Actuating, Sensing and Control Mechatronic Systems

Grzegorz Góra PhD

D1-Lab 20

ggora@agh.edu.pl

http://home.agh.edu.pl/~ggora/

Department of Robotics and Mechatronics

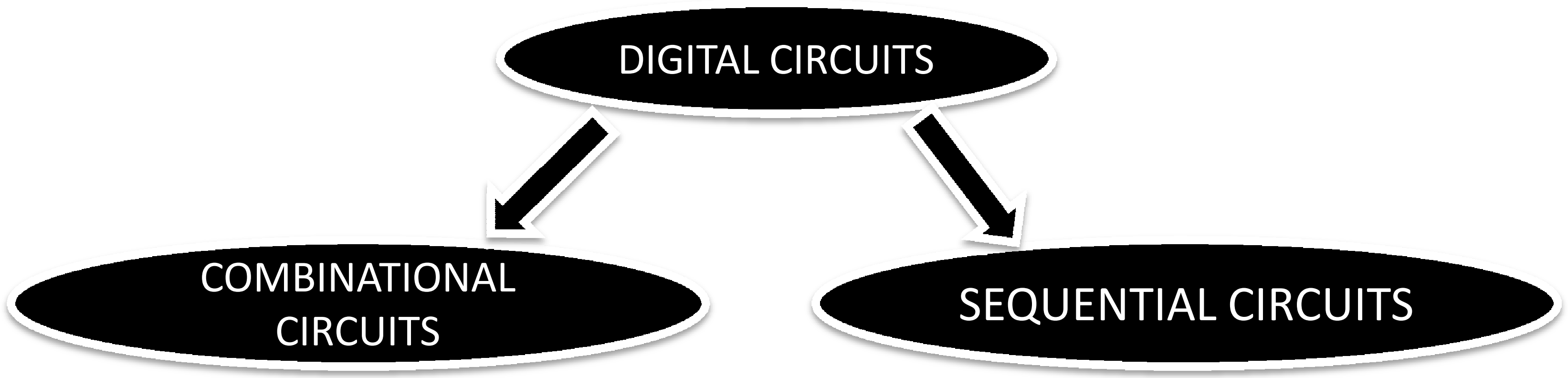
Faculty of Mechanical Engineering and Robotics

AGH University of Science and Technology

Agenda

1. Combinational and Sequential Logic Circuits
2. Flip-flops
3. Moore State Machine
4. Mealye State Machine
5. State Mechine Synthesis
6. Example: Guarded railway crossing

Digital circuits: combinational and sequential circuits



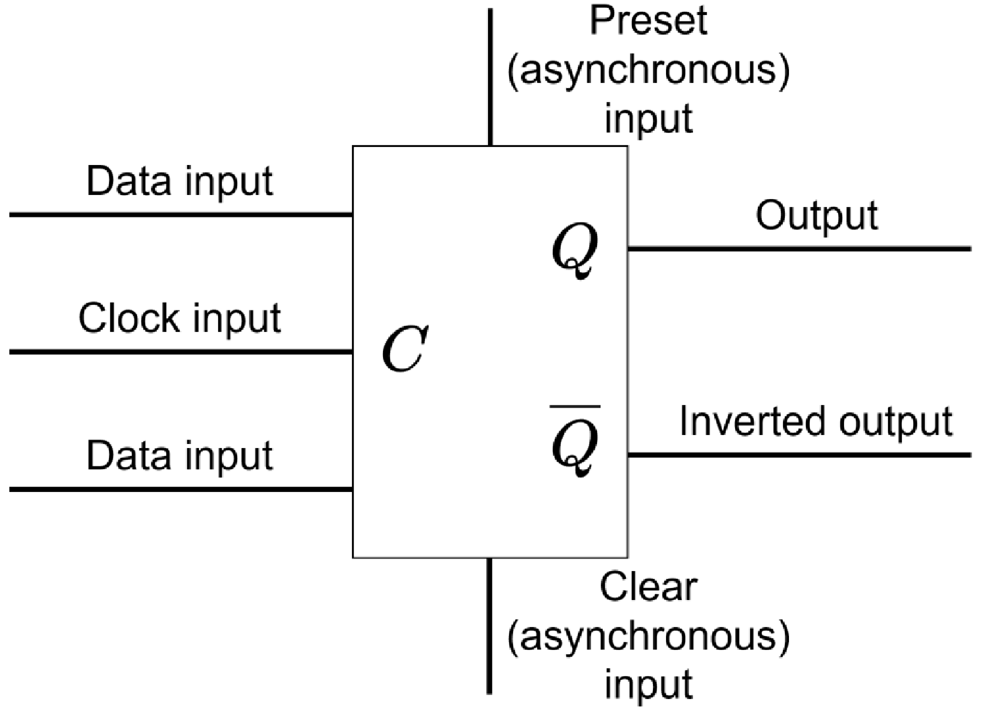
COMBINATIONAL CIRCUITS *– is a type of logic circuit whose* ***output depends only on the present value of its input signals****. Combinational cicruits are eg. adders, substractors,* multiplexers*, comparators, transcoders, etc.*

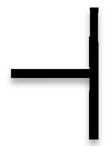
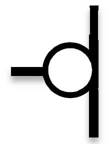
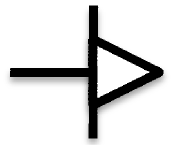
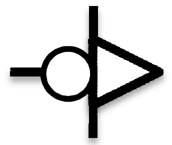
SEQUENTIAL CIRCUITS *– is a type of logic circuit whose* ***output depends on the present value of its input signals and on the sequence of past inputs (previous state of the circuit)****, eg. counters, state machines.*

- *Sequential circuits contain elements that store the current state.*
- *Flip-flops are used to store the current state.*
- *In sequential circuits, the flip-flop serves as a 1-bit memory.*
- *The flip-flop state (stored value) in synchronous systems is updated on each rising (or falling) edge of the clock.*

Flip-flop

Flip-flops are systems that have two stable states in which information can be stored. The system can change state using signals applied to one or more control (data) inputs. Flip-flop is the basic memory element in sequential logic.



Input symbol	Input definition
	**Static input with active state 1**
	**Static input with active state 0**
	**Dynamic input with active state change 0 to 1 (rising edge)**
	**Dynamic input with active state change from 1 to 0 (falling edge)**

Special flip-flop inputs

EN (*enable*), G (*gate*), STB (*strobe*), LE (*lach enable*) – gating flip-flop input

CE (*clock enable*) – clock signal blocking input

PRE (*preset*), AS (*asynchronous set*) – asynchronous input setting the flip-flop

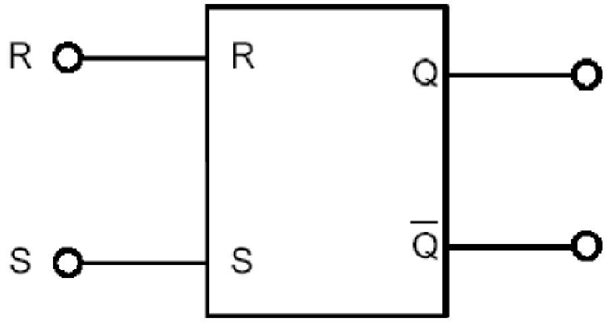
CLR (*clear*), AR (*asynchronous reset*) – asynchronous flip-flop reset input

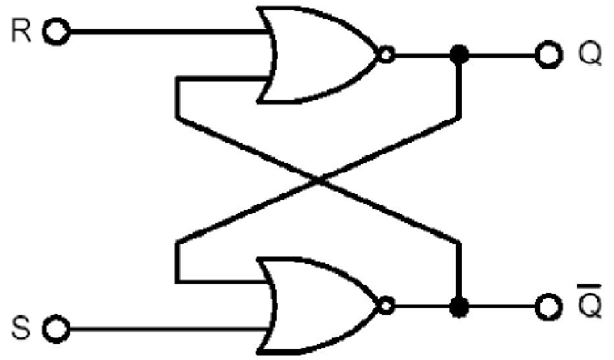
S (*set*), SET, SS (*synchronous set*) – synchronous input setting the flip-flop

R (*reset*), RES, SR (*synchronous reset*) – synchronous flip-flop reset input

OE (*output enable*) – input blocking the flip-flop output

RS - Flip_flop (asynchronous)





S	R	Q_n	$\overline{Q}_n$
0	0	Q_{n-1}	$\overline{Q}_{n-1}$
0	1	0	1
1	0	1	0
1	1	0	0

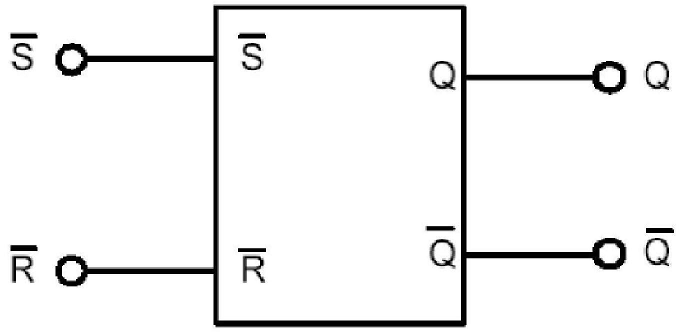
$S \bullet R = 0$

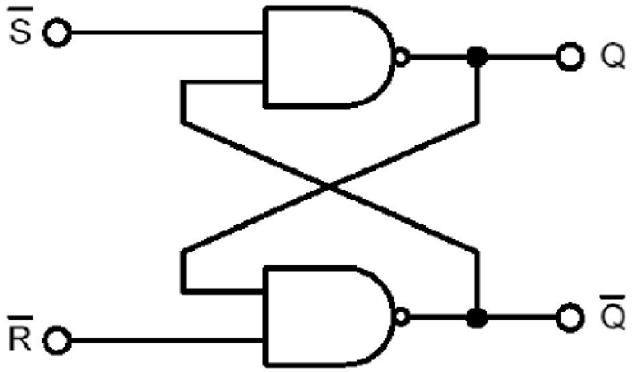
$Q_{n-1} \rightarrow Q_n$	S	R
$0 \rightarrow 0$	0	X
$0 \rightarrow 1$	1	0
$1 \rightarrow 0$	0	1
$1 \rightarrow 1$	X	0

Forbidden state –
in the forbidden state,
both outputs are '0'

Q \ SR	00	01	11	10
0	0	0	-	1
1	1	0	-	1

RS - Flip_flop (asynchronous)





$\overline{S}$	$\overline{R}$	Q_n	$\overline{Q}_n$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Q_{n-1}	$\overline{Q}_{n-1}$

$\overline{S} \vee \overline{R} = 1$

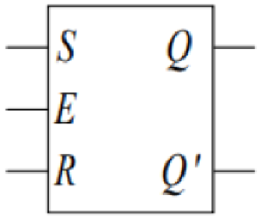
$Q_{n-1} \rightarrow Q_n$	$\overline{S}$	$\overline{R}$
$0 \rightarrow 0$	1	X
$0 \rightarrow 1$	0	1
$1 \rightarrow 0$	1	0
$1 \rightarrow 1$	X	1

Q \ $\overline{S}\overline{R}$	00	01	11	10
0	-	1	0	0
1	-	1	1	0

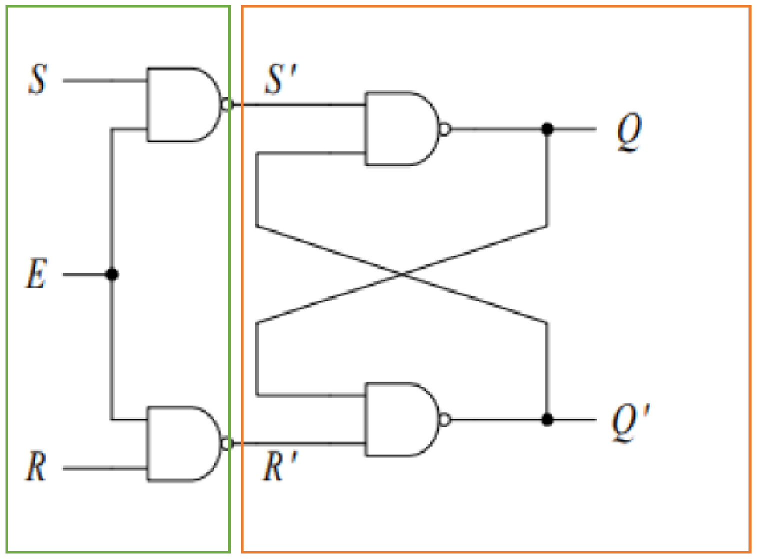
Forbidden state –
in the forbidden state, both outputs are ‘1'

RS - Latch (asynchronous)

Logic symbol



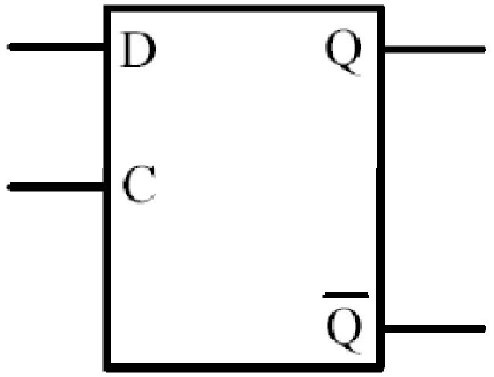
Circuit

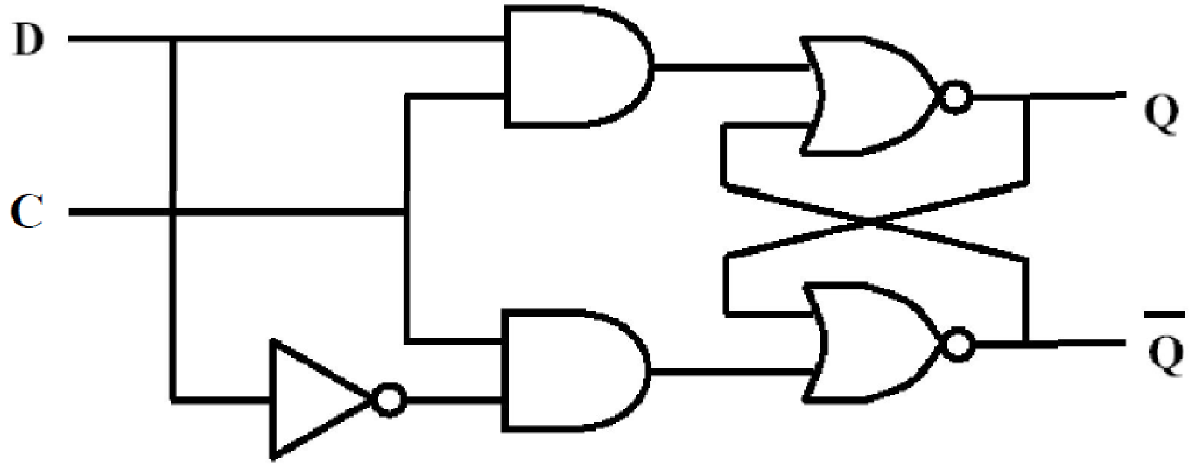


Truth table

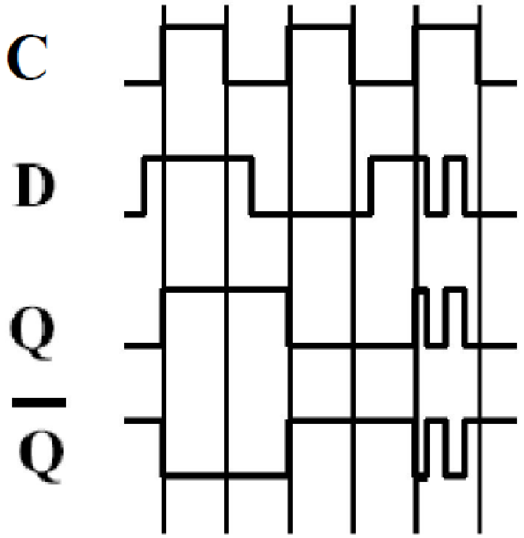
Enable	S	R	Q	$\overline{Q}$
0	0	0	Latch	
0	0	1	Latch	
0	1	0	Latch	
0	1	1	Latch	
1	0	0	Latch	
1	0	1	0	1
1	1	0	1	0
1	1	1	INVALID	

D-Latch (Data-Latch)



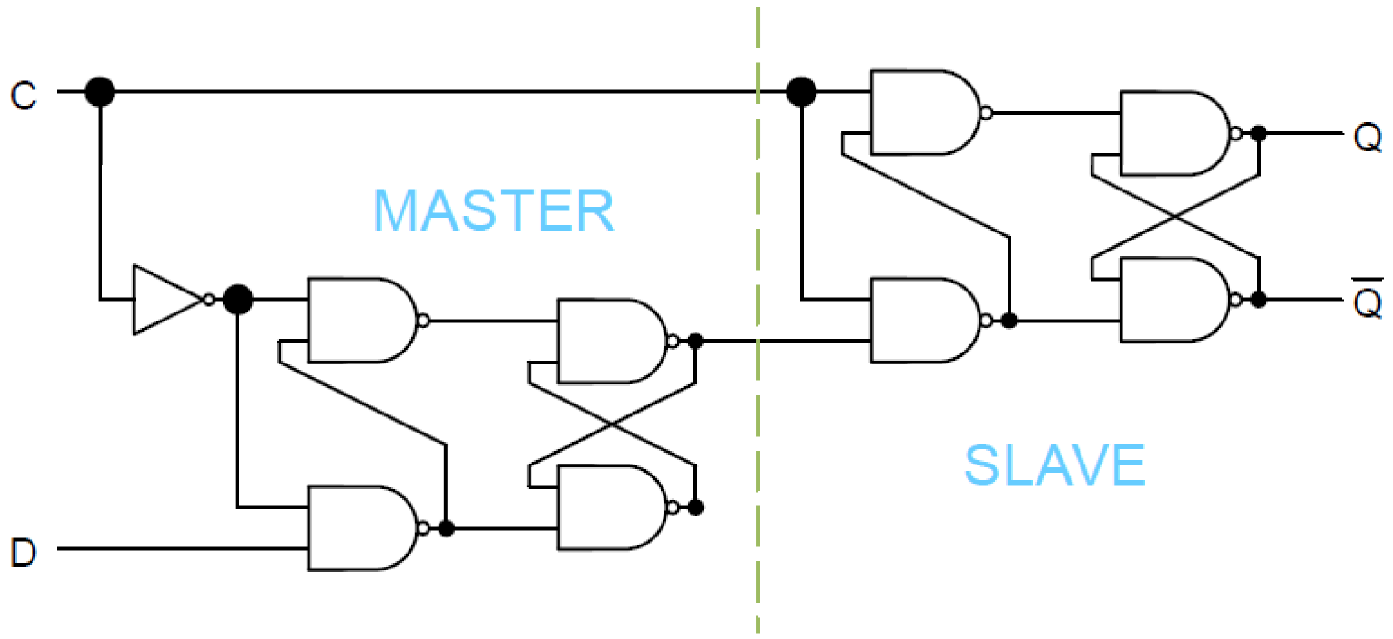


C	D	Q
0	0	Q_{n-1}
0	1	Q_{n-1}
1	0	0
1	1	1

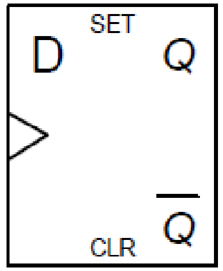


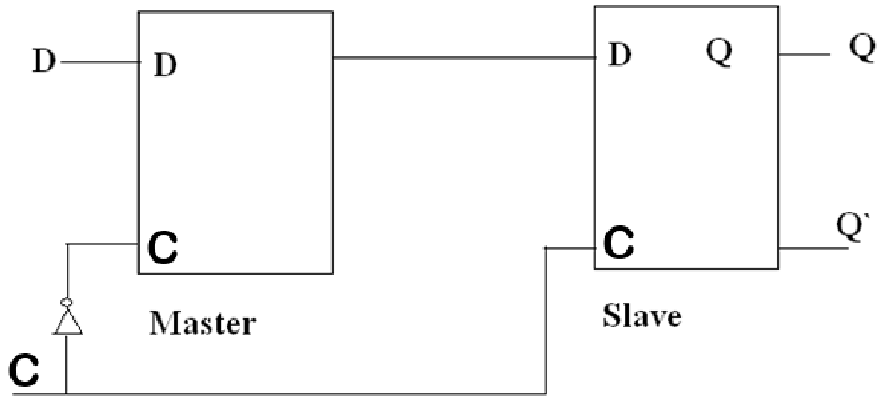
$Q_{n-1} \rightarrow Q_n$	D
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

Master-Slave D Flip-flop

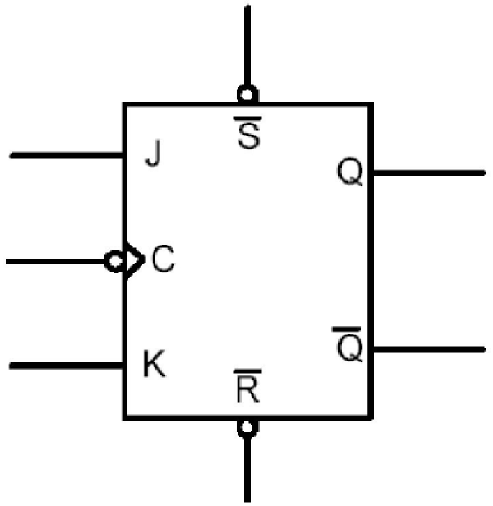


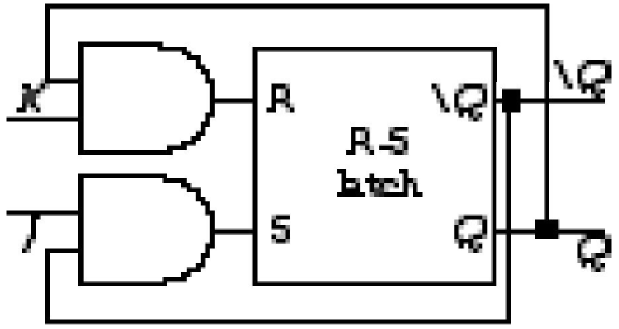
C	D	Q
0	x	Q_{-1}
1	x	Q_{-1}
↓	x	Q_{-1}
↑	0	0
↑	1	1





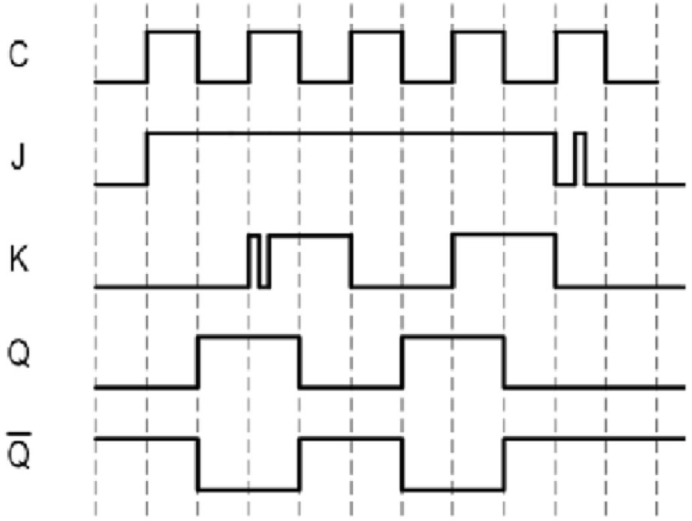
JK – Flip-flop





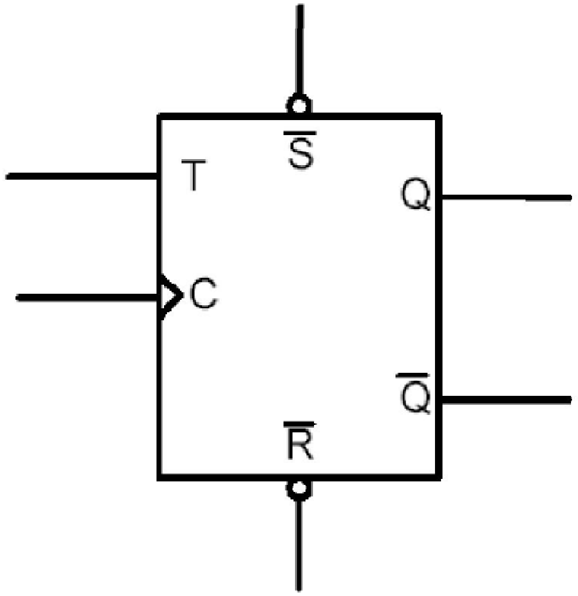
J	K	Q_n
0	0	Q_{n-1}
0	1	0
1	0	1
1	1	$\overline{Q}_{n-1}$

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1



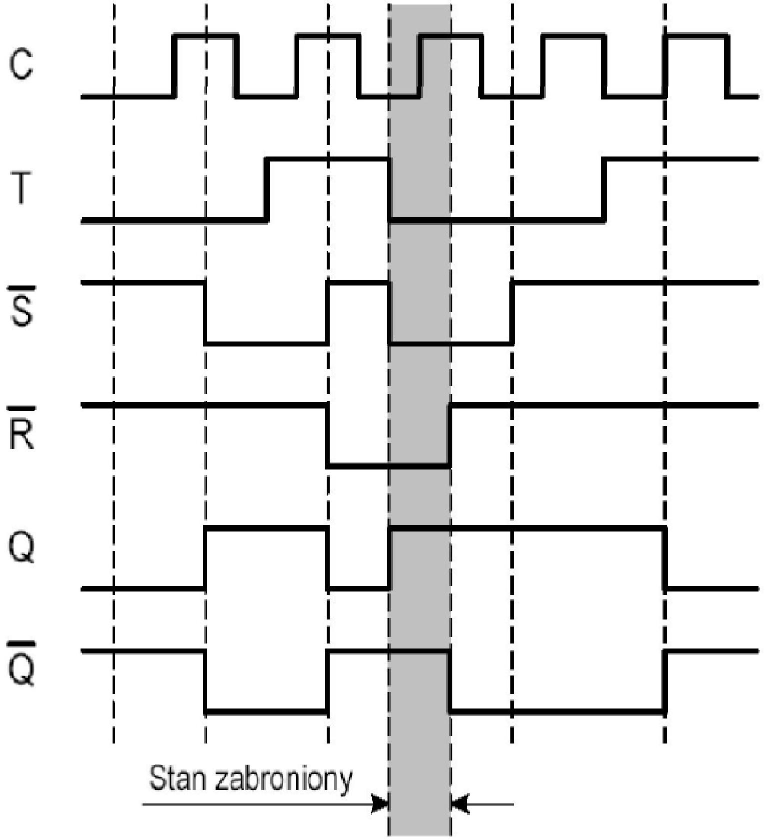
$Q_{n-1} \rightarrow Q_n$	J	K
$0 \rightarrow 0$	0	X
$0 \rightarrow 1$	1	X
$1 \rightarrow 0$	X	1
$1 \rightarrow 1$	X	0

T – Flip-flop

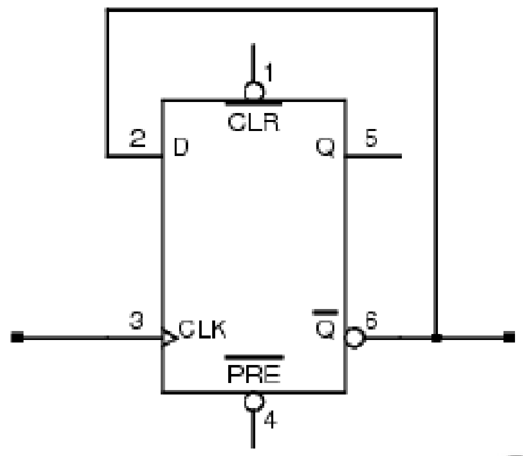


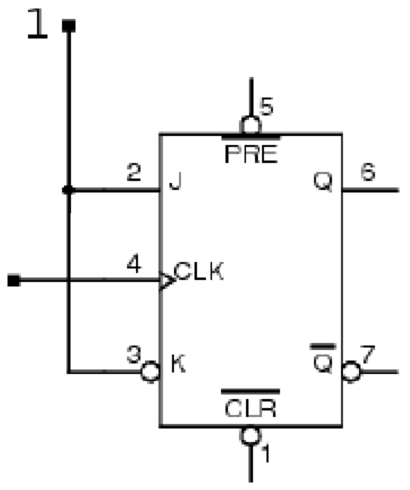
T	Q_n
0	Q_{n-1}
1	$\overline{Q}_{n-1}$

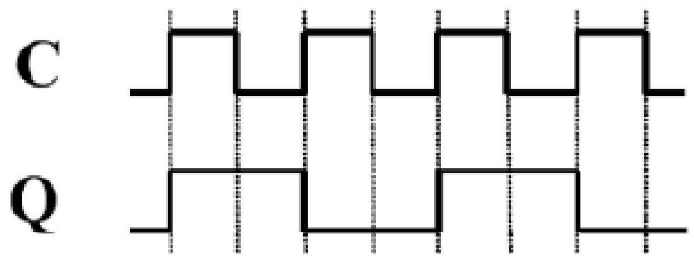
$Q_{n-1} \rightarrow Q_n$	T
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	1
$1 \rightarrow 1$	0

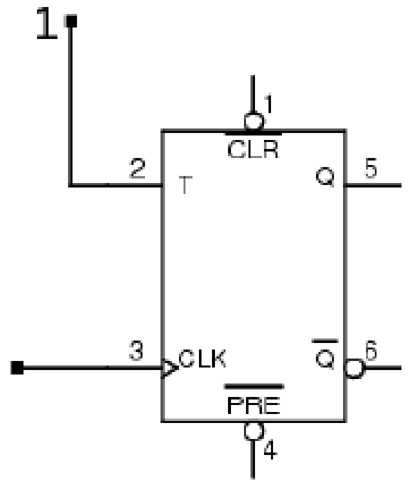


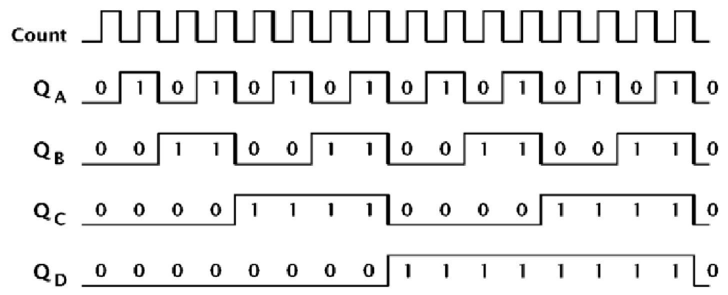
Toggle mode

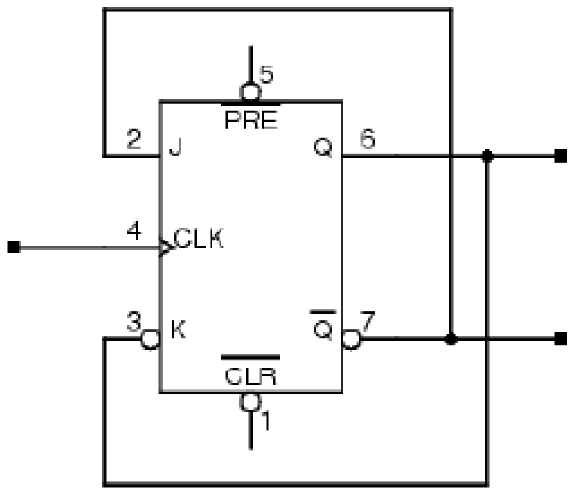

1
CLR
2
D
Q
5
3
CLK
Q̄
6
PRE
4


1
5
PRE
2
J
Q
6
4
CLK
3
K
Q̄
7
CLR
1


C
Q

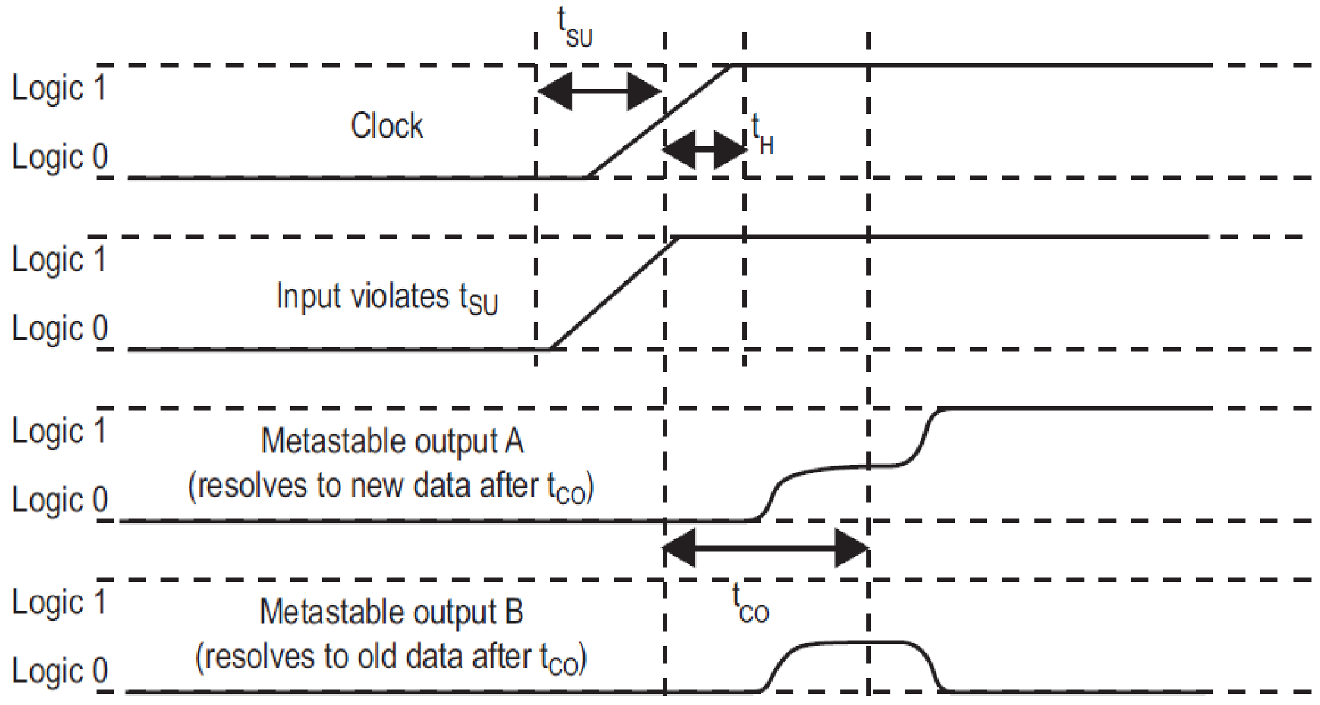

1
1
CLR
2
T
Q
5
3
CLK
Q̄
6
PRE
4


Count
Q A 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
Q B 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
Q C 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0
Q D 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0


5
PRE
2
J
Q
6
4
CLK
3
K
Q̄
7
CLR
1

Metastability

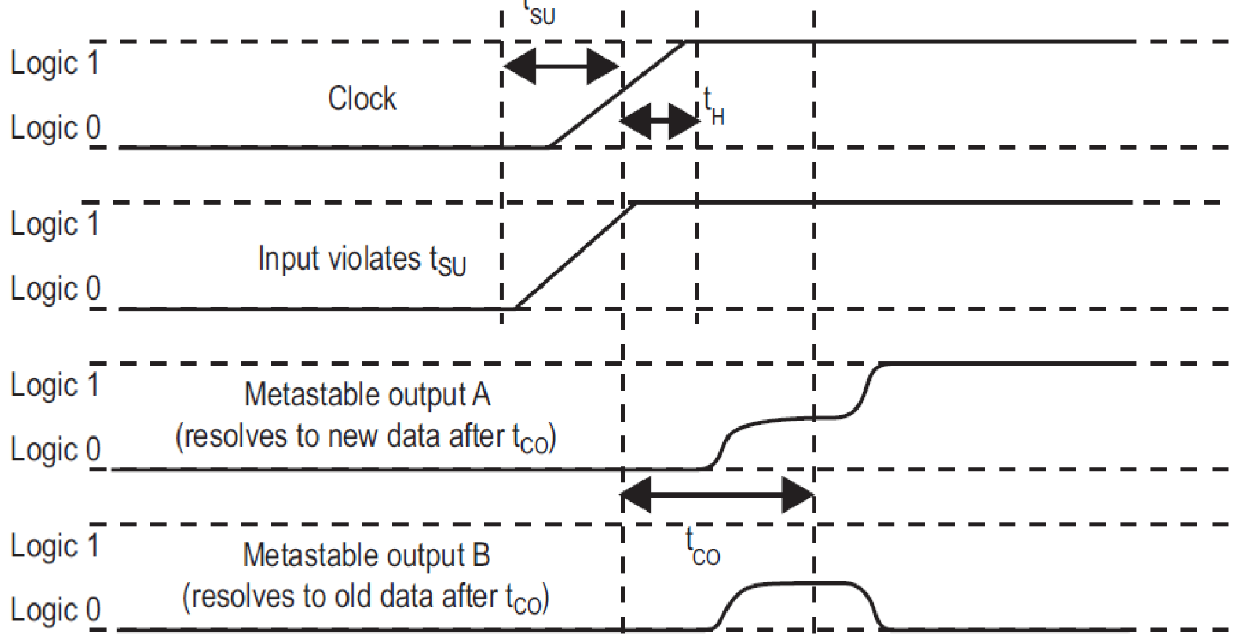
All registers in digital systems have defined signal timing requirements that allow each register to correctly capture data at its inputs and produce output signals. To ensure correct operation, the input to a register must be stable for a minimum time before the clock edge and for a minimum time after the clock edge (these times are assigned as setup time t_{SU} and hold time t_H). Then, the register output is available after a specified delay assigned as clock-to-output delay (t_{CO}).



Metastability

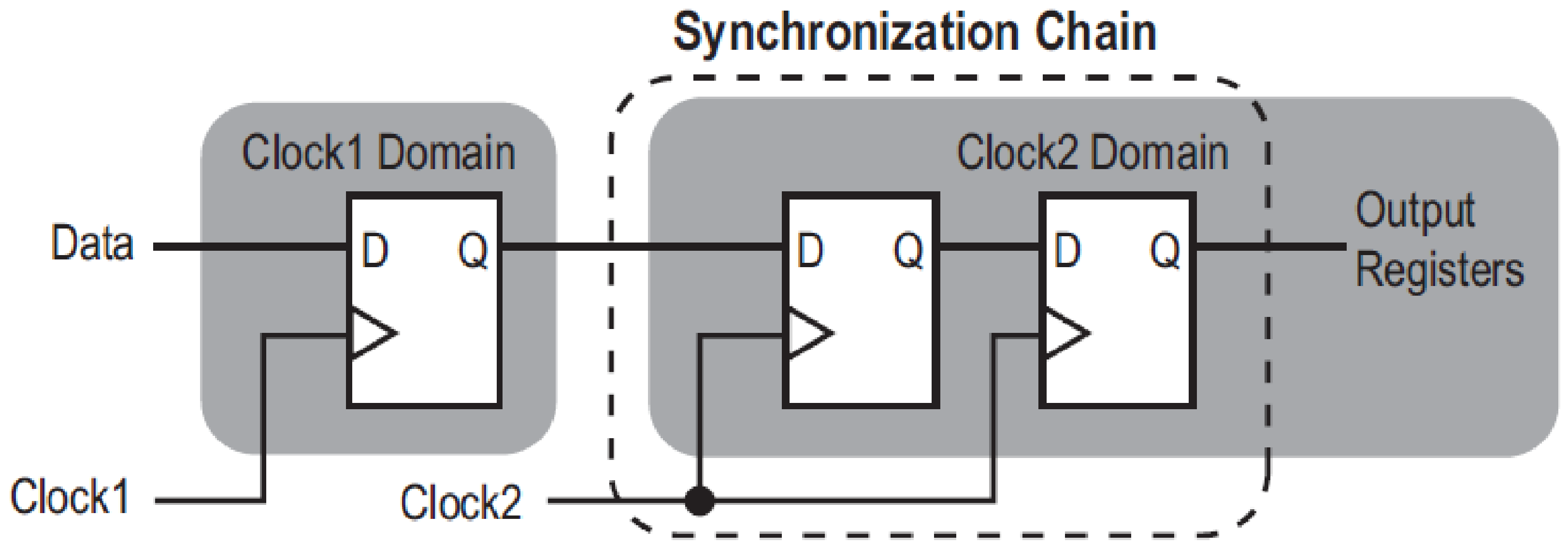
If a data signal transition don't meet these requirements, the output signals can go to a metastable state. In a metastable state, the register output hovers at a value between the high and low states for some period of time. This means that the output transition to a defined high or low state is delayed beyond the specified clock-to-output delay (t_{CO}). For this reason, the phenomenon of metastability is very dangerous.

In synchronous systems, the input signals always meet the register timing requirements, so metastability doesn't appear. Metastability problems commonly appear when a signal is transferred in asynchronous clock domains (for example it is input signal from another device). In that case, the designer cannot guarantee that the signal will meet timing requirements, because the signal can arrive at any time relative to the destination clock.

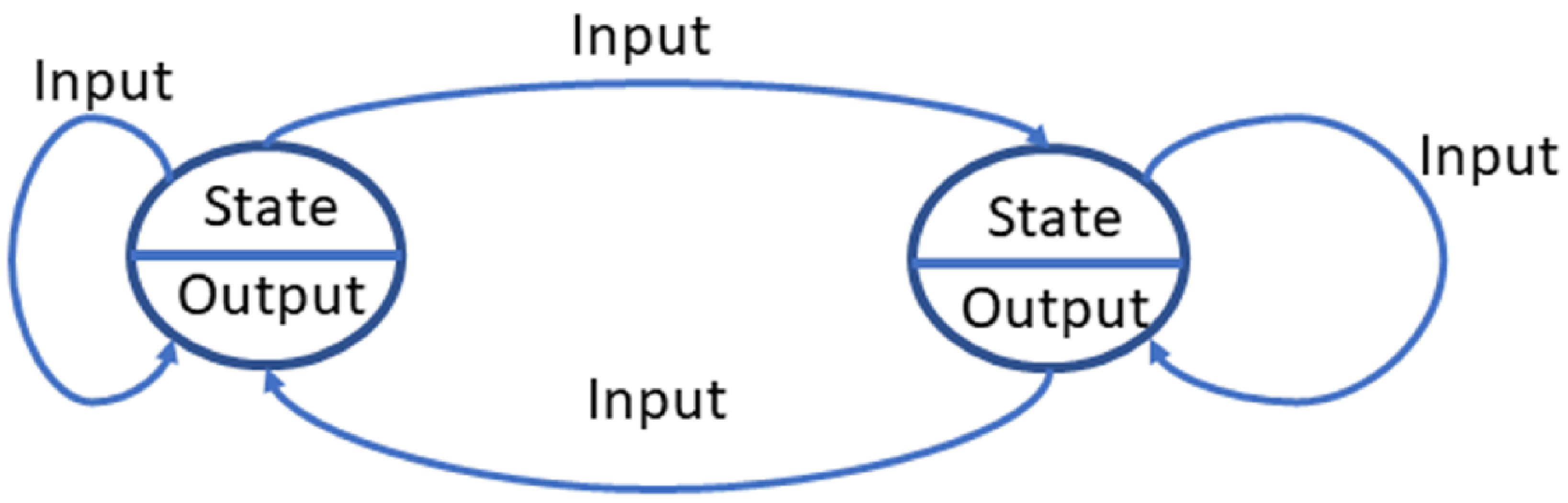


Metastability

When a signal is transferred between circuits in asynchronous clock domains, it is necessary to synchronize this signal to the new clock domain before it can be used. The first register in the new clock domain acts as a synchronization register.
To minimize the failures due to metastability in asynchronous signal transfers, circuit designers typically use a sequence of registers (a synchronization register chain or synchronizer) in the destination clock domain to resynchronize the signal to the new clock domain. These registers allow additional time for a potentially metastable signal to resolve to a known value before the signal is used in the rest of the design.



Moore State Machine



Generic Moore model

- The most frequently used form of description (behavior) of state machine is a diagram.
- The states of the machine are represented by circles.
- Transitions between states are represented by arrows.
- The transition conditions (combinations of inputs) are placed on the arrows.
- **In a Moore machine, the state of the outputs is clearly related to the state of the machine. The status of the outputs is placed inside the circles.**

Moore State Machine

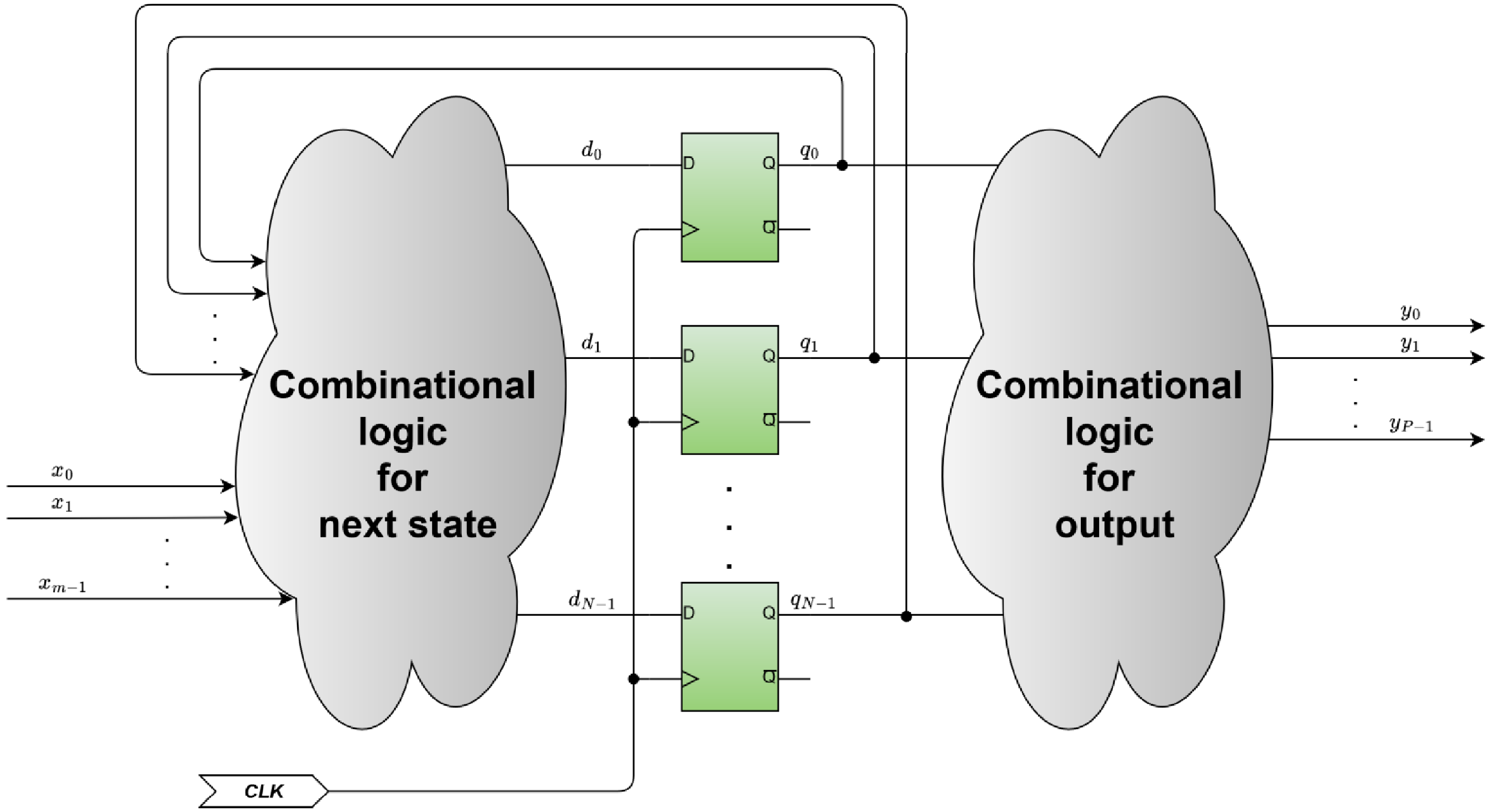
Q – machine state
(q_{N-1}, … q_1, q_0);

X – inputs
(x_{M-1}, … x_1, x_0);

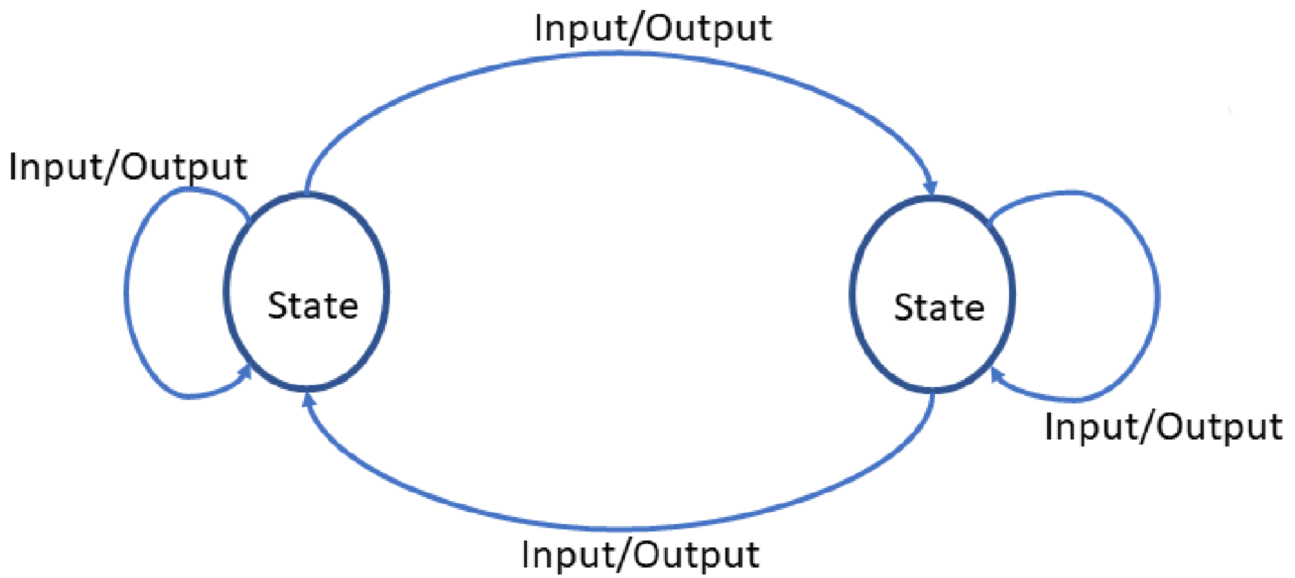
Y – outputs
(y_{P-1}, … y_1, y_0);

Transition logic:
$Q_{n+1} = f(X, Q_n)$;

Output logic:
$Y = f(Q_n)$;



Automat Mealy'ego



Generic Mealy model

- The most frequently used form of description (behavior) of state machine is a diagram.
- The states of the machine are represented by circles.
- Transitions between states are represented by arrows.
- The transition conditions (combinations of inputs) are placed on the arrows.
- **In a Mealy machine, the state of the outputs is a function of the state of the machine and the inputs. The status of the outputs is placed on the arrows next to the status of the inputs.**

Automat Mealy'ego

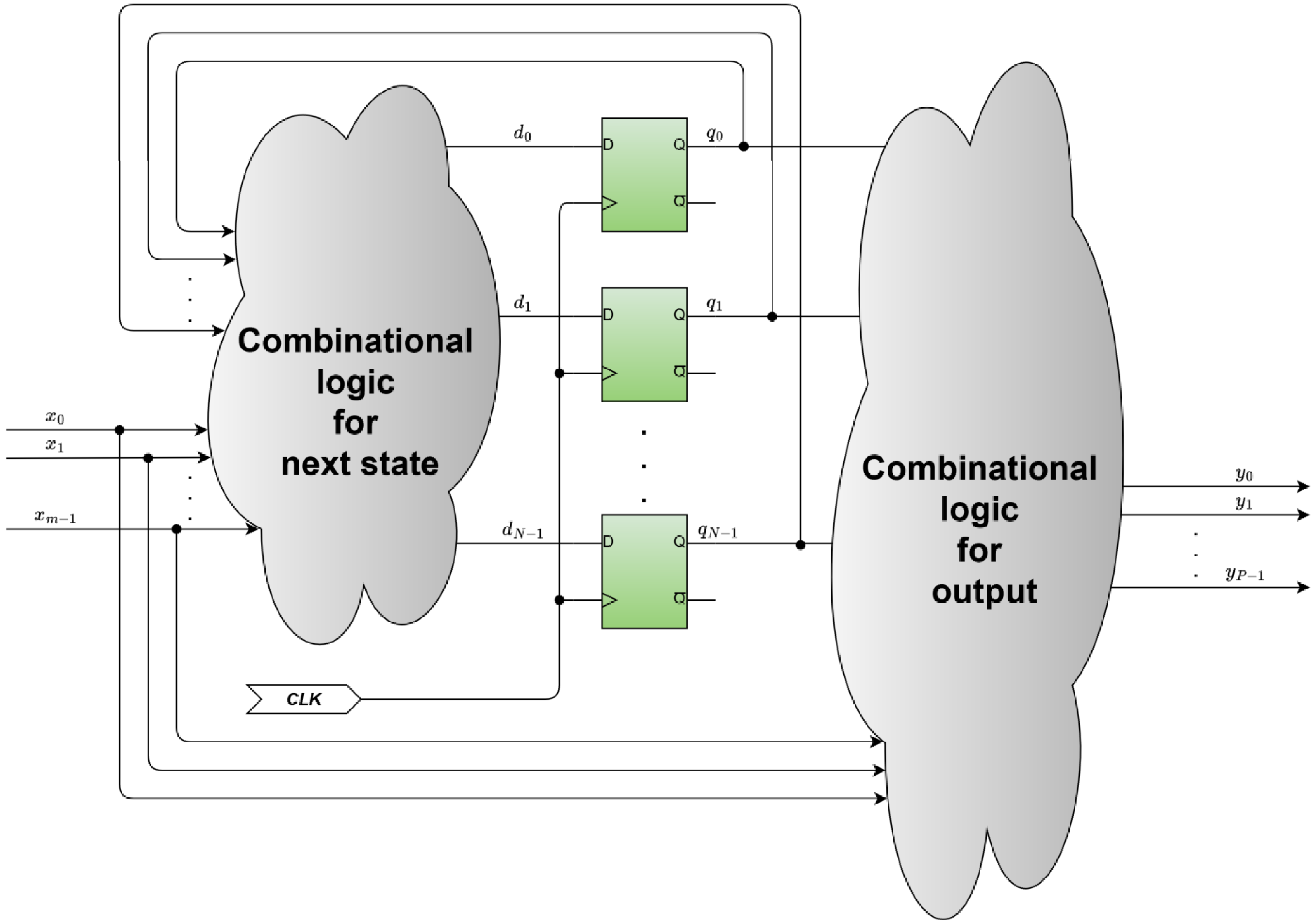
Q – machine state
(q_{N-1}, … q_1, q_0);

X – inputs
(x_{M-1}, … x_1, x_0);

Y – outputs
(y_{P-1}, … y_1, y_0);

Transition logic:
$Q_{n+1} = f(X, Q_n)$;

Output logic:
$Y = f(X, Q_n)$;

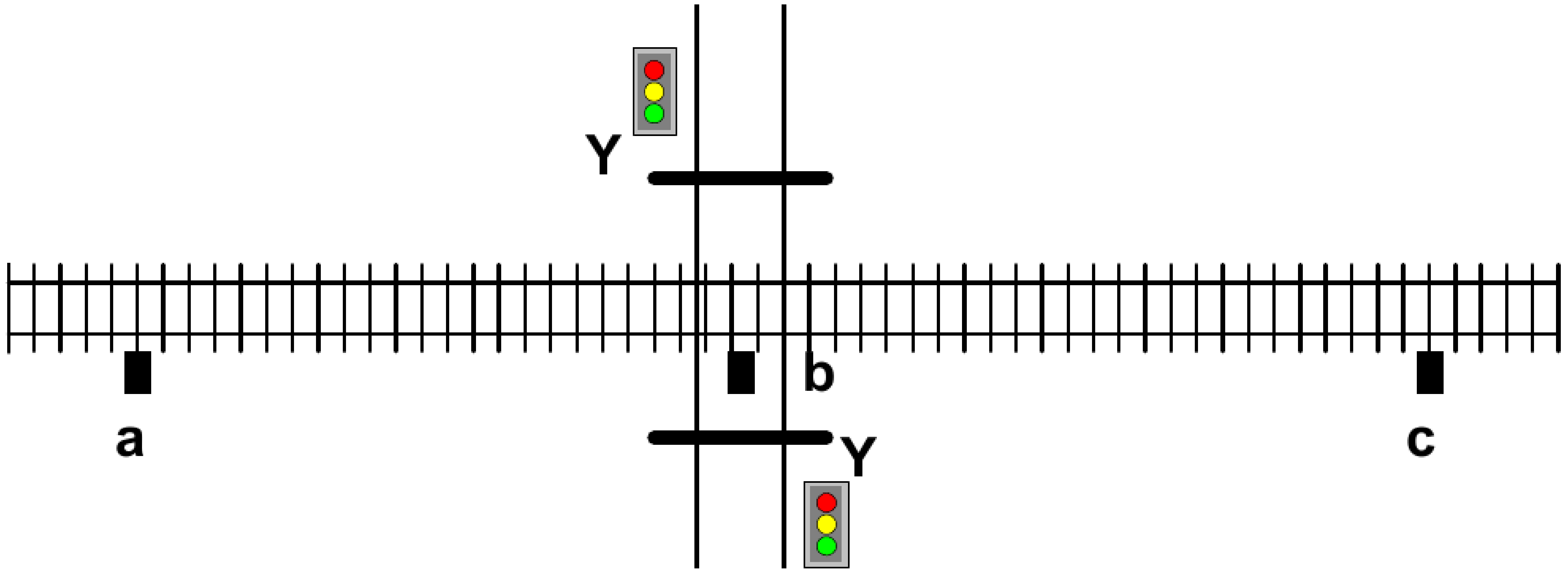


Synthesis of State Machines

1. Drawing a diagram describing the state machine.
2. Based on the diagram of the machine, draw a transition table and output table.
3. Simplify the machine if possible.
4. Encoding states.
5. Selecting a flip-flop.
6. Synthesis of combinational logic of transitions based on the transition table of the machine and the excitation table of the flip-flop.
7. Synthesis of combinational logic of outputs based on the machine's output table.
8. Drawing a complete diagram with gates and flip-flops.

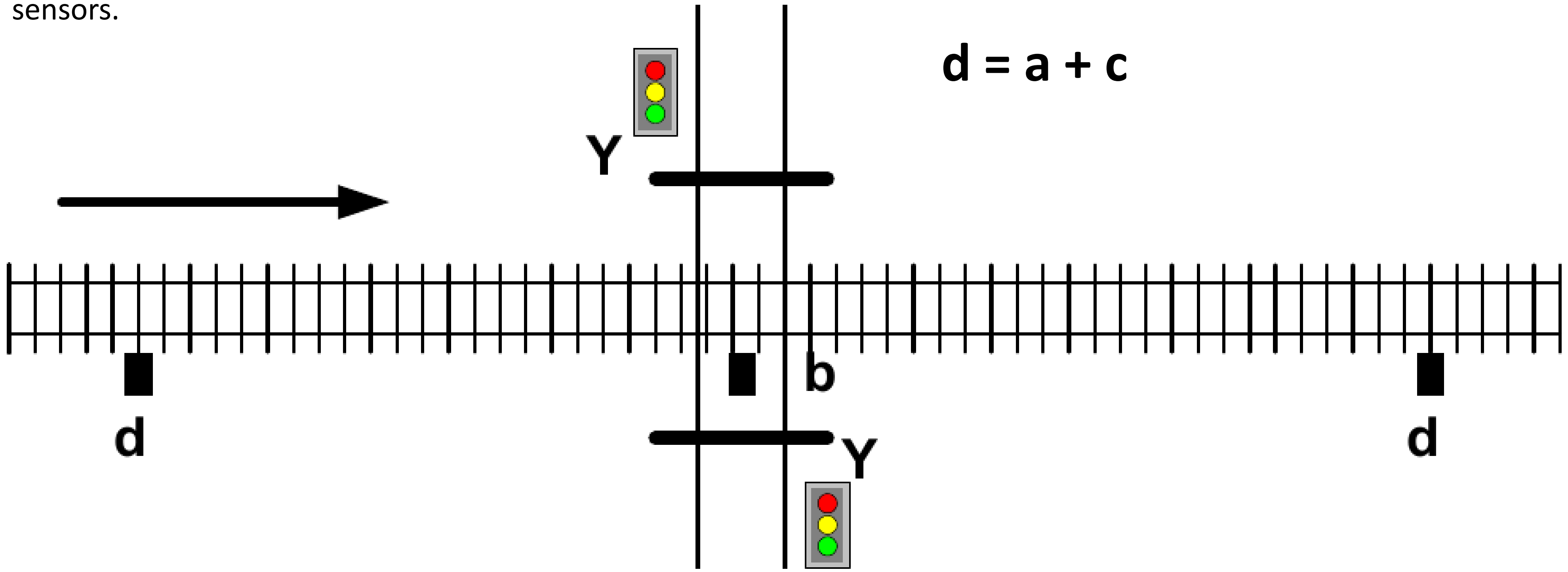
Example of State Machine synthesis - Guarded railway crossing

TASK: Design a control system for raising/lowering railway barriers, based on the signal from three train position sensors.



Example of State Machine synthesis - Guarded railway crossing

TASK: Design a control system for raising/lowering railway barriers, based on the signal from three train position sensors.



$d = a + c$

$\overline{Y}$	Y	Y	Y	$\overline{Y}$	$\overline{Y}$	$\overline{Y}$
$\overline{b}\,\overline{d}$	$\overline{b}d$	$\overline{b}\,\overline{d}$	$b\overline{d}$	$\overline{b}\,\overline{d}$	$\overline{b}d$	$\overline{b}\,\overline{d}$
A	B	C	D	E	F	A

Example of State Machine synthesis - Guarded railway crossing Diagram

$$\begin{tikzcd}
& A \arrow[loop above, "\overline{b}\,\overline{d} / \overline{Y}"] \arrow[dr, "\overline{b}d / Y"] & \\
F \arrow[ur, "\overline{b}\,\overline{d} / \overline{Y}"] \arrow[loop left, "\overline{b}d / \overline{Y}"] & & B \arrow[loop right, "\overline{b}d / Y"] \arrow[d, "\overline{b}\,\overline{d} / Y"] \\
E \arrow[u, "\overline{b}d / \overline{Y}"] \arrow[loop left, "\overline{b}\,\overline{d} / \overline{Y}"] & & C \arrow[loop right, "\overline{b}\,\overline{d} / Y"] \arrow[dl, "b\overline{d} / Y"] \\
& D \arrow[loop below, "b\overline{d} / Y"] \arrow[ul, "\overline{b}\,\overline{d} / \overline{Y}"] &
\end{tikzcd}$$

Example of State Machine synthesis - Guarded railway crossing
Transition and output table

S_n (Stan aktualny)	X (wejścia)		S_{n+1} (Stan następny)
	b	d	
A	0	0	A
	0	1	B
	1	0	-
	1	1	-
B	0	0	C
	0	1	B
	1	0	-
	1	1	-
C	0	0	C
	0	1	-
	1	0	D
	1	1	-
D	0	0	E
	0	1	-
	1	0	D
	1	1	-
E	0	0	E
	0	1	F
	1	0	-
	1	1	-
F	0	0	A
	0	1	F
	1	0	-
	1	1	-

S_n (Stan aktualny)	X (wejścia)		Y (Wyjście)
	b	d	
A	0	0	0
	0	1	1
	1	0	-
	1	1	-
B	0	0	1
	0	1	1
	1	0	-
	1	1	-
C	0	0	1
	0	1	-
	1	0	1
	1	1	-
D	0	0	0
	0	1	-
	1	0	1
	1	1	-
E	0	0	0
	0	1	0
	1	0	-
	1	1	-
F	0	0	0
	0	1	0
	1	0	-
	1	1	-

Example of State Machine synthesis - Guarded railway crossing
Rreduction of the number of states

B + C => BC

D + E => DE

$$\begin{tikzcd}
 & A \arrow[loop above, "\bar{b}\bar{d} / \bar{Y}"] \arrow[dr, bend left, "\bar{b}d / Y"] & \\
F \arrow[loop left, "\bar{b}d / \bar{Y}"] \arrow[ur, bend left, "\bar{b}\bar{d} / \bar{Y}"] & & BC \arrow[loop right, "\bar{b}\bar{d} / Y \\ \bar{b}d / Y"] \arrow[dl, bend left, "b\bar{d} / Y"] \\
 & DE \arrow[loop below, "\bar{b}\bar{d} / \bar{Y} \\ b\bar{d} / Y"] \arrow[ul, bend left, "\bar{b}d / \bar{Y}"] &
\end{tikzcd}$$

Example of State Machine synthesis - Guarded railway crossing
Rreduction of the number of states

B + C => BC　　　**D + E => DE**

S_n (Stan aktualny)	X (wejścia)		S_{n+1} (Stan następny)
	b	d	
A	0	0	A
	0	1	BC
	1	0	-
	1	1	-
BC	0	0	BC
	0	1	BC
	1	0	DE
	1	1	-
DE	0	0	DE
	0	1	F
	1	0	DE
	1	1	-
F	0	0	A
	0	1	F
	1	0	-
	1	1	-

S_n (Stan aktualny)	X (wejścia)		Y (Wyjście)
	b	d	
A	0	0	0
	0	1	1
	1	0	-
	1	1	-
BC	0	0	1
	0	1	1
	1	0	1
	1	1	-
DE	0	0	0
	0	1	0
	1	0	1
	1	1	-
F	0	0	0
	0	1	0
	1	0	-
	1	1	-

Example of State Machine synthesis - Guarded railway crossing
State encoding

S_n (Stan aktualny)	Q	
	q_1	q_0
A	0	0
BC	0	1
DE	1	1
F	1	0

Example of State Machine synthesis - Guarded railway crossing
Transition and output table

S_n			X		S_{n+1}		
	q_1	q_0	b	d	q_1'	q_0'	
A	0	0	0	0	0	0	A
	0	0	0	1	0	1	BC
	0	0	1	0	-	-	-
	0	0	1	1	-	-	-
BC	0	1	0	0	0	1	BC
	0	1	0	1	0	1	BC
	0	1	1	0	1	1	DE
	0	1	1	1	-	-	-
DE	1	1	0	0	1	1	DE
	1	1	0	1	1	0	F
	1	1	1	0	1	1	DE
	1	1	1	1	-	-	-
F	1	0	0	0	0	0	A
	1	0	0	1	1	0	F
	1	0	1	0	-	-	-
	1	0	1	1	-	-	-

S_n			X		Y
	q_1	q_0	b	d	y_0
A	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	-
	0	0	1	1	-
BC	0	1	0	0	0
	0	1	0	1	0
	0	1	1	0	1
	0	1	1	1	-
DE	1	1	0	0	1
	1	1	0	1	1
	1	1	1	0	1
	1	1	1	1	-
F	1	0	0	0	0
	1	0	0	1	1
	1	0	1	0	-
	1	0	1	1	-

Example of State Machine synthesis - Guarded railway crossing
Transition logic

b d / q_1q_0	00	01	11	10
00	0	0	-	-
01	0	0	-	1
11	1	1	-	1
10	0	1	-	-

b d / q_1q_0	00	01	11	10
00	0	1	-	-
01	1	1	-	1
11	1	0	-	1
10	0	0	-	-

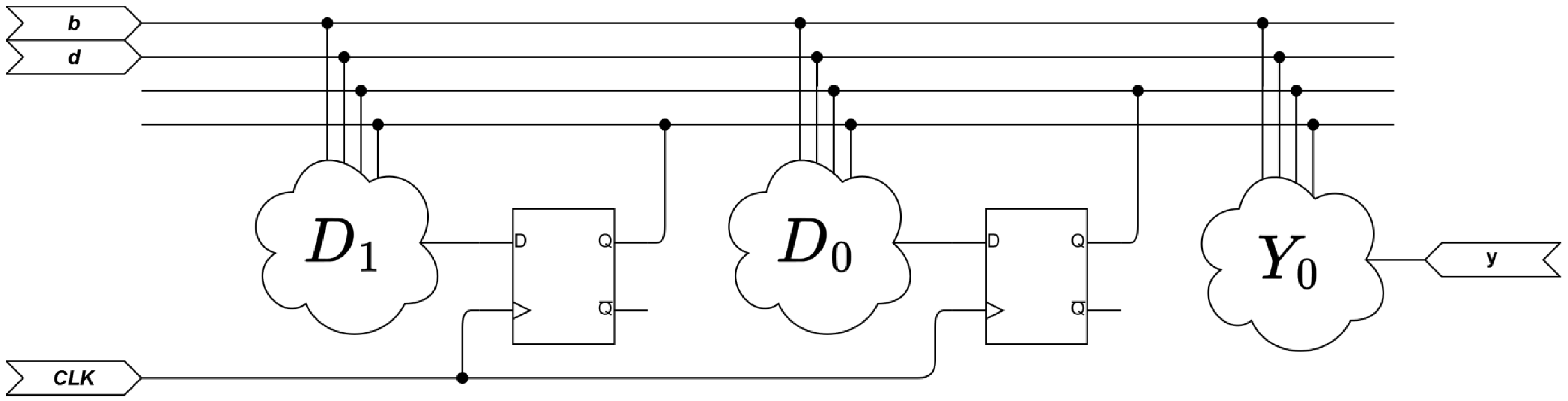
$$D_1 = b + q_1 \cdot q_0 + q_1 \cdot d \qquad D_0 = b + \overline{q_1} \cdot d + \overline{d} \cdot q_0$$

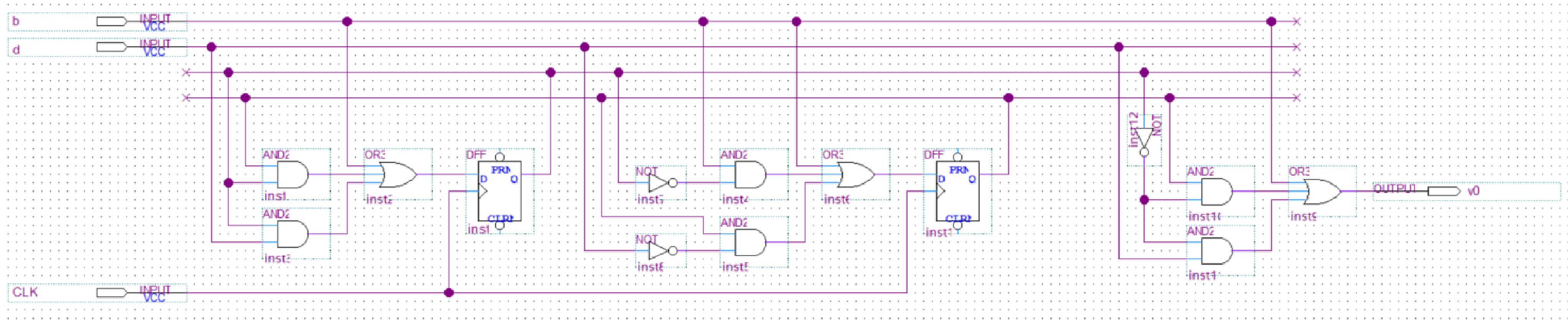
Example of State Machine synthesis - Guarded railway crossing
Output logic

b d / q_1q_0	00	01	11	10
00	0	1	-	-
01	1	1	-	1
11	0	0	-	1
10	0	0	-	-

$$Y_0 = b + \overline{q_1} \cdot q_0 + \overline{q_1} \cdot d$$

Example of State Machine synthesis - Guarded railway crossing Full diagram





REFERENCE

[1] dr hab. inż. Maciej Petko: Wykład – *Układy Sekwencyjne* z przedmiotu *Mechatronicze Systemy Wykonawcze, Sensoryczne i Sterujące*; KRiM AGH;

[2] dr inż. Konrad Gac: Wykład – *Układy Sekwencyjne* z przedmiotu *Mechatronicze Systemy Wykonawcze, Sensoryczne i Sterujące*; KRiM AGH;

[3] Barry Wilkinson: *Układy cyfrowe*; WKŁ

[4] Halina Kamionka-Mikuła, Henryk Małysiak, Bolesław Pochopień: Układy cyfrowe - teoria i przykłady; WPKJS;

[5] Wojciech Głocki: Układy cyfrowe; WSiP;

[6] Andrzej Skorupski: Podstawy techniki cyfrowej; WKŁ;

[7] prof. dr hab. in˙z. Joanna Józefowska: *Kodowanie informacji - Reprezentacja liczb*; Poznań;
rok akademicki 2010/2011;

[8] dr hab. inż. Mikołaj Morzy: *Wykład - kodowanie liczb*;

[9] Politechnika Łódzka: *Kodowanie liczb całkowitych w systemach komputerowych*;
http://neo.dmcs.p.lodz.pl/ak/kodowanie_liczb_calkowitych.pdf