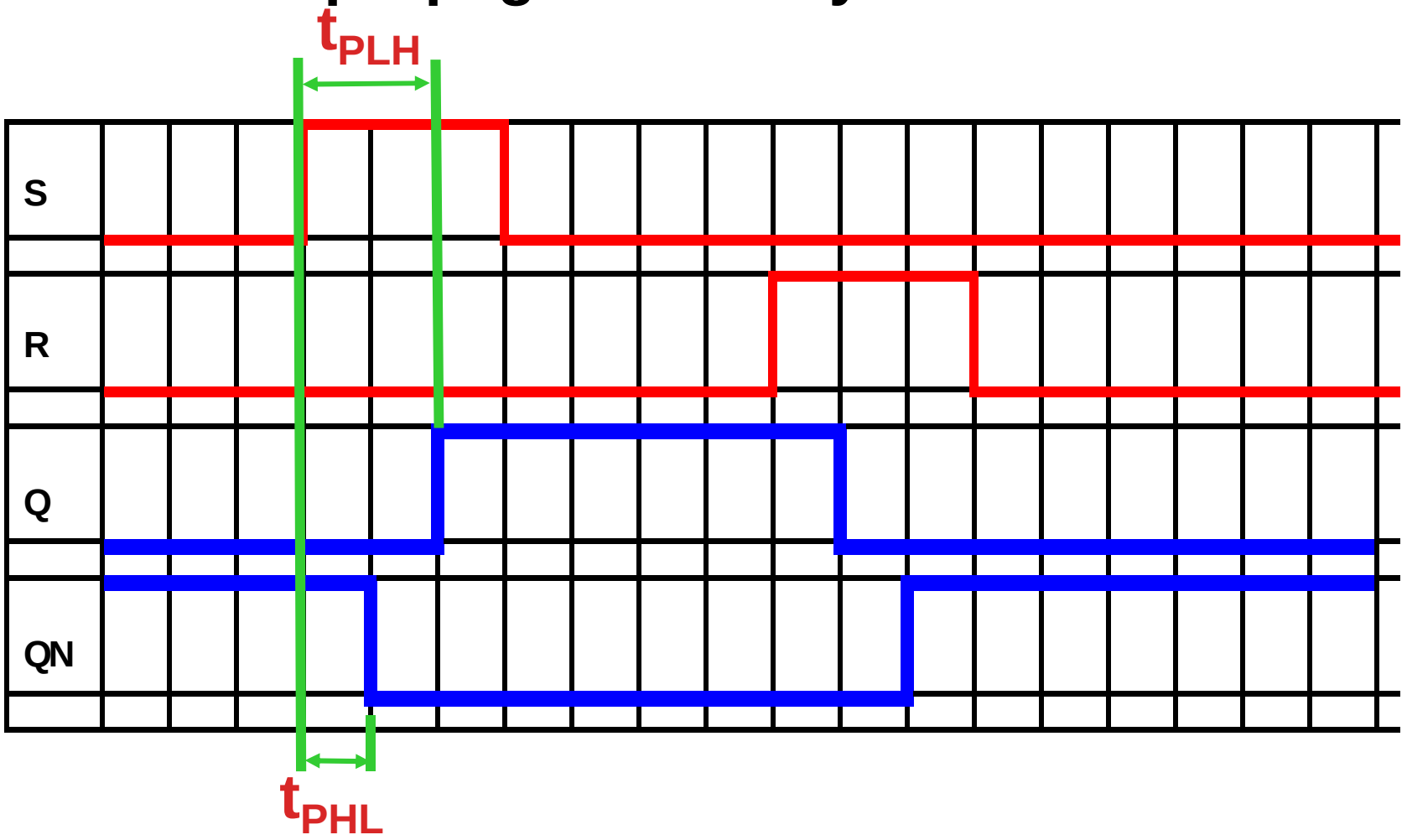


Exercise

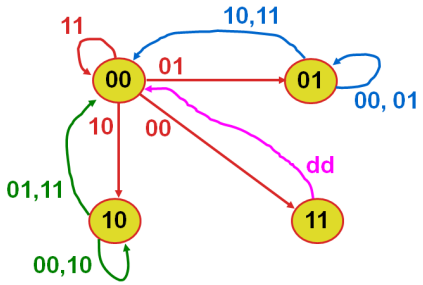


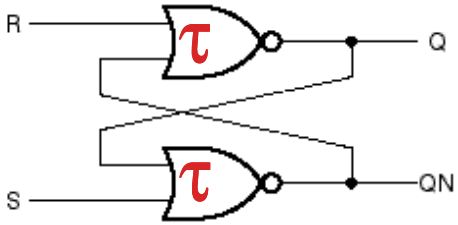


- **Note the propagation delays**

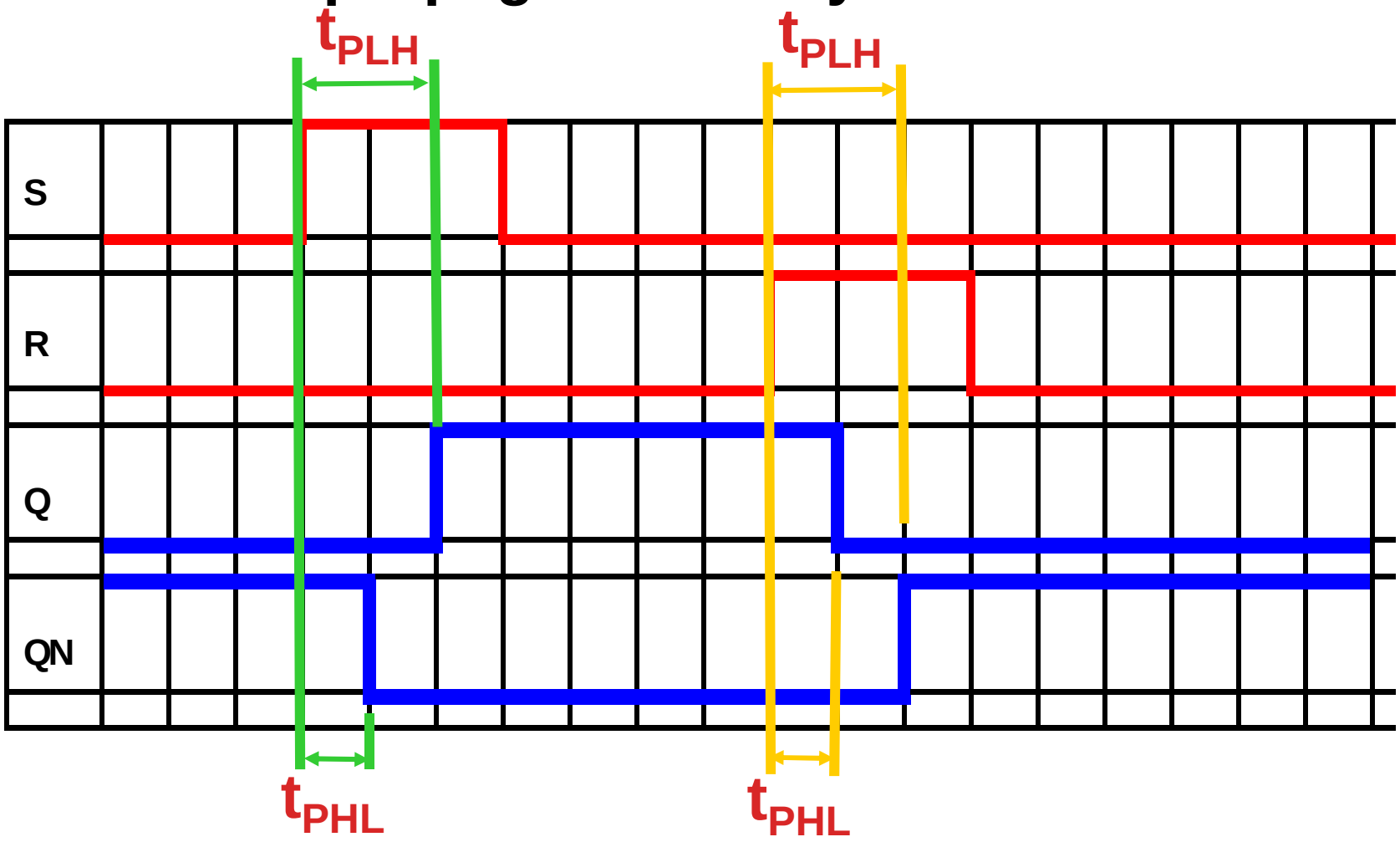


Exercise

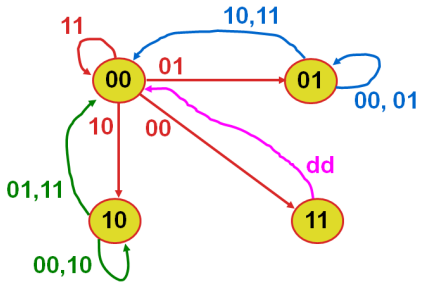


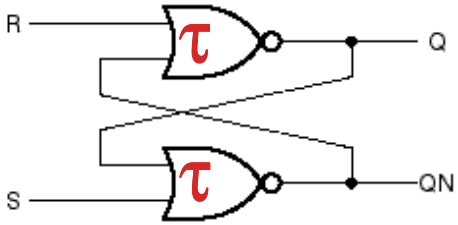


- **Note the propagation delays**

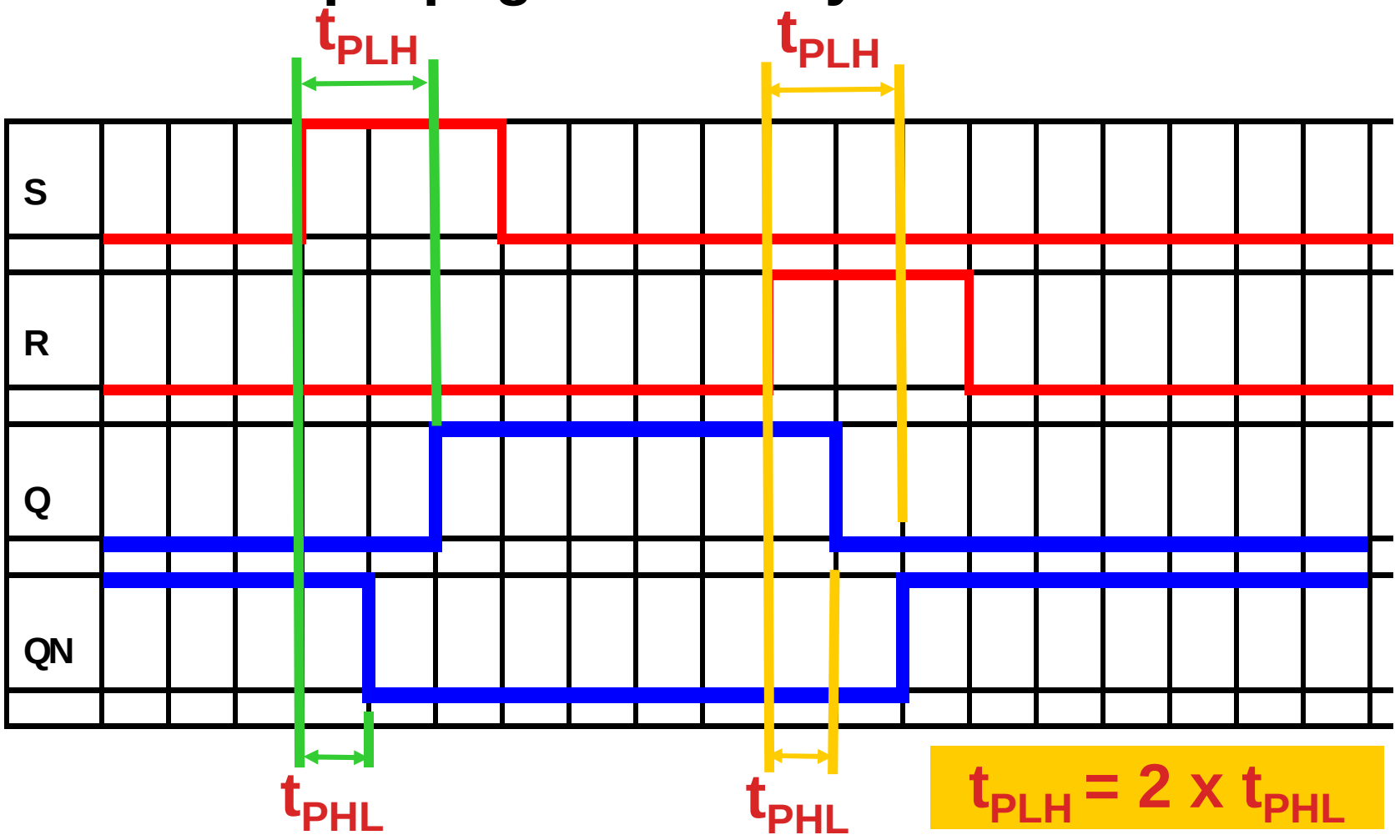


Exercise

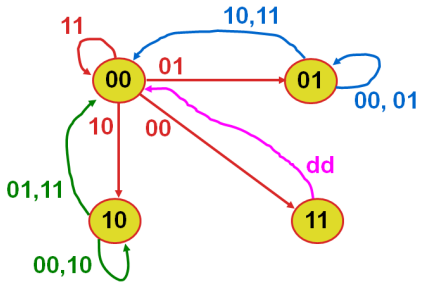


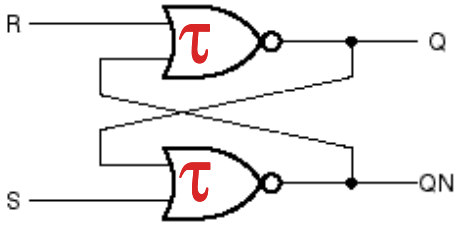


- **Note the propagation delays**

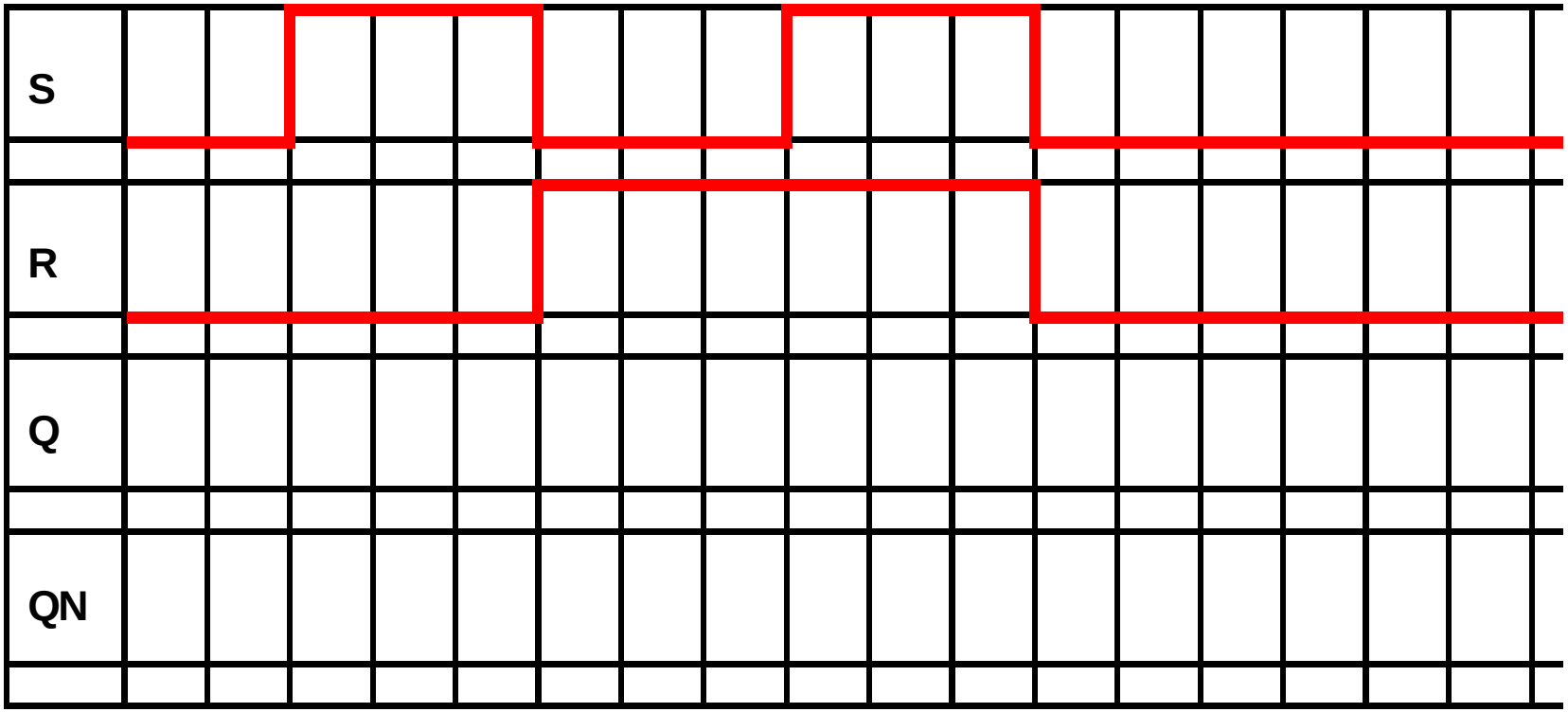


Exercise

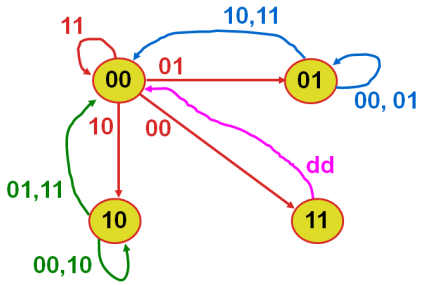


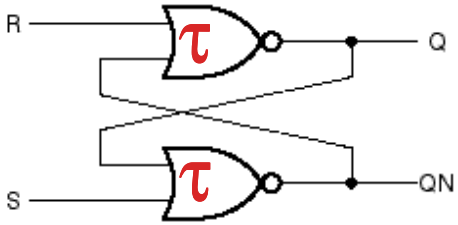


- **Investigate the response of an S-R latch to the “1-1” input combination**

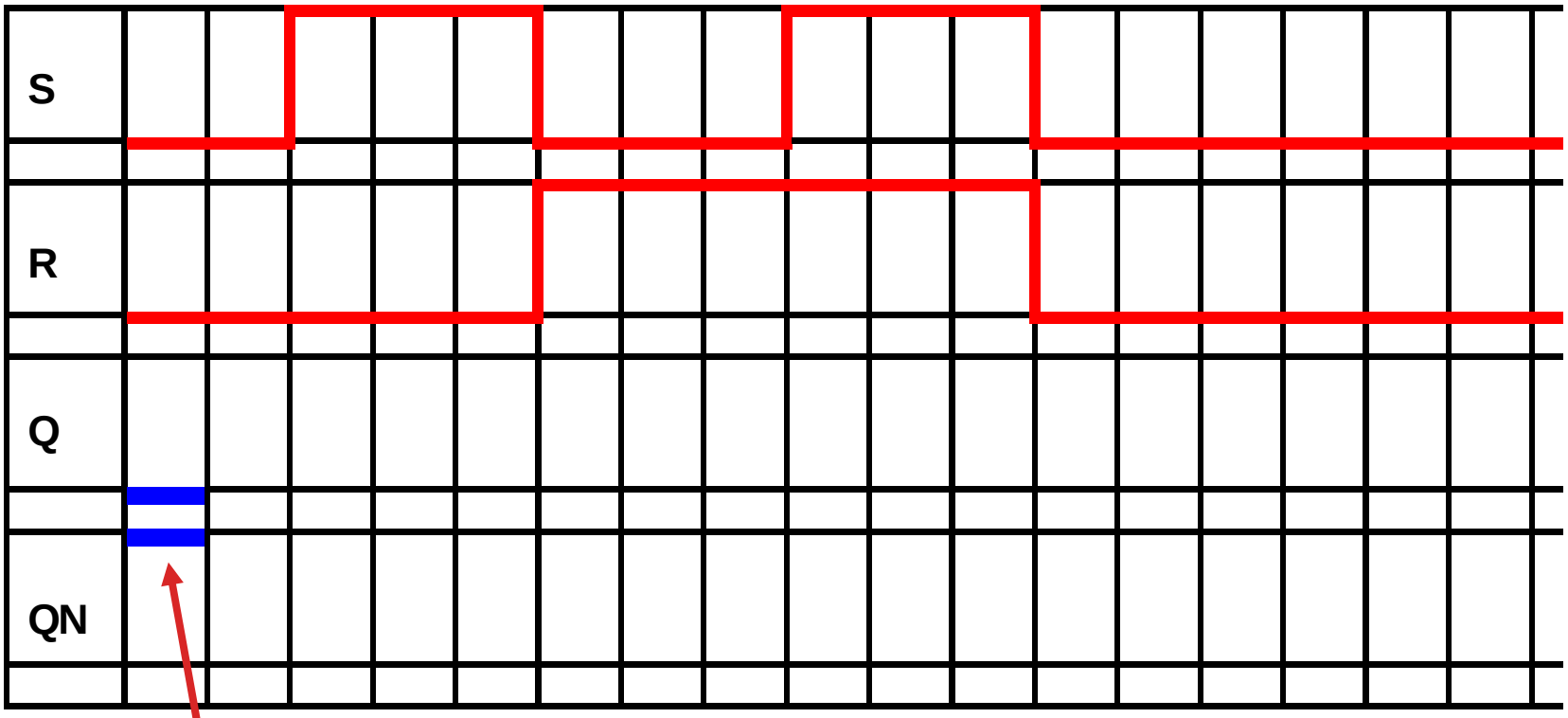


Exercise

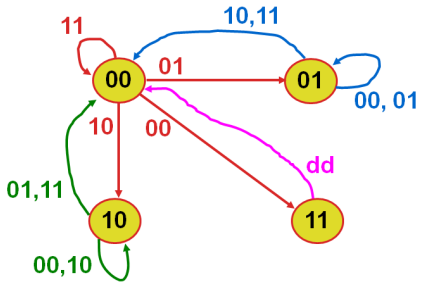


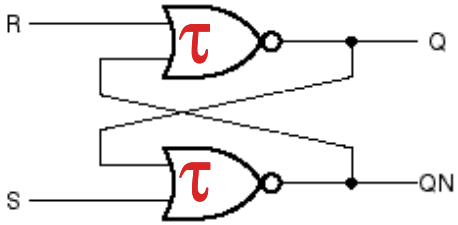


- **Investigate the response of an S-R latch to the “1-1” input combination**

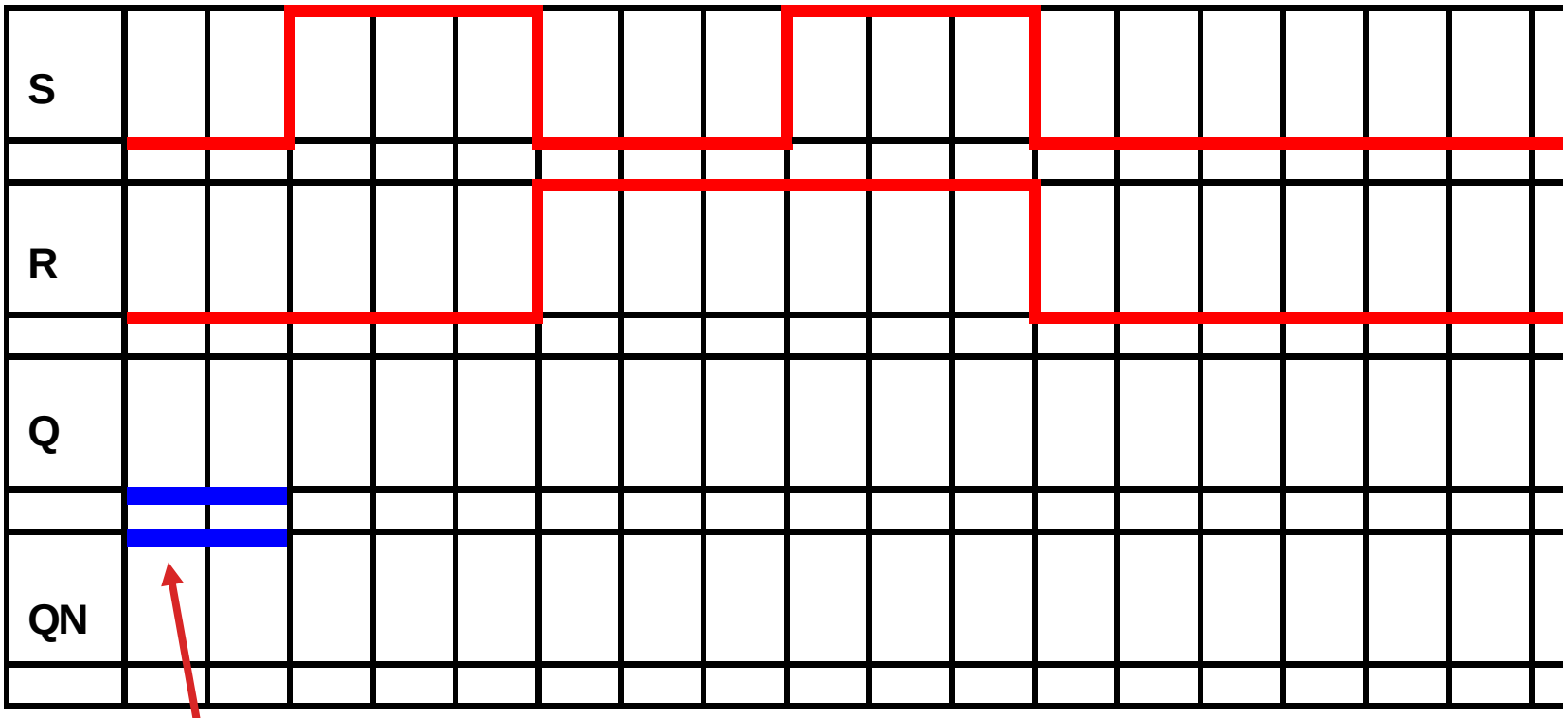


Exercise

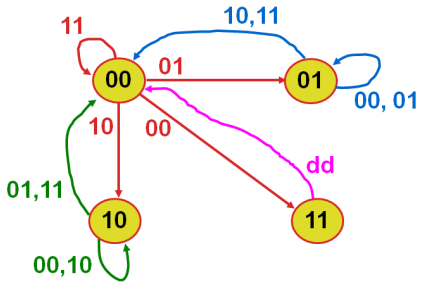


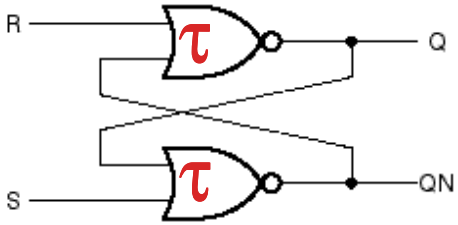


- **Investigate the response of an S-R latch to the “1-1” input combination**

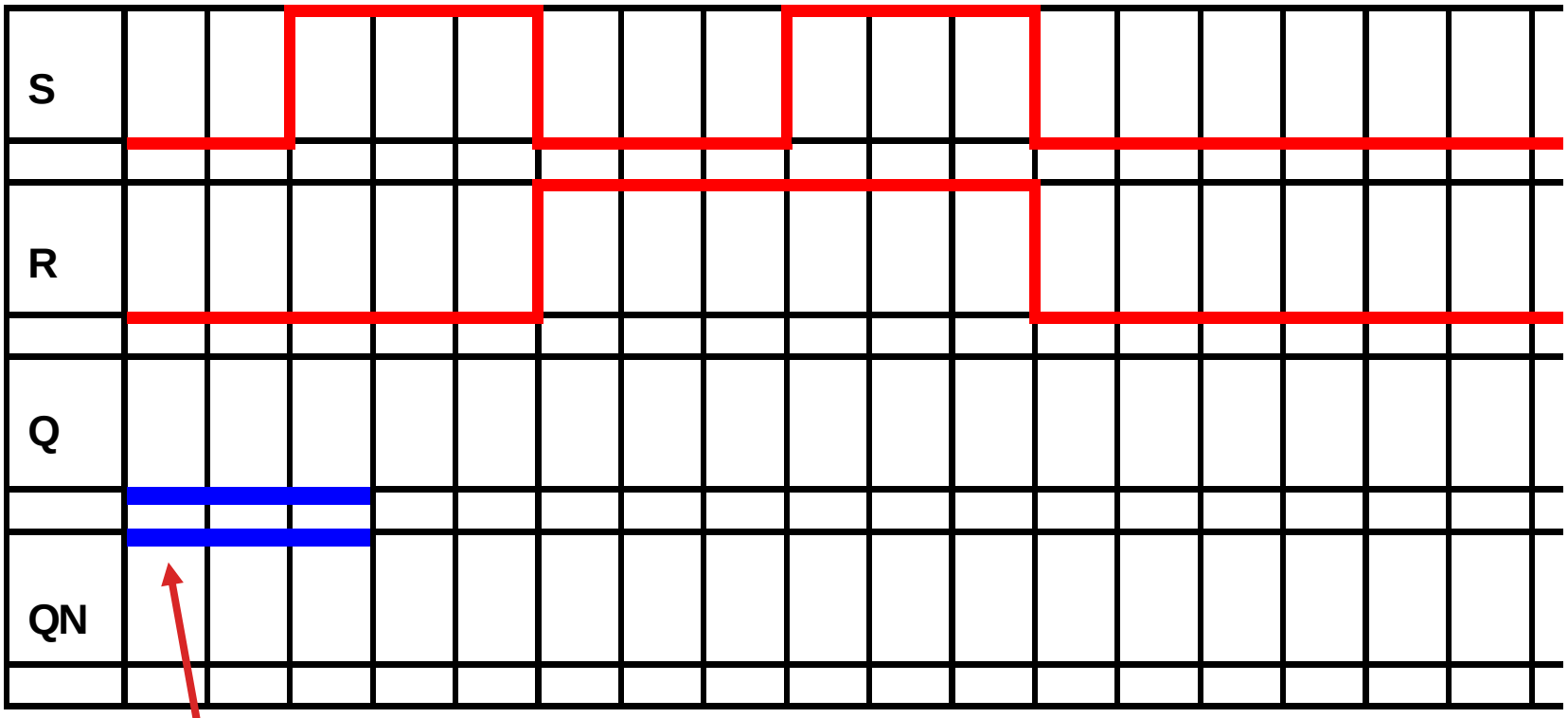


Exercise

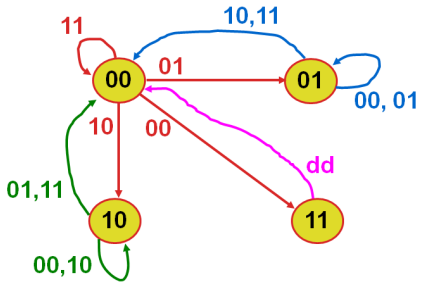


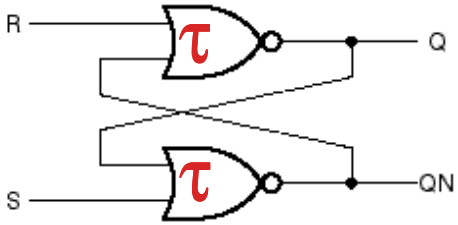


- **Investigate the response of an S-R latch to the “1-1” input combination**

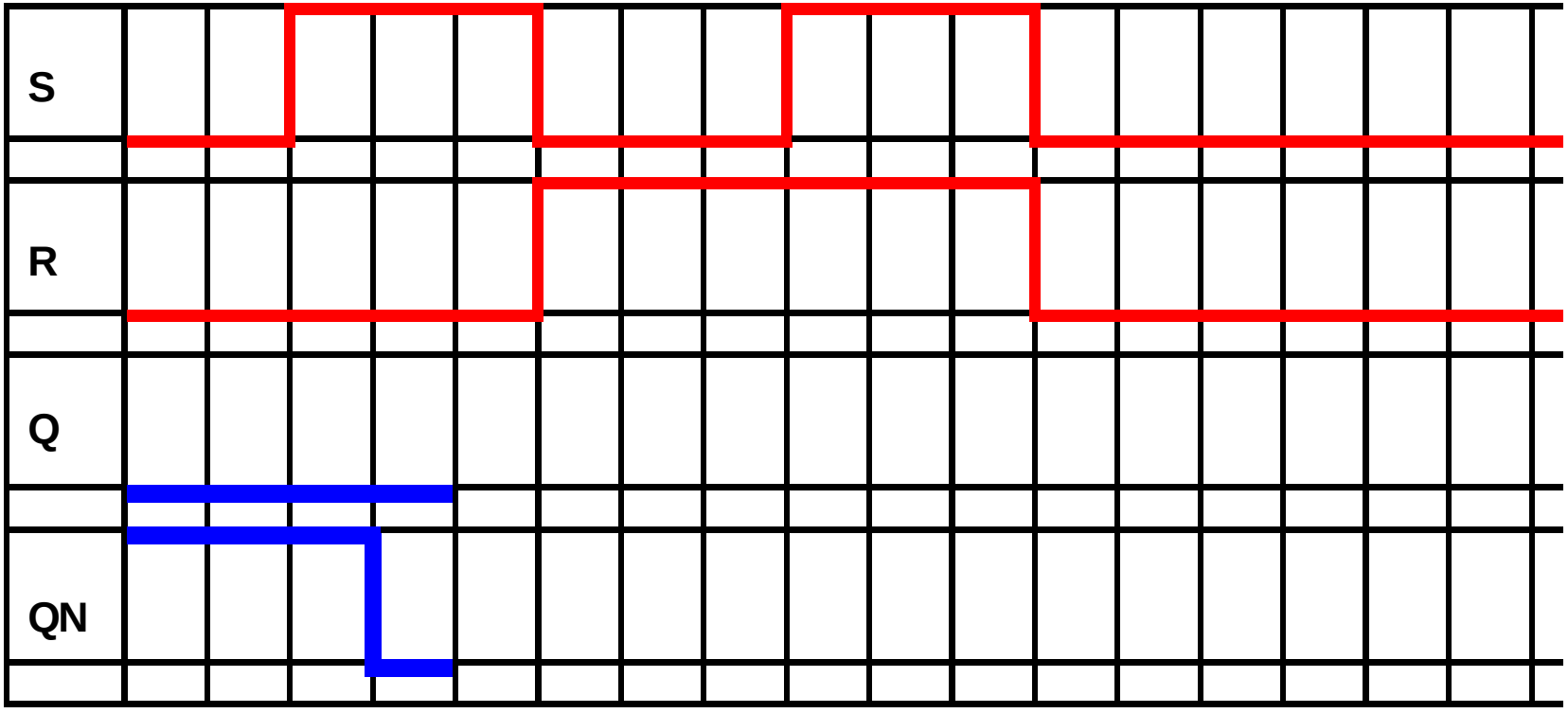


Exercise

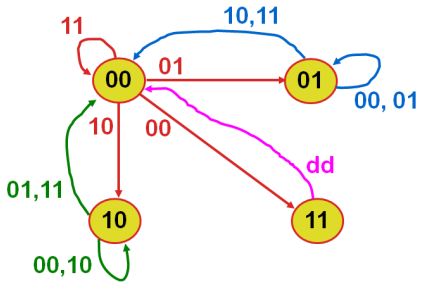


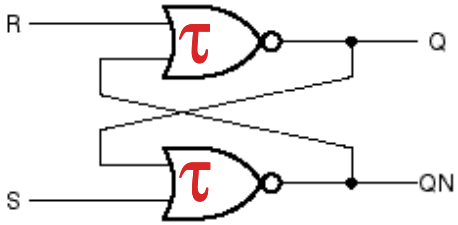


- **Investigate the response of an S-R latch to the “1-1” input combination**

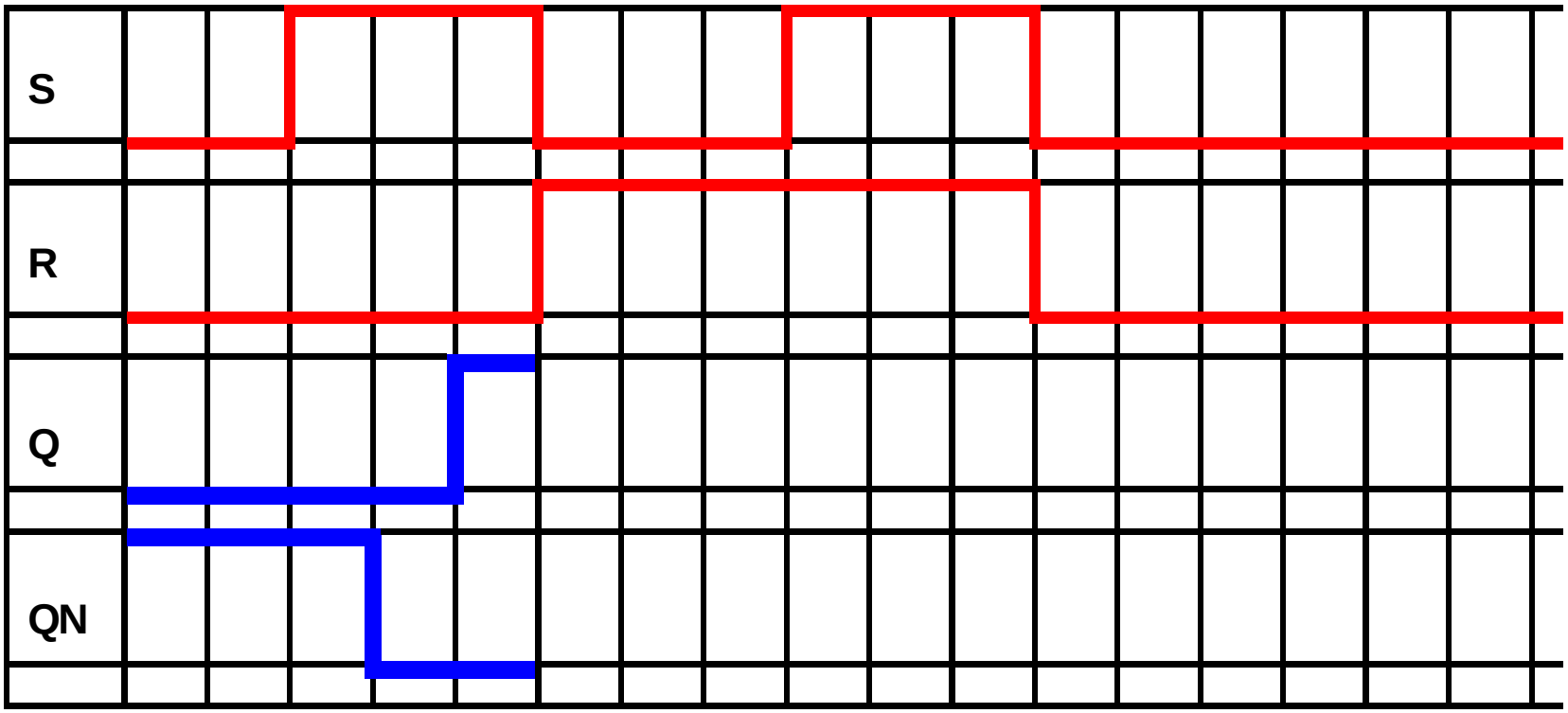


Exercise

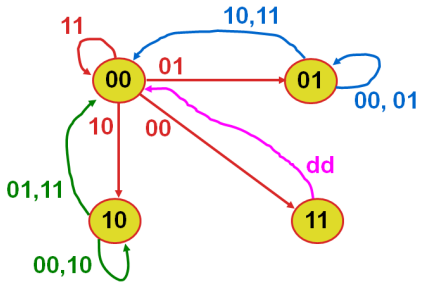


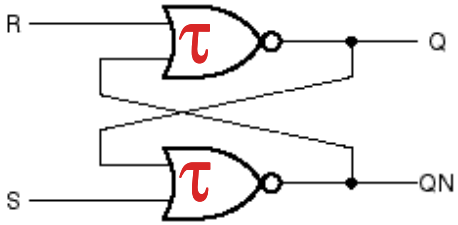


- **Investigate the response of an S-R latch to the “1-1” input combination**

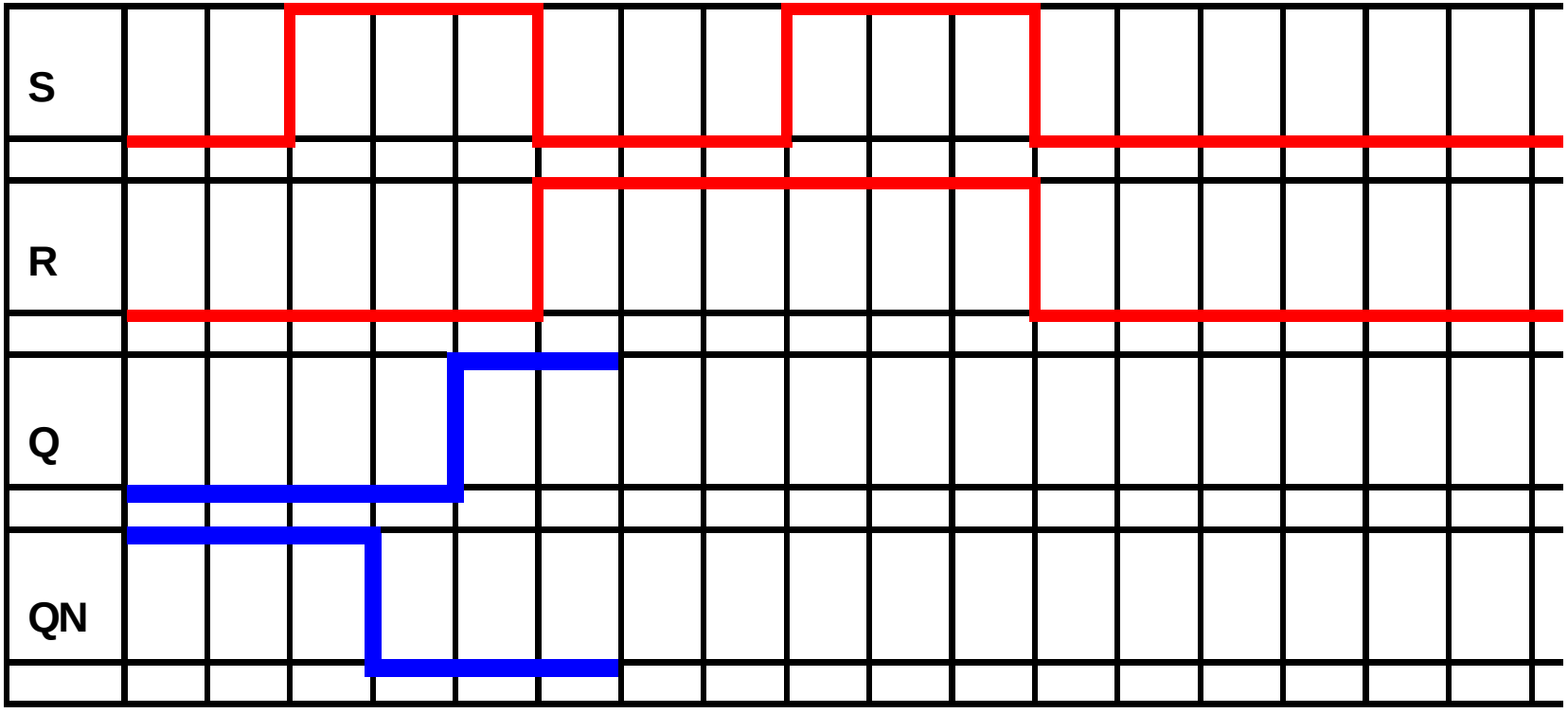


Exercise

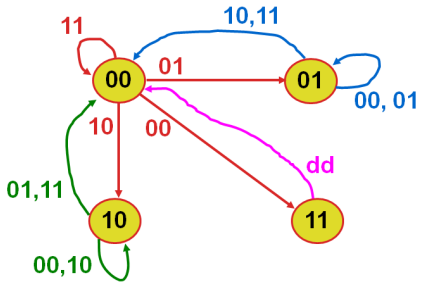


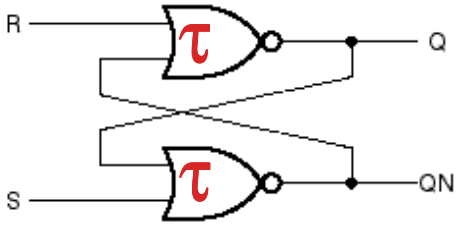


- **Investigate the response of an S-R latch to the “1-1” input combination**

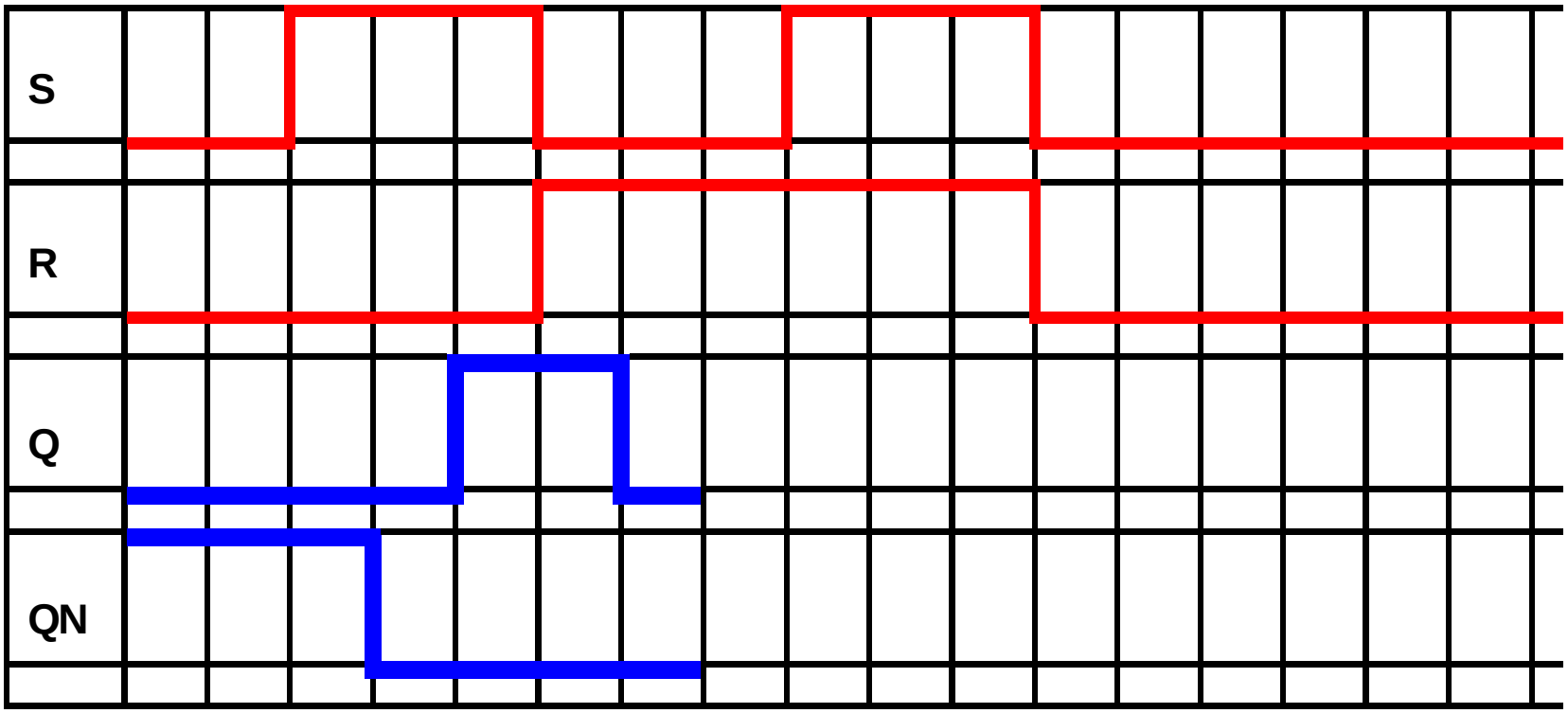


Exercise

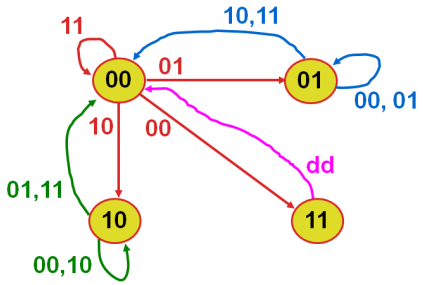


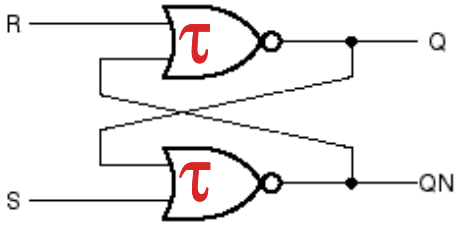


- **Investigate the response of an S-R latch to the “1-1” input combination**

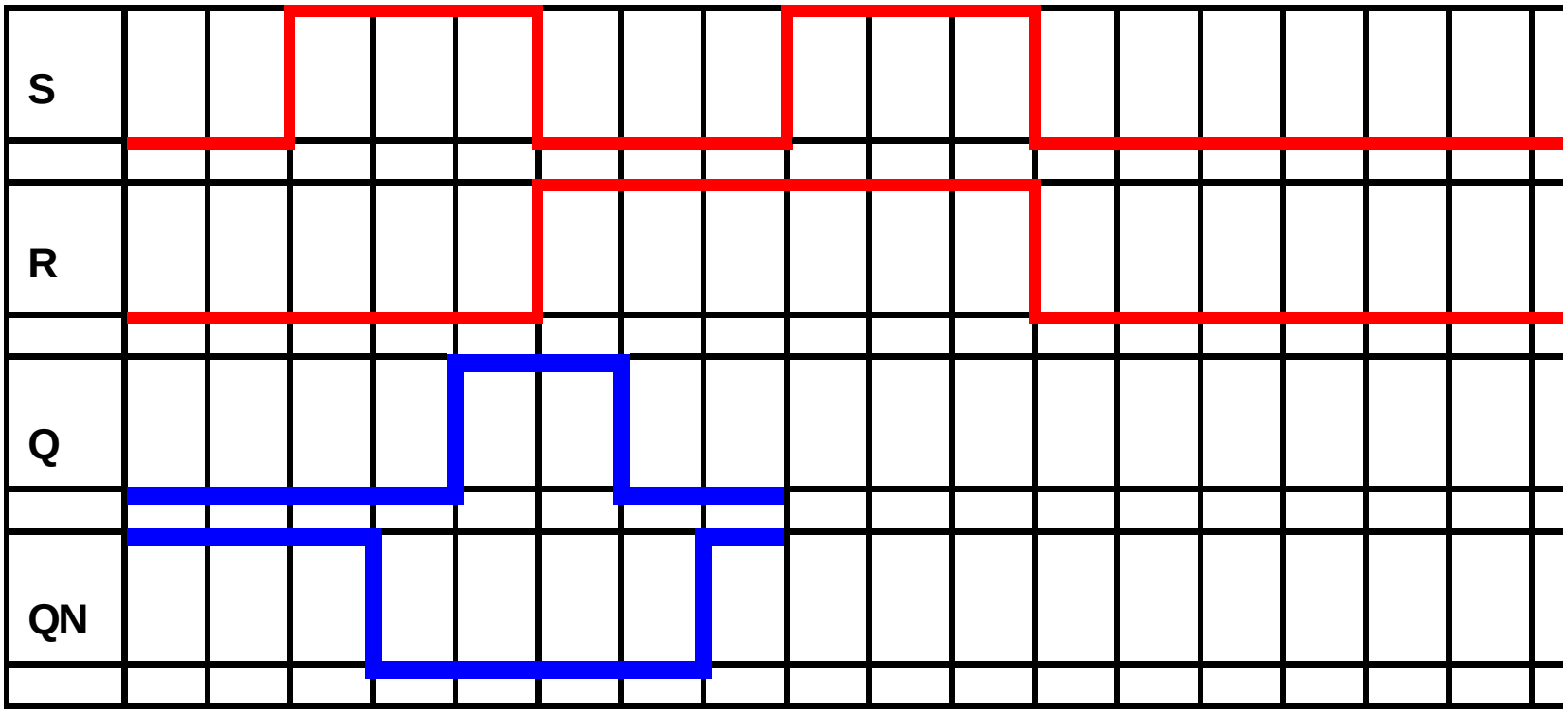


Exercise

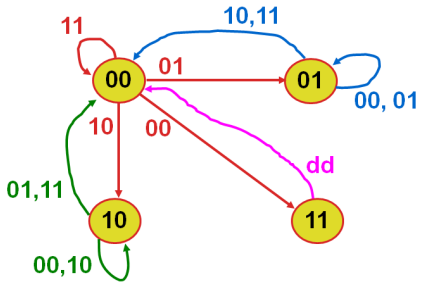


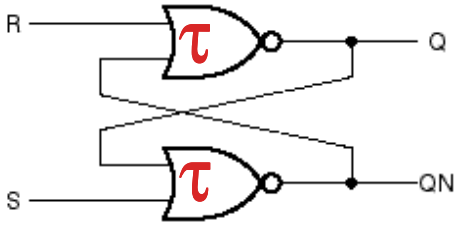


- **Investigate the response of an S-R latch to the “1-1” input combination**

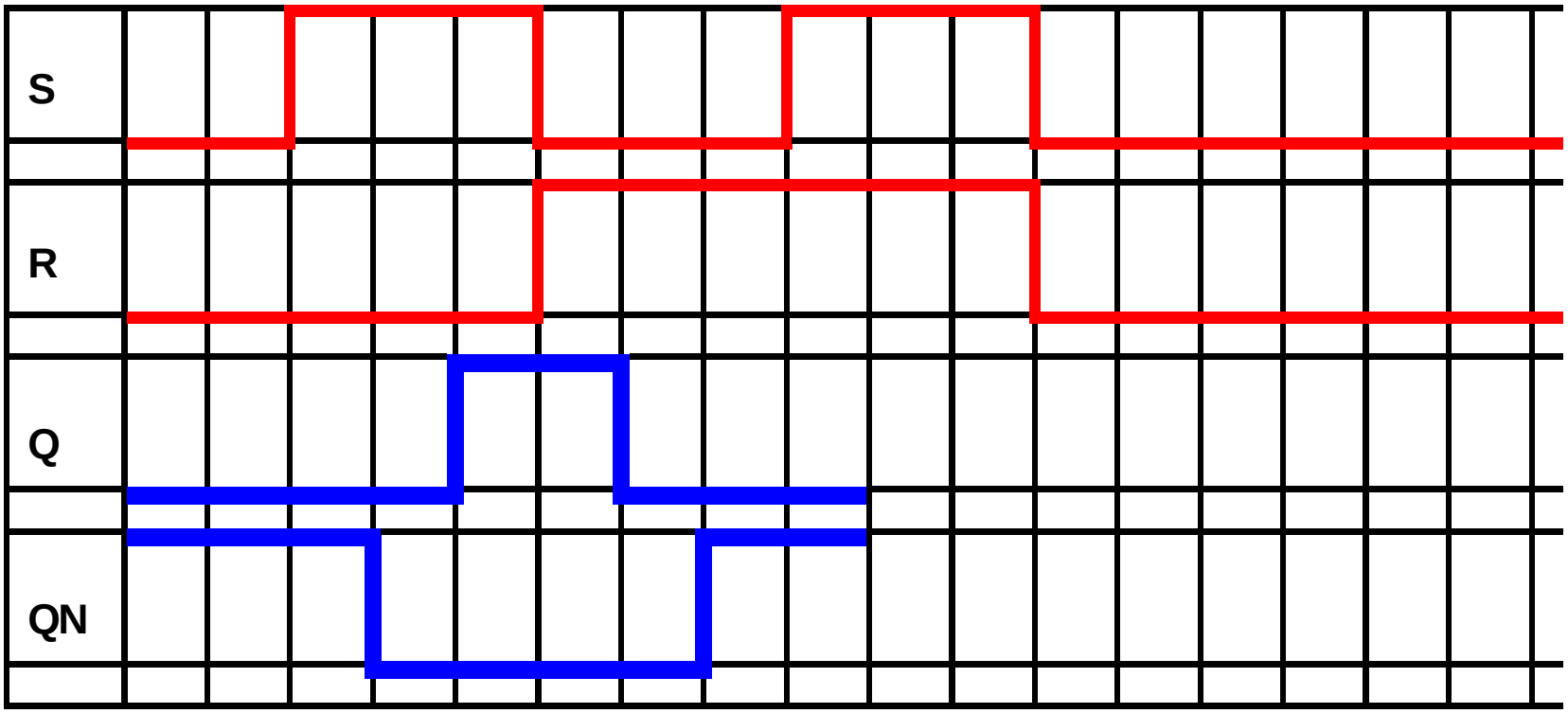


Exercise

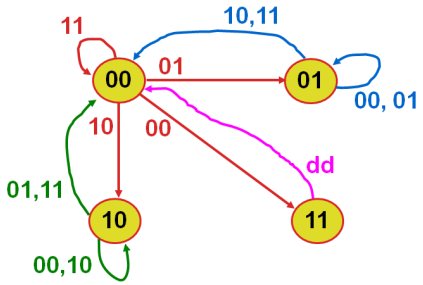


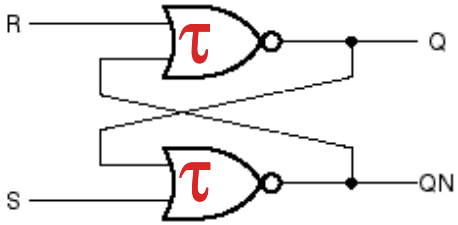


- **Investigate the response of an S-R latch to the “1-1” input combination**

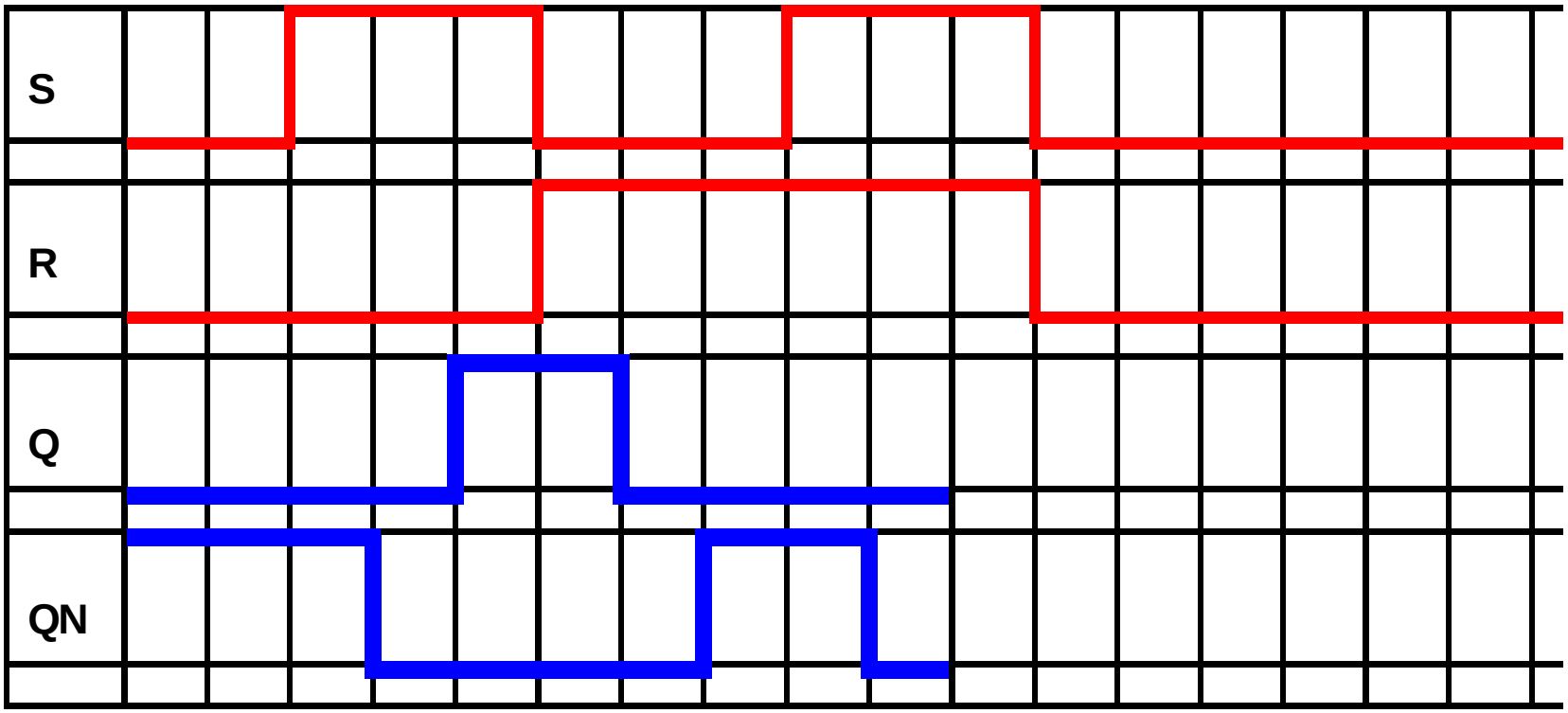


Exercise

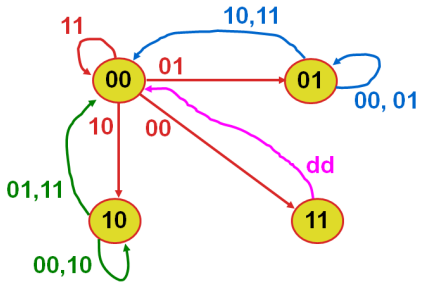


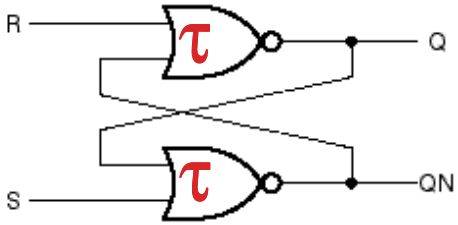


- **Investigate the response of an S-R latch to the “1-1” input combination**

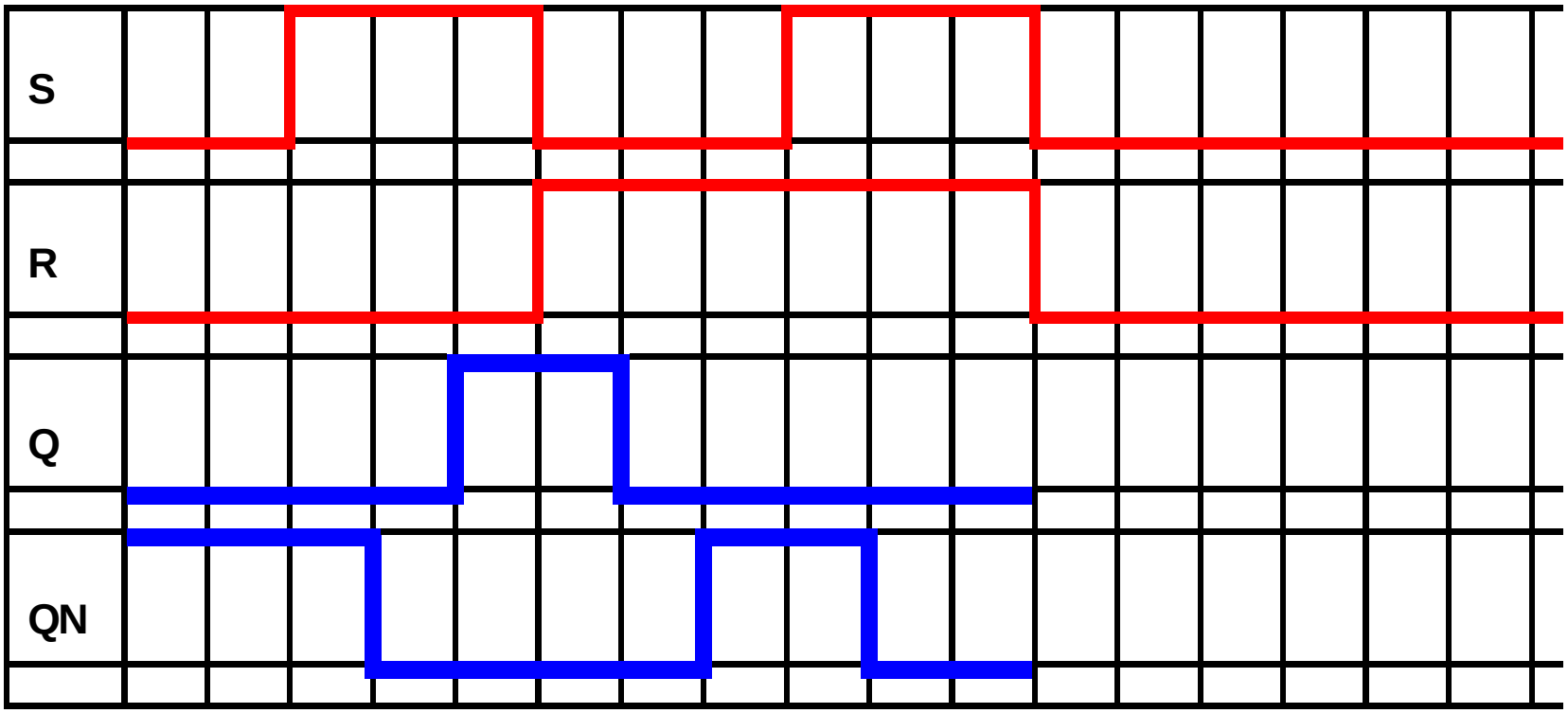


Exercise

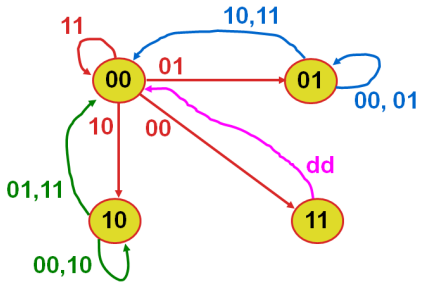


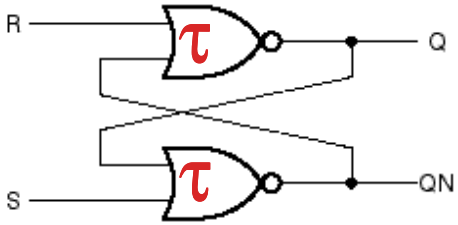


- **Investigate the response of an S-R latch to the “1-1” input combination**

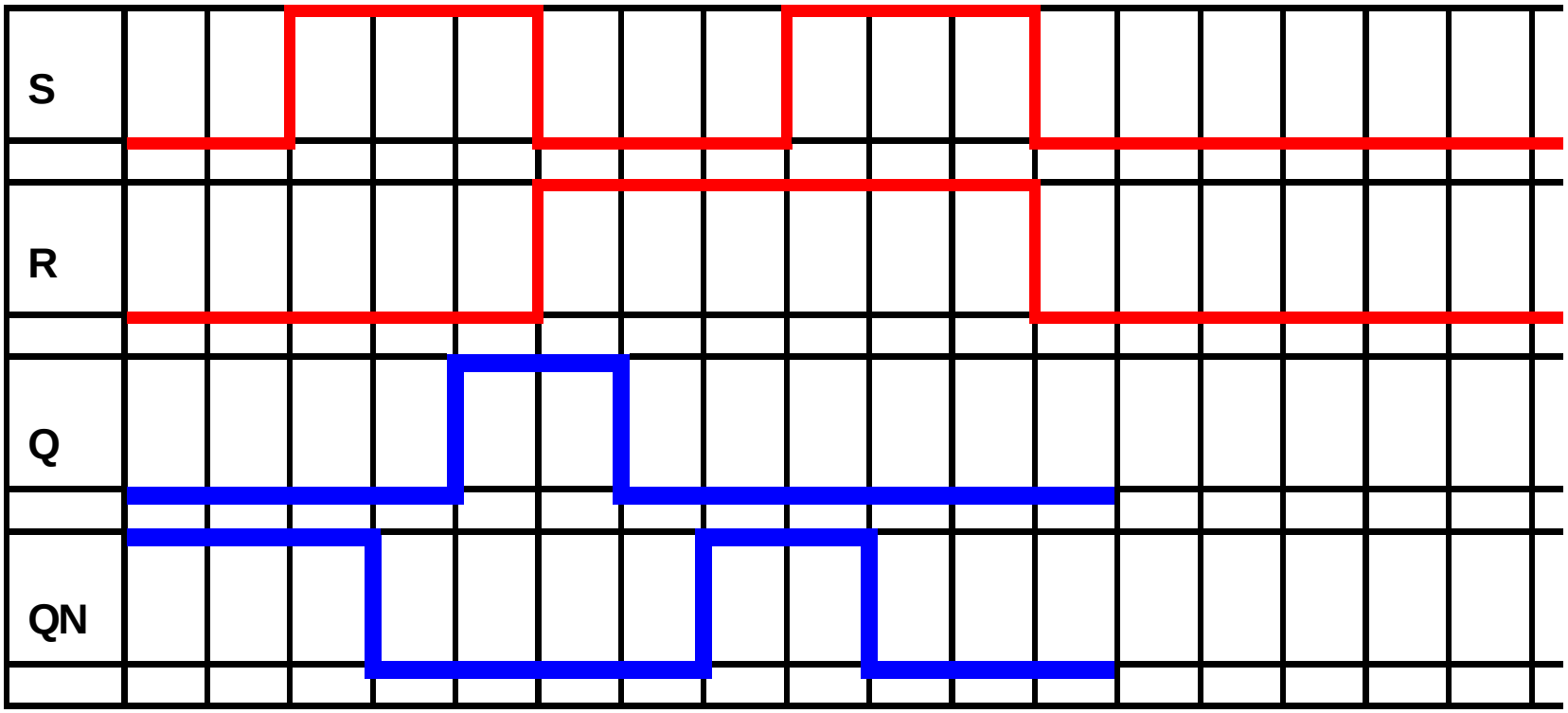


Exercise

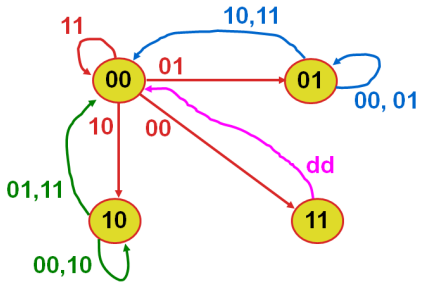


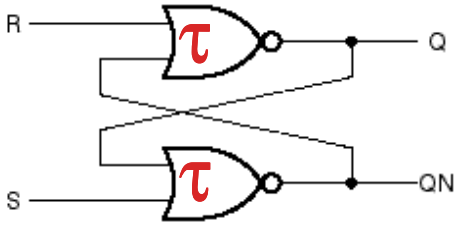


- **Investigate the response of an S-R latch to the “1-1” input combination**

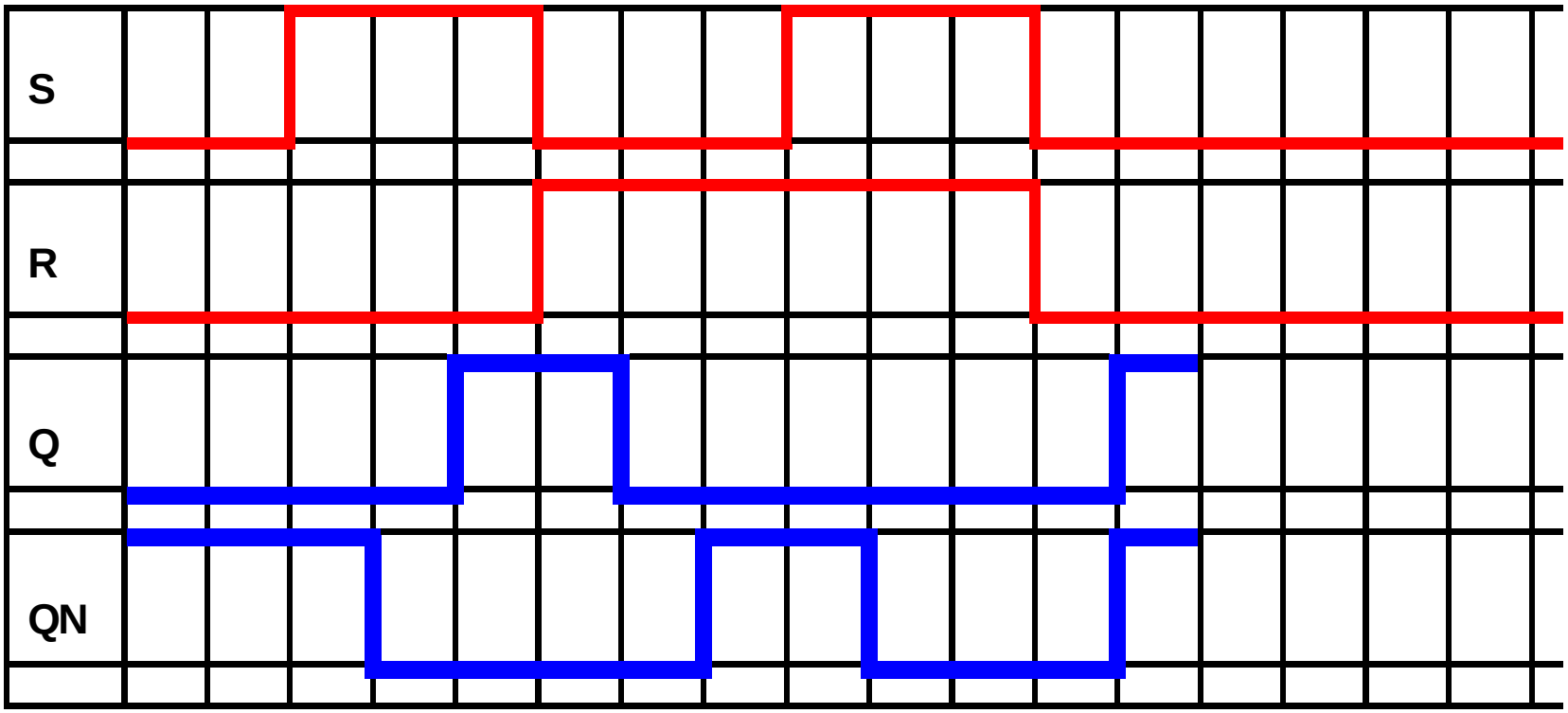


Exercise

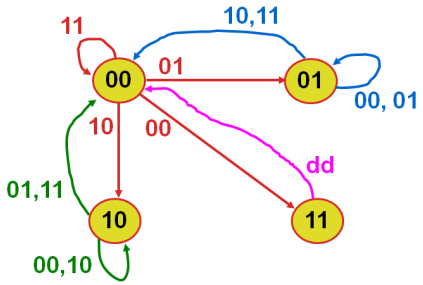


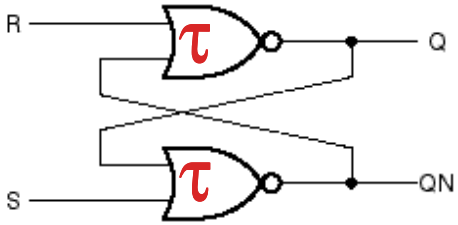


- **Investigate the response of an S-R latch to the “1-1” input combination**

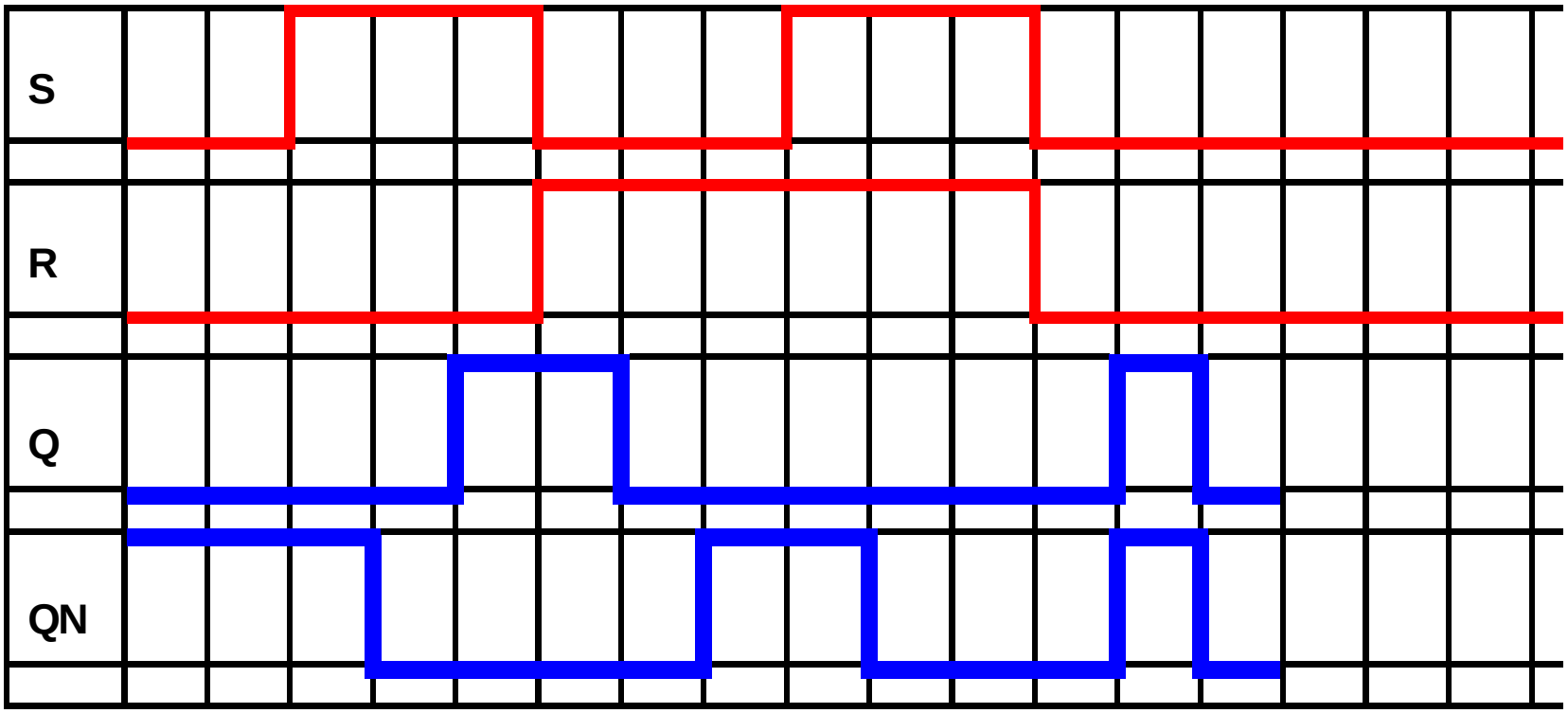


Exercise

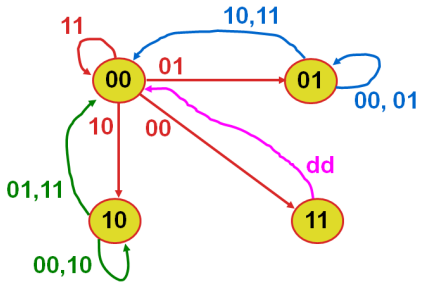


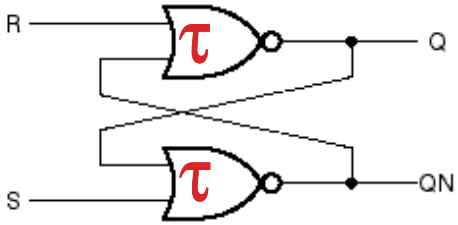


- **Investigate the response of an S-R latch to the “1-1” input combination**

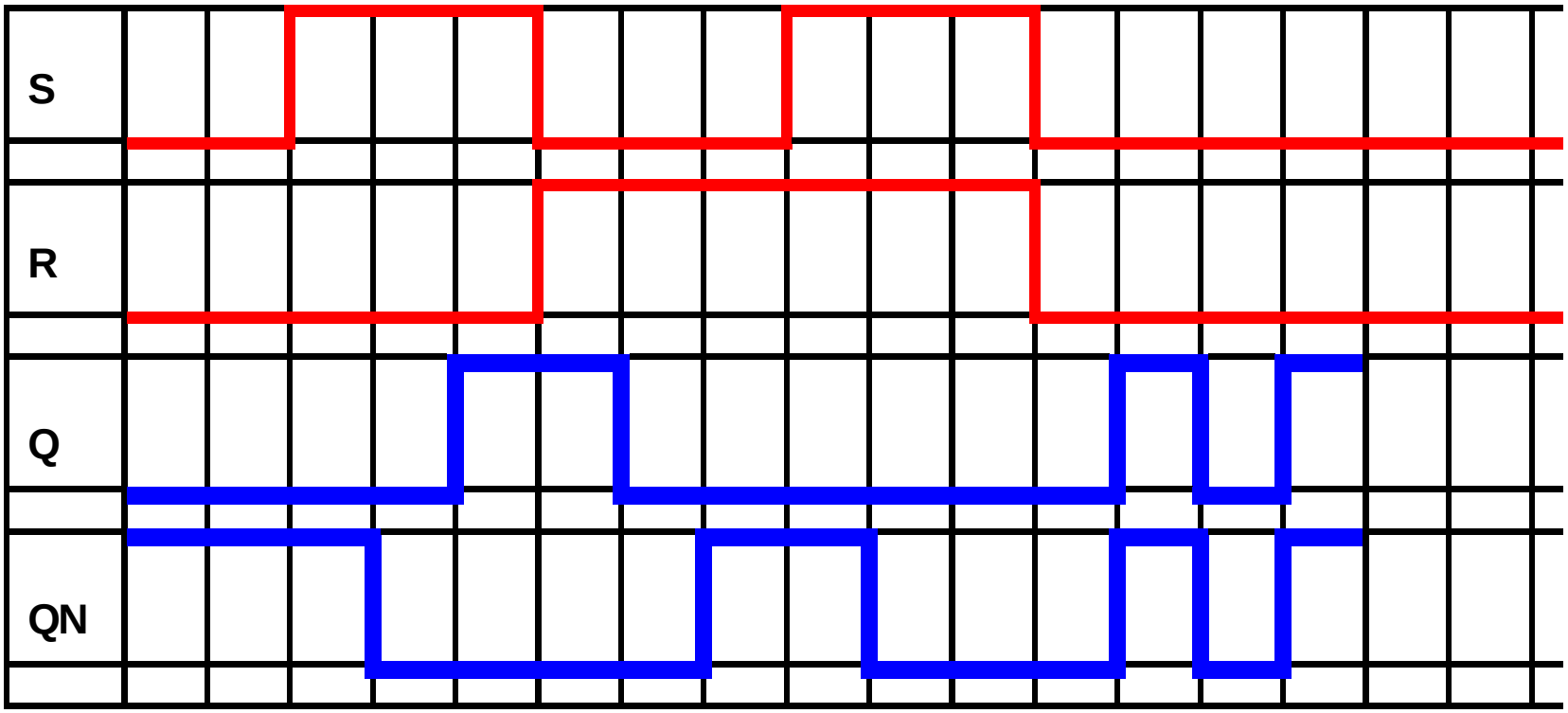


Exercise

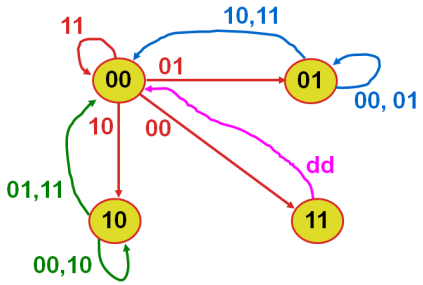


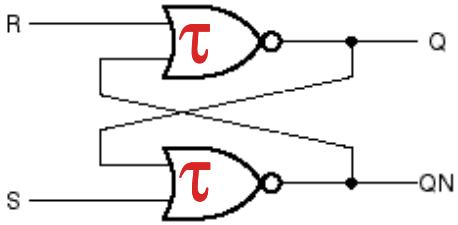


- **Investigate the response of an S-R latch to the “1-1” input combination**

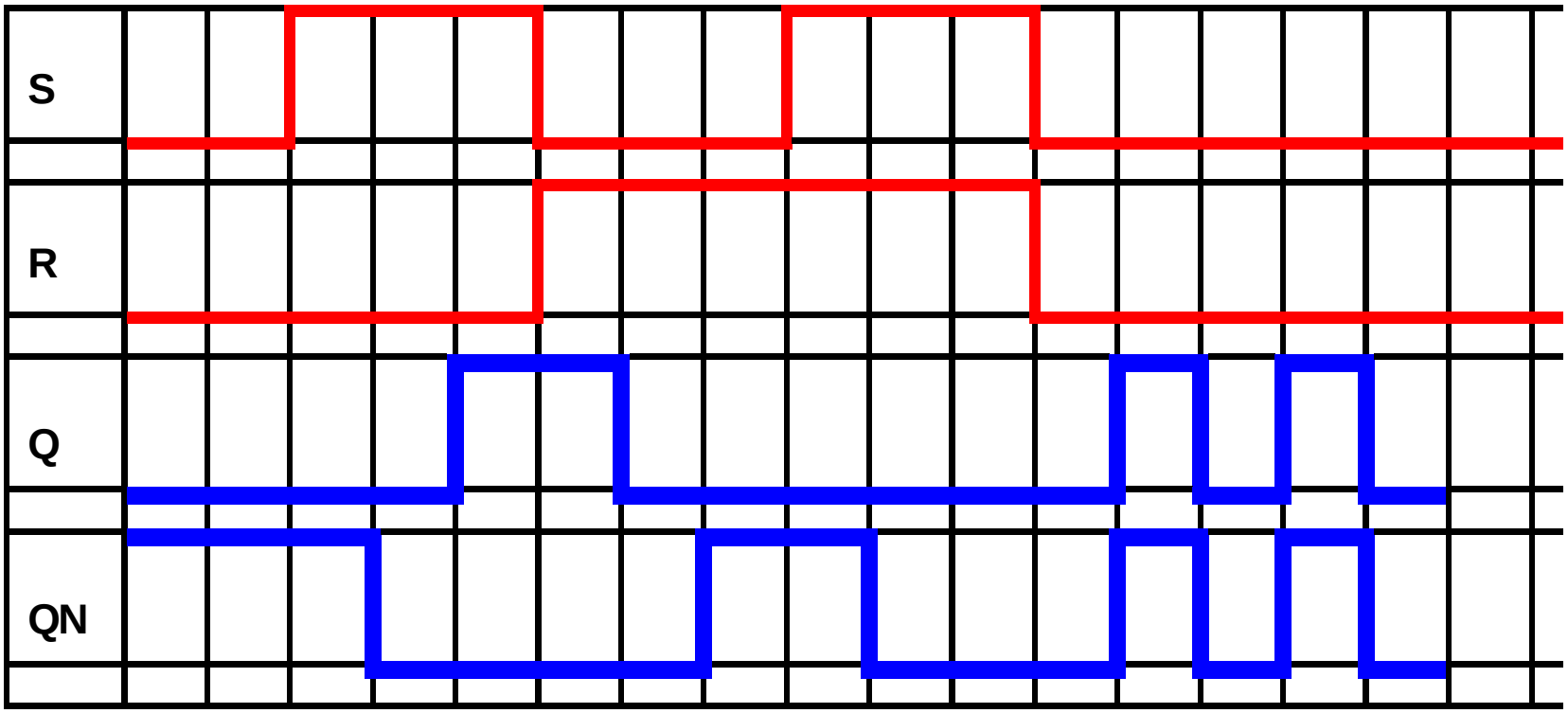


Exercise

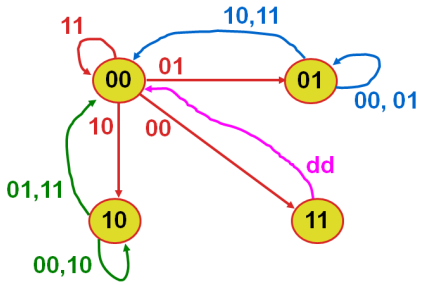


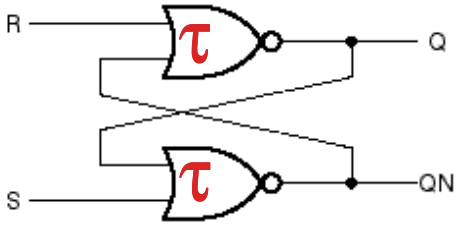


- **Investigate the response of an S-R latch to the “1-1” input combination**

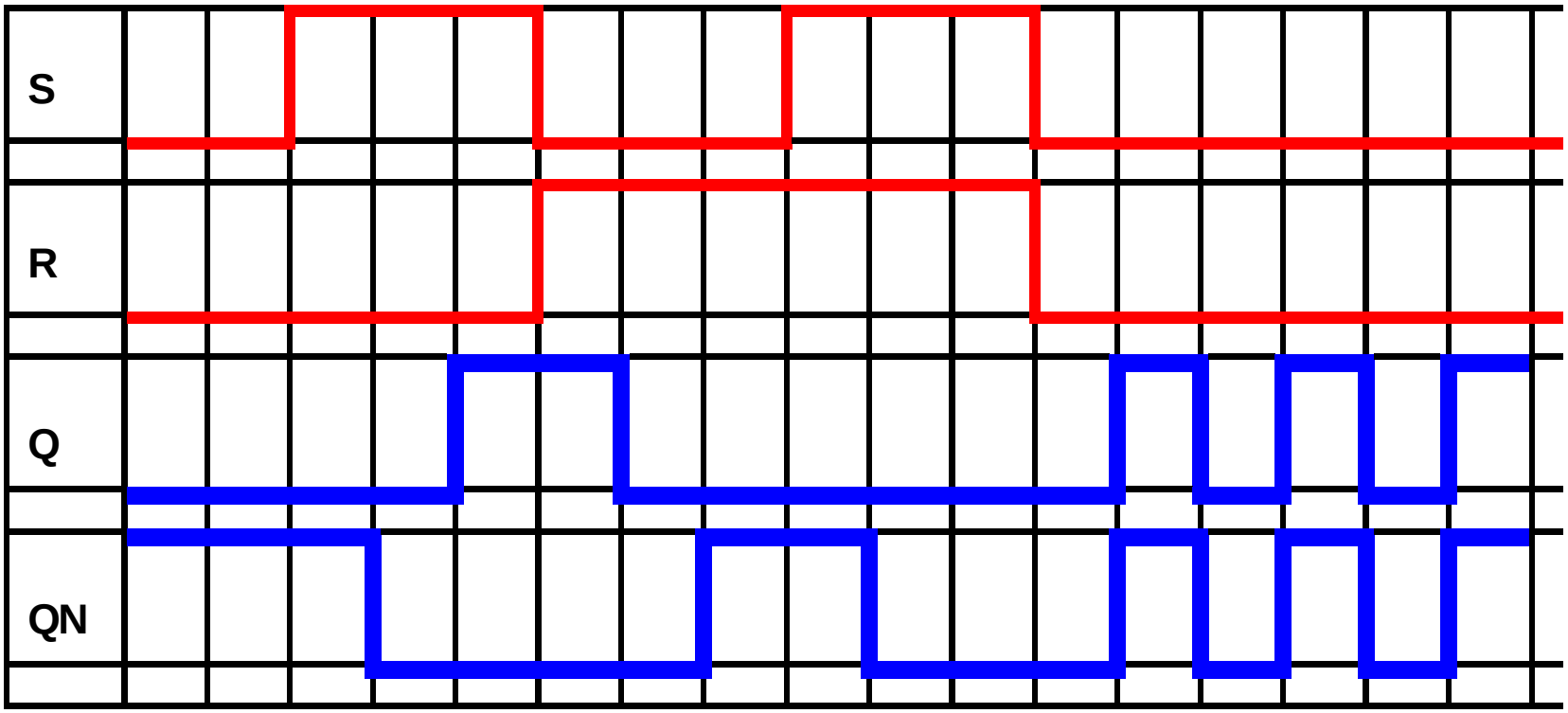


Exercise





- **Investigate the response of an S-R latch to the “1-1” input combination**



Clicker Quiz

Q1. For the NOR-implemented SR latch, the following output combination **cannot occur at any time:**

