ITMO

Digital circuits design basics.

Encoders, Decoders, Multiplexers, Flip-flops (SR, D, JK), Registers, Counters

Nikolay Nikolaev (nanikolaev@itmo.ru)
Nikolai Poliakov (polyakov_n_a@itmo.ru)
Arina Arbuzina (arbyzina99@gmail.com)

Contents

- Combinational logic elements:
  -Multiplexers;
  -Encoders;
  -Decoders.

- Sequential logic elements:
  -Flip-flops (SR, D, JK);
  -Registers;
  -Counters.

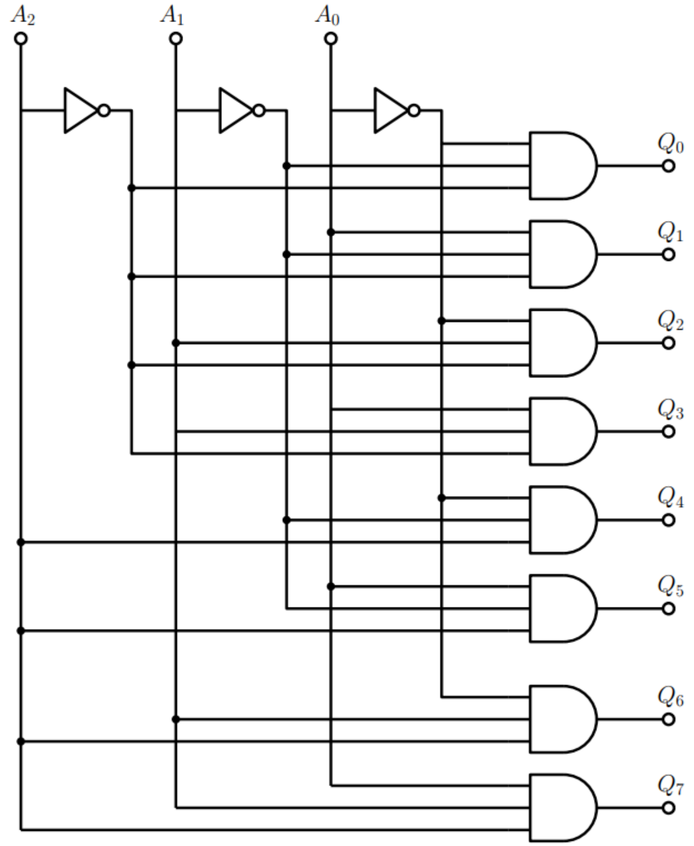
Combinational logic elements

In this section we consider some common digital system components, such as:

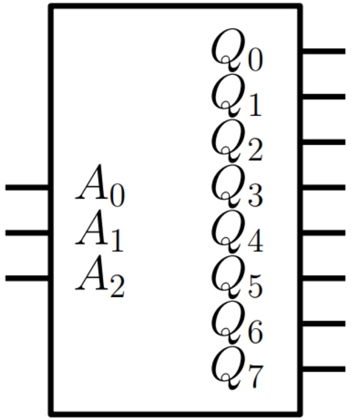
- decoders;
- encoders;
- multiplexers;
- demultiplexers

Decoder

A decoder is a combinational network that decodes (converts) the *n*-bit binary-coded input to *m* outputs ($m \leq 2^n$). The block diagram of a 3-bit to 8-element decoder is shown in Figure, wherein the three inputs are decoded into eight outputs, one for each combination of the input variables.



Decoder. True table.



A_2	A_1	A_0	Q_7	Q_6	Q_5	Q_4	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

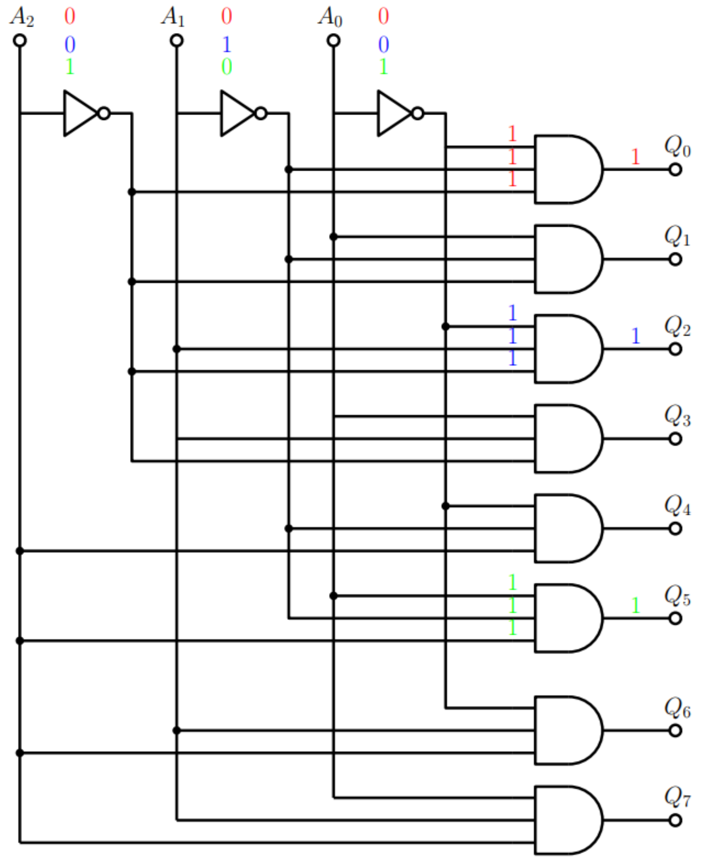
Decoder. Examples

In the Figure you can see transformations of three binary combinations into respective output signals.

000→ Q_0

010→ Q_2

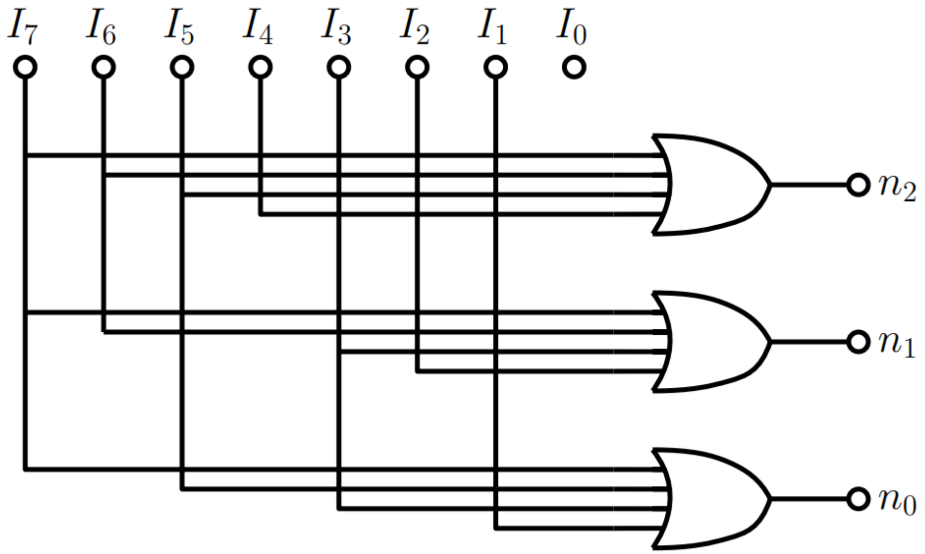
101→ Q_5



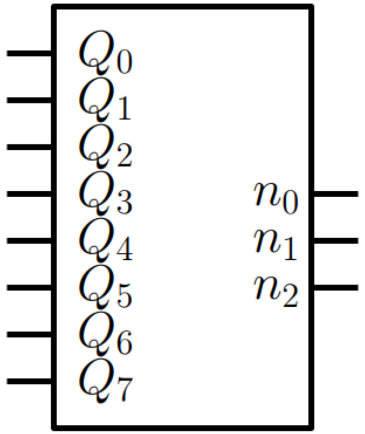
Encoder

Encoding is the process of forming an encoded representation of a set of inputs, and it is the converse of the decoding operation. An encoder is a combinational network that generates an *n*-bit binary code that uniquely identifies the one out of *m* activated inputs ($0 \leq m \leq 2^n - 1$).

Figure shows the block diagram of an 8-element to 3-bit encoder.



Encoder. True table.

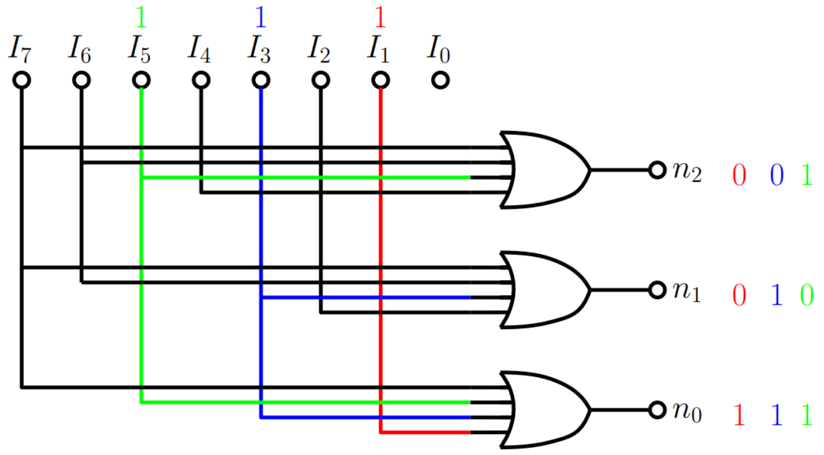


I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	n_2	n_1	n_0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Encoder. Examples

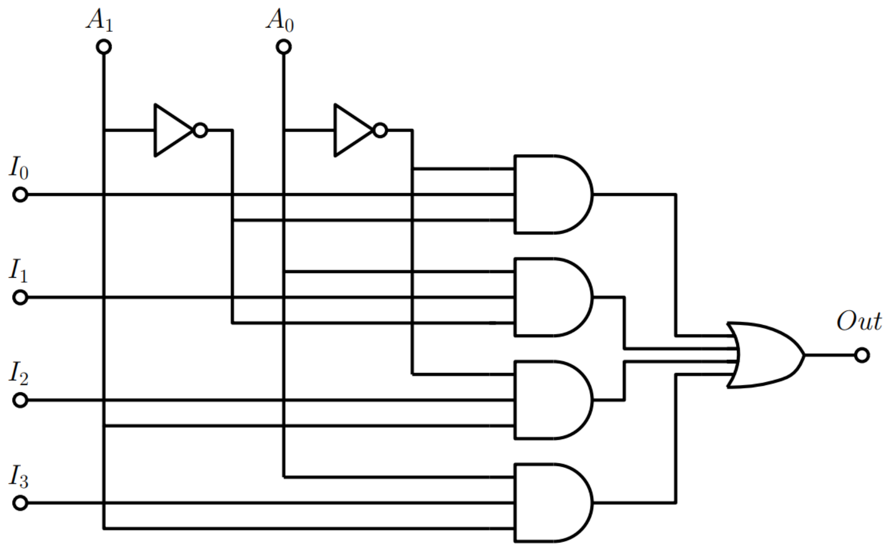
In the Figure you can see transformations of input signals to binary combinations.

