Electronic Technology Design and Workshop

Presented and updated by

Przemek Sekalski

DMCS room 2

2007

Electronic Technology Design and Workshop

Lecture 7

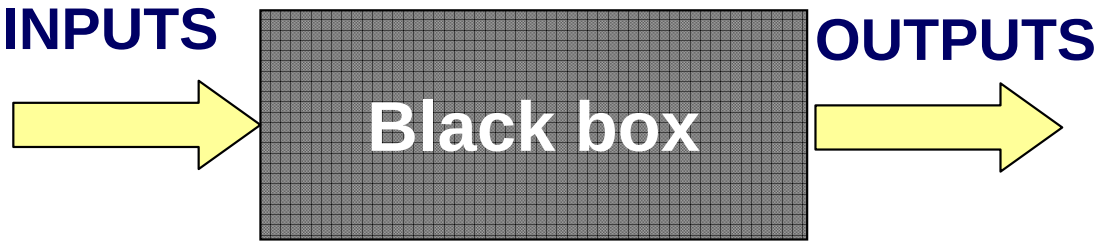
Electronic Circuit Synthesis

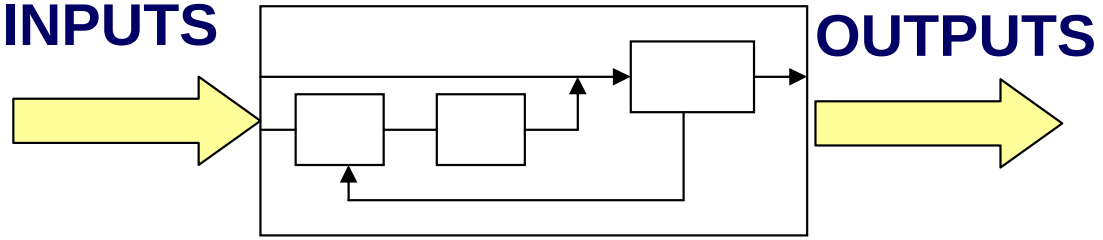
Synthesis of electronic circuits

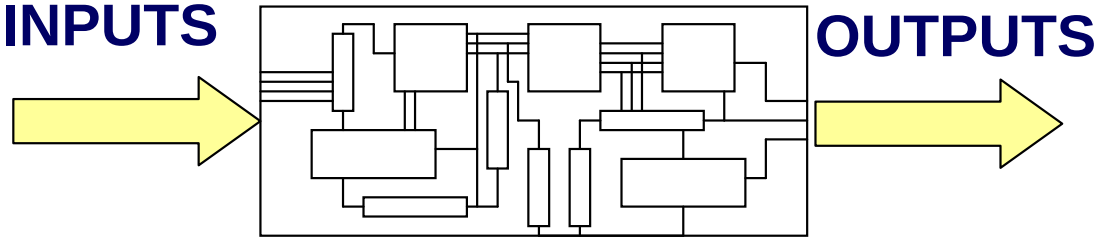
Behavioral representation

Structural representation

Physical representation



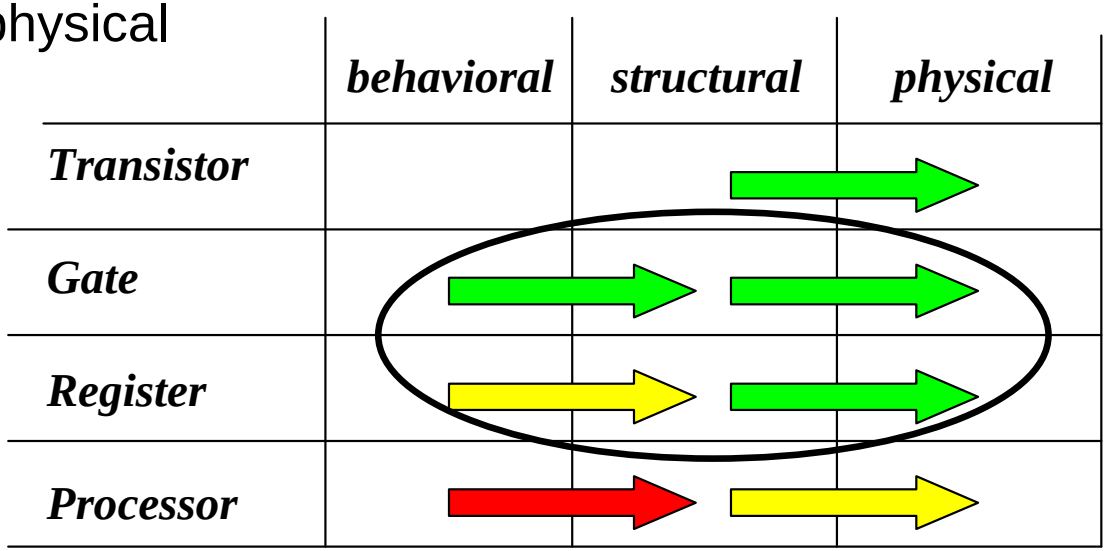




Synthesis of digital electronic circuits

Automatic translation of the circuit description to less abstract representations e.g.