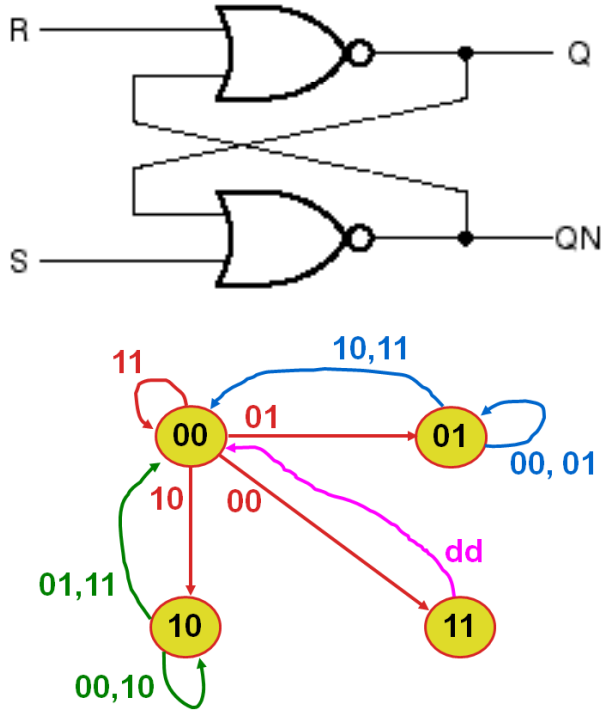
A. Q=0, QN=0

B. Q=0, QN=1

C. Q=1, QN=0

D. Q=1, QN=1

E. none of the above



Q2. If the **input** combination **S=0, R=1** is applied to this circuit, the (steady state) output will be:

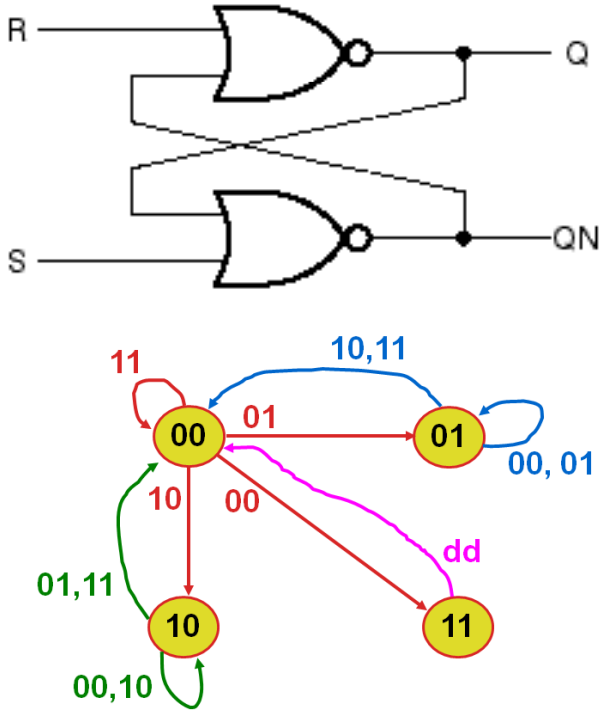
A. Q=0, QN=0

B. Q=0, QN=1

C. Q=1, QN=0

D. Q=1, QN=1

E. none of the above



Q3. If the **input** combination **S=1, R=0** is applied to this circuit, the (steady state) output will be:

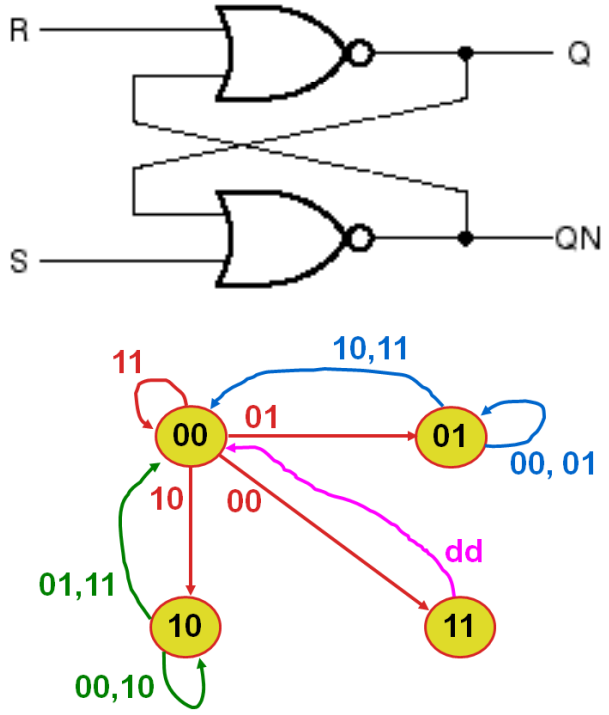
A. Q=0, QN=0

B. Q=0, QN=1

C. Q=1, QN=0

D. Q=1, QN=1

E. none of the above



Q4. If the **input** combination **S=1, R=1** is applied to this circuit, the (steady state) output will be:

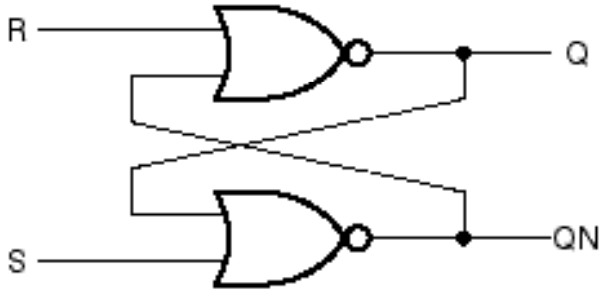
A. Q=0, QN=0

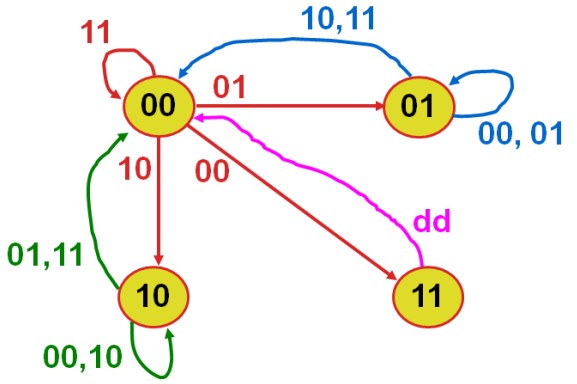
B. Q=0, QN=1

C. Q=1, QN=0

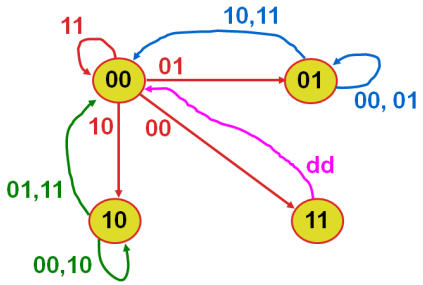
D. Q=1, QN=1

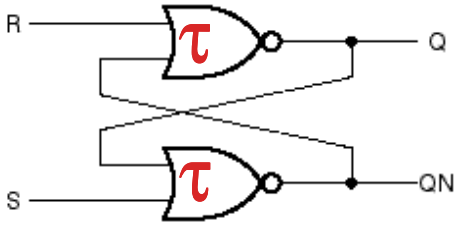
E. none of the above



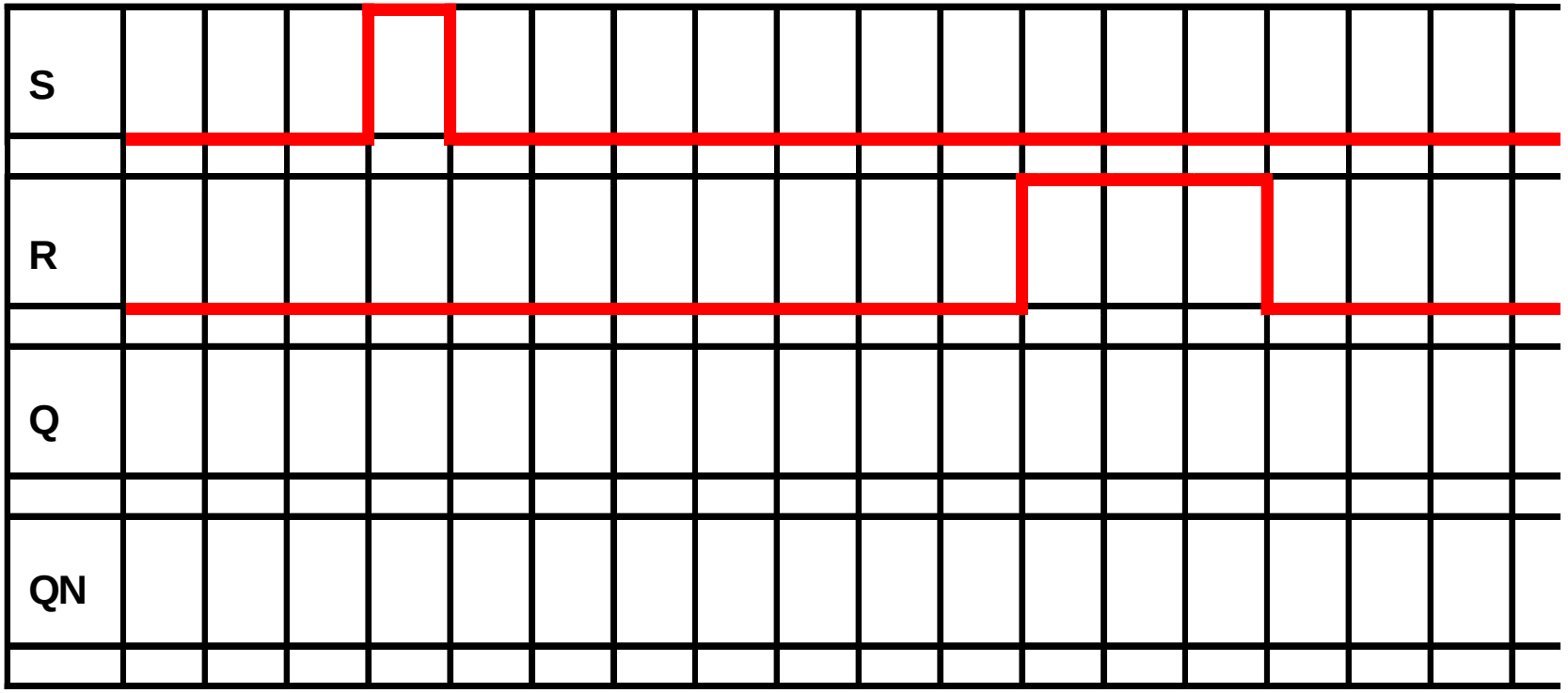


Exercise

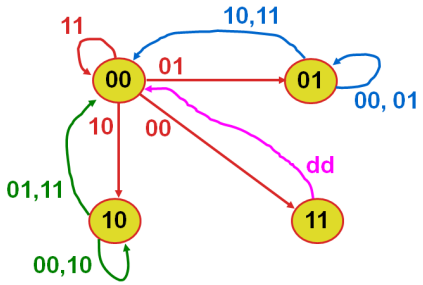


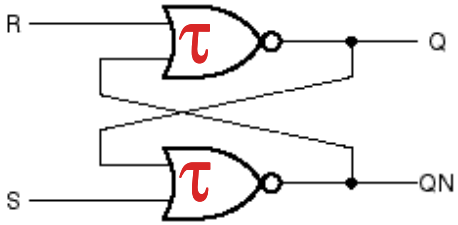


- **Investigate the response of an S-R latch to a glitch or hazard**

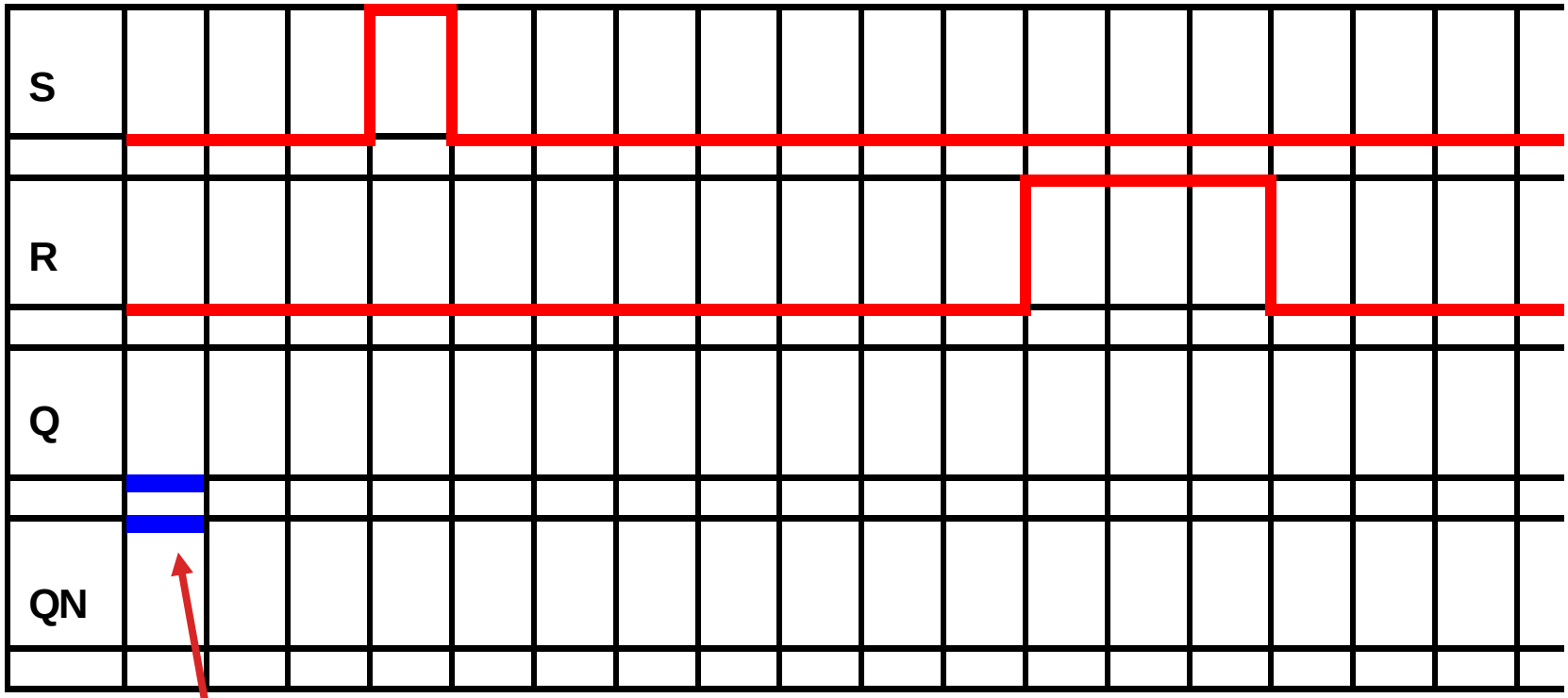


Exercise

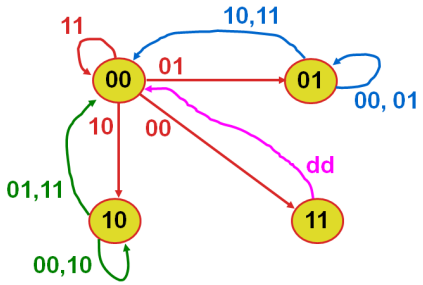


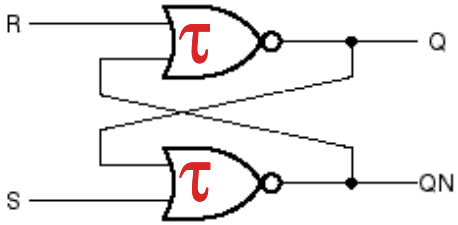


- **Investigate the response of an S-R latch to a glitch or hazard**

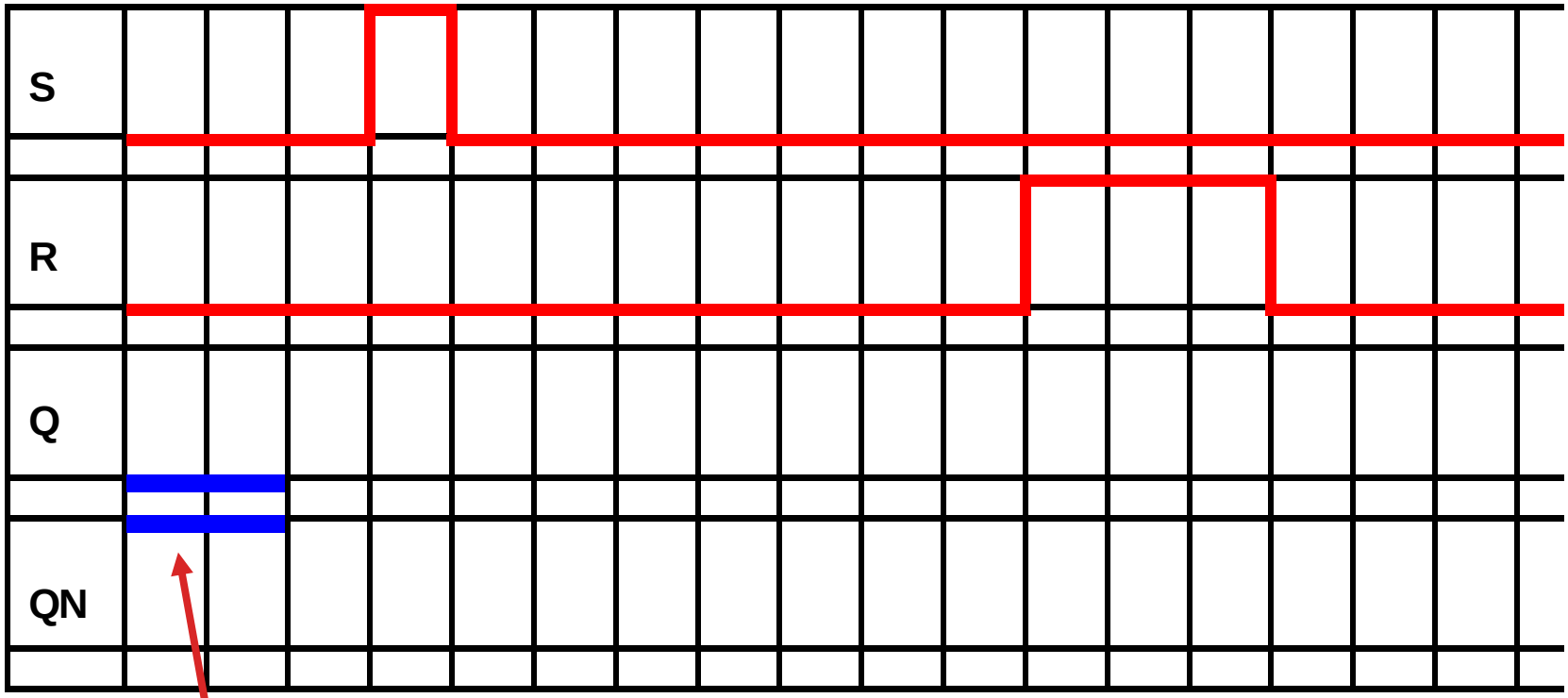


Exercise

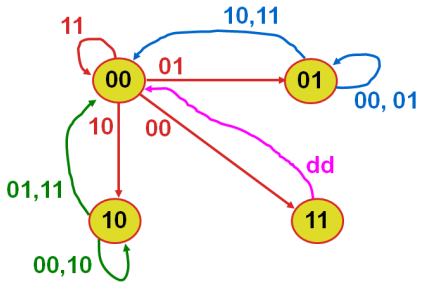


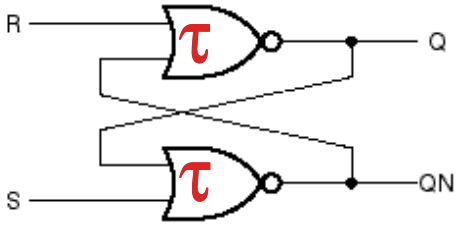


- **Investigate the response of an S-R latch to a glitch or hazard**

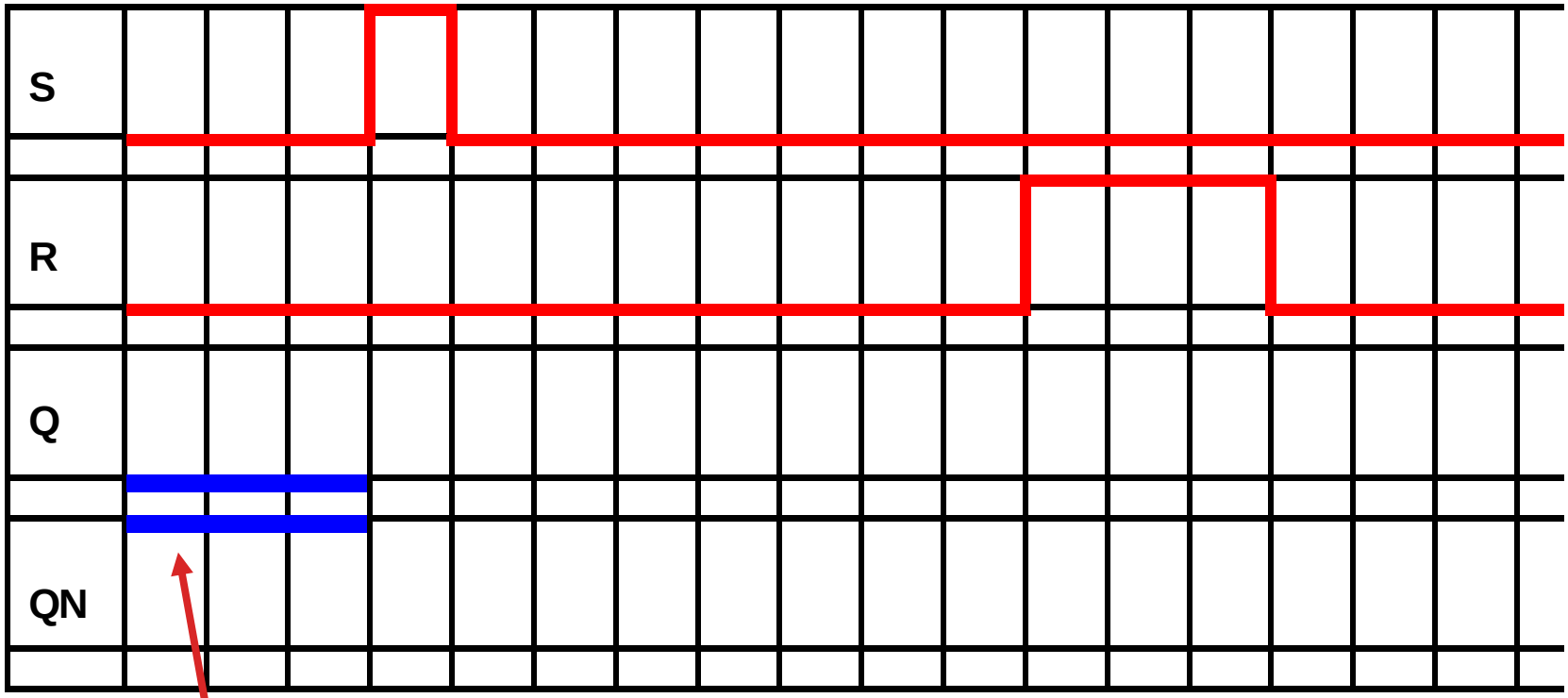


Exercise

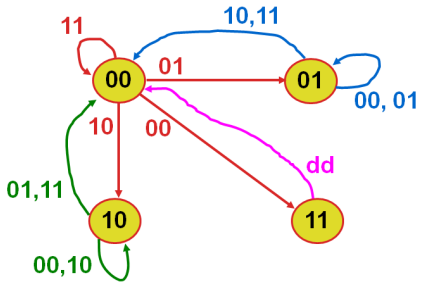


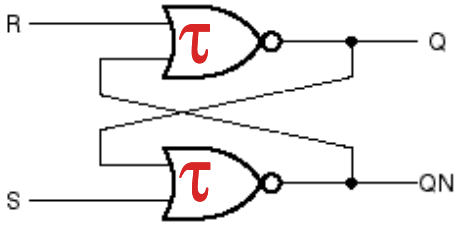


- **Investigate the response of an S-R latch to a glitch or hazard**

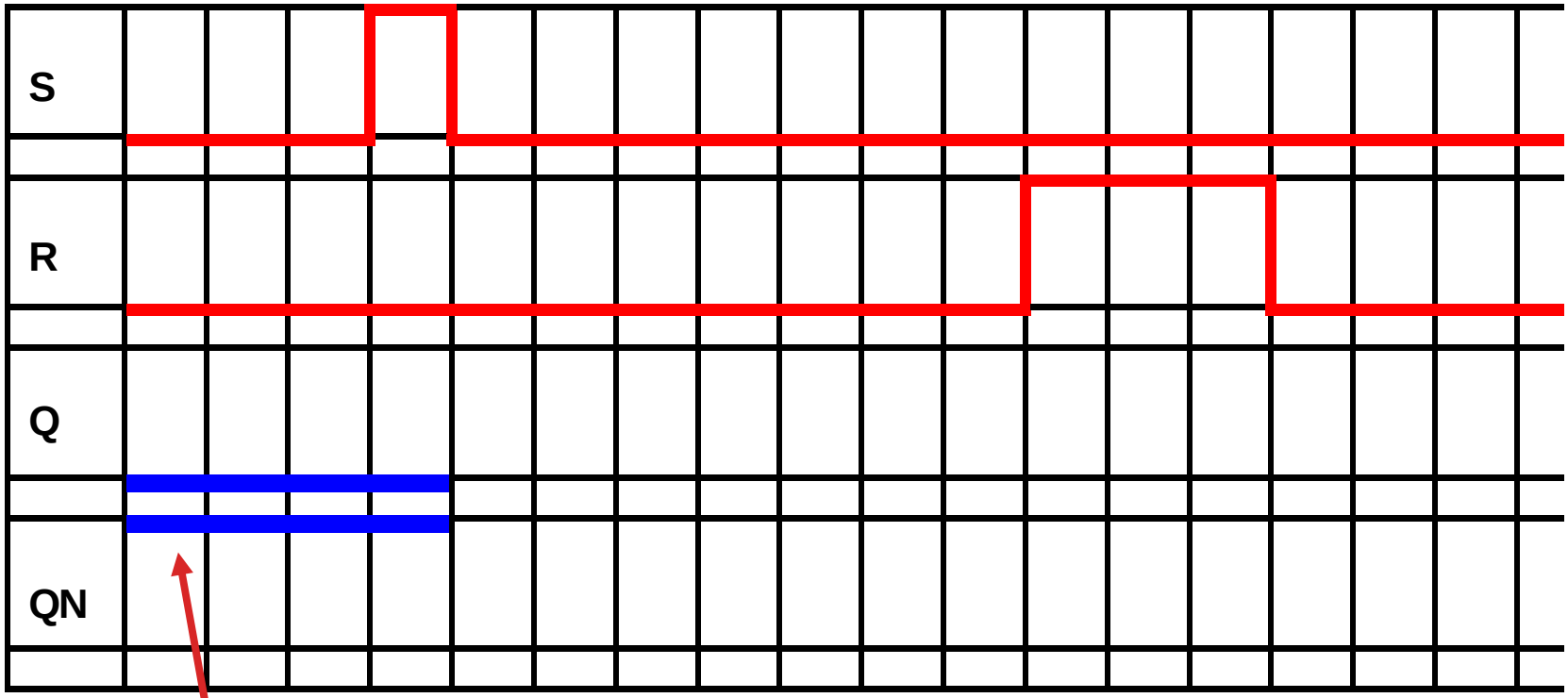


Exercise

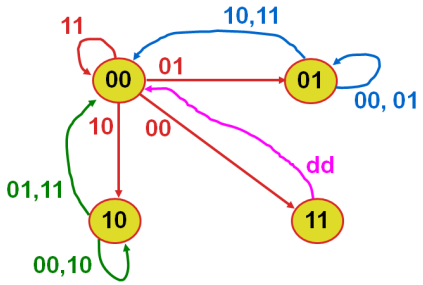


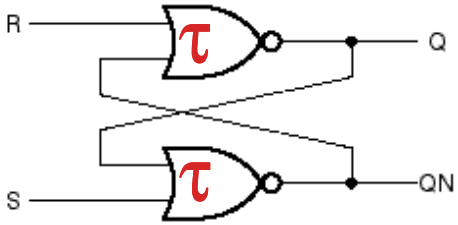


- **Investigate the response of an S-R latch to a glitch or hazard**

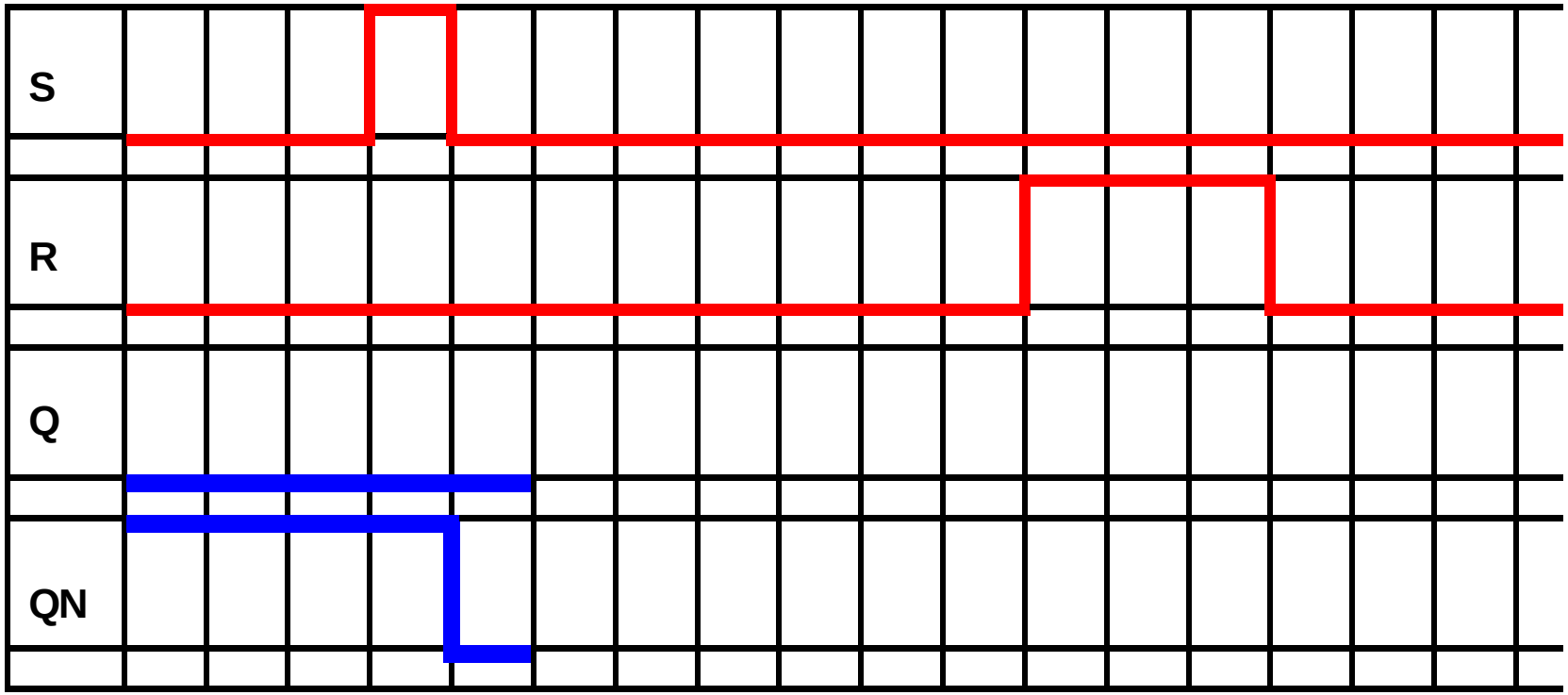


Exercise

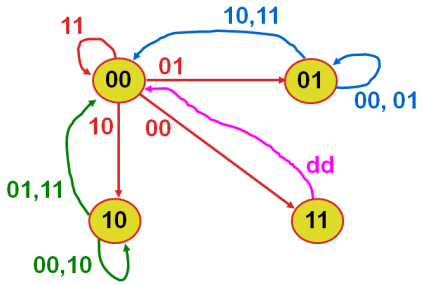


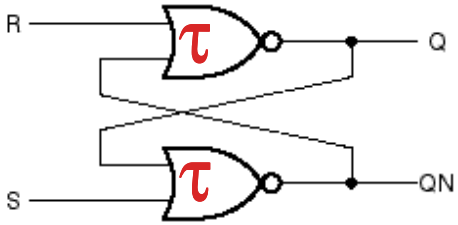


- **Investigate the response of an S-R latch to a glitch or hazard**

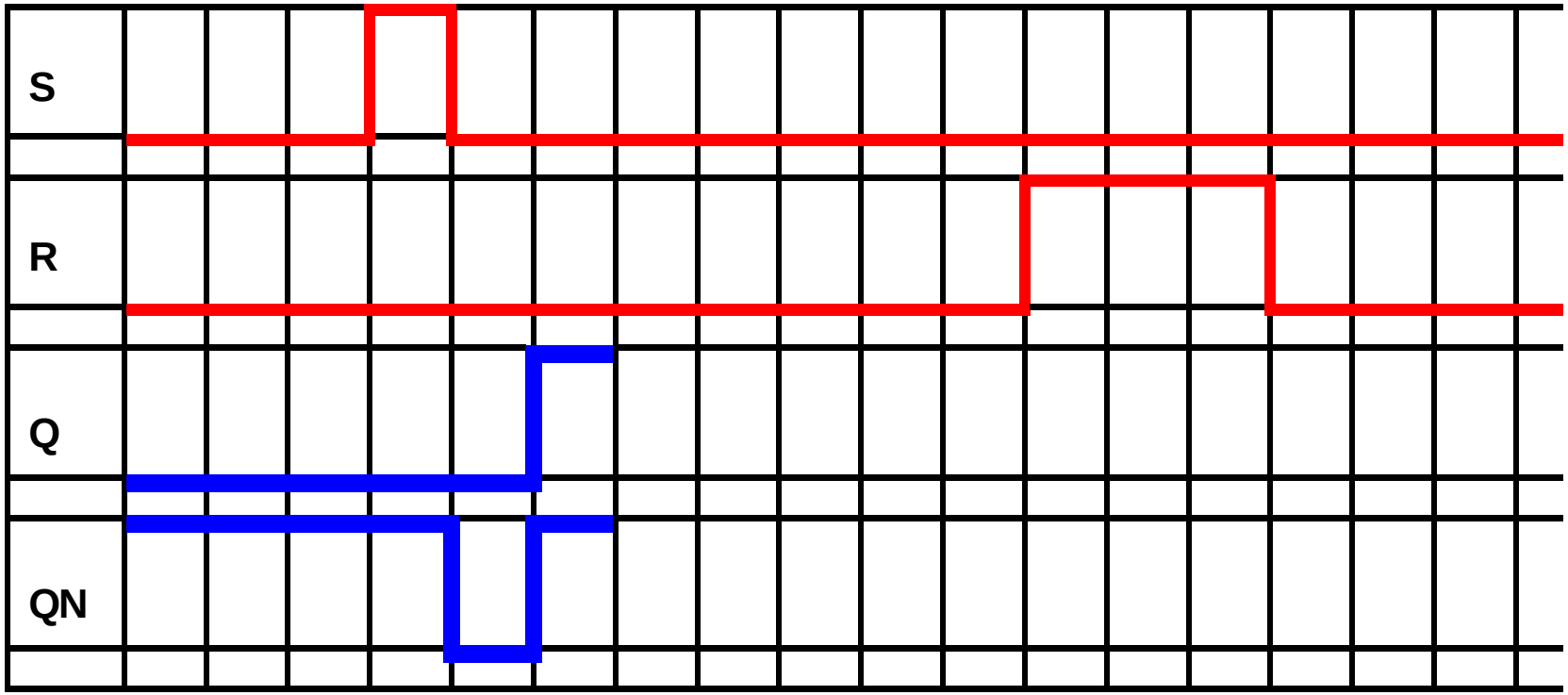


Exercise

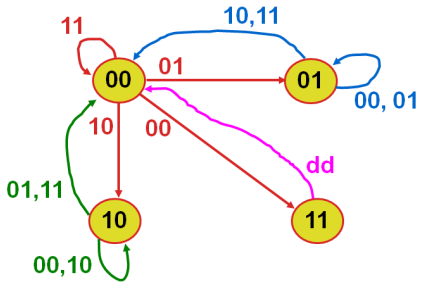


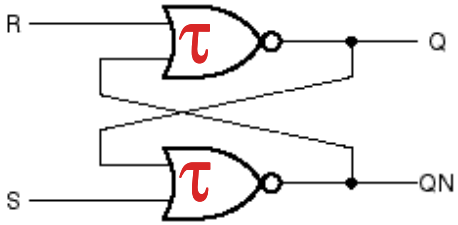


- **Investigate the response of an S-R latch to a glitch or hazard**

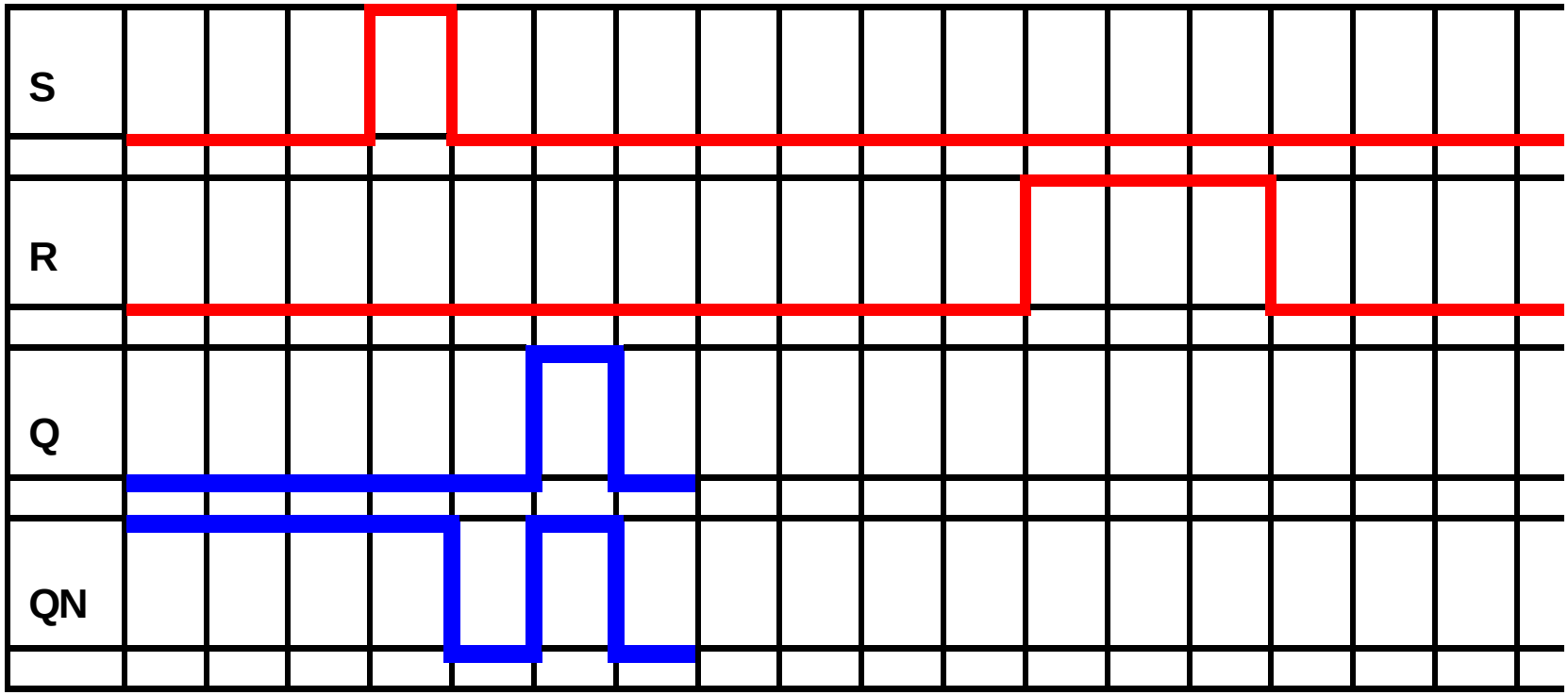


Exercise

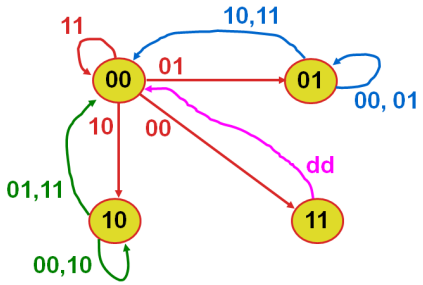


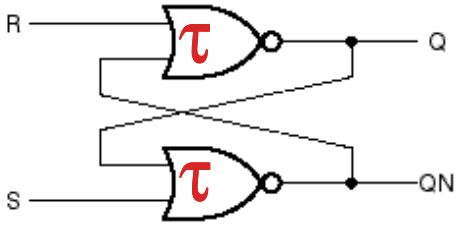


- **Investigate the response of an S-R latch to a glitch or hazard**

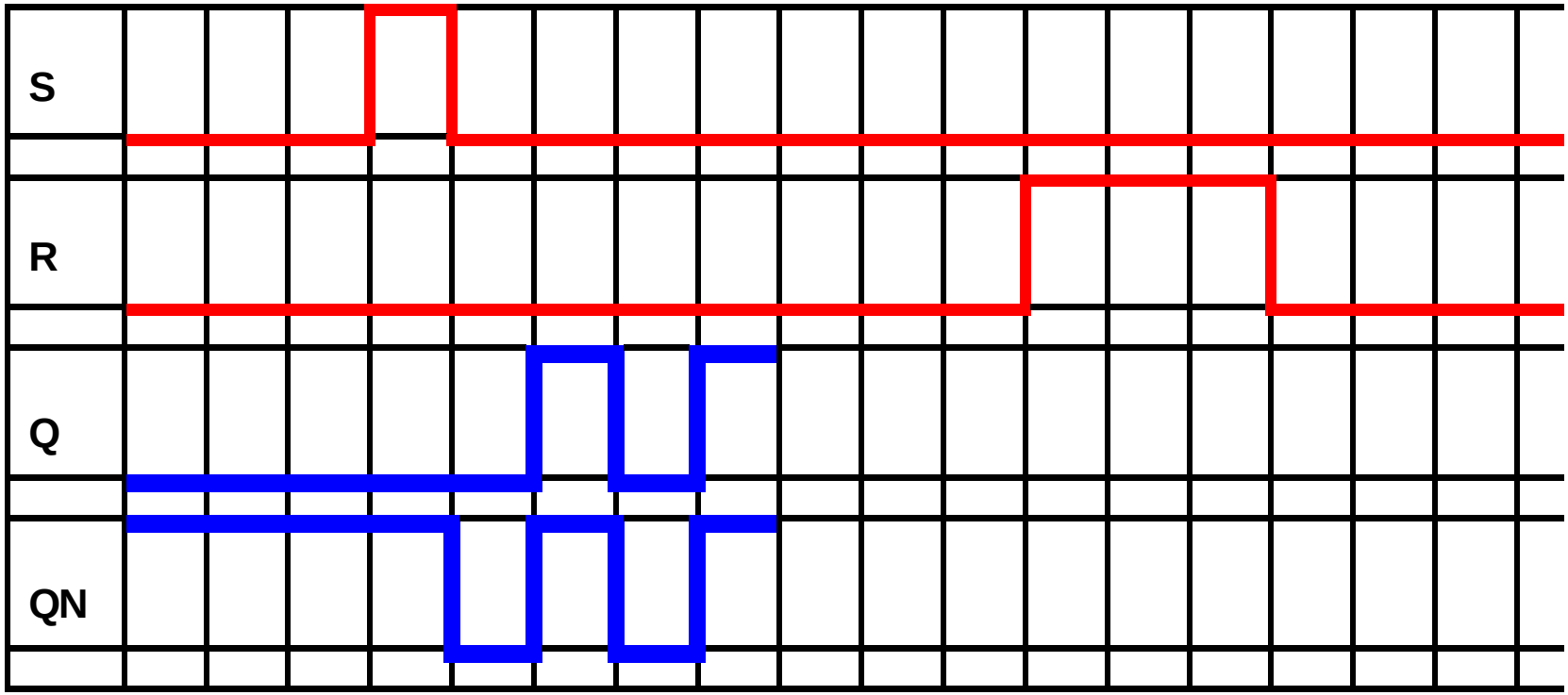


Exercise

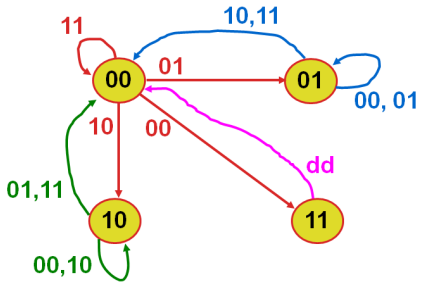


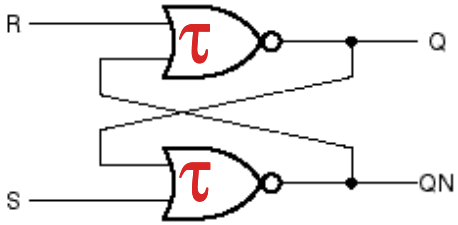


- **Investigate the response of an S-R latch to a glitch or hazard**

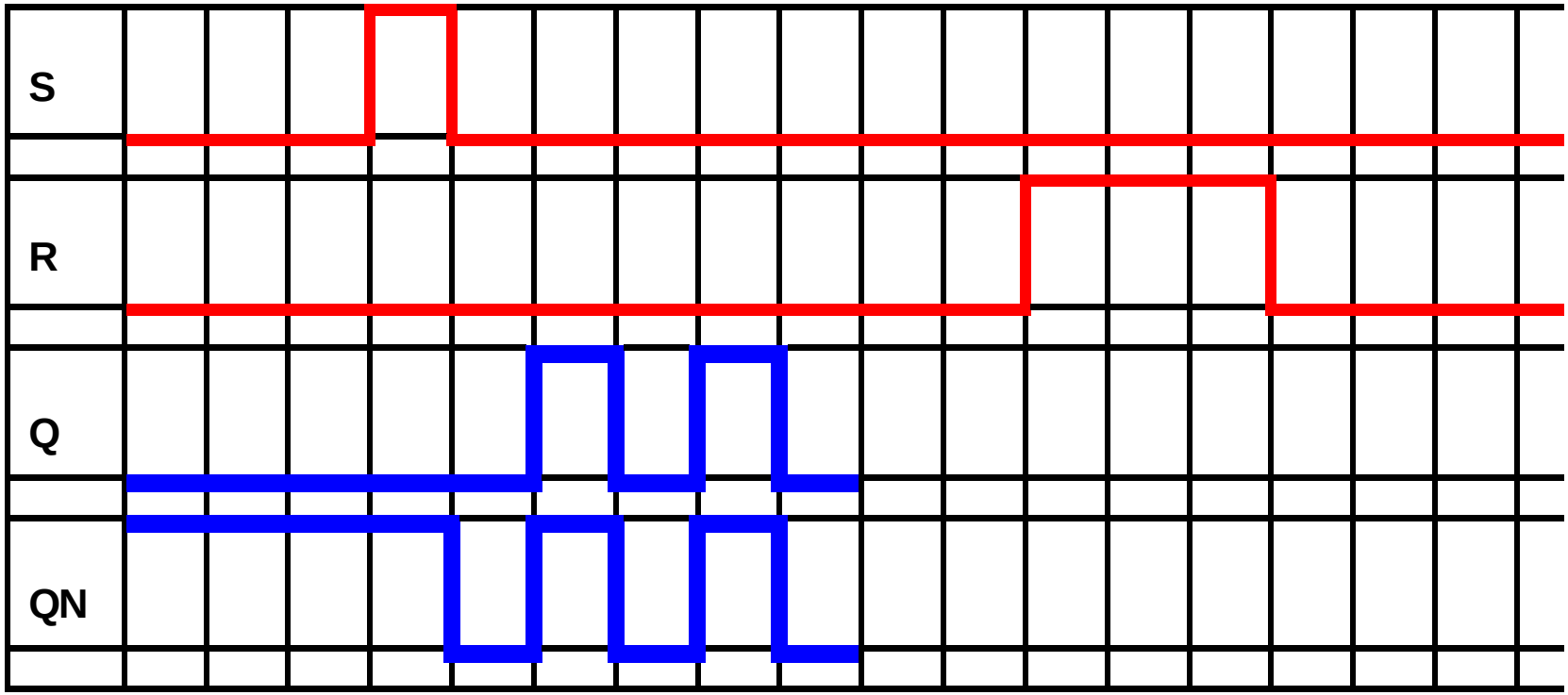


Exercise

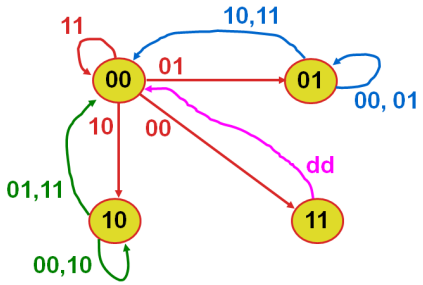


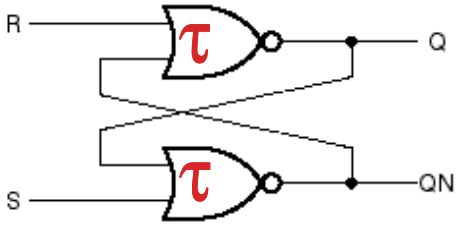


- **Investigate the response of an S-R latch to a glitch or hazard**

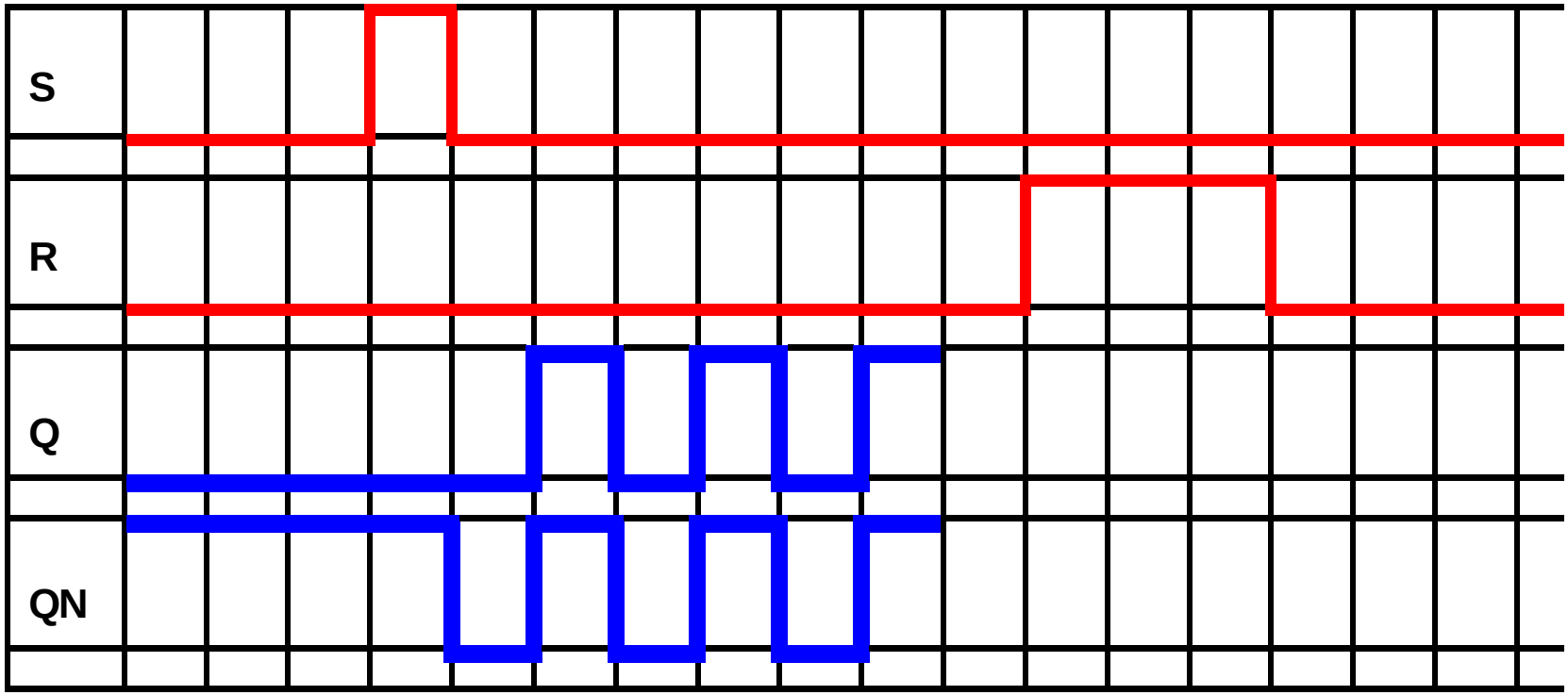


Exercise

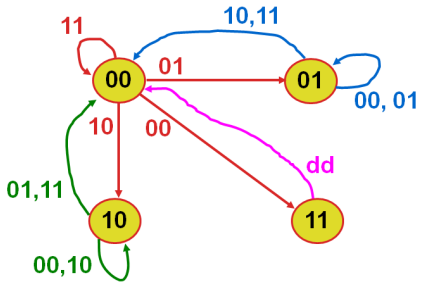


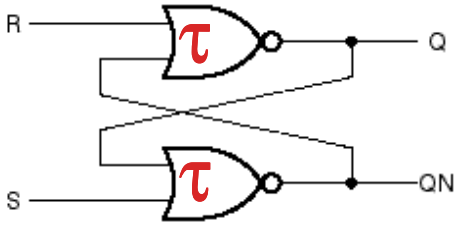


- **Investigate the response of an S-R latch to a glitch or hazard**

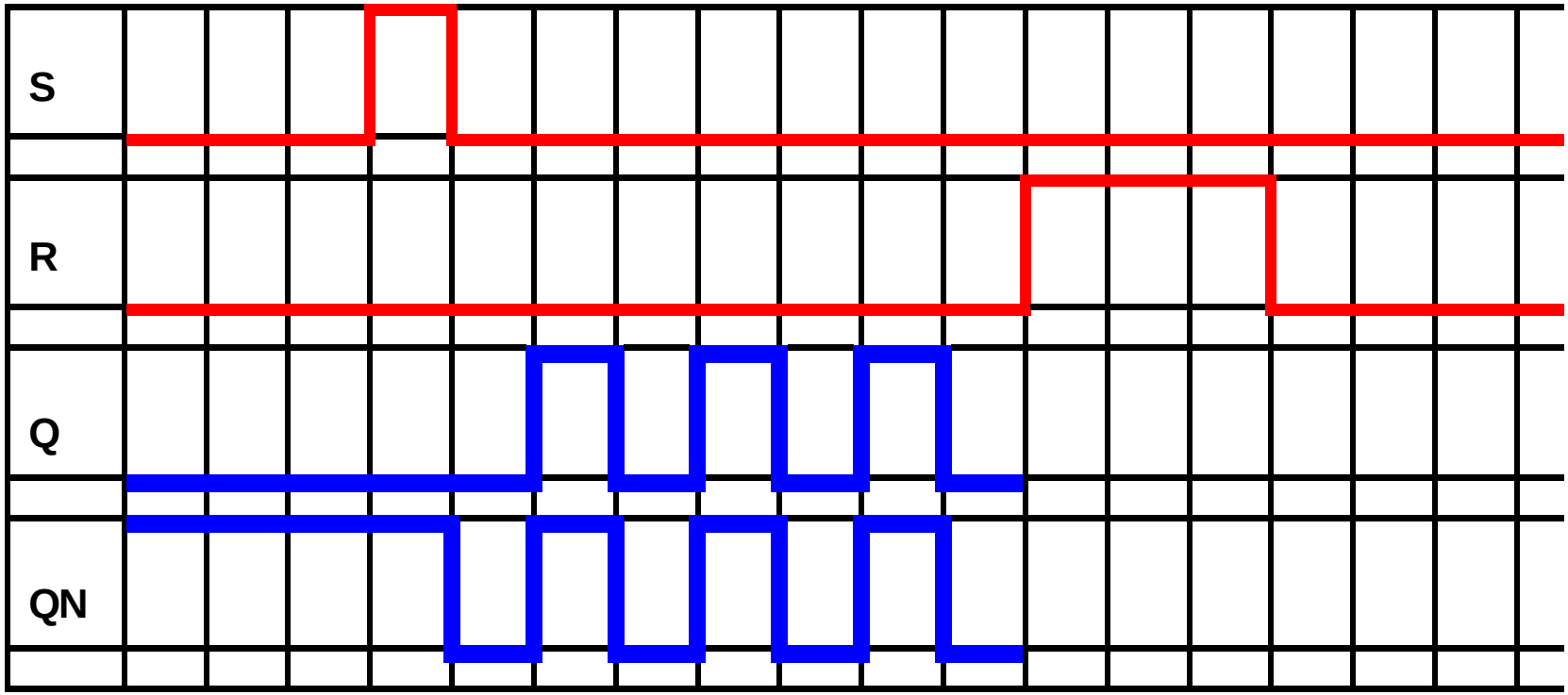


Exercise

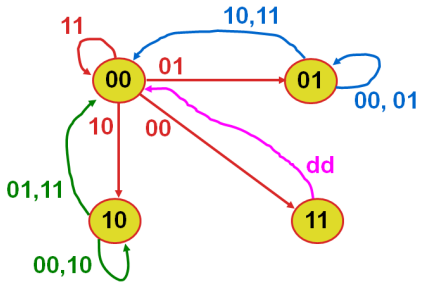


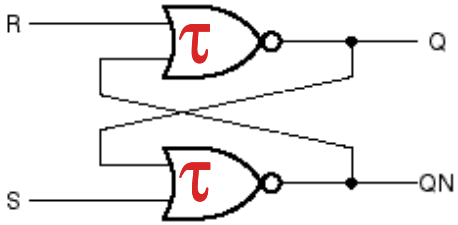


- **Investigate the response of an S-R latch to a glitch or hazard**

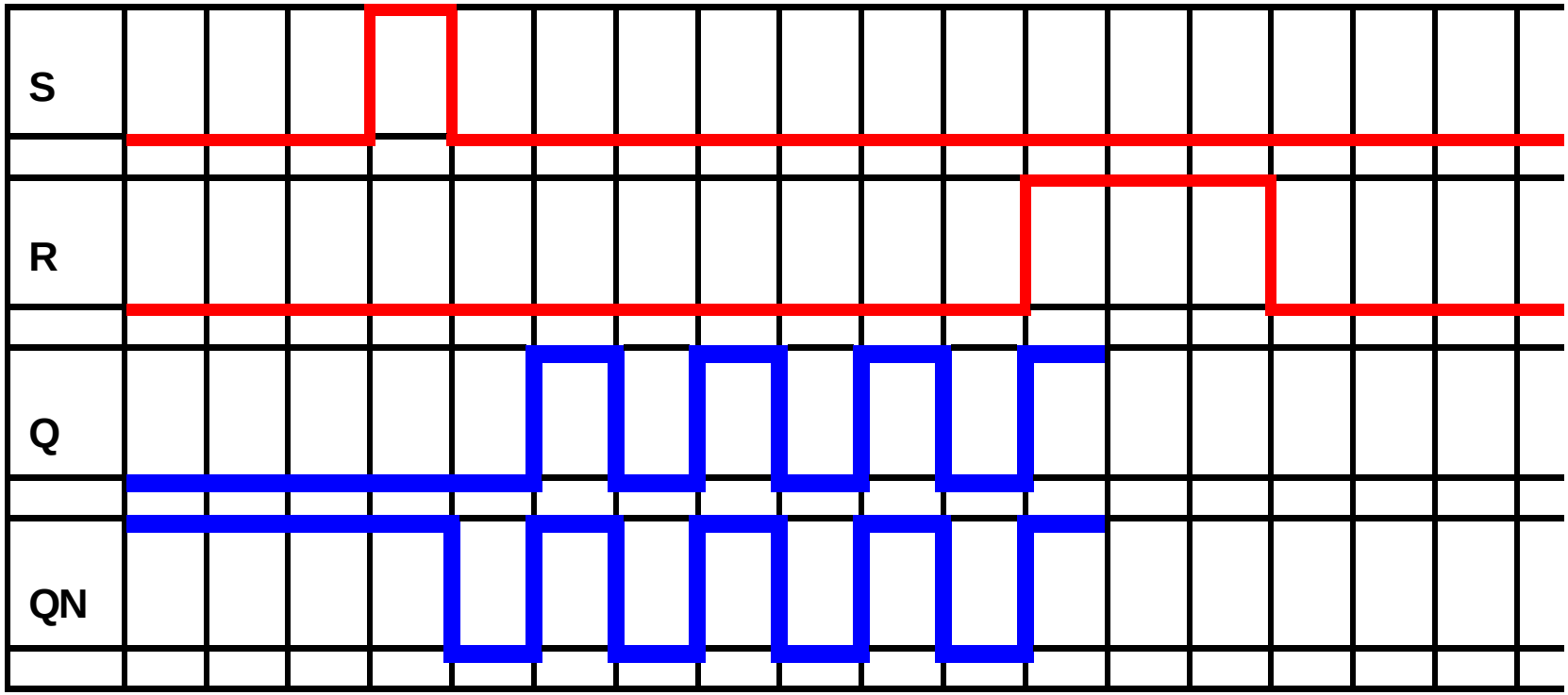


Exercise

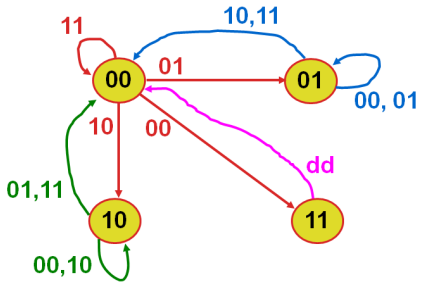


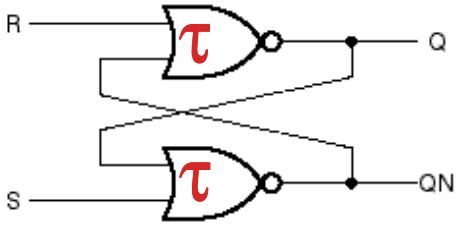


- **Investigate the response of an S-R latch to a glitch or hazard**

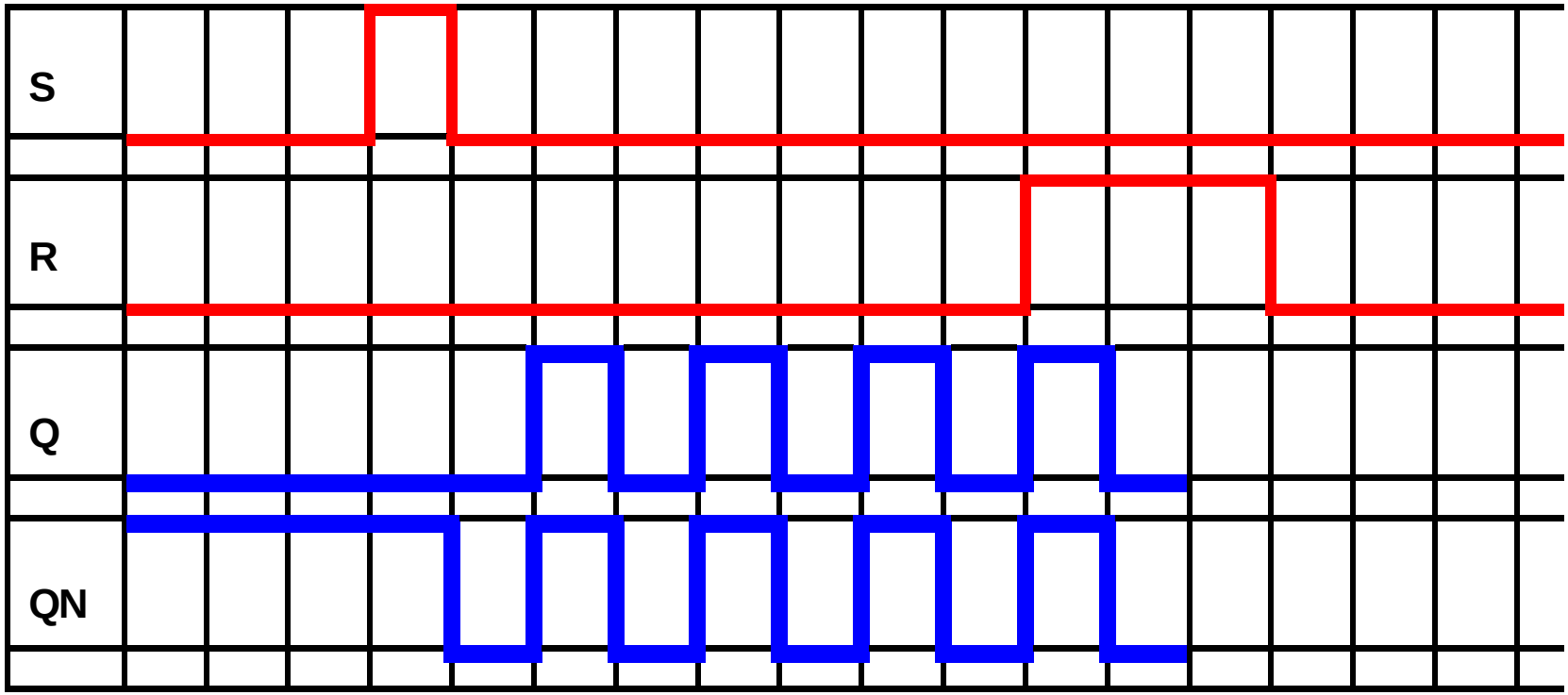


Exercise

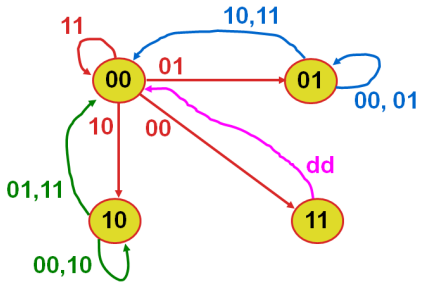


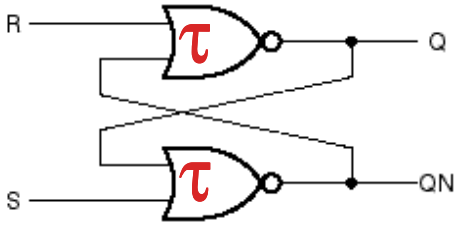


- **Investigate the response of an S-R latch to a glitch or hazard**

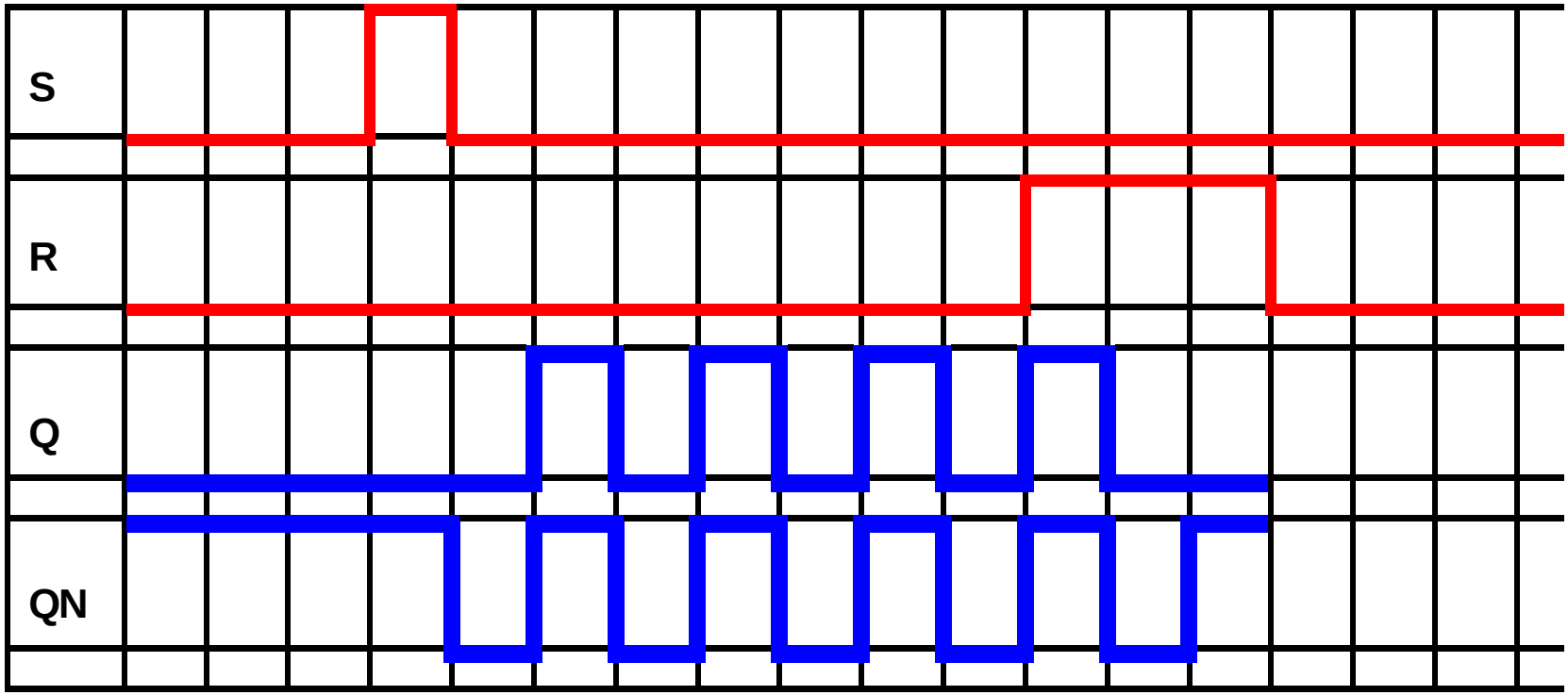


Exercise

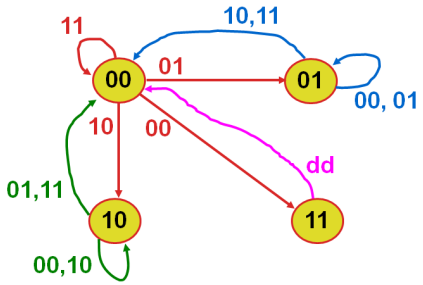


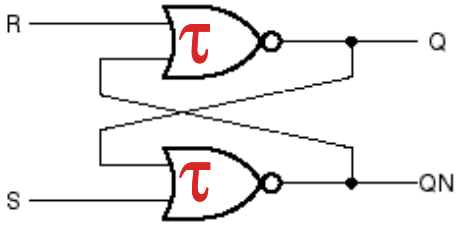


- **Investigate the response of an S-R latch to a glitch or hazard**

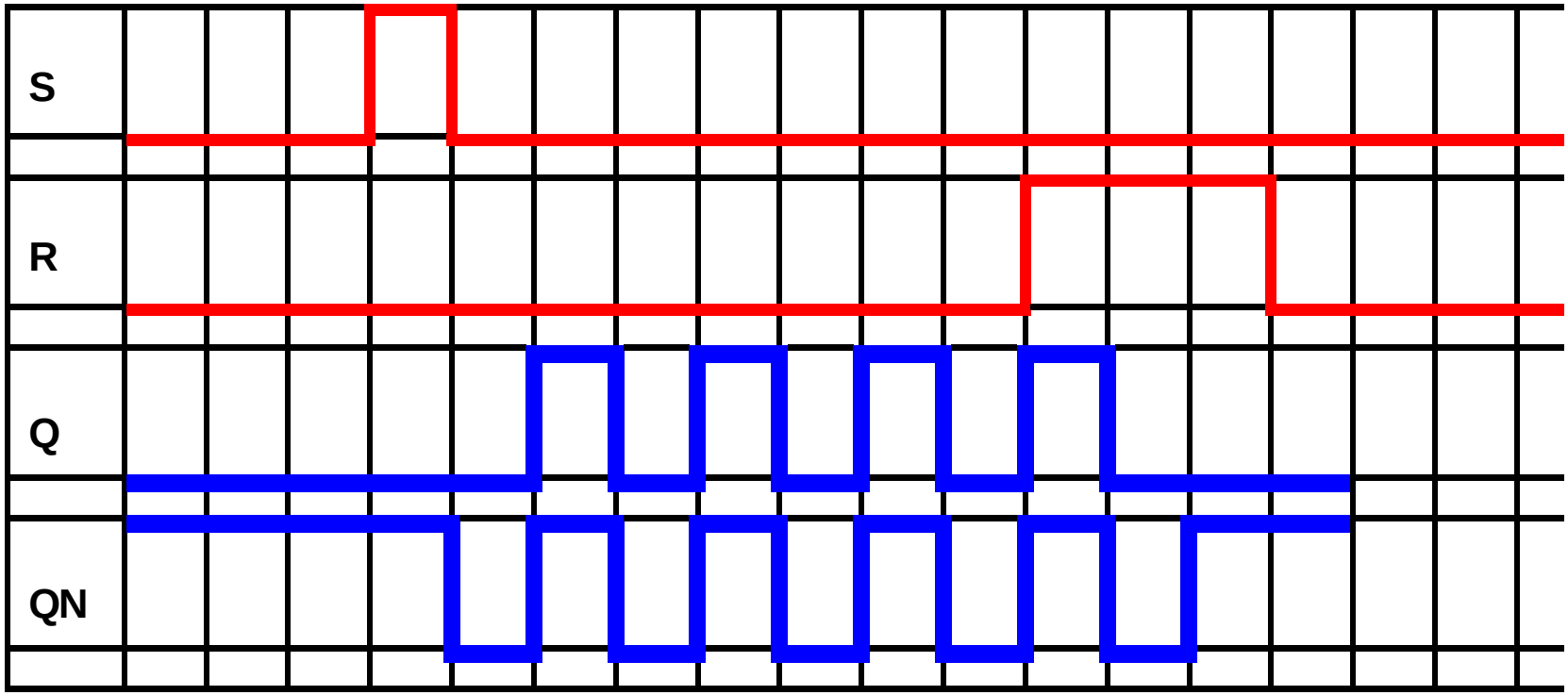


Exercise

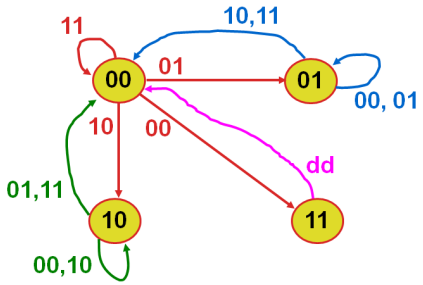


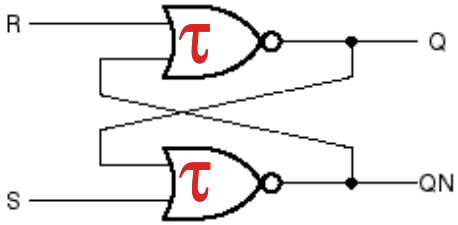


- **Investigate the response of an S-R latch to a glitch or hazard**

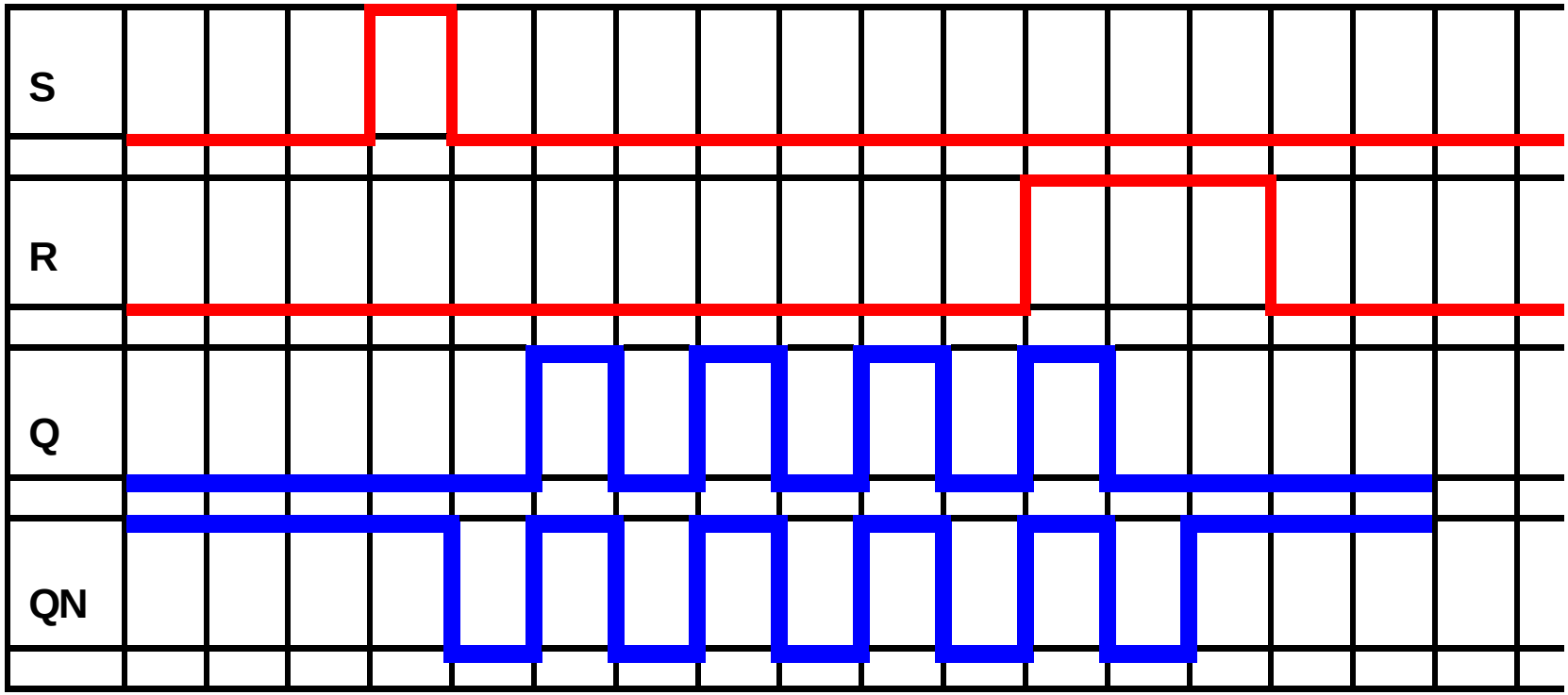


Exercise

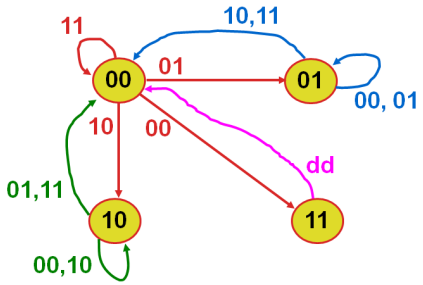


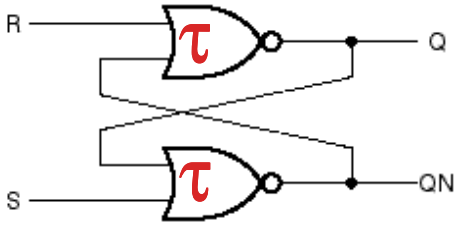


- **Investigate the response of an S-R latch to a glitch or hazard**

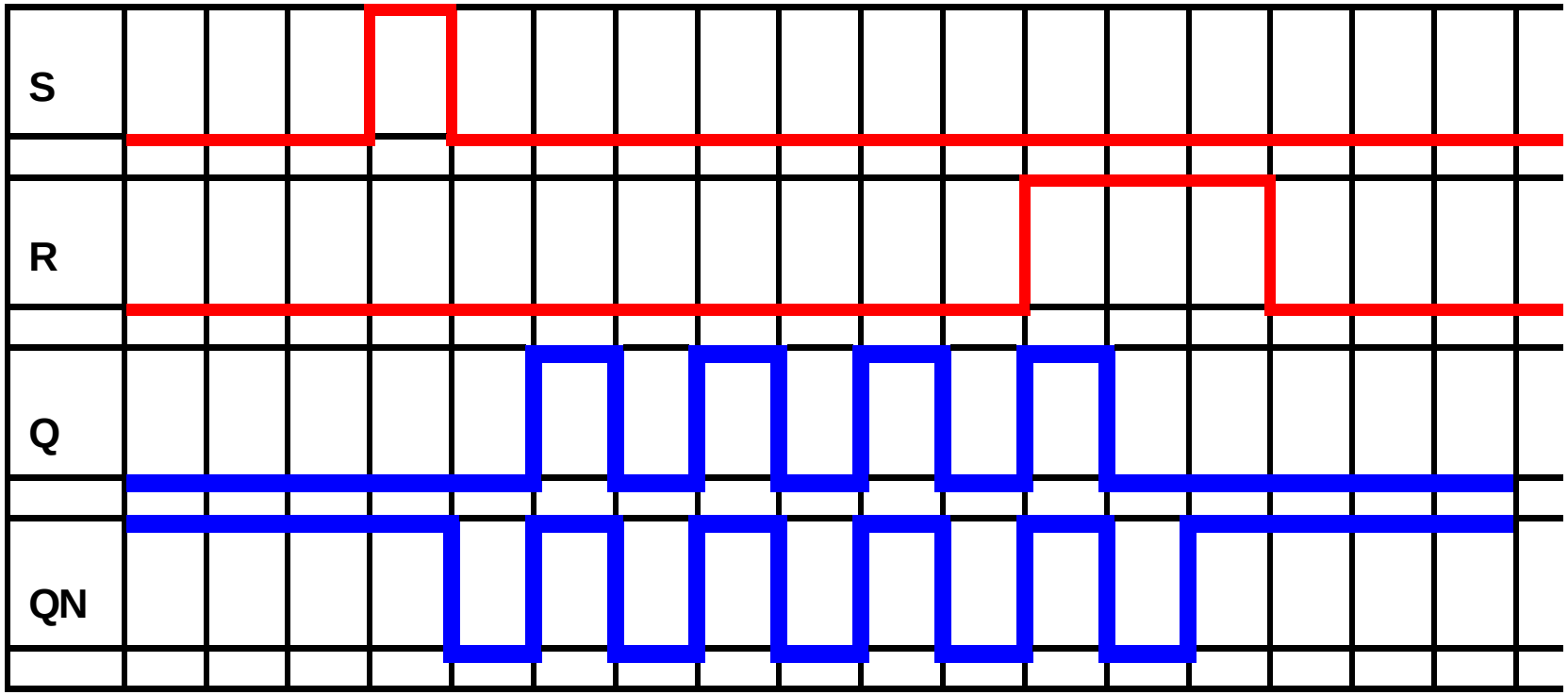


Exercise





- **Investigate the response of an S-R latch to a glitch or hazard**

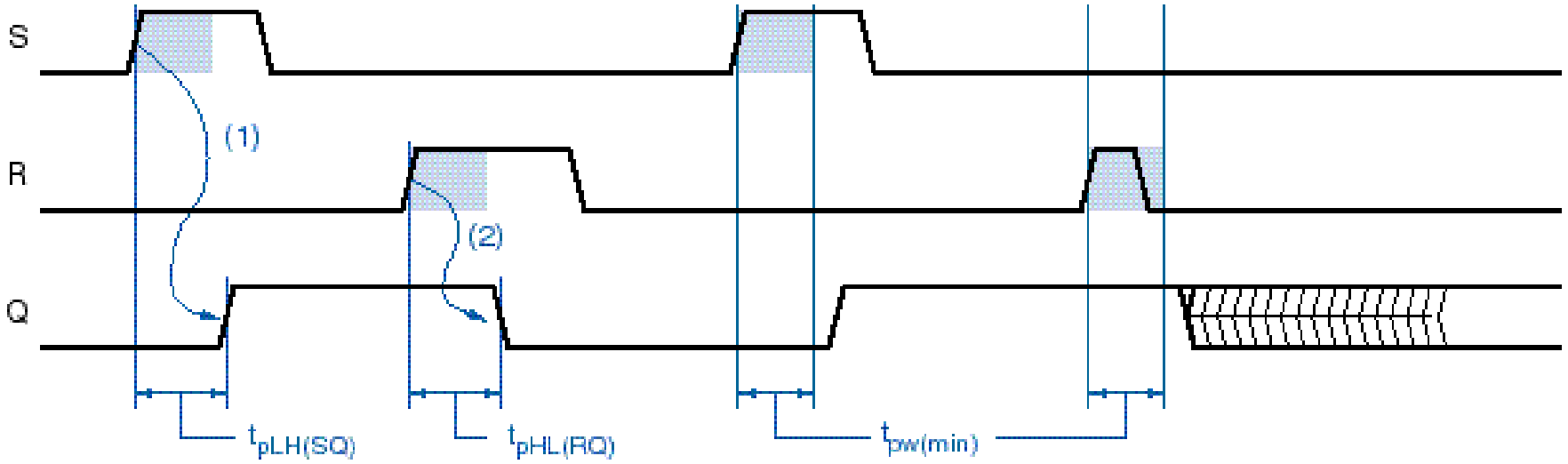


S-R Latch Propagation Delays

- **The propagation delay of a latch is the time it takes for a transition on an input signal to produce a transition on an output signal**
- **A given latch may have several different propagation delay specifications, one for each pair of input and output signals**
- **Also, the propagation delay may be different depending on whether the output makes a LOW-to-HIGH or HIGH-to-LOW transition**
- **<u>Example</u>: $t_{pLH(S \rightarrow Q)}$ is the *rise propagation delay* of the Q output in response to the S input being asserted (latch being "set")**

S-R Latch Input Pulse Width

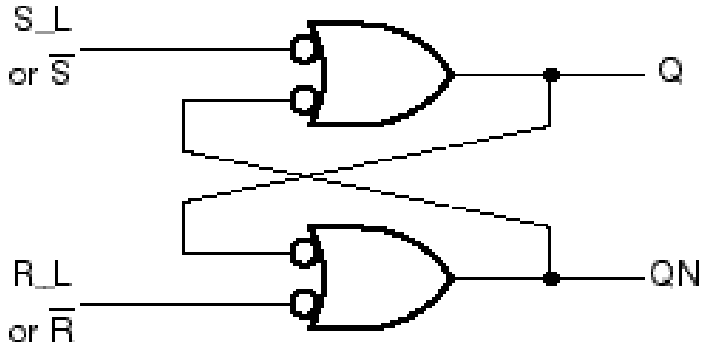
- **Minimum-pulse-width specifications are usually given for the S and R inputs (the latch may go into the metastable state if a pulse shorter than $T_{PW(min)}$ is applied to S or R**



S´-R´ Latch

- **An S′-R′ "S-bar, R-bar" latch – with *active low* set and reset inputs – can be built using NAND gates**

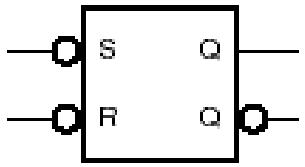
(a)



(b)

S_L	R_L	Q	QN
0	0	1	1
0	1	1	0
1	0	0	1
1	1	last Q	last QN

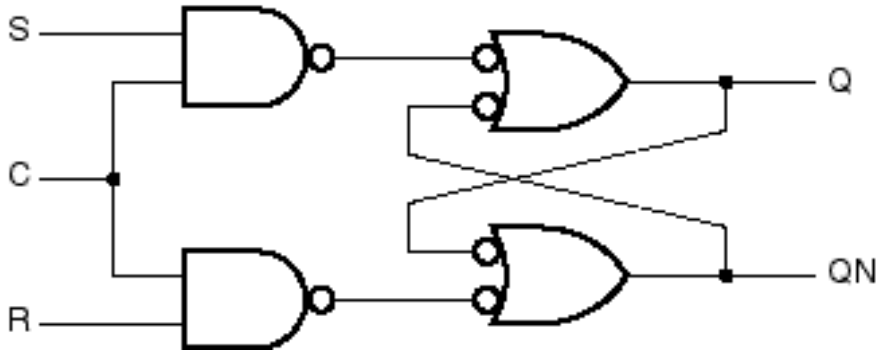
(c)



S-R Latch with Enable

- **An S´-R´ latch can be modified to be sensitive to its inputs only when an enabling input "C" is asserted**
- **The circuit behaves like an S-R latch when C is "1", and retains its state when C is "0"**
- **If both S and R are "1" when C changes from "1" to "0", the next state is unpredictable and the output may become metastable**

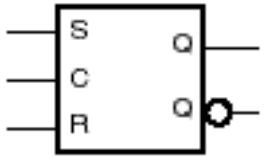
(a)



(b)

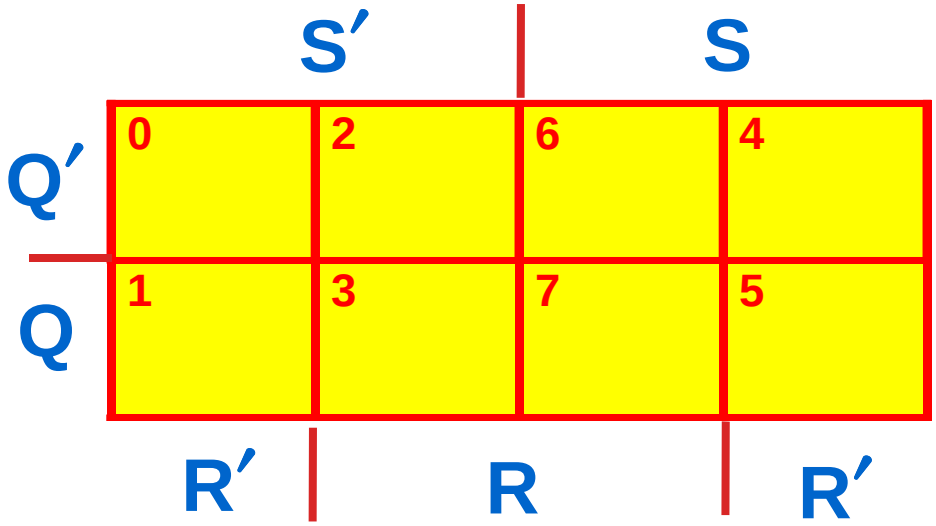
S	R	C	Q	QN
0	0	1	last Q	last QN
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
x	x	0	last Q	last QN

(c)



Exercise – Complete the PS-NS table for an S-R latch and derive its characteristic equation

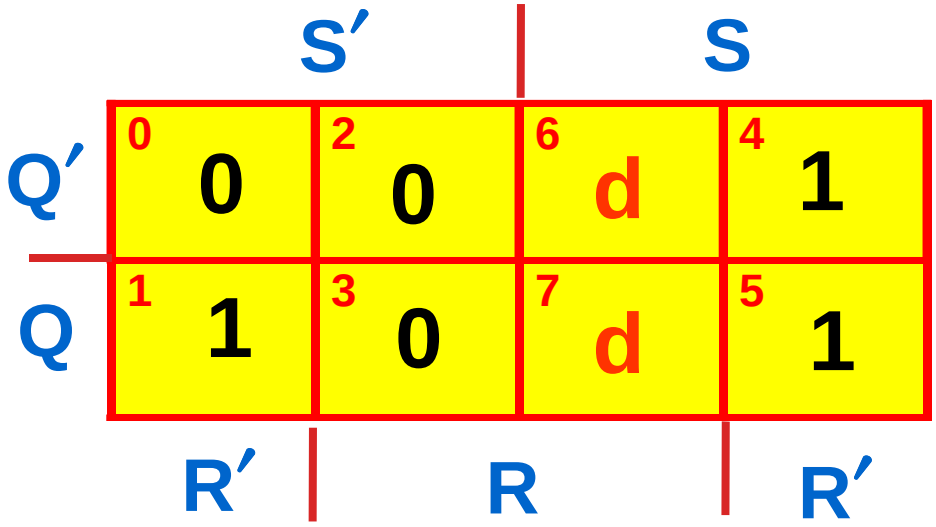
S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d



Q* = _____

Exercise – Complete the PS-NS table for an S-R latch and derive its characteristic equation

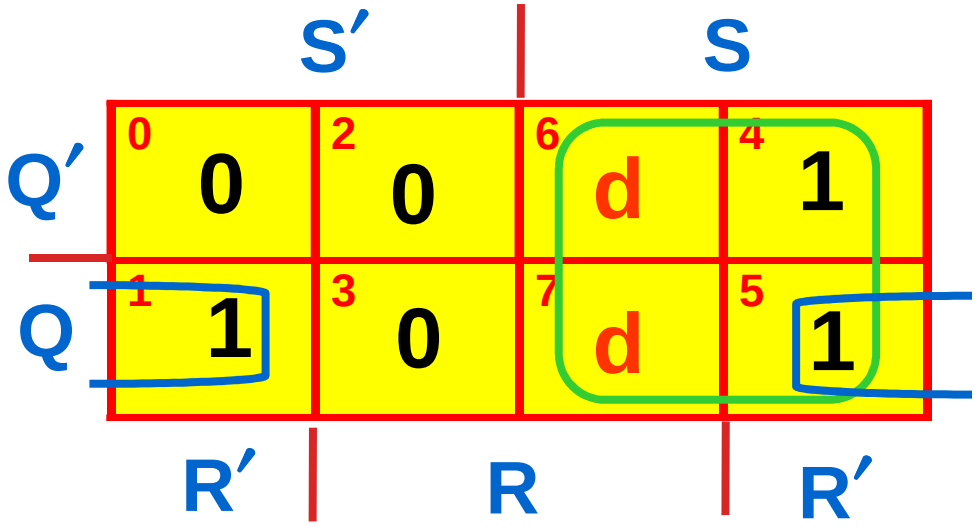
S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d



Q* = _____

Exercise – Complete the PS-NS table for an S-R latch and derive its characteristic equation

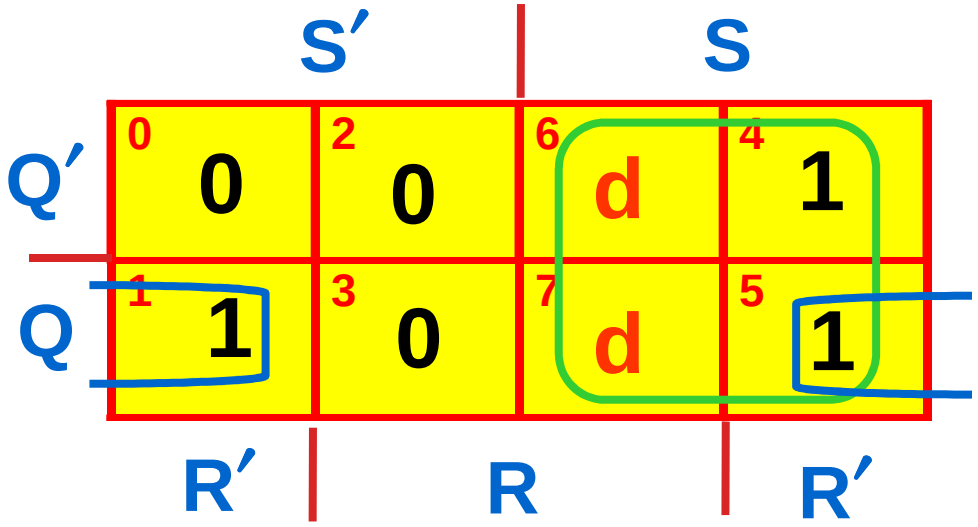
S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d



Q* = _____

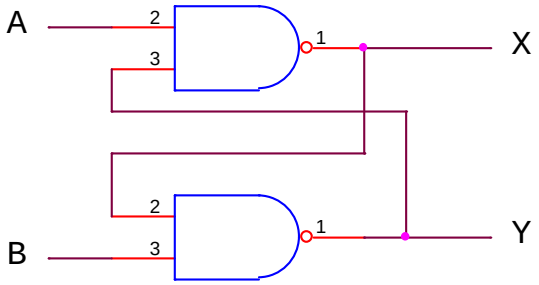
Exercise – Complete the PS-NS table for an S-R latch and derive its characteristic equation

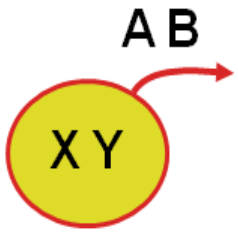
S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d

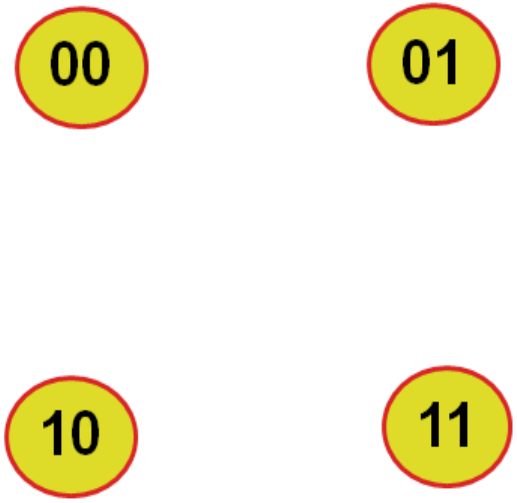


$Q^* = S + R' \cdot Q$

Clicker Quiz







X(t+τ) = _____

Y(t+τ) = _____

Present State	Present Input A(t) B(t)			
X(t) Y(t)	0 0	0 1	1 0	1 1
0 0				
0 1				
1 0				
1 1				

Next State X(t+τ) Y(t+τ)

Q1. For the circuit shown, the following output combination **cannot occur at any time:**

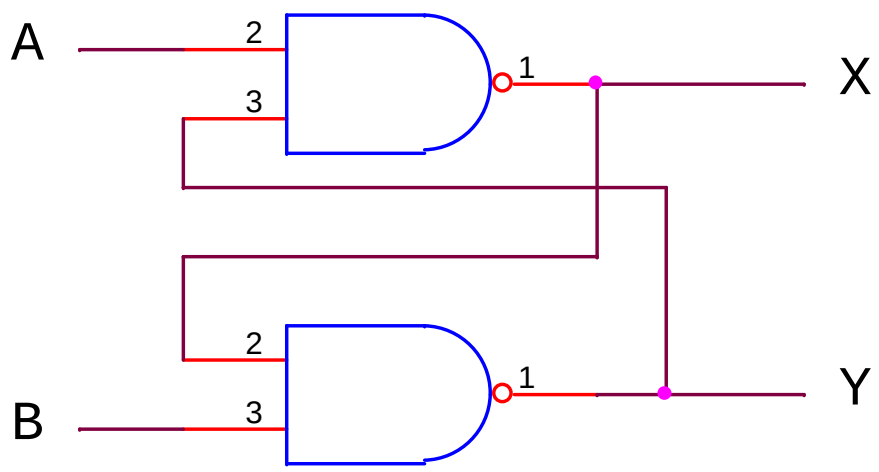
A. X=0, Y=0

B. X=0, Y=1

C. X=1, Y=0

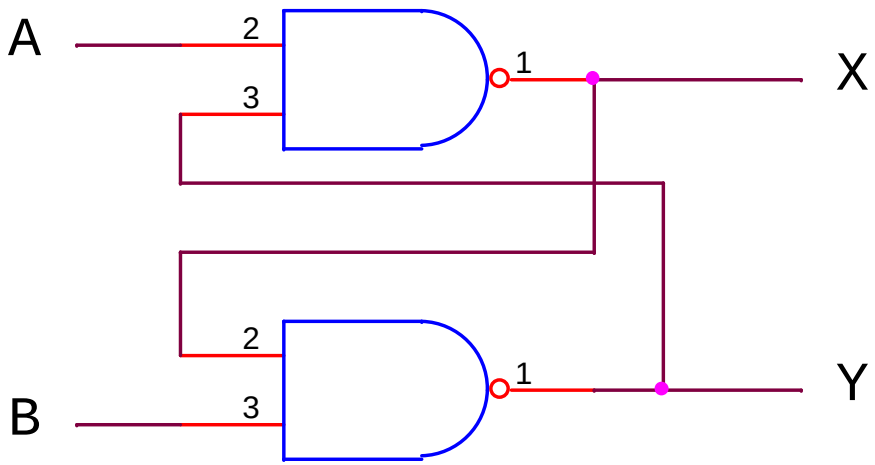
D. X=1, Y=1

E. none of the above



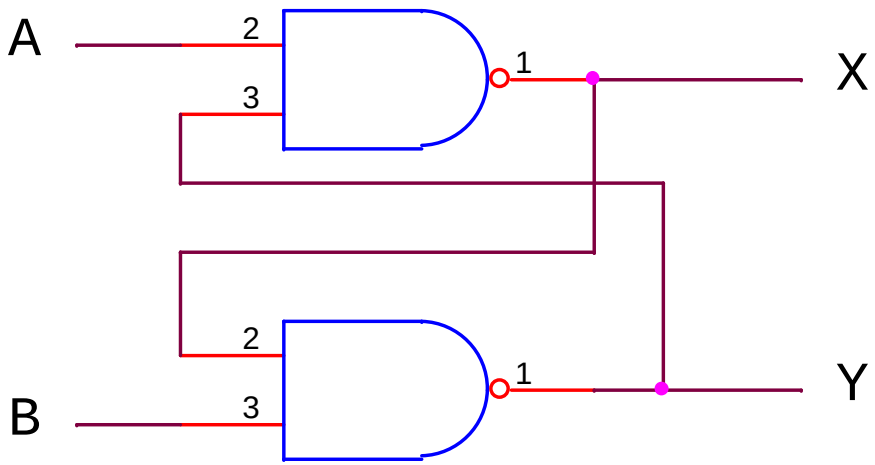
Q2. If the **input** combination **A=0, B=1** is applied to this circuit, the (steady state) output will be:

A. X=0, Y=0
B. X=0, Y=1
C. X=1, Y=0
D. X=1, Y=1
E. unpredictable



Q3. If the **input** combination **A=1, B=0** is applied to this circuit, the (steady state) output will be:

A. X=0, Y=0
B. X=0, Y=1
C. X=1, Y=0
D. X=1, Y=1
E. unpredictable



Q4. If the **input** combination **A=0, B=0** is applied to this circuit, **followed immediately** by the **input** combination **A=1, B=1,** the (steady state) output will be:

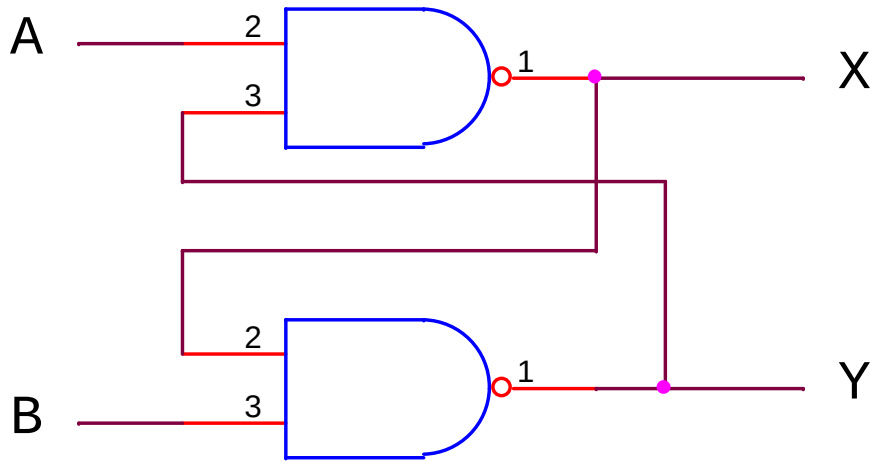
A. X=0, Y=0

B. X=0, Y=1

C. X=1, Y=0

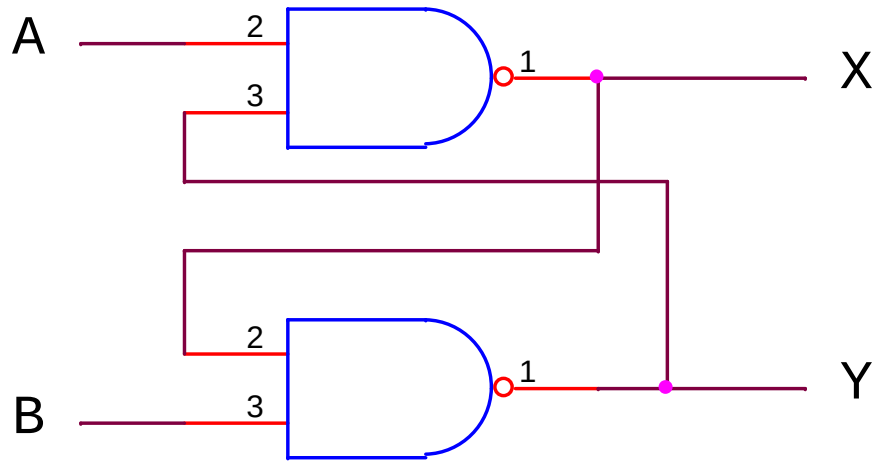
D. X=1, Y=1

E. unpredictable



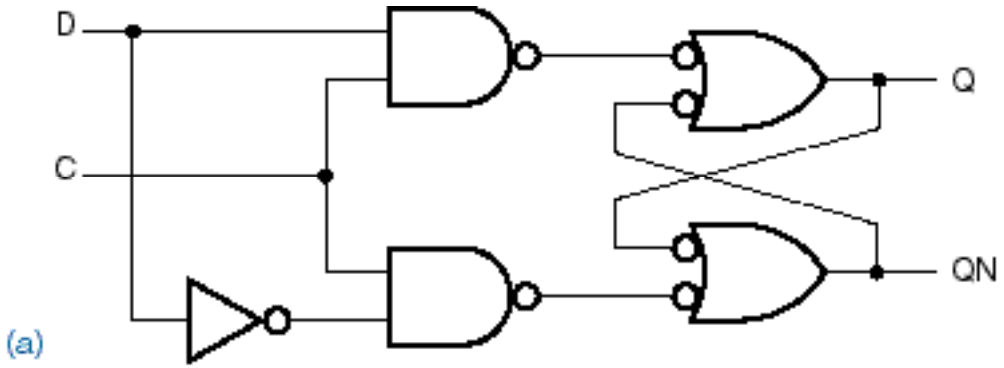
Q5. If the **propagation delay** of each gate is **10 ns**, the **minimum length of time** that (valid) input combinations need to be asserted **in order to prevent metastable behavior** is:

A. 10 ns
B. 20 ns
C. 30 ns
D. 40 ns
E. none of the above



Transparent D Latch

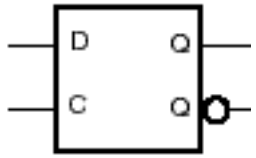
- **In situations where we simply need to store a "bit" of information, a *D latch* can be used**



(a)

C	D	Q	QN
1	0	0	1
1	1	1	0
0	x	last Q	last QN

(b)

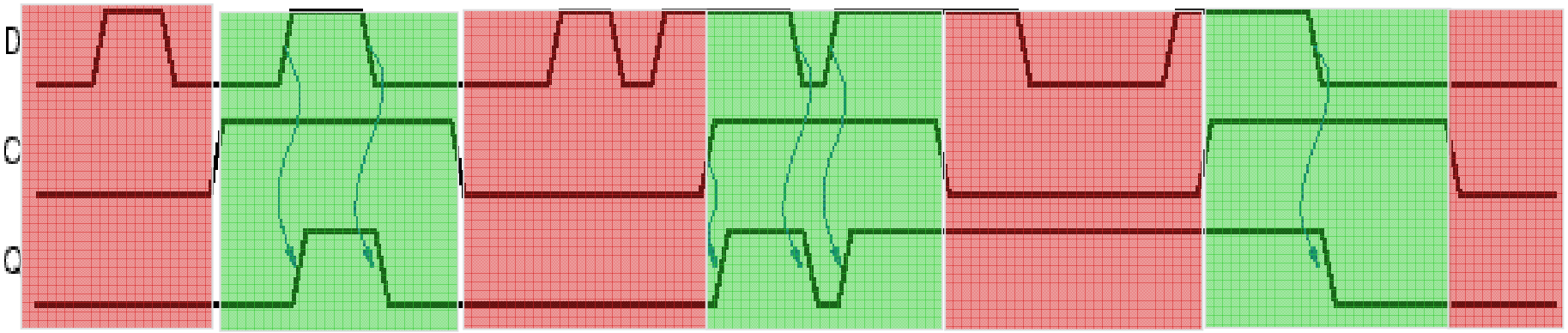


(c)

- **Note that a D latch is just an S-R latch, with D connected to the S input and D′ connected to the R input (this eliminates the troublesome "1-1" input combination)**

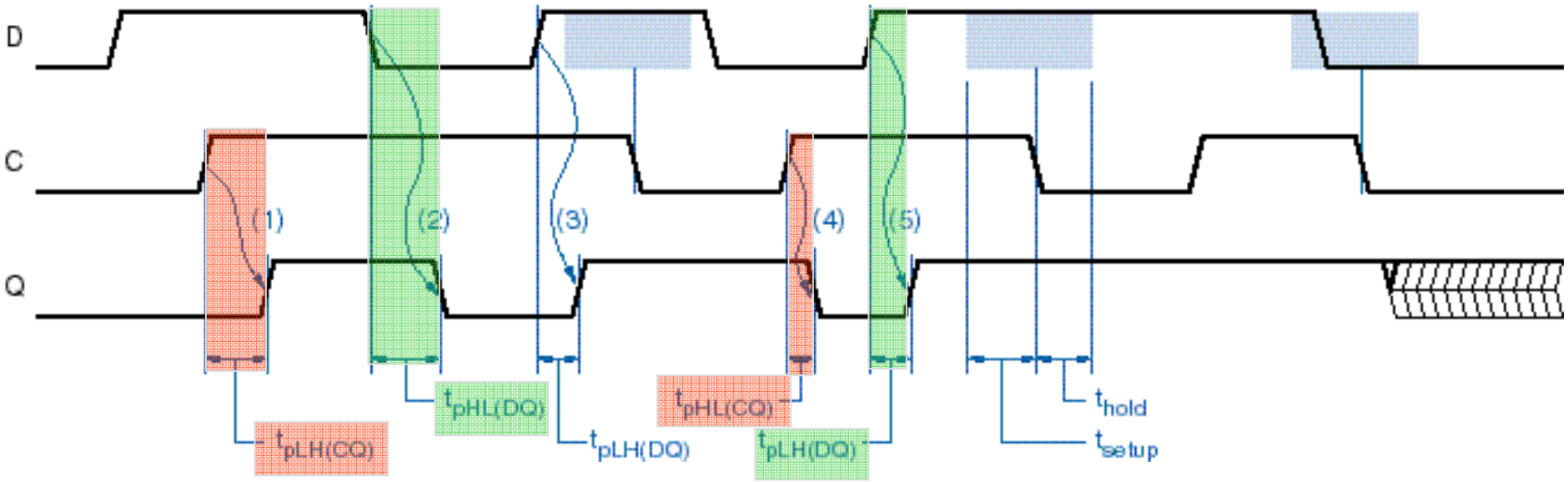
Transparent D Latch

- **When the enable input C is asserted, the latch is said to be “open” and the path from the D input to the Q output is “transparent” – hence the name *transparent latch***
- **When the enable input C is negated, the latch “closes” – the Q output retains its last value and no longer changes in response to D**



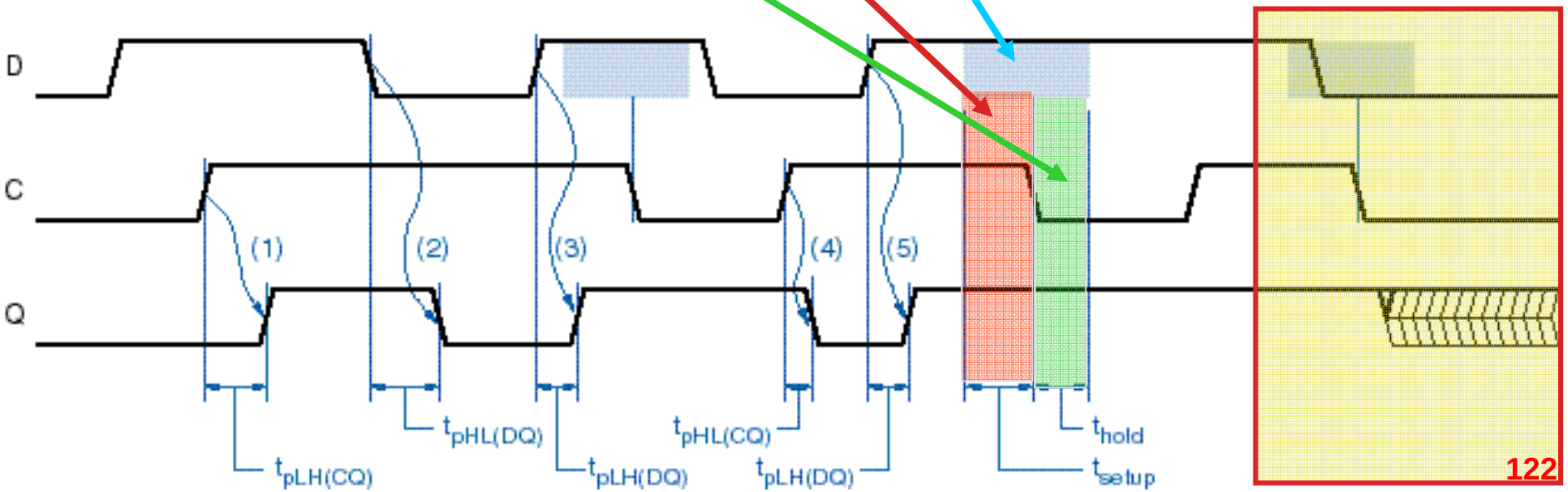
D Latch Propagation Delays

- **There are four propagation delay parameters that must be considered:**
  - **$t_{pLH(C \rightarrow Q)}$ and $t_{pHL(C \rightarrow Q)}$**
  - **$t_{pLH(D \rightarrow Q)}$ and $t_{pHL(D \rightarrow Q)}$**



D Latch Setup and Hold Times

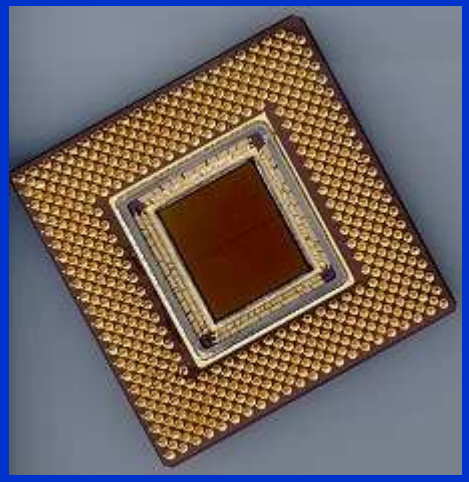
- **There is a "window" of time around the falling edge of C when the D input *must not change***
  - **the time *prior* to this edge that the D input must remain stable is the *setup time***
  - **the time *after* this edge that the D input must remain stable is the *hold time***



Clicker Quiz

Q1. A **“D” latch** is called **transparent** because its output:

A. is always equal to its input
B. is equal to its input when the latch is closed
C. is equal to its input when the latch is open
D. changes state as soon as the latch is clocked
E. none of the above



Introduction to Digital System Design

Module 3-C

Data (D) and Toggle (T) Flip-Flops

Reading Assignment:

DDPP 4th Ed., pp. 532-535, 541-542

Learning Objectives:

- **Draw a circuit for an edge-triggered data ("D") flip-flop and analyze its behavior**
- **Compare the response of a latch and a flip-flop to the same set of stimuli**
- **Define setup and hold time and determine their nominal values from a timing chart**
- **Determine the frequency and duty cycle of a clocking signal**
- **Identify latch and flip-flop propagation delay paths and determine their values from a timing chart**
- **Describe the operation of a toggle ("T") flip-flop and analyze its behavior**
- **Derive a characteristic equation for any type of latch or flip-flop**

Outline

- Overview
- Positive edge-triggered D flip-flop
- Negative edge-triggered D flip-flop
- D flip-flop characteristic equation
- D flip-flop setup and hold times
- D flip-flop with enable
- Edge-triggered T flip-flop
- T flip-flop characteristic equation
- Flip-flop timing parameters
- Response of latch vs. flip-flop
- Summary

Overview

- **Definition: A *flip-flop* is a sequential circuit that samples its inputs and *changes its outputs* only at times determined by a *clocking signal* ("CLK")**
- **Flip-flops change state in response to the *transition* ("edge") of a clocking signal**
  - ***positive-edge-triggered* flip-flops change state on the *low-to-high* transition of a clocking signal**
  - ***negative-edge-triggered* flip-flops change state on the *high-to-low* transition of a clocking signal**

Positive Edge-Triggered D Flip-Flop

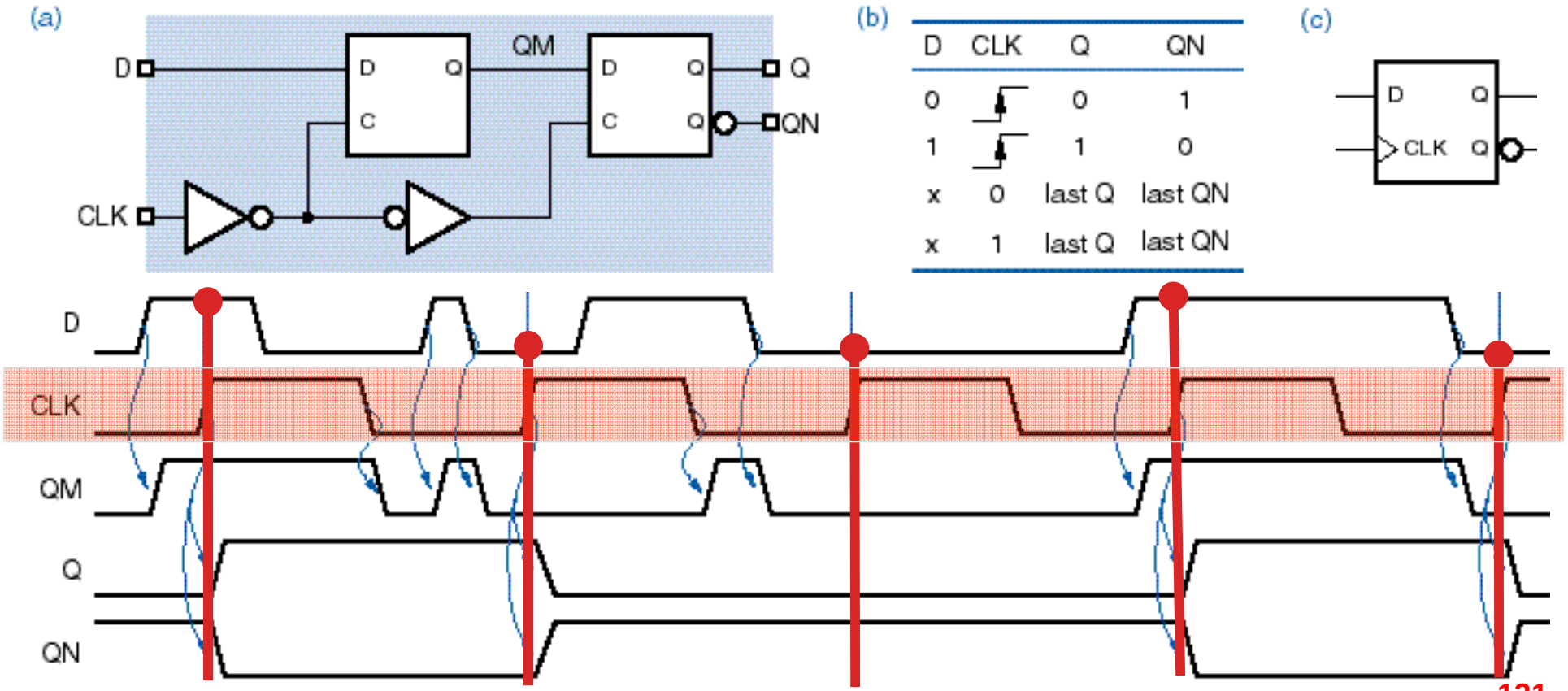
- **A *positive-edge-triggered D flip flop* combines a pair of D latches to create a circuit that samples its D input and changes its Q and QN outputs *at the rising edge* of a controlling CLOCK (CLK) signal**
  - **the first latch, called the *master*, opens and follows the input when CLK is 0**
  - **the second latch, called the *slave*, opens and reads the master's output when CLK is 1 – this is when the output state change occurs (note that the master latch is *closed* at this point and thus "immune" to input changes)**

Positive Edge-Triggered D Flip-Flop

- **A *triangle* on the D flip-flop's CLK input indicates edge-triggered behavior and is called a *dynamic input indicator***
- **The characteristic equation of a D flip-flop is Q* = D – i.e. the next state is the current input, shorthand for Q(t+Δ) = D(t), where Δ is the *clocking period***
- **D flip-flops are included in the macrocells of virtually all PLDs, and are therefore the "most popular" (and easiest) way to realize clocked synchronous state machines**

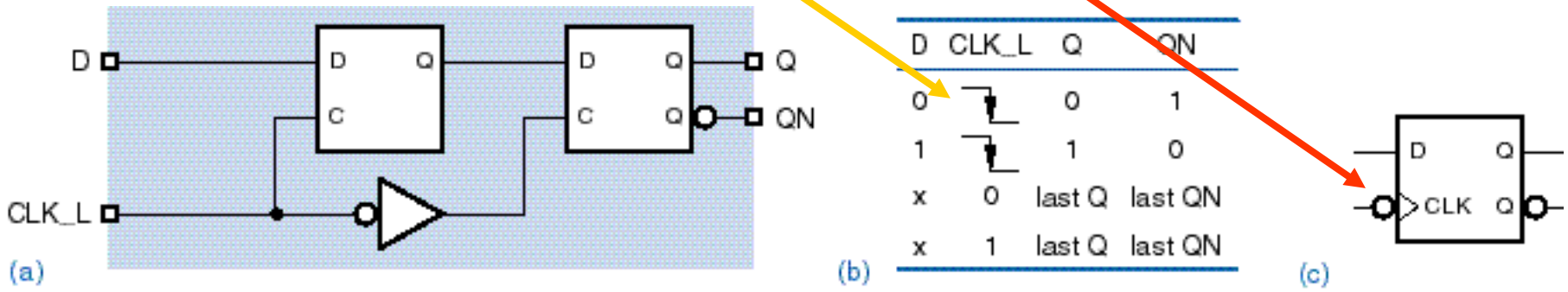
Positive Edge-Triggered D Flip-Flop

- **One way an edge-triggered D flip flop can be constructed is illustrated below**

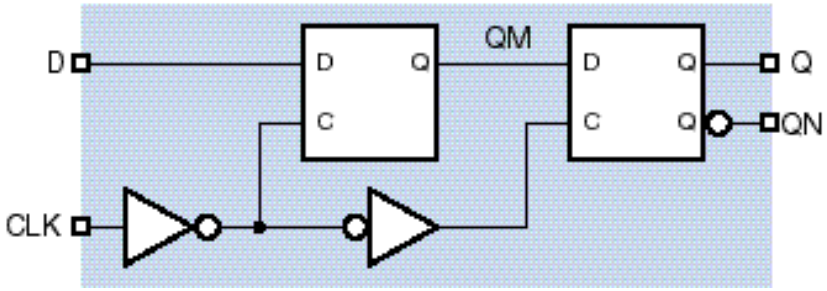


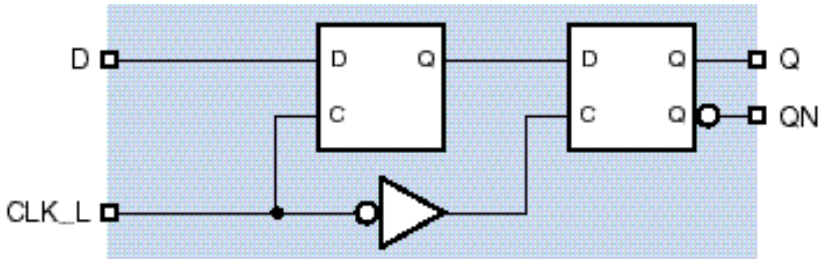
Negative Edge-Triggered D Flip-Flop

- **D flip flops can also be designed to be *negative-edge-triggered***
- **An *inversion bubble* on the CLK input is used to indicate that a flip flop is triggered on the HIGH-to-LOW transition of the CLK signal**



Clicker Quiz


D
CLK
D
Q
C
QM
D
Q
C
Q
Q
QN


D
CLK_L
D
Q
C
D
Q
C
Q
Q
QN

Q1. The **minimum** number of gates needed to implement a **positive edge-triggered D flip-flop** using **only 2-input NAND** gates is:

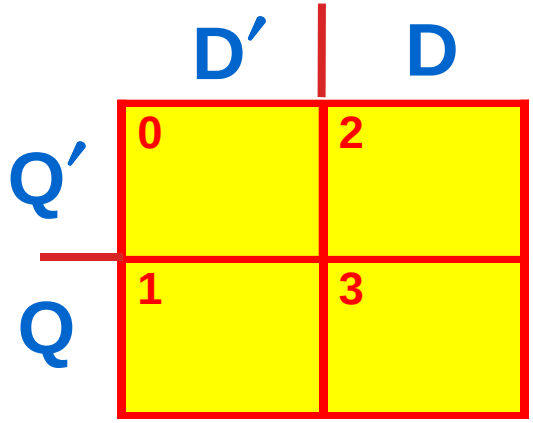
A. 8 B. 9 C. 10 D. 11 E. 12

Q2. The **minimum** number of gates needed to implement a **negative edge-triggered D flip-flop** using **only 2-input NAND** gates is:

A. 8 B. 9 C. 10 D. 11 E. 12

Exercise – Complete the PS-NS table for an D flip-flop and derive its characteristic equation

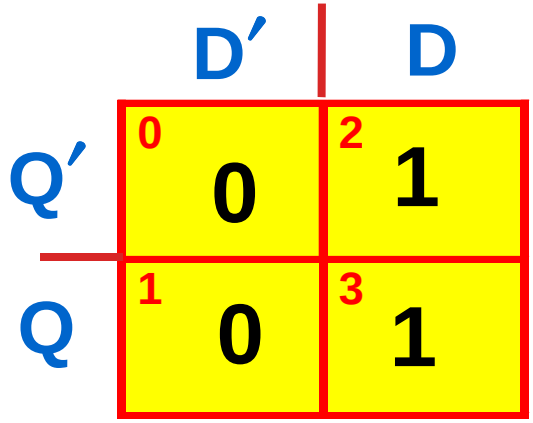
D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1



Q* = _____

Exercise – Complete the PS-NS table for an D flip-flop and derive its characteristic equation

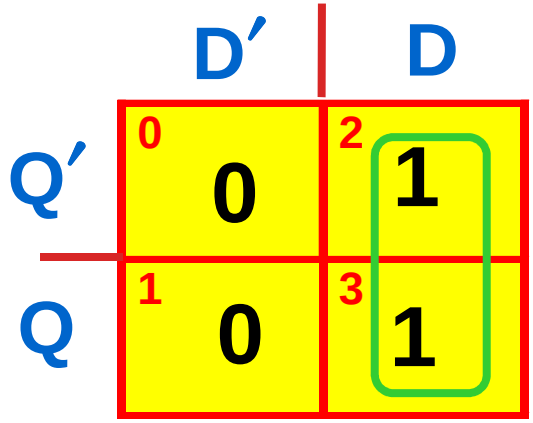
D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1



Q* = _____

Exercise – Complete the PS-NS table for an D flip-flop and derive its characteristic equation

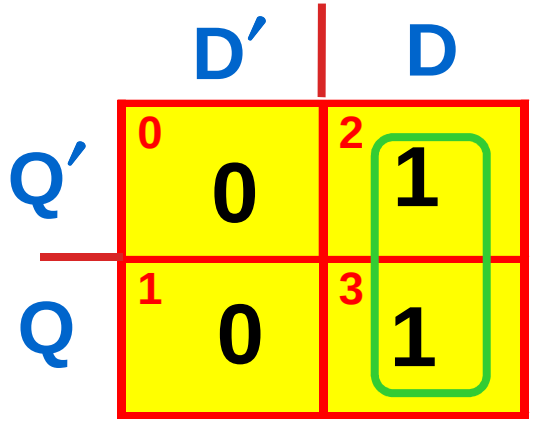
D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1



Q* = _____

Exercise – Complete the PS-NS table for an D flip-flop and derive its characteristic equation

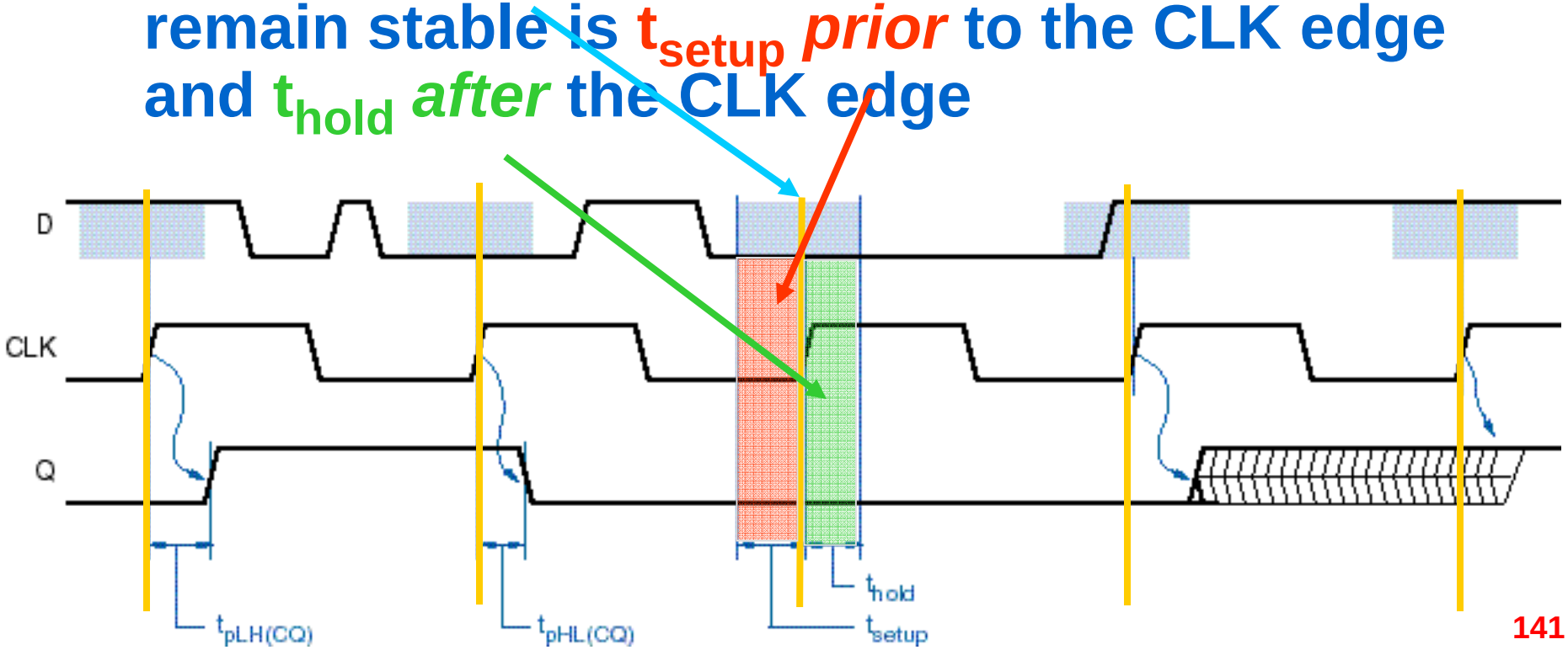
D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1



$Q^* =$ D

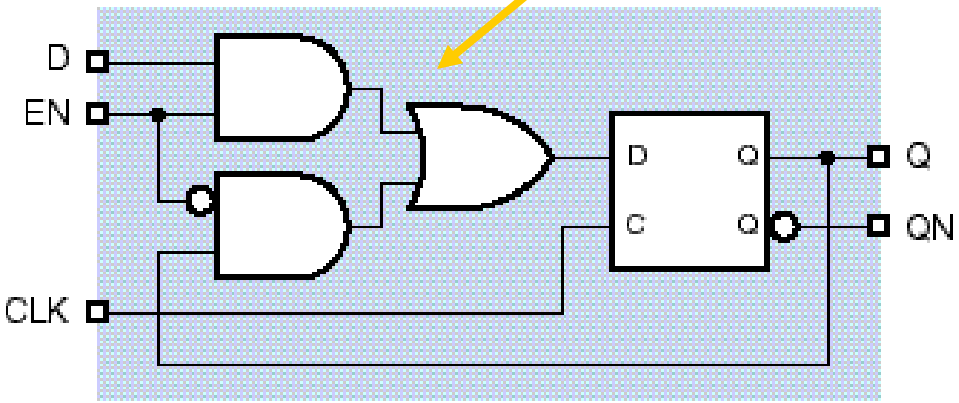
D Flip-Flop Setup and Hold Times

- **For edge-triggered flip-flops, all propagation delays are measured from the *rising edge* of the CLK signal**
- **The "window" during which the D input must remain stable is t_{setup} *prior* to the CLK edge and t_{hold} *after* the CLK edge**

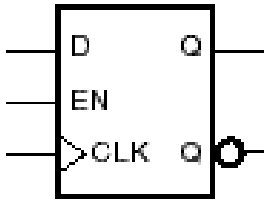


D Flip-Flop with Enable

- **A commonly desired function in D flip-flops is to retain the last value stored (rather than load a new one) at the clock edge**
- **This is accomplished by adding an enable input, called EN or CE (clock enable), which uses a 2:1 multiplexer to control the value applied to the internal D flip-flop input**

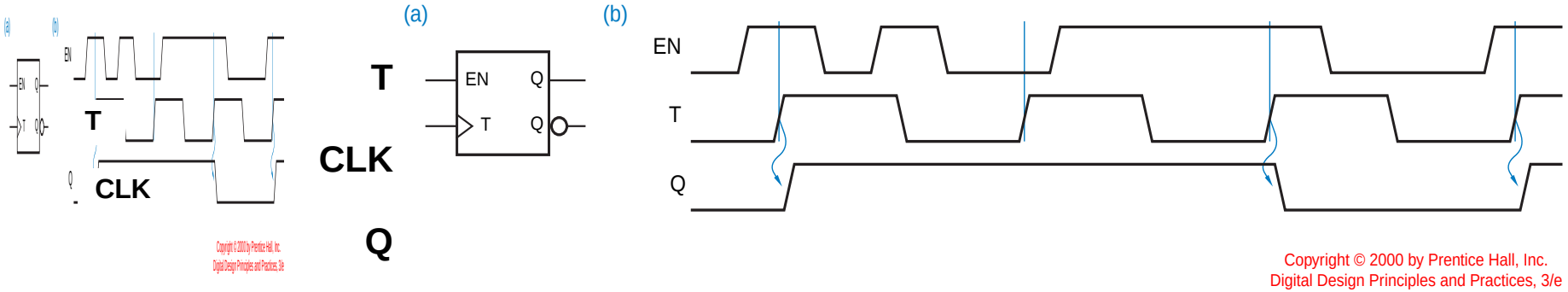


D	EN	CLK	Q	QN
0	1	↑	0	1
1	1	↑	1	0
x	0	↑	last Q	last QN
x	x	0	last Q	last QN
x	x	1	last Q	last QN



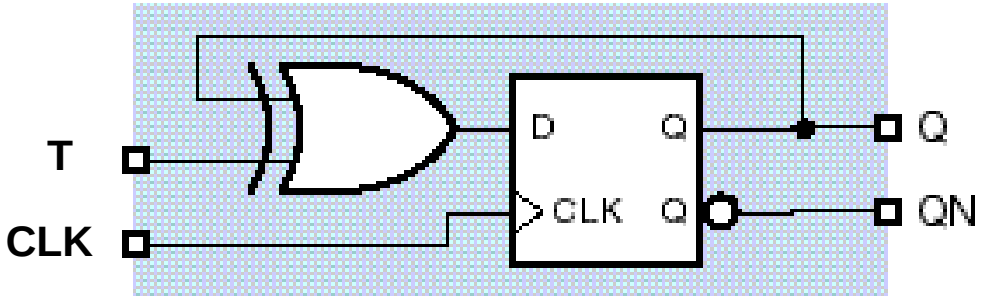
Edge-Triggered T Flip Flop

- **A positive edge-triggered toggle (T) flip-flop changes to the *complement* of its former state ("toggles") in response to a positive clock edge *when enabled***
- **The T input is used to enable/disable the flip-flop from toggling**
  - **when T=0, Q* = Q *stays in same state***
  - **when T=1, Q* = Q′ *toggles***
- **The characteristic equation for a T flip-flop is**
  Q* = T′•Q + T•Q′ = T⊕Q



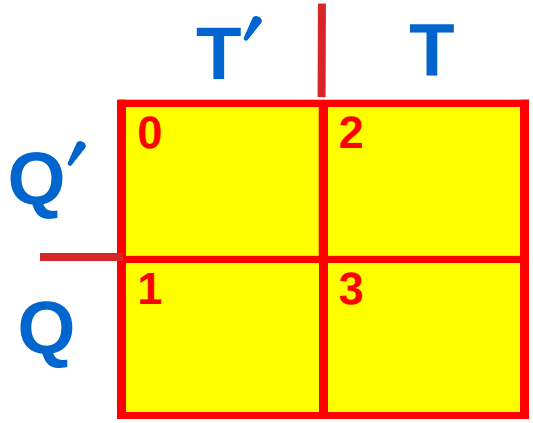
Edge-Triggered T Flip Flop

- **A T flip-flop can be realized using a D flip-flop by implementing the T flip-flop characteristic equation**



Exercise – Complete the PS-NS table for a T flip-flop and derive its characteristic equation

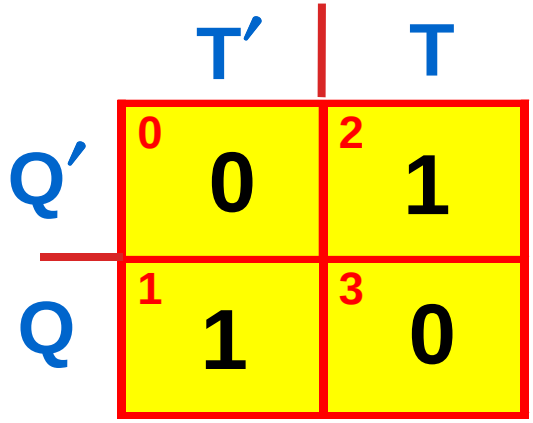
T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0



Q* = _____

Exercise – Complete the PS-NS table for a T flip-flop and derive its characteristic equation

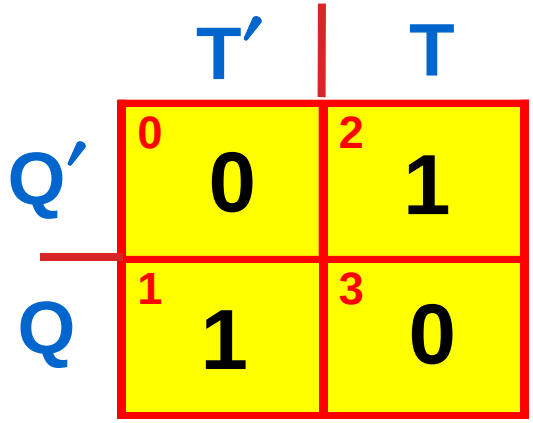
T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0



Q* = _____

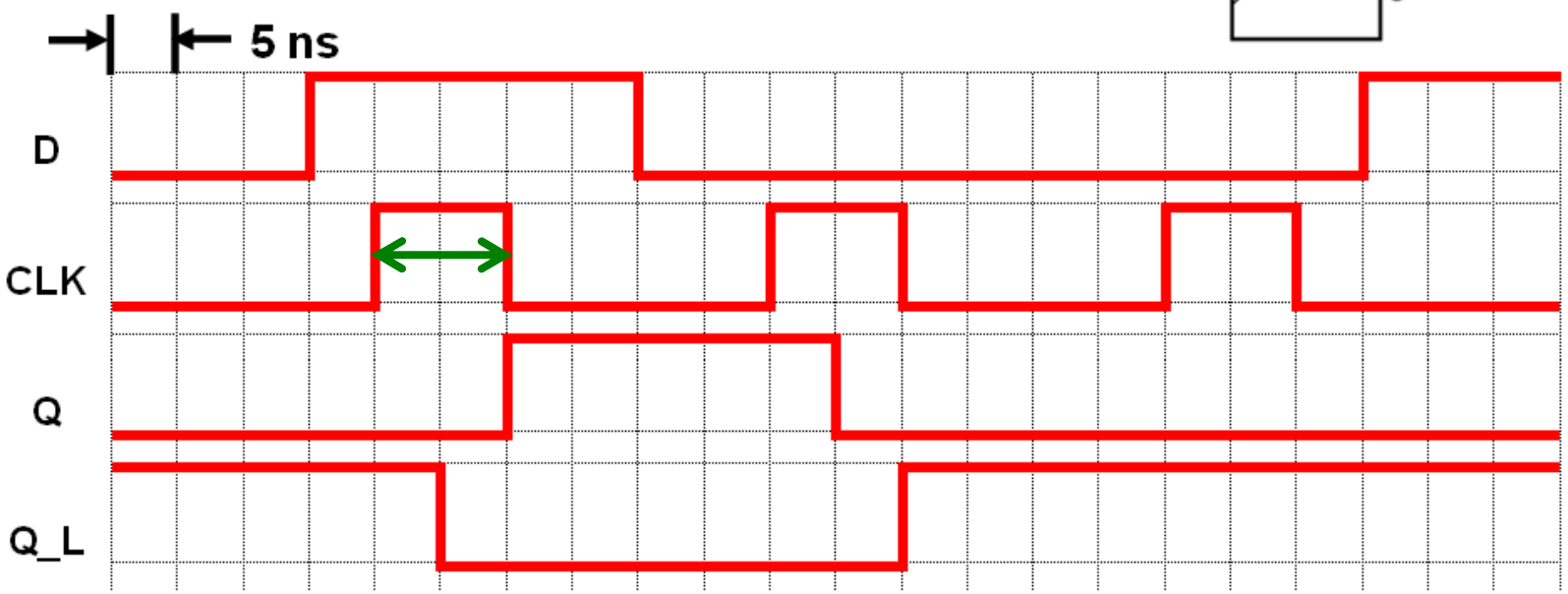
Exercise – Complete the PS-NS table for a T flip-flop and derive its characteristic equation

T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0



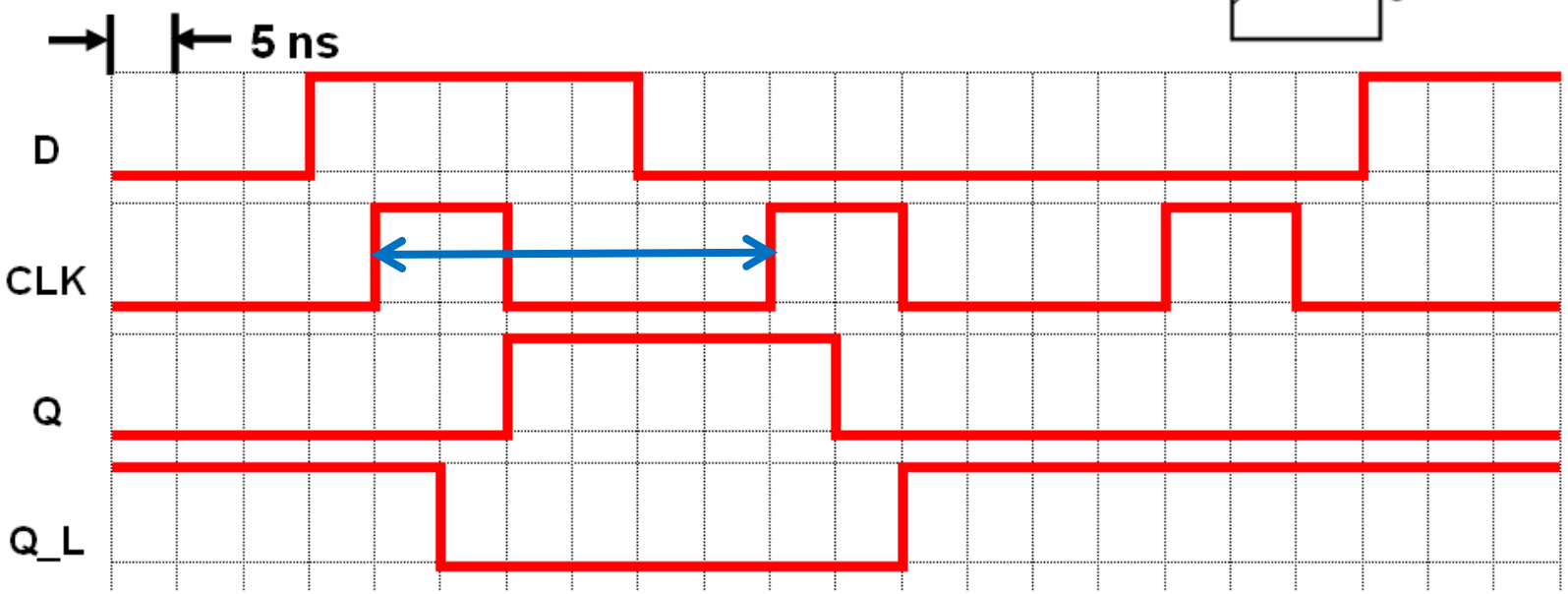
$$Q^* = \underline{Q \cdot T' + Q' \cdot T = Q \oplus T}$$

Example – Flip-Flop Timing Parameters



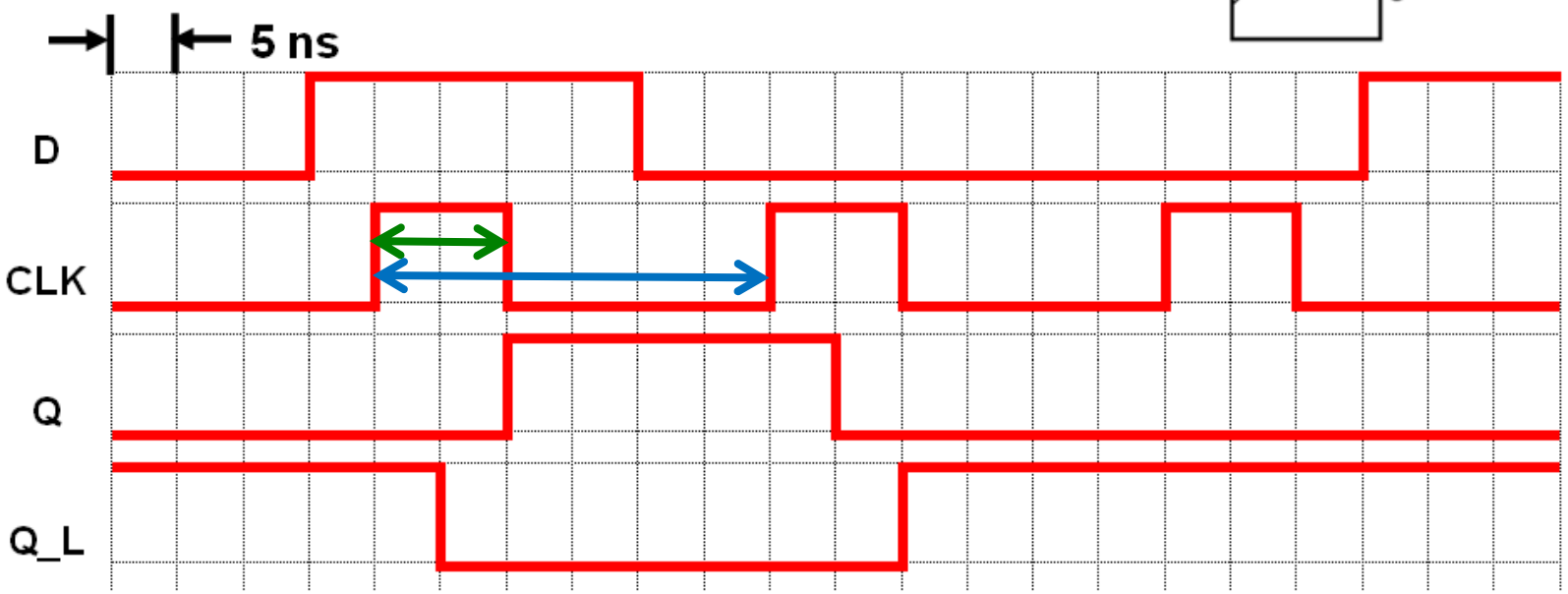
The **clock pulse width** provided for the D flip-flop is 10 ns

Example – Flip-Flop Timing Parameters



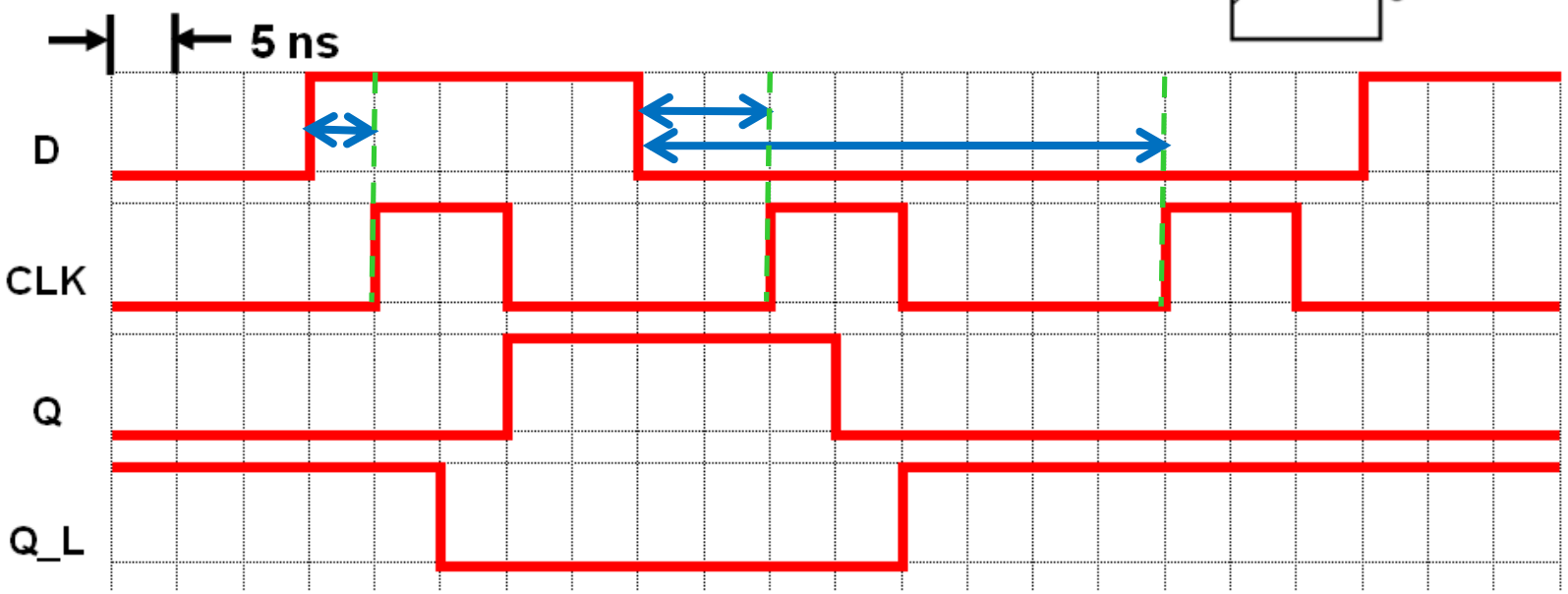
The **clock period** provided for the D flip-flop is 30 ns

Example – Flip-Flop Timing Parameters



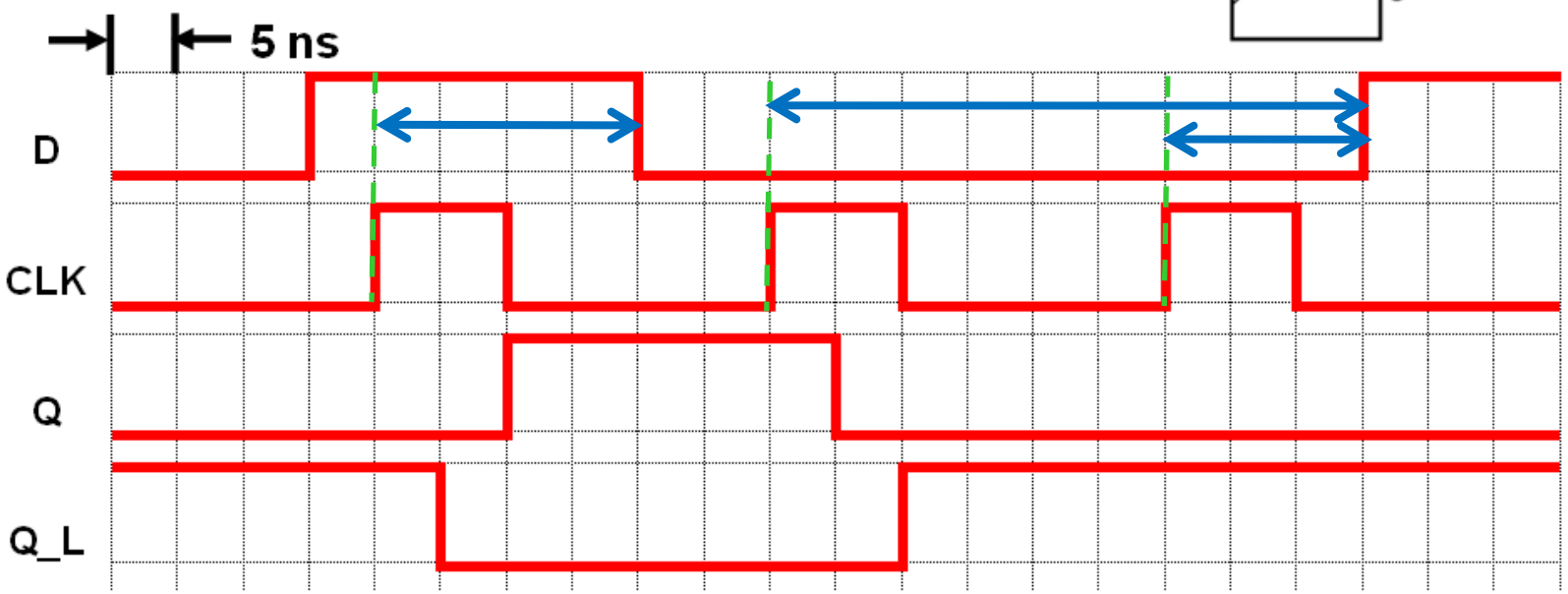
The **duty cycle** of the clocking signal is
10/30 x 100% = 33%

Example – Flip-Flop Timing Parameters



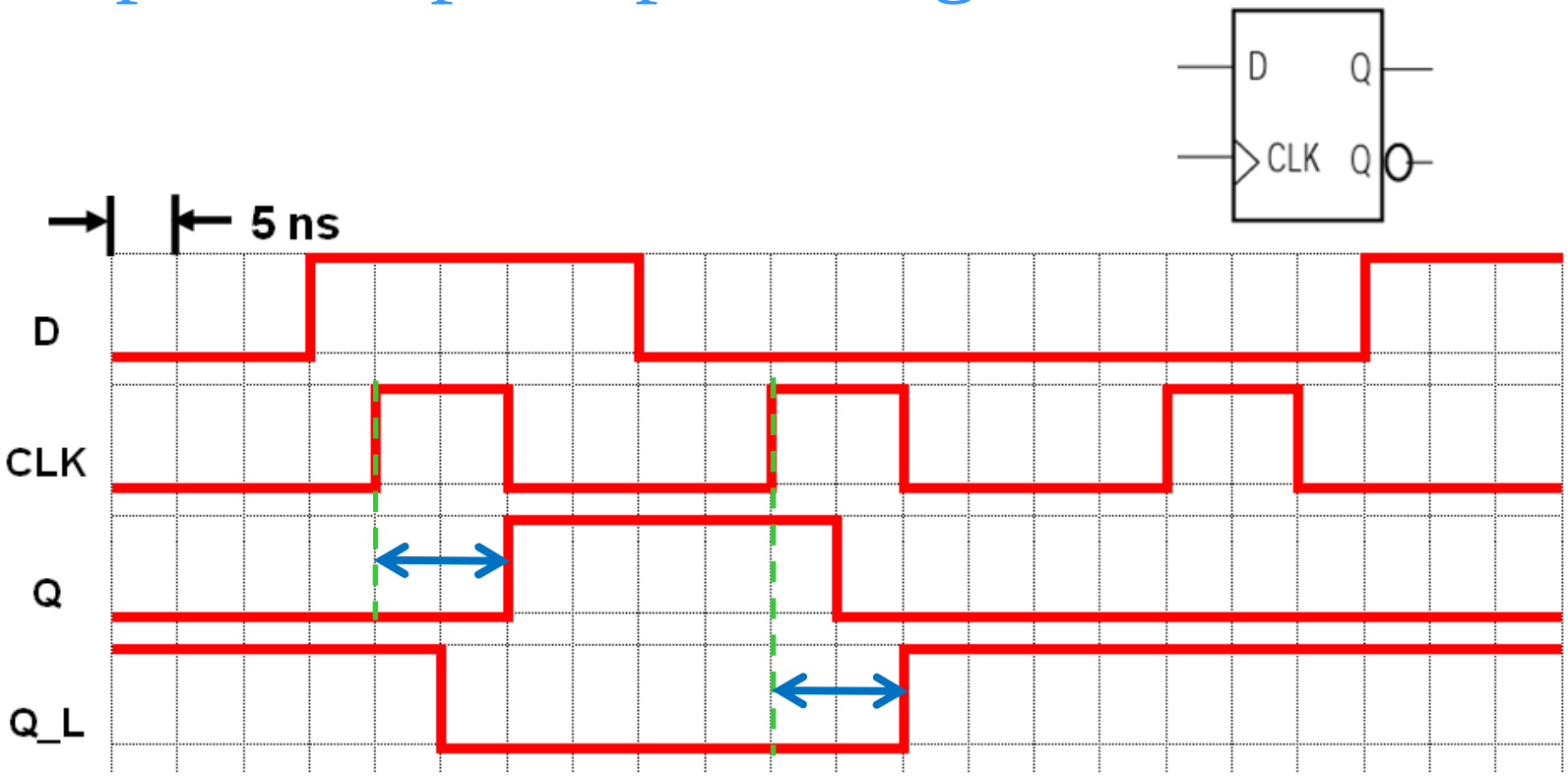
The **nominal** **setup time** provided for the D flip-flop is 5 ns

Example – Flip-Flop Timing Parameters



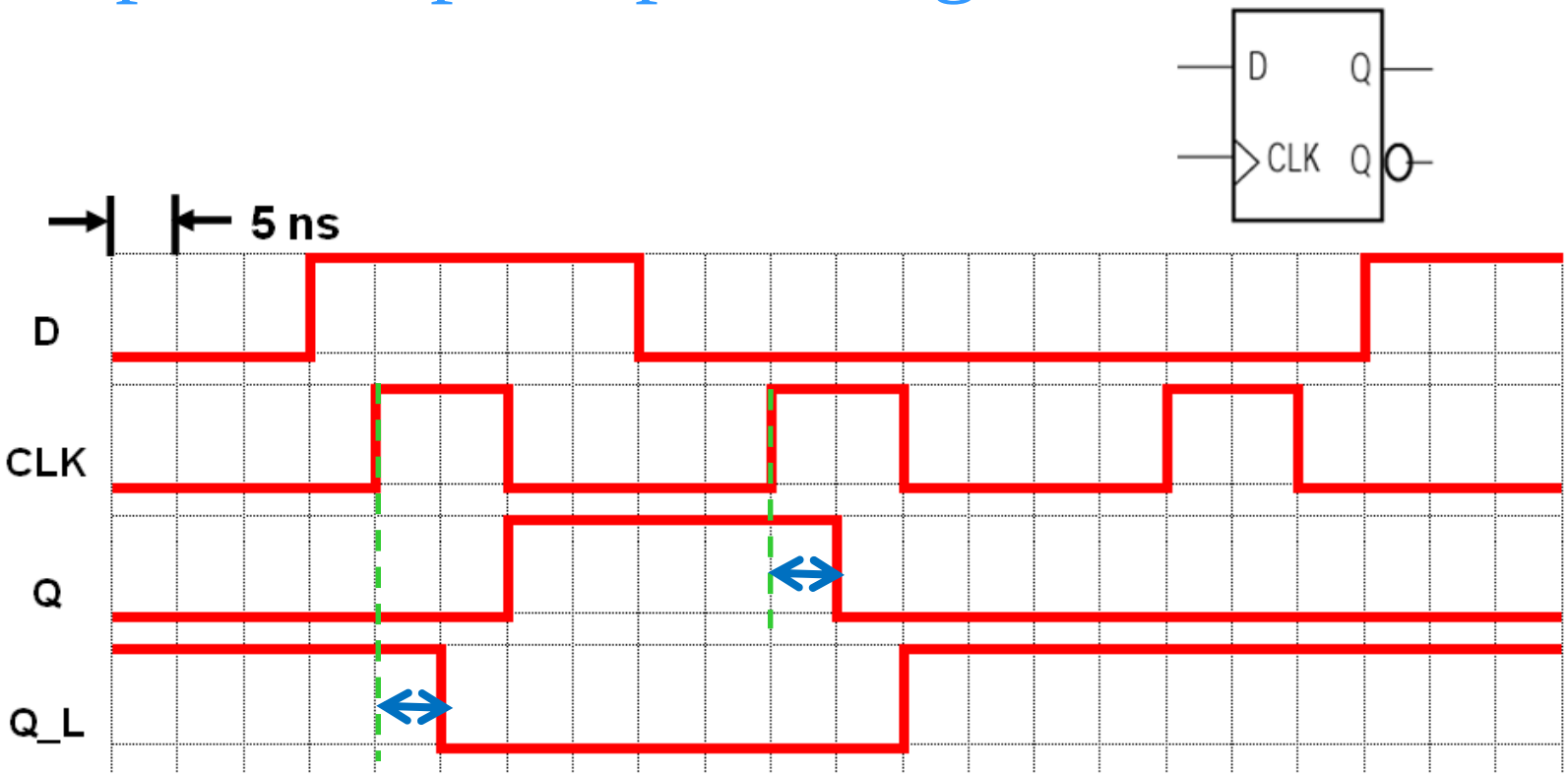
The **nominal** **hold time** provided for the D flip-flop is 15 ns

Example – Flip-Flop Timing Parameters



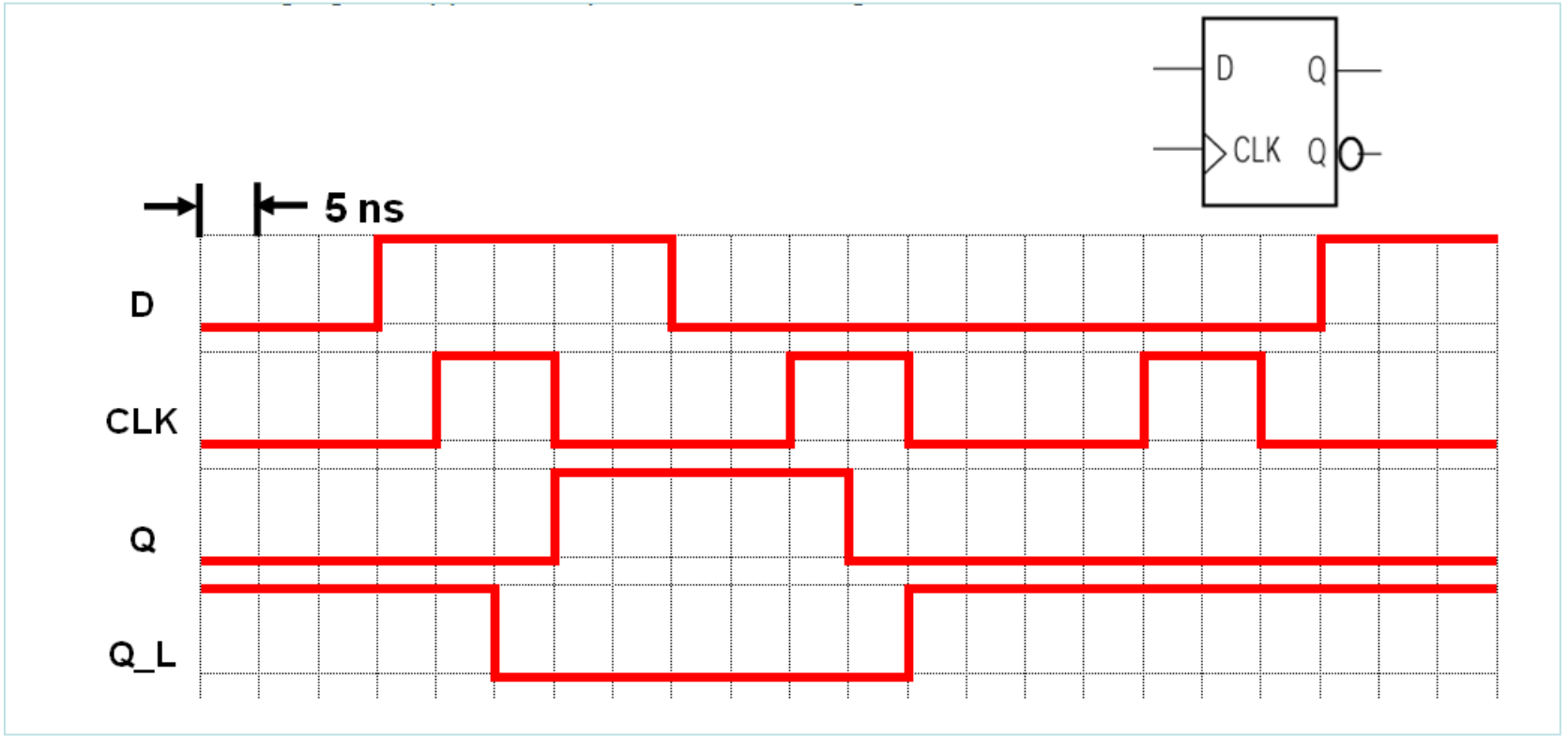
The $t_{PLH(C \rightarrow Q)}$ = $t_{PLH(C \rightarrow Q\_L)}$ of the D flip-flop is 10 ns

Example – Flip-Flop Timing Parameters



The $t_{PHL(C \rightarrow Q)}$ = $t_{PHL(C \rightarrow Q\_L)}$ of the D flip-flop is 5 ns

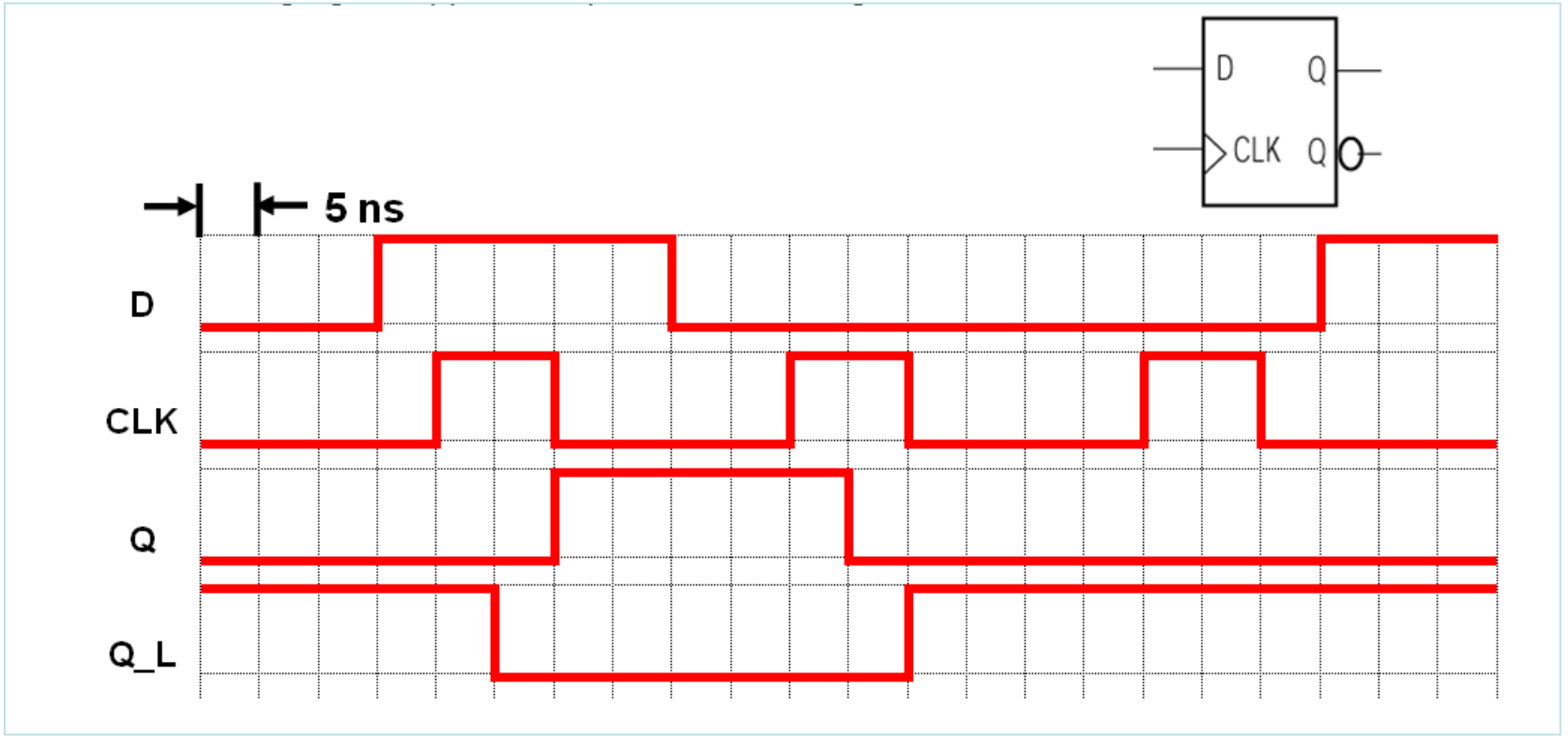
Clicker Quiz



Q1. The **duty cycle** of the clocking signal is:

A. 20% B. 33% C. 40% D. 67%

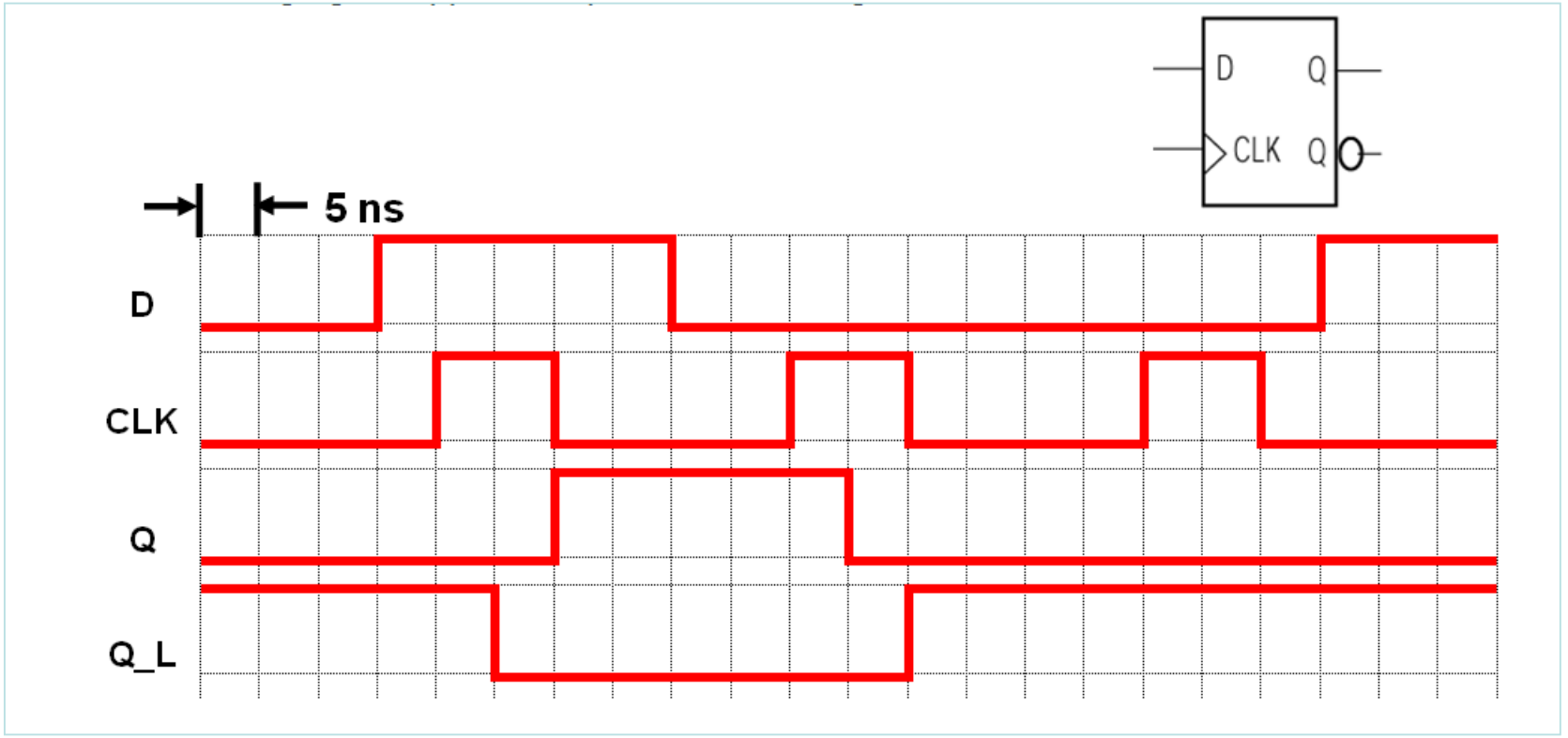
E. none of the above



Q2. The **nominal setup time** provided for the D flip-flop is:

A. 5 ns B. 10 ns C. 15 ns D. 20 ns

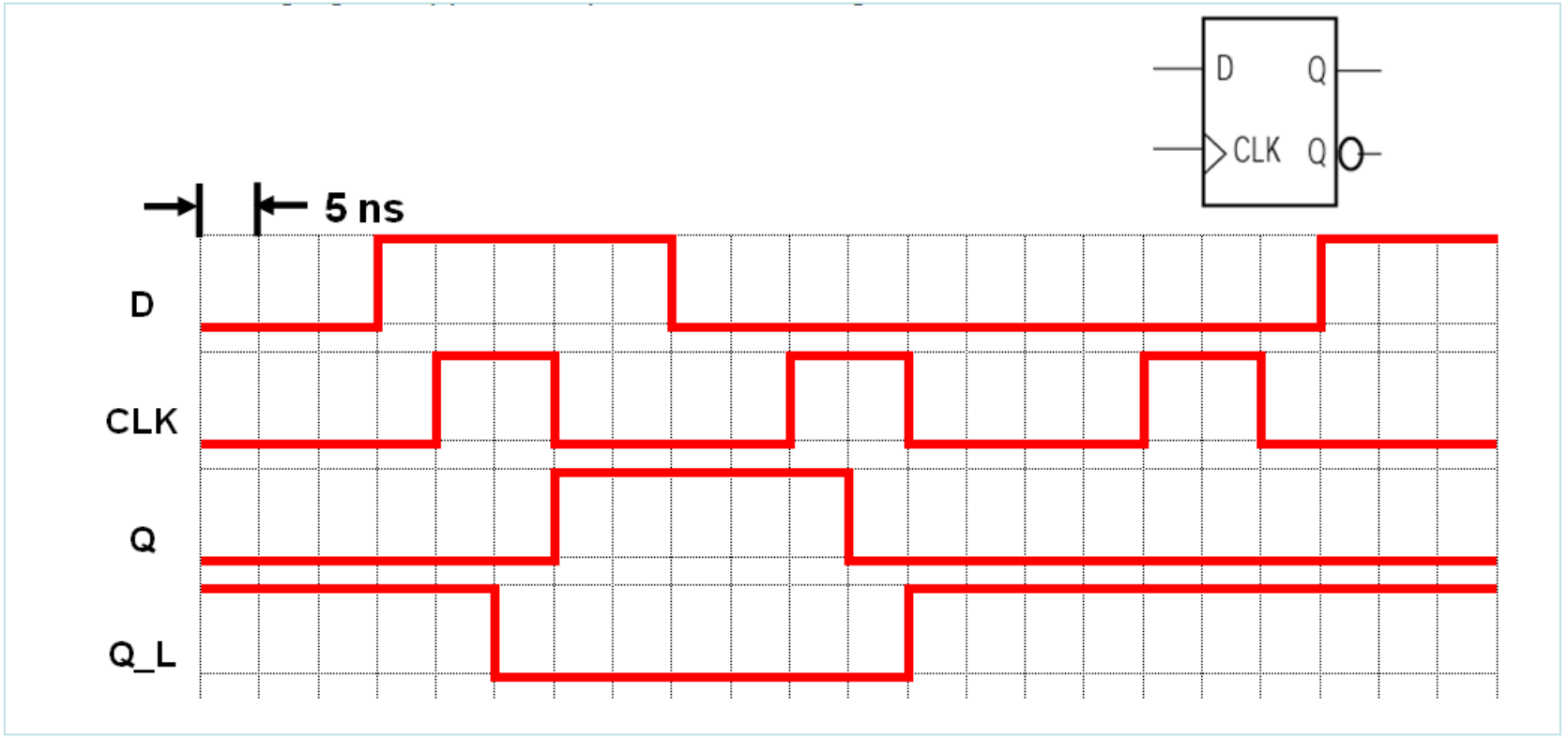
E. none of the above



Q3. The **nominal hold time** provided for the D flip-flop is:

A. 5 ns B. 10 ns C. 15 ns D. 20 ns

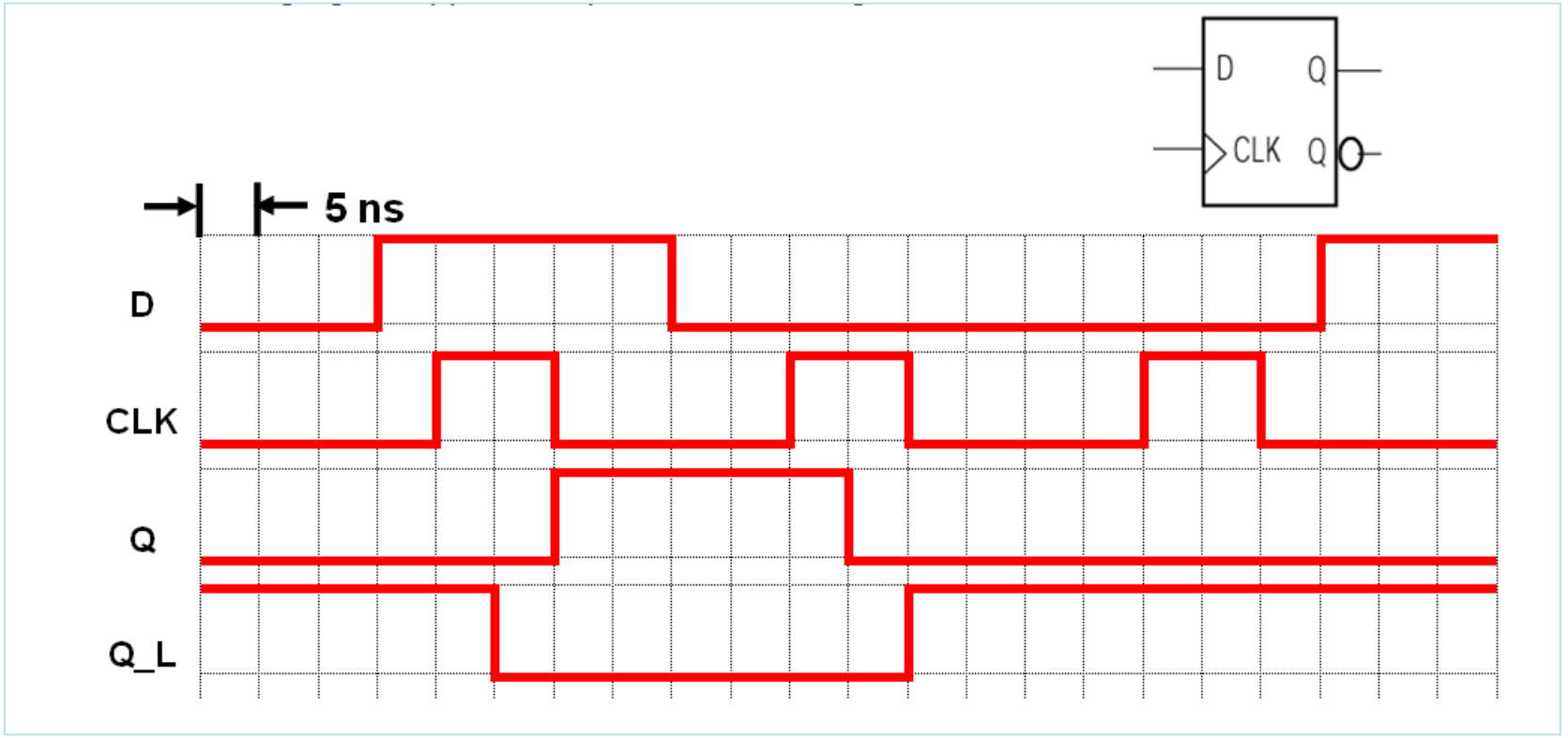
E. none of the above



Q4. The **clock pulse width** provided for the D flip-flop is:

A. 5 ns B. 10 ns C. 15 ns D. 20 ns

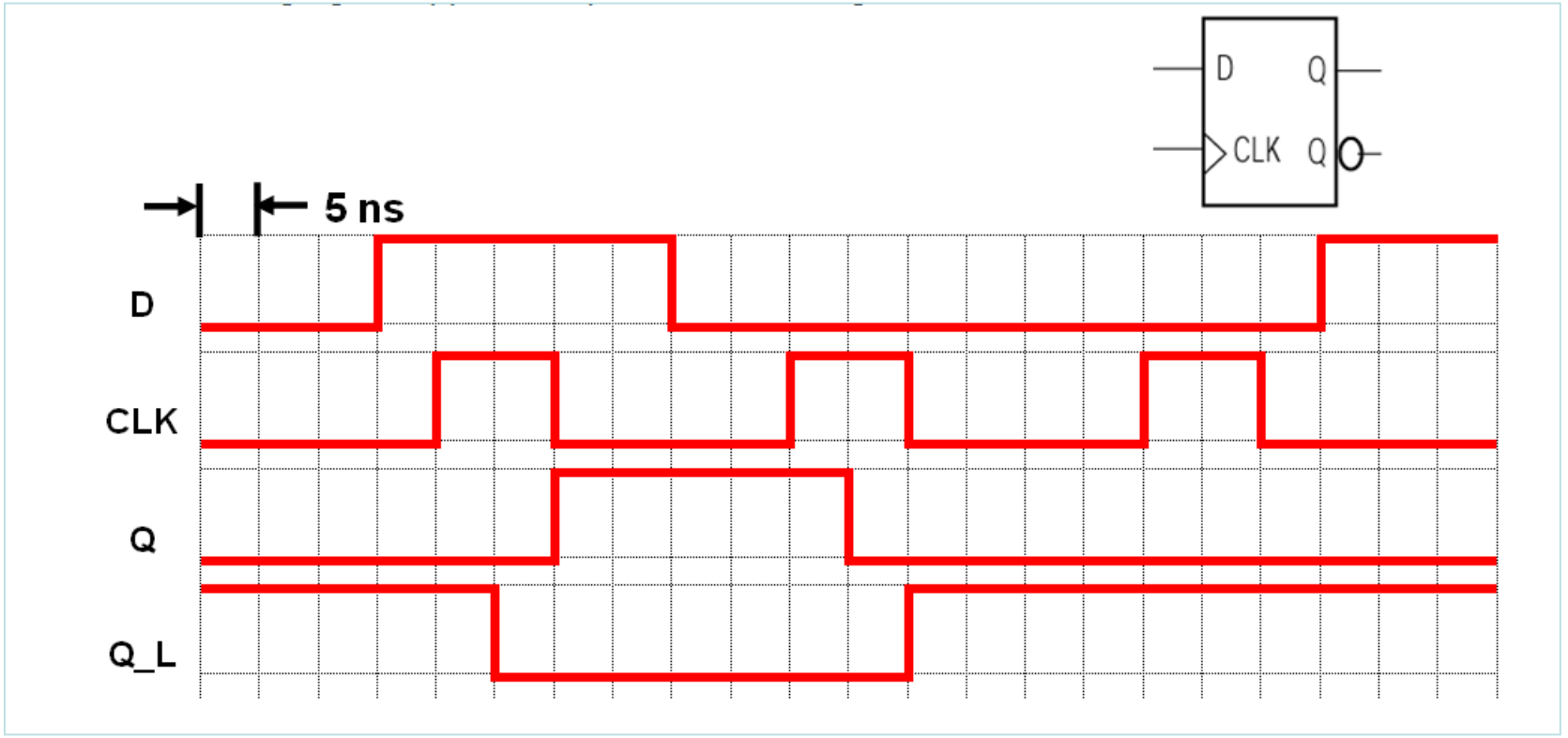
E. none of the above



Q5. The $t_{PLH(C \rightarrow Q)}$ of the D flip-flop is:

A. 5 ns B. 10 ns C. 15 ns D. 20 ns

E. none of the above



Q6. The $t_{PHL(C \rightarrow Q)}$ of the D flip-flop is:

A. 5 ns B. 10 ns C. 15 ns D. 20 ns

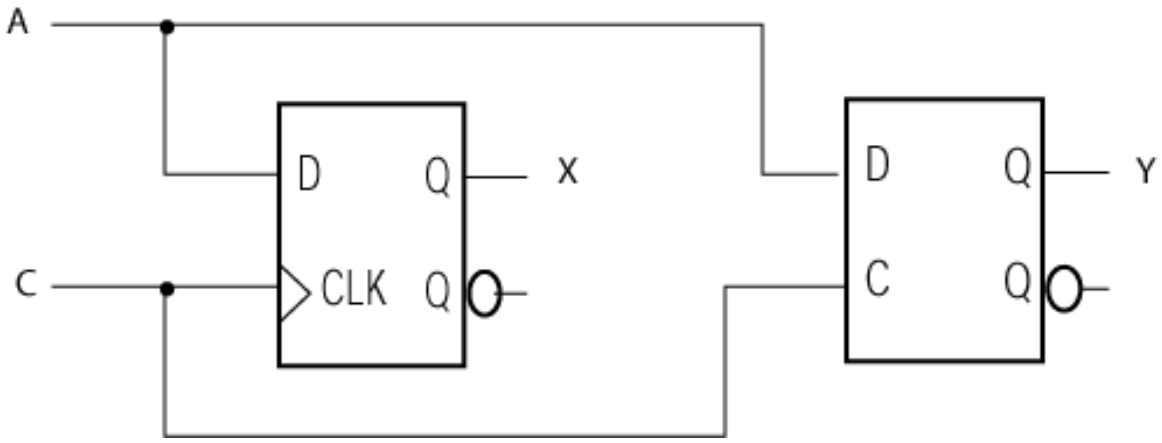
E. none of the above

Q7. **Metastable behavior** of an edge-triggered D flip-flop can be caused by:

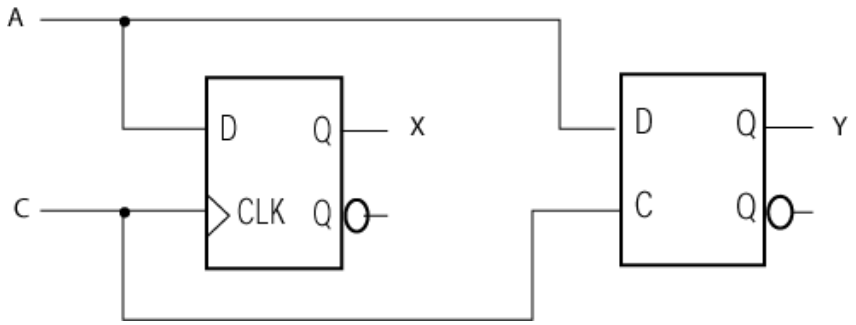
A. violating its minimum setup time requirement
B. violating its minimum hold time requirement
C. violating its minimum clock pulse width requirement
D. all of the above
E. none of the above

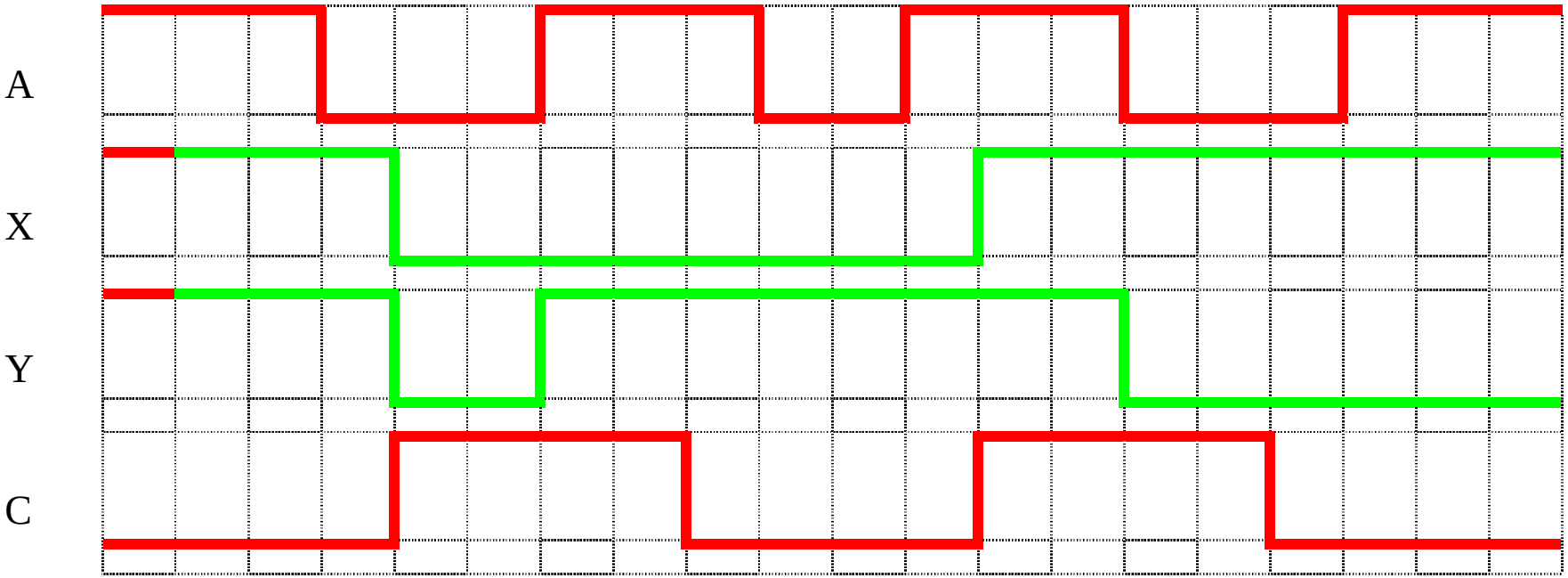
Example – Response of Latch vs. Flip-Flop

Assume a positive edge-triggered D flip-flop (X) and a transparent D latch (Y) are supplied the signals given on the timing chart (next slide).
Plot the response of each, noting the initial states.
Assume the *propagation delays* of the flip-flop and latch are *negligible* relative to the period of "C".



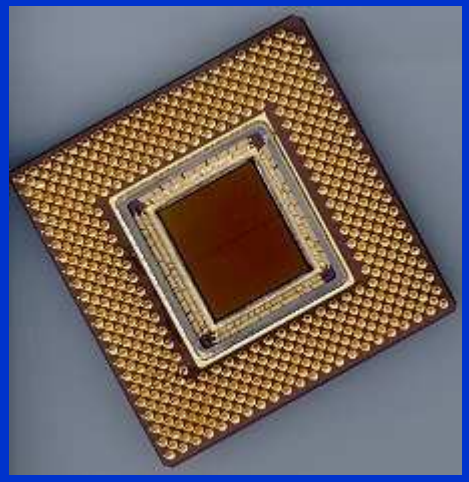
Example – Response of Latch vs. Flip-Flop





Summary

- **Latches and flip-flops are the basic building blocks of virtually all sequential circuits**
  - **a *latch* is a sequential device that watches all of its inputs *continuously* and changes its outputs *at any time* (independent of a clocking signal)**
  - **a *flip-flop* is a sequential device that samples its inputs and *changes its outputs* only at times determined by a *clocking signal***
- **Because the *functional behavior* of latches and flip flops is quite *different*, it is important to know which type is being used in a design**



Introduction to Digital System Design

Module 3-D

Clocked Synchronous State Machine Structure and Analysis

Reading Assignment:

DDPP 4th Ed., pp. 542-553

Learning Objectives:

- **Identify the key elements of a clocked synchronous state machine: next state logic, state memory (flip-flops), and output logic**
- **Differentiate between Mealy and Moore model state machines, and draw a block diagram of each**
- **Analyze a clocked synchronous state machine realized as either a Mealy or Moore model**

Outline

- Overview
- State machine structure
  - Moore machine
  - Mealy machine
- State machine analysis
  - Moore machine analysis
  - Mealy machine analysis

Overview

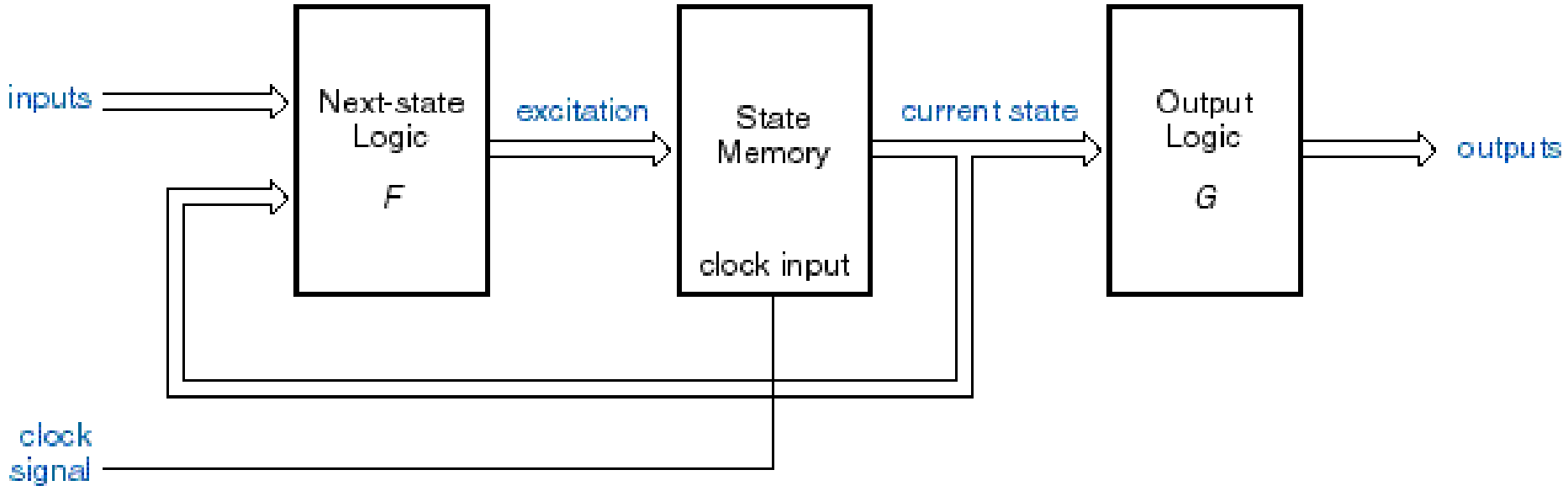
- **"State machine" (or "finite state machine") is a generic name given to sequential circuits**
- **"Clocked" indicates that the flip-flops employ a CLOCK (CLK) input**
- **"Synchronous" means that all the flip-flops in the state machine use the same CLOCK signal**
- **"Analysis" means to analyze the behavior of a given state machine**
  - **construct a PS-NS table**
  - **derive PS-NS equations**
  - **draw a state transition diagram**
  - **draw a timing chart**

State Machine Structure

- **Clocked synchronous state machines consist of three basic blocks:**
  - ***next state logic*** **– combinational circuitry that provides the "excitation" necessary to transition to the next state, based on the current state and the present inputs**
  - ***state memory (flip flops)*** **– set of N flip-flops that store the current state of the machine (providing 2^N distinct states)**
  - ***output logic*** **– combinational circuitry that uses the current state (and possibly current inputs) to determine the outputs generated**

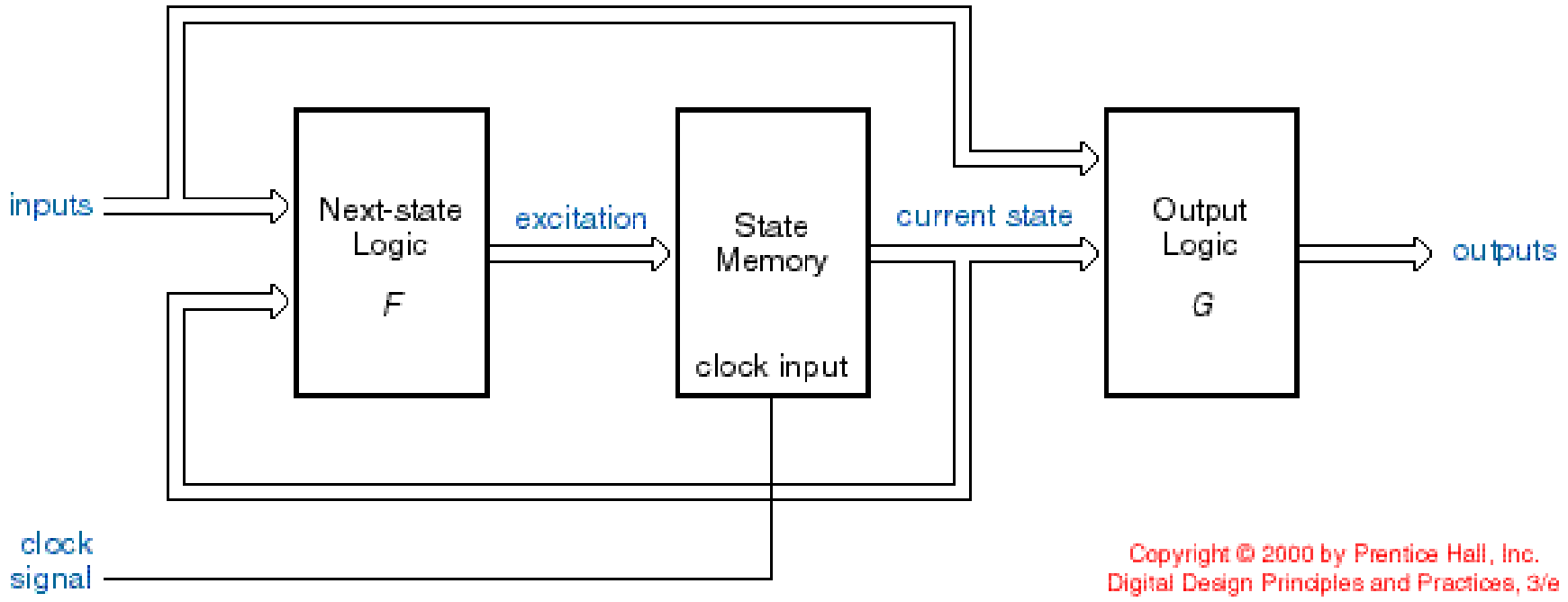
Moore Machine

- **In a *Moore* machine, the outputs are only a function of the current state**



Mealy Machine

- **In a *Mealy* machine, the outputs are a function of the current state as well as the current inputs**



State Machine Structure

- **With appropriate circuit or drawing manipulations, one state machine model can be mapped into another**
- **The exact classification of a state machine into one style or another is ultimately not very important**
- **What *is* important is how the structure chosen satisfies your design requirements**

Characteristic Equations (Review)

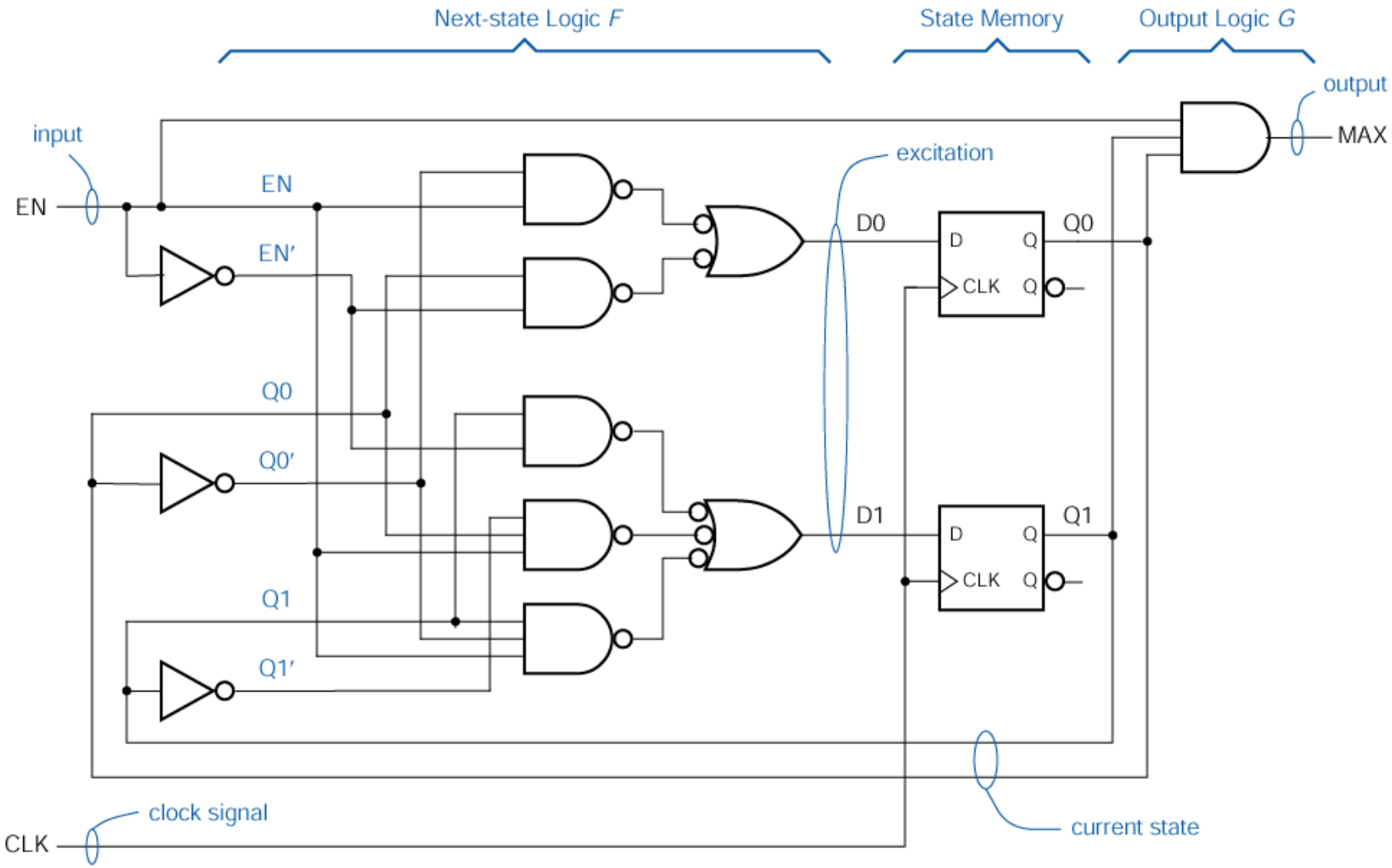
- The characteristic equations of the various flip-flops described previously are:
  - S-R: $Q^* = S + R' \cdot Q$
  - D: $Q^* = D$
  - T: $Q^* = Q \oplus T$
- We will use these characteristic equations as the basis for analyzing state machines
- Analysis in this context means ***writing the next state equations*** that describe the circuit's behavior

State Machine Analysis

- **The analysis of a clocked synchronous state machine has four basic steps:**
  - **Determine the *next state* and the *output* functions based on the circuit diagram**
  - **Use the next state and output functions to construct a *present state - next state / output table (PS-NS / O)***
  - **Draw a *state transition diagram* that presents the information tabulated in the present state - next state / output table in graphical form**
  - **Draw a *timing diagram* that shows the timing relationship between the input, output, and clocking signals**

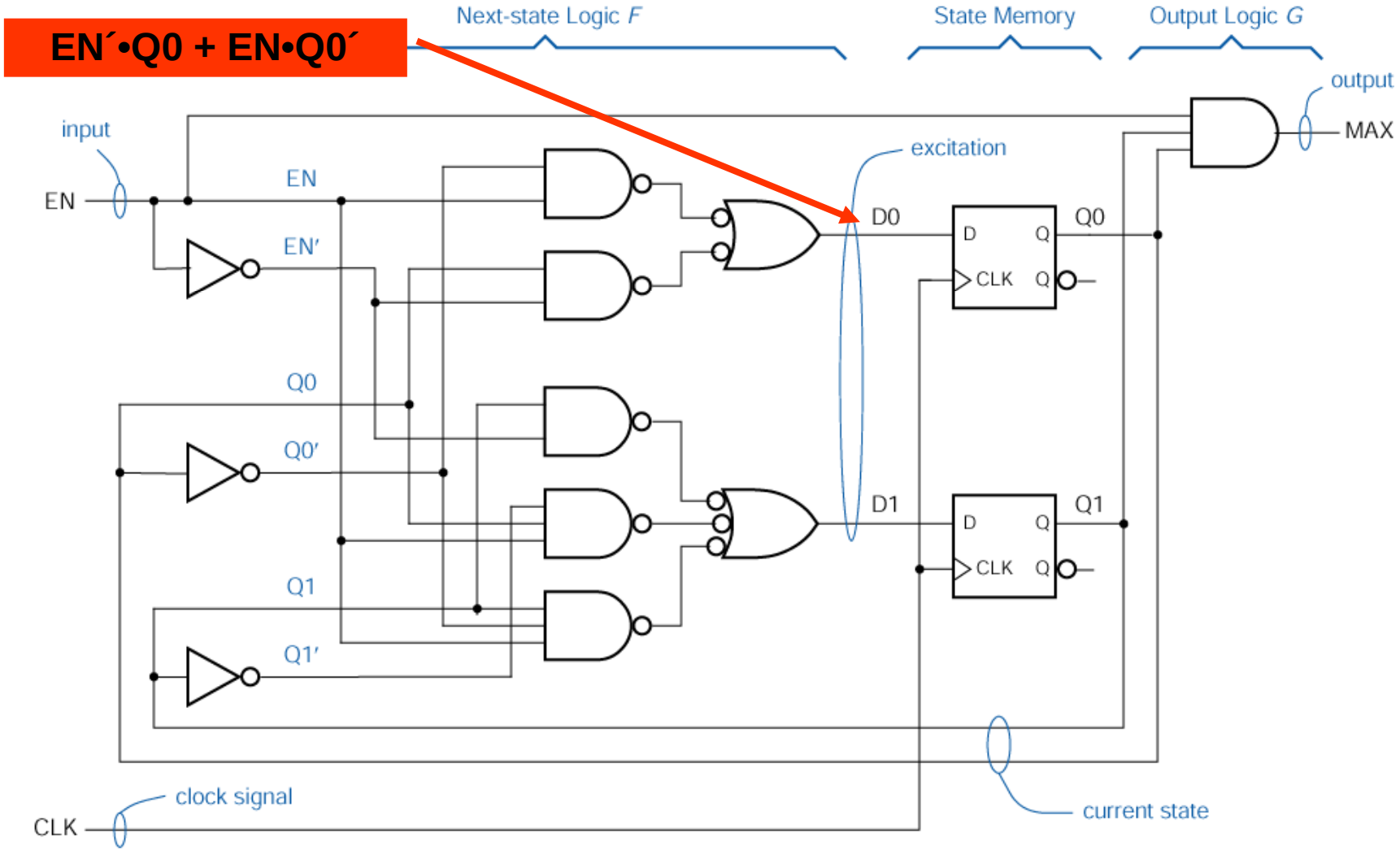
Exercise 1

- **Analyze the following Mealy state machine:**



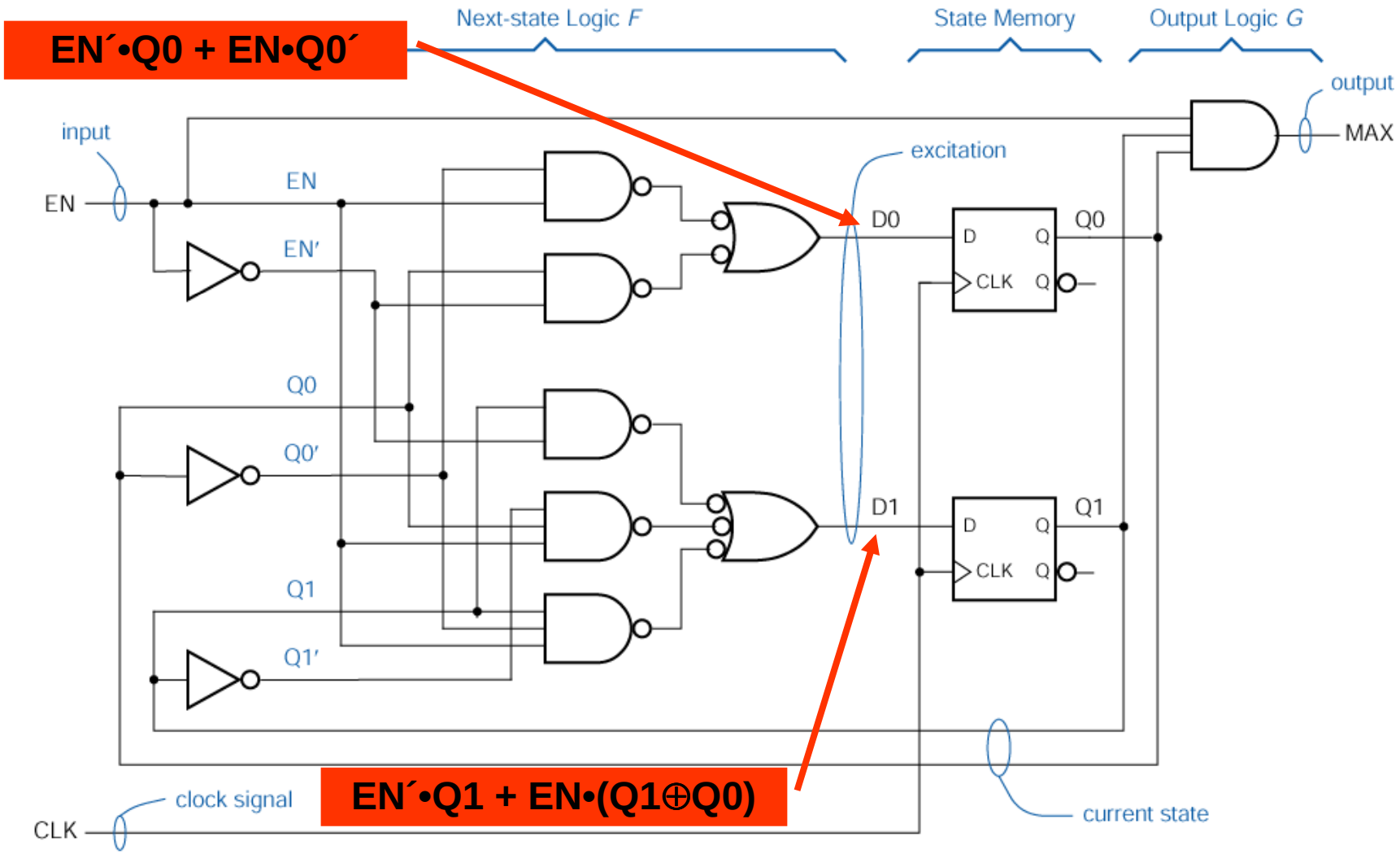
Exercise 1

- **Analyze the following Mealy state machine:**



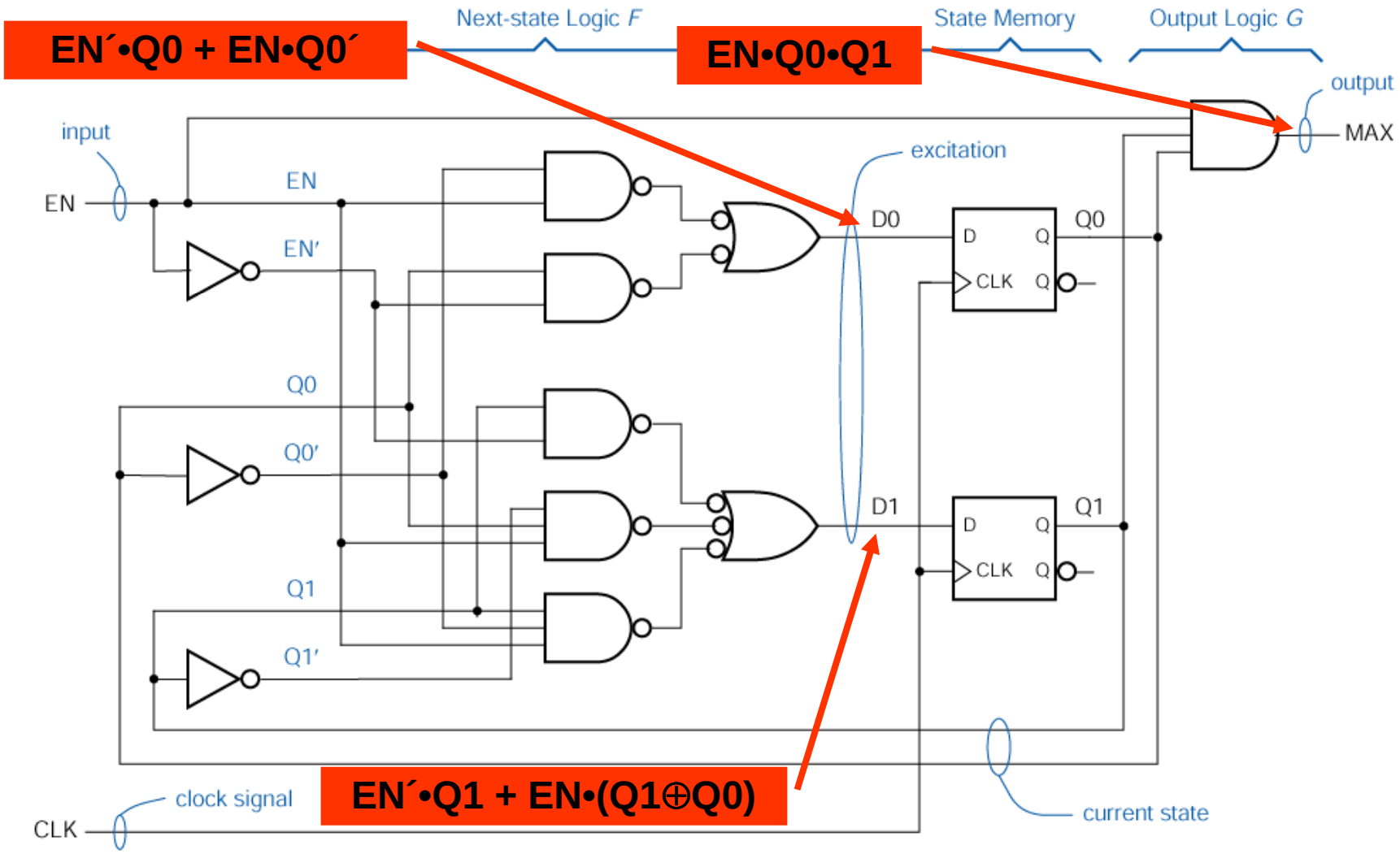
Exercise 1

- **Analyze the following Mealy state machine:**



Exercise 1

- **Analyze the following Mealy state machine:**



Exercise 1

- **STEP 1: Write the next state equations for each D flip-flop and the output logic function**

Q0* = EN´•Q0 + EN•Q0´ = EN ⊕ Q0

Q1* = EN´•Q1 + EN•(Q1 ⊕ Q0)

MAX = EN•Q0•Q1

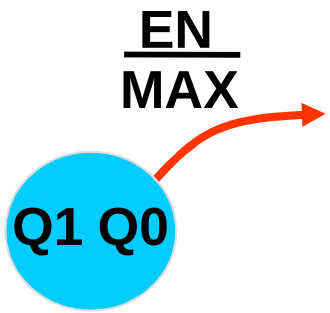
Exercise 1

- **STEP 2: Construct a PS-NS / O table**

PS		PI	NS		Output
Q1	Q0	EN	Q1*	Q0*	MAX
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	0	0	1

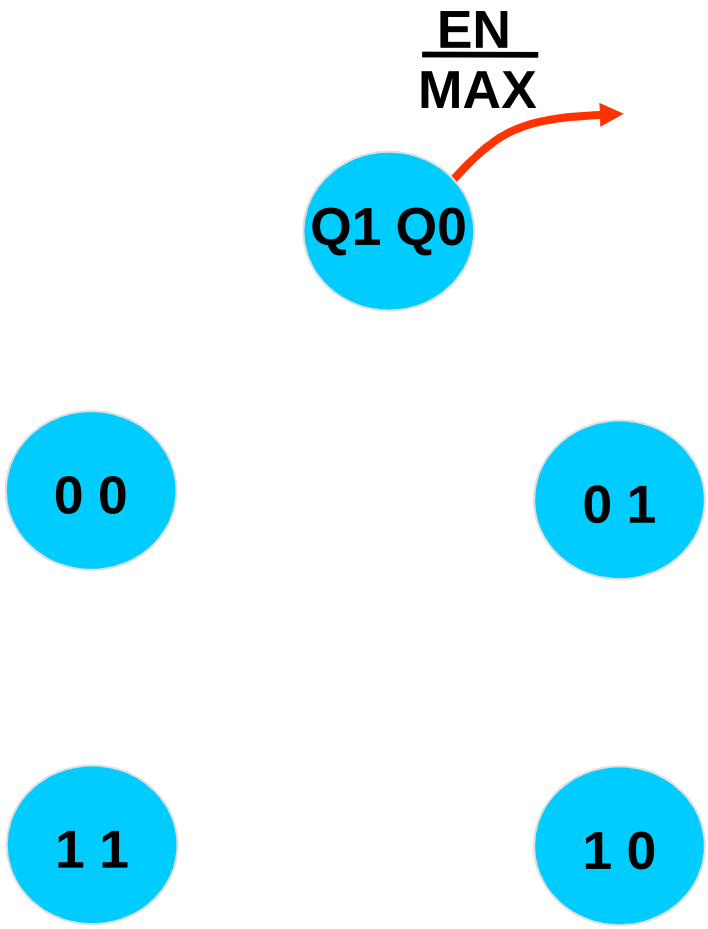
Exercise 1

- **STEP 3: Construct a Mealy state transition diagram**



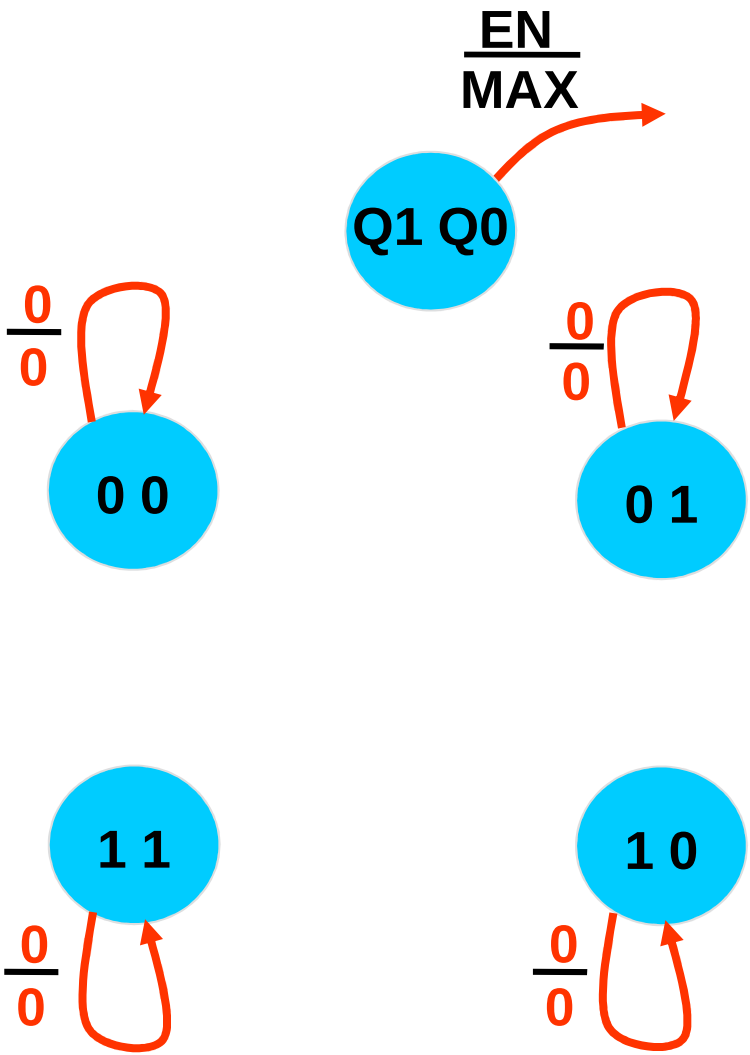
Exercise 1

- **STEP 3: Construct a Mealy state transition diagram**



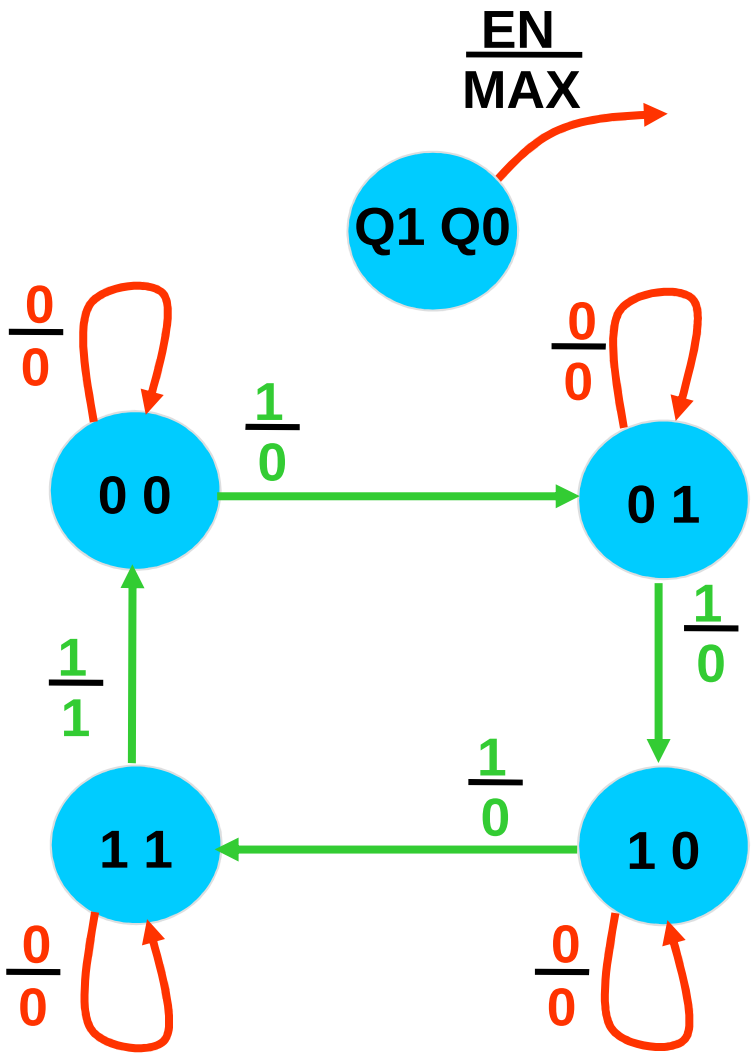
Exercise 1

- **STEP 3: Construct a Mealy state transition diagram**



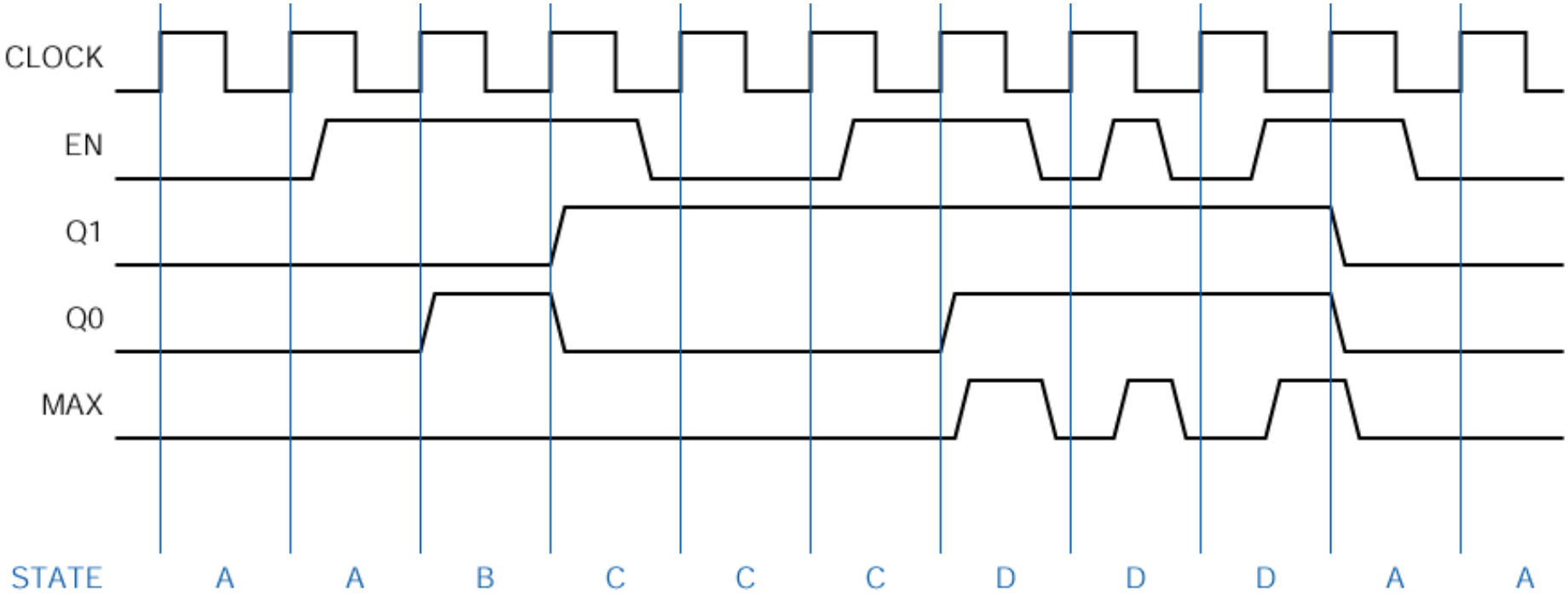
Exercise 1

- **STEP 3: Construct a Mealy state transition diagram**



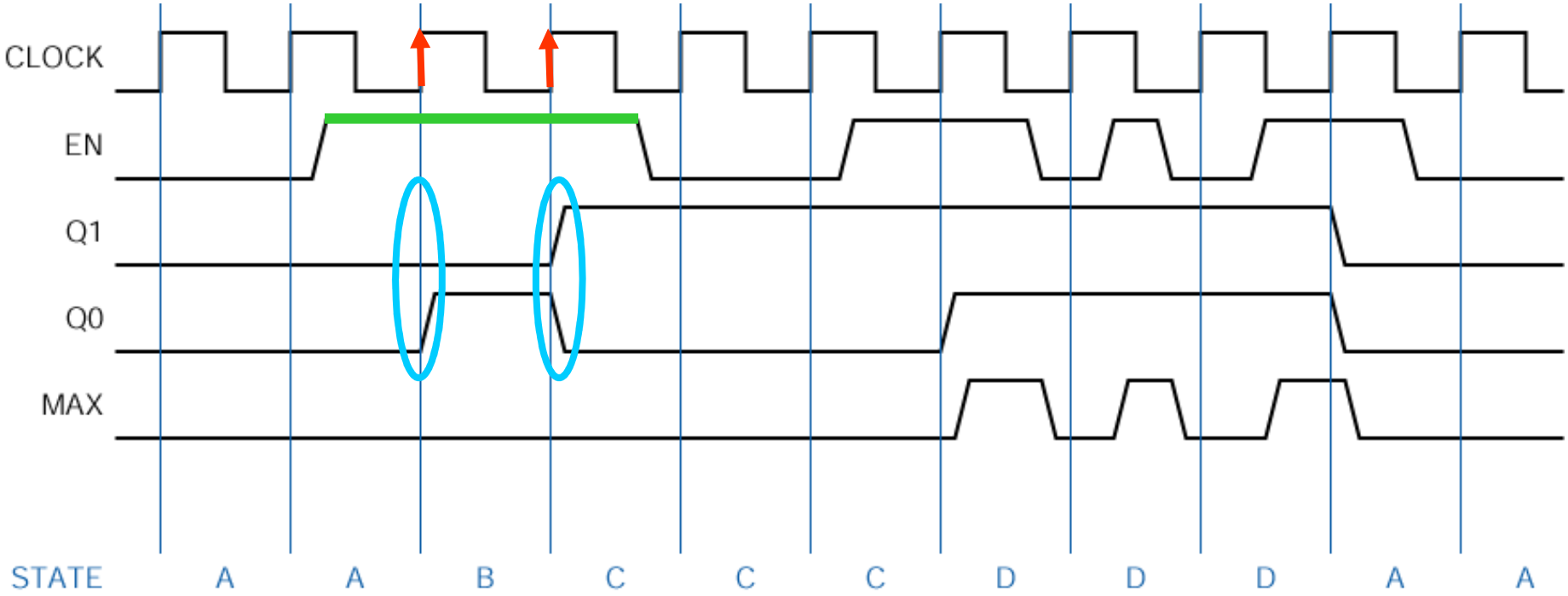
Exercise 1

- **STEP 4: Draw a timing chart**



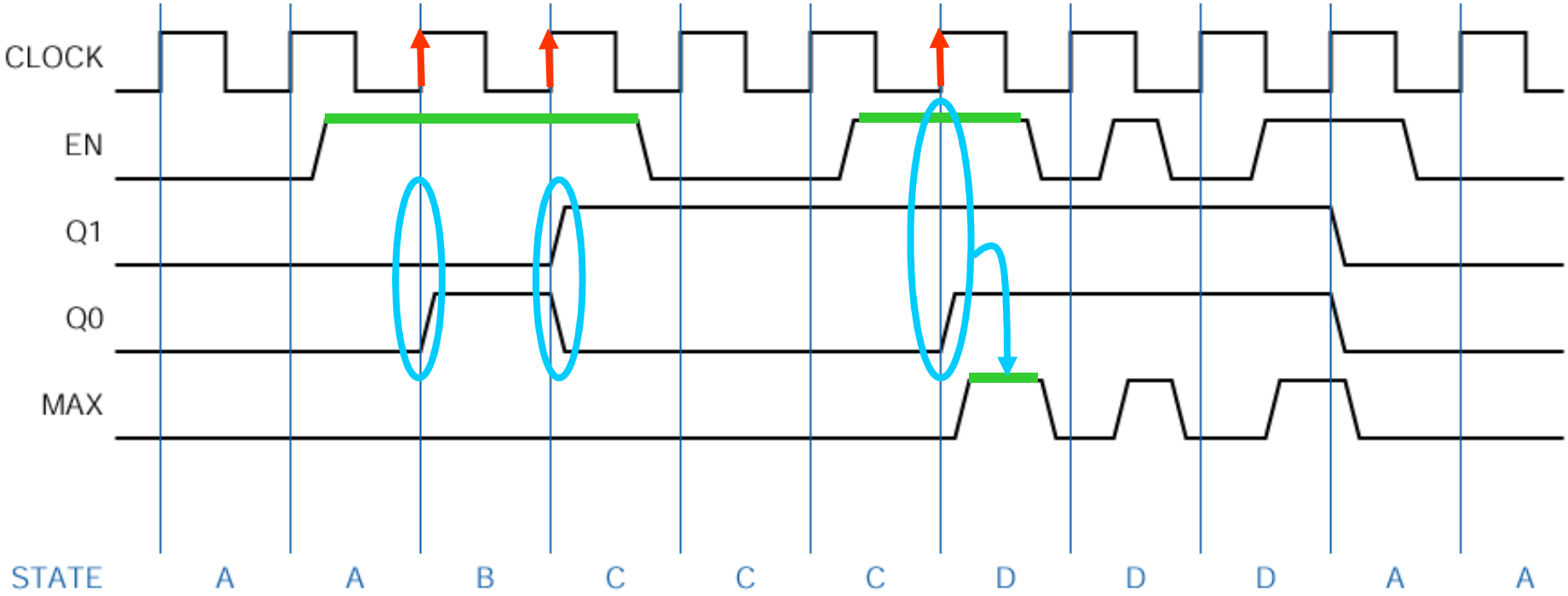
Exercise 1

- **STEP 4: Draw a timing chart**



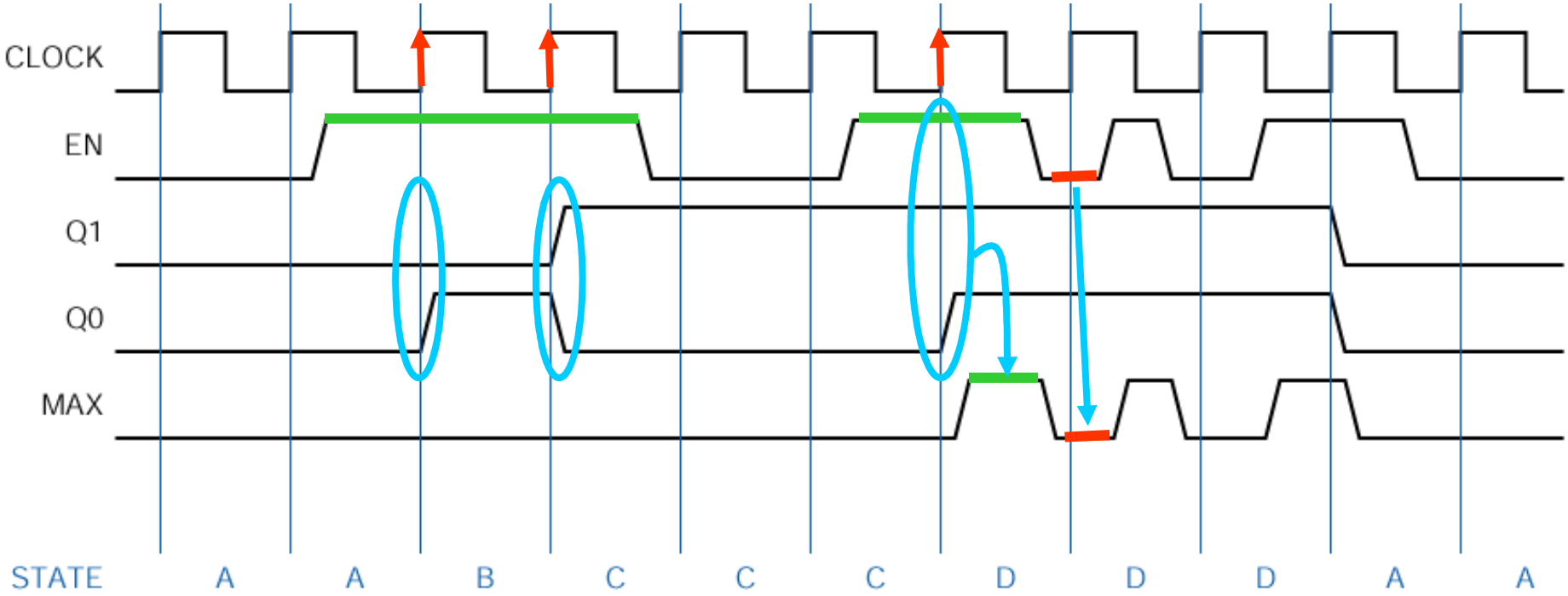
Exercise 1

- **STEP 4: Draw a timing chart**



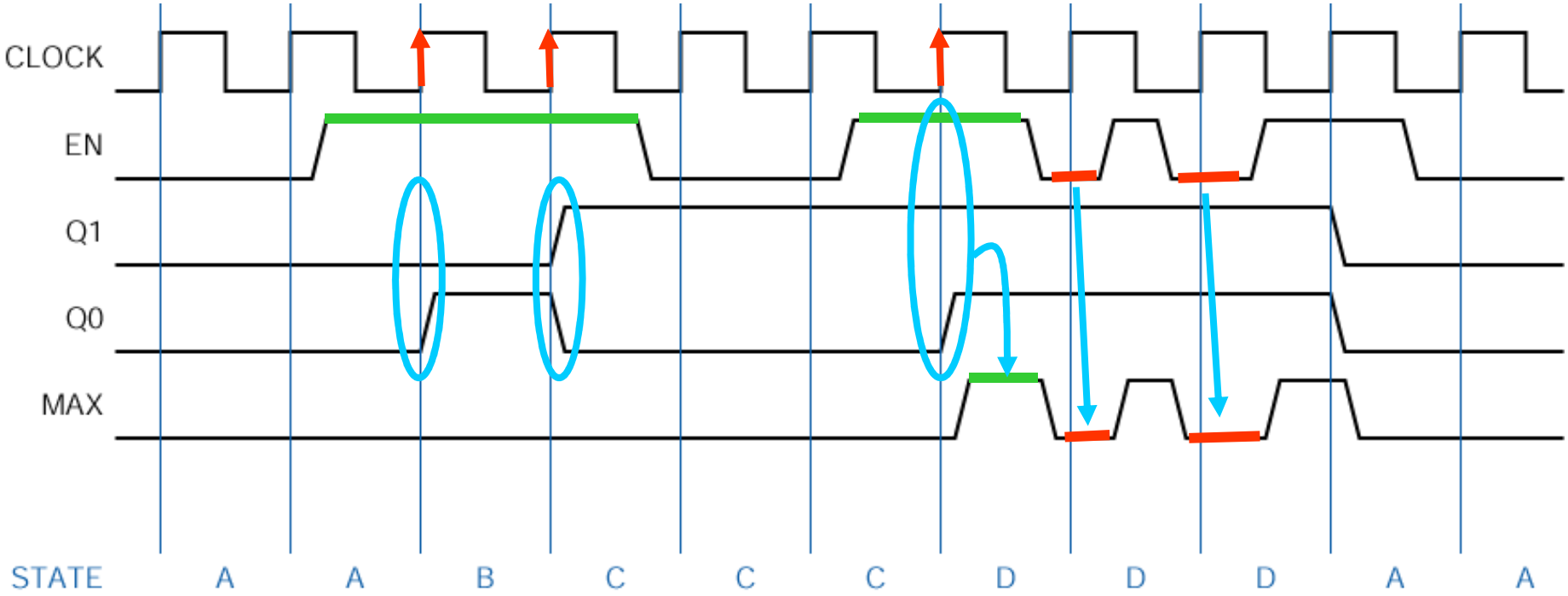
Exercise 1

- **STEP 4: Draw a timing chart**



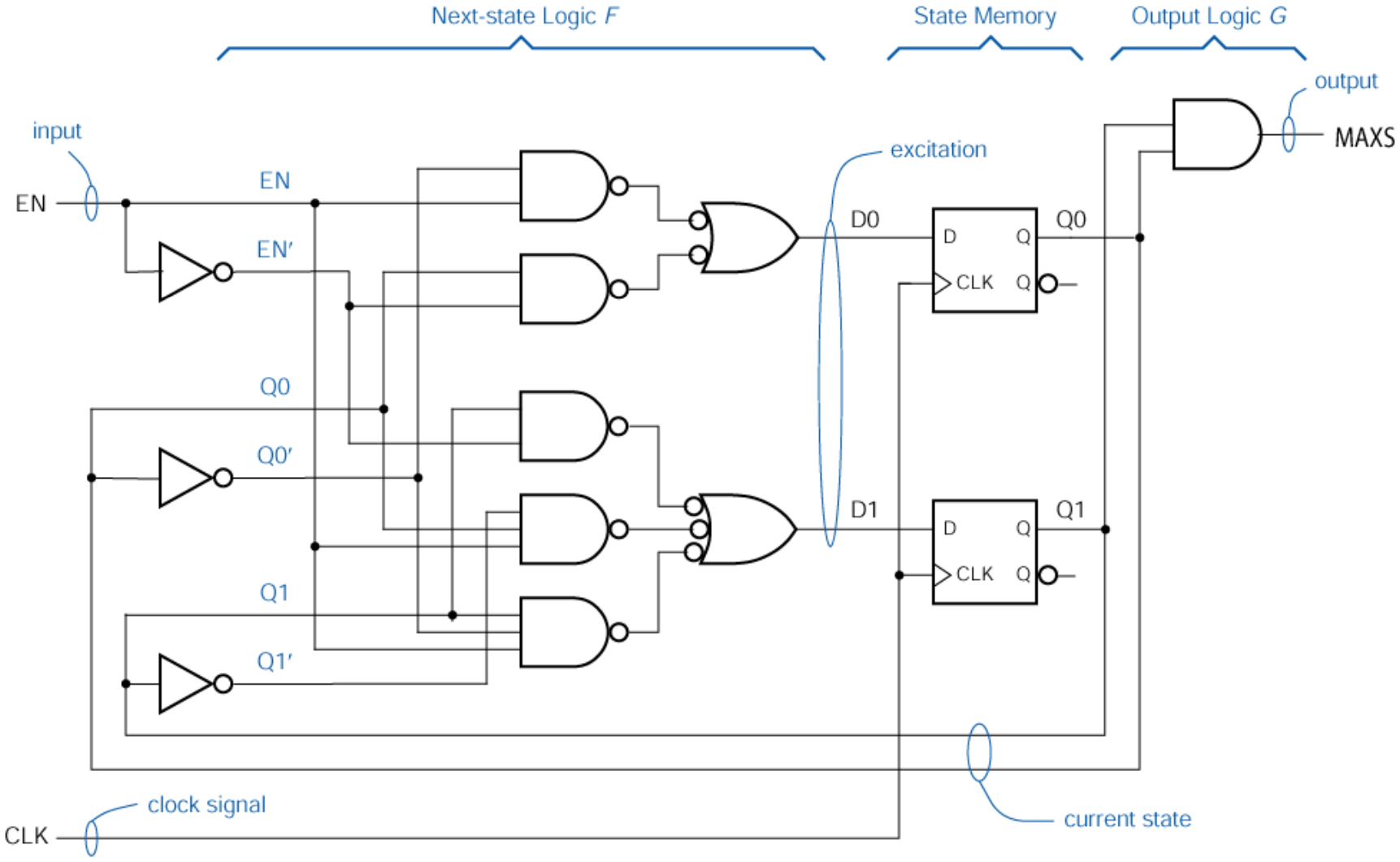
Exercise 1

- **STEP 4: Draw a timing chart**



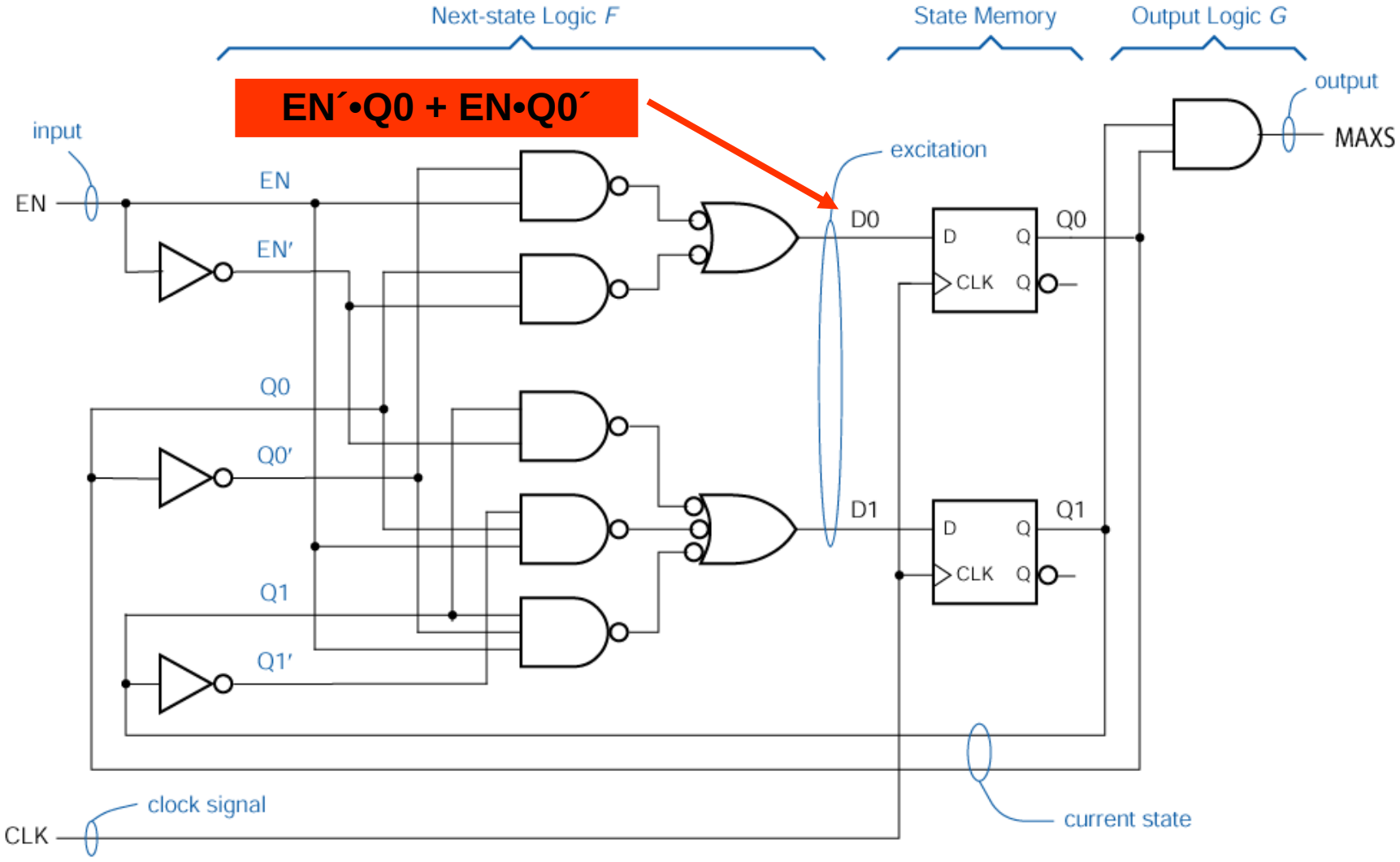
Exercise 2

- **Analyze the following Moore state machine:**



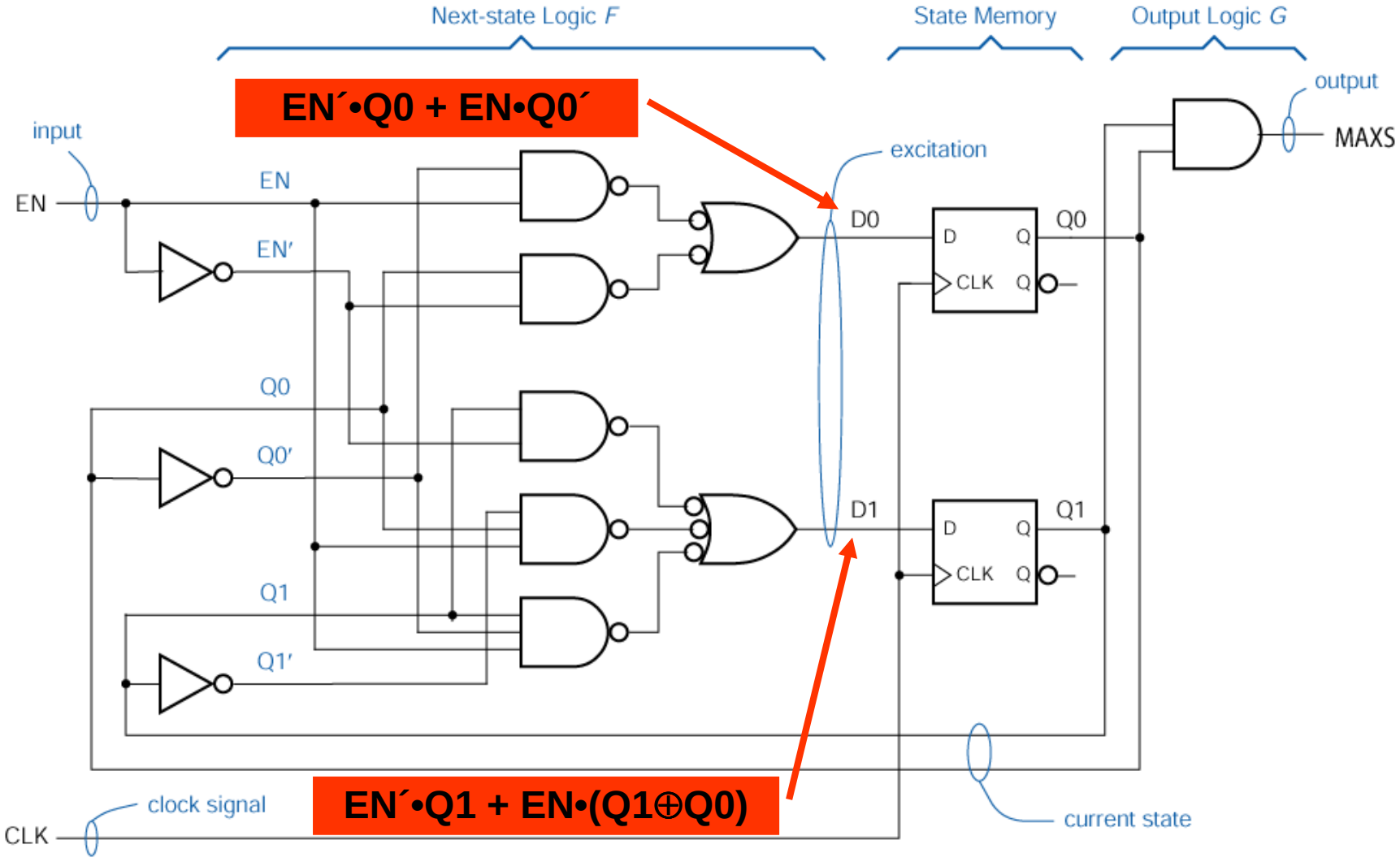
Exercise 2

- **Analyze the following Moore state machine:**



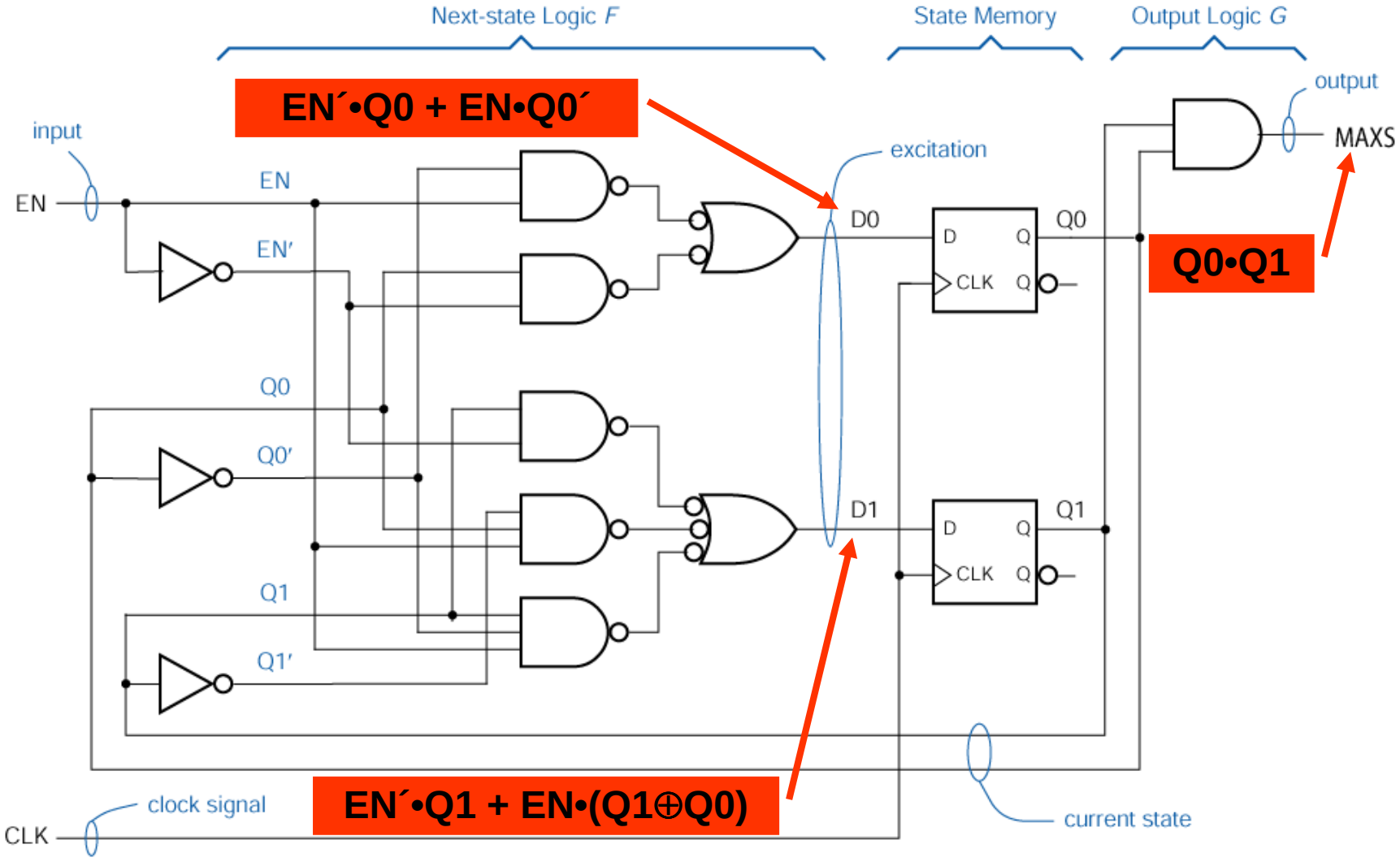
Exercise 2

- **Analyze the following Moore state machine:**



Exercise 2

- **Analyze the following Moore state machine:**



Exercise 2

- **STEP 1: Write the next state equations for each D flip-flop and the output logic function**

Q0* = EN´•Q0 + EN•Q0´ = EN ⊕ Q0

Q1* = EN´•Q1 + EN•(Q1 ⊕ Q0)

MAXS = Q0•Q1

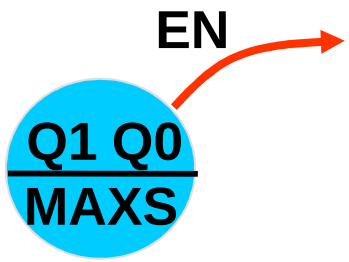
Exercise 2

- **STEP 2: Construct a PS-NS / O table**

PS		PI	NS		Output
Q1	Q0	EN	Q1*	Q0*	MAXS
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

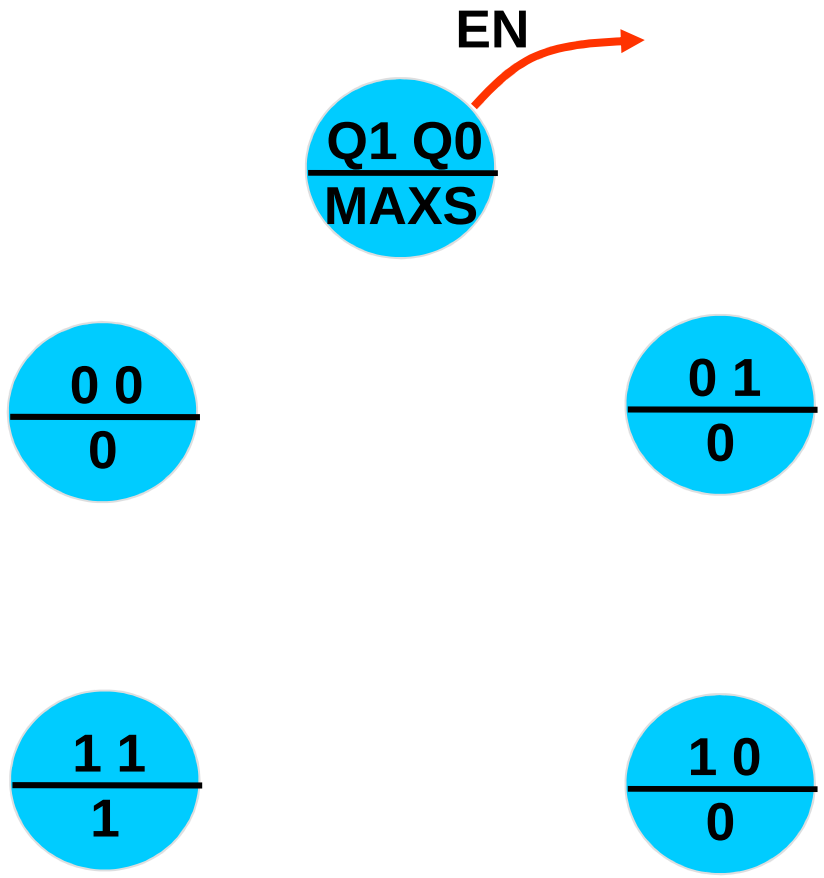
Exercise 2

- **STEP 3: Construct a Moore state transition diagram**



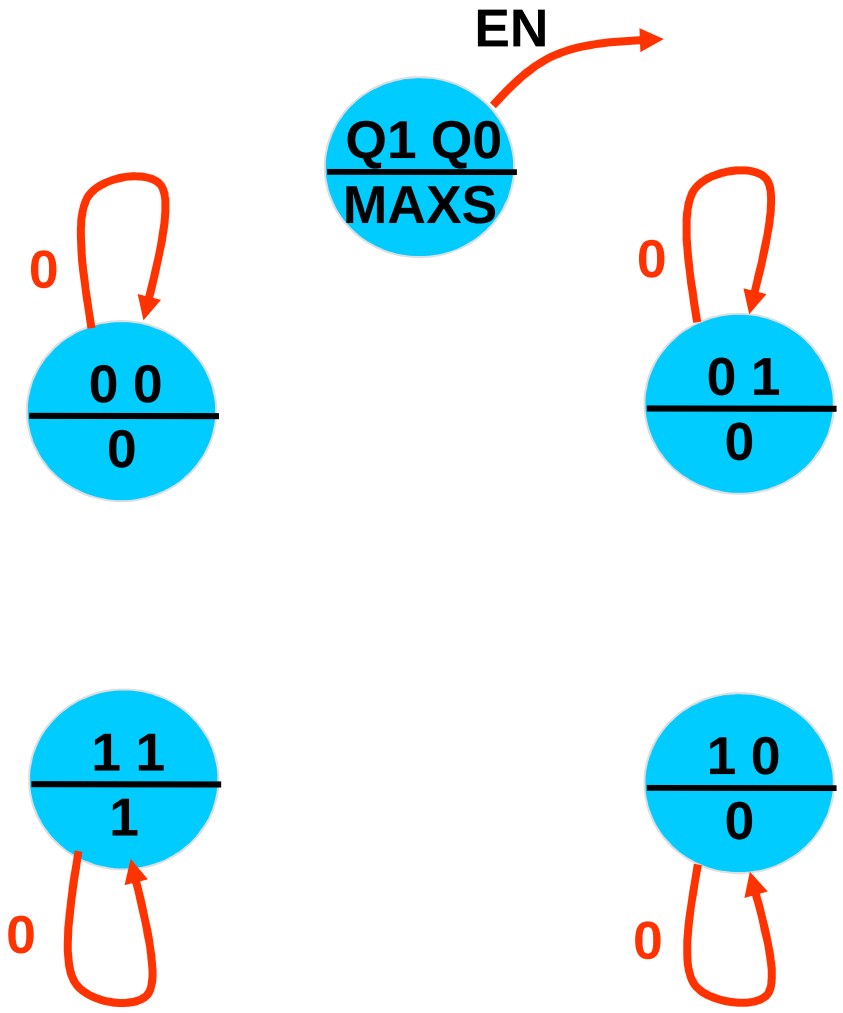
Exercise 2

- **STEP 3: Construct a Moore state transition diagram**



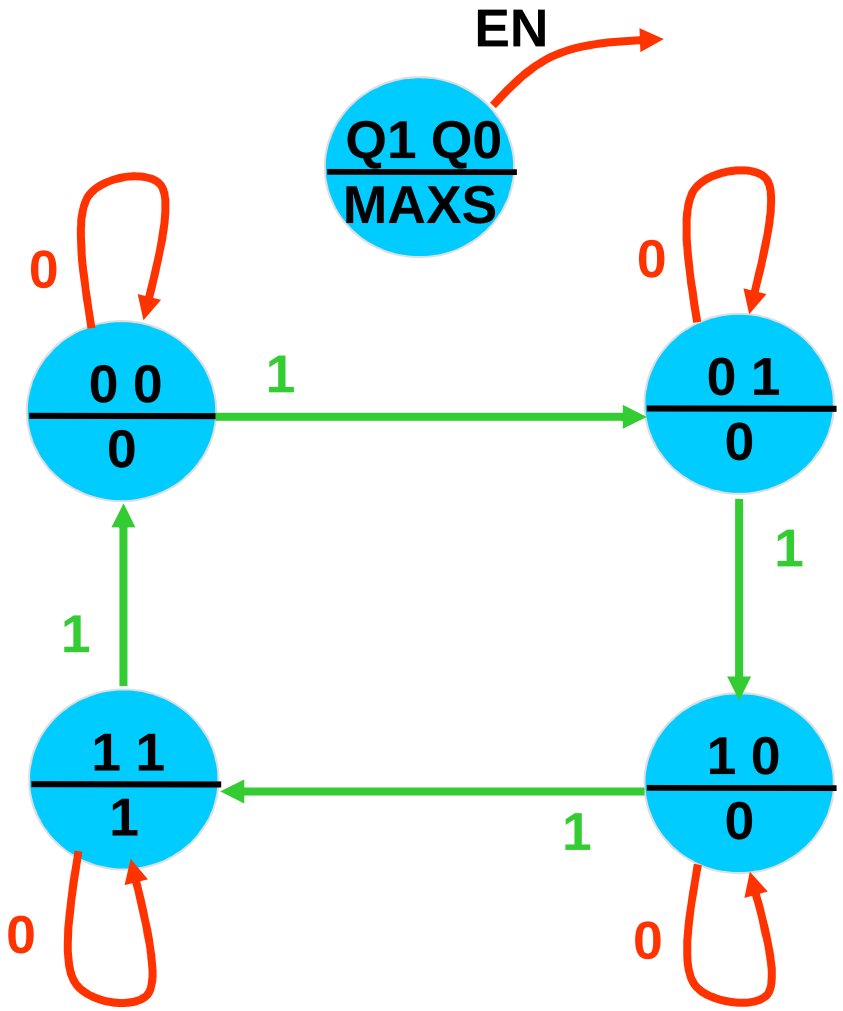
Exercise 2

- **STEP 3: Construct a Moore state transition diagram**



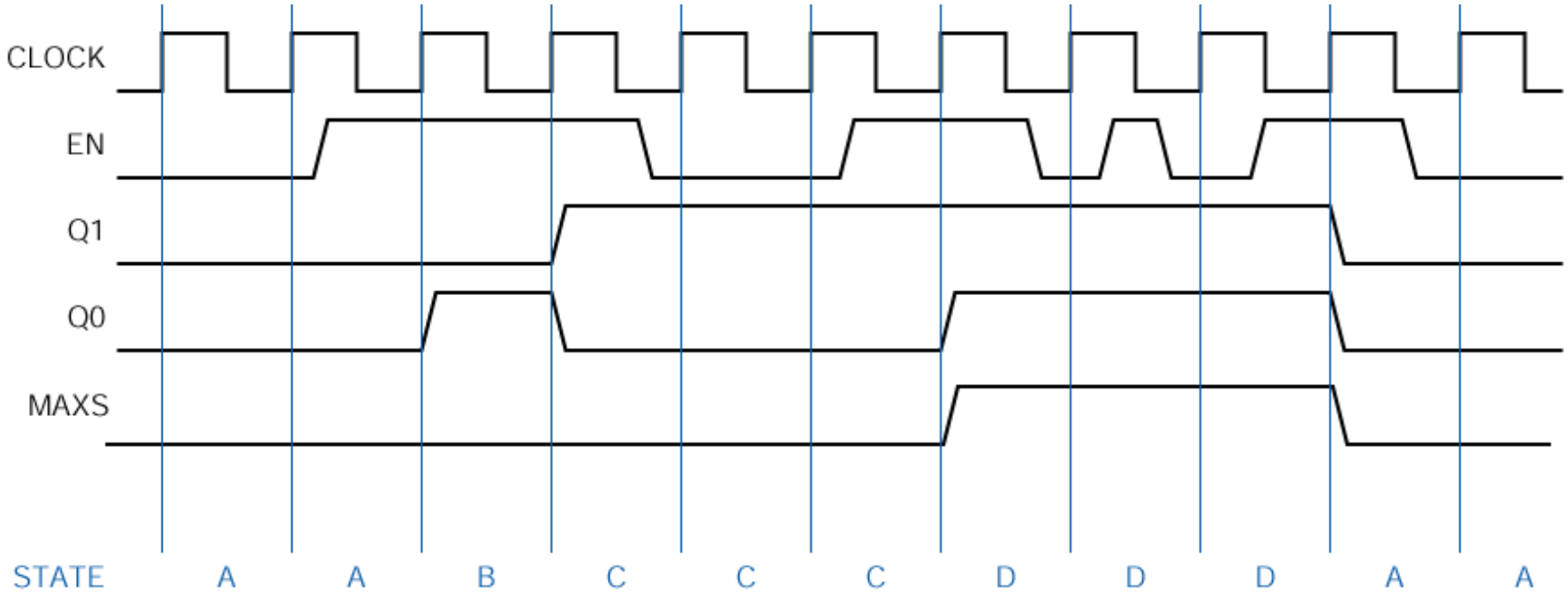
Exercise 2

- **STEP 3: Construct a Moore state transition diagram**



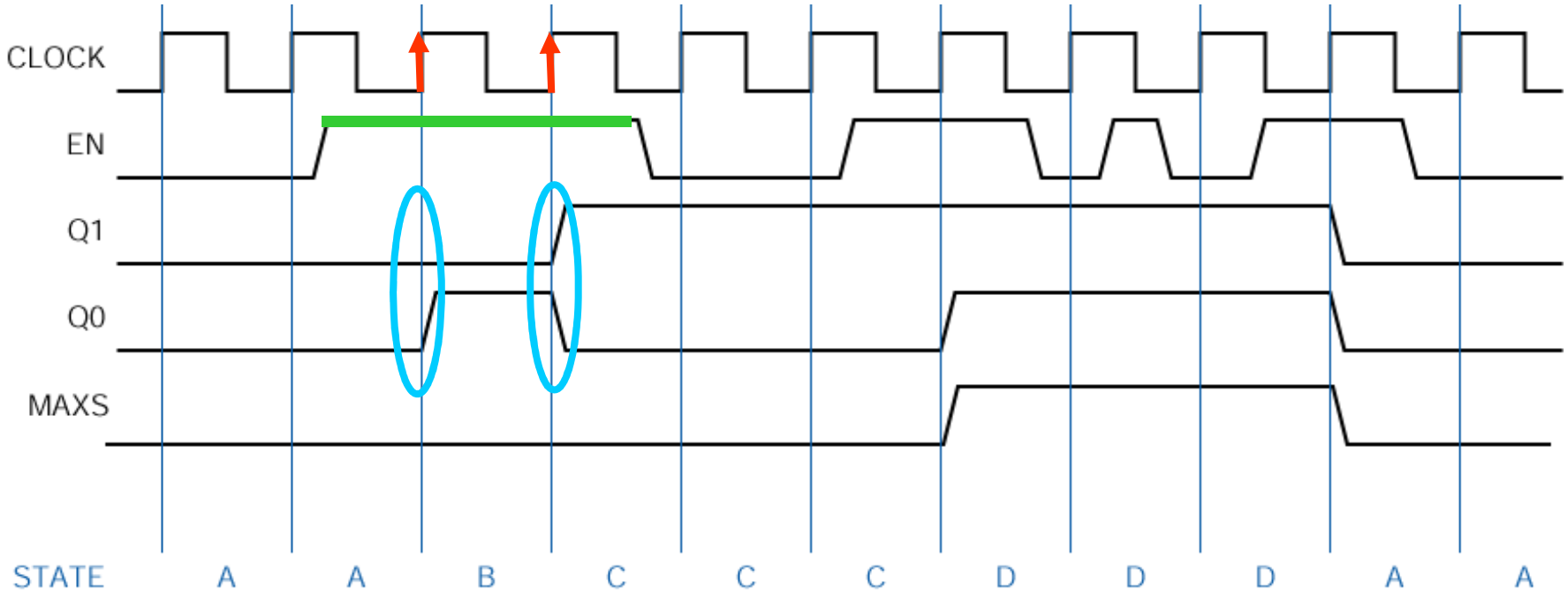
Exercise 2

- **STEP 4: Draw a timing chart**



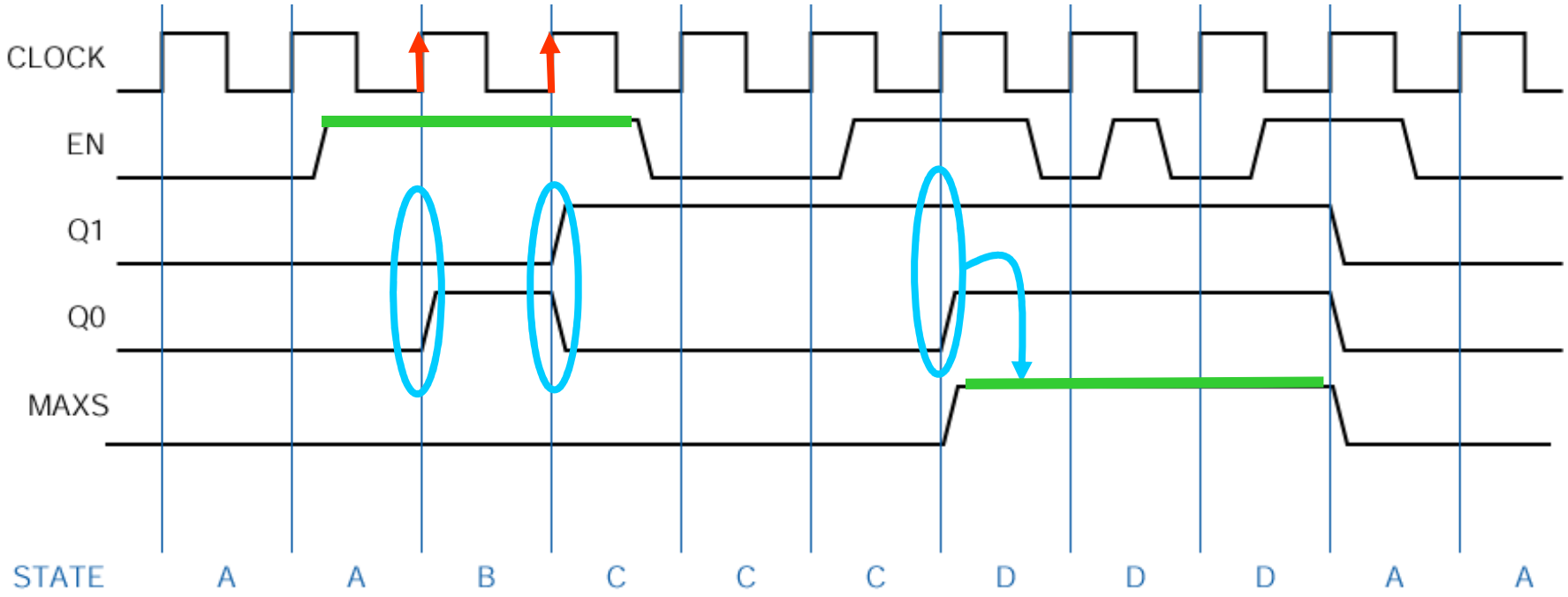
Exercise 2

- **STEP 4: Draw a timing chart**



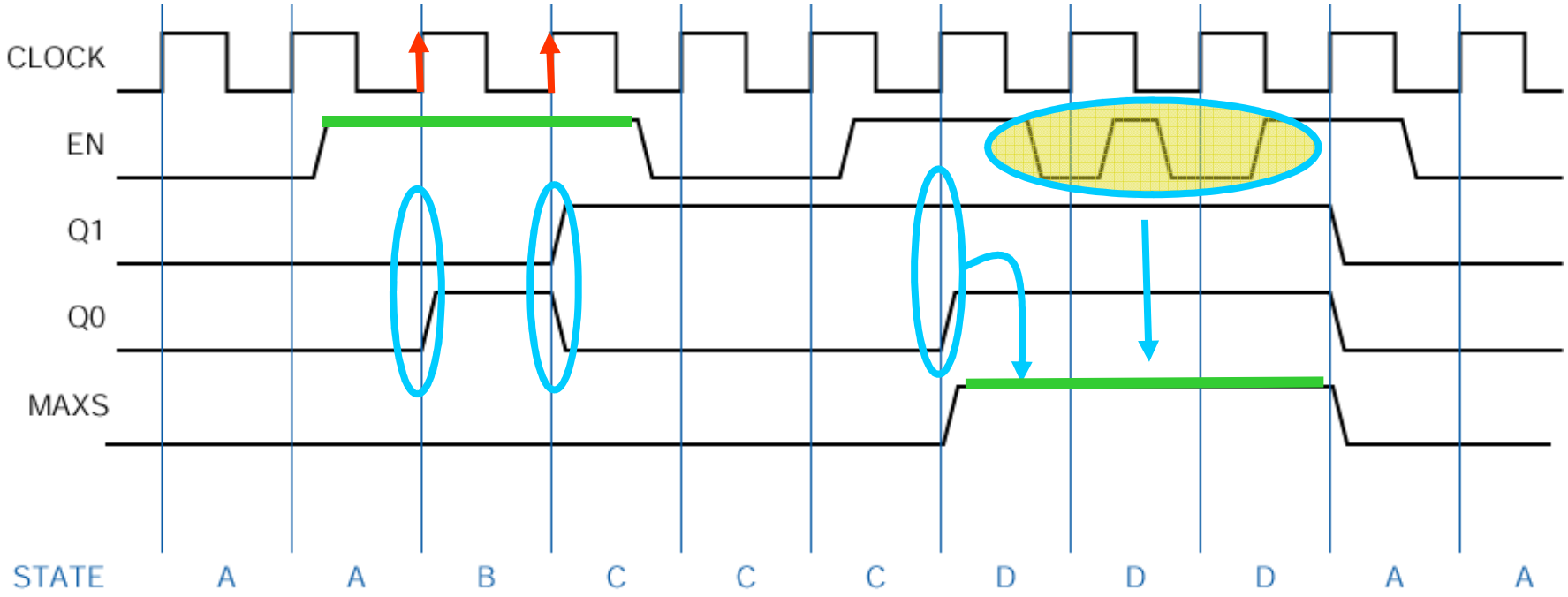
Exercise 2

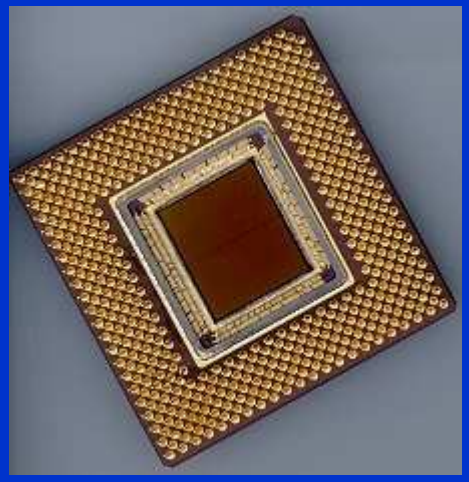
- **STEP 4: Draw a timing chart**



Exercise 2

- **STEP 4: Draw a timing chart**





Introduction to Digital System Design

Module 3-E

Clocked Synchronous State Machine Synthesis

Reading Assignment:

DDPP 4th Ed., pp. 553-566, 612-625, 682-689

Learning Objectives:

- **Outline the steps required for state machine synthesis**
- **Derive the excitation table for any type of latch or flip-flop**
- **Discuss reasons why formal state-minimization procedures are seldom used by experienced digital system designers**
- **Describe three ways that state machines can be specified in ABEL: using a clocked truth table, using clocked assignment operators, or using a state diagram approach**
- **List the ABEL attribute suffixes that pertain to sequential circuits**
- **Draw a circuit for an oscillator and calculate its frequency of operation**
- **Draw a circuit for a bounce-free switch based on an S-R latch and analyze its behavior**

Outline

- Overview
- State machine design steps
  - Derivation of flip-flop excitation tables
  - Flip-flop choice
- State machines in ABEL
  - Attribute suffixes
  - Macrocell structure
- Clocking considerations
  - Periodic clock generation circuits
  - Timing diagram and specifications
  - Event clock generation circuits

Overview

- **Designing a finite state machine (FSM) is a *creative process* that is, in many ways, like writing a computer program:**
  - **You have a fairly good idea of what the input and output signals should be, but perhaps an *imprecise description* of the desired relationship between them**
  - **During the design you may have to identify and choose among *different ways of doing things* – sometimes using common sense, sometimes arbitrarily**
  - **You may have to identify and handle *special cases* that weren't included in the original description**

Overview

- **Creative process…**
  - **You will probably have to keep track of several ideas in *your head* during the design process**
  - **Since the design process is *not an algorithm*, there's no guarantee that you can complete it using a finite number of states or lines of code**
  - **When you finally run the state machine or program, it will do *exactly* what you told it to do – no more, no less**
  - **There's *no guarantee* the thing will work the first time – you may have to debug and iterate the entire process**

State Machine Design Steps

- **State machine design steps**
  - **Given a word description, construct a *state/output table* or *transition diagram***
  - **Minimize any "obvious" *redundant states* in the translated description**
  - **Choose a set of state variables and assign *binary state-variable combinations* to the named states**
  - **Substitute the *state-variable combinations* into the *state/output table* (and/or *state transition diagram*) to create a table that shows the desired next state-variable combination and output for each state/input combination**

State Machine Design Steps

- **State machine design steps...**
  - **If you haven't done so already, choose a *flip-flop or latch type* for the state memory**
  - **Construct an *excitation table* that shows the excitation values required to obtain the desired next state for each state-input combination**
  - **Derive *excitation equations* from the excitation table**
  - **Derive *output equations* from the transition/output table**
  - **Draw a *logic diagram* (or realize the equations directly in a PLD)**

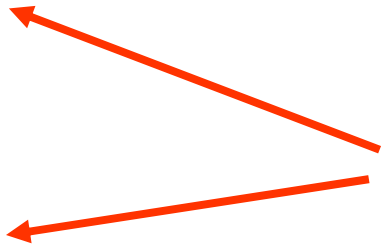
Example: Derive the excitation table for an S-R latch

S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d

Q	Q*	S	R
0	0		
0	1		
1	0		
1	1		

Example: Derive the excitation table for an S-R latch

S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d



Q	Q*	S	R
0	0	0	d
0	1		
1	0		
1	1		

Example: Derive the excitation table for an S-R latch

S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d

Q	Q*	S	R
0	0	0	d
0	1	1	0
1	0		
1	1		

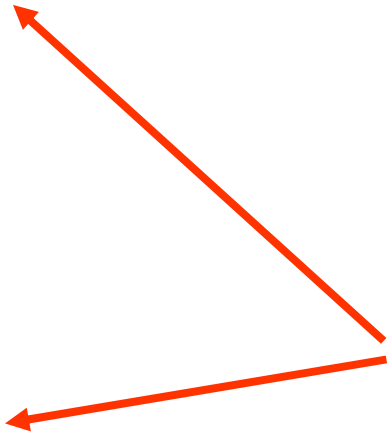
Example: Derive the excitation table for an S-R latch

S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d

Q	Q*	S	R
0	0	0	d
0	1	1	0
1	0	0	1
1	1		

Example: Derive the excitation table for an S-R latch

S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	d
1	1	1	d



Q	Q*	S	R
0	0	0	d
0	1	1	0
1	0	0	1
1	1	d	0

NOTE: Two excitation equations are required for each flip-flop…probably not desirable!

Example: Derive the excitation table for a T flip-flop

T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0

Q	Q*	T
0	0	0
0	1	
1	0	
1	1	

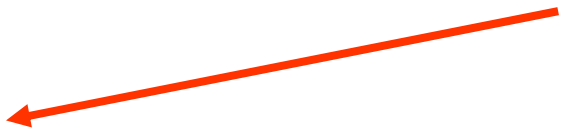
Example: Derive the excitation table for a T flip-flop

T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0

Q	Q*	T
0	0	0
0	1	1
1	0	
1	1	

Example: Derive the excitation table for a T flip-flop

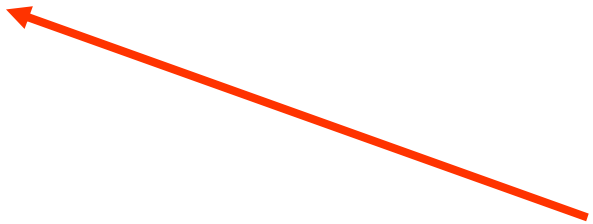
T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0



Q	Q*	T
0	0	0
0	1	1
1	0	1
1	1	

Example: Derive the excitation table for a T flip-flop

T	Q	Q*
0	0	0
0	1	1
1	0	1
1	1	0



Q	Q*	T
0	0	0
0	1	1
1	0	1
1	1	0

NOTE: Here, only one excitation equation is required for each flip-flop – a common application is *binary counters*

Example: Derive the excitation table for a D flip-flop

D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1

Q	Q*	D
0	0	0
0	1	
1	0	
1	1	

Example: Derive the excitation table for a D flip-flop

D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1

Q	Q*	D
0	0	0
0	1	1
1	0	
1	1	

Example: Derive the excitation table for a D flip-flop

D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1

Q	Q*	D
0	0	0
0	1	1
1	0	0
1	1	

Example: Derive the excitation table for a D flip-flop

D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1

Q	Q*	D
0	0	0
0	1	1
1	0	0
1	1	1

NOTE: For D flip-flops, Q*=D, which means the excitation equation is identical to the next state equation, which makes synthesis using D flip-flops very straight-forward!

Flip-Flop Choice

- **Any type of latch or flip-flop (S-R, D, T) may be chosen for a sequential cicruit's state memory; this choice, however, will determine *how much work* you will have to do when it's time to "turn the crank" (i.e., transform the next state equations into a circuit)**
- **Our focus for state machine synthesis will be on use of *edge-triggered D flip-flops***
  - **they are incorporated directly into the PLDs used in lab**
  - **they require the least amount of "crank work" to realize next state equations**

Clicker Quiz

Q1. Identify which statement concerning **state machine models** is **true**:

A. Mealy and Moore models that represent equivalent state machines will ***always*** have the **same** number of states

B. Mealy and Moore models that represent equivalent state machines will ***always*** have a **different** number of states

C. any **Mealy model** can be transformed into **an equivalent Moore model,** and *vice-versa*

D. Mealy and Moore models that represent equivalent state machines, when realized, will exhibit the **same observable behavior** (i.e., if placed in a "black box", their **observable behavior** would be **indistinguishable**)

E. none of the above

Q2. Designing a state machine based on **minimum risk** means:

A. there are no hazards in the clocking signal
B. there are no “don’t cares” in the output equations
C. there are no “don’t cares” in the next state equations
D. all of the above
E. none of the above

Q3. An FSM design has 212 states; to reduce the number of flip-flops required by one, you would have to identify and eliminate _____ redundant state(s).

A. 1
B. 2
C. 44
D. 84
E. none of the above

Q4. Formal **state-minimization procedures** are **seldom used** by most digital designers because:

A. there are situations where *increasing* the number of states may simplify the design or reduce its cost

B. the *designer can do more* to simplify a state machine [than using formal state-minimization procedures] *during the state-assignment phase* of the design

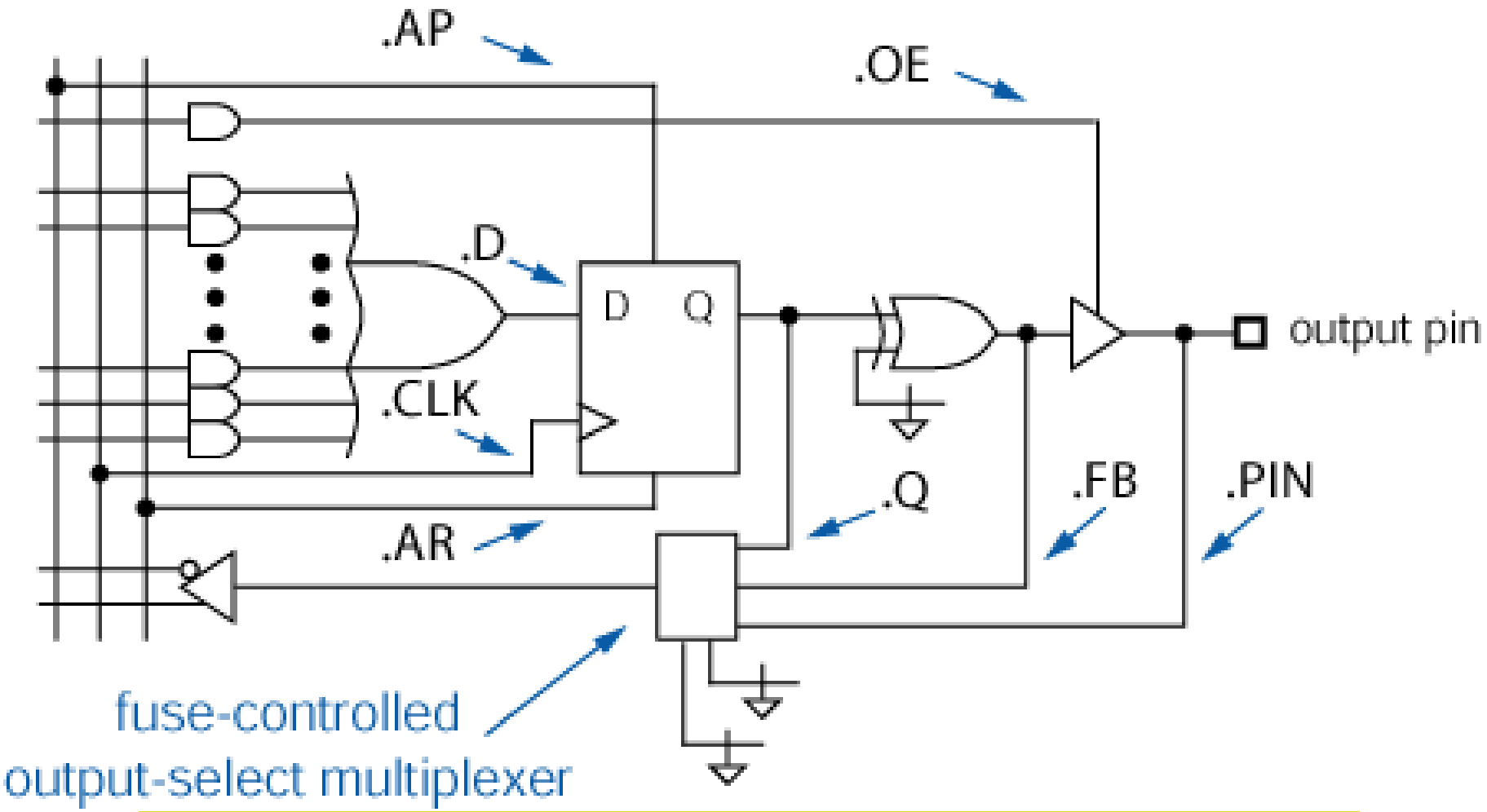
C. by carefully matching state meanings to the requirements of the problem, experienced digital designers can produce state tables with a minimal or near-minimal number of states

D. all of the above

E. none of the above

State Machines in ABEL

- There are **three** basic ways state machines can be specified in ABEL
  - as a ***"clocked" truth table***, using the ***clocked truth table*** operator **`:>`**
  - as a set of ***next state equations***, using the ***clocked assignment*** operator **`:=`**
  - as a ***state diagram***, using a series of **`GOTO`** and/or **`IF-THEN-ELSE`** clauses to specify the state transitions
- Differences in ***macrocell architecture*** will determine the complexity of state machine that can be implemented with a given PLD



Attribute Suffixes in a Complex PLD

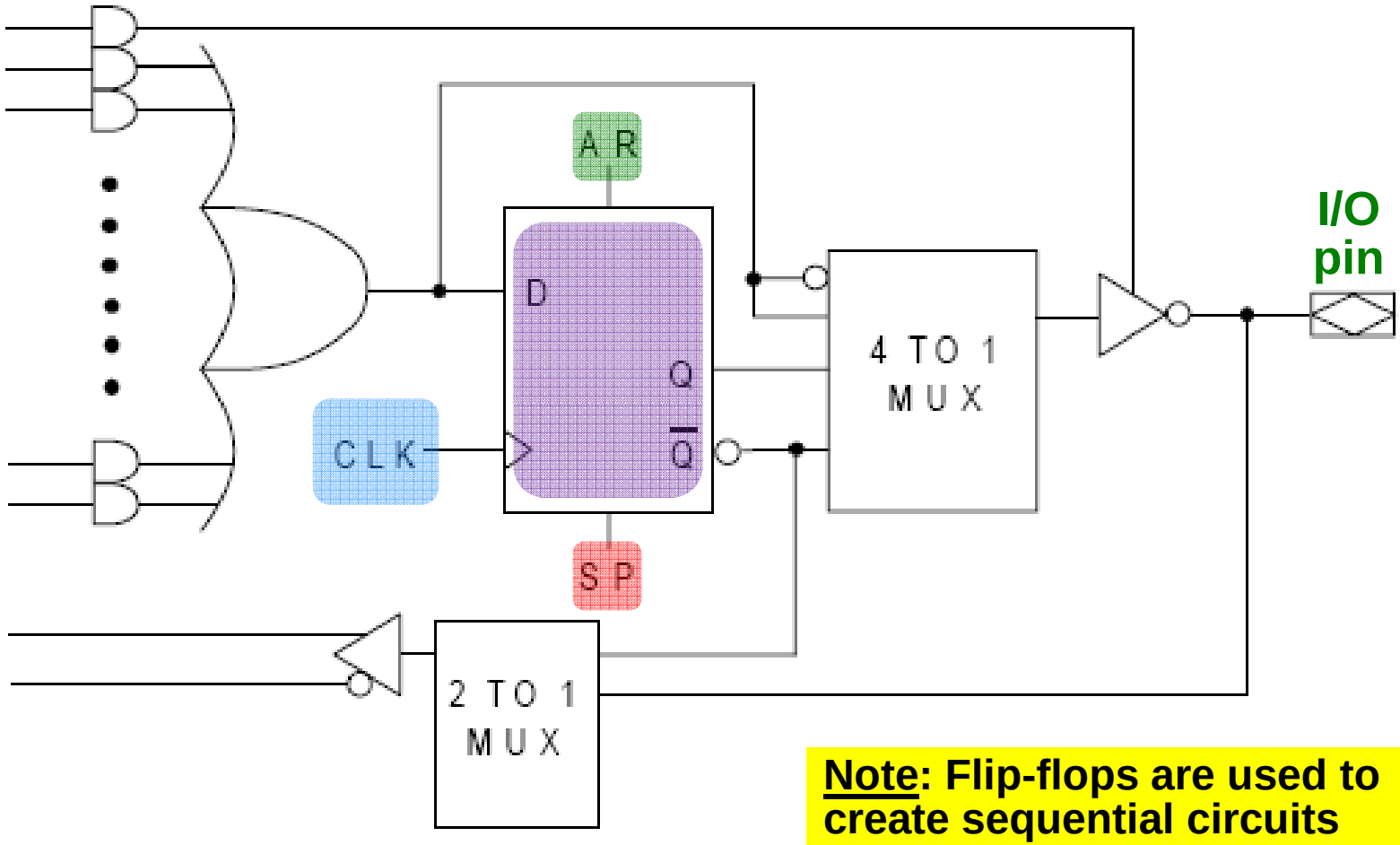
Attribute Suffixes

- **ABEL allows three possible values on the *right-hand side* of an equation using an *attribute suffix* on the signal name**
  - **“`.Q`” the actual flip-flop output pin before any programmable inversion**
  - **“`.FB`” a value equal to the value that the output pin would have if enabled**
  - **“`.PIN`” the actual signal at the PLD output pin (this signal is floating or may be used as an input if the tri-state driver is not enabled)**

Attribute Suffixes

- **ABEL allows equations to be written for other functions within a macrocell, again using *attribute suffixes* on the signal name:**
  - **“ `.D` ” the flip-flop D input**
  - **“ `.CLK` ” the flip-flop CLOCK input**
  - **“ `.OE` ” the pin tri-state buffer output enable**
  - **“ `.AP` ” the flip-flop asynchronous preset (equivalent to the “S” input on an S R latch)**
  - **“ `.AR` ” the flip-flop asynchronous reset (equivalent to the “R” input on an S R latch)**

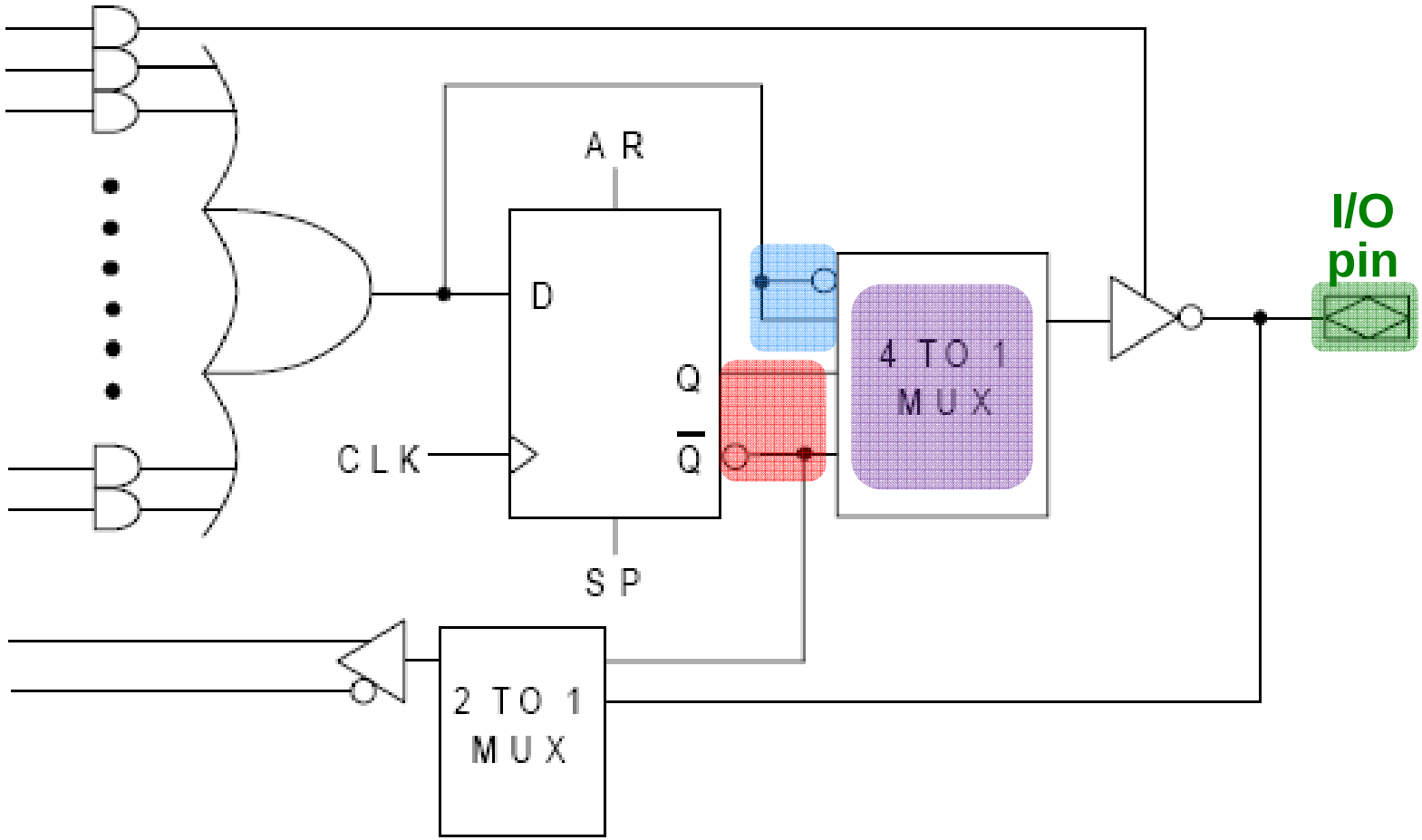
GAL22V10 Output Logic Macrocell ("OLMC")



Note: Flip-flops are used to create sequential circuits

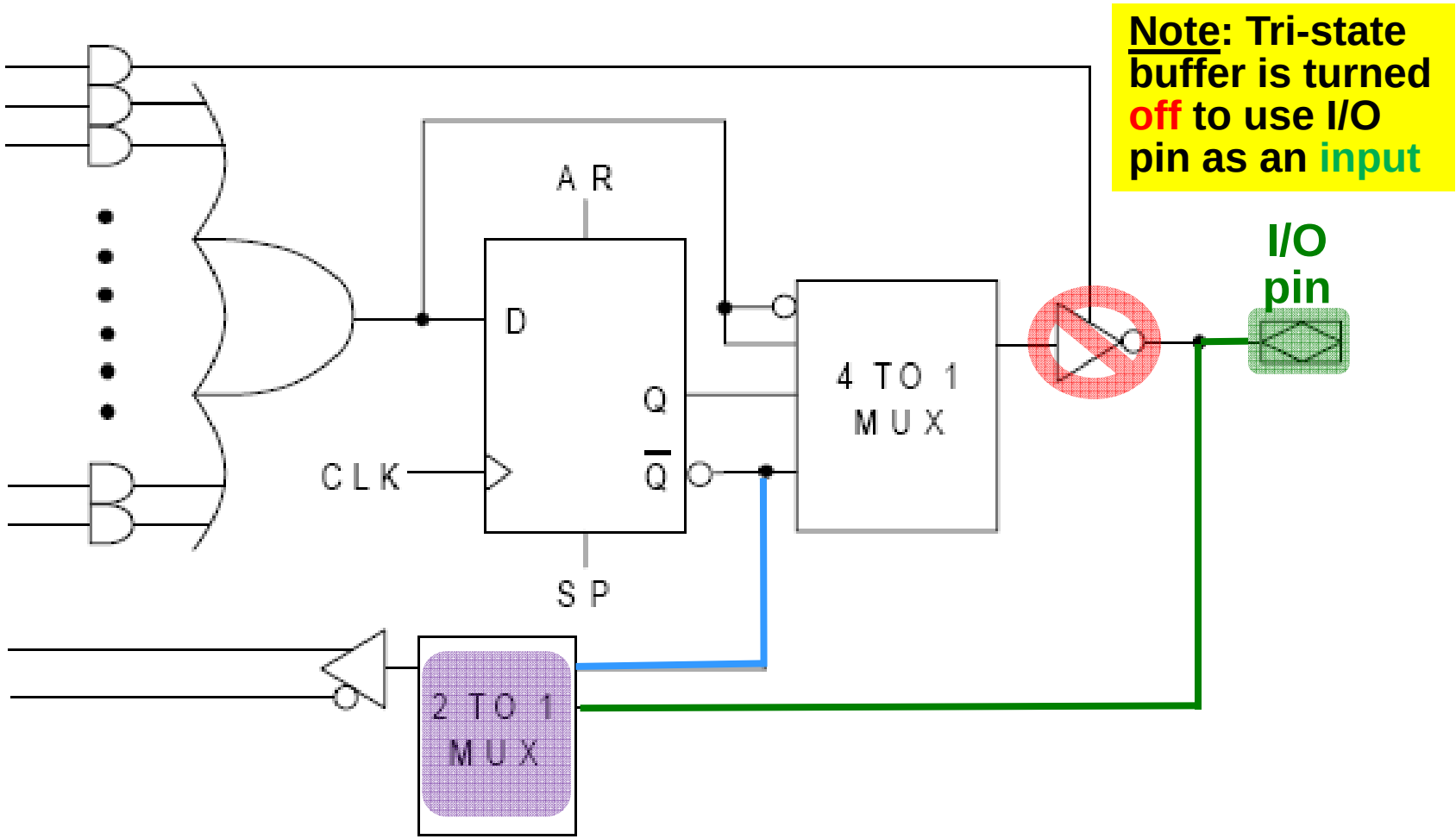
All OLMC edge-triggered D flip-flops utilize common clock (CLK) , asynchronous reset (AR), and asynchronous preset (SP) signals

GAL22V10 Output Logic Macrocell ("OLMC")



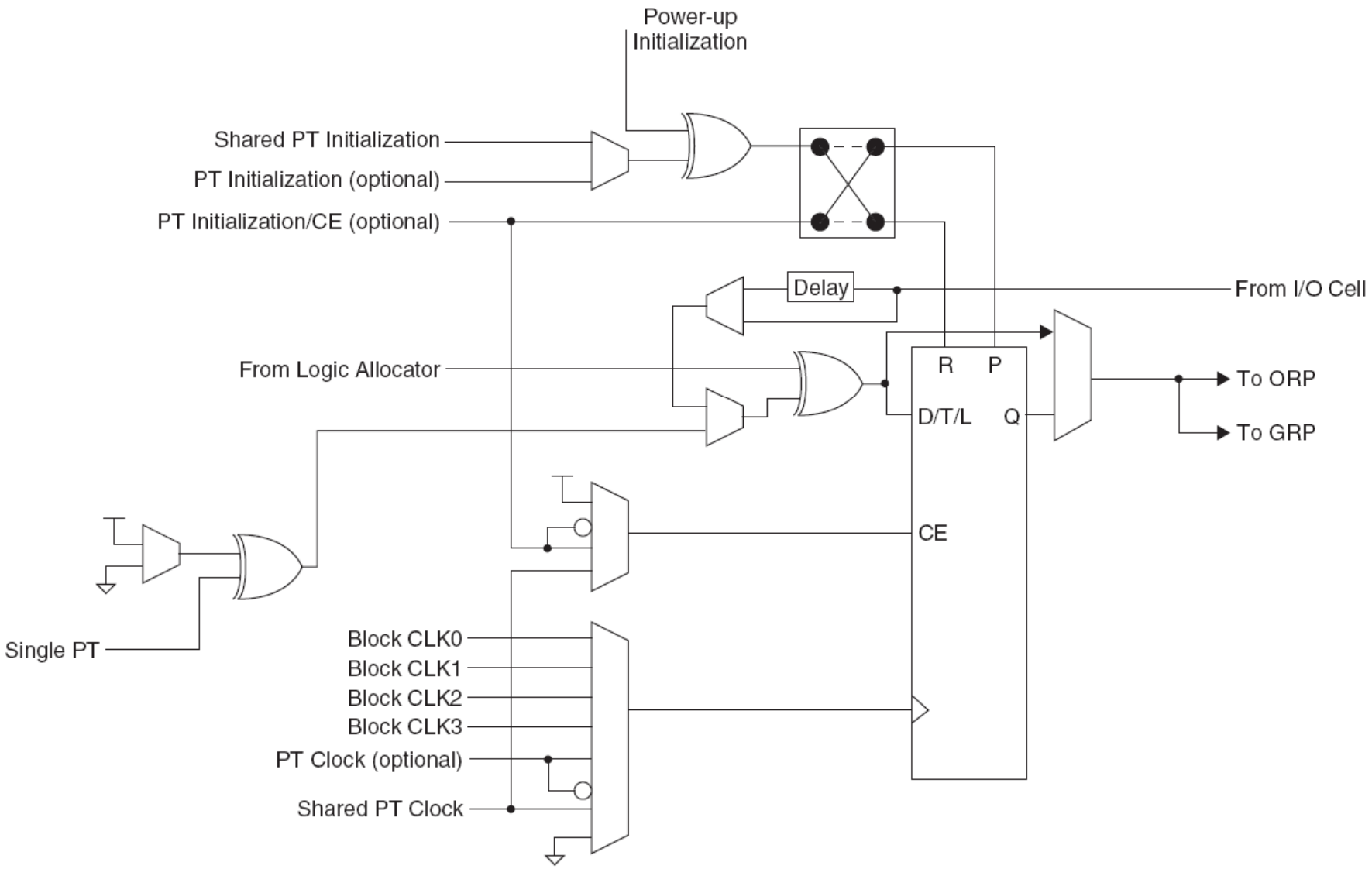
4:1 multiplexer selects (routes) true/complemented combinational or true/complemented registered function to the I/O pin

GAL22V10 Output Logic Macrocell ("OLMC")

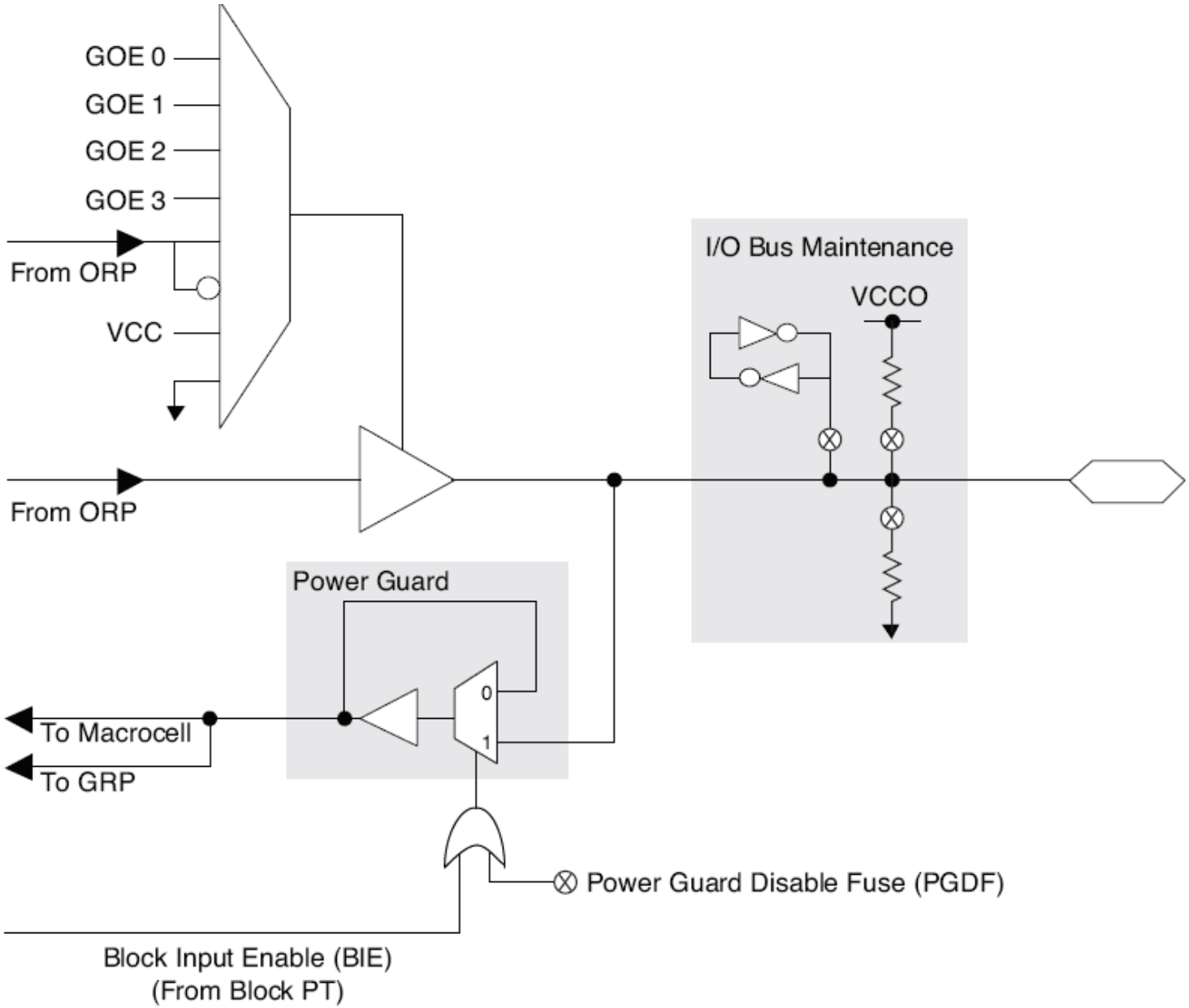


2:1 multiplexer selects (routes) true/complemented I/O pin or true/complemented registered feedback to the P-term array

ispMACH 4000ZE Macrocell



ispMACH 4000ZE I/O Cell



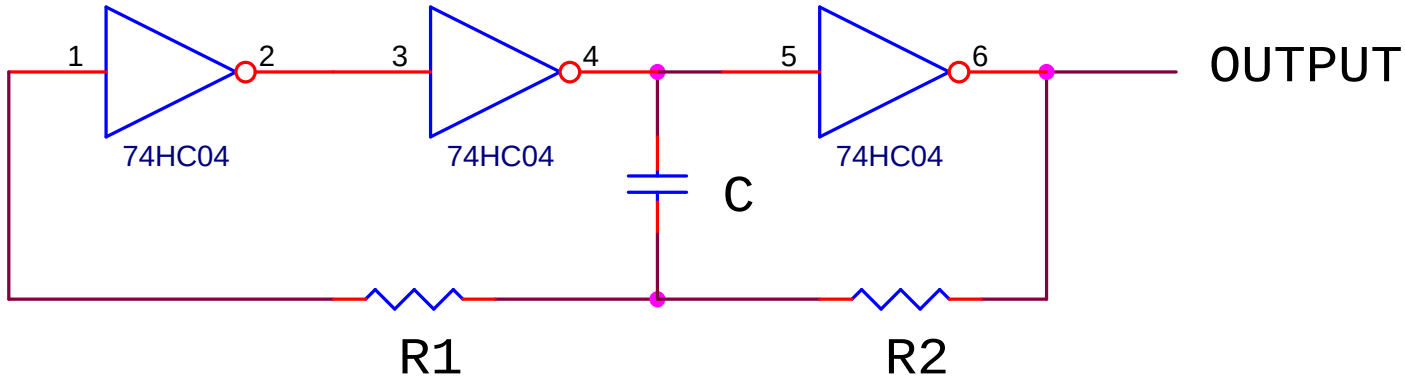
Clocking Considerations

- State machines require a ***clocking signal*** in order to operate "sequentially"
- There are two basic types of clocking signals that can be used:
  - ***periodic*** ("continuously running"), generated using an ***oscillator circuit***
  - ***event*** (non-periodic, single clock edge), generated using a ***bounce-free*** switch or sensor contact closure
- A ***timing diagram*** can be used to show the relationship between the clock and various input, output, and internal signals – it can also be used to help answer the key question facing computer system designers: ***"How fast can this thing run?"***

Periodic Clock Generation Circuits

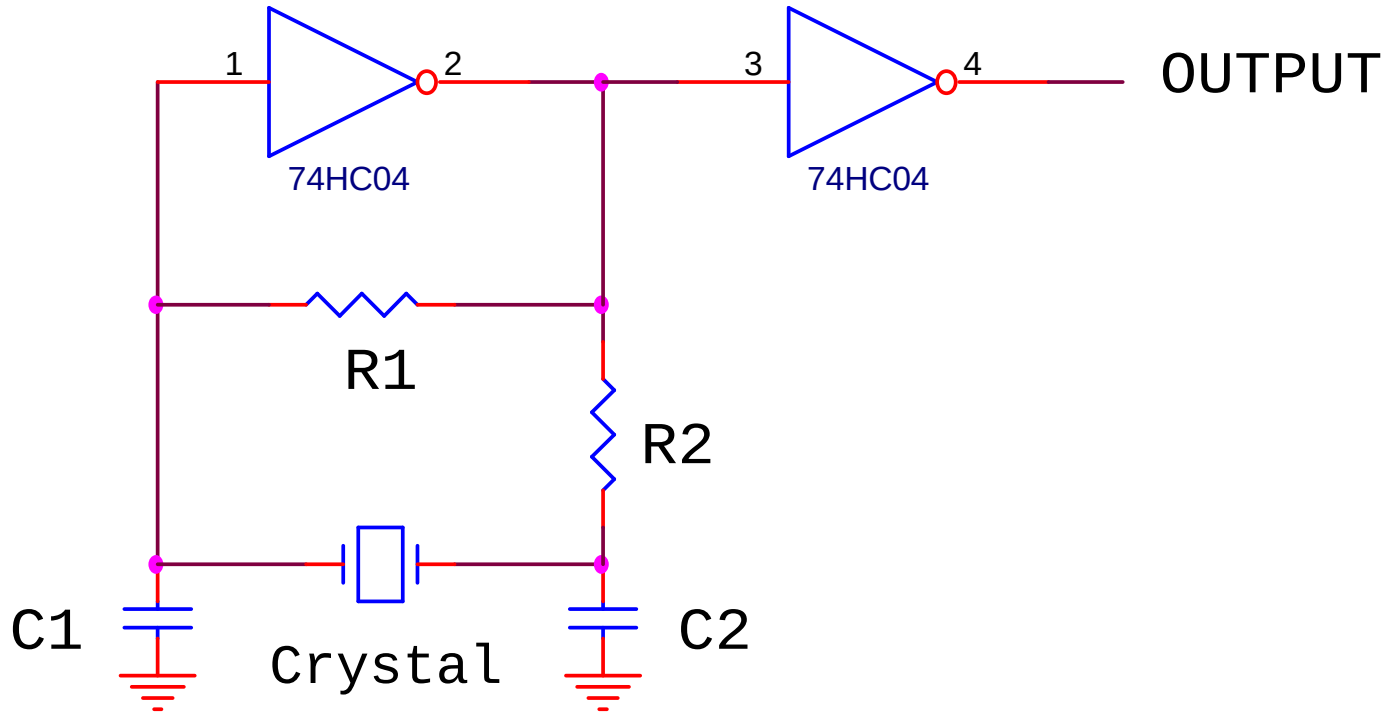
- **Periodic clock signals can be generated using several different types of *oscillator* circuits:**
  - **based on an R-C time constant *(least accurate)***
  - **based on a ceramic resonator**
  - **based on a quartz crystal *(most accurate)***
- **Issues of interest include the following:**
  - **frequency of operation**
  - **duty cycle**
  - **transition time**
  - **ringing (undershoot / overshoot)**
  - **stability (long term drift / short term "jitter")**
  - **driving capability / need for buffers**
  - **skew (different length paths on PCB)**

Example - CMOS “Ring” Oscillator



$$f \cong (2C(0.4R_{eq} + 0.7R_2))^{-1} \text{ where } R_{eq} = (R_1R_2)/(R_1+R_2)$$

Example - Crystal Oscillator Circuit



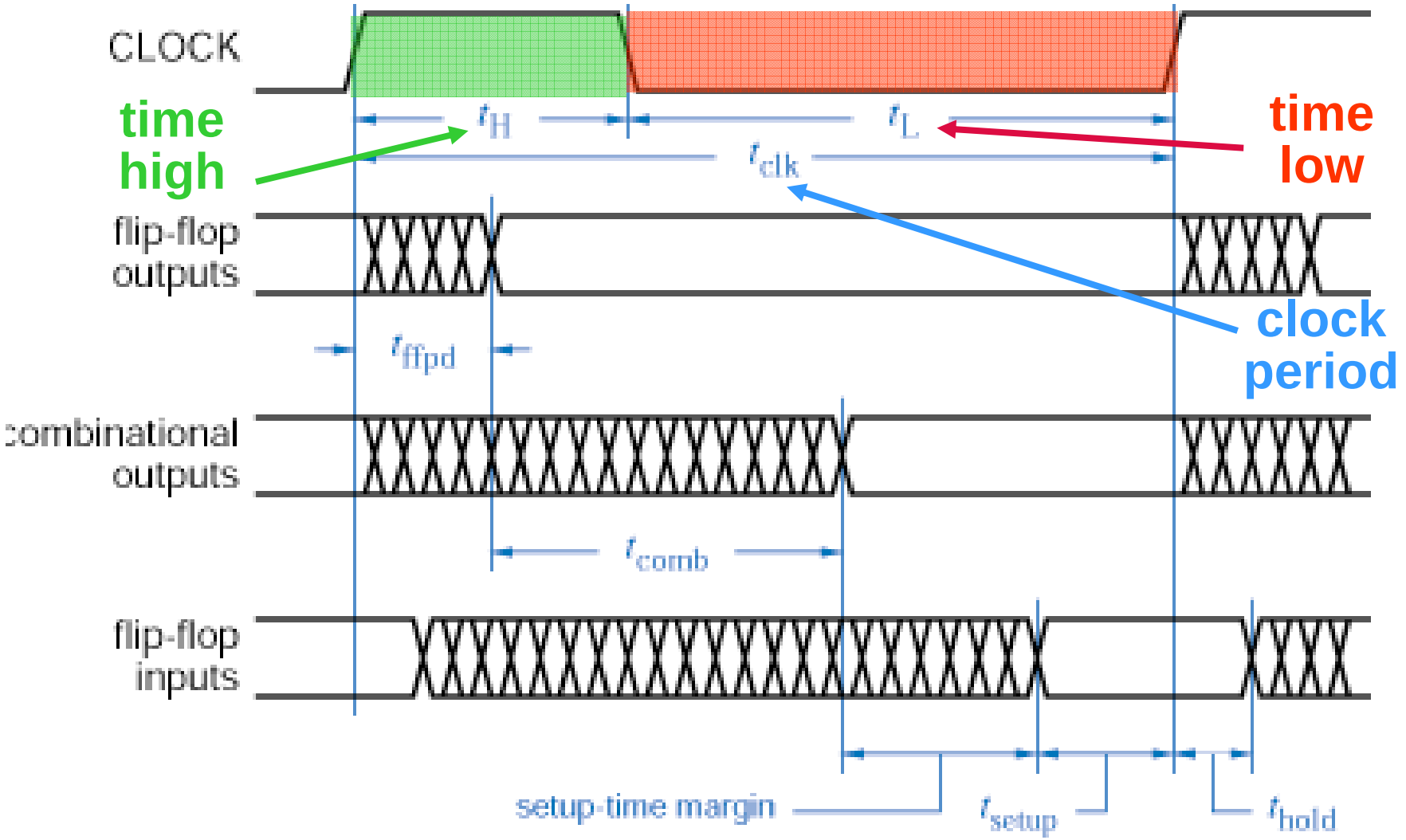
For a 1 MHz oscillator, use R1 = 22 MΩ, R2 = 22 KΩ, C1 = 20 pF, and C2 = 10 pF

ispMach 4000ZE Internal Oscillator

```
MODULE OscTest

TITLE 'ispMach4256ZE Oscillator Setup'

LIBRARY 'lattice';

DECLARATIONS
" Use maximum possible internal divisor -> yields approx 4 Hz output frequency

XLAT_OSCTIMER(DYNOSCDIS, TIMERRES, OSCOUT, TIMEROUT, 1048576);

timdiv node istype 'reg_d,buffer';
osc_dis, osc_rst, osc_out, tmr_out node istype 'com';

EQUATIONS

osc_dis=0;
osc_rst=0;

I1 OSCTIMER(osc_dis, osc_rst, osc_out, tmr_out);

" Divide internal oscillator frequency by 2 to get approx 2 Hz output at TIMDIV
timdiv.clk = tmr_out;
timdiv := !timdiv;

END
```

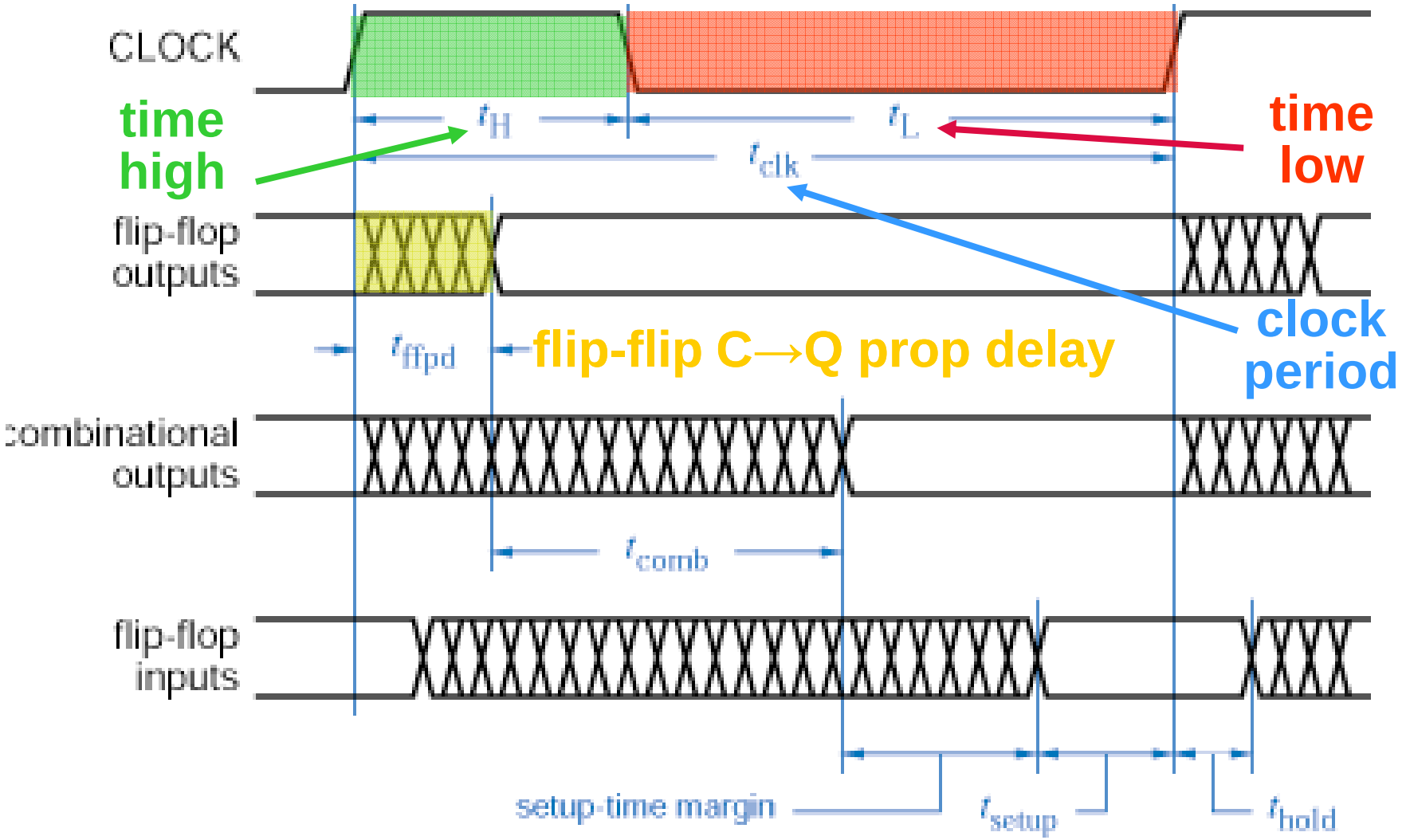
Example – Timing Diagram and Specifications

clock frequency (f) = $1/t_{clk}$ ***duty cycle = $t_H/(t_H+t_L)$***



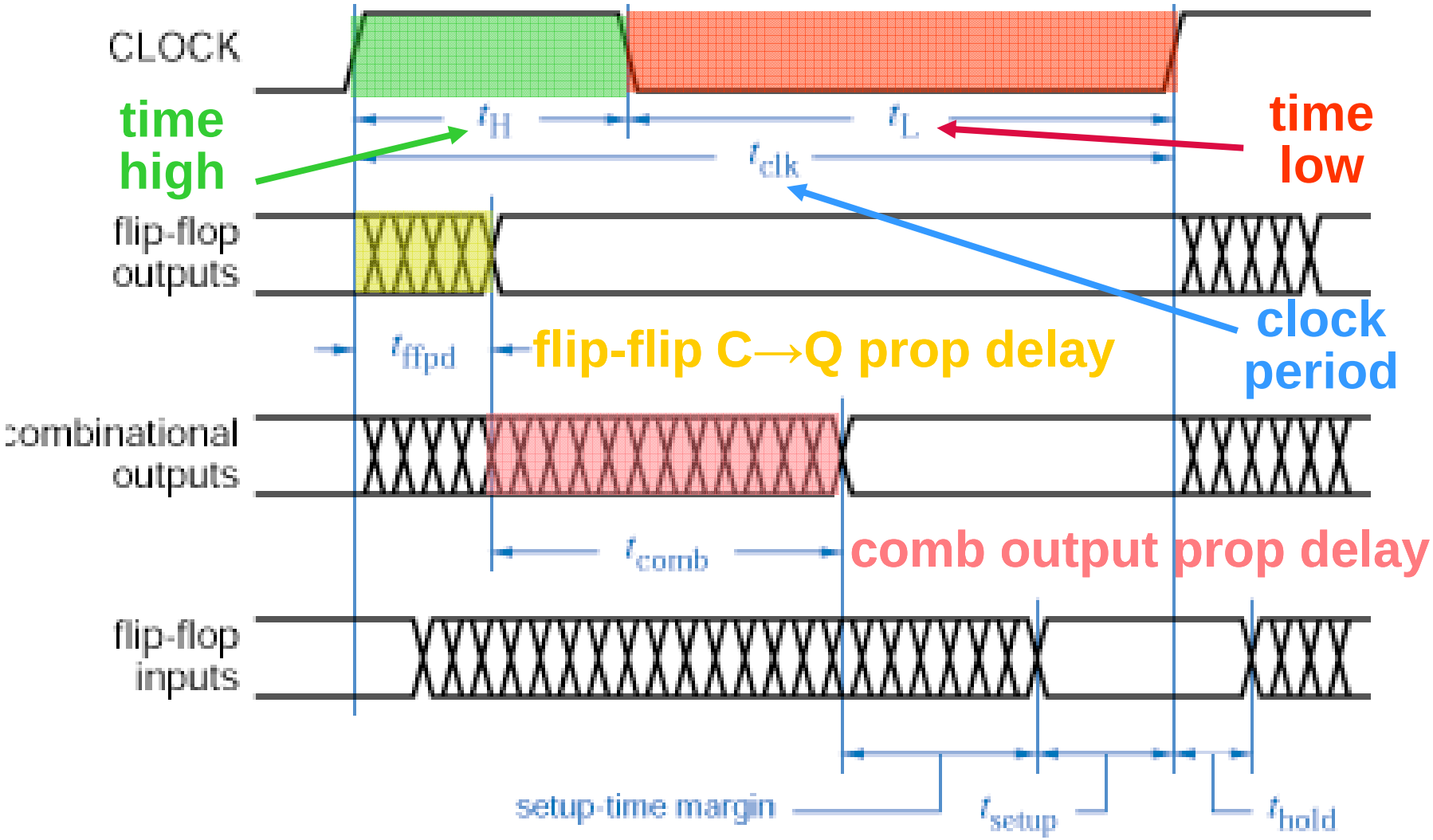
Example – Timing Diagram and Specifications

clock frequency (f) = $1/t_{clk}$ | ***duty cycle = $t_H/(t_H+t_L)$***



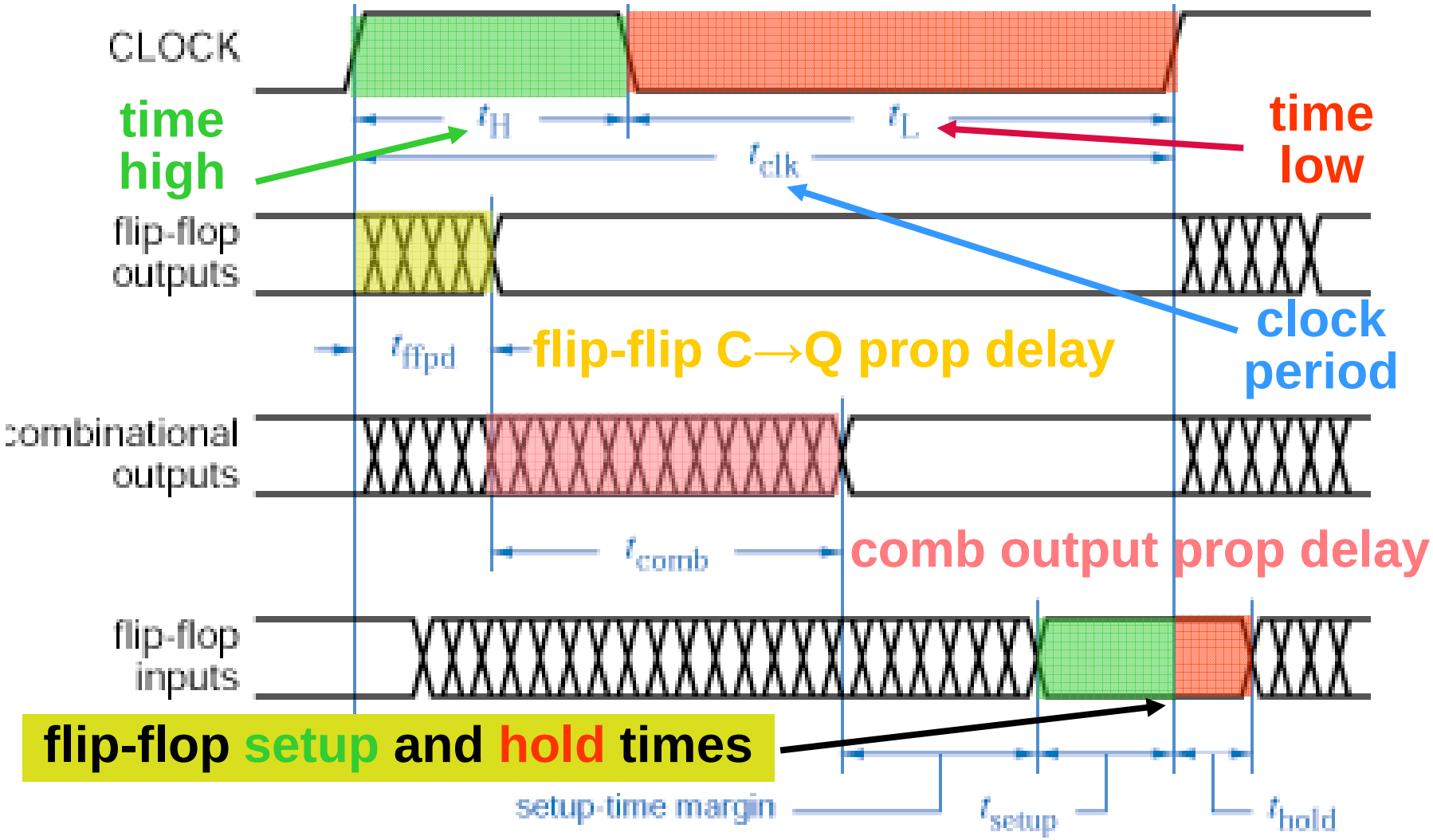
Example – Timing Diagram and Specifications

clock frequency (f) = $1/t_{clk}$ | ***duty cycle = $t_H/(t_H+t_L)$***



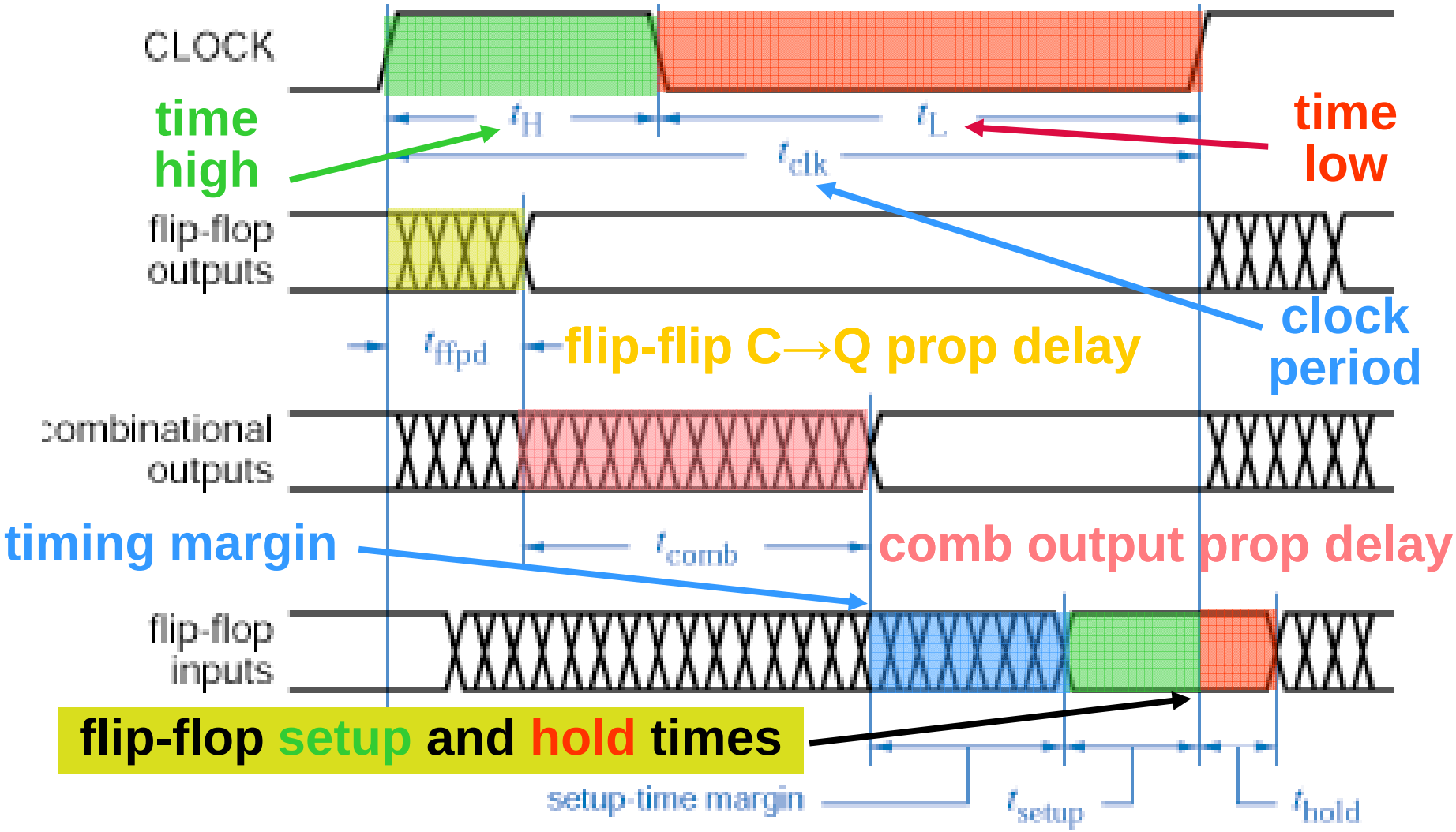
Example – Timing Diagram and Specifications

clock frequency (f) = $1/t_{clk}$ ***duty cycle = $t_H/(t_H+t_L)$***



Example – Timing Diagram and Specifications

clock frequency (f) = $1/t_{clk}$ ***duty cycle = $t_H/(t_H+t_L)$***



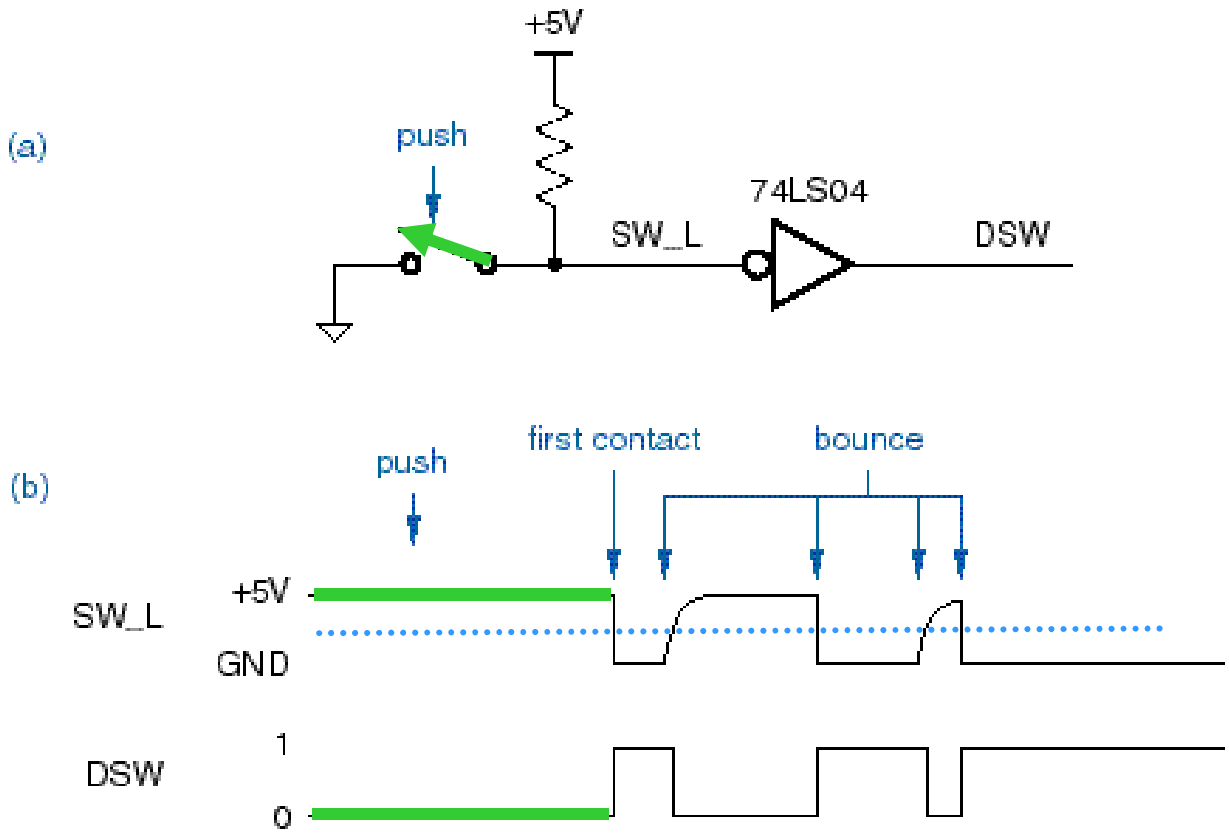
Event Clock Generation Circuits

- **Some applications of sequential circuits require that they be clocked by an *event***
  - **sensor firing (open drain transistor changing from high impedance to low impedance)**
  - **contact closure (pushing a button)**
- **Problem: Mechanical switches have contacts that "*bounce*" (i.e., "make"/"break" multiple times before the contacts "settle")**
- **Illustration: Use of a single-pole, single throw (S.P.S.T.) normally-open (N.O.) pushbutton as a clocking signal**

Example – S.P.S.T. Pushbutton Used as Clock

S.P.S.T. stands for “single pole, single throw”

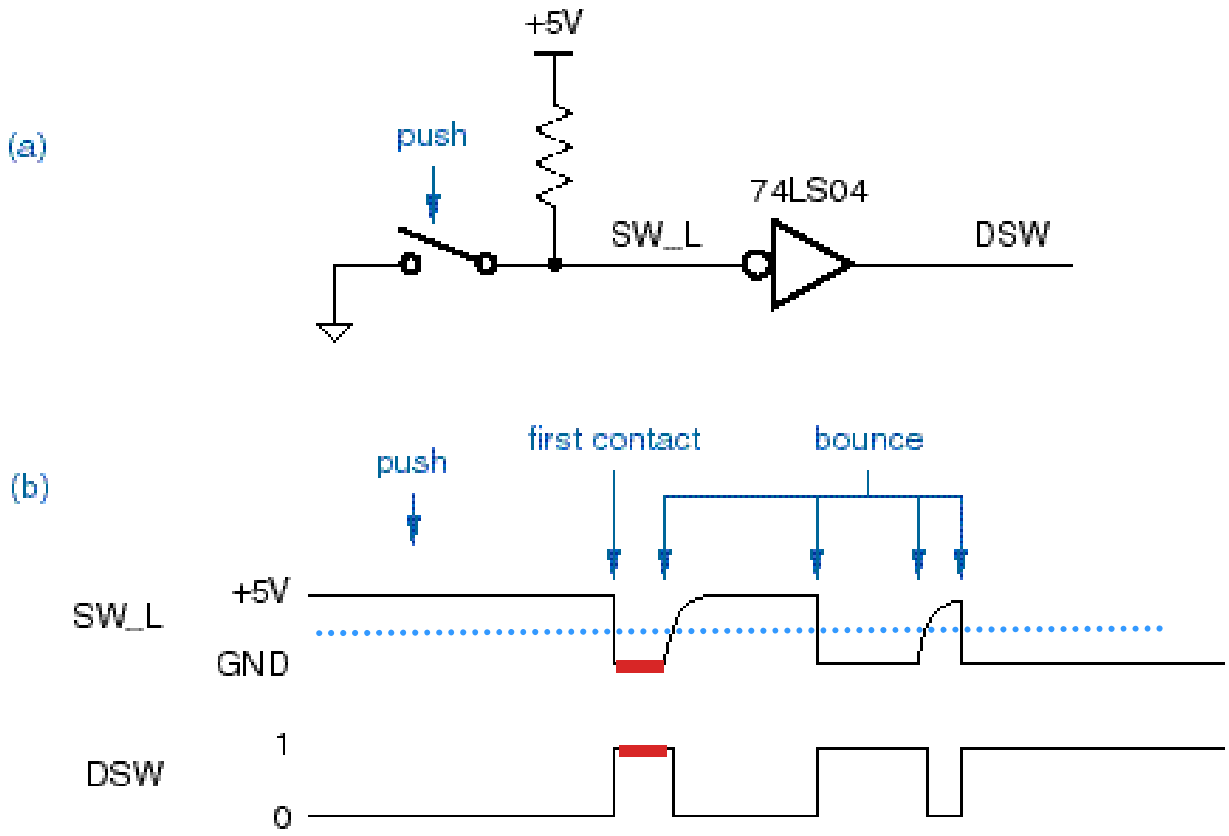
N.O. stands for “normally open”



Example – S.P.S.T. Pushbutton Used as Clock

S.P.S.T. stands for “single pole, single throw”

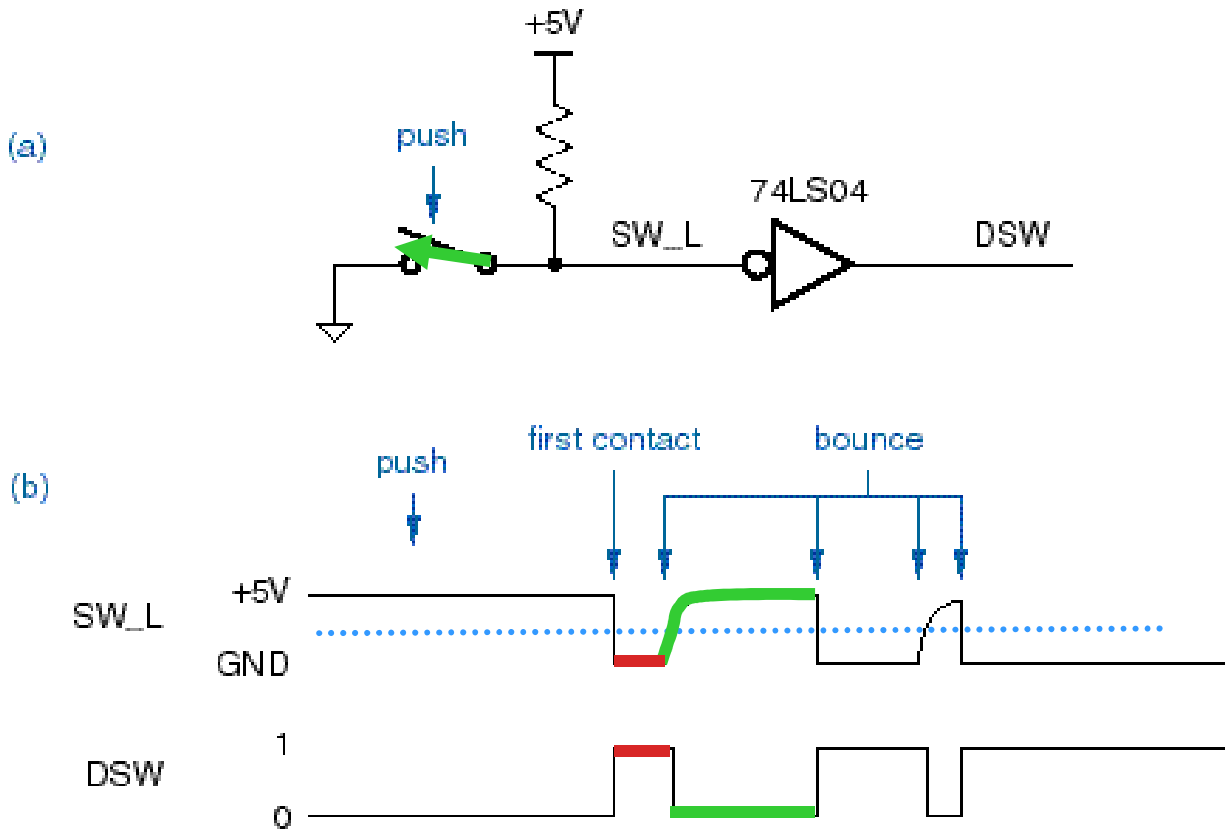
N.O. stands for “normally open”



Example – S.P.S.T. Pushbutton Used as Clock

S.P.S.T. stands for "single pole, single throw"

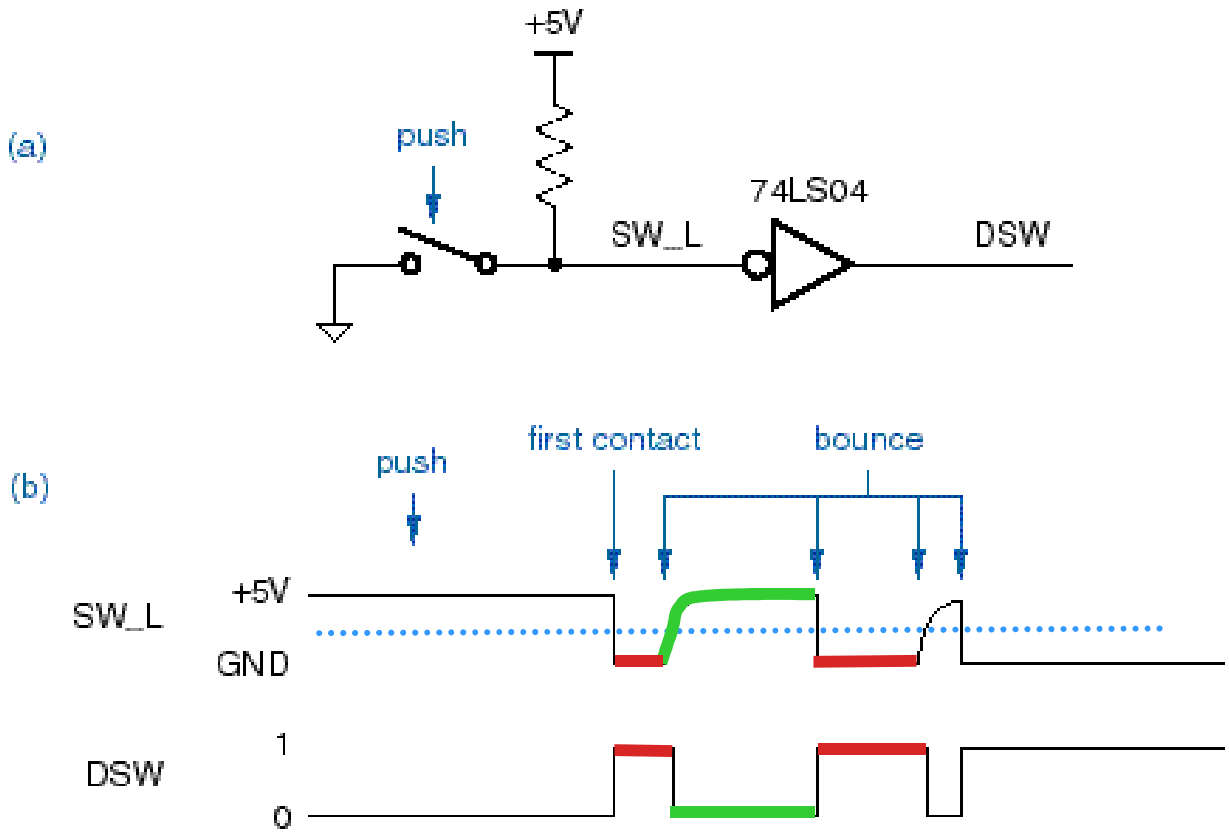
N.O. stands for "normally open"



Example – S.P.S.T. Pushbutton Used as Clock

S.P.S.T. stands for "single pole, single throw"

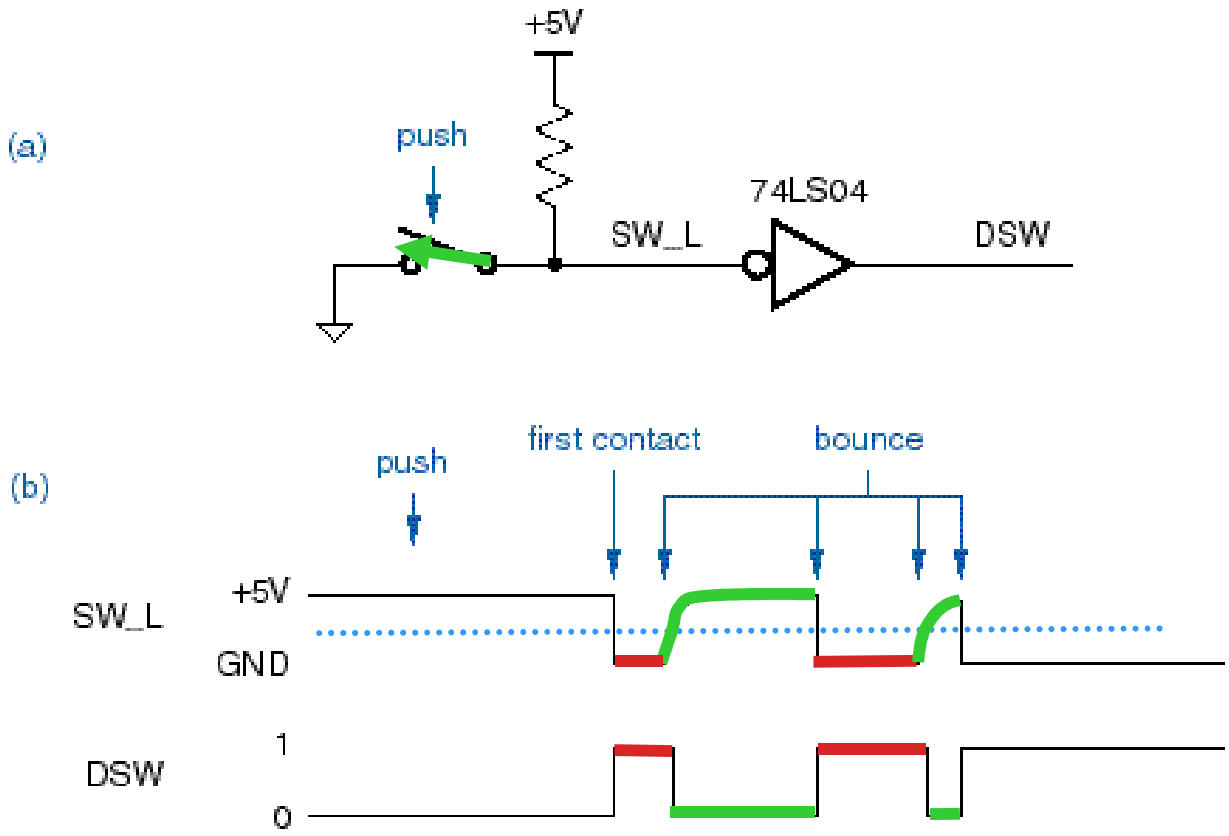
N.O. stands for "normally open"



Example – S.P.S.T. Pushbutton Used as Clock

S.P.S.T. stands for “single pole, single throw”

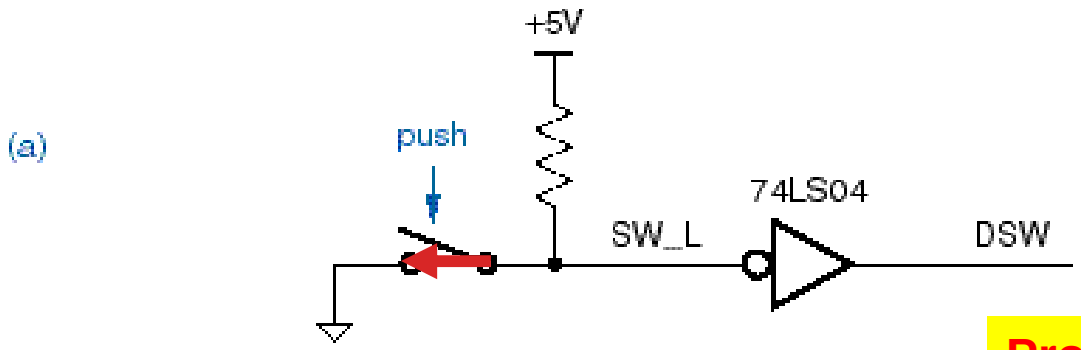
N.O. stands for “normally open”



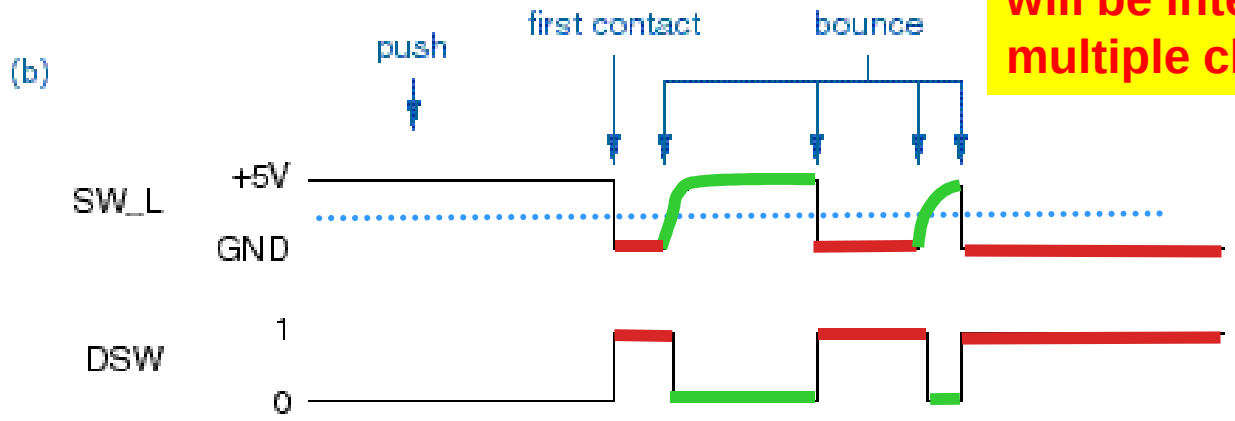
Example – S.P.S.T. Pushbutton Used as Clock

S.P.S.T. stands for "single pole, single throw"

N.O. stands for "normally open"

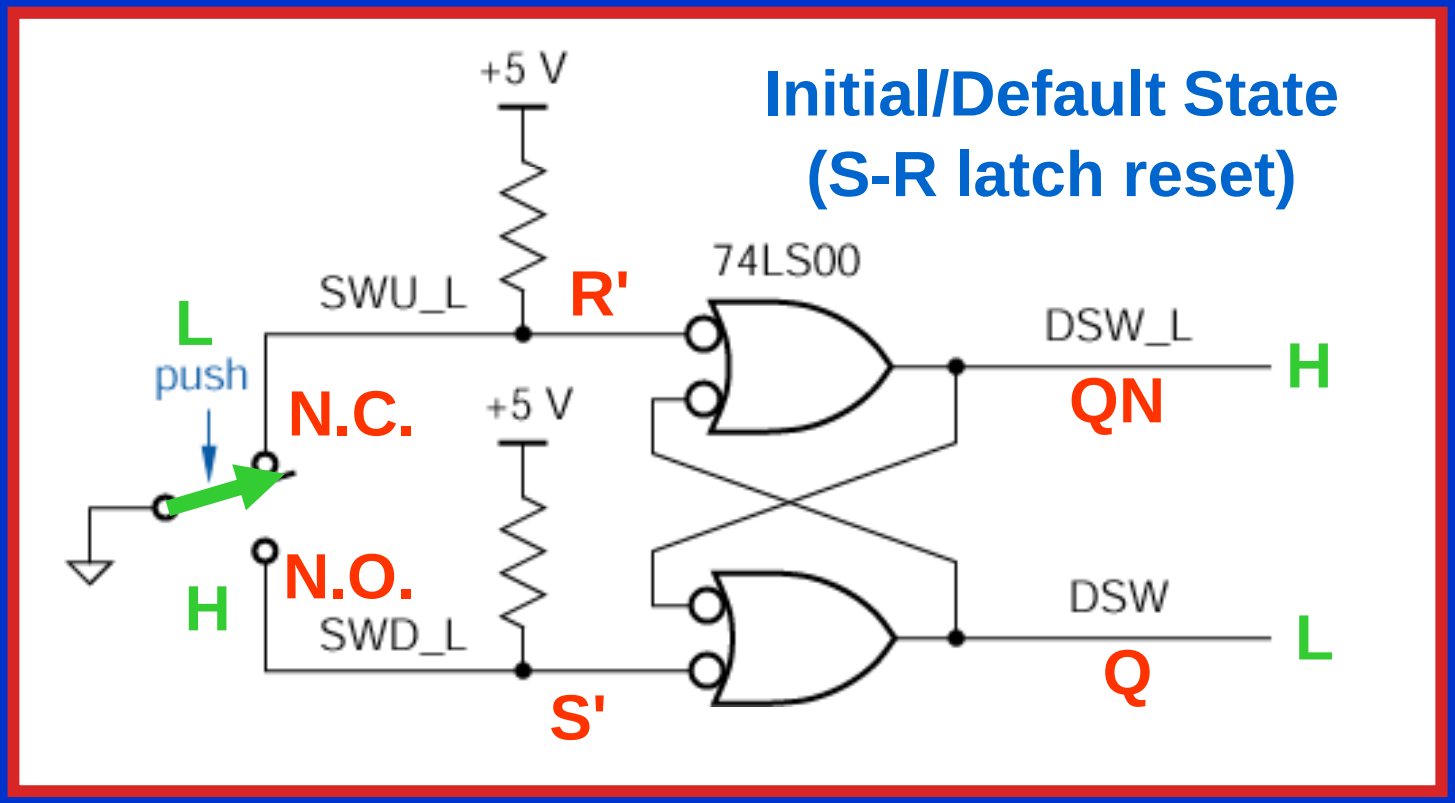


<u>Problem</u>: The "bounces" will be interpreted as multiple clock edges



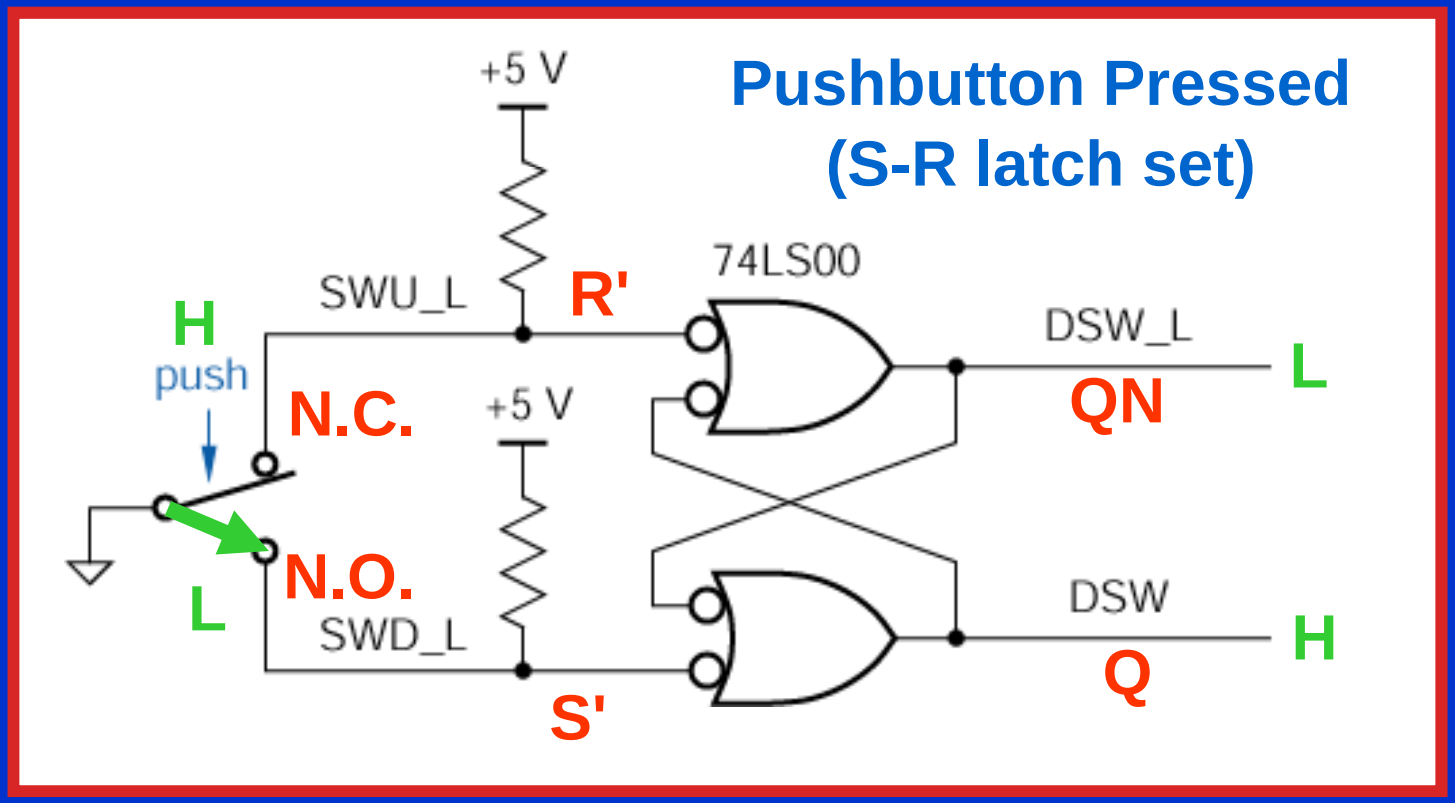
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



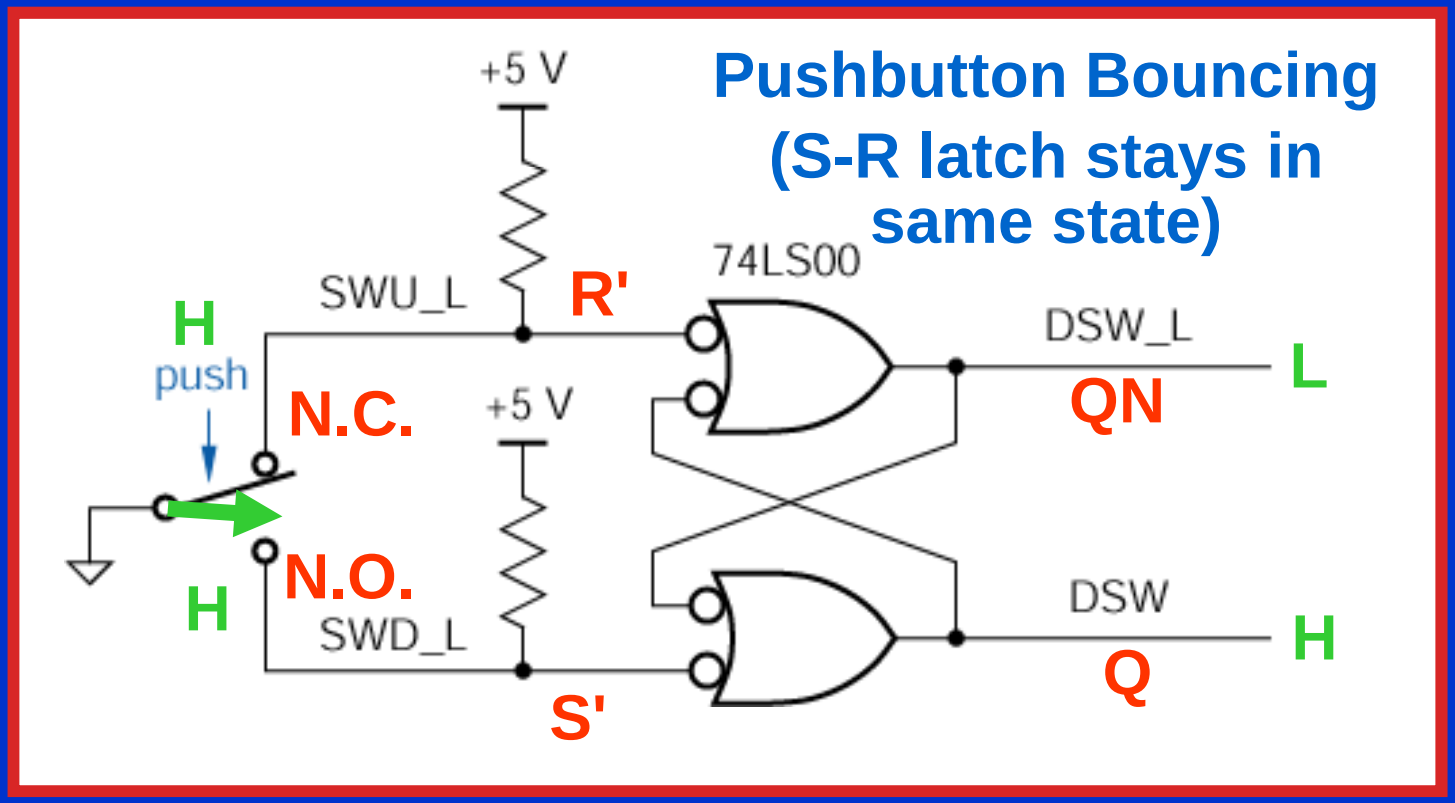
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



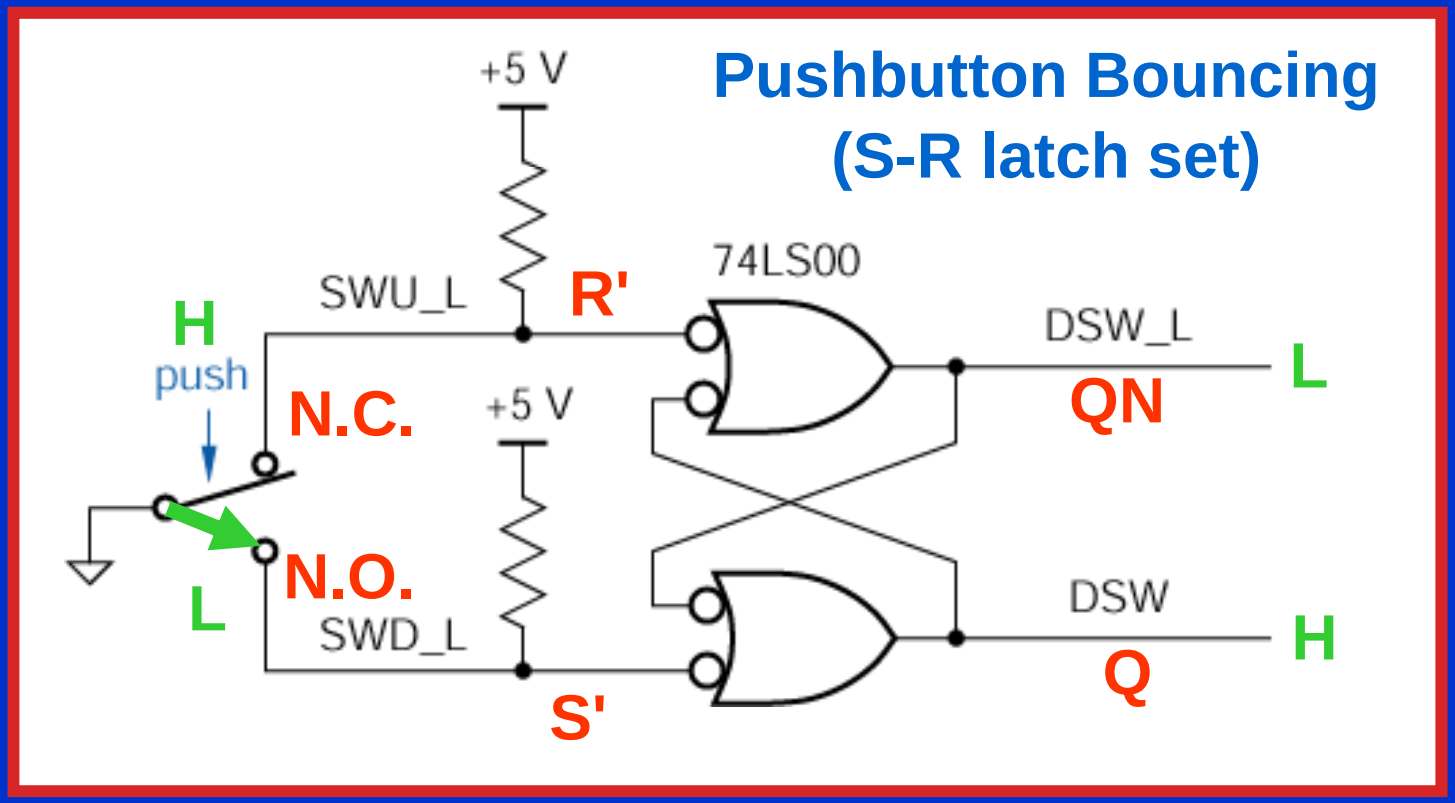
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



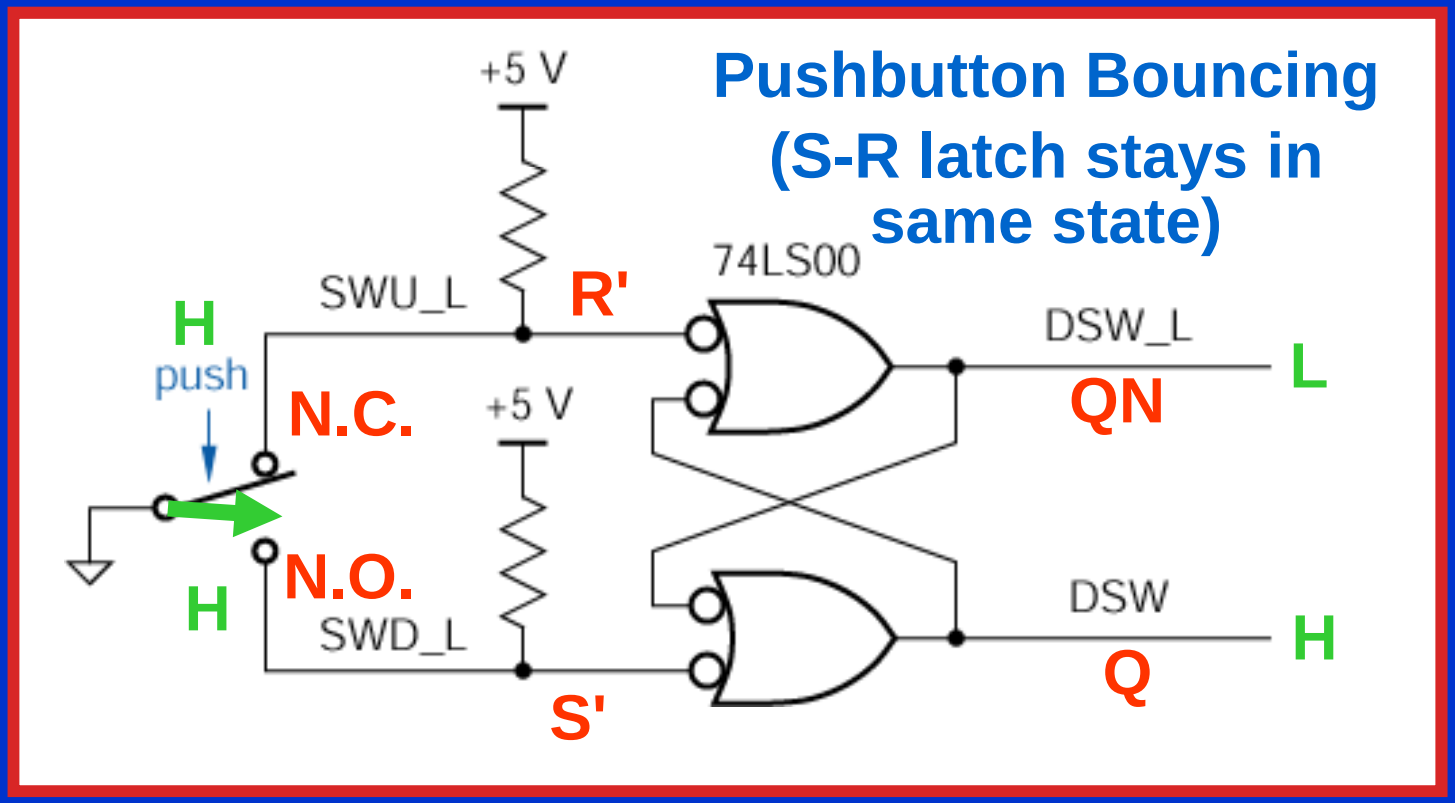
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



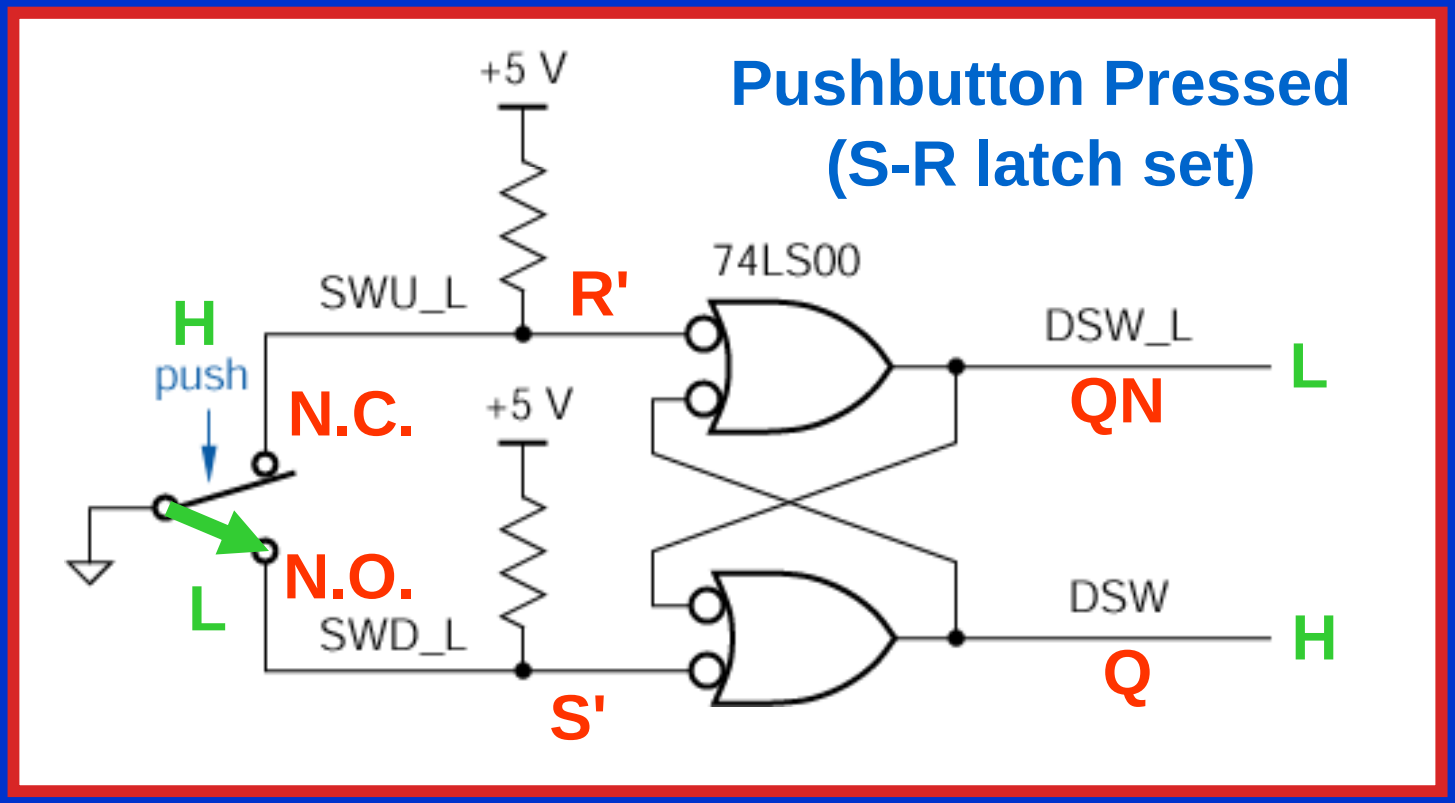
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



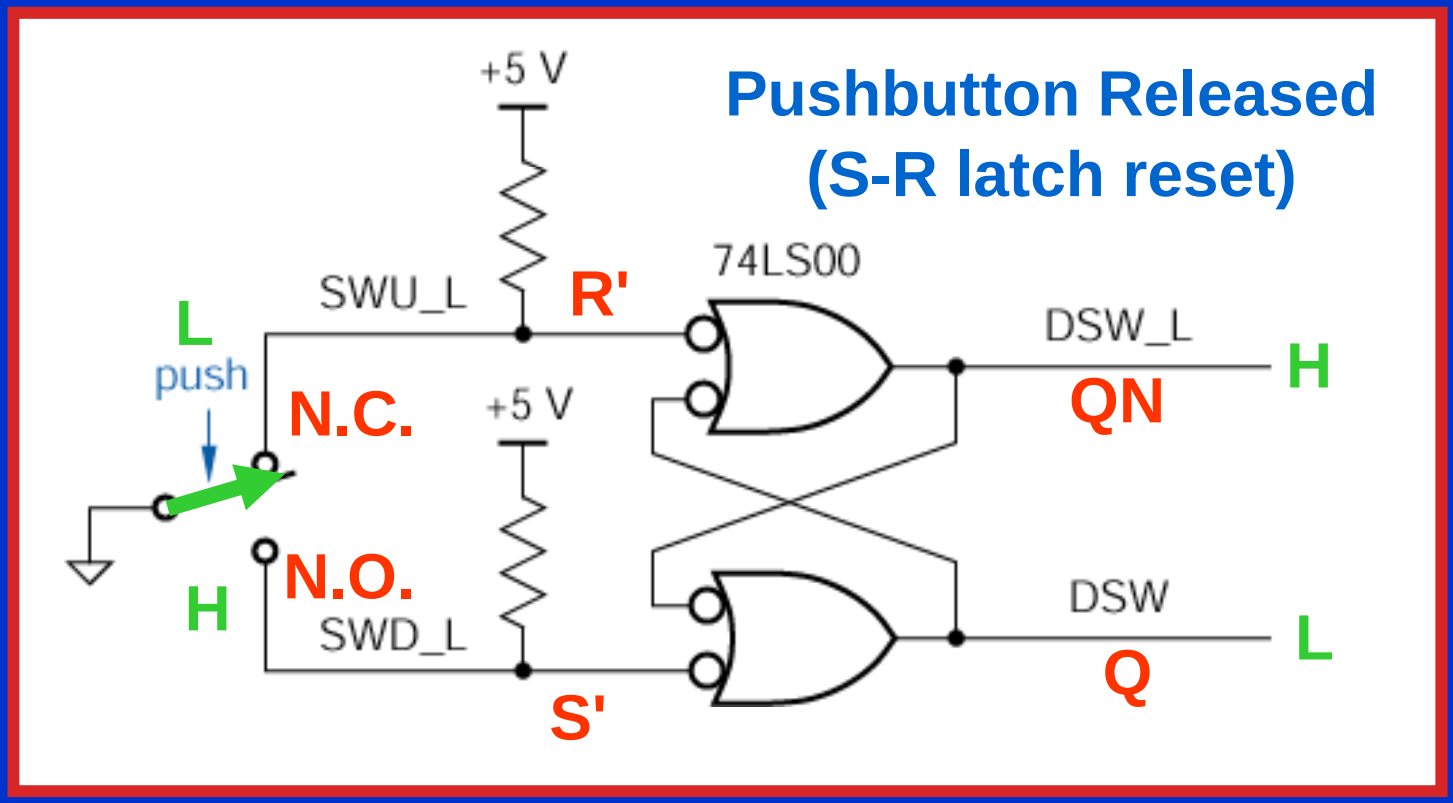
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



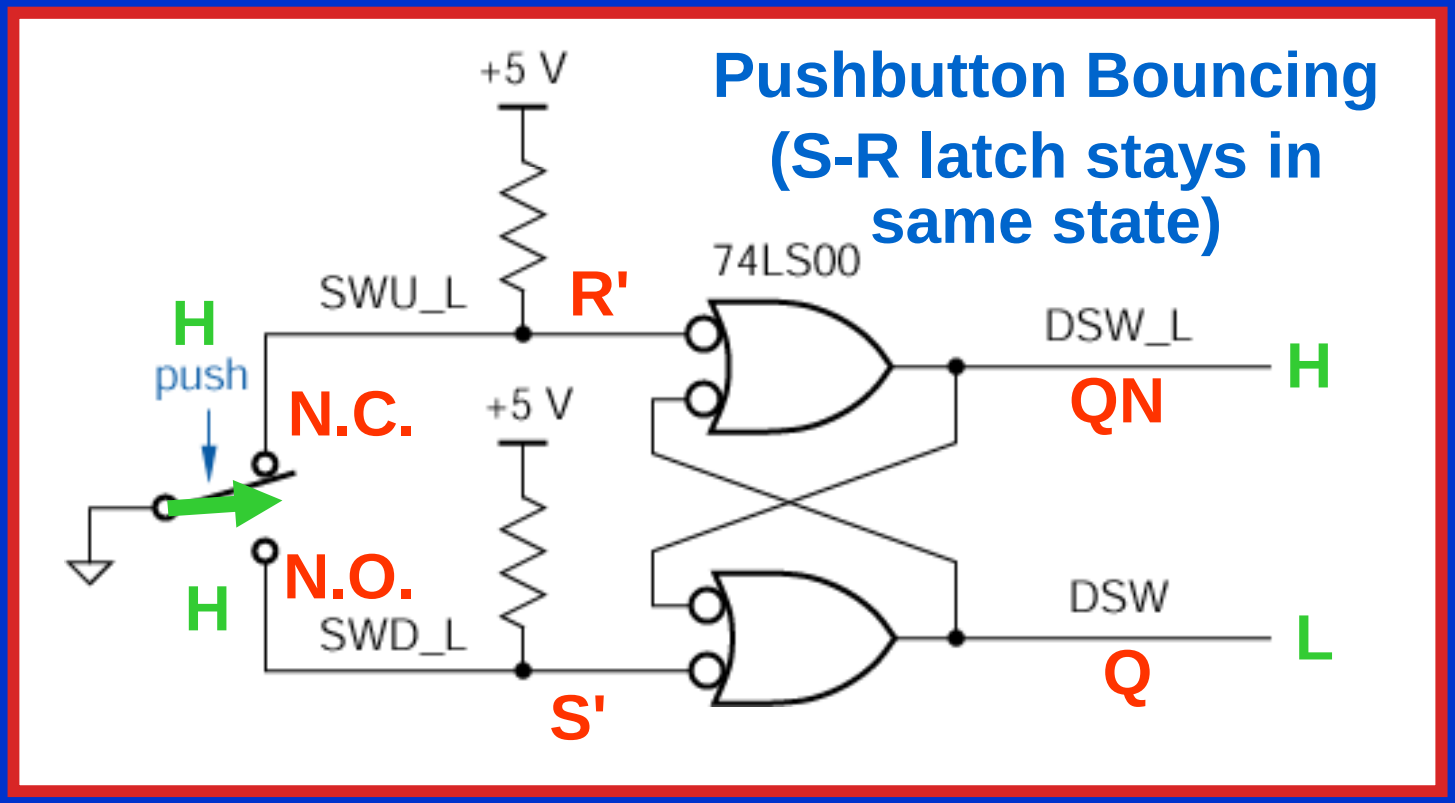
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



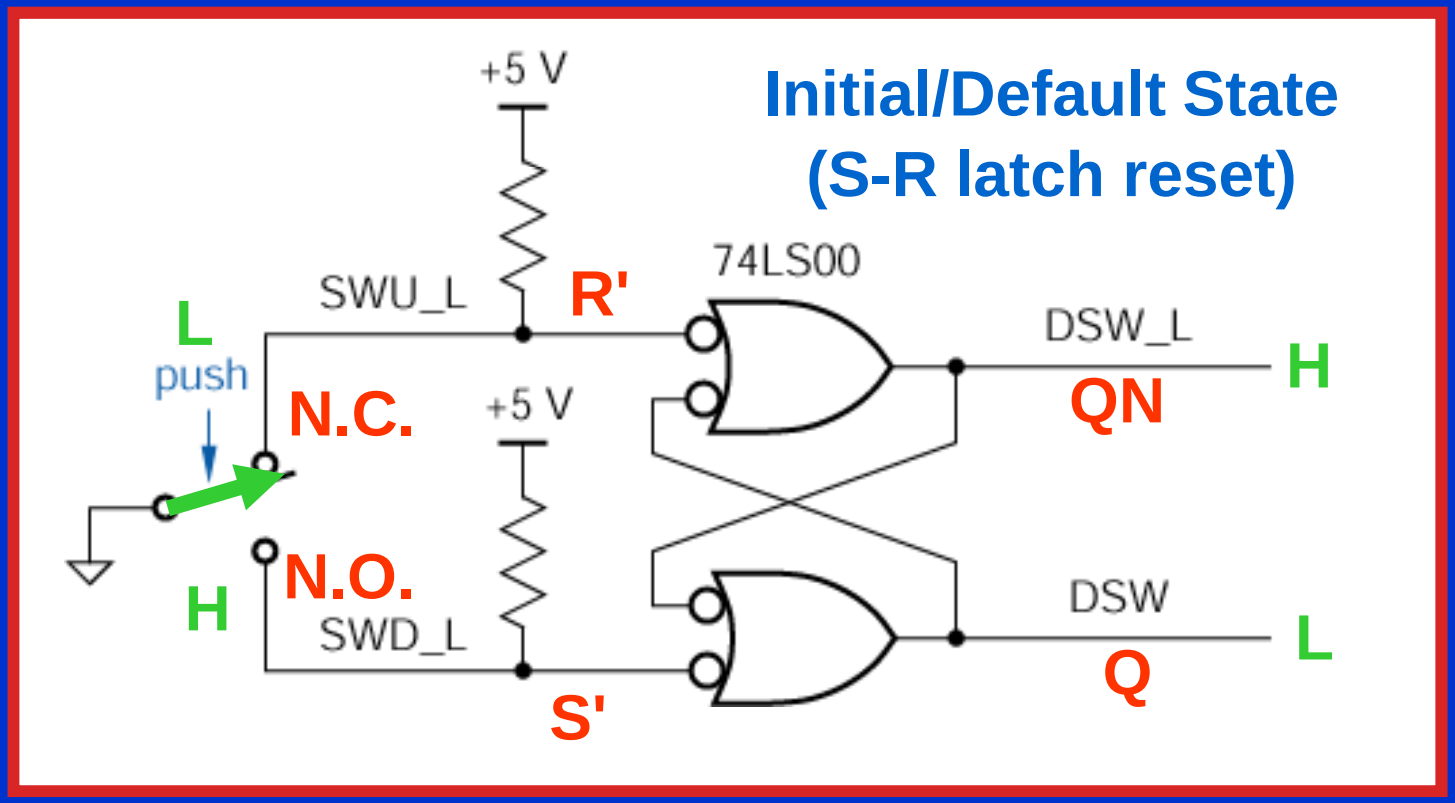
Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**



Classic Bounce-free Switch Circuit

- ***Bounce-free switch*** **implemented using S.P.D.T. ("single pole, double throw") pushbutton with an S′ R′ latch**

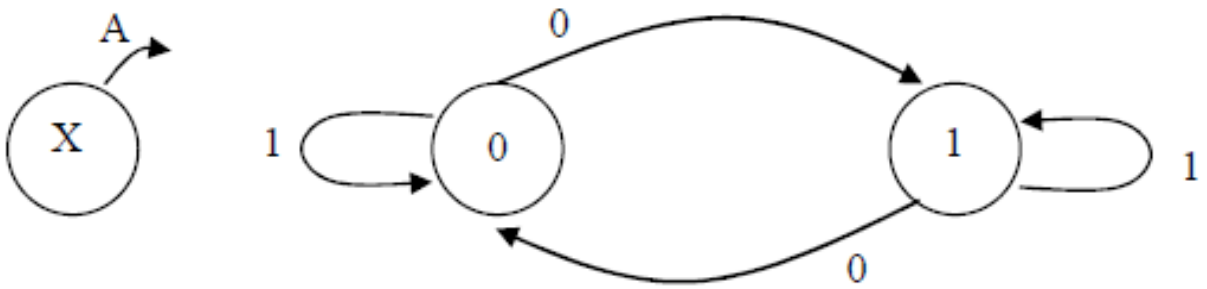


Example – Bounce-free Switch in ABEL

```
MODULE  bf_switch

TITLE 'Bounce-free Switch in ABEL'

DECLARATIONS
" Inputs are active low
!NO pin; " normally open switch contact
!NC pin; " normally closed switch contact

" Bounce-free clock output
BFC pin istype 'reg'; " can be a node instead of a pin

EQUATIONS
BFC.D = 0;
BFC.CLK = 0;
BFC.AP = NO;
BFC.AR = NC;

END
```

Here, we are essentially using the D flip-flop as an S-R latch via its asynchronous preset (.AP) and asynchronous reset (.AR) inputs

Clicker Quiz



Q1. The next state equation represented by the following state transition diagram is:

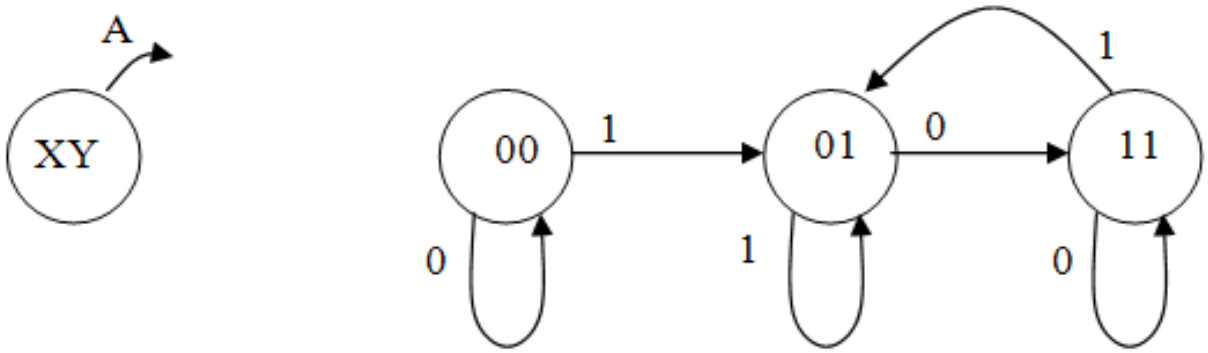
A. X* = A′·X′ + A·X

B. X* = A′·X + A·X′

C. X* = A + X

D. X* = A·X

E. none of the above



Q2. If designed for **minimum cost**, the next state equation for **X** is:

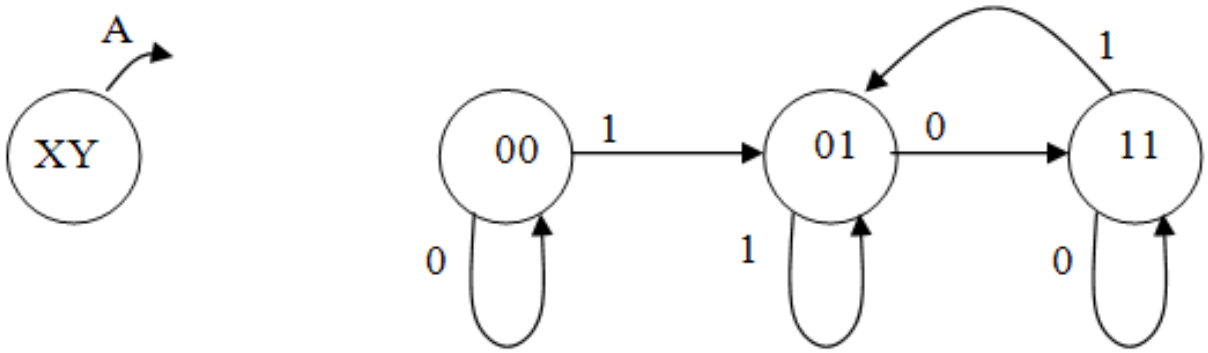
A. X* = A′·Y

B. X* = X + Y

C. X* = X′·Y + A′·X

D. X* = A′·Y + X·Y

E. none of the above



Q3. If designed for **minimum cost**, the next state equation for **Y** is:

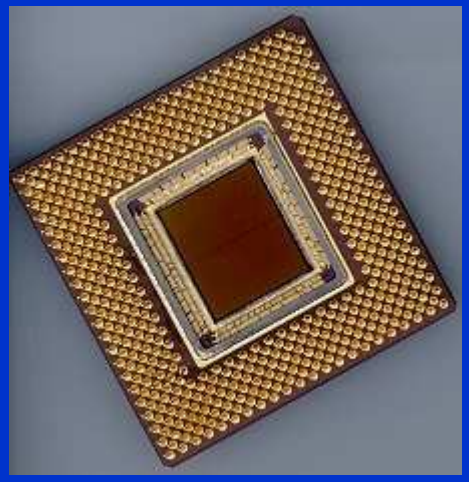
A. Y* = A′·Y

B. Y* = A + Y

C. Y* = X′·Y + A′·X

D. Y* = A′·Y + X·Y

E. none of the above



Introduction to Digital System Design

Module 3-F
State Machine Design Examples: Sequence Generators

Reading Assignment:

DDPP 4th Ed., pp. 566-576

Learning Objectives:

- **Design a clocked synchronous state machine and verify its operation**
- **Define minimum risk and minimum cost state machine design strategies, and discuss the tradeoffs between the two approaches**
- **Compare state assignment strategy and state machine model choice (Mealy vs. Moore) with respect to PLD resources (P-terms and macrocells) required for realization**

Outline

- Overview
- Simple character sequence display
- “Dual mode” moving dot / building dot sequence generator
  - Moore model realizations
  - Mealy model realizations
- Summary

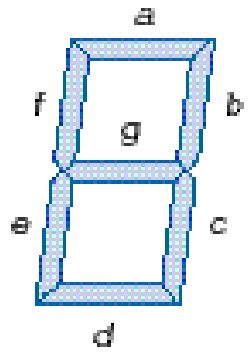
Overview

- A ***sequence generator*** state machine produces a (periodic) ***series of output signal assertions*** that constitute a ***pre-defined pattern:***
  - vehicle tail lights (e.g., “T-bird”)
  - traffic control signs (e.g., “blinkers” and stoplights)
  - character displays (e.g., “GO BOILERS”)
  - process control sequences (e.g., wash, rinse, dry)
- Either a Mealy or a Moore model can be used as the basis for designing a sequence generator
- Two different design strategies can be employed:
  - **minimum cost** – unused states are assumed to be don’t cares, potentially reducing realization cost while increasing risk of undefined behavior if machine gets into an unknown (unused) state
  - **minimum risk** – unused states are explicitly assigned a next state, eliminating risk of undefined behavior but potentially increasing realization cost

Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

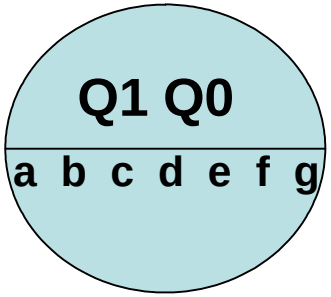
Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)

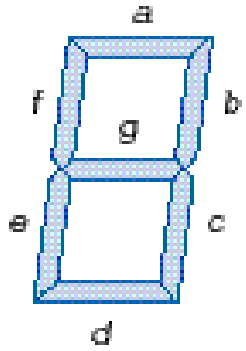


Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)



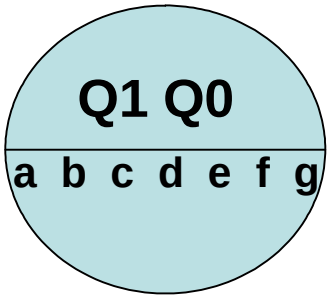


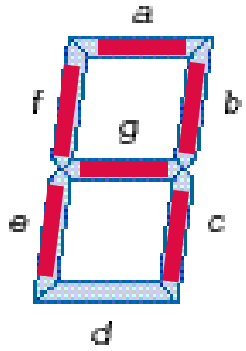
Only need 4 states

Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)





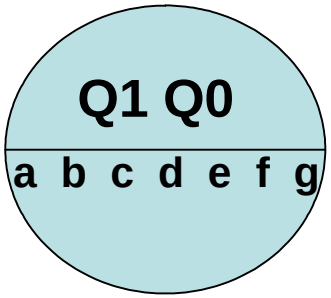
Only need 4 states

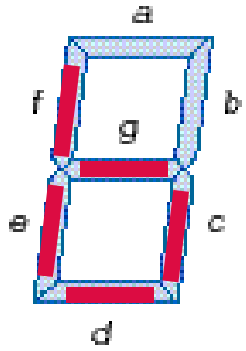
"A" = 1110111

Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)





Only need 4 states

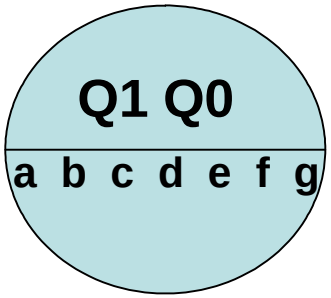
"A" = 1110111

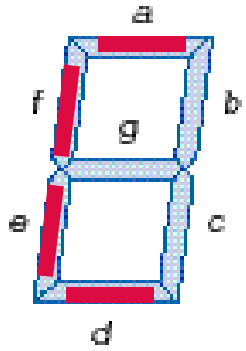
"b" = 0011111

Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)





Only need 4 states

"A" = 1110111

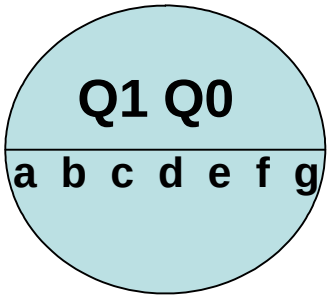
"b" = 0011111

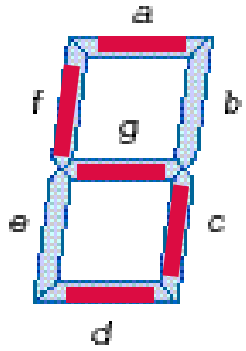
"C" = 1001110

Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)





Only need 4 states

"A" = 1110111

"b" = 0011111

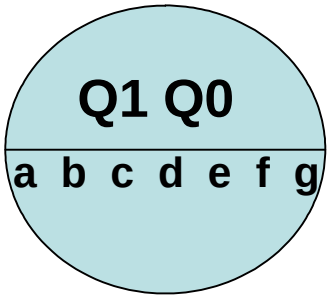
"C" = 1001110

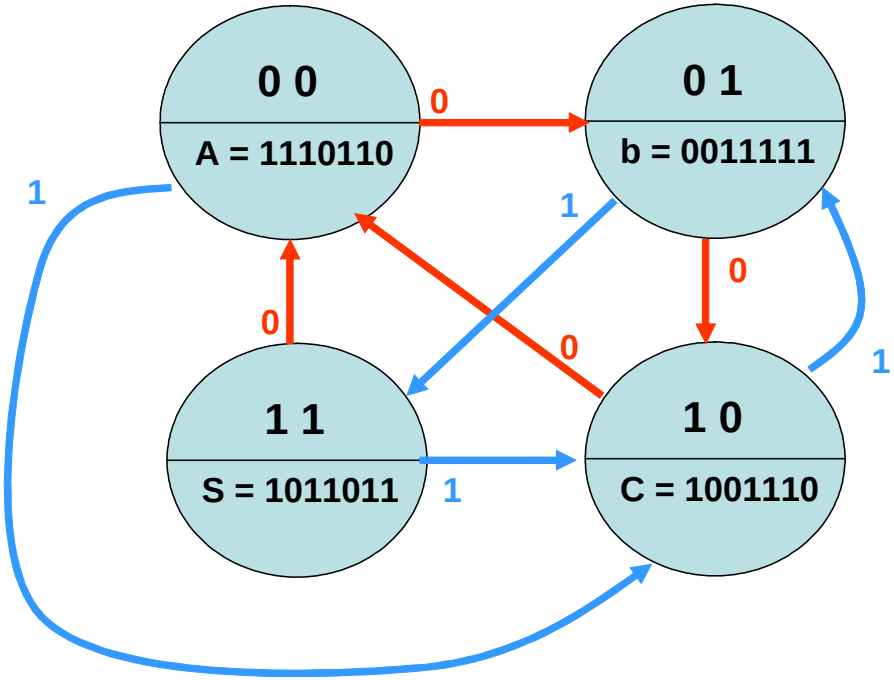
"S" = 1011011

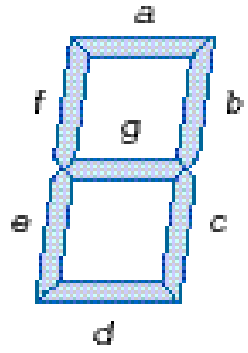
Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)







Only need 4 states

"A" = 1110111

"b" = 0011111

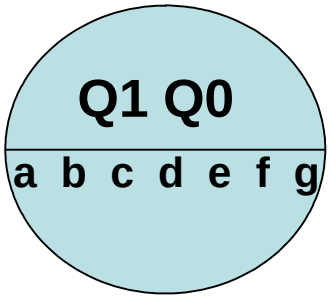
"C" = 1001110

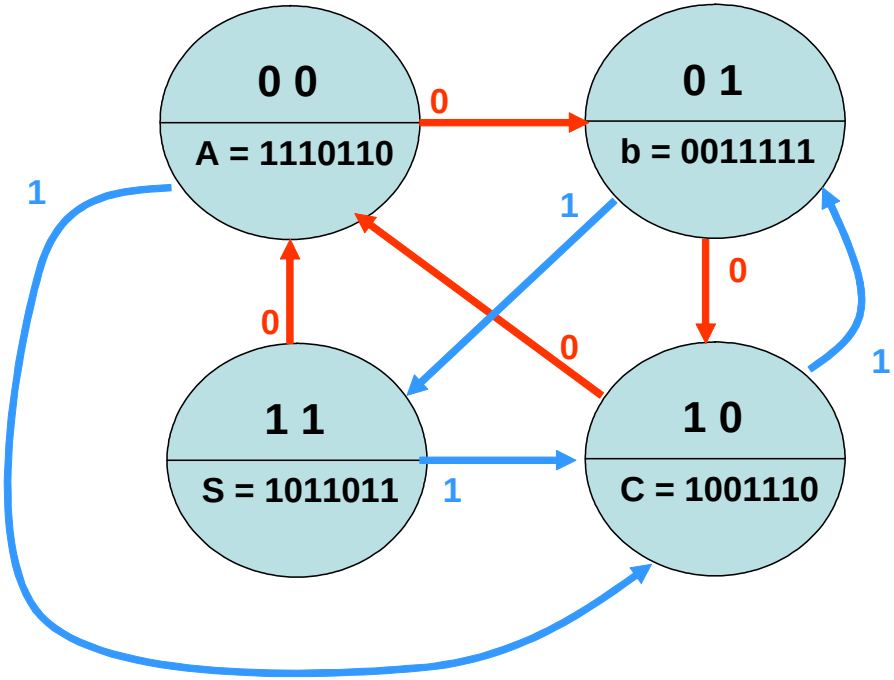
"S" = 1011011

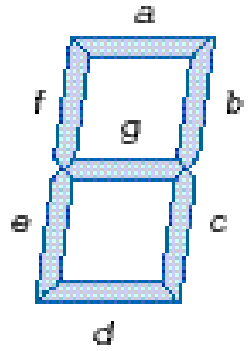
Example - Character Sequence Display

Design a circuit that produces the character sequence AbC or CbS on a 7-segment LED

Draw a *Moore* model state transition diagram. Note that there is one input (M) and seven active-low outputs (segments a-g)







Only need 4 states

"A" = 1110111

"b" = 0011111

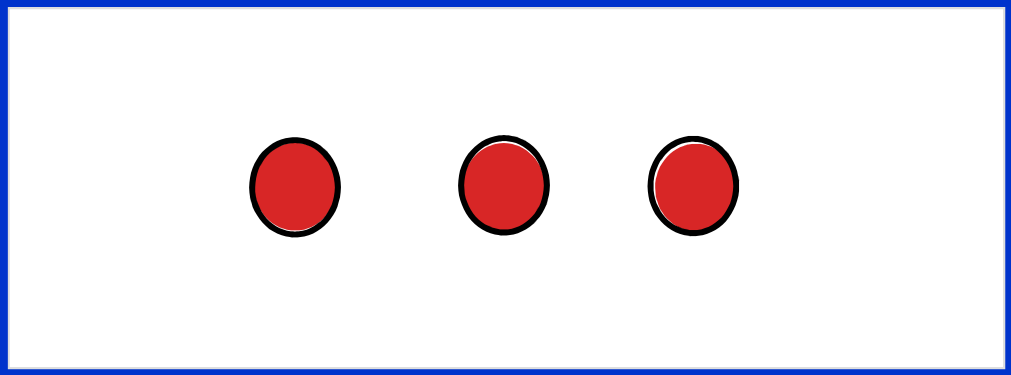
"C" = 1001110

"S" = 1011011

```
MODULE tv_disp
TITLE 'Character Sequence Display'
DECLARATIONS
CLOCK pin;
M pin;     " mode control
Q1..Q0 pin istype 'reg';
" 7-segment display outputs (common anode, active low)
!LA,!LB,!LC,!LD,!LE,!LF,!LG pin istype 'com';

TRUTH_TABLE ([Q1, Q0] -> [LA, LB, LC, LD, LE, LF, LG])
             [ 0, 0 ] -> [ 1,  1,  1,  0,  1,  1,  0];  " A
             [ 0, 1 ] -> [ 0,  0,  1,  1,  1,  1,  1];  " b
             [ 1, 0 ] -> [ 1,  0,  0,  1,  1,  1,  0];  " C
             [ 1, 1 ] -> [ 1,  0,  1,  1,  0,  1,  1];  " S

TRUTH_TABLE ([Q1, Q0,  M] :> [Q1, Q0])
             [ 0,  0,  0] :> [ 0,  1];
             [ 0,  0,  1] :> [ 1,  0];
             [ 0,  1,  0] :> [ 1,  0];
             [ 0,  1,  1] :> [ 1,  1];
             [ 1,  0,  0] :> [ 0,  0];
             [ 1,  0,  1] :> [ 0,  1];
             [ 1,  1,  0] :> [ 0,  0];
             [ 1,  1,  1] :> [ 1,  0];
EQUATIONS
[Q1..Q0].CLK = CLOCK;
END
```

Example – Dual Mode Light Sequencer

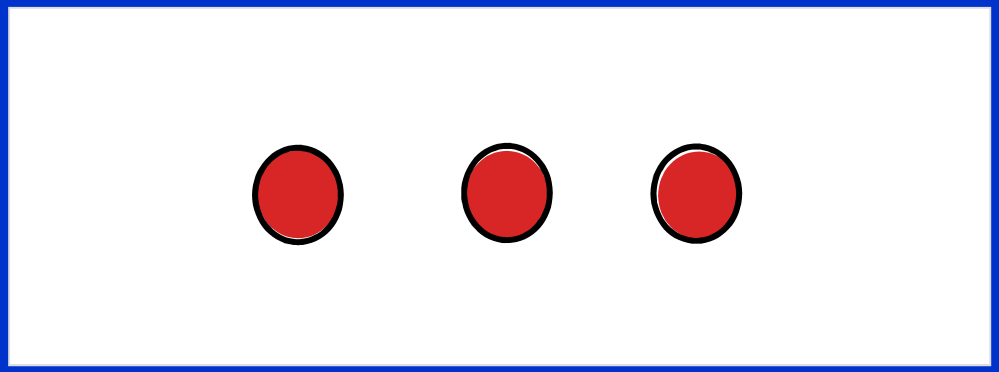
- **Design a clocked synchronous state machine that generates the following "light patterns" (using three LEDs)**



Mode 0: "single dot, left-to-right"

Example – Dual Mode Light Sequencer

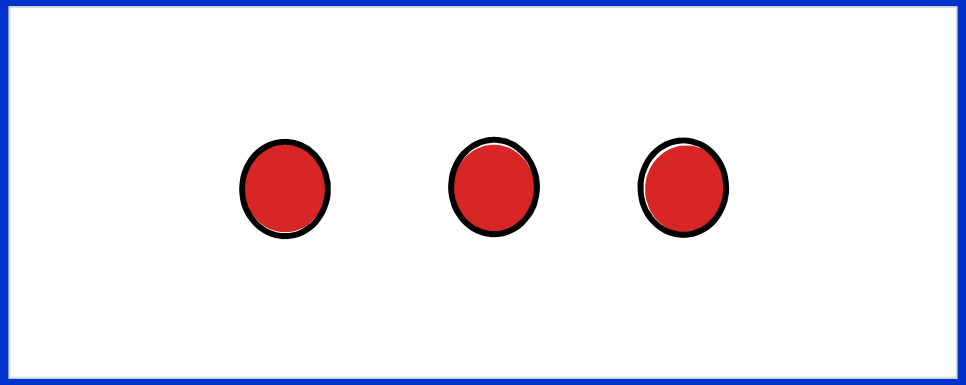
- **Design a clocked synchronous state machine that generates the following "light patterns" (using three LEDs)**



Mode 1: "single dot, right-to-left"

Example – Dual Mode Light Sequencer

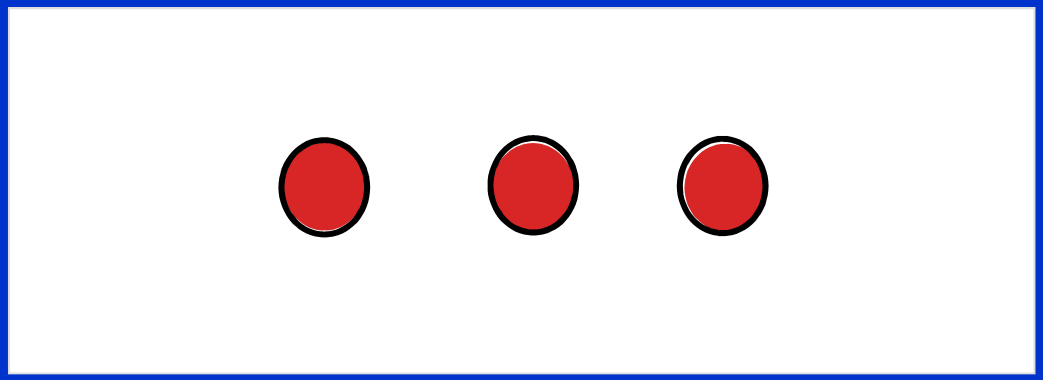
- **Design a clocked synchronous state machine that generates the following "light patterns" (using three LEDs)**



Mode 2: "building dots, left-to-right"

Example – Dual Mode Light Sequencer

- **Design a clocked synchronous state machine that generates the following "light patterns" (using three LEDs)**



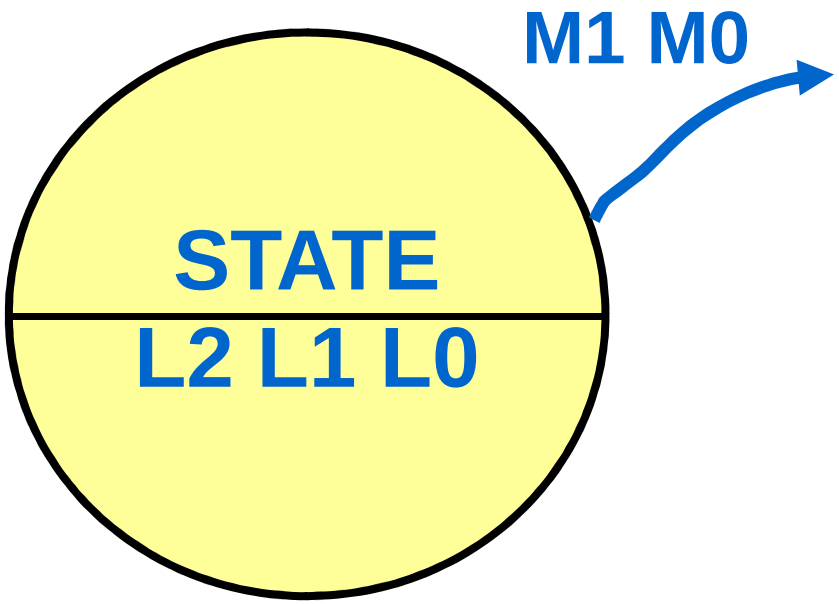
Mode 3: "building dots, right-to-left"

Moore Model Realizations

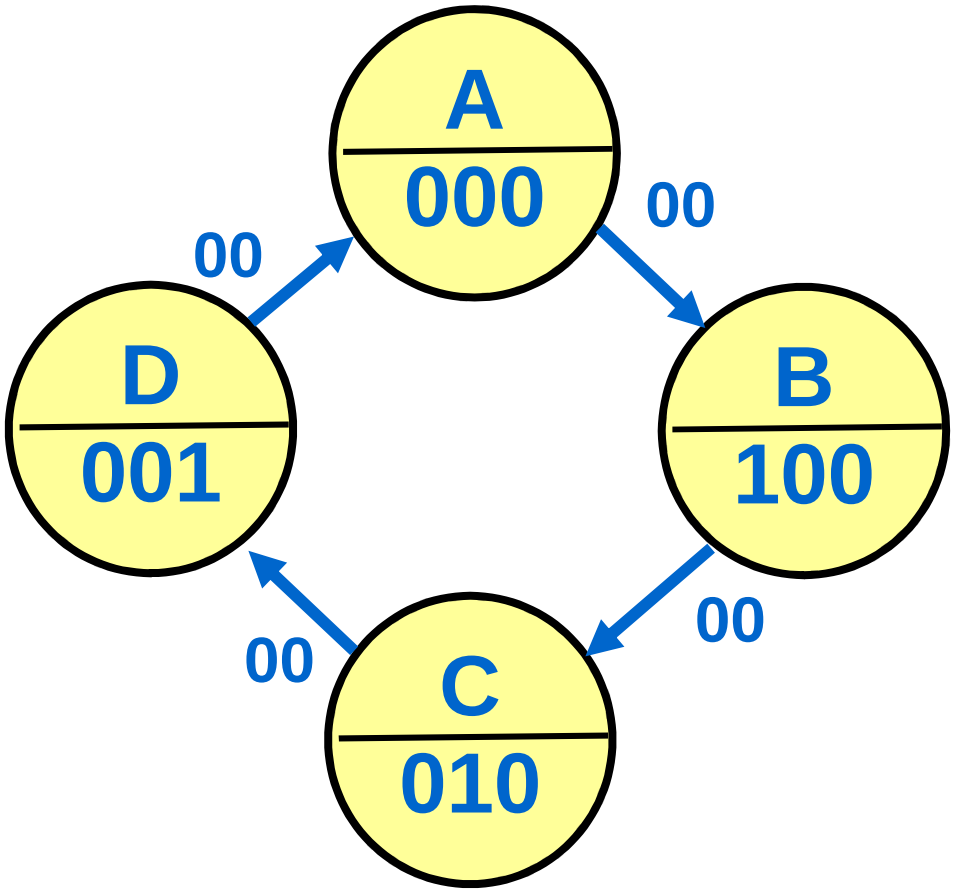
- To specify in which of the 4 modes we want the circuit to operate, we will need 2 "mode control" inputs, **M1** and **M0**, where:
  - 0 0 → single dot, left-to-right
  - 0 1 → single dot, right-to-left
  - 1 0 → building dots, left-to-right
  - 1 1 → building dots, right-to-left
- A separate output function needs to be determined for each of the 3 LED outputs: **L2**, **L1**, and **L0** (from left-to-right)
- A state will be needed corresponding to the **"all LEDs off"** condition

STEP 1: Construct a state transition diagram

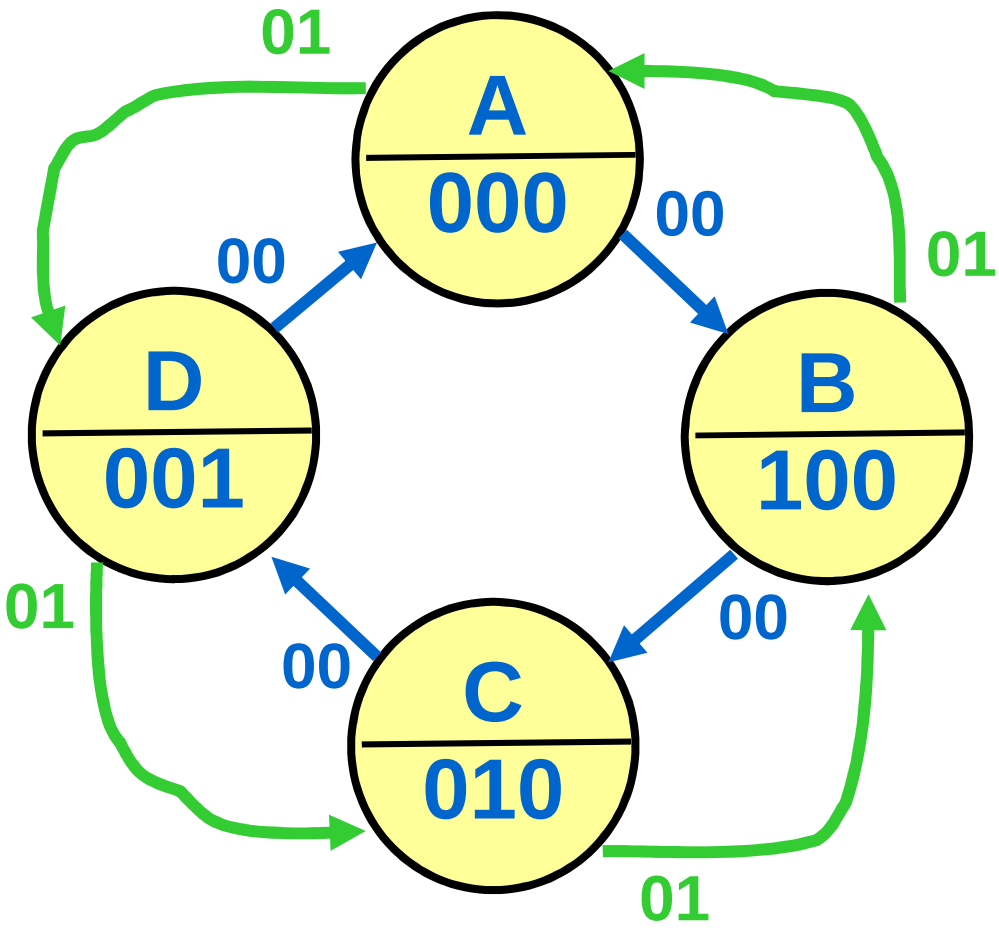
Moore Model:



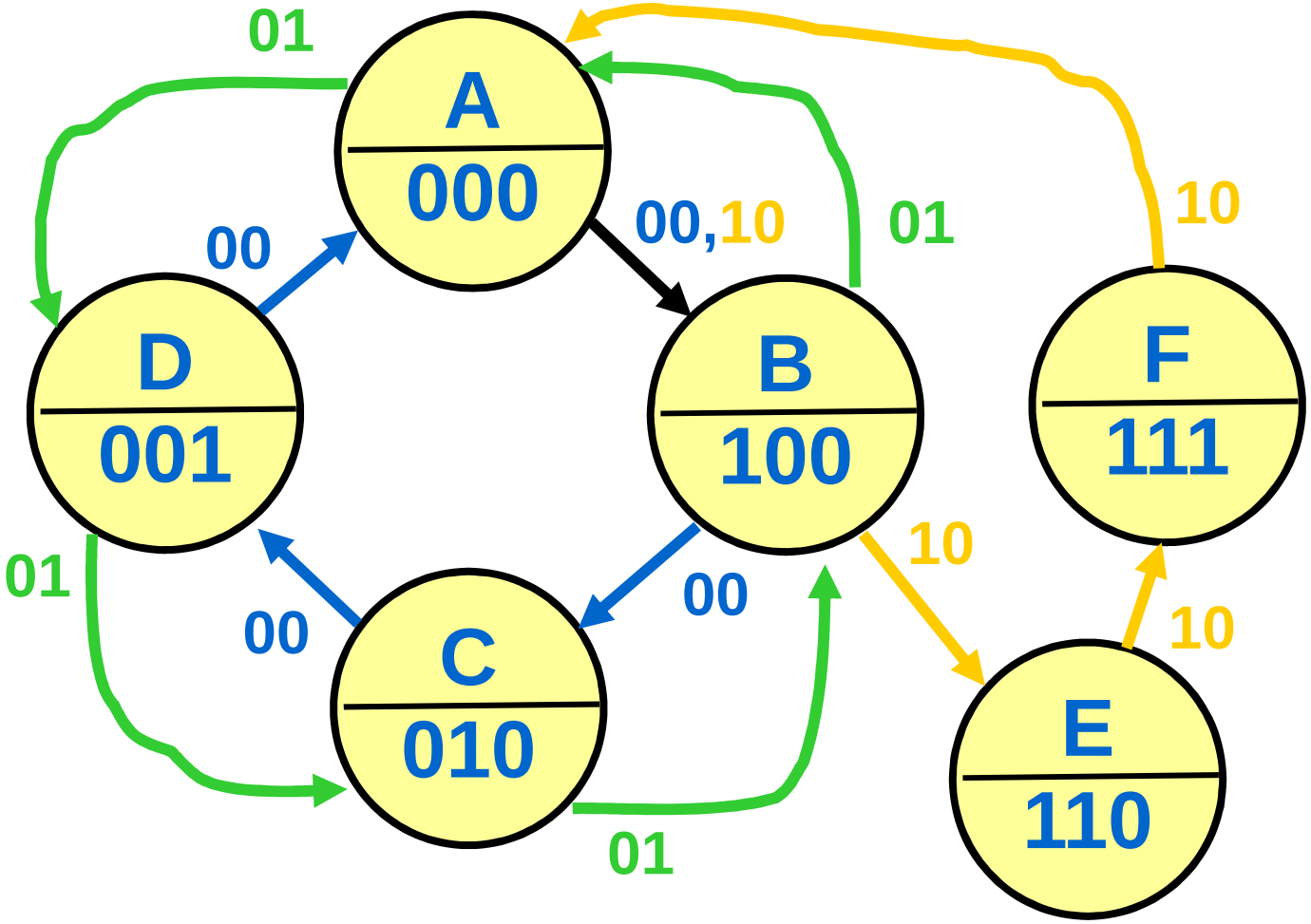
STEP 1: Construct a state transition diagram



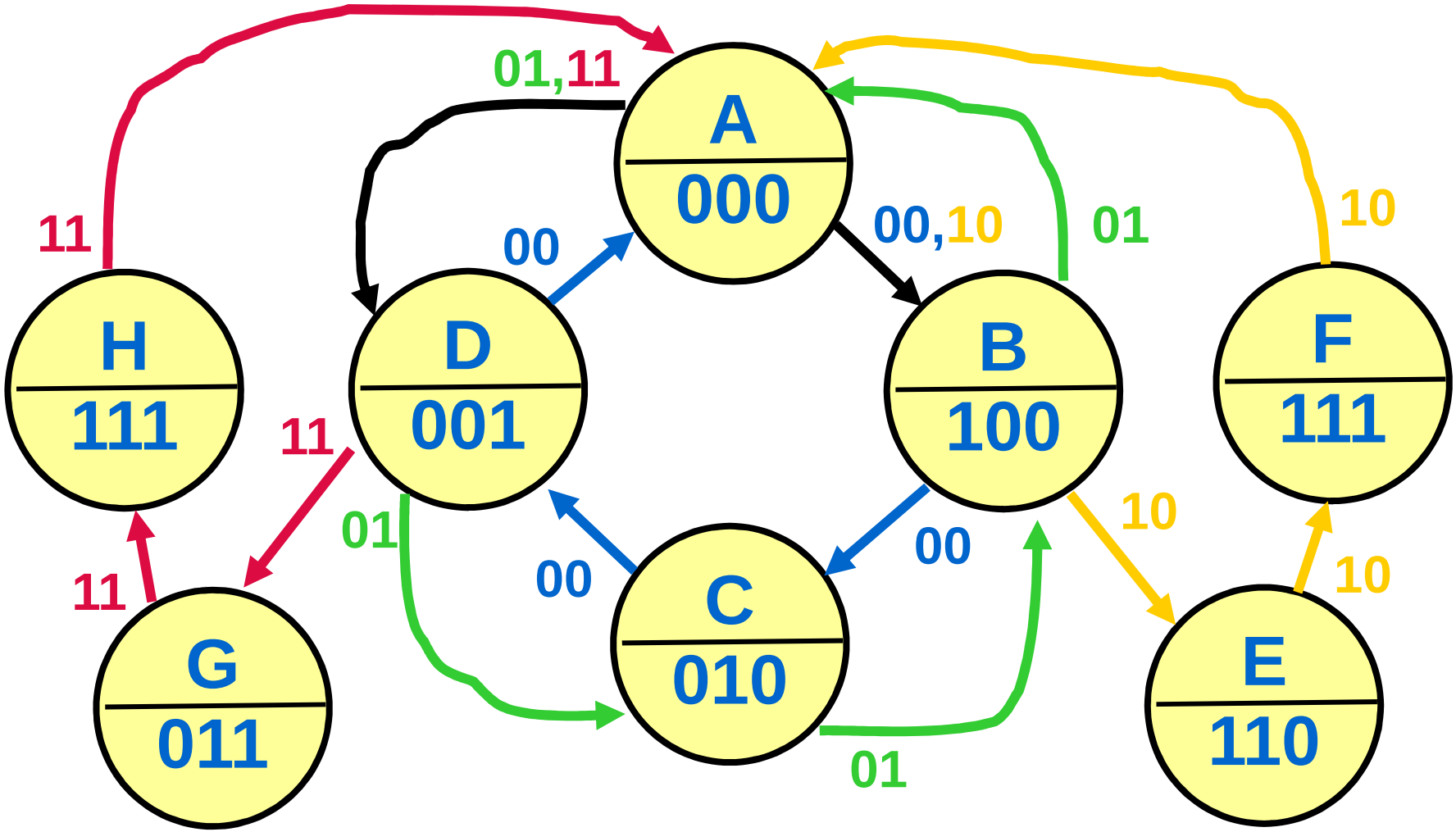
STEP 1: Construct a state transition diagram



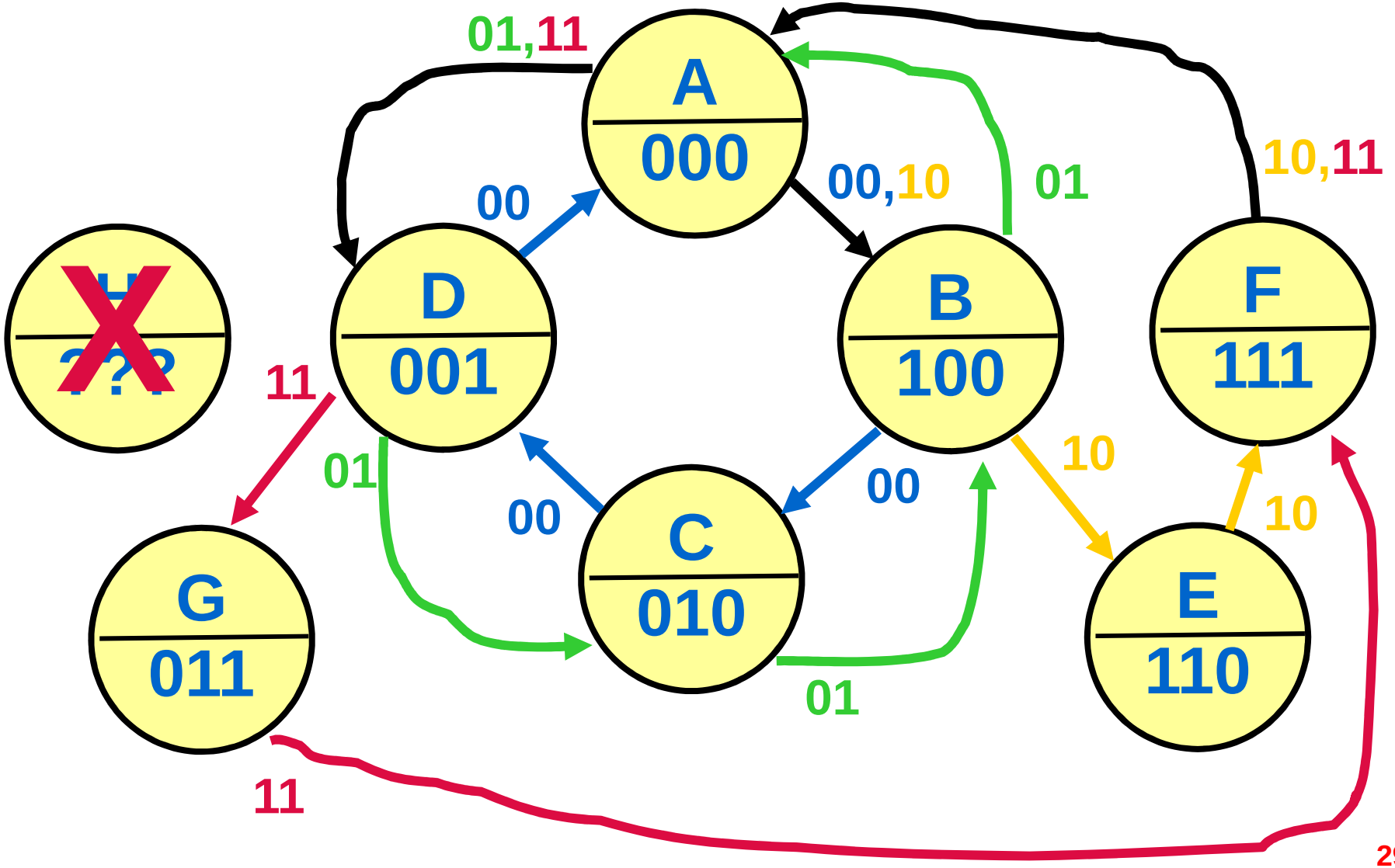
STEP 1: Construct a state transition diagram



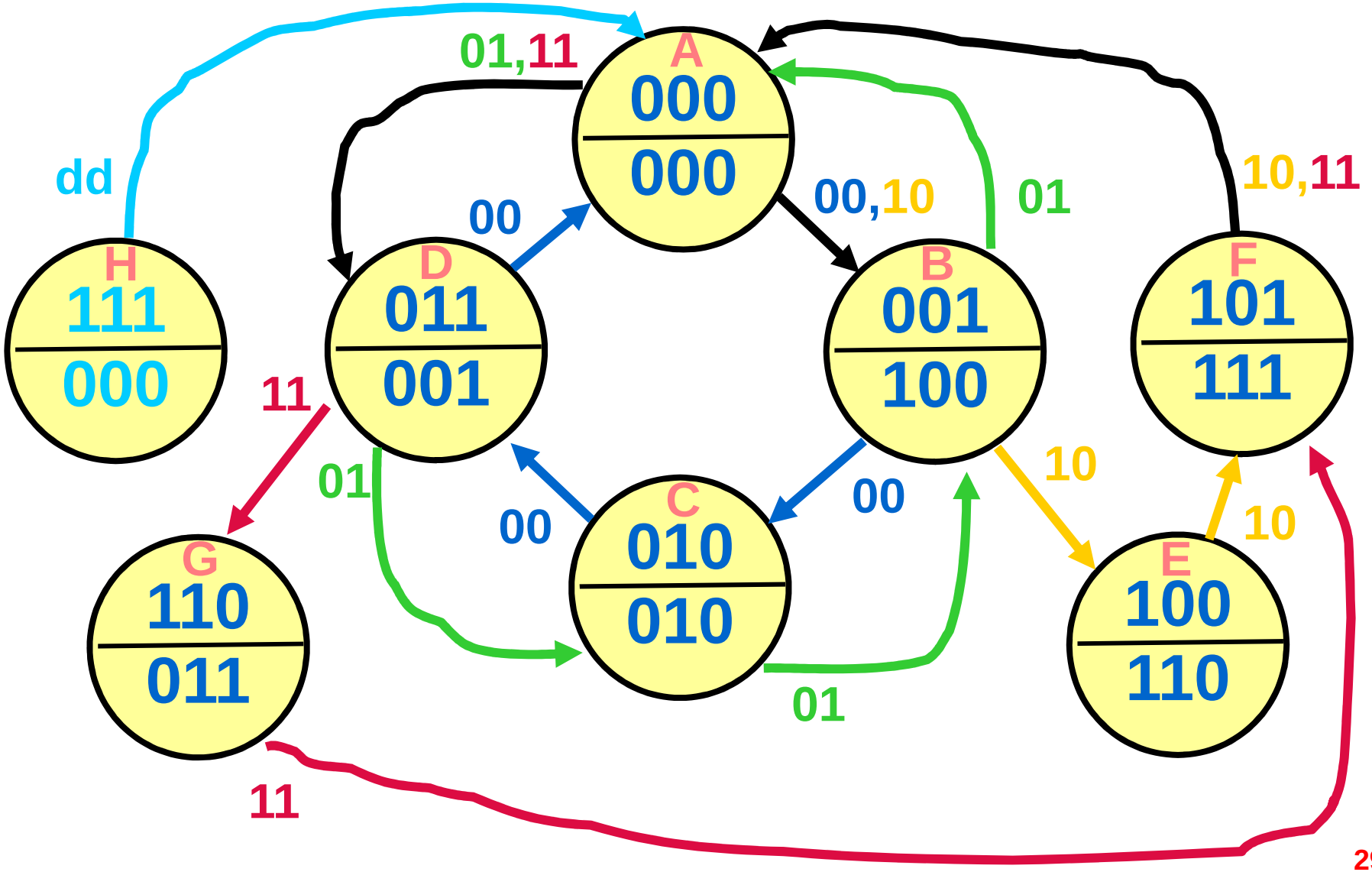
STEP 1: Construct a state transition diagram



STEP 2: Minimize the number of states



STEP 3: Assign state variable combinations



STEP 4: Construct a PS-NS/PO Table

PS			PI		NS			PO		
Q2	Q1	Q0	M1	M0	Q2*	Q1*	Q0*	L2	L1	L0
0	0	0	0	0	0	0	1	0	0	0
			0	1	0	1	1			
			1	0	0	0	1			
			1	1	0	1	1			
0	0	1	0	0	0	1	0	1	0	0
			0	1	0	0	0			
			1	0	1	0	0			
			1	1	0	0	0			
0	1	0	0	0	0	1	1	0	1	0
			0	1	0	0	1			
			1	0	0	0	0			
			1	1	0	0	0			
0	1	1	0	0	0	0	0	0	0	1
			0	1	0	1	0			
			1	0	0	0	0			
			1	1	1	1	0			

STEP 4: Construct a PS-NS/PO Table...

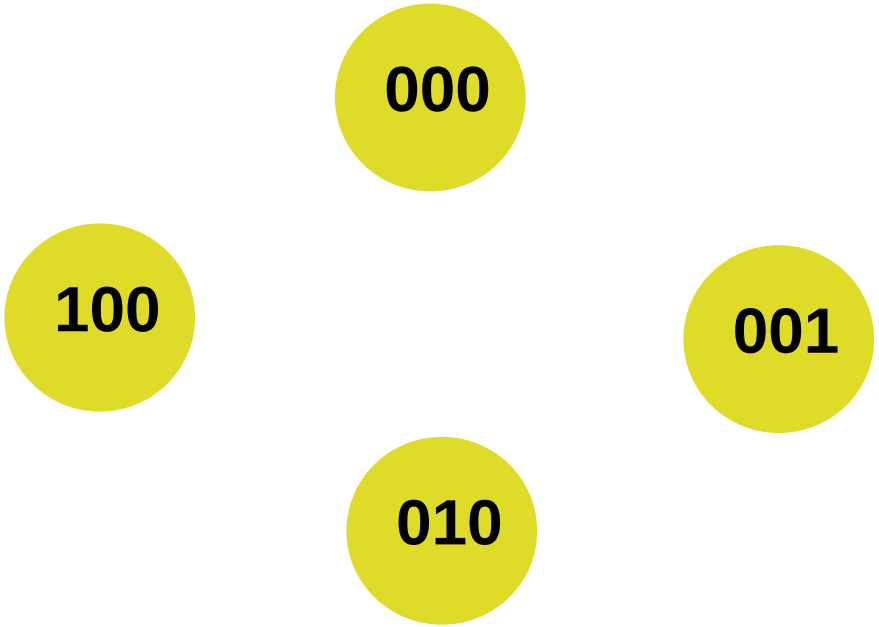
PS			PI		NS			PO		
Q2	Q1	Q0	M1	M0	Q2*	Q1*	Q0*	L2	L1	L0
1	0	0	0	0	0	0	0	1	1	0
			0	1	0	0	0			
			1	0	1	0	1			
			1	1	0	0	0			
1	0	1	0	0	0	0	0	1	1	1
			0	1	0	0	0			
			1	0	0	0	0			
			1	1	0	0	0			
1	1	0	0	0	0	0	0	0	1	1
			0	1	0	0	0			
			1	0	0	0	0			
			1	1	1	0	1			
1	1	1	0	0	0	0	0	0	0	0
			0	1	0	0	0			
			1	0	0	0	0			
			1	1	0	0	0			

```
MODULE moorelsA
TITLE 'Light Sequencer – Moore Model A'

DECLARATIONS
CLOCK pin;
M0, M1 pin;
Q2, Q1, Q0 pin istype 'reg';
L2, L1, L0 pin istype 'com';

truth_table ([Q2,Q1,Q0,M1,M0]:>[Q2,Q1,Q0])

             [ 0, 0, 0, 0, 0]:>[ 0, 0, 1];
             [ 0, 0, 0, 0, 1]:>[ 0, 1, 1];
             [ 0, 0, 0, 1, 0]:>[ 0, 0, 1];
             [ 0, 0, 0, 1, 1]:>[ 0, 1, 1];

             [ 0, 0, 1, 0, 0]:>[ 0, 1, 0];
             [ 0, 0, 1, 0, 1]:>[ 0, 0, 0];
             [ 0, 0, 1, 1, 0]:>[ 1, 0, 0];
             [ 0, 0, 1, 1, 1]:>[ 0, 0, 0];

             [ 0, 1, 0, 0, 0]:>[ 0, 1, 1];
             [ 0, 1, 0, 0, 1]:>[ 0, 0, 1];
             [ 0, 1, 0, 1, 0]:>[ 0, 0, 0];
             [ 0, 1, 0, 1, 1]:>[ 0, 0, 0];

             [ 0, 1, 1, 0, 0]:>[ 0, 0, 0];
             [ 0, 1, 1, 0, 1]:>[ 0, 1, 0];
             [ 0, 1, 1, 1, 0]:>[ 0, 0, 0];
             [ 0, 1, 1, 1, 1]:>[ 1, 1, 0];
```

This realization uses 6 macrocells

```
             [ 1, 0, 0, 0, 0]:>[ 0, 0, 0];
             [ 1, 0, 0, 0, 1]:>[ 0, 0, 0];
             [ 1, 0, 0, 1, 0]:>[ 1, 0, 1];
             [ 1, 0, 0, 1, 1]:>[ 0, 0, 0];

             [ 1, 0, 1, 0, 0]:>[ 0, 0, 0];
             [ 1, 0, 1, 0, 1]:>[ 0, 0, 0];
             [ 1, 0, 1, 1, 0]:>[ 0, 0, 0];
             [ 1, 0, 1, 1, 1]:>[ 0, 0, 0];

             [ 1, 1, 0, 0, 0]:>[ 0, 0, 0];
             [ 1, 1, 0, 0, 1]:>[ 0, 0, 0];
             [ 1, 1, 0, 1, 0]:>[ 0, 0, 0];
             [ 1, 1, 0, 1, 1]:>[ 1, 0, 1];

             [ 1, 1, 1, 0, 0]:>[ 0, 0, 0];
             [ 1, 1, 1, 0, 1]:>[ 0, 0, 0];
             [ 1, 1, 1, 1, 0]:>[ 0, 0, 0];
             [ 1, 1, 1, 1, 1]:>[ 0, 0, 0];

truth_table ([Q2,Q1,Q0]->[L2,L1,L0])
             [ 0, 0 ,0]->[ 0, 0, 0];
             [ 0, 0, 1]->[ 1, 0, 0];
             [ 0, 1, 0]->[ 0, 1, 0];
             [ 0, 1, 1]->[ 0, 0, 1];
             [ 1, 0, 0]->[ 1, 1, 0];
             [ 1, 0, 1]->[ 1, 1, 1];
             [ 1, 1, 0]->[ 0, 1, 1];
             [ 1, 1, 1]->[ 0, 0, 0];

EQUATIONS
[Q2..Q0].CLK = CLOCK;
END
```

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



00: Dot L → R

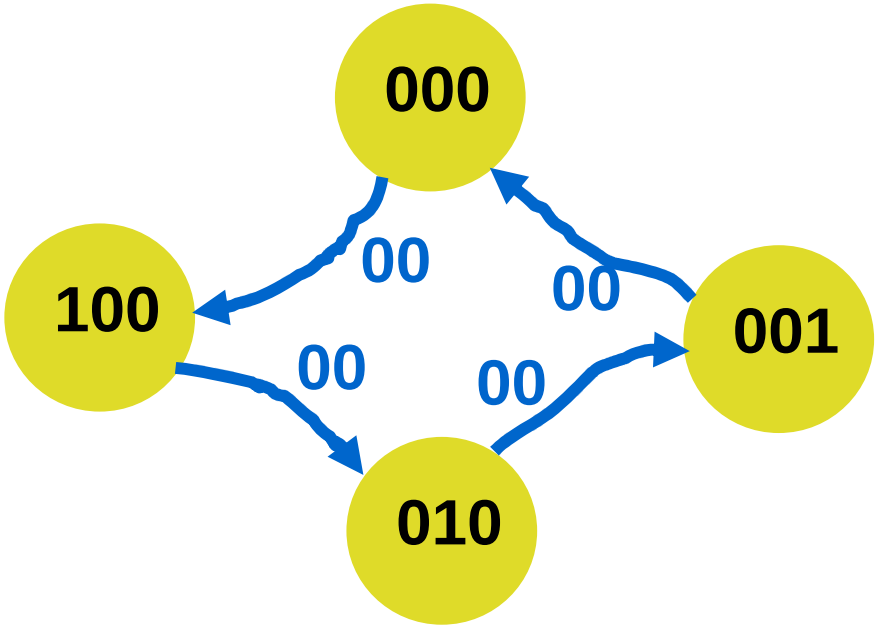
01: Dot L ← R

10: Build L → R

11: Build L ← R

Here, let the output functions be the state assignment

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



00: Dot L → R

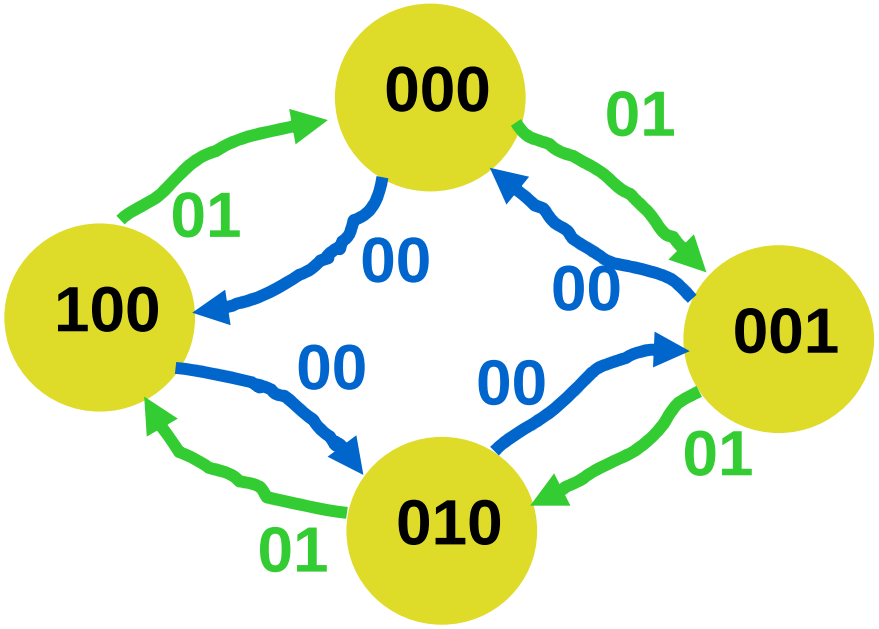
01: Dot L ← R

10: Build L → R

11: Build L ← R

Here, let the output functions be the state assignment

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



00: Dot L → R

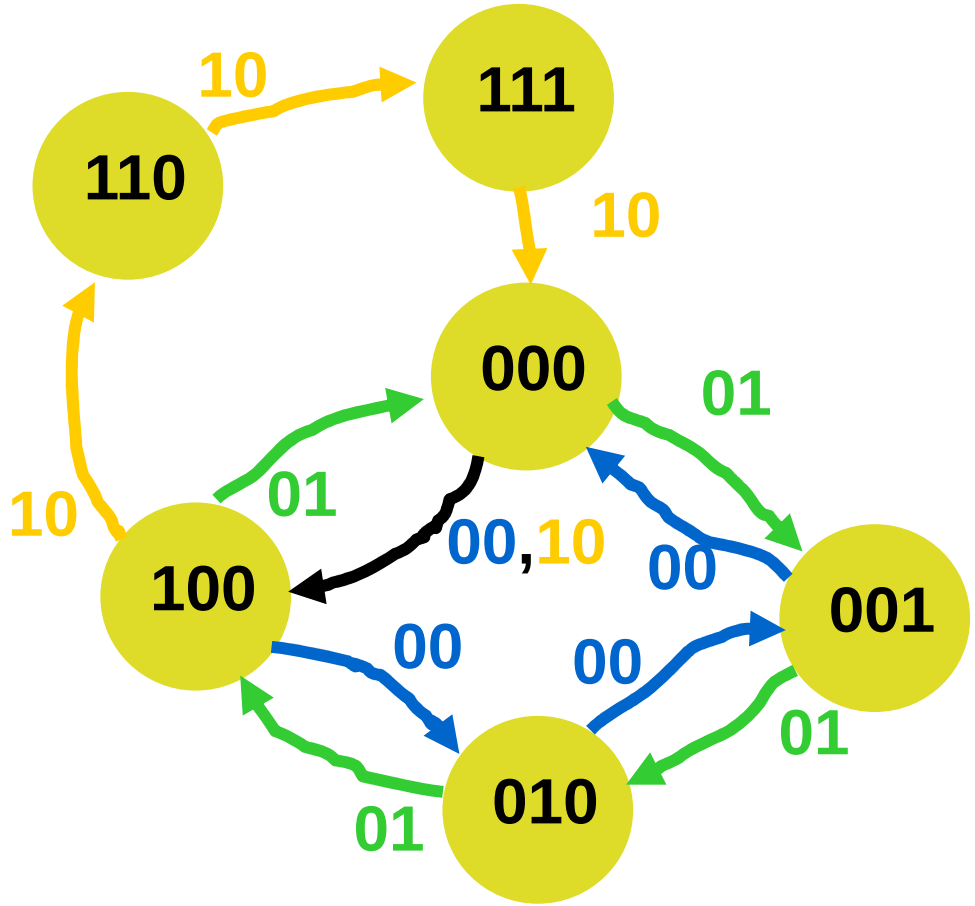
01: Dot L ← R

10: Build L → R

11: Build L ← R

Here, let the output functions be the state assignment

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



00: Dot L → R

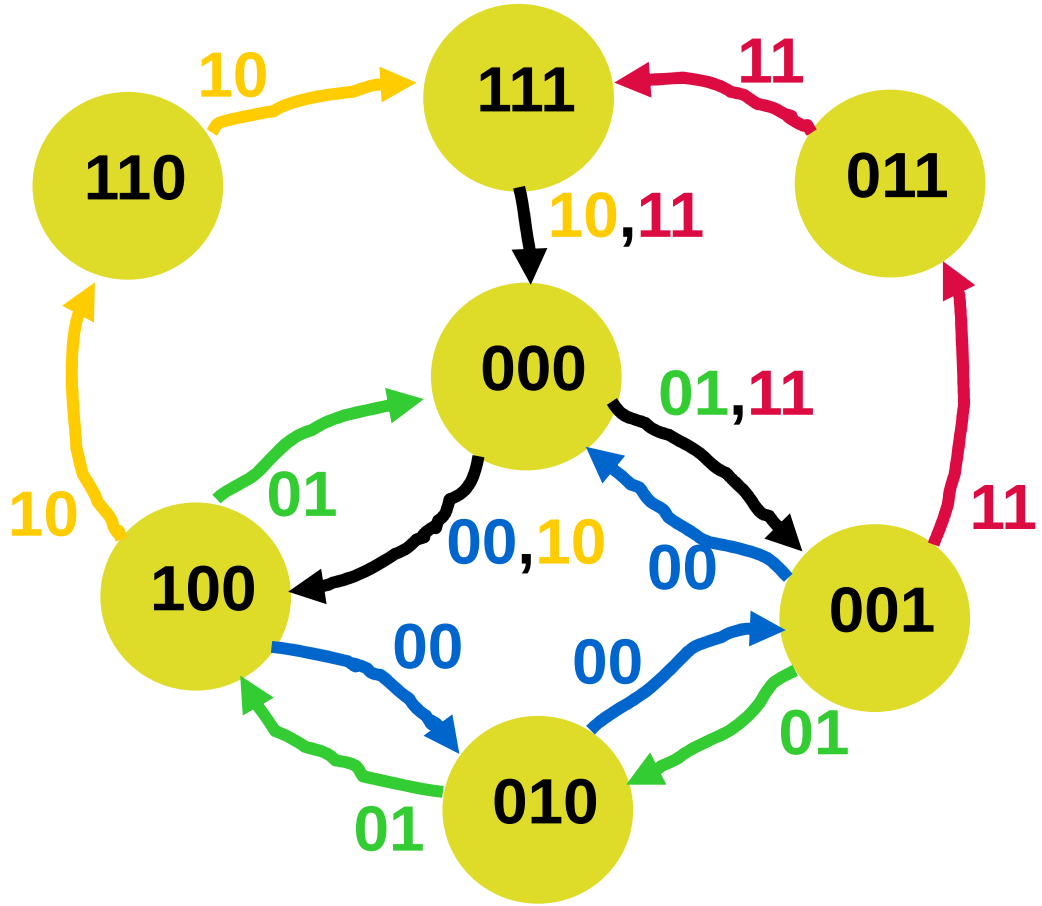
01: Dot L ← R

10: Build L → R

11: Build L ← R

Here, let the output function be the state assignment

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



00: Dot L → R

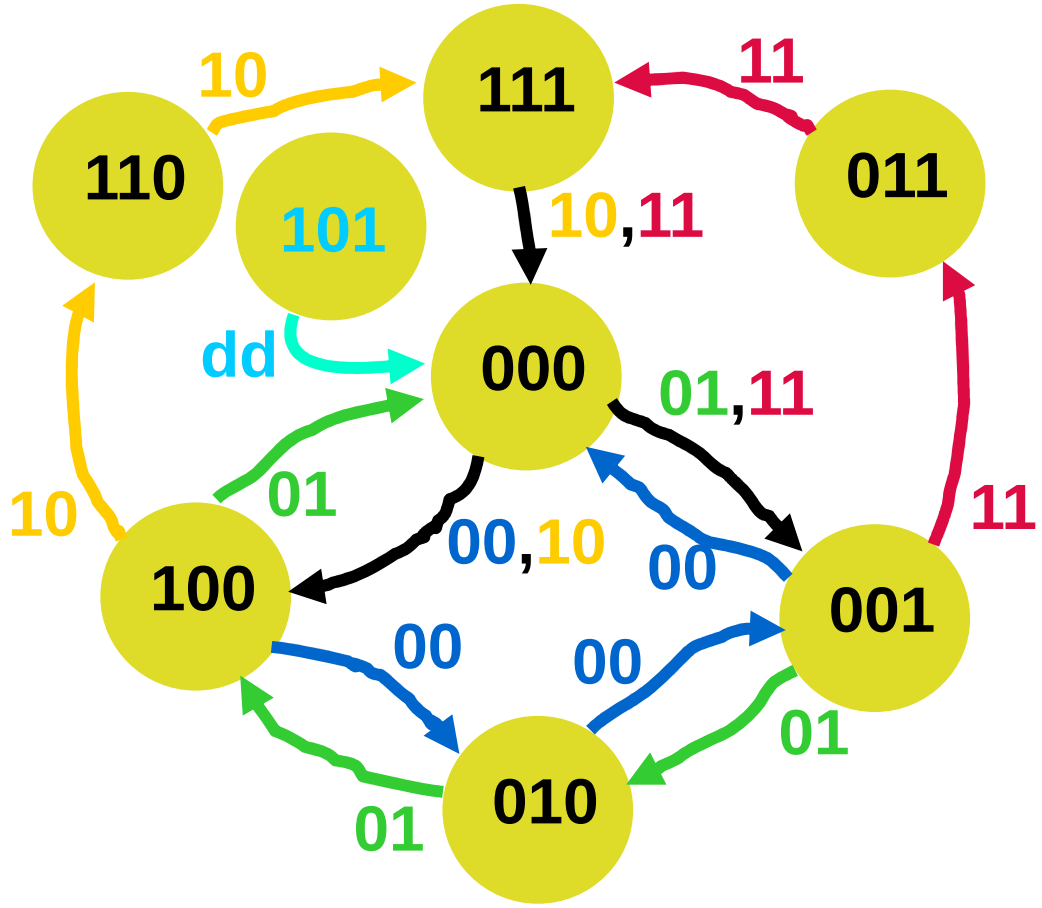
01: Dot L ← R

10: Build L → R

11: Build L ← R

Here, let the output functions be the state assignment

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



00: Dot L → R

01: Dot L ← R

10: Build L → R

11: Build L ← R

Here, let the output functions be the state assignment

```
MODULE moorelsB
TITLE 'Light Sequencer – Moore Model B'

DECLARATIONS
CLOCK pin;
M0, M1 pin;
Q2, Q1, Q0 pin istype 'reg'; "(serve as L2, L1, L0)

truth_table ([Q2,Q1,Q0,M1,M0]:>[Q2,Q1,Q0])

             [ 0, 0, 0, 0, 0]:>[ 1, 0, 0];
             [ 0, 0, 0, 0, 1]:>[ 0, 0, 1];
             [ 0, 0, 0, 1, 0]:>[ 1, 0, 0];
             [ 0, 0, 0, 1, 1]:>[ 0, 0, 1];

             [ 0, 0, 1, 0, 0]:>[ 0, 0, 0];
             [ 0, 0, 1, 0, 1]:>[ 0, 1, 0];
             [ 0, 0, 1, 1, 0]:>[ 0, 0, 0];
             [ 0, 0, 1, 1, 1]:>[ 0, 1, 1];

             [ 0, 1, 0, 0, 0]:>[ 0, 0, 1];
             [ 0, 1, 0, 0, 1]:>[ 1, 0, 0];
             [ 0, 1, 0, 1, 0]:>[ 0, 0, 0];
             [ 0, 1, 0, 1, 1]:>[ 0, 0, 0];

             [ 0, 1, 1, 0, 0]:>[ 0, 0, 0];
             [ 0, 1, 1, 0, 1]:>[ 0, 0, 0];
             [ 0, 1, 1, 1, 0]:>[ 0, 0, 0];
             [ 0, 1, 1, 1, 1]:>[ 1, 1, 1];
```

This realization uses 3 macrocells

Note: Here the output functions are merely the state variables – L2=Q2, L1=Q1, L0=Q0

```
             [ 1, 0, 0, 0, 0]:>[ 0, 1, 0];
             [ 1, 0, 0, 0, 1]:>[ 0, 0, 0];
             [ 1, 0, 0, 1, 0]:>[ 1, 1, 0];
             [ 1, 0, 0, 1, 1]:>[ 0, 0, 0];

             [ 1, 0, 1, 0, 0]:>[ 0, 0, 0];
             [ 1, 0, 1, 0, 1]:>[ 0, 0, 0];
             [ 1, 0, 1, 1, 0]:>[ 0, 0, 0];
             [ 1, 0, 1, 1, 1]:>[ 0, 0, 0];

             [ 1, 1, 0, 0, 0]:>[ 0, 0, 0];
             [ 1, 1, 0, 0, 1]:>[ 0, 0, 0];
             [ 1, 1, 0, 1, 0]:>[ 1, 1, 0];
             [ 1, 1, 0, 1, 1]:>[ 0, 0, 0];

             [ 1, 1, 1, 0, 0]:>[ 0, 0, 0];
             [ 1, 1, 1, 0, 1]:>[ 0, 0, 0];
             [ 1, 1, 1, 1, 0]:>[ 0, 0, 0];
             [ 1, 1, 1, 1, 1]:>[ 0, 0, 0];

EQUATIONS
[Q2..Q0].CLK = CLOCK;
END
```

```
MODULE moorelsB_sd

TITLE 'Light Sequencer Using State Diagram'

M1, M0 pin;
CLOCK pin;
Q2, Q1, Q0 pin istype 'reg';

QALL = [Q2,Q1,Q0];
A0   = [ 0, 0, 0];
A1   = [ 0, 0, 1];
A2   = [ 0, 1, 0];
A3   = [ 0, 1, 1];
A4   = [ 1, 0, 0];
A5   = [ 1, 0, 1];
A6   = [ 1, 1, 0];
A7   = [ 1, 1, 1];

STATE_DIAGRAM QALL

state A0:      if (M1==0)&(M0==0) then A4
          else if (M1==0)&(M0==1) then A1
          else if (M1==1)&(M0==0) then A4
          else if (M1==1)&(M0==1) then A1;

state A1:      if (M1==0)&(M0==0) then A0
          else if (M1==0)&(M0==1) then A2
          else if (M1==1)&(M0==0) then A0
          else if (M1==1)&(M0==1) then A3;

state A2:      if (M1==0)&(M0==0) then A1
          else if (M1==0)&(M0==1) then A4
          else if (M1==1)&(M0==0) then A0
          else if (M1==1)&(M0==1) then A0;
```

Same design realized using STATE DIAGRAM notation

```
state A3:      if (M1==0)&(M0==0) then A0
          else if (M1==0)&(M0==1) then A0
          else if (M1==1)&(M0==0) then A0
          else if (M1==1)&(M0==1) then A7;

state A4:      if (M1==0)&(M0==0) then A2
          else if (M1==0)&(M0==1) then A0
          else if (M1==1)&(M0==0) then A6
          else if (M1==1)&(M0==1) then A0;

state A5: goto A0;

state A6:      if (M1==0)&(M0==0) then A0
          else if (M1==0)&(M0==1) then A0
          else if (M1==1)&(M0==0) then A7
          else if (M1==1)&(M0==1) then A0;

state A7:      if (M1==0)&(M0==0) then A0
          else if (M1==0)&(M0==1) then A0
          else if (M1==1)&(M0==0) then A0
          else if (M1==1)&(M0==1) then A0;

EQUATIONS

QALL.CLK = CLOCK;

END
```

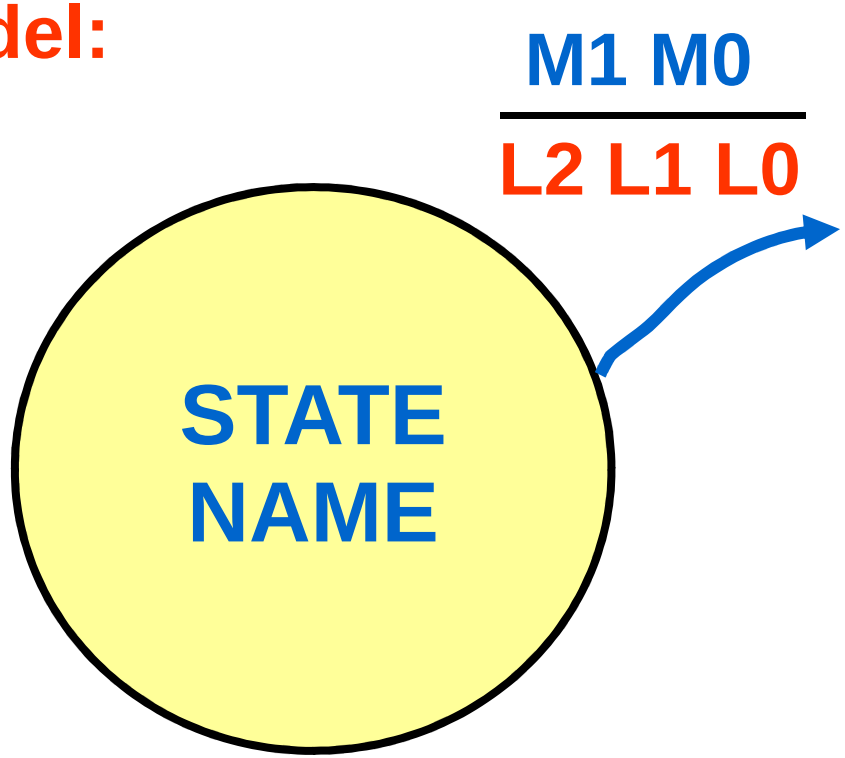
This realization also uses 3 macrocells

Mealy Model Realizations

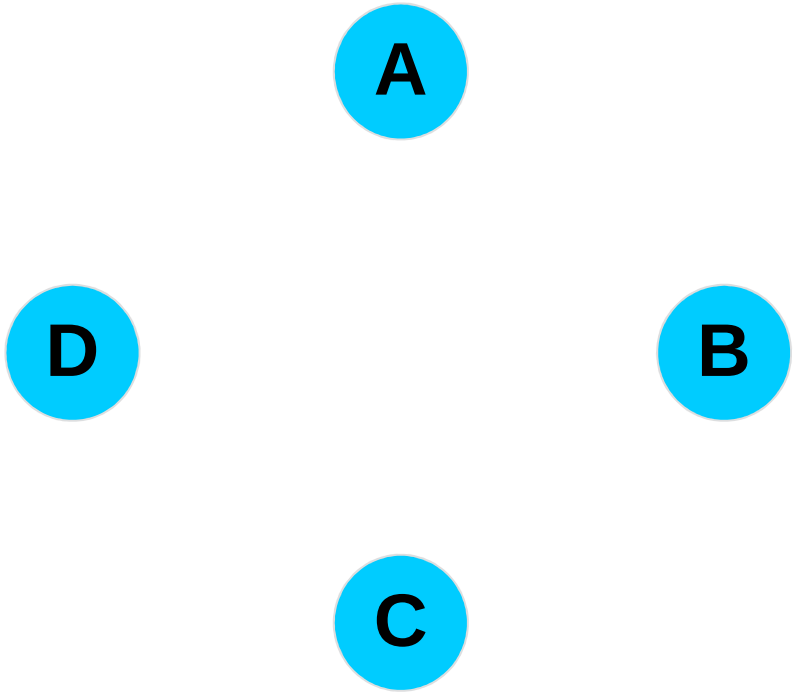
- Now try **MEALY** model implementation, and compare with **MOORE** model done previously
- Inputs and outputs **(review)**
  - To specify which of the 4 modes we want the circuit to operate, we will need 2 "mode control" inputs, **M1** and **M0**, where:
    - **0 0** $\rightarrow$ **single dot, left-to-right**
    - **0 1** $\rightarrow$ **single dot, right-to-left**
    - **1 0** $\rightarrow$ **building dots, left-to-right**
    - **1 1** $\rightarrow$ **building dots, right-to-left**
  - A separate output function needs to be determined for each of the 3 LED outputs: **L2**, **L1**, and **L0** (from left-to-right)

STEP 1: Construct a state transition diagram

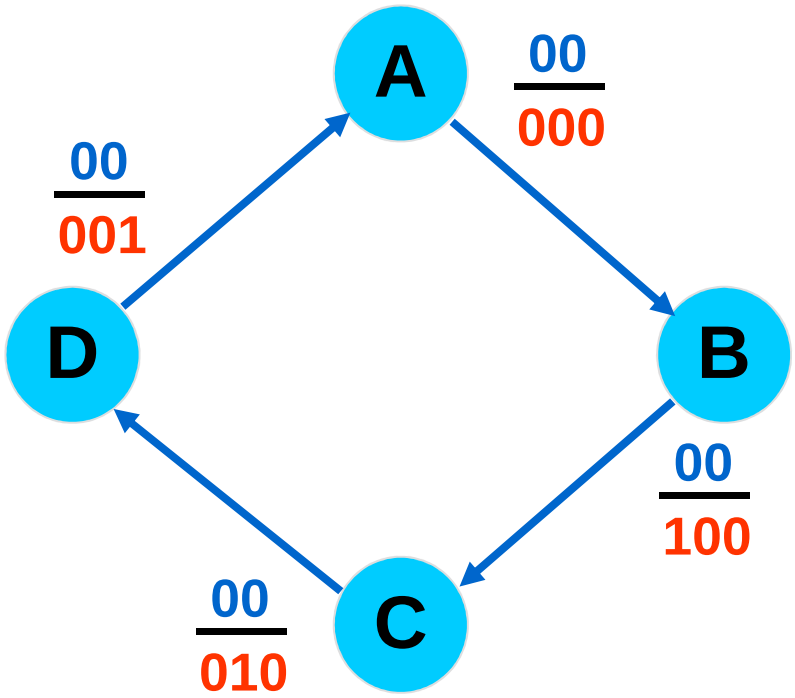
Mealy Model:



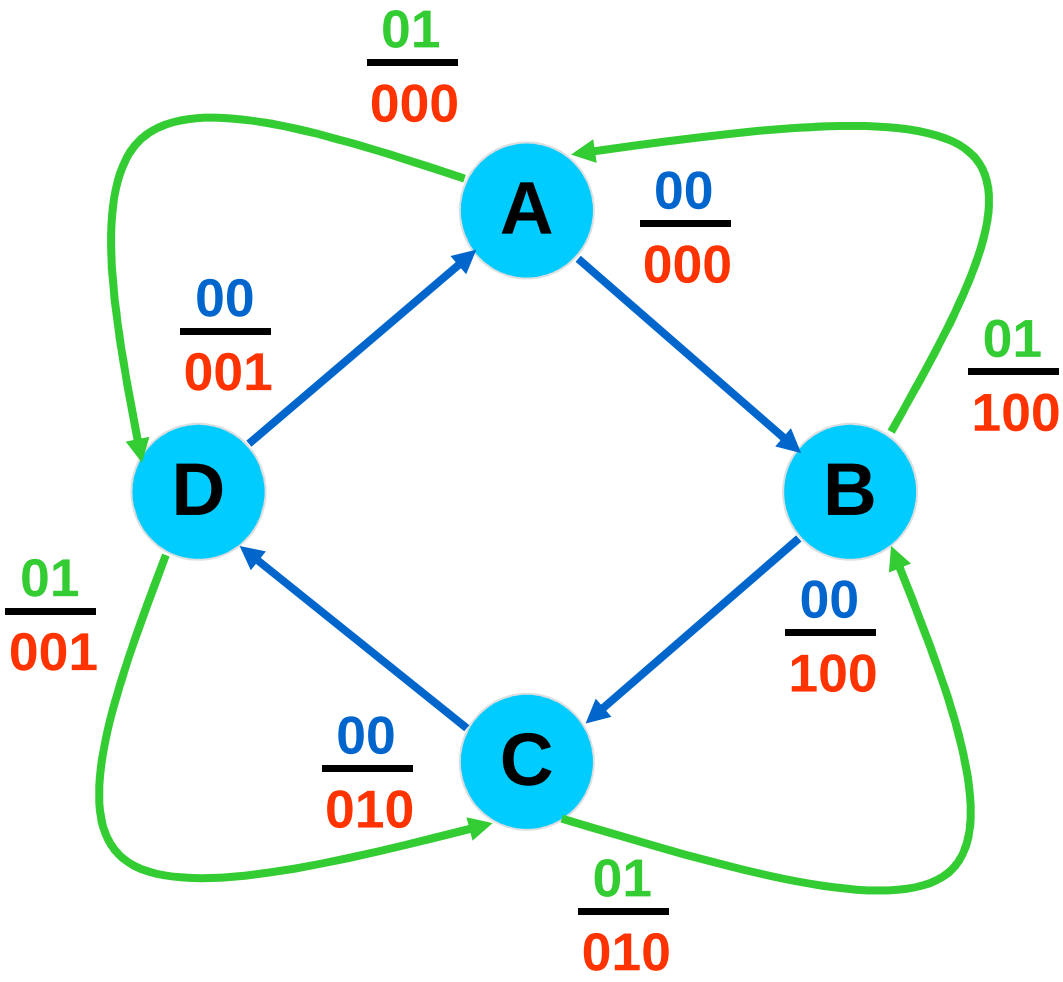
STEP 1: Construct a state transition diagram



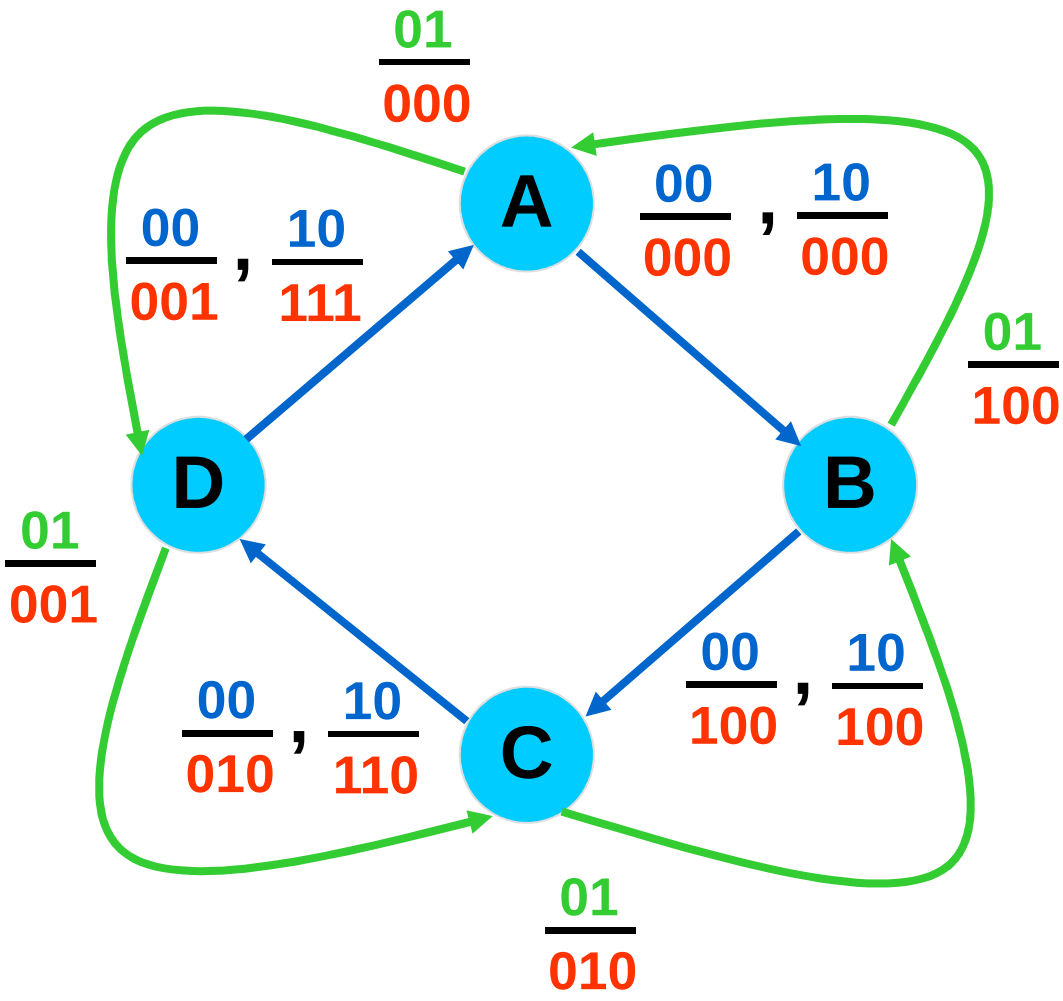
STEP 1: Construct a state transition diagram



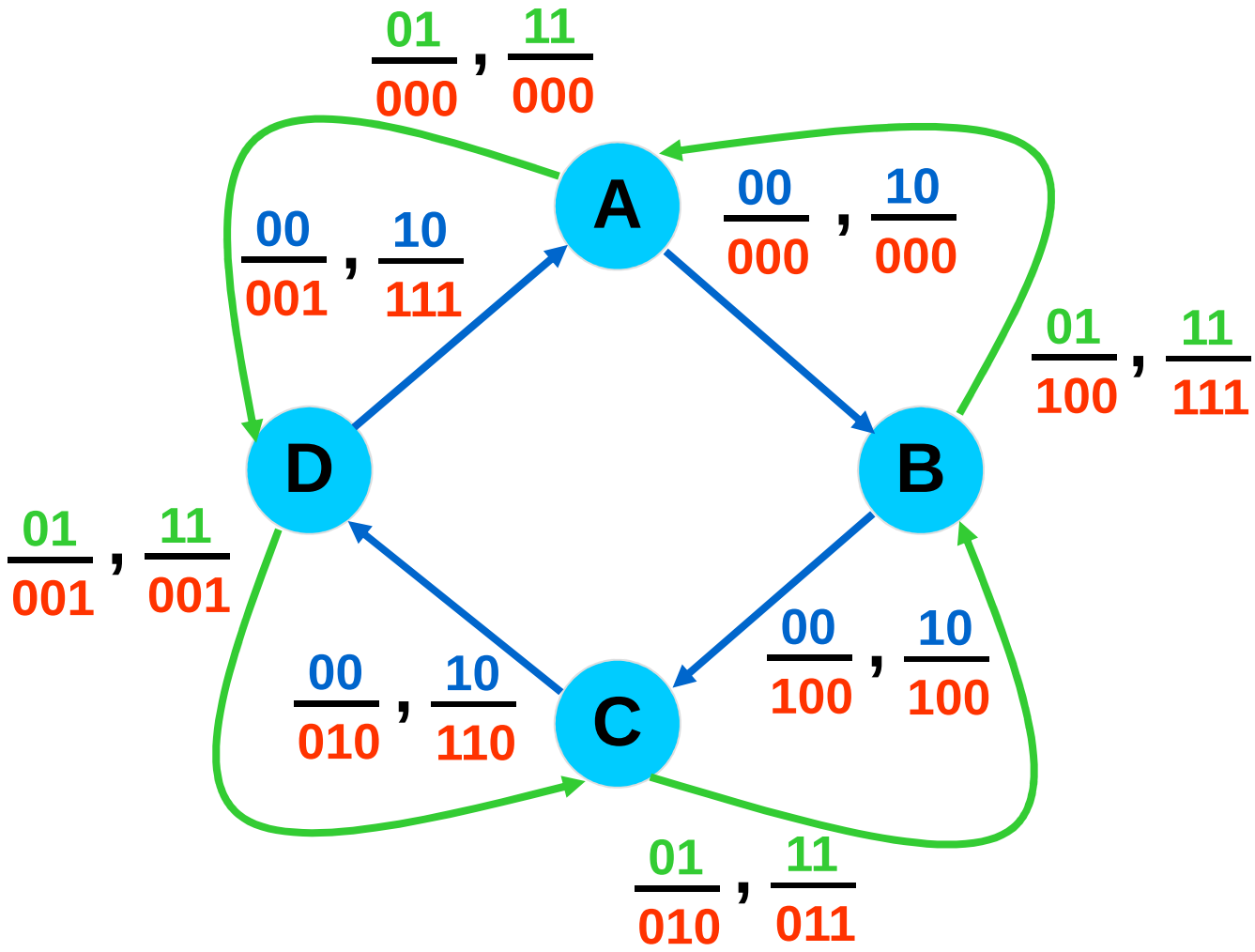
STEP 1: Construct a state transition diagram



STEP 1: Construct a state transition diagram

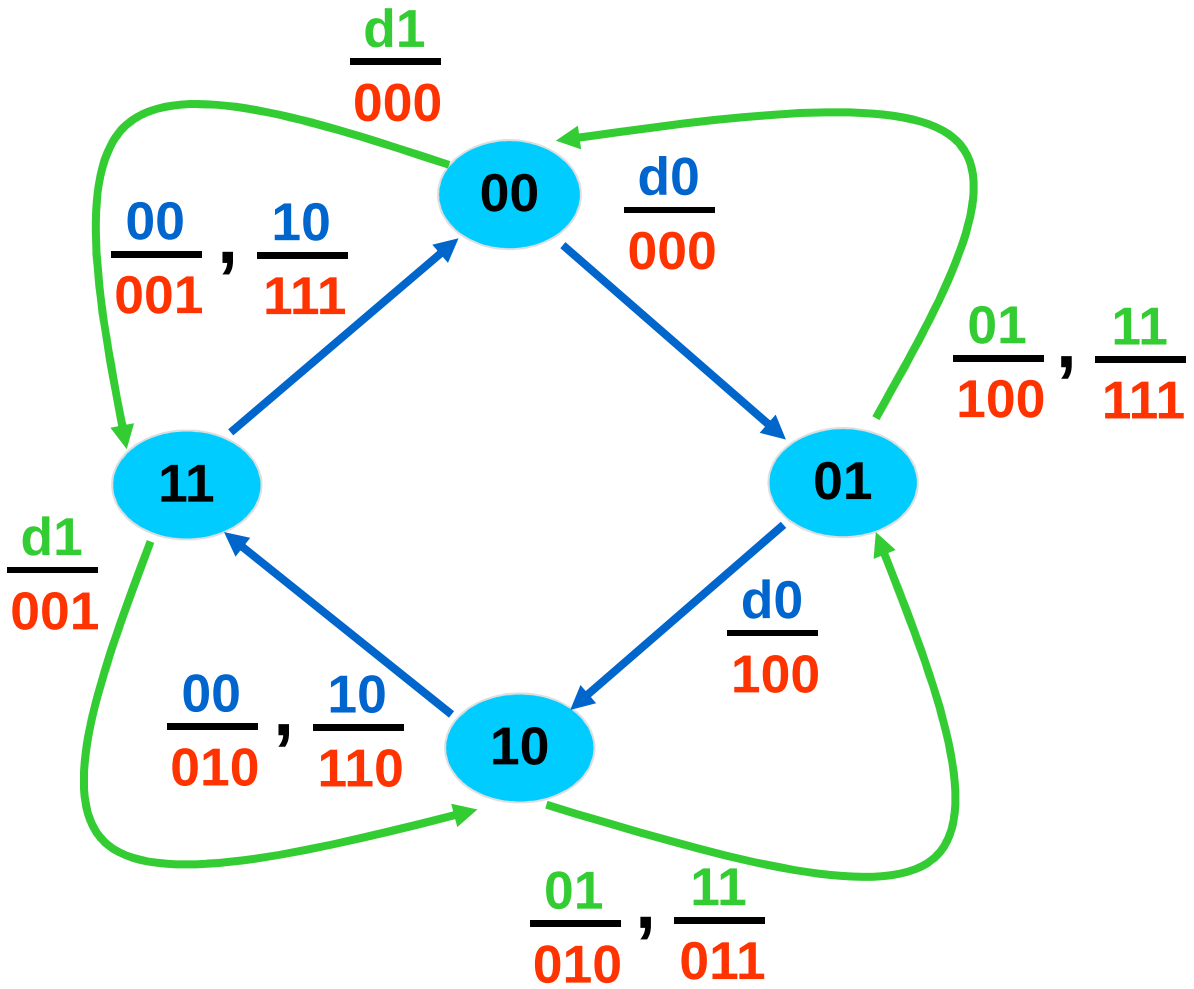


STEP 1: Construct a state transition diagram



STEP 2: Minimize the number of states √

STEP 3: Assign state variable combinations



STEP 4: Construct a PS-NS/PO Table

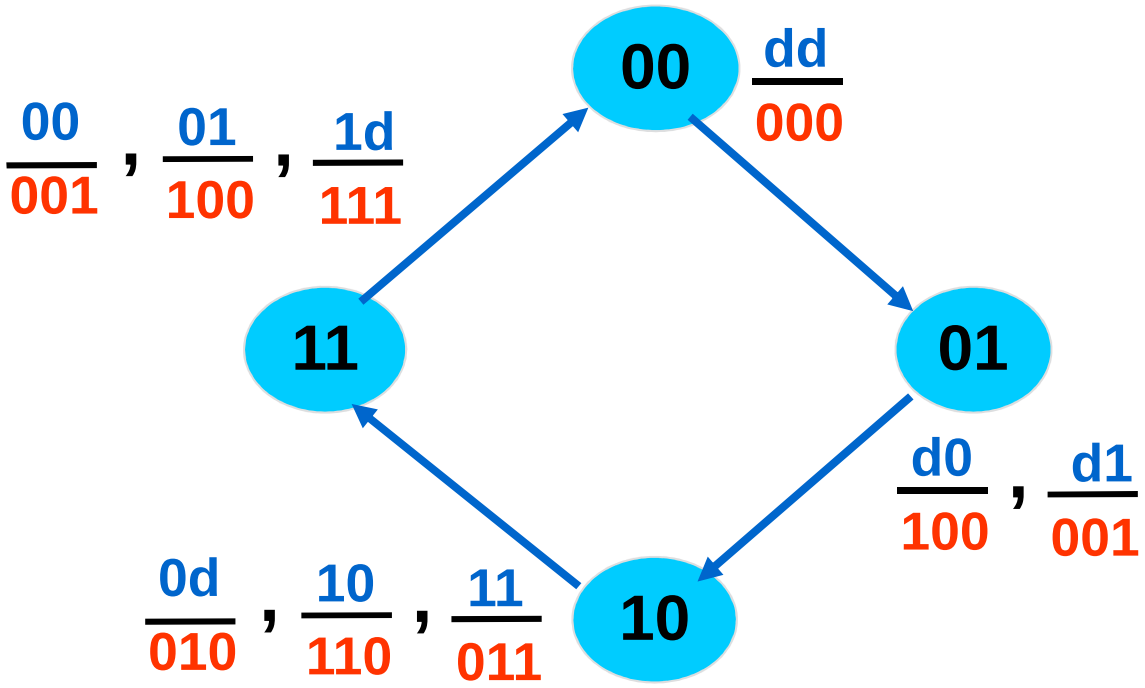
PS		PI		NS		PO		
Q1	Q0	M1	M0	Q1*	Q0*	L2	L1	L0
0	0	0	0	0	1	0	0	0
		0	1	1	1	0	0	0
		1	0	0	1	0	0	0
		1	1	1	1	0	0	0
0	1	0	0	1	0	1	0	0
		0	1	0	0	1	0	0
		1	0	1	0	1	0	0
		1	1	0	0	1	1	1
1	0	0	0	1	1	0	1	0
		0	1	0	1	0	1	0
		1	0	1	1	1	1	0
		1	1	0	1	0	1	1
1	1	0	0	0	0	0	0	1
		0	1	1	0	0	0	1
		1	0	0	0	1	1	1
		1	1	1	0	0	0	1

```
MODULE mealylsa

TITLE 'Light Sequencer - Mealy Model A'

DECLARATIONS
CLOCK pin;
M0, M1 pin;
Q1, Q0 pin istype 'reg';
L2, L1, L0 pin istype 'com';

truth_table ([Q1,Q0,M1,M0]:>[Q1,Q0])

             [ 0, 0, 0, 0]:>[ 0, 1];
             [ 0, 0, 0, 1]:>[ 1, 1];
             [ 0, 0, 1, 0]:>[ 0, 1];
             [ 0, 0, 1, 1]:>[ 1, 1];

             [ 0, 1, 0, 0]:>[ 1, 0];
             [ 0, 1, 0, 1]:>[ 0, 0];
             [ 0, 1, 1, 0]:>[ 1, 0];
             [ 0, 1, 1, 1]:>[ 0, 0];

             [ 1, 0, 0, 0]:>[ 1, 1];
             [ 1, 0, 0, 1]:>[ 0, 1];
             [ 1, 0, 1, 0]:>[ 1, 1];
             [ 1, 0, 1, 1]:>[ 0, 1];

             [ 1, 1, 0, 0]:>[ 0, 0];
             [ 1, 1, 0, 1]:>[ 1, 0];
             [ 1, 1, 1, 0]:>[ 0, 0];
             [ 1, 1, 1, 1]:>[ 1, 0];
```

```
truth_table ([Q1,Q0,M1,M0]->[L2,L1,L0])

             [ 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, 0, 0];

             [ 0, 1, 0, 0]->[ 1, 0, 0];
             [ 0, 1, 0, 1]->[ 1, 0, 0];
             [ 0, 1, 1, 0]->[ 1, 0, 0];
             [ 0, 1, 1, 1]->[ 1, 1, 1];

             [ 1, 0, 0, 0]->[ 0, 1, 0];
             [ 1, 0, 0, 1]->[ 0, 1, 0];
             [ 1, 0, 1, 0]->[ 1, 1, 0];
             [ 1, 0, 1, 1]->[ 0, 1, 1];

             [ 1, 1, 0, 0]->[ 0, 0, 1];
             [ 1, 1, 0, 1]->[ 0, 0, 1];
             [ 1, 1, 1, 0]->[ 1, 1, 1];
             [ 1, 1, 1, 1]->[ 0, 0, 1];


EQUATIONS
[Q1..Q0].CLK = CLOCK;

END
```

This realization uses 5 macrocells

Revisit Steps 2 & 3: Did we pick the "best" state/output assignments possible?



STEP 4: Construct a PS-NS/PO Table

PS		PI		NS		PO		
Q1	Q0	M1	M0	Q1*	Q0*	L2	L1	L0
0	0	0	0	0	1	0	0	0
		0	1	0	1	0	0	0
		1	0	0	1	0	0	0
		1	1	0	1	0	0	0
0	1	0	0	1	0	1	0	0
		0	1	1	0	0	0	1
		1	0	1	0	1	0	0
		1	1	1	0	0	0	1
1	0	0	0	1	1	0	1	0
		0	1	1	1	0	1	0
		1	0	1	1	1	1	0
		1	1	1	1	0	1	1
1	1	0	0	0	0	0	0	1
		0	1	0	0	1	0	0
		1	0	0	0	1	1	1
		1	1	0	0	1	1	1

```
MODULE mealylsb

TITLE 'Light Sequencer - Mealy Model B'

DECLARATIONS
CLOCK pin;
M0, M1 pin;
Q1, Q0 pin istype 'reg';
L2, L1, L0 pin istype 'com';

truth_table ([Q1,Q0,M1,M0]:>[Q1,Q0])

             [ 0, 0, 0, 0]:>[ 0, 1];
             [ 0, 0, 0, 1]:>[ 0, 1];
             [ 0, 0, 1, 0]:>[ 0, 1];
             [ 0, 0, 1, 1]:>[ 0, 1];

             [ 0, 1, 0, 0]:>[ 1, 0];
             [ 0, 1, 0, 1]:>[ 1, 0];
             [ 0, 1, 1, 0]:>[ 1, 0];
             [ 0, 1, 1, 1]:>[ 1, 0];

             [ 1, 0, 0, 0]:>[ 1, 1];
             [ 1, 0, 0, 1]:>[ 1, 1];
             [ 1, 0, 1, 0]:>[ 1, 1];
             [ 1, 0, 1, 1]:>[ 1, 1];

             [ 1, 1, 0, 0]:>[ 0, 0];
             [ 1, 1, 0, 1]:>[ 0, 0];
             [ 1, 1, 1, 0]:>[ 0, 0];
             [ 1, 1, 1, 1]:>[ 0, 0];
```

```
truth_table ([Q1,Q0,M1,M0]->[L2,L1,L0])

             [ 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, 0, 0];

             [ 0, 1, 0, 0]->[ 1, 0, 0];
             [ 0, 1, 0, 1]->[ 0, 0, 1];
             [ 0, 1, 1, 0]->[ 1, 0, 0];
             [ 0, 1, 1, 1]->[ 0, 0, 1];

             [ 1, 0, 0, 0]->[ 0, 1, 0];
             [ 1, 0, 0, 1]->[ 0, 1, 0];
             [ 1, 0, 1, 0]->[ 1, 1, 0];
             [ 1, 0, 1, 1]->[ 0, 1, 1];

             [ 1, 1, 0, 0]->[ 0, 0, 1];
             [ 1, 1, 0, 1]->[ 1, 0, 0];
             [ 1, 1, 1, 0]->[ 1, 1, 1];
             [ 1, 1, 1, 1]->[ 1, 1, 1];


EQUATIONS
[Q1..Q0].CLK = CLOCK;

END
```

This realization uses 5 macrocells

```
MODULE mealylsb_sd

TITLE 'Mealy Model Implemented with State Diagram'

DECLARATIONS
M0, M1 pin;
CLOCK pin;
Q1, Q0 pin istype 'reg';
L2, L1, L0 pin istype 'com';

" State definitions
QALL = [Q1,Q0];
A0   = [ 0, 0];
A1   = [ 0, 1];
A2   = [ 1, 0];
A3   = [ 1, 1];

state_diagram QALL

state A0: goto A1;
state A1: goto A2;
state A2: goto A3;
state A3: goto A0;
```

Same design realized using STATE DIAGRAM notation

```
truth_table ([Q1,Q0,M1,M0]->[L2,L1,L0])

             [ 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, 0, 0];

             [ 0, 1, 0, 0]->[ 1, 0, 0];
             [ 0, 1, 0, 1]->[ 0, 0, 1];
             [ 0, 1, 1, 0]->[ 1, 0, 0];
             [ 0, 1, 1, 1]->[ 0, 0, 1];

             [ 1, 0, 0, 0]->[ 0, 1, 0];
             [ 1, 0, 0, 1]->[ 0, 1, 0];
             [ 1, 0, 1, 0]->[ 1, 1, 0];
             [ 1, 0, 1, 1]->[ 0, 1, 1];

             [ 1, 1, 0, 0]->[ 0, 0, 1];
             [ 1, 1, 0, 1]->[ 1, 0, 0];
             [ 1, 1, 1, 0]->[ 1, 1, 1];
             [ 1, 1, 1, 1]->[ 1, 1, 1];

EQUATIONS

QALL.CLK = CLOCK;

END
```

This realization uses 5 macrocells

Clicker Quiz

```
MODULE MCLEDS
TITLE 'Multi-Color LED Light Machine'
DECLARATIONS
M pin; " mode control input
Q0..Q1 pin istype 'reg'; " state variables
R,G,Y,B pin istype 'com'; " LEDs (red/green/yellow/blue)
CLOCK pin;

TRUTH_TABLE([Q1,Q0, M]:>[Q1,Q0])
             [ 0, 0, 0]:>[ 1, 0];
             [ 0, 0, 1]:>[ 1, 1];
             [ 0, 1, 0]:>[ 1, 1];
             [ 0, 1, 1]:>[ 0, 0];
             [ 1, 0, 0]:>[ 0, 1];
             [ 1, 0, 1]:>[ 0, 1];
             [ 1, 1, 0]:>[ 0, 0];
             [ 1, 1, 1]:>[ 1, 0];

TRUTH_TABLE([Q1,Q0, M]->[ R, G, Y, B])
             [ 0, 0, 0]->[ 1, 0, 0, 0];
             [ 0, 0, 1]->[ 1, 0, 0, 0];
             [ 0, 1, 0]->[ 0, 0, 1, 0];
             [ 0, 1, 1]->[ 1, 1, 1, 1];
             [ 1, 0, 0]->[ 0, 1, 0, 0];
             [ 1, 0, 1]->[ 1, 1, 1, 0];
             [ 1, 1, 0]->[ 0, 0, 0, 1];
             [ 1, 1, 1]->[ 1, 1, 0, 0];

EQUATIONS
[Q1..Q0].CLK = CLOCK;
END
```

Q1. When **M=0,** the (repeating) colored LED sequence produced will be:

A. **R→G→Y→B→…**

B. **R→Y→G→B→…**

C. **B→Y→G→R→…**

D. **B→G→Y→R→…**

E. **none of the above**

Q2. When **M=1,** the (repeating) colored LED sequence produced will be:

A. R→RGYB→RGY→RG→…

B. R→RG→RGY→RGYB→…

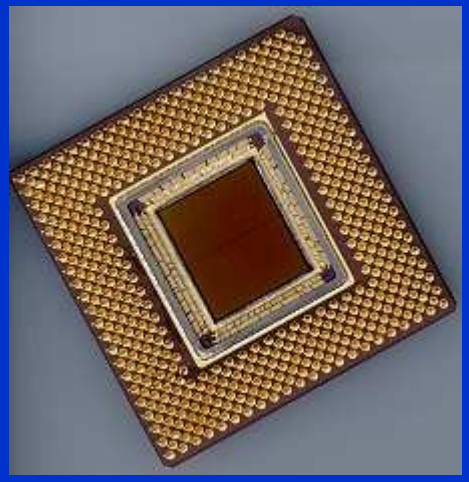
C. RGYB→RGY→RG→R→…

D. R→RGY→RG→RGYB→…

E. none of the above

Summary

- The choice of model (Mealy vs. Moore) can have a ***significant impact*** on the ***complexity*** of the realization and PLD resources (macrocells) consumed
- The state assignment ***strategy*** employed can make a ***significant difference*** in the ***amount of work required***
- "Obvious" state minimization can also sometimes be useful (formal state-minimization procedures are seldom used by most digital designers, however)
- The only formal way to find the ***best*** state assignment is to try ***all*** the assignments – that's too much work (even for ***students***)!
- To do this well, we need ***experience*** as well as have knowledge of some practical guidelines (see text)
- There is no substitute for ***practice*** in designing state machines – much of engineering is ***applied intuition***, and this is a good example of it!



Introduction to Digital System Design

Module 3-G
State Machine Design Examples: Counters and Shift Registers

Reading Assignment:

DDPP 4th Ed., pp. 710-721, 727-736

Learning Objectives:

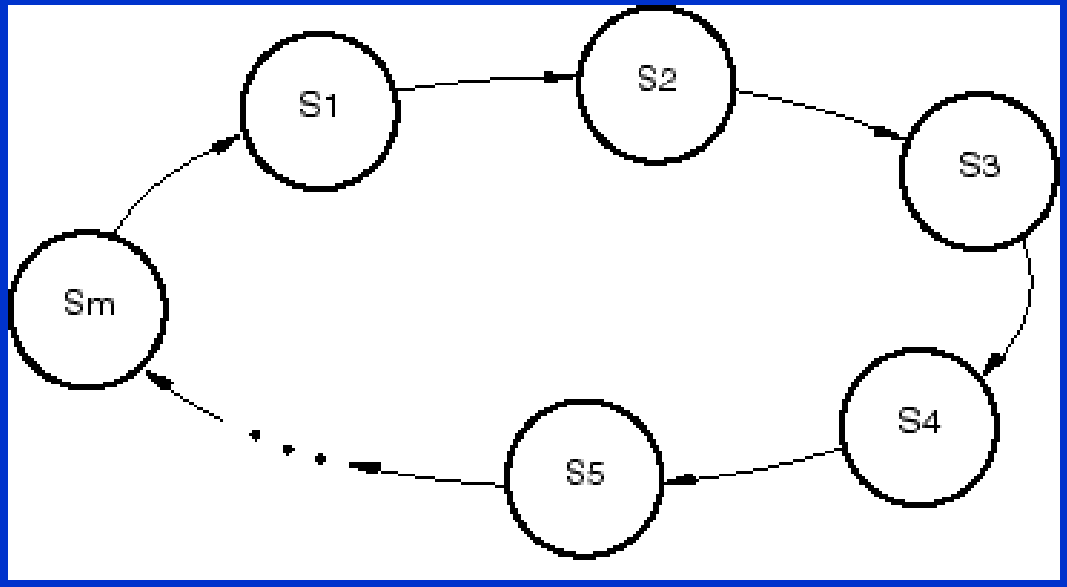
- **Compare and contrast the operation of binary and shift register counters**
- **Derive the next state equations for binary "up" and "down" counters**
- **Describe the feedback necessary to make ring and Johnson counters self-correcting**
- **Compare and contrast state decoding for binary and shift register counters**
- **Describe why "glitches" occur in some state decoding strategies and discuss how to eliminate them**

Outline

- Overview
- Binary counter registers
- UP and DOWN counter derivations
- Basic binary counter extensions
  - ENABLE input
  - ASYNCHRONOUS RESET
  - UP and DOWN count modes
- Synchronously resettable counters
  - SYNCHRONOUS RESET
  - MODULUS control
- Shift register counters
- State decoding
- Summary

Overview

- **Definition: The name *counter* is used for any clocked sequential circuit whose state diagram contains a single cycle**



- **Definition: A *register* is a collection of two or more flip-flops with a common clock and, generally, a common purpose**

Binary Counter Registers

- **Definition: The *modulus* of a counter is the number of states in the cycle – a counter with M states is called a *modulo-M counter* (or sometimes a *divide-by-M counter*)**
- **Definition: A *synchronous counter* connects all of its flip-flop clock inputs to the same common CLOCK signal, so that all the flip-flop outputs change state *simultaneously***
- **The most commonly used counter type is an *n-bit binary counter*, with *n* flip-flops and 2^n states, visited in the sequence 0, 1, 2, … , 2^n-1, 0, 1, 2, ...**

Binary UP Counter Derivation

- **The design of a basic binary UP counter is derived as follows:**

Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

When does Q0 change state?

Every clock cycle

What is the equation for Q0*?

Q0* = Q0´

Binary UP Counter Derivation

- **The design of a basic binary UP counter is derived as follows:**

Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

When does Q1 change state?

When Q0 = 1

What is the equation for Q1*?

$Q1^* = Q1 \oplus Q0$

Binary UP Counter Derivation

- **The design of a basic binary UP counter is derived as follows:**

Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

When does Q2 change state?

When Q0 = 1 AND Q1 = 1

What is the equation for Q2*?

$Q2^* = Q2 \oplus (Q1 \cdot Q0)$

Binary UP Counter Derivation

- **The design of a basic binary UP counter is derived as follows:**

What is the next state equation for an arbitrary stage "K" (Q_K^*) of a binary UP counter?

$$Q_K^* = Q_K \oplus (Q_{K-1} \bullet Q_{K-2} \bullet \cdots \bullet Q_1 \bullet Q_0)$$

```
MODULE count8u

TITLE 'Basic 8-bit Binary UP Counter'

DECLARATIONS
Q0..Q7 pin istype 'reg';
CLOCK pin;

EQUATIONS
Q0 := !Q0;
Q1 := Q1 $ Q0;
Q2 := Q2 $ (Q1&Q0);
Q3 := Q3 $ (Q2&Q1&Q0);
Q4 := Q4 $ (Q3&Q2&Q1&Q0);
Q5 := Q5 $ (Q4&Q3&Q2&Q1&Q0);
Q6 := Q6 $ (Q5&Q4&Q3&Q2&Q1&Q0);
Q7 := Q7 $ (Q6&Q5&Q4&Q3&Q2&Q1&Q0);

[Q0..Q7].CLK = CLOCK;

END
```

Binary DOWN Counter Derivation

- **The design of a basic binary DOWN counter is derived as follows:**

Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

When does Q0 change state?

Every clock cycle

What is the equation for Q0*?

Q0* = Q0´

Binary DOWN Counter Derivation

- **The design of a basic binary DOWN counter is derived as follows:**

Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

When does Q1 change state?

When Q0 = 0

What is the equation for Q1*?

$Q1^* = Q1 \oplus Q0'$

Binary DOWN Counter Derivation

- **The design of a basic binary DOWN counter is derived as follows:**

Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

When does Q2 change state?

When Q0 = 0 AND Q1 = 0

What is the equation for Q2*?

$Q2^* = Q2 \oplus (Q1' \cdot Q0')$

Binary DOWN Counter Derivation

- **The design of a basic binary DOWN counter is derived as follows:**

What is the next state equation for an arbitrary stage "K" (Q_K^*) of a binary DOWN counter?

$$Q_K^* = Q_K \oplus (Q'_{K-1} \cdot Q'_{K-2} \cdot \ldots \cdot Q'_1 \cdot Q'_0)$$

```
MODULE count8d
TITLE 'Basic 8-bit Binary DOWN Counter'

DECLARATIONS
Q0..Q7 pin istype 'reg';
CLOCK pin;

EQUATIONS
Q0 := !Q0;
Q1 := Q1 $ !Q;
Q2 := Q2 $ (!Q1&!Q0);
Q3 := Q3 $ (!Q2&!Q1&!Q0);
Q4 := Q4 $ (!Q3&!Q2&!Q1&!Q0);
Q5 := Q5 $ (!Q4&!Q3&!Q2&!Q1&!Q0);
Q6 := Q6 $ (!Q5&!Q4&!Q3&!Q2&!Q1&!Q0);
Q7 := Q7 $ (!Q6&!Q5&!Q4&!Q3&!Q2&!Q1&!Q0);

[Q0..Q7].CLK = CLOCK;

END
```

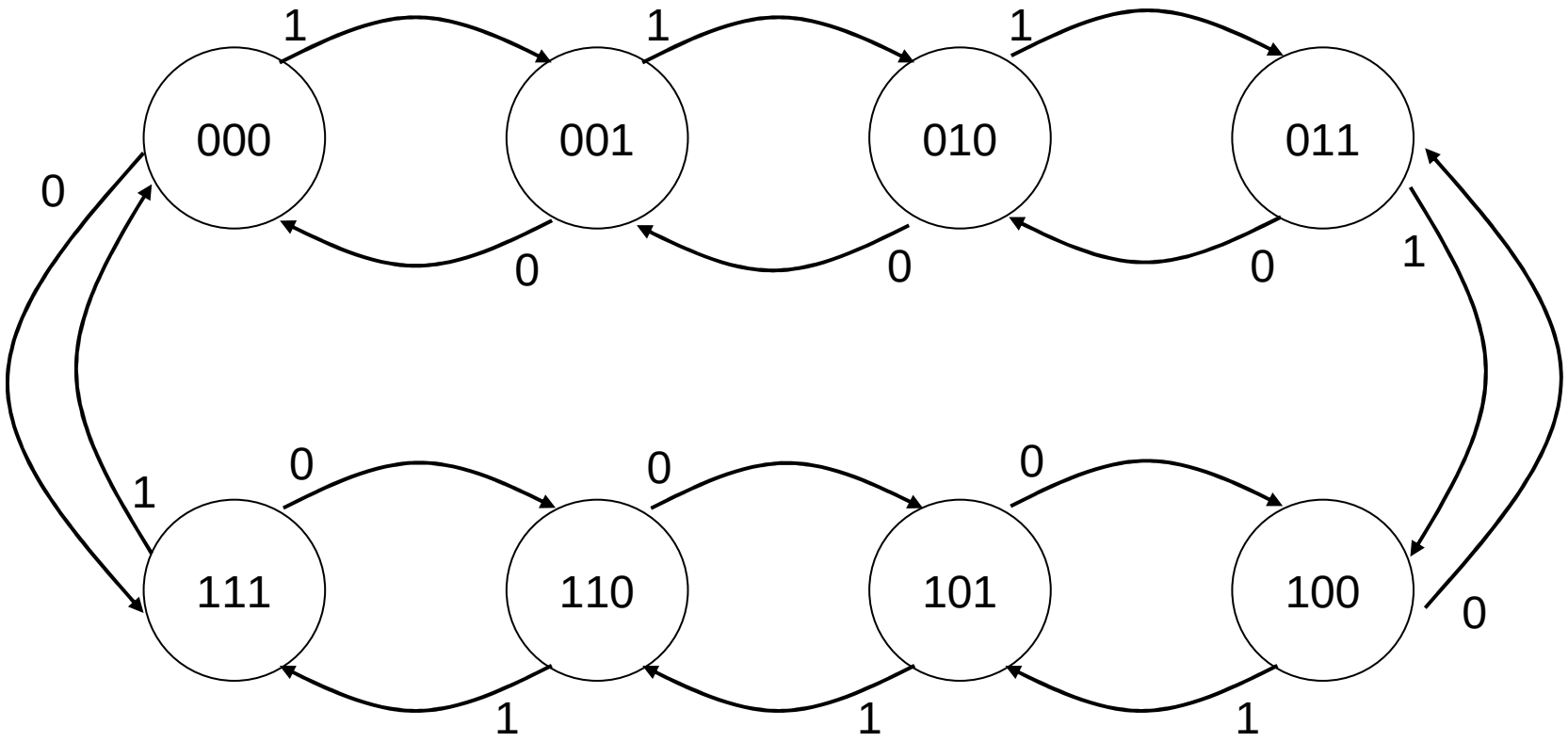
Basic Binary Counter Extensions

- **Extensions to the basic binary counter commonly of interest include:**
  - **providing both UP and DOWN COUNT modes**
  - **providing an ENABLE input**
  - **providing an ASYNCHRONOUS RESET**

```
MODULE cnt8ud
TITLE '8-bit Binary UP/DOWN Counter'

DECLARATIONS
Q0..Q7 pin istype 'reg';
M, CLOCK pin;

" M=0 count DOWN, M=1 count UP

EQUATIONS
Q0 := !Q0;
Q1 := Q1 $ (M&Q0 # !M&!Q0);
Q2 := Q2 $ (M&Q1&Q0 # !M&!Q1&!Q0);
Q3 := Q3 $ (M&Q2&Q1&Q0 # !M&!Q2&!Q1&!Q0);
Q4 := Q4 $ (M&Q3&Q2&Q1&Q0 # !M&!Q3&!Q2&!Q1&!Q0);
Q5 := Q5 $ (M&Q4&Q3&Q2&Q1&Q0 # !M&!Q4&!Q3&!Q2&!Q1&!Q0);
Q6 := Q6 $ (M&Q5&Q4&Q3&Q2&Q1&Q0 # !M&!Q5&!Q4&!Q3&!Q2&!Q1&!Q0);
Q7 := Q7 $ (M&Q6&Q5&Q4&Q3&Q2&Q1&Q0 # !M&!Q6&!Q5&!Q4&!Q3&!Q2&!Q1&!Q0);

[Q0..Q7].CLK = CLOCK;

END
```

```
MODULE count8er

TITLE '8-bit Binary UP Counter with ENABLE and Asynchronous Reset'

DECLARATIONS
Q0..Q7 pin istype 'reg';
CLOCK pin;
EN pin;
ARESET pin;

" If EN=0, stays in same state
" If EN=1, counts UP
" If ARESET asserted, resets to 00...0 (regardless of whether or not enabled)

EQUATIONS
Q0 := !EN&Q0 # EN&!Q0;
Q1 := !EN&Q1 # EN&(Q1 $ Q0);
Q2 := !EN&Q2 # EN&(Q2 $ (Q1&Q0));
Q3 := !EN&Q3 # EN&(Q3 $ (Q2&Q1&Q0));
Q4 := !EN&Q4 # EN&(Q4 $ (Q3&Q2&Q1&Q0));
Q5 := !EN&Q5 # EN&(Q5 $ (Q4&Q3&Q2&Q1&Q0));
Q6 := !EN&Q6 # EN&(Q6 $ (Q5&Q4&Q3&Q2&Q1&Q0));
Q7 := !EN&Q7 # EN&(Q7 $ (Q6&Q5&Q4&Q3&Q2&Q1&Q0));

[Q7..Q0].AR = ARESET;

[Q0..Q7].CLK = CLOCK;

END
```

Clicker Quiz


000
001
010
011
111
110
101
100
1
1
1
0
0
0
0
1
0
0
0
1
1
1

```
Module CQ
Title 'Program (A)'
DECLARATIONS
CLOCK, M pin;
Q0..Q2 pin istype 'reg';
EQUATIONS
Q0 := !Q0;
Q1 := !Q1 $ (!M&!Q0 # M&Q0);
Q2 := !Q2 $ (!M&!Q1&!Q0 # M&Q1&Q0);
[Q2..Q0].CLK = CLOCK;
END
```

```
Module CQ
Title 'Program (C)'
DECLARATIONS
CLOCK, M pin;
Q0..Q2 pin istype 'reg';
EQUATIONS
Q0 := !Q0;
Q1 := Q1 $ (!M&!Q0 # M&Q0);
Q2 := Q2 $ (!M&!Q1&!Q0 # M&Q1&Q0);
[Q2..Q0].CLK = CLOCK;
END
```

```
Module CQ
Title 'Program (B)'
DECLARATIONS
CLOCK, M pin;
Q0..Q2 pin istype 'reg';
EQUATIONS
Q0 := !Q0;
Q1 := Q1 $ (!M&Q0 # M&!Q0);
Q2 := Q2 $ (!M&Q1&Q0 # M&!Q1&!Q0);
[Q2..Q0].CLK = CLOCK;
END
```

```
Module CQ
Title 'Program (D)'
DECLARATIONS
CLOCK, M pin;
Q0..Q2 pin istype 'reg';
EQUATIONS
[Q2..Q0] := [Q2..Q0] + 1
[Q2..Q0].CLK = CLOCK;
END
```

(E) None of the above

Resettable Counters

- In addition to an ***asynchronous reset*** (which allows to the counter to be placed in a known initial state), it is sometimes useful to provide a ***synchronous reset*** capability
- Such a counter is useful in applications where the number of states in the counting sequence is determined ***dynamically***
- Example: State counter in a computer's execute unit, where the number of cycles necessary to complete an instruction varies
- Another variation: Counter with a "programmable" final state (modulo M)

```
MODULE rcnt8u
TITLE 'Resettable 8-bit Binary UP Counter'

DECLARATIONS
Q0..Q7 pin istype 'reg';
R, CLOCK pin;

" if R=1, next state will be 00...0

EQUATIONS
Q0 := !R & !Q0;
Q1 := !R & (Q1 $ Q0);
Q2 := !R & (Q2 $ (Q1&Q0));
Q3 := !R & (Q3 $ (Q2&Q1&Q0));
Q4 := !R & (Q4 $ (Q3&Q2&Q1&Q0));
Q5 := !R & (Q5 $ (Q4&Q3&Q2&Q1&Q0));
Q6 := !R & (Q6 $ (Q5&Q4&Q3&Q2&Q1&Q0));
Q7 := !R & (Q7 $ (Q6&Q5&Q4&Q3&Q2&Q1&Q0));

[Q0..Q7].CLK = CLOCK;

END
```

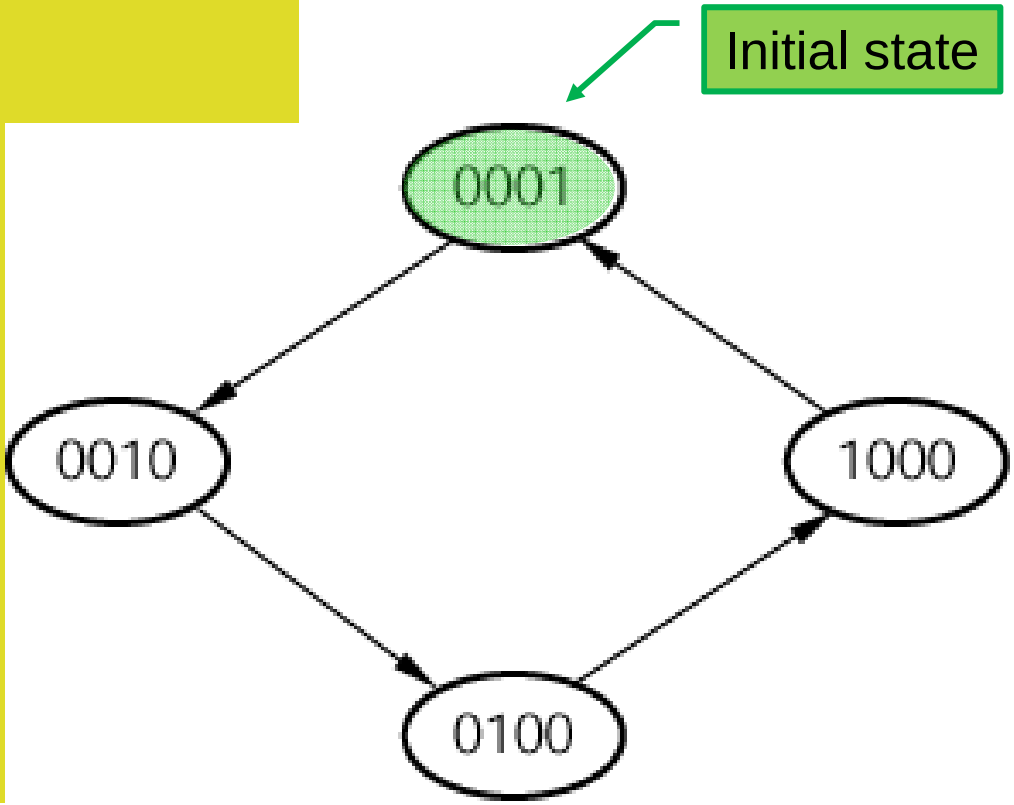
```
MODULE mrcnt8ud

TITLE 'Modulo Resettable 8-bit Binary UP Counter'

DECLARATIONS
Q0..Q7 pin istype 'reg';
D0..D7 pin;
CLOCK pin;
R pin istype 'com';

" Counts UP to value input on D7..D0; next state is 00...0

EQUATIONS
R = !(D0$Q0) & !(D1$Q1) & !(D2$Q2) & !(D3$Q3) &
    !(D4$Q4) & !(D5$Q5) & !(D6$Q6) & !(D7$Q7);

Q0 := !R & !Q0;
Q1 := !R & (Q1 $ Q0);
Q2 := !R & (Q2 $ (Q1&Q0));
Q3 := !R & (Q3 $ (Q2&Q1&Q0));
Q4 := !R & (Q4 $ (Q3&Q2&Q1&Q0));
Q5 := !R & (Q5 $ (Q4&Q3&Q2&Q1&Q0));
Q6 := !R & (Q6 $ (Q5&Q4&Q3&Q2&Q1&Q0));
Q7 := !R & (Q7 $ (Q6&Q5&Q4&Q3&Q2&Q1&Q0));

[Q0..Q7].CLK = CLOCK;

END
```

Shift-Register Counters

- **Definition: A shift register whose state diagram is *cyclic* is called a *shift-register counter***
- **Unlike a binary counter, a shift-register counter does not count in an "up" or "down" binary sequence, but is useful in many "control" applications nonetheless**
- **The simplest shift-register counter uses an n-bit shift register to obtain a counter with n states, and is called a *ring counter***
- **A ring counter sequence is sometimes referred to as *"one hot"***

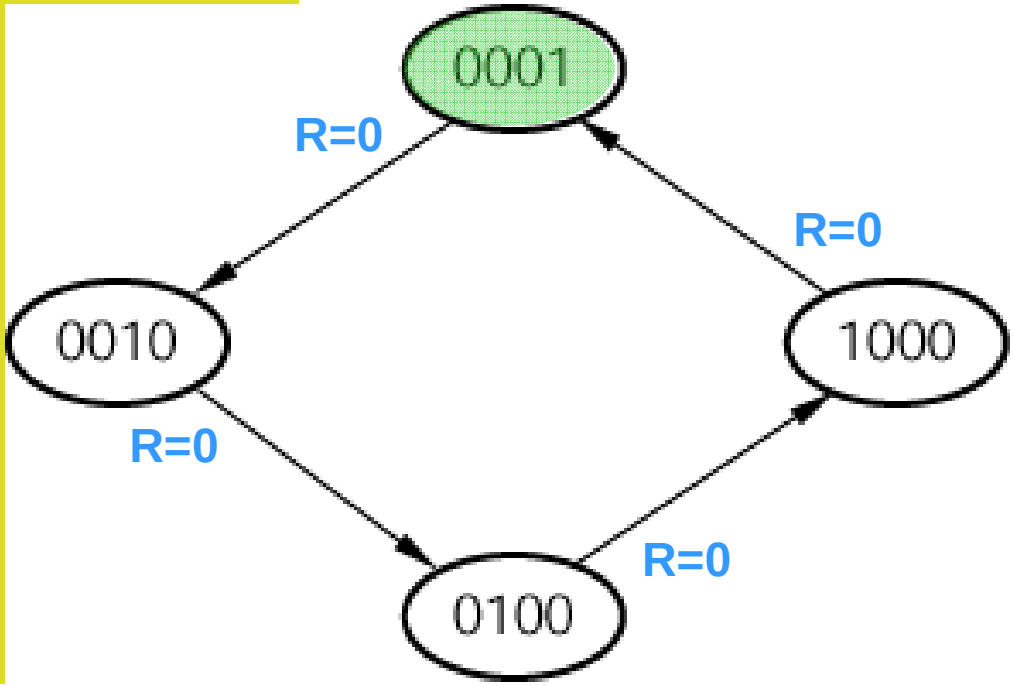
Example – Simple 4-bit Ring Counter

```
MODULE ring4

TITLE 'Simple 4-bit Ring Counter'

" Assertion of R causes next state
" of ring counter to be 0001

DECLARATIONS
CLOCK  pin;
R      pin;   " synchronous reset
Q0..Q3 pin istype 'reg';

EQUATIONS

Q3 := !R&Q2;
Q2 := !R&Q1;
Q1 := !R&Q0;

Q0 := !R&Q3 # R;

[Q0..Q3].CLK = CLOCK;

END
```



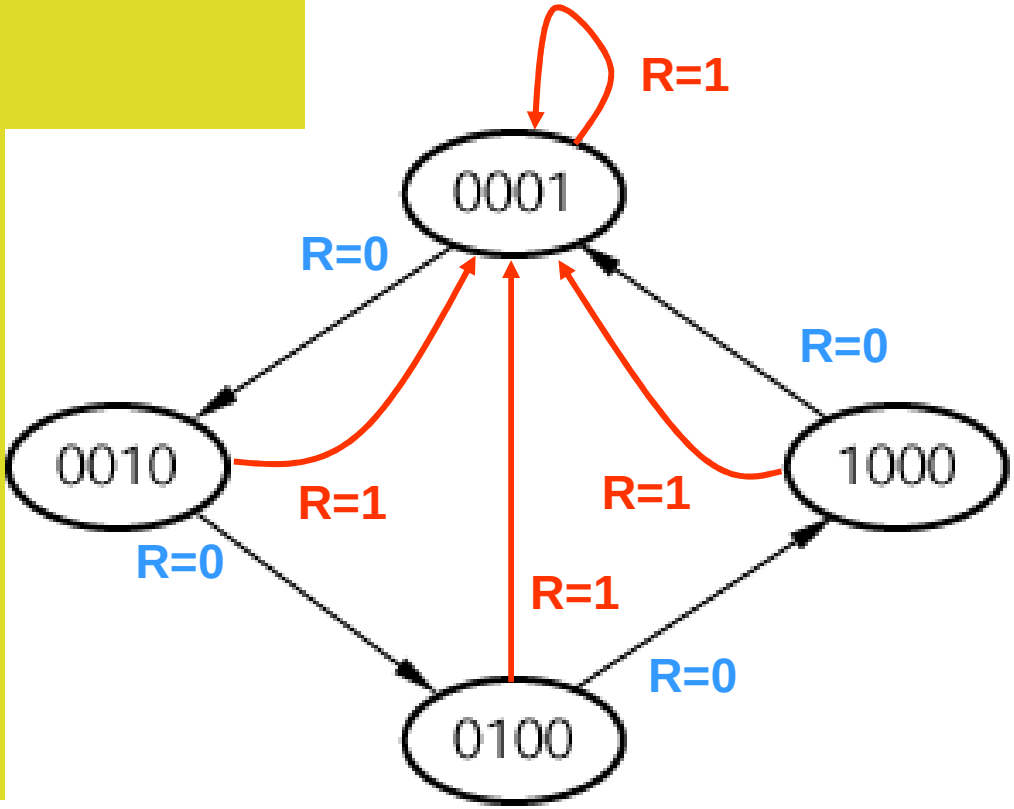
Example – Simple 4-bit Ring Counter

```
MODULE ring4

TITLE 'Simple 4-bit Ring Counter'

" Assertion of R causes next state
" of ring counter to be 0001

DECLARATIONS
CLOCK  pin;
R      pin;   " synchronous reset
Q0..Q3 pin istype 'reg';

EQUATIONS

Q3 := !R&Q2;
Q2 := !R&Q1;
Q1 := !R&Q0;

Q0 := !R&Q3 # R;

[Q0..Q3].CLK = CLOCK;

END
```



Example – Simple 4-bit Ring Counter

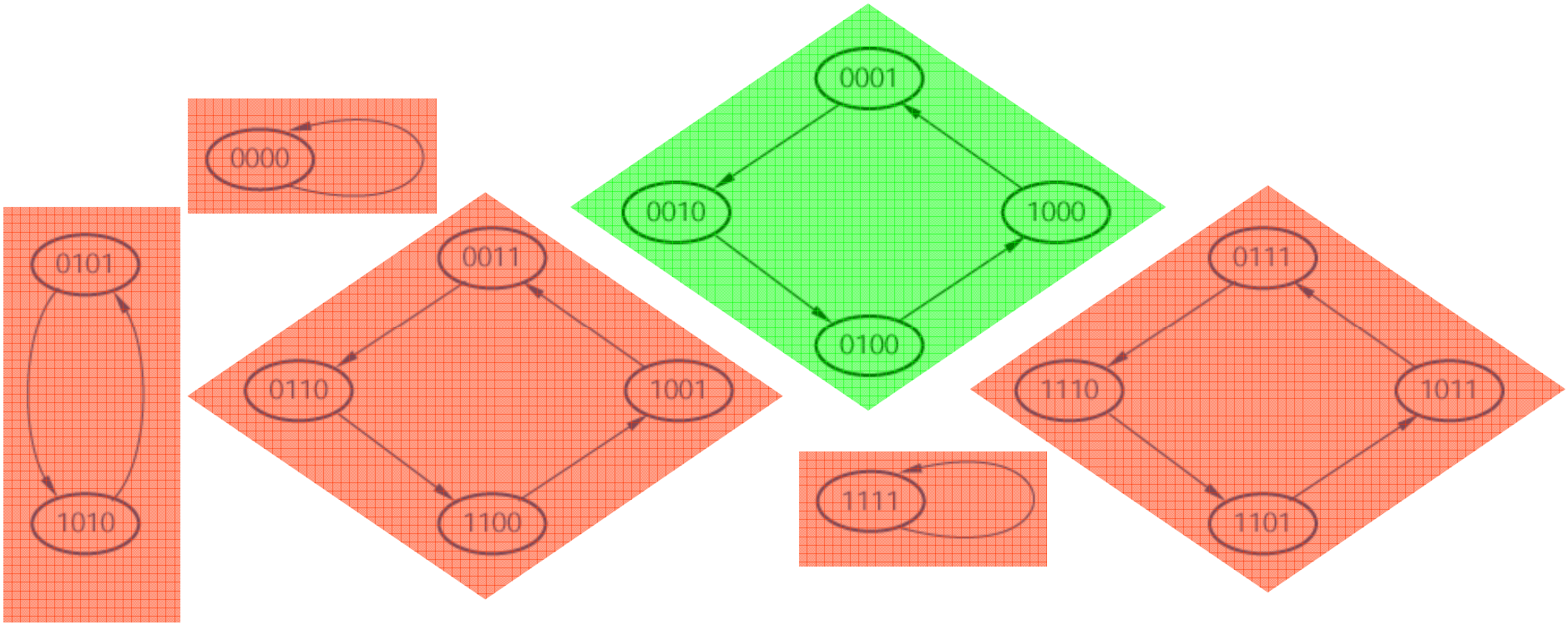
```
MODULE ring4

TITLE 'Simple 4-bit Ring Counter'

" Assertion of R causes next state
" of ring counter to be 0001

DECLARATIONS
CLOCK  pin;
R      pin;   " synchronous reset
Q0..Q3 pin istype 'reg';

EQUATIONS

Q3 := !R&Q2;
Q2 := !R&Q1;
Q1 := !R&Q0;

Q0 := !R&Q3 # R;

[Q0..Q3].CLK = CLOCK;

END
```



Self-Correcting Ring Counters

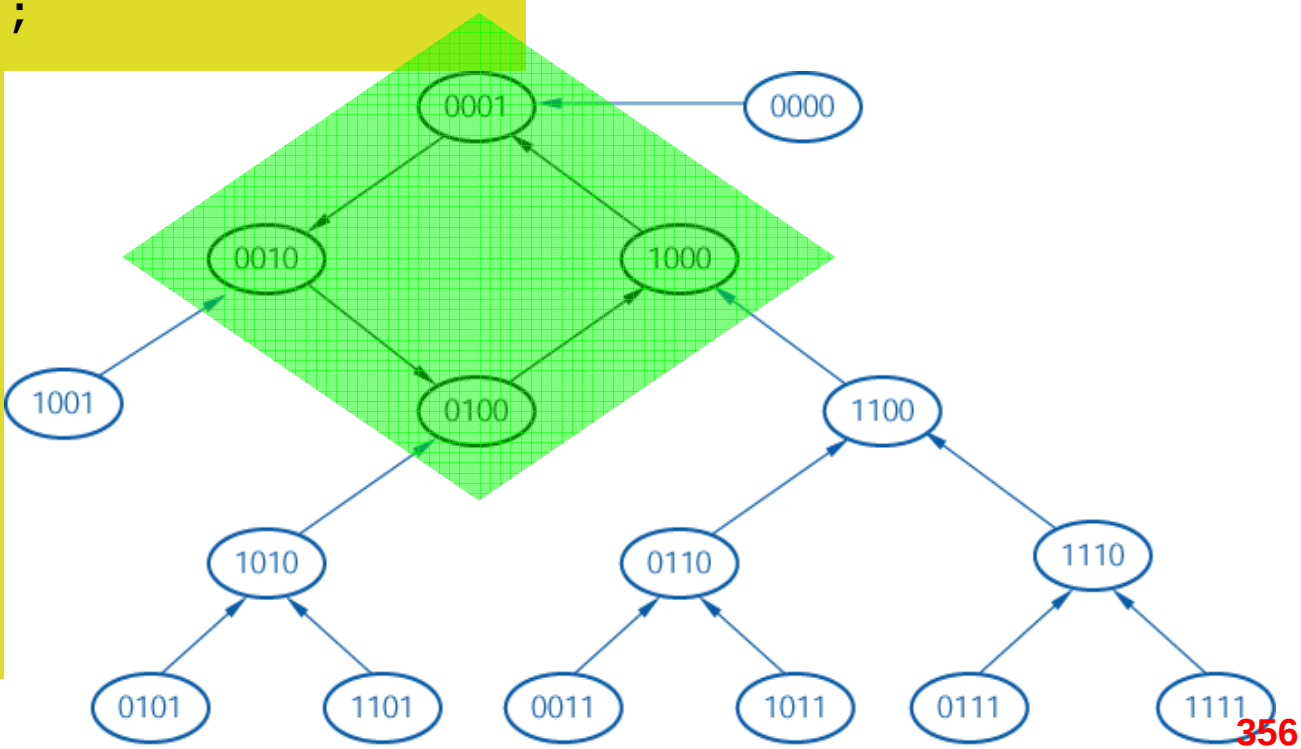
- **Problem: The simple ring counter is *not robust* – if it somehow gets off the normal 4-state cycle (e.g., due to noise), it *stays off***
- **Solution: A *self-correcting* counter is designed so that all "abnormal" states have transitions leading to "normal" states**
  - **uses an n-1 input *NOR function* to shift in a "1" only when the n-1 least significant bits of an n-bit ring counter are "0" (i.e., shifts in a "0" until the counter reaches state d000)**
  - **all *"abnormal" states* lead back into the normal n-state ring cycle**

State Transition Diagrams for Simple 4-bit Ring Counter



Example – Self-Correcting 4-bit Ring Counter

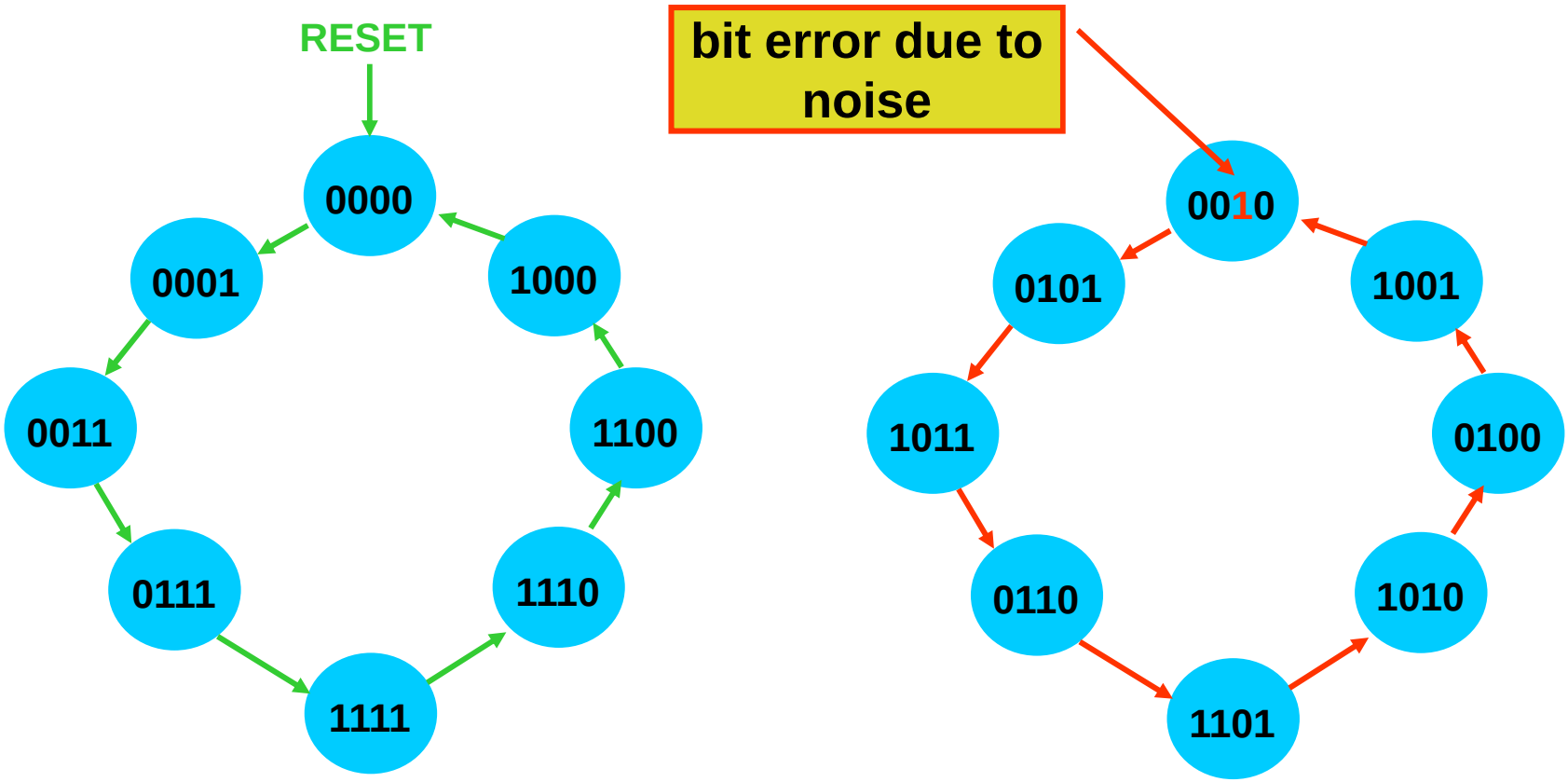
```
MODULE ring4sc

TITLE 'Self-Correcting 4-bit Ring Counter'

" Uses NOR function to make sure that the
" next state after d000 is 0001

DECLARATIONS
CLOCK  pin;
Q0..Q3 pin istype 'reg';

EQUATIONS

Q3 := Q2;
Q2 := Q1;
Q1 := Q0;

Q0 := !(Q2#Q1#Q0);

[Q0..Q3].CLK = CLOCK;

END
```



Johnson Counters

- **Definition: An n-bit shift register with the *complement* of the serial output fed back into the serial input is a counter with 2n states and is called a *switchtail*, *twisted-ring*, or *Johnson counter***
- **Problem: A Johnson counter has the same robustness problem that the simple ring counter has**
- **Solution: Make it *self-correcting* by using appropriate feedback, here to load "0001" as the next state whenever the current state is "0dd0" (or, for an n-bit counter, when the current state is 0d…d0)**

Example – Simple 4-bit Johnson (“Switchtail”) Counter

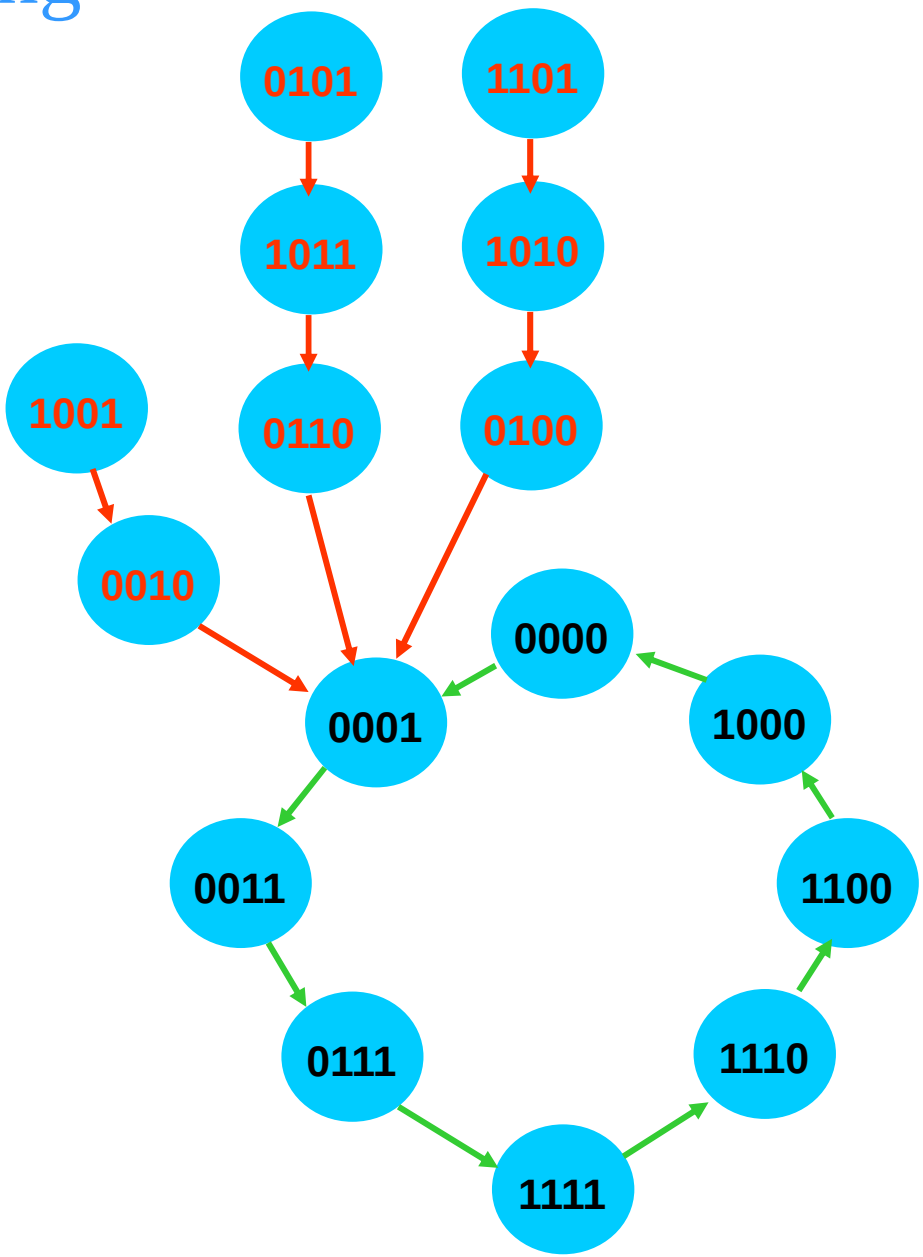


Normal Sequence After RESET

“Misfiring” Due to Noise

Example – Self-Correcting 4-bit Johnson Counter

```
MODULE john4sc

TITLE 'Self-Correcting 4-bit
Johnson Counter'

DECLARATIONS
CLOCK  pin;
Q0..Q3 pin istype 'reg';

R = !Q3&!Q0; " match 0dd0

EQUATIONS
Q3 := !R&Q2;
Q2 := !R&Q1;
Q1 := !R&Q0;

" Loads 0001 as next state when
" current state is 0dd0
Q0 := !R&!Q3 # R;

[Q0..Q3].CLK = CLOCK;

END
```

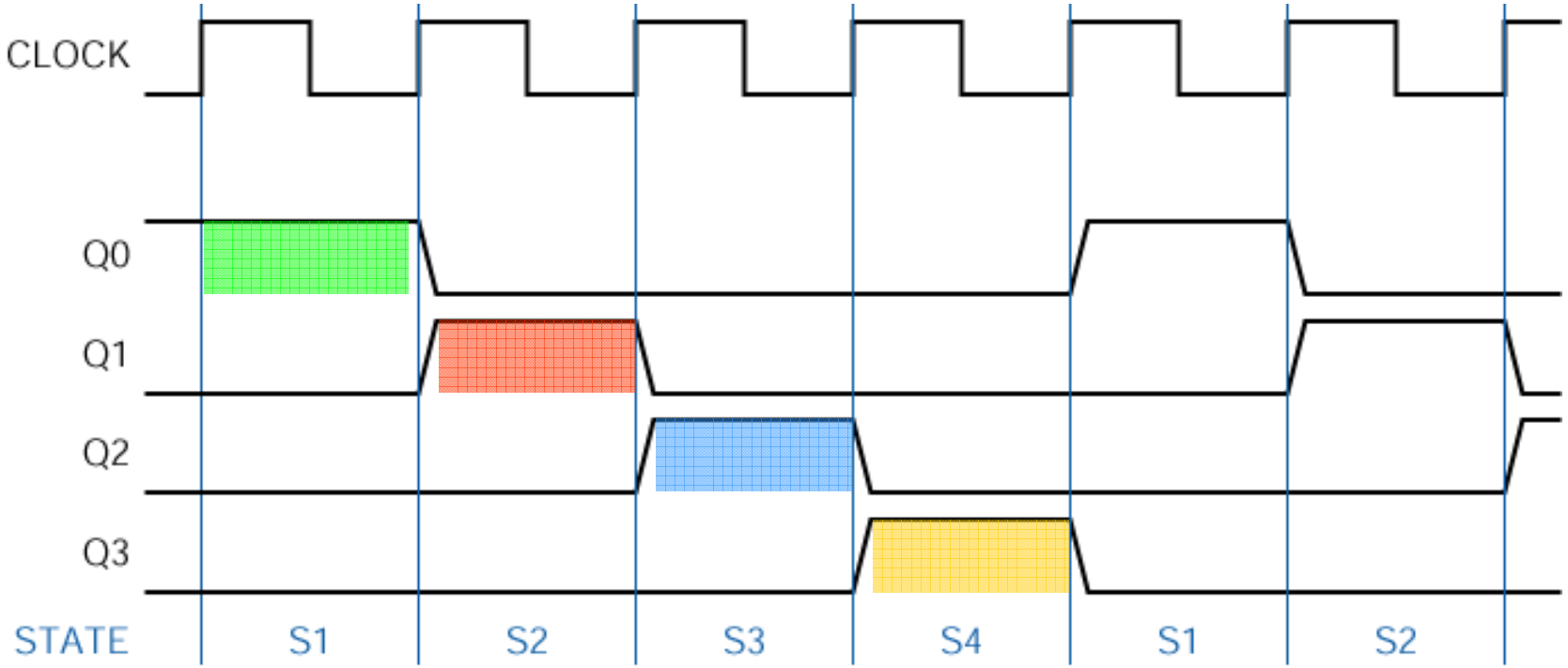


State Decoding

- **What is needed to decode the states of an n-bit (n state) ring counter?**

Nothing – just use state variables directly

Ring Counter State Decoding



S1 = Q0

S2 = Q1

S3 = Q2

S4 = Q3

State Decoding

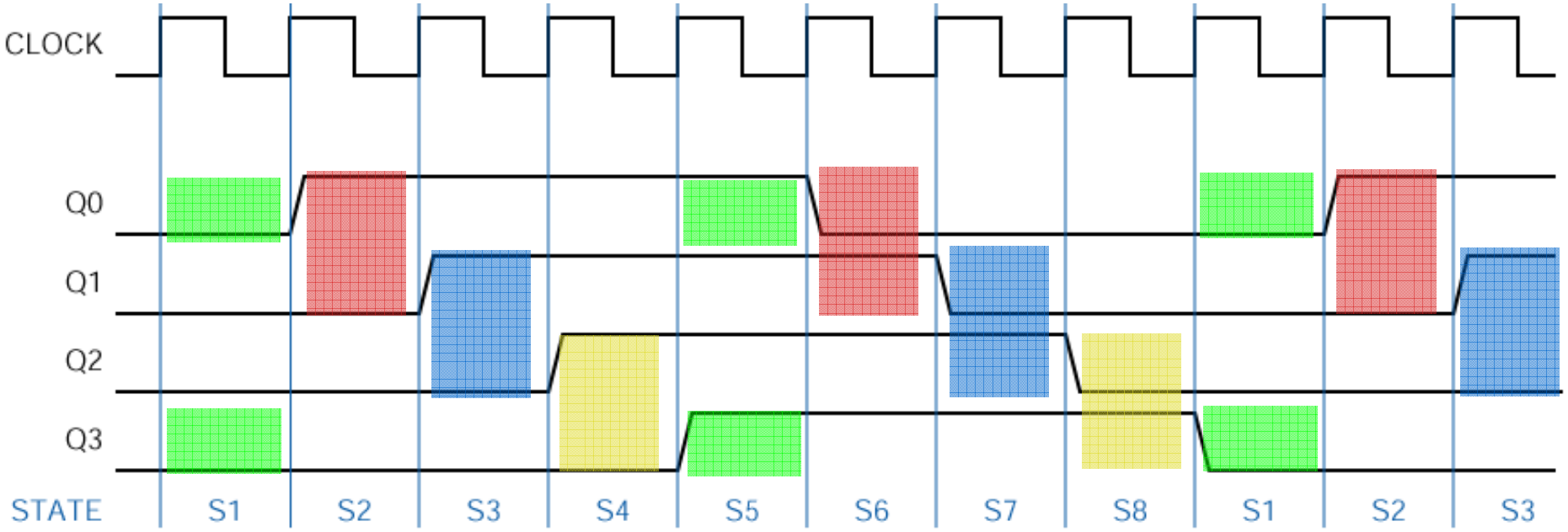
- **What is needed to decode the states of an n-bit (n state) ring counter?**

  Nothing – just use state variables directly

- **What is needed to decode the states of an n-bit (2n state) Johnson counter?**

  2n 2-input AND or NAND gates

Johnson Counter State Decoding



S1 = Q0′• Q3′

S2 = Q0 • Q1′

S3 = Q1 • Q2′

S4 = Q2 • Q3′

S5 = Q0 • Q3

S6 = Q0′• Q1

S7 = Q1′• Q2

S8 = Q2′• Q3

```
MODULE john4scd

TITLE 'Self-Correcting 4-bit Johnson Counter with Decoded Outputs'

" Loads 0001 as next state when current state is 0xx0

DECLARATIONS
CLOCK  pin;
Q0..Q3 pin istype 'reg';
S1..S8 pin istype 'com'; " decoded outputs (text, p. 728)

R = !Q3&!Q0; " match 0xx0

EQUATIONS
Q3 := !R&Q2;
Q2 := !R&Q1;
Q1 := !R&Q0;
Q0 := !R&!Q3 # R;

[Q0..Q3].CLK = CLOCK;

S1 = !Q3 & !Q0;
S2 = !Q1 &  Q0;
S3 = !Q2 &  Q1;
S4 = !Q3 &  Q2;
S5 =  Q3 &  Q0;
S6 =  Q1 & !Q0;
S7 =  Q2 & !Q1;
S8 =  Q3 & !Q2;

END
```

state decoding

State Decoding

- What is needed to decode the states of an n-bit (n state) ring counter?

  Nothing – just use state variables directly

- What is needed to decode the states of an n-bit (2n state) Johnson counter?

  2n 2-input AND or NAND gates

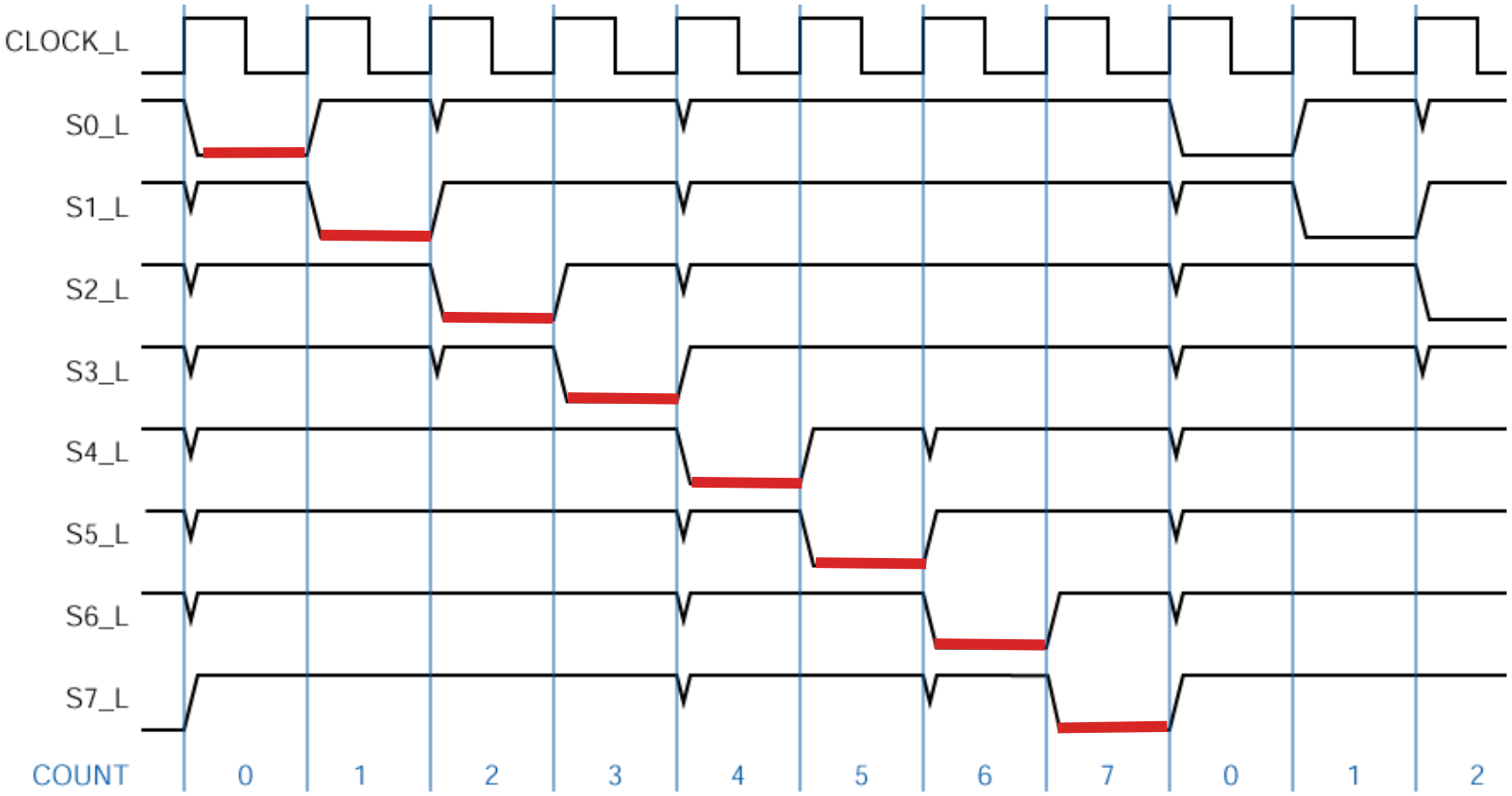
- How does this compare with decoding the states of an n-bit (2^n state) binary counter?

  Need 2^n n-input AND or NAND gates, where n is the number of state variables (or, an n-to-2^n decoder)

State Decoding

- **Problem: Because more than one bit position changes simultaneously in a binary count sequence, there will be “glitches” in the decoded outputs**
- **Solution: Connect the decoder outputs to a register that samples the stable decoded outputs on the next clock tick**

Decoded Outputs of a 3-bit Binary Counter



Thought Questions

- **Give an example of an application where state decoding glitches can cause problems**

  When decoded outputs are used as "clocking" signals

- **Given that 8 glitch-free decoded outputs are required for a given application, which solution would be best: a 3-bit binary counter, decoder, and de-glitching register; or a 4-bit self-correcting Johnson counter?**

  Johnson counter

Thought Questions

- **Is it possible to construct an n-bit counter with 2^n states that can be decoded in a glitch-free fashion?**

  YES – a Gray-code counter

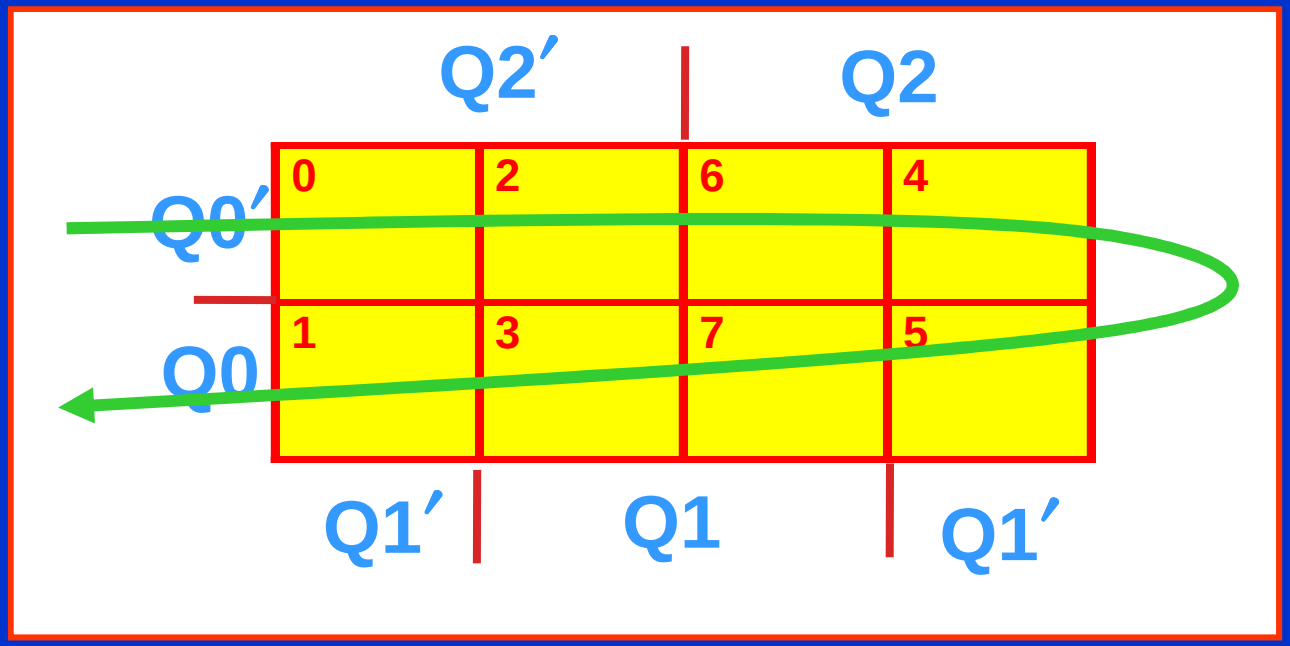
- **If so, what property should the count sequence possess?**

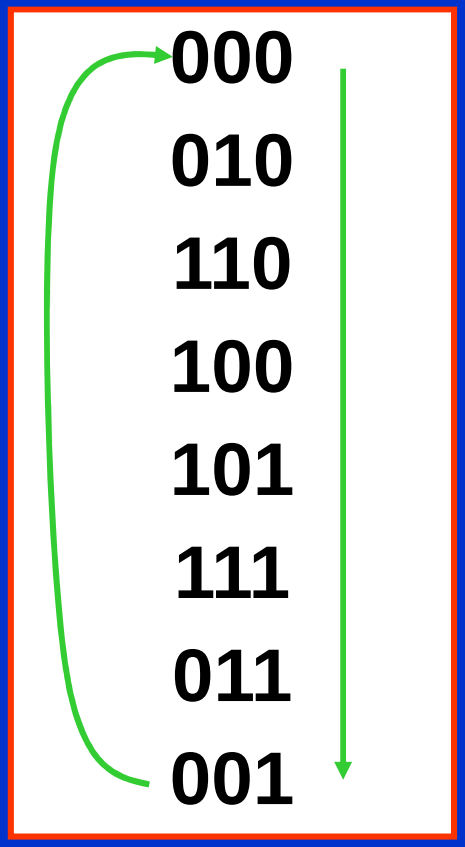
  Each successive combination should differ in only a single bit position

Thought Questions

- **Where have we seen this before?**

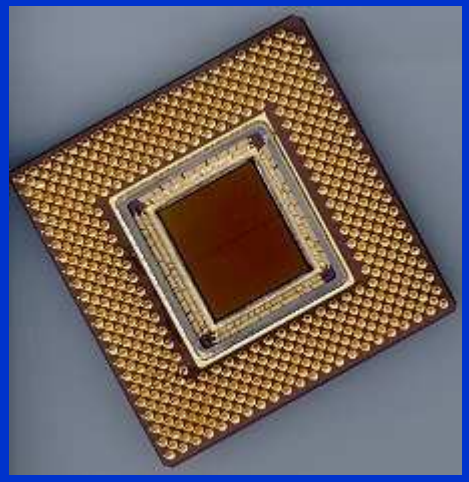
On K-maps!





Summary

- Counters are a common building block used in sequential circuit design, particularly with sequence generator state machines
- There are two basic types of counters
  - binary
  - shift register (types differ based on feedback)
- Counter states can be decoded different ways (some are glitch-free, others are not)
  - binary: standard decoders, not glitch-free
  - Johnson: 2-input AND gates, glitch-free
  - ring: nothing (use flip-flop outputs directly), glitch-free



Introduction to Digital System Design

Module 3-H
State Machine Design Examples: Sequence Recognizers

Reading Assignment:

DDPP 4th Ed., pp. 580-587

Learning Objective:

- **Identify states utilized by a sequence recognizer: accepting sequence, final, and trap**
- **Determine the embedded binary sequence detected by a sequence recognizer**

Outline

- Overview
- Simple pattern recognizer
- Digital lock
- Summary

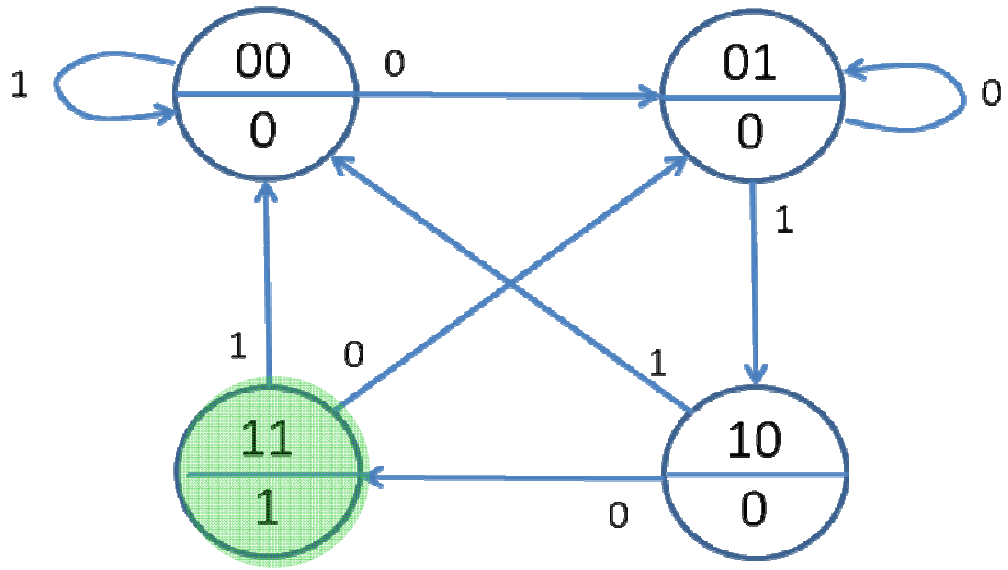
Overview

- A ***sequence recognizer*** state machine responds to a ***pre-defined input pattern*** of signal assertions and produces corresponding output signal assertions
  - digital lock / access code control
  - bit sequence detector
- Use of **Moore** models to design sequencer recognizer is generally **preferred**, because you typically ***don't want any output signals to change*** (based on input signal changes) ***until the machine is clocked to the next state*** (i.e., the outputs should only be a function of the state variables)
- Because "action" (output signal assertions) occur in response to a pre-defined pattern, a sequence recognizer has different kinds of **"final states"** (denoted with concentric circles on ST diagram):
  - final state of accepting sequence (e.g., "unlock")
  - trap state (e.g., "alarm")

Example – Simple Pattern Recognizer

- **Assuming the state machine is initialized to state 00, determine the output sequence generated in response to the following input sequence: 1 1 0 1 0 0 0 1 0 0**

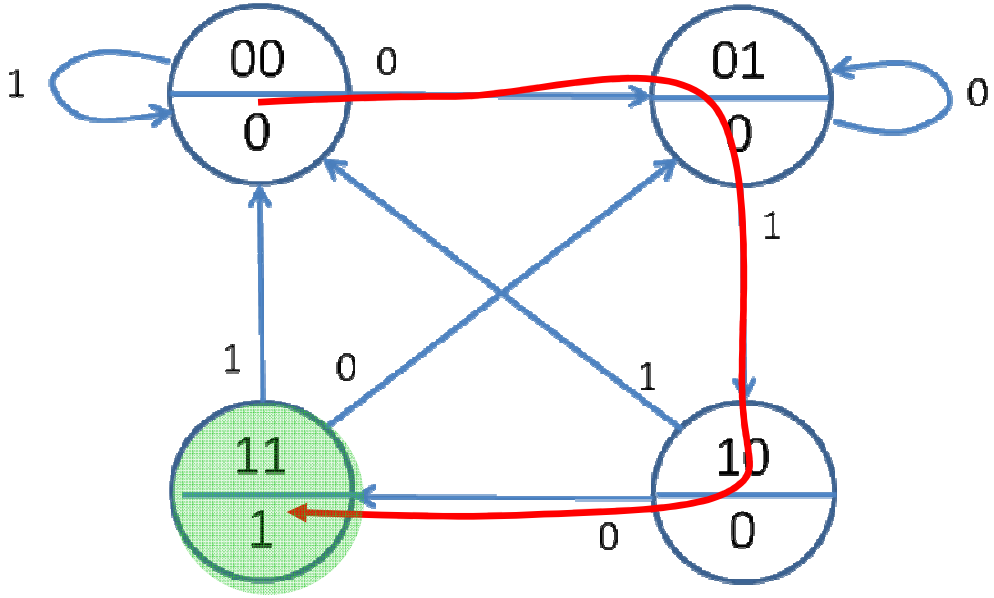
0 0 0 0 1 0 0 0 1 0



final state in pattern accepting sequence

Example – Simple Pattern Recognizer

- **Assuming the state machine is initialized to state 00, determine the output sequence generated in response to the following input sequence: 1 1 0 1 0 0 0 1 0 0**



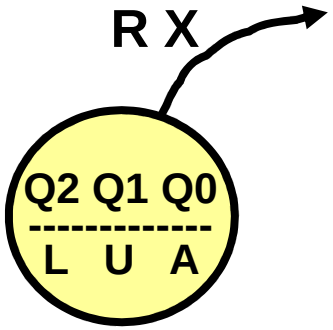
- **Determine the embedded binary sequence recognized by this state machine: 0 1 0**

Example - Digital Combination Lock

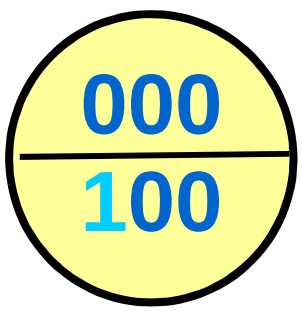
- **Design a digital combination lock**
  - **unlocks when a fixed combination (binary sequence) is entered: 101110**
  - **has three inputs:**
    - **X – combination data**
    - **R – relock / reset**
    - **RESET – asynchronous reset**
  - **has three output signals:**
    - **LOCKED**
    - **UNLOCKED**
    - **ALARM**

Example - Digital Combination Lock

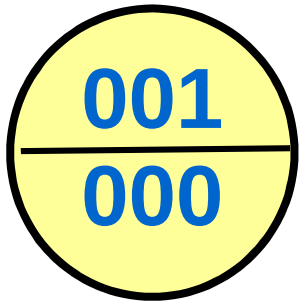
- Implement using Moore model
  - will need an initial “locked” state
  - will need six states to accept digits of combination (the last is “unlocked”)
  - will need an “alarm” state
  - total number of states is eight; therefore, can implement with three state variables
- Types of states
  - accepting sequence (entering combination)
  - final state (sequence correctly entered)
  - trap state (error made while entering combination)

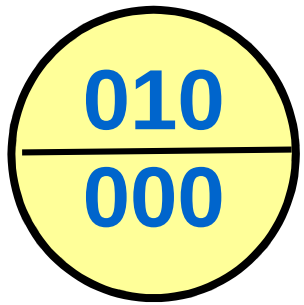


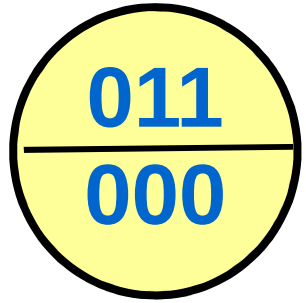
Combination: 101110

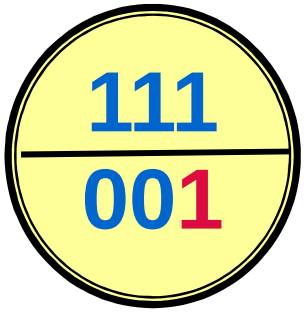


LOCKED

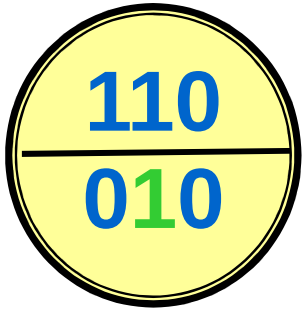




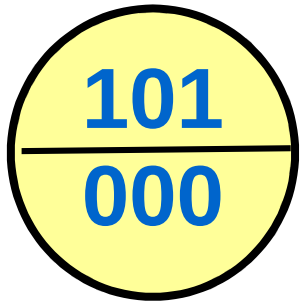


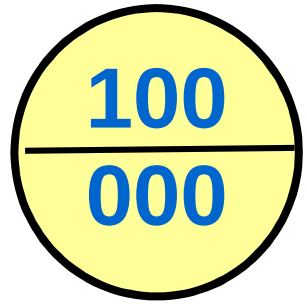


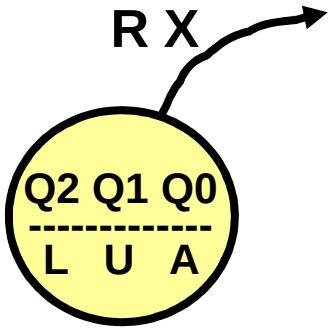
ALARM



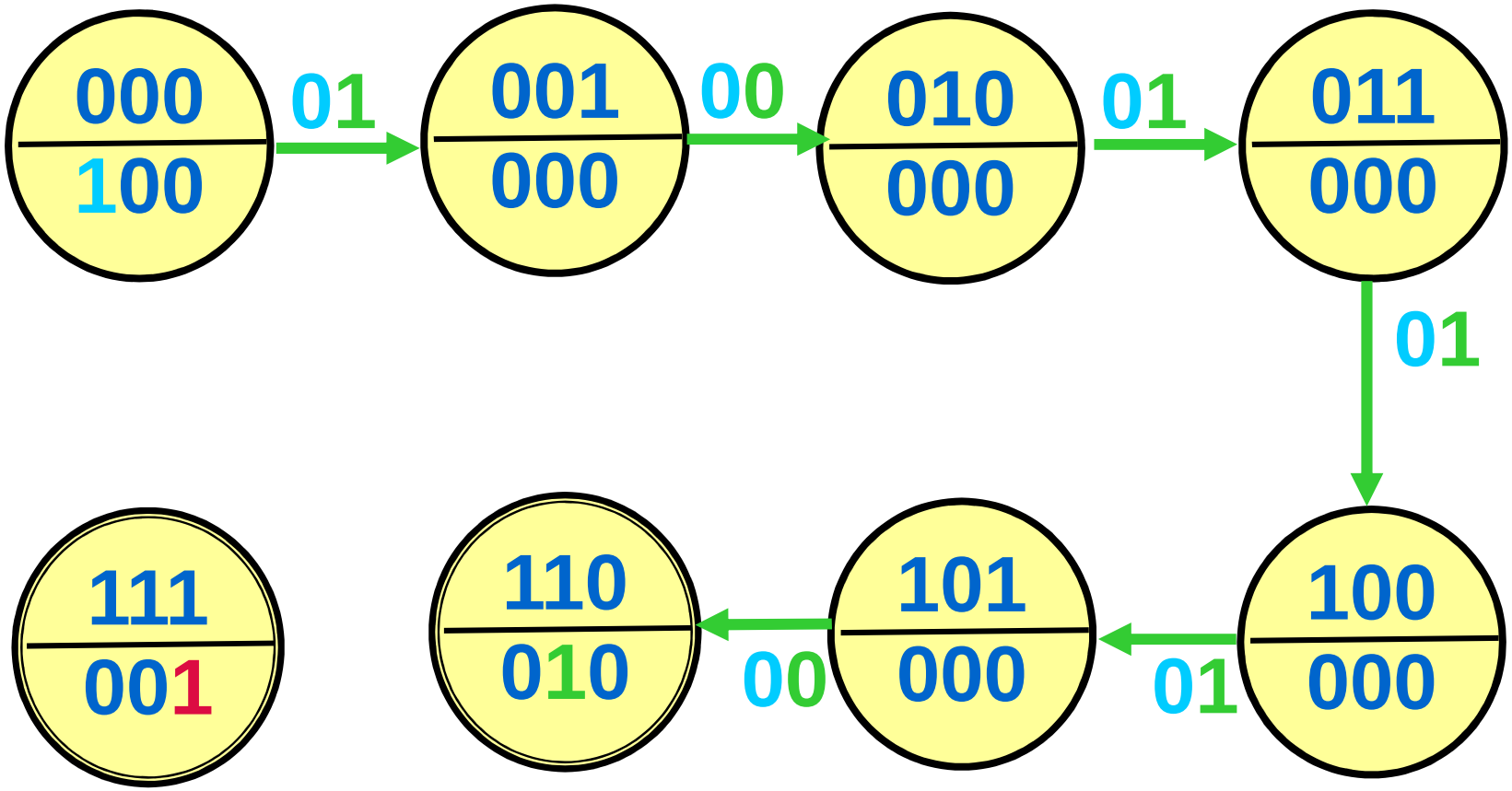
UNLOCKED

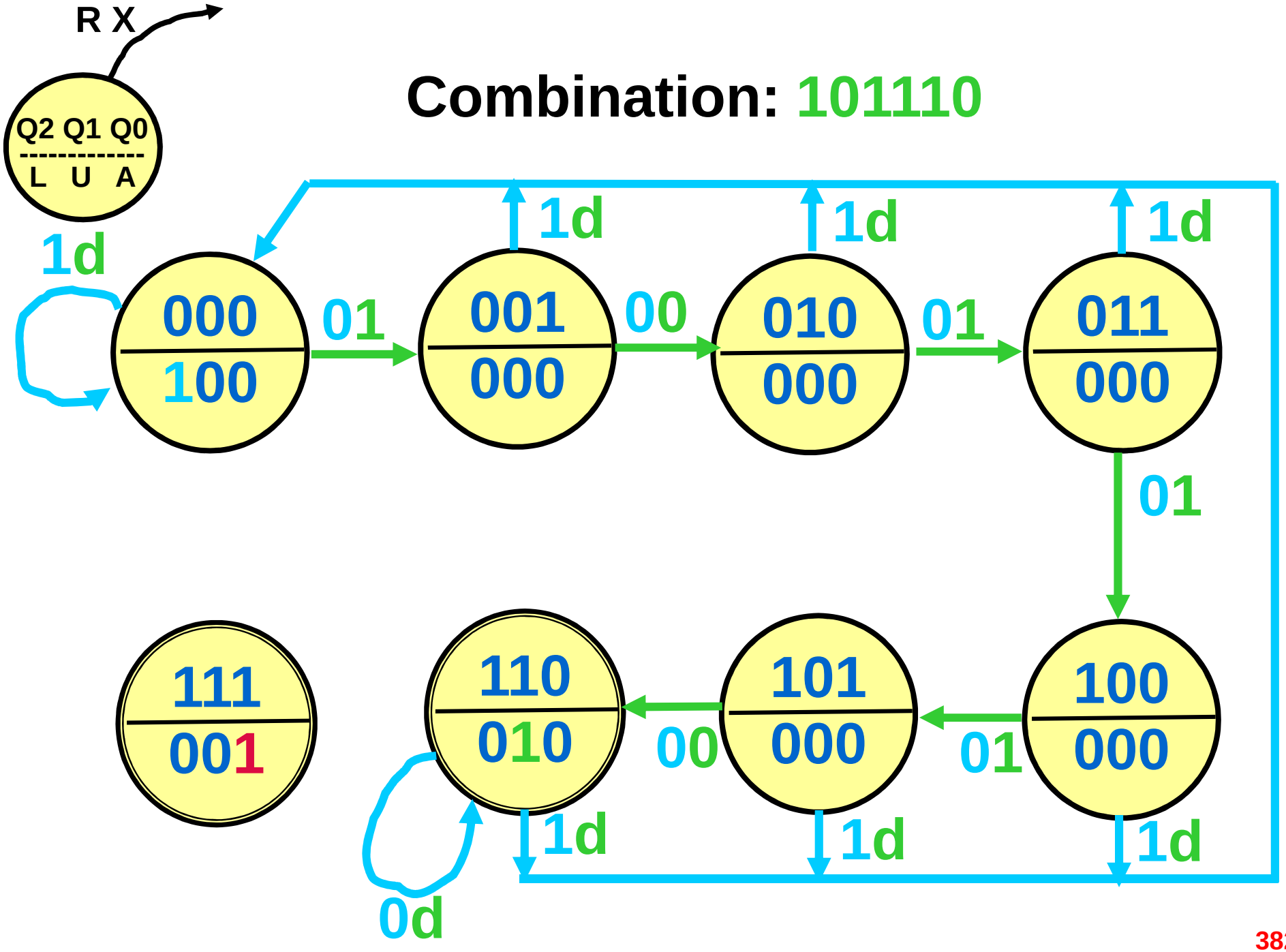


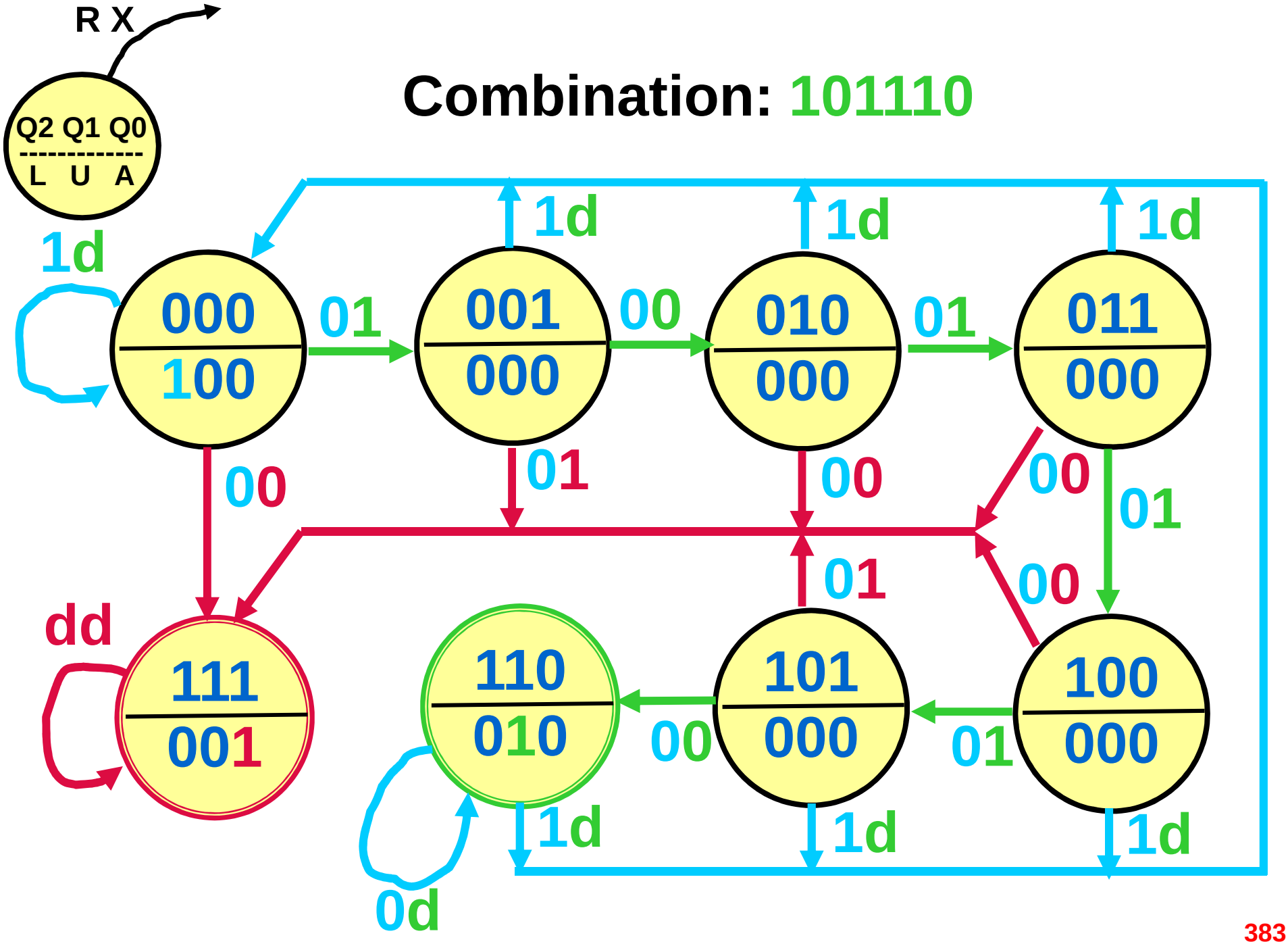




Combination: 101110




R X
Q2 Q1 Q0
L U A
Combination: 101110
1d
000
100
01
001
000
00
010
000
01
011
000
1d
1d
1d
01
100
000
01
101
000
00
110
010
1d
1d
1d
111
001
0d


R X
Q2 Q1 Q0
L U A
Combination: 101110
000
100
001
000
010
000
011
000
111
001
110
010
101
000
100
000
1d
01
00
dd
0d

```
MODULE dcl

TITLE 'Digital Combination Lock'

X pin;      "combination data input"
R pin;      "relock input"
RESET pin;  "asynchronous reset"
CLOCK pin;
Q2, Q1, Q0 pin istype 'reg';
LOCKED   pin istype 'com';  "LOCKED indicator"
UNLOCKED pin istype 'com';  "UNLOCKED indicator"
ALARM    pin istype 'com';  "ALARM indicator"

QALL = [Q2,Q1,Q0];
A0   = [ 0, 0, 0];
A1   = [ 0, 0, 1];
A2   = [ 0, 1, 0];
A3   = [ 0, 1, 1];
A4   = [ 1, 0, 0];
A5   = [ 1, 0, 1];
A6   = [ 1, 1, 0];
A7   = [ 1, 1, 1];

STATE_DIAGRAM QALL

state A0:             if (R==1) then A0
          else if (R==0)&(X==0) then A7
          else if (R==0)&(X==1) then A1;

state A1:             if (R==1) then A0
          else if (R==0)&(X==0) then A2
          else if (R==0)&(X==1) then A7;

state A2:             if (R==1) then A0
          else if (R==0)&(X==0) then A7
          else if (R==0)&(X==1) then A3;
```

```
state A3:             if (R==1) then A0
          else if (R==0)&(X==0) then A7
          else if (R==0)&(X==1) then A4;

state A4:             if (R==1) then A0
          else if (R==0)&(X==0) then A7
          else if (R==0)&(X==1) then A5;

state A5:             if (R==1) then A0
          else if (R==0)&(X==0) then A6
          else if (R==0)&(X==1) then A7;

state A6:             if (R==1) then A0
          else if        (R==0) then A6;

state A7: goto A7;

EQUATIONS

QALL.CLK = CLOCK;
QALL.AR = RESET;

LOCKED = !Q2&!Q1&!Q0;
UNLOCKED = Q2&Q1&!Q0;
ALARM = Q2&Q1&Q0;

END
```

Summary

- A sequencer recognizer is a state machine that produces output signal assertions in response to an input pattern
- Output signal assertions typically occur when the machine enters a “final state” associated with the accepting sequence
- Sequence recognizers are typically realized with Moore models, to prevent “spurious” behavior that might occur if the machine’s outputs could potentially change in response to an input signal change without clocking it