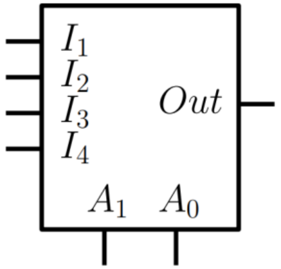
$I_1 \rightarrow 001$
$I_3 \rightarrow 010$
$I_5 \rightarrow 101$



Multiplexer

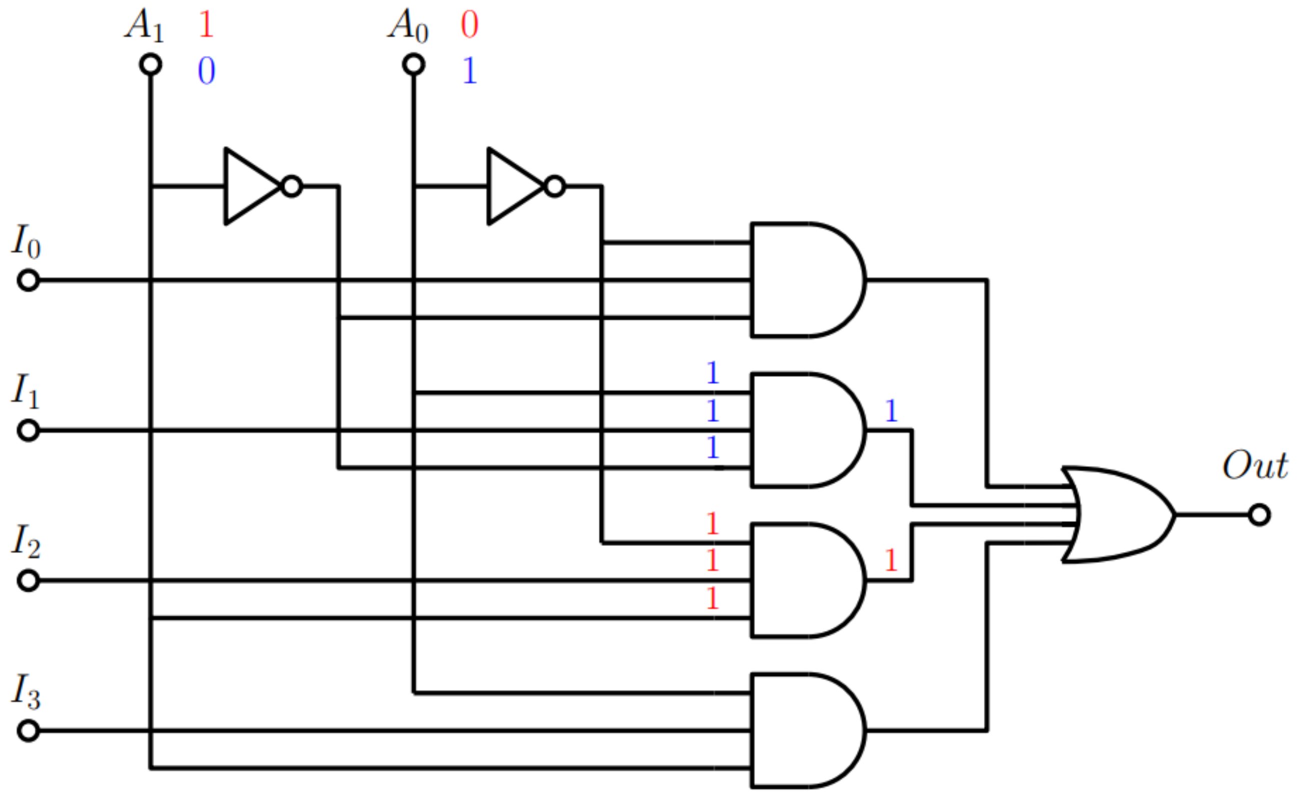
A *multiplexer* is a data selector.
A multiplexer is a combinational network that selects one of several possible input signals and directs that signal to a single output terminal. The selection of a particular input is controlled by a set of selection variables. A multiplexer with n selection variables can usually select one out of 2^n input signals.





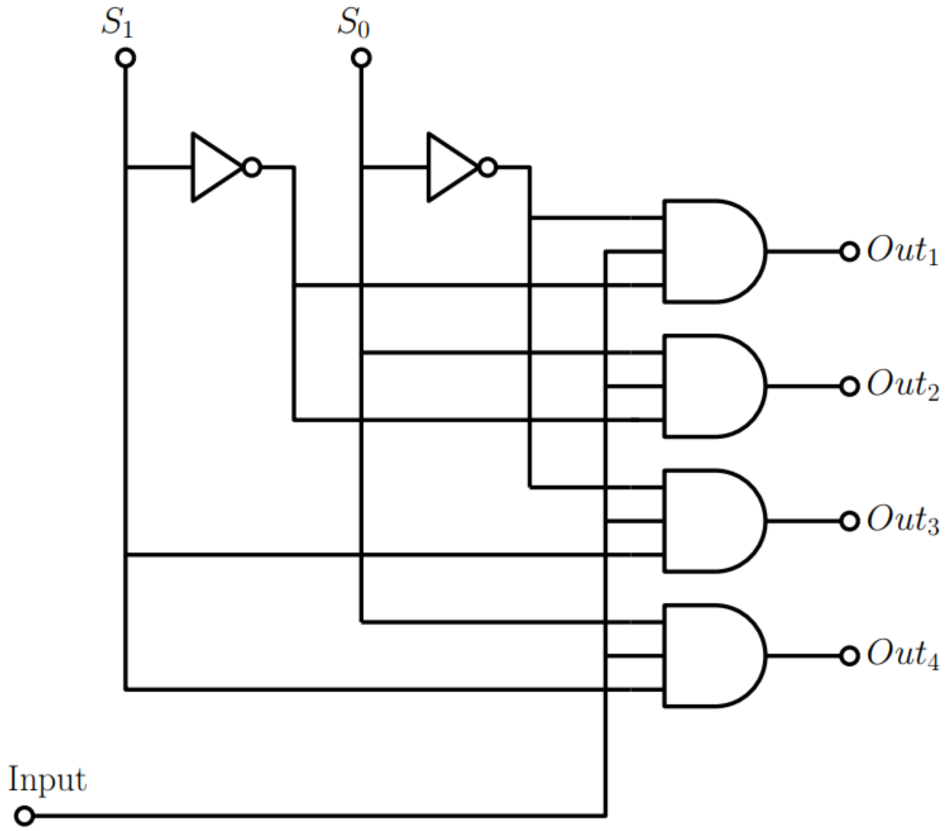
A_1	A_0	Out
0	0	I_1
0	1	I_2
1	0	I_3
1	1	I_4

Multiplexer. Example



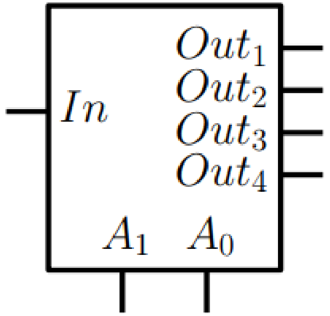
Demultiplexer

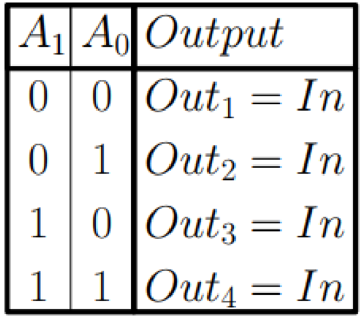
A demultiplexer (or data distributor) is the opposite of a multiplexer. It takes a single data input and routes it to one of several possible outputs.
To select the output (Out_1, Out_2, Out_3, or Out_4) to which you want to send the input signal, you apply logic levels to the data select inputs (S_0, S_1)



Demultiplexer. True Table.

Demultiplexer works according to the True Table.





A_1	A_0	$Output$
0	0	$Out_1 = In$
0	1	$Out_2 = In$
1	0	$Out_3 = In$
1	1	$Out_4 = In$

Sequential logic elements

In this section we consider some common Sequential logic components, such as:

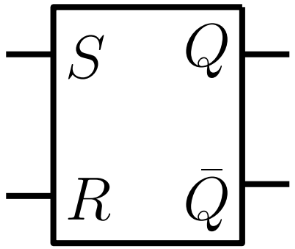
-Flip-flops (SR, D, JK);

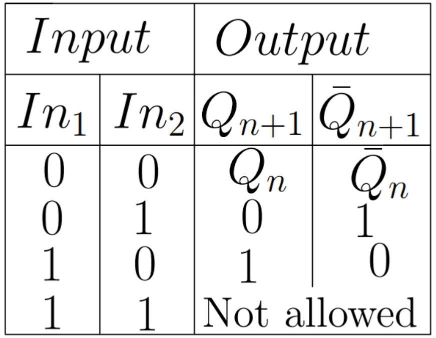
-Registers;

-Counters.

SR FLIP-FLOP (SRFF)

The symbol for the SRFF and True table are shown in Figure.

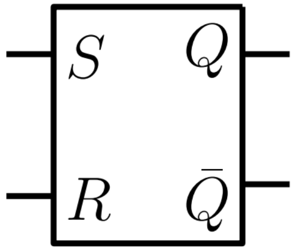




$Input$		$Output$	
In_1	In_2	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	Q_n	$\bar{Q}_n$
0	1	0	1
1	0	1	0
1	1	Not allowed	

SR FLIP-FLOP (SRFF)

The symbol for the SRFF and True table are shown in Figure.

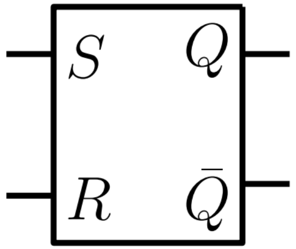


$Input$		$Output$	
In_1	In_2	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	Q_n	$\bar{Q}_n$
0	1	0	1
1	0	1	0
1	1	Not allowed	

Can we construct SRFF by using combinations of logic blocks?

SR FLIP-FLOP (SRFF)

The symbol for the SRFF and True table are shown in Figure.



$Input$		$Output$	
In_1	In_2	Q_{n+1}	$\bar{Q}_{n+1}$
0	0	Q_n	$\bar{Q}_n$
0	1	0	1
1	0	1	0
1	1	Not allowed	

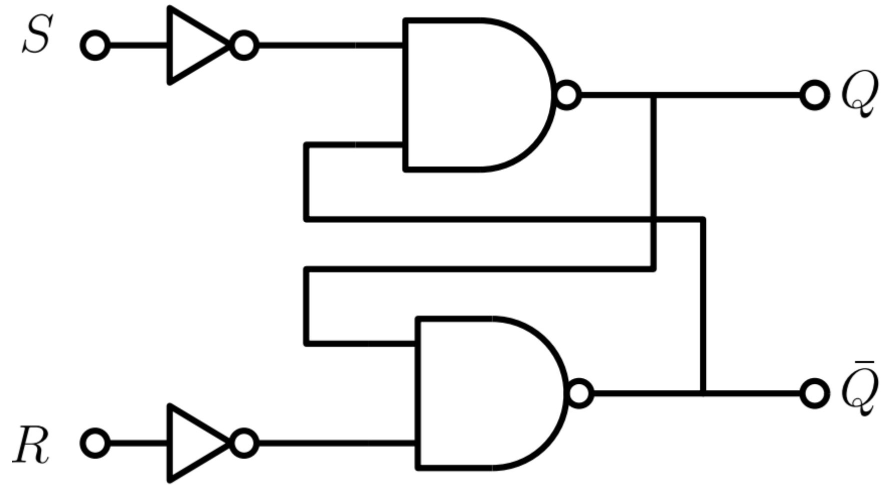
Can we construct SRFF by using combinations of logic blocks?

Yes, we can!