a) from behavioural to structural
b) from structural to physical
c) from behavioural to physical



Not supported
(some support possible in specific cases - *software-hardware codesign*)

Partially supported (fully supported for certain types of digital designs)

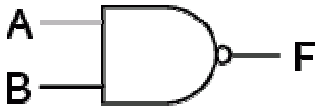
Fully supported

Digital synthesis

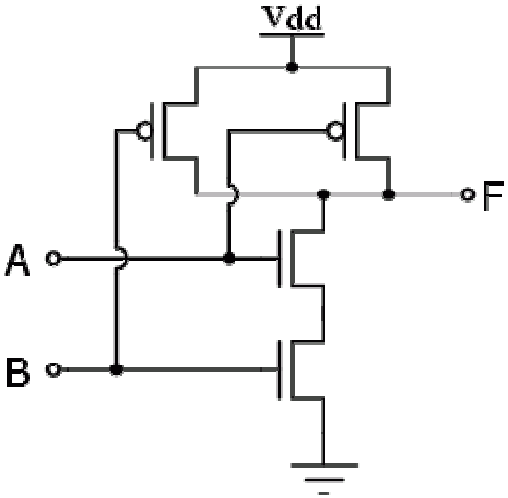
Behavioural

$$F = \overline{A \cdot B}$$

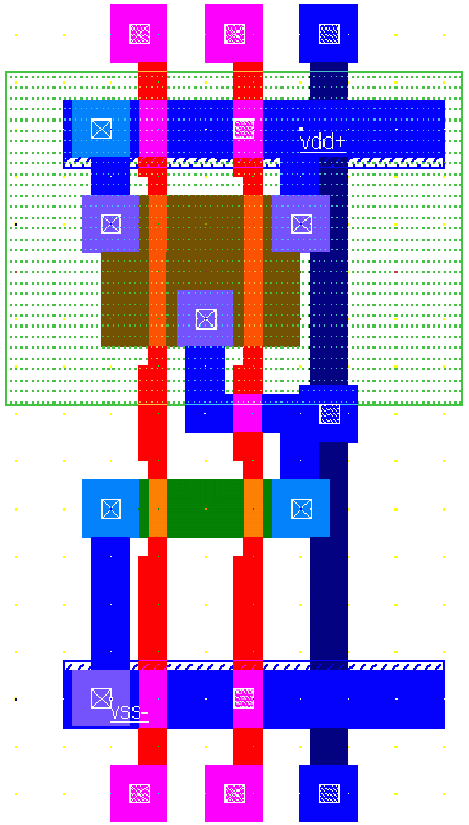
Structural (gate level)



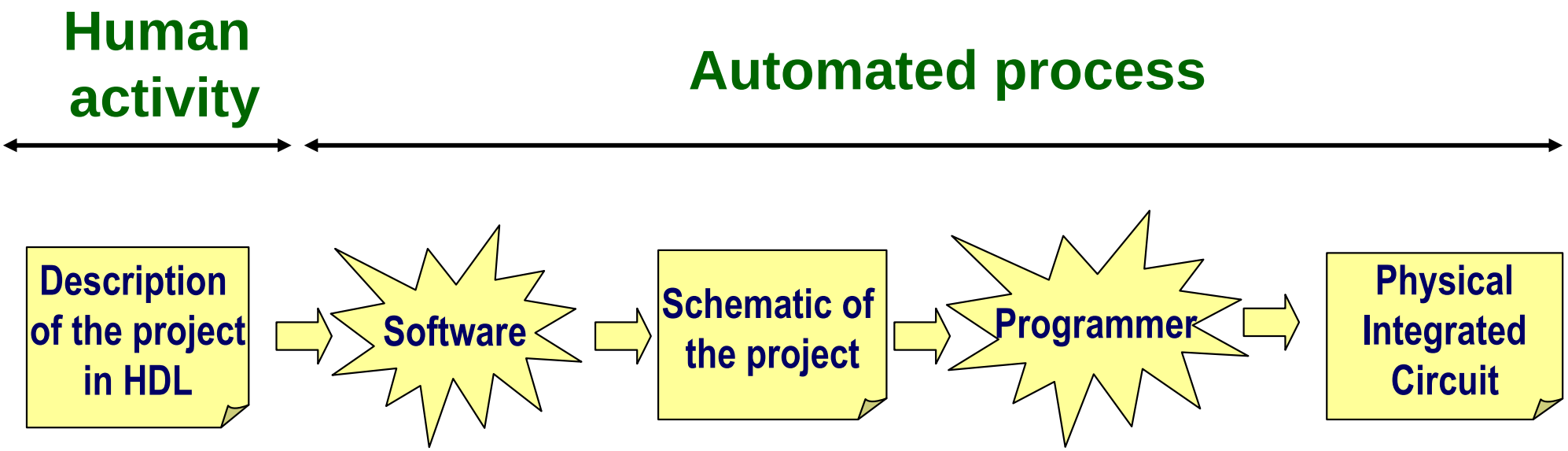
Structural (transistor level)



Physical (transistor level - IC layout)