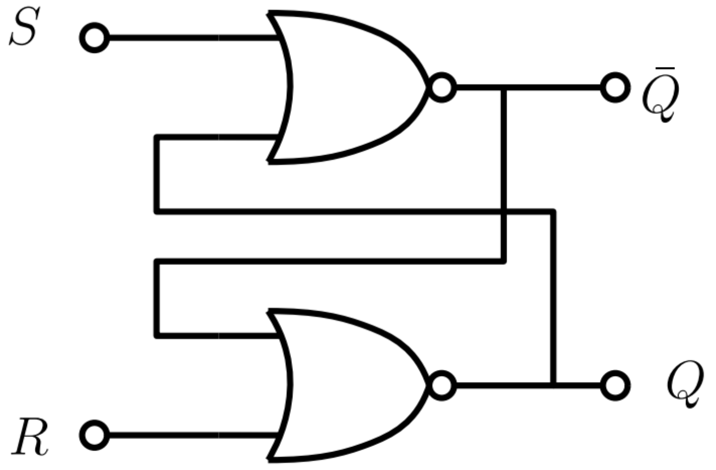
SR FLIP-FLOP (SRFF)

We can make SRFF from two types of basic elements – NAND and NOR

Cross-NAND SR flip-flop

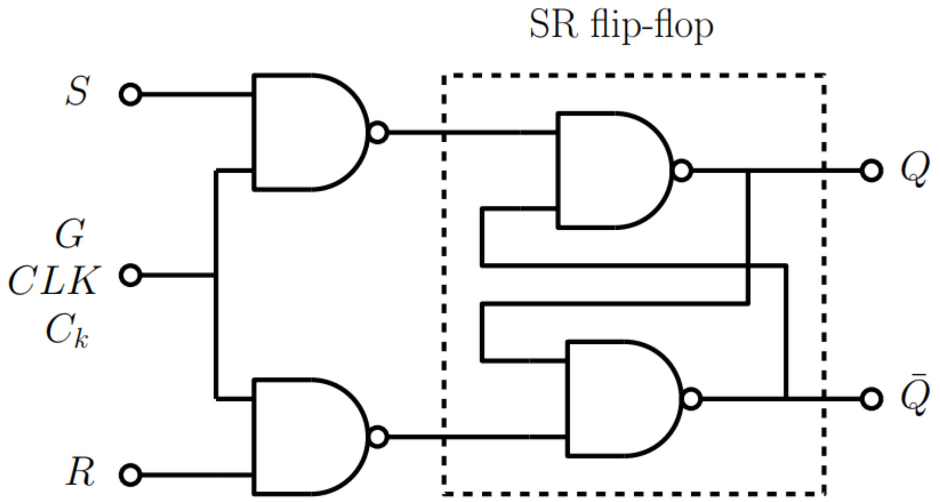


Cross-NOR SR flip-flop



Level-Triggered SR Flip-Flop

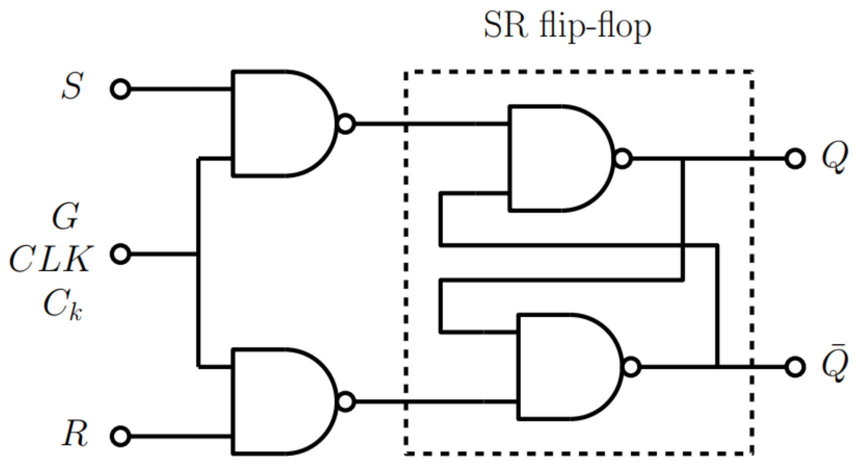
Level-Triggered SR Flip-Flop – this is a synchronous flip-flop—meaning making the *S* and *R* inputs either enabled or disabled by a control pulse.

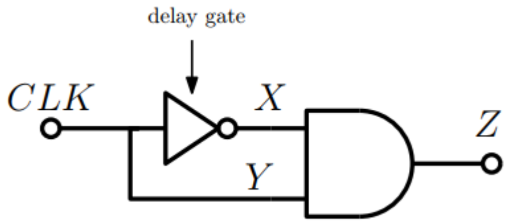


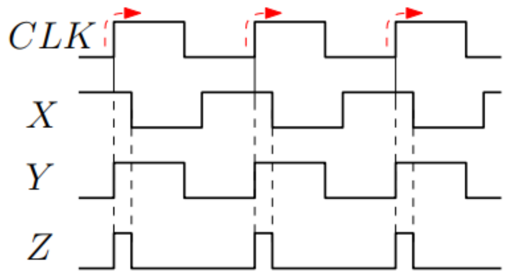
C_k	S	R	Q	$\bar{Q}$	Mode
0	0	0	Q	$\bar{Q}$	Hold
0	0	1	Q	$\bar{Q}$	Hold SR-inputs
0	1	0	Q	$\bar{Q}$	Hold disabled
0	1	1	Q	$\bar{Q}$	Hold
1	0	0	0	0	Hold
1	0	1	0	1	RESET SR-inputs
1	1	0	1	0	SET enabled
1	1	1	Q	$\bar{Q}$	Inderterminate

Edge-Triggered SR Flip-Flops

An edge-triggered flipflop samples the inputs only during either a positive or negative clock edge (↑ = positive edge, ↓ = negative edge).

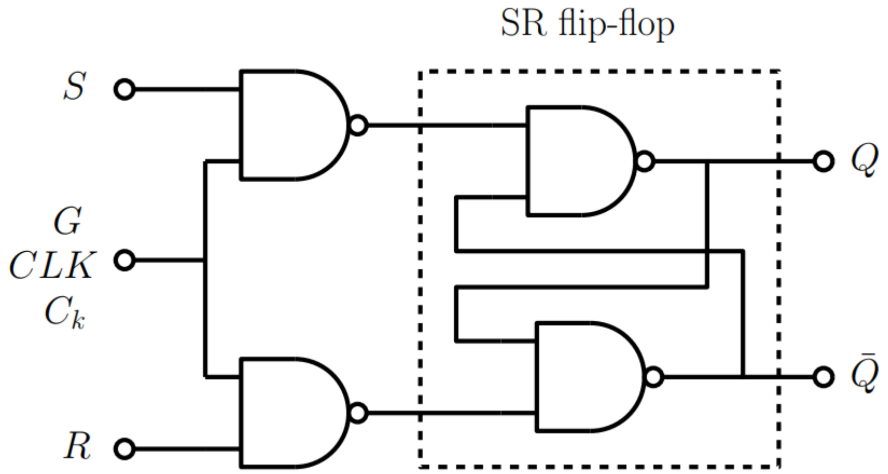


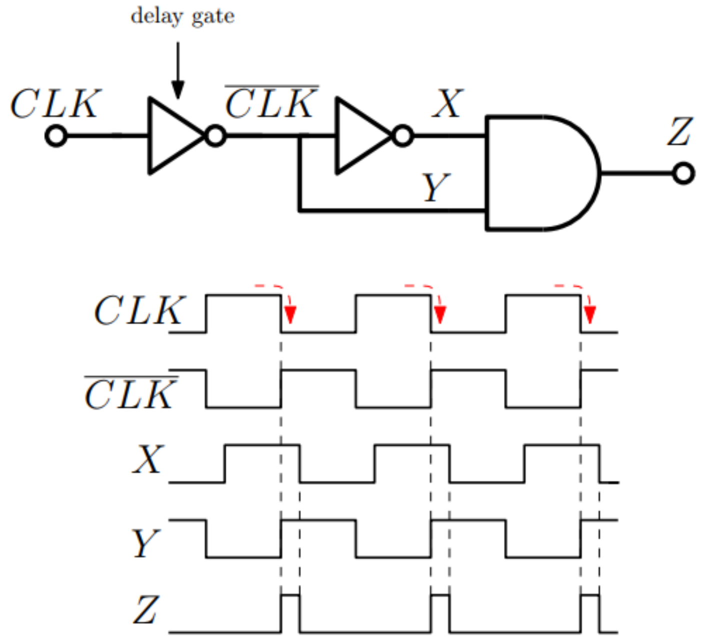




Positive edge-triggered

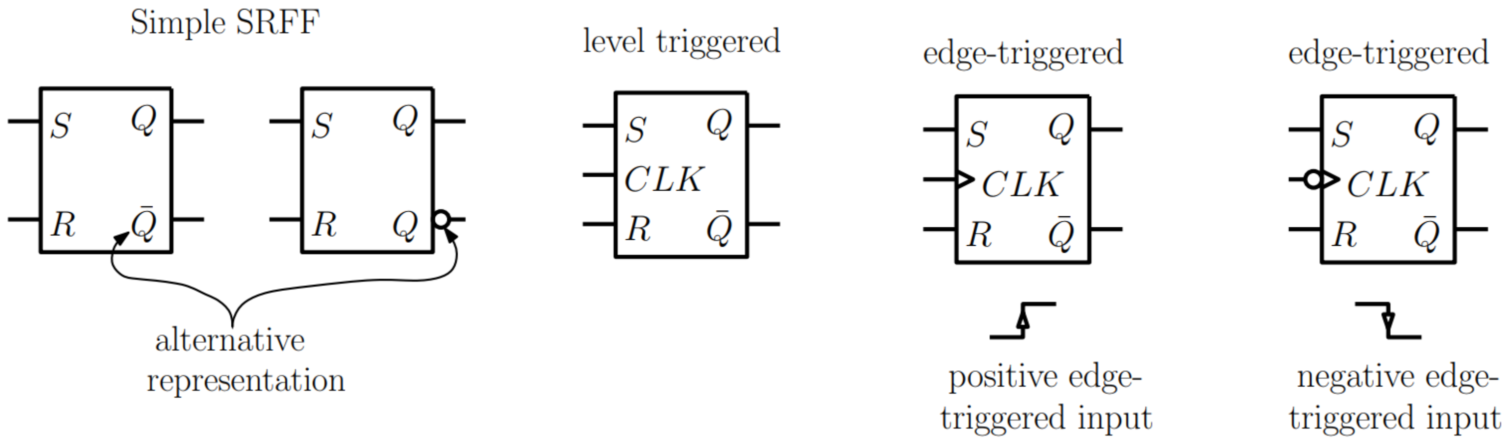
Edge-Triggered SR Flip-Flops





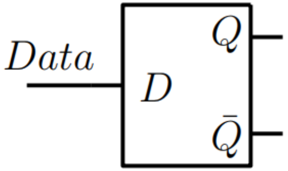
Negative edge-triggered

Symbolic representation SRFF



D Flip-Flops

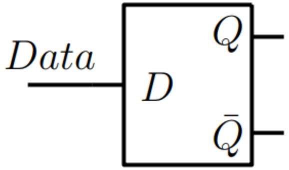
D FLIP-FLOP (LATCH OR DELAY ELEMENT).
A D-type flip-flop is a single input device. It is basically an SR flipflop, where *S* is replaced with *D* and *R* is replaced *D* (inverted *D*).

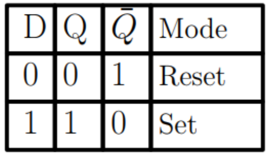


D	Q	$\bar{Q}$	Mode
0	0	1	Reset
1	1	0	Set

D Flip-Flops

D FLIP-FLOP (LATCH OR DELAY ELEMENT).
A D-type flip-flop is a single input device. It is basically an SR flipflop, where *S* is replaced with *D* and *R* is replaced *D* (inverted *D*).



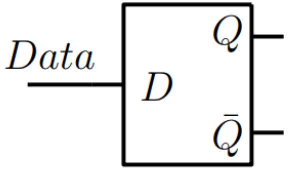


D	Q	$\bar{Q}$	Mode
0	0	1	Reset
1	1	0	Set

How can we make D-type flip-flop?

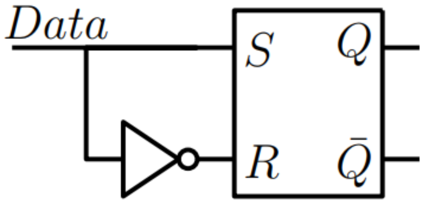
D Flip-Flops

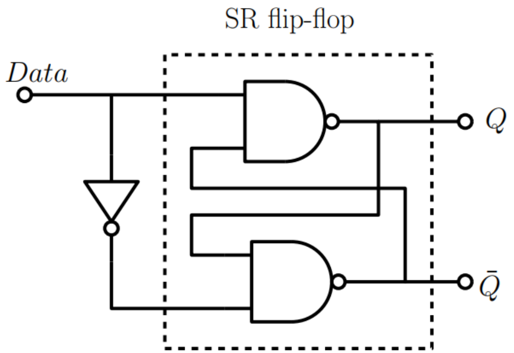
D FLIP-FLOP (LATCH OR DELAY ELEMENT).
A D-type flip-flop is a single input device. It is basically an SR flipflop, where *S* is replaced with *D* and *R* is replaced *D* (inverted *D*).



D	Q	$\bar{Q}$	Mode
0	0	1	Reset
1	1	0	Set

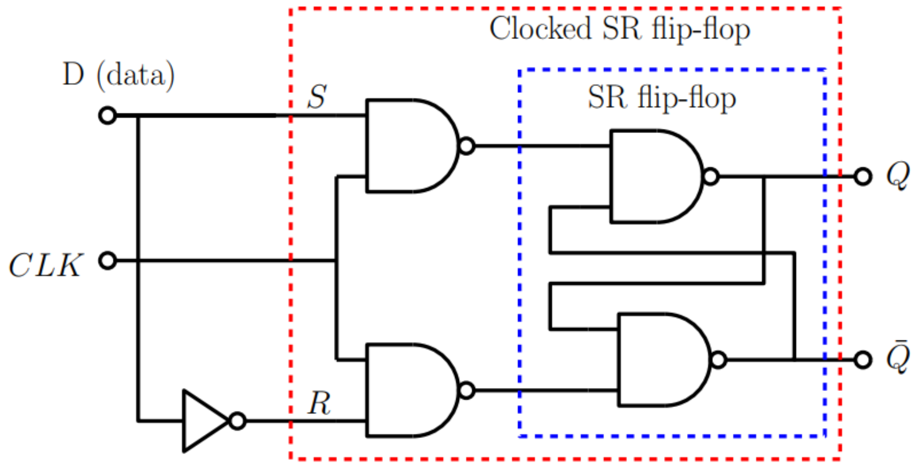
How can we make D-type flip-flop?
Of course, we can make D flip-flop from logic elements and also from SRFF.

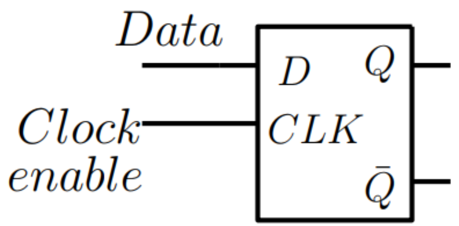




D Flip-Flops

Also we can make a clocked D-type level-triggered flip-flop.

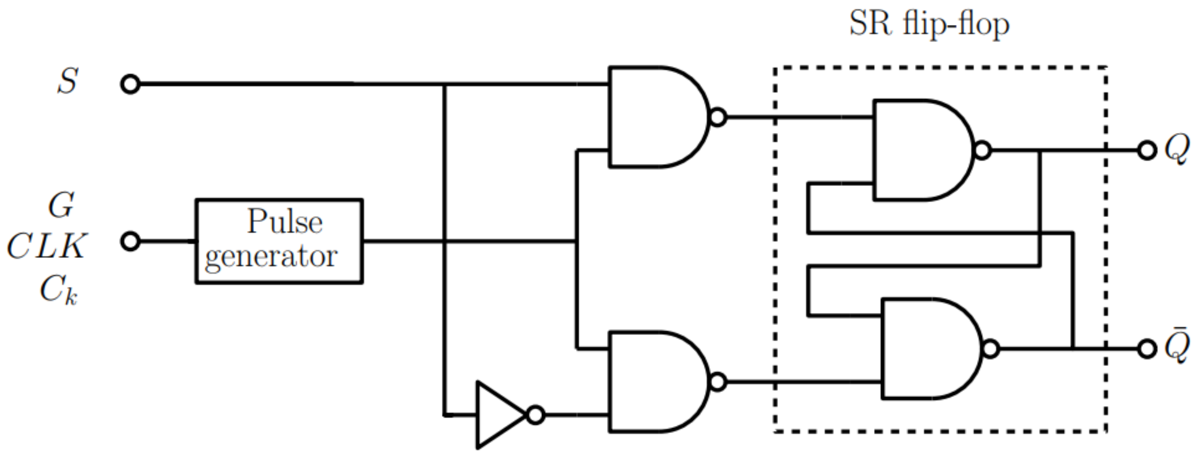


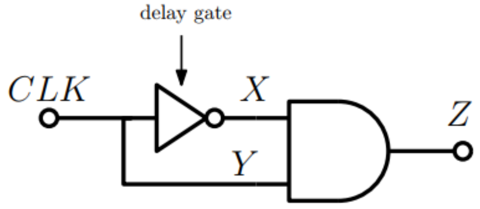


Input		*Output*		
CLK	D	Q	$\bar{Q}$	*Mode*
0	X	Q	$\bar{Q}$	*Hold*
1	0	0	1	*Reset*
1	1	1	0	*Set*

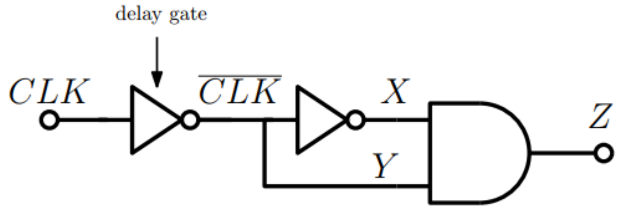
D Flip-Flops

Also we can make a clocked, edge-triggered D-type flip-flop .



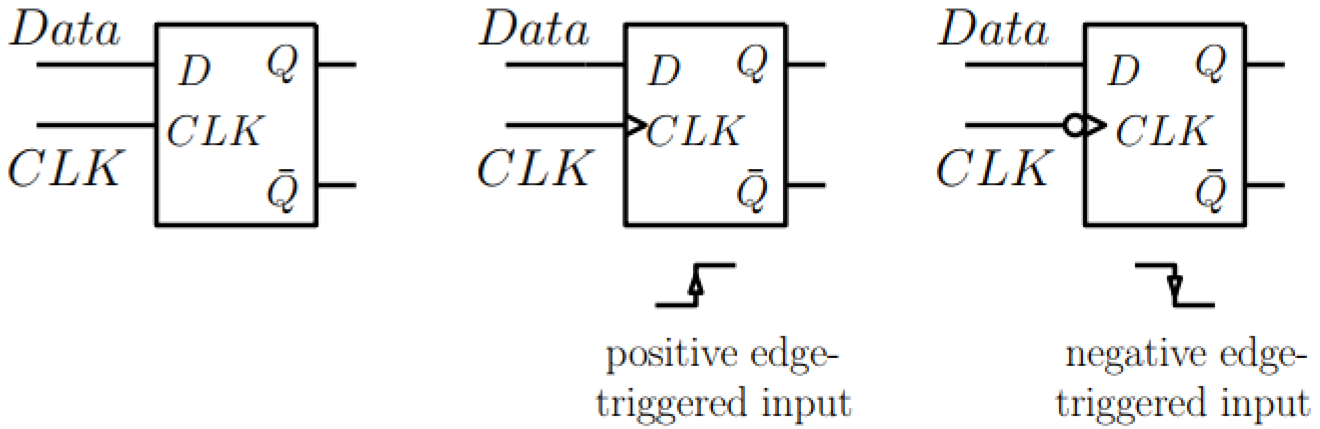


Positive edge-triggered



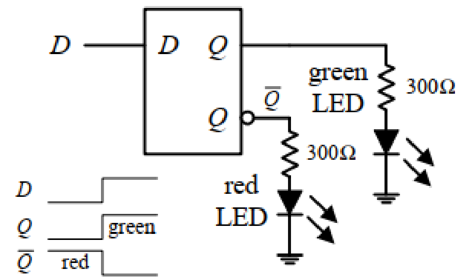
Negative edge-triggered

Symbolic representation D flip-flop

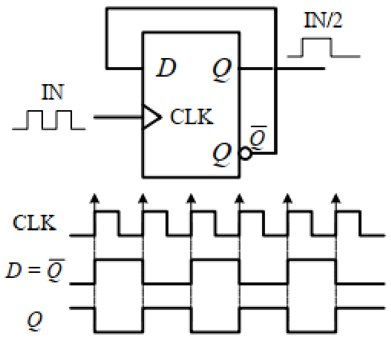


D-Type Flip-Flop Applications

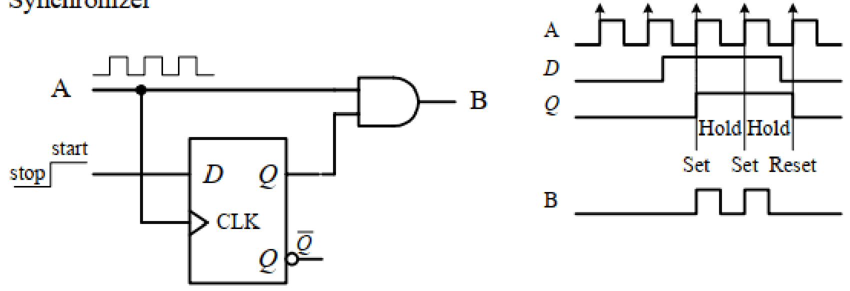
Stop-Go indicator



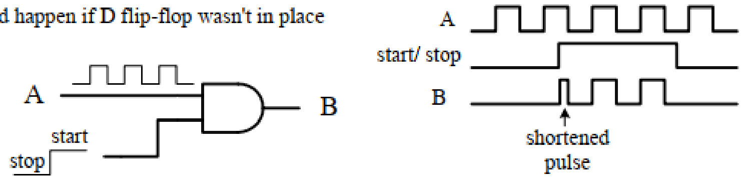
Divide-by-two counter



Synchronizer



What would happen if D flip-flop wasn't in place



JK Flip-Flop

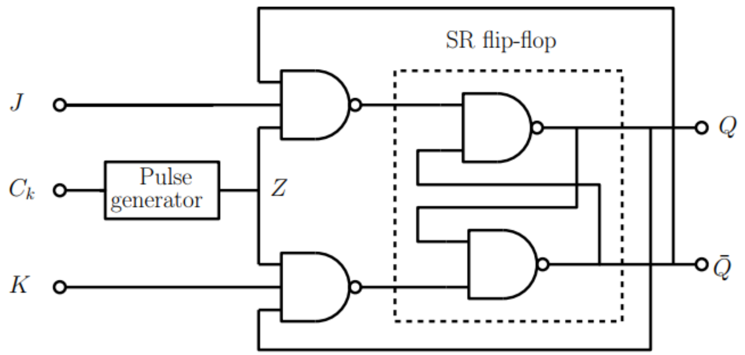
A JK flip-flop resembles an SR flip-flop, where J acts like S and K acts like R. However, unlike the SR flip-flop, which has an indeterminate mode when $S = 1$, $R = 1$, the JK flip-flop has a *toggle* mode when $J = 1$, $K = 1$. *Toggle* means that the Q and Q outputs switch to their opposite states at each active clock edge.

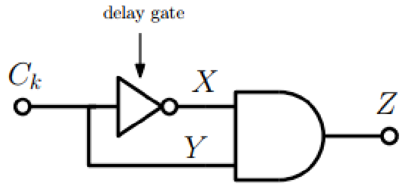
As D flip-flop we can make JK flip-flop based on SRFF.

To make a JK flip-flop, modify the SR flip-flop's internal logic circuit to include two cross-coupled feedback lines between the output and input. This modification, however, means that the JK flip-flop cannot be level-triggered; it can only be edge-triggered or pulse-triggered.

JK Flip-Flop

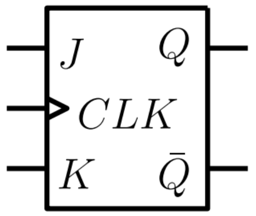
Positive edge-triggered





C_k	J	K	Q	$\bar{Q}$	Mode
0	X	X	Q	$\bar{Q}$	Hold
1	X	X	Q	$\bar{Q}$	Hold
↓	X	X	Q	$\bar{Q}$	Hold
↑	0	0	Q	$\bar{Q}$	Hold
↑	0	1	0	1	Reset
↑	1	0	1	0	Set
↑	1	1	$\bar{Q}$	Q	Toggle

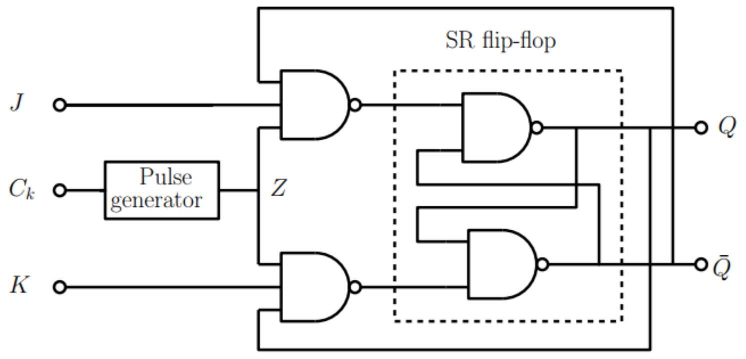
edge-triggered

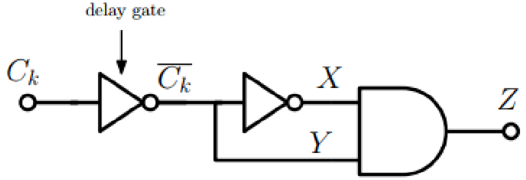


positive edge-triggered input

JK Flip-Flop

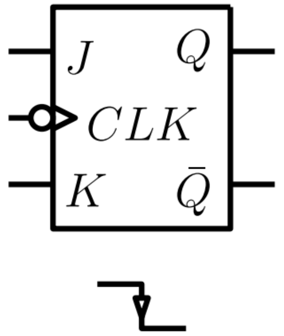
Negative edge-triggered





C_k	J	K	Q	$\bar{Q}$	Mode
0	X	X	Q	$\bar{Q}$	Hold
1	X	X	Q	$\bar{Q}$	Hold
↑	X	X	Q	$\bar{Q}$	Hold
↓	0	0	Q	$\bar{Q}$	Hold
↓	0	1	0	1	Reset
↓	1	0	1	0	Set
↓	1	1	$\bar{Q}$	Q	Toggle

edge-triggered

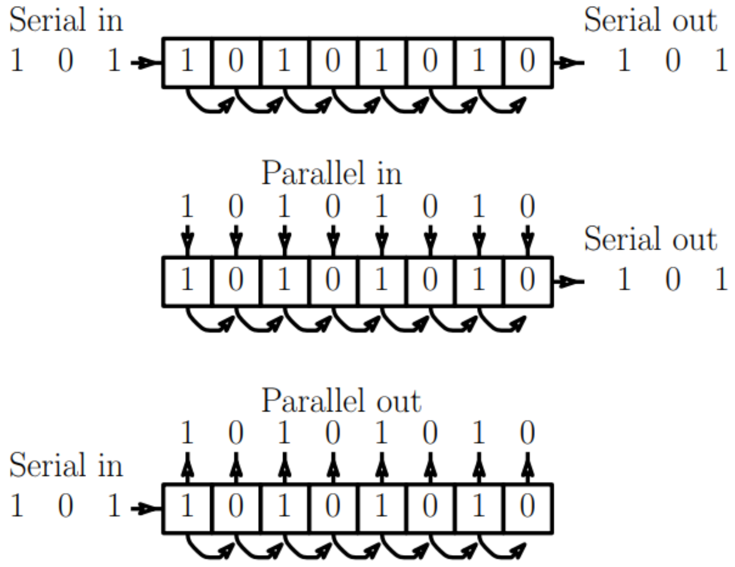


negative edge-triggered input

Shift Registers

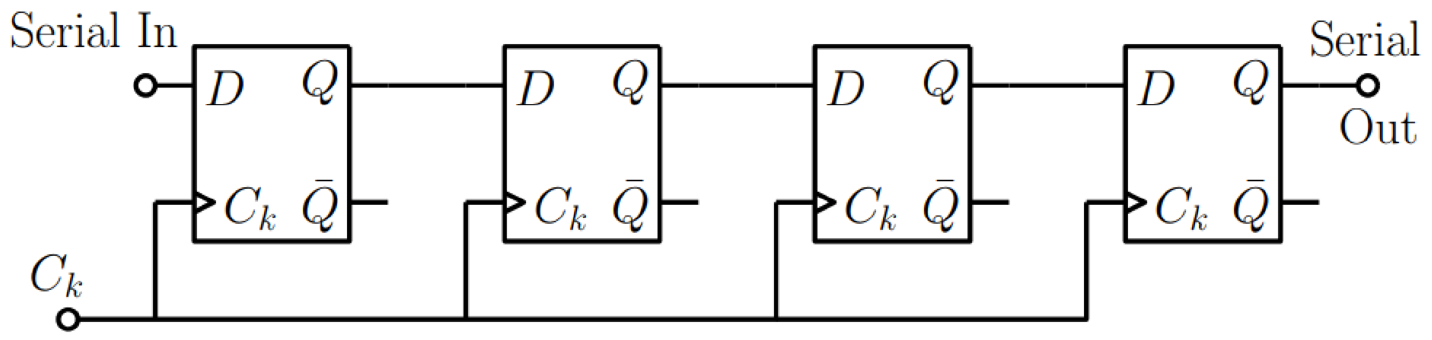
Data words traveling through a digital system frequently must be temporarily
held, copied, and bit-shifted to the left or to the right. A device that can be used for
such applications is the *shift register.*
There are three types of shift register arrangements:

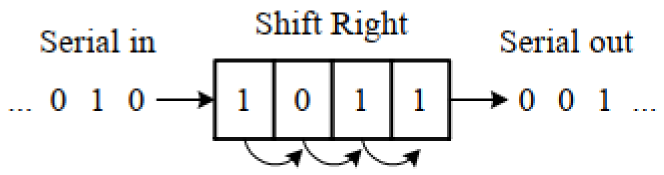
- serial-in/serial-out,
- parallel-in/parallel-out
- parallel-in/serial-out and serial-in/parallel out.



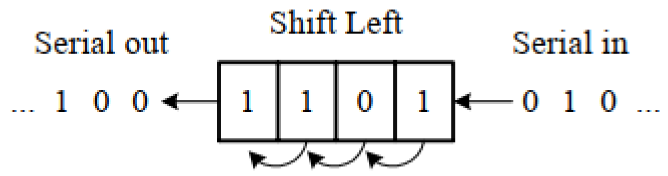
Shift Registers

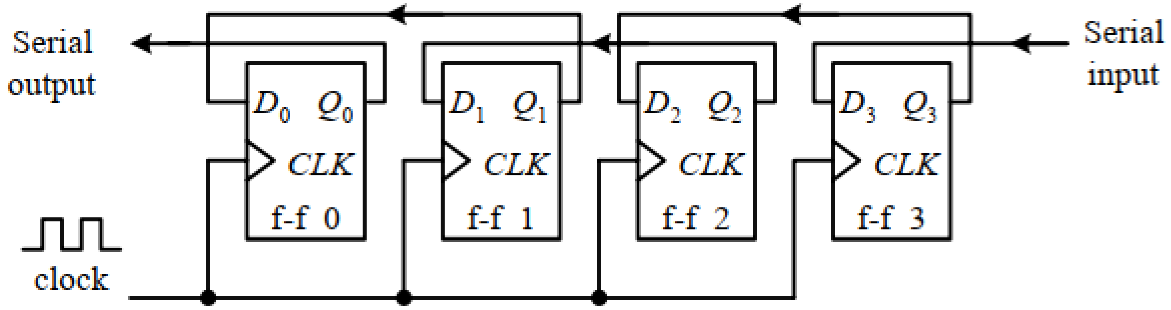
Simple example - ***Serial-In/Serial-Out Shifter Registers***





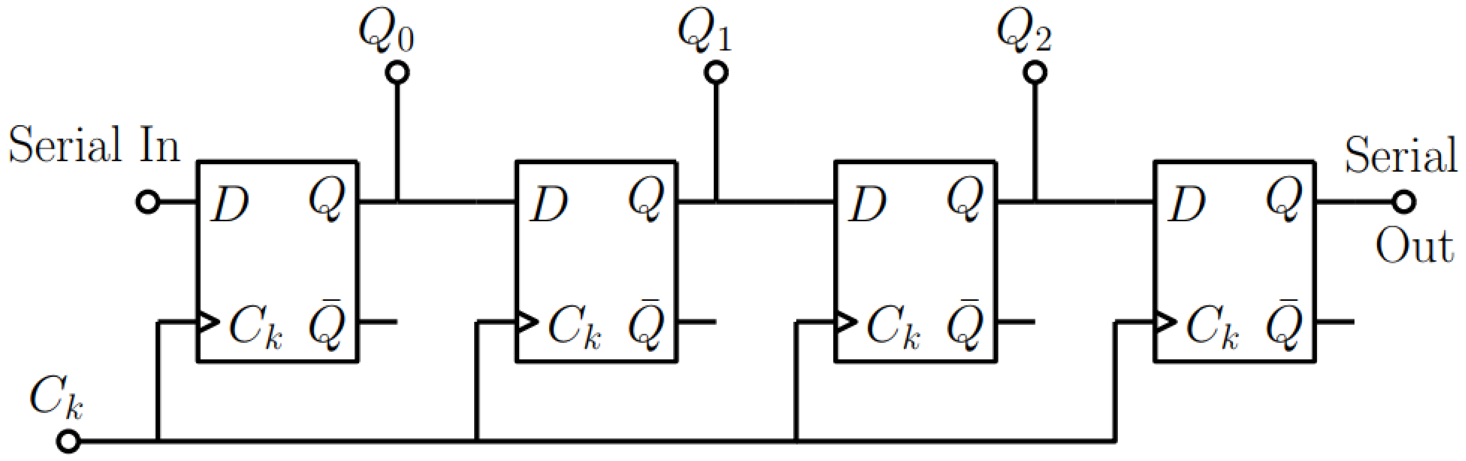
Shift Registers





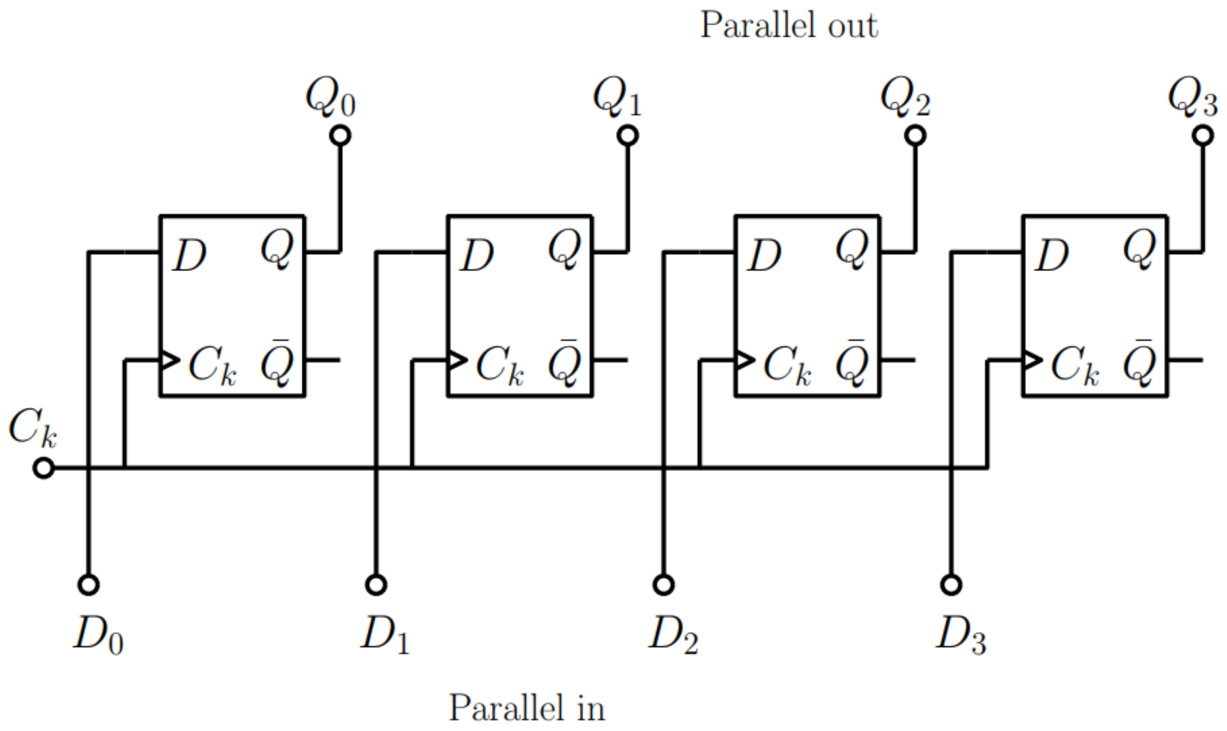
Shift Registers

Simple example - ***Serial-In/Parallel-Out Shift Registers***



Shift Registers

Simple example - ***Parallel-In/Parallel-Out Shift Registers***



Counter

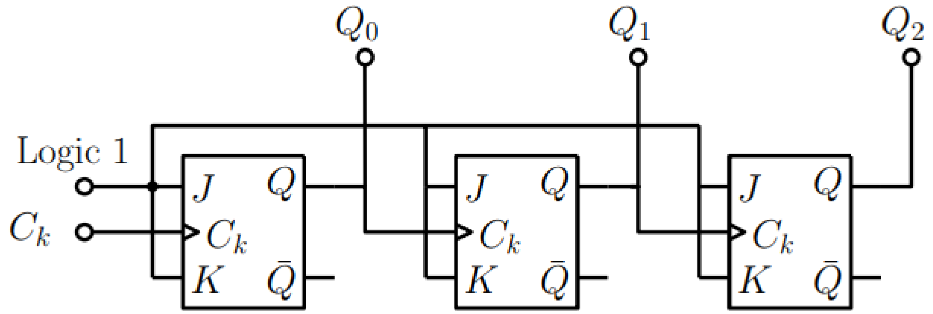
The shift register can be used as a counter because the data are shifted for each clock pulse. A counter is a register that goes through a predetermined sequence of states when input pulses are received. Besides, computers, timers, frequency meters, and various other digital devices contain counters for counting events.

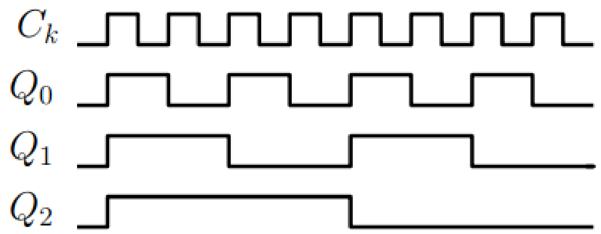
There are some types of counters:

- *ripple* (*asynchronous*),
- *synchronous,*
- *ring*.

Counter

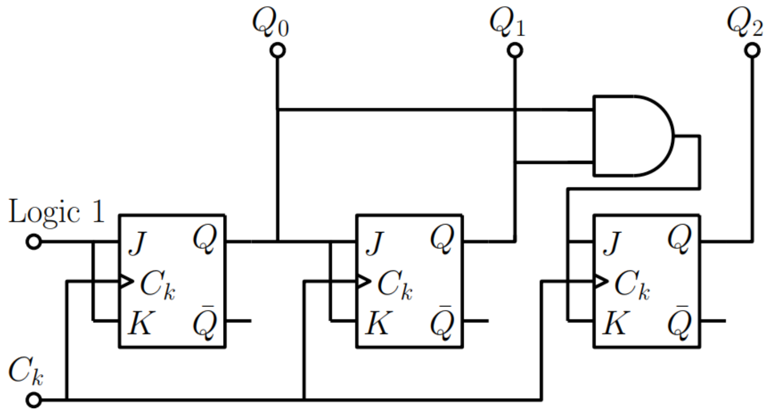
Ripple Counter (Asynchronous Counter)

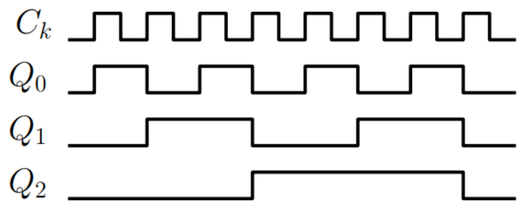




Counter

Synchronous Counter

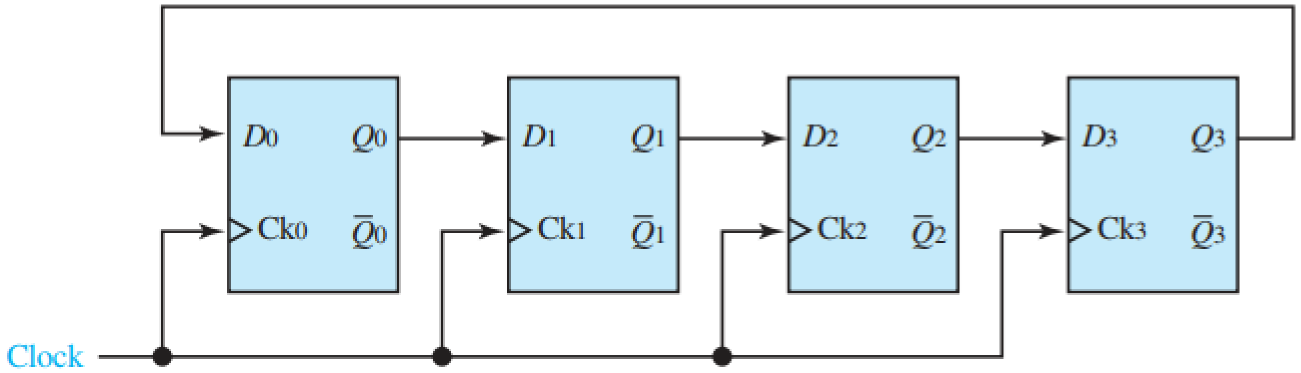




Counter

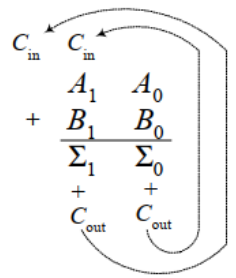
Ring Counter (Shift Register Sequencer).

The ring counter (shift register sequencer) is a unique type of shift register that incorporates feedback from the output of the last flip-flop to the input of the first flip-flop.



Binary Adders

Mechanics of adding



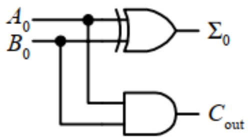
MSB addition

A_1	B_1	C_{in}	Σ_1	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

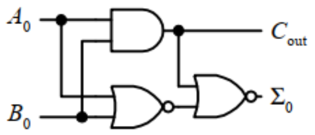
LSB addition

A_0	B_0	Σ_0	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

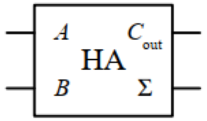
Half-adder



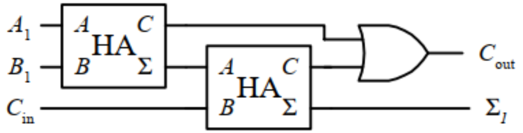
alternative representation



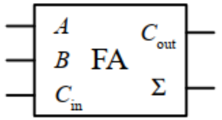
half-adder symbol



Full-adder

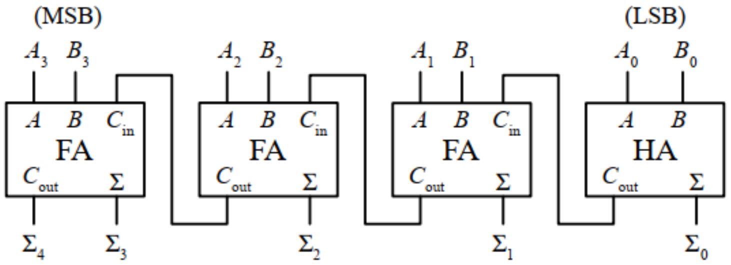


full-adder symbol



4-bit adder

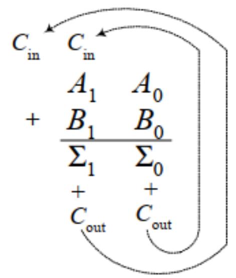
$$\begin{array}{r} A_3\,A_2\,A_1\,A_0 \\ +\quad B_3\,B_2\,B_1\,B_0 \\ \hline \Sigma_4\,\Sigma_3\,\Sigma_2\,\Sigma_1\,\Sigma_0 \end{array}$$



(MSB)
(LSB)
A_3 B_3
A_2 B_2
A_1 B_1
A_0 B_0
A B C_{in} FA C_{out} Σ
A B C_{in} FA C_{out} Σ
A B C_{in} FA C_{out} Σ
A B HA C_{out} Σ
Σ_4 Σ_3 Σ_2 Σ_1 Σ_0

Binary Adders

Mechanics of adding



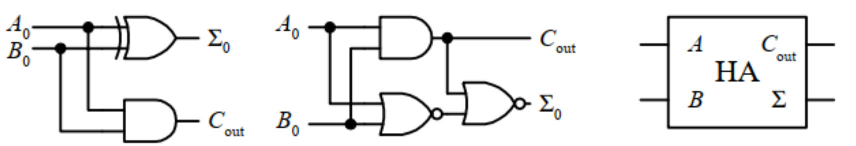
MSB addition

A_1	B_1	C_{in}	Σ_1	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

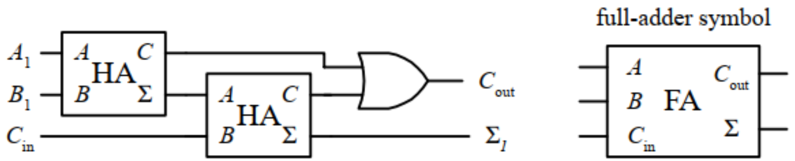
LSB addition

A_0	B_0	Σ_0	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

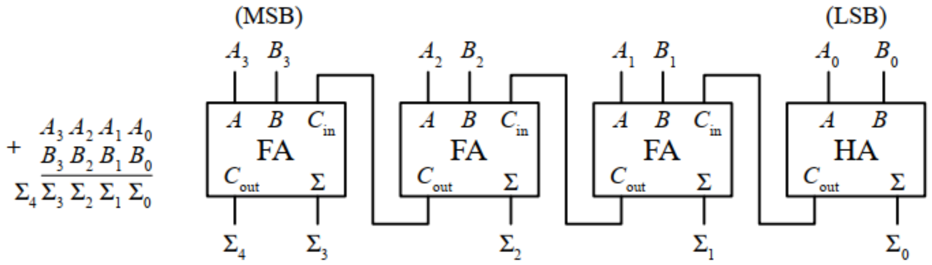
Half-adder



Full-adder



4-bit adder



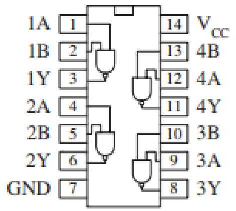
(MSB)
(LSB)
A_3 B_3
A_2 B_2
A_1 B_1
A_0 B_0
A B C_{in} FA C_{out} Σ
A B C_{in} FA C_{out} Σ
A B C_{in} FA C_{out} Σ
A B HA C_{out} Σ
Σ_4 Σ_3 Σ_2 Σ_1 Σ_0

$$\begin{array}{r} A_3\,A_2\,A_1\,A_0 \\ +\;B_3\,B_2\,B_1\,B_0 \\ \hline \Sigma_4\,\Sigma_3\,\Sigma_2\,\Sigma_1\,\Sigma_0 \end{array}$$

Digital Logic Gate ICs

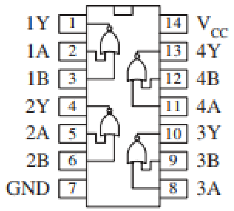
74HC00 Series

Quad 2-input NAND



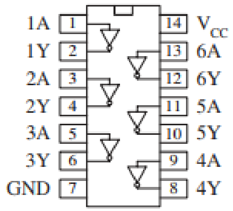
7400, 74LS00, 74F00, 74HC00, etc.

Quad 2-input NOR



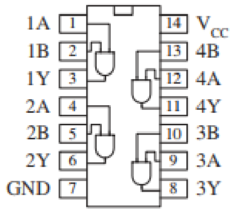
7402, 74LS02, 74F02, 74HC02, etc.

Hex Inverter



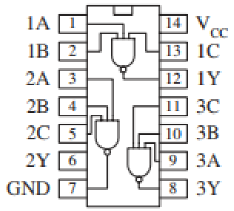
7404, 74LS04, 74F04, 74HC04, etc.

Quad 2-input AND



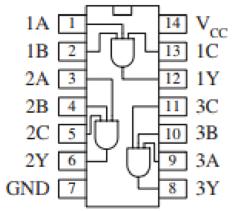
7408, 74LS08, 74F08, 74HC08, etc.

Triple 3-input NAND



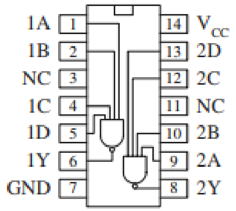
7410, 74LS10, 74F10, 74HC10, etc.

Triple 3-input AND



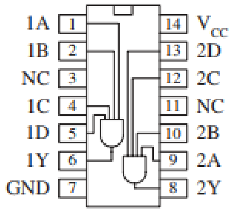
7411, 74LS11, 74F11, 74HC11, etc.

Dual 4-input NAND



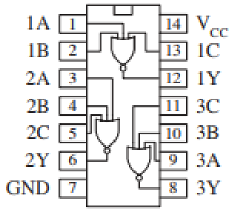
7420, 74LS20, 74F20, 74HC20, etc.

Dual 4-input AND



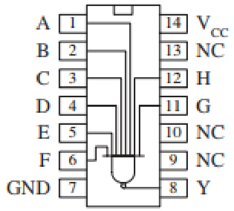
7421, 74LS21, 74F21, 74HC21, etc.

Triple 3-input NOR



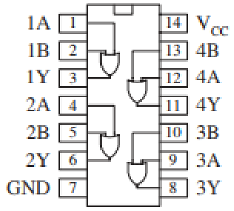
7427, 74LS27, 74F27, 74HC27, etc.

8-input NAND



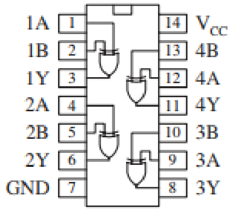
7430, 74LS30, 74F30, 74HC30, etc.

Quad 2-input OR



7432, 74LS32, 74F32, 74HC32, etc.

Quad 2-input XOR

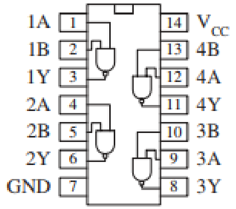


7486, 74LS86, 74F86, 74HC86, etc.

Digital Logic Gate ICs

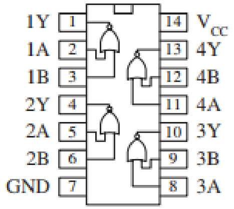
74HC00 Series

Quad 2-input NAND



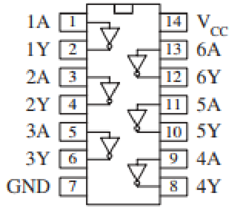
7400, 74LS00, 74F00, 74HC00, etc.

Quad 2-input NOR



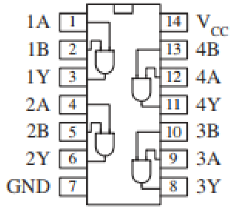
7402, 74LS02, 74F02, 74HC02, etc.

Hex Inverter



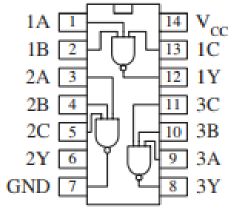
7404, 74LS04, 74F04, 74HC04, etc.

Quad 2-input AND



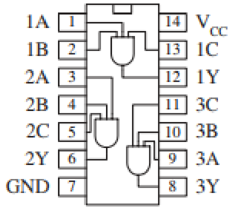
7408, 74LS08, 74F08, 74HC08, etc.

Triple 3-input NAND



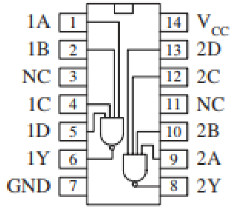
7410, 74LS10, 74F10, 74HC10, etc.

Triple 3-input AND



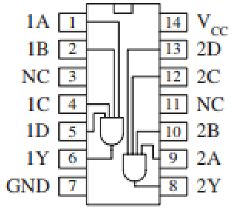
7411, 74LS11, 74F11, 74HC11, etc.

Dual 4-input NAND



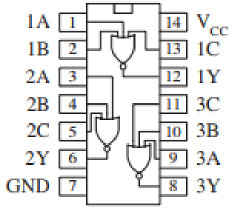
7420, 74LS20, 74F20, 74HC20, etc.

Dual 4-input AND



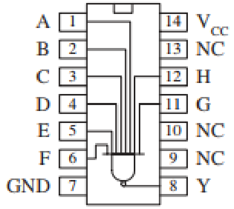
7421, 74LS21, 74F21, 74HC21, etc.

Triple 3-input NOR



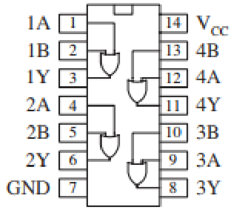
7427, 74LS27, 74F27, 74HC27, etc.

8-input NAND



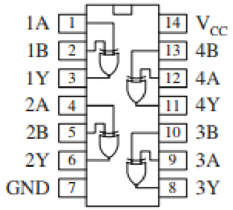
7430, 74LS30, 74F30, 74HC30, etc.

Quad 2-input OR



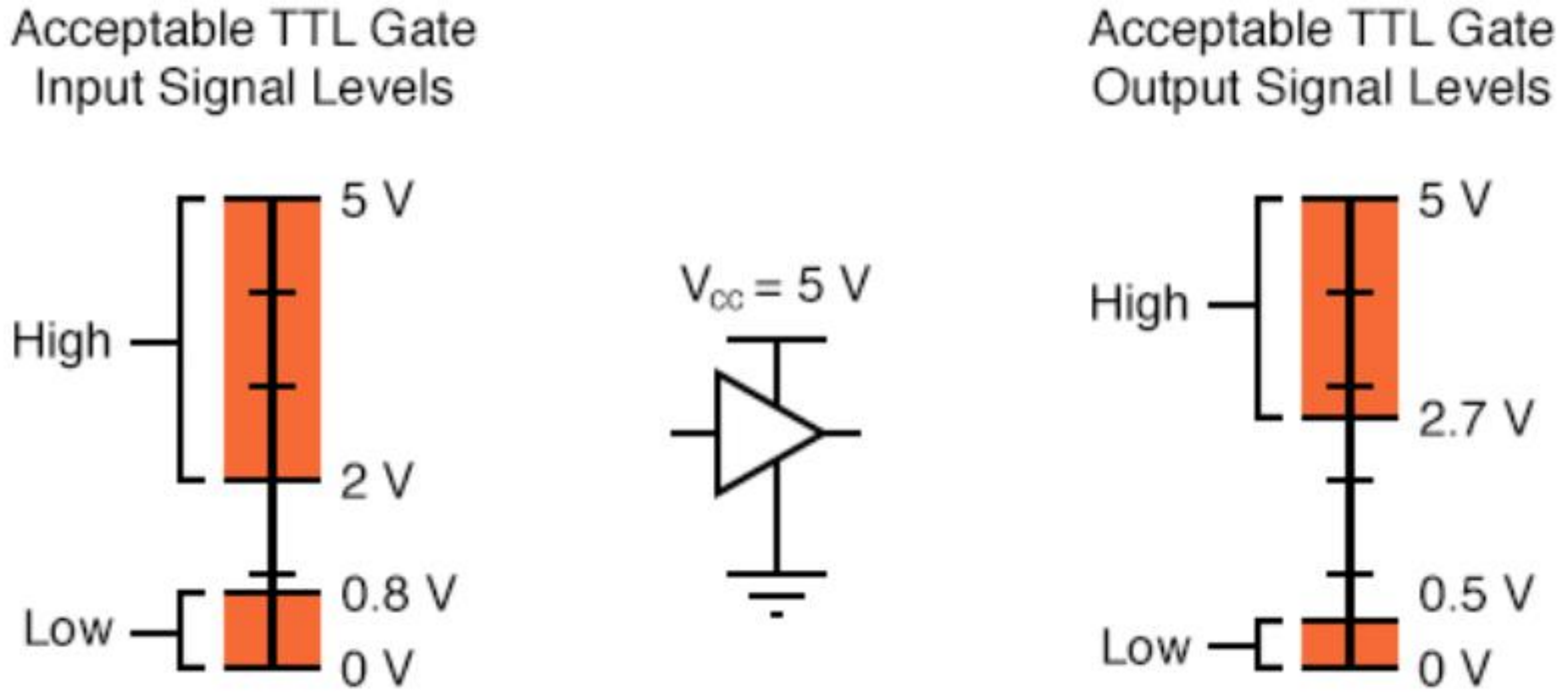
7432, 74LS32, 74F32, 74HC32, etc.

Quad 2-input XOR

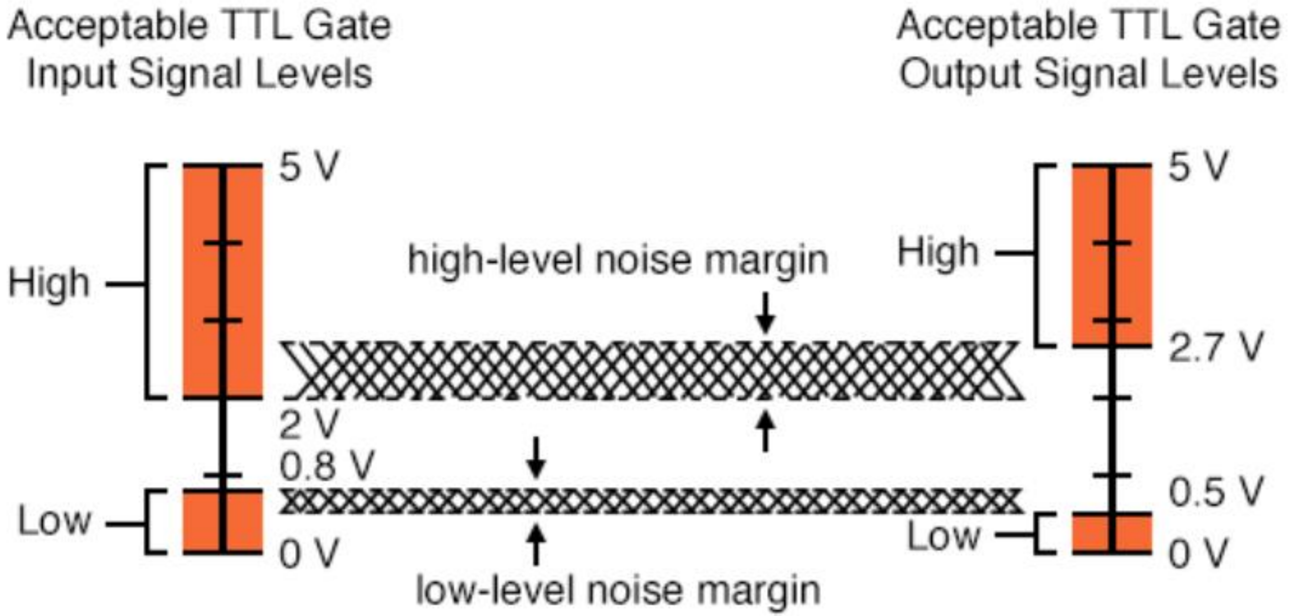


7486, 74LS86, 74F86, 74HC86, etc.

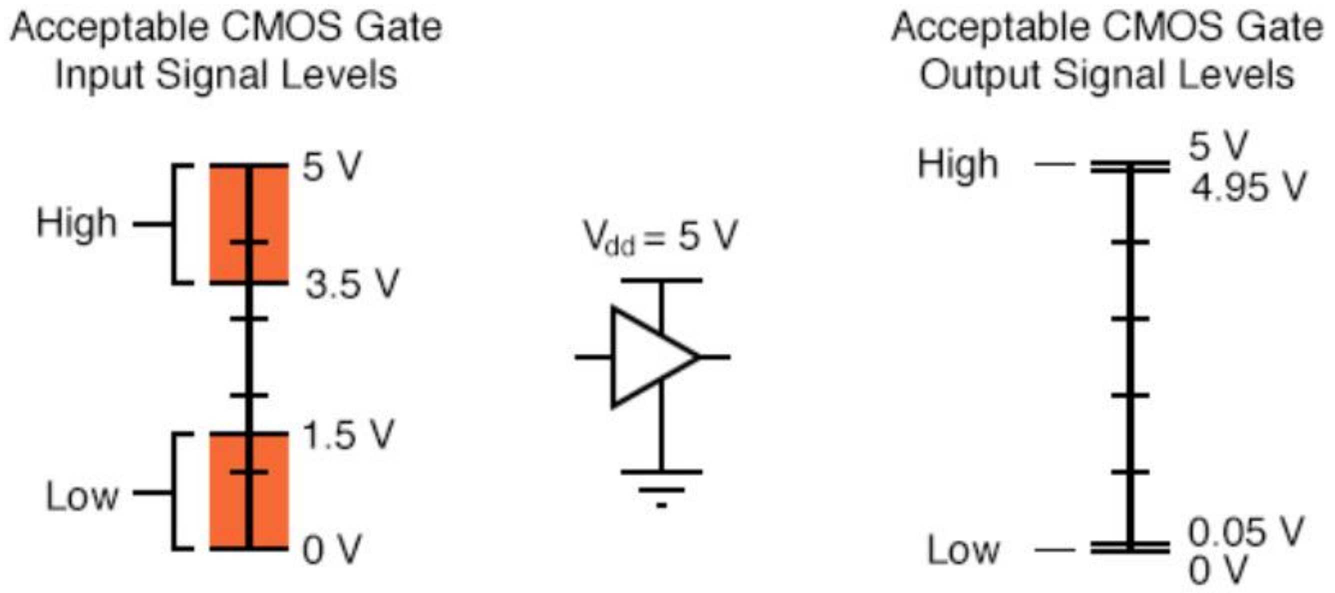
Logic Signal Voltage Levels


Acceptable TTL Gate Input Signal Levels
5 V
High
2 V
0.8 V
Low
0 V
V_{CC} = 5 V
Acceptable TTL Gate Output Signal Levels
5 V
High
2.7 V
0.5 V
Low
0 V

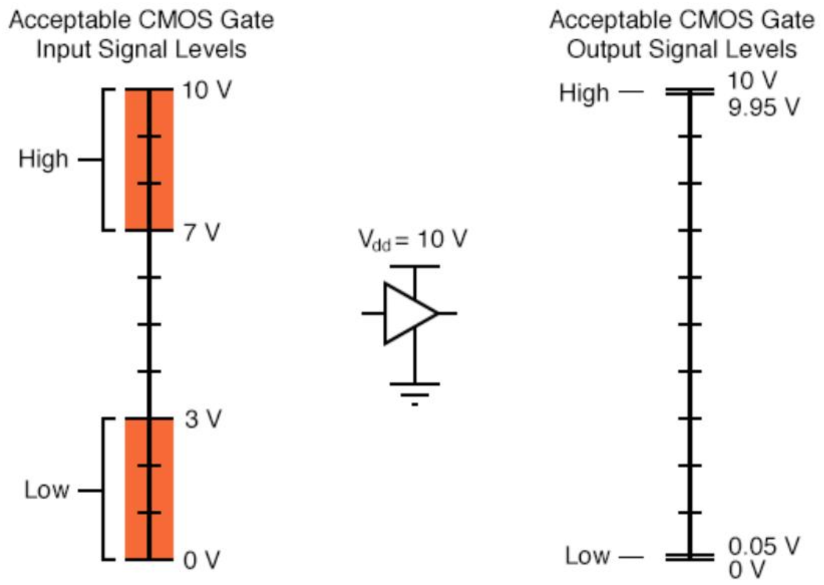
Logic Signal Voltage Levels

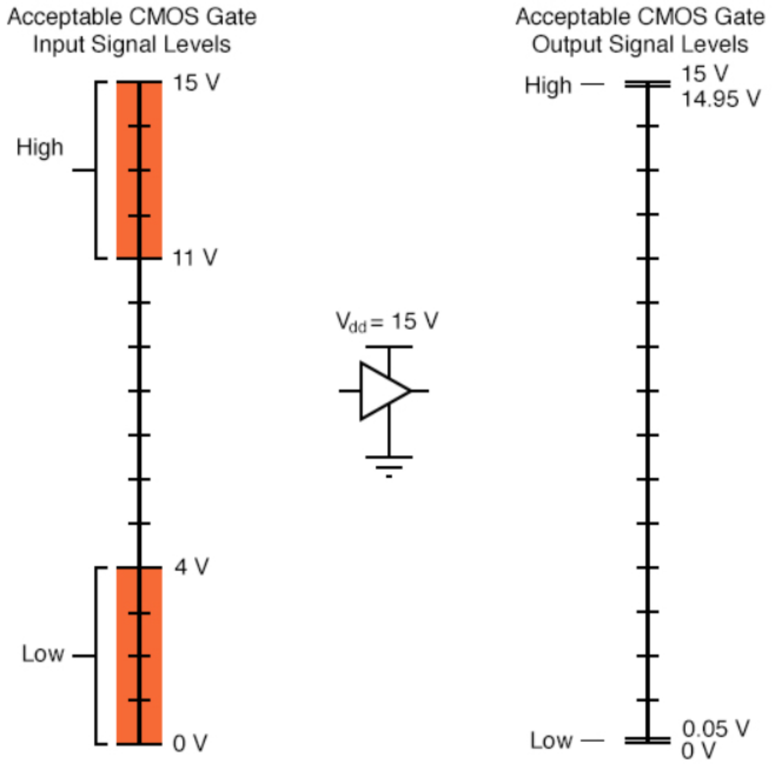


Voltage Tolerance of CMOS Gate Inputs


Acceptable CMOS Gate Input Signal Levels
High
Low
5 V
3.5 V
1.5 V
0 V
V_{dd} = 5 V
Acceptable CMOS Gate Output Signal Levels
High
Low
5 V
4.95 V
0.05 V
0 V

CMOS Noise Margin


Acceptable CMOS Gate Input Signal Levels
10 V
High
7 V
3 V
Low
0 V
Vdd = 10 V
Acceptable CMOS Gate Output Signal Levels
High
10 V
9.95 V
Low
0.05 V
0 V


Acceptable CMOS Gate Input Signal Levels
15 V
High
11 V
4 V
Low
0 V
Vdd = 15 V
Acceptable CMOS Gate Output Signal Levels
High
15 V
14.95 V
Low
0.05 V
0 V

References

1. Sarma M. S. Introduction to electrical engineering. – New York : Oxford University Press, 2001. – C. 715-716.
2. Tokheim R. L. Digital Electronics: Principles and Applications, 8th Edition. – McGraw-Hill, Inc., 2014.
3. Kleitz W. Digital Electronics: A practical approach with VHDL. – Prentice Hall, 2011.
4. Harris S., Harris D. Digital design and computer architecture: arm edition. – Morgan Kaufmann, 2015.
5. Paul Scherz, Simon Monk. Practical Electronics for Inventors, Fourth Edition. - McGraw-Hill, Inc., 2016.

Attendance check

https://forms.yandex.com/cloud/635481a073cee77f17e7626e/

https://clck.ru/32Skte

ITMO

Thank you for your attention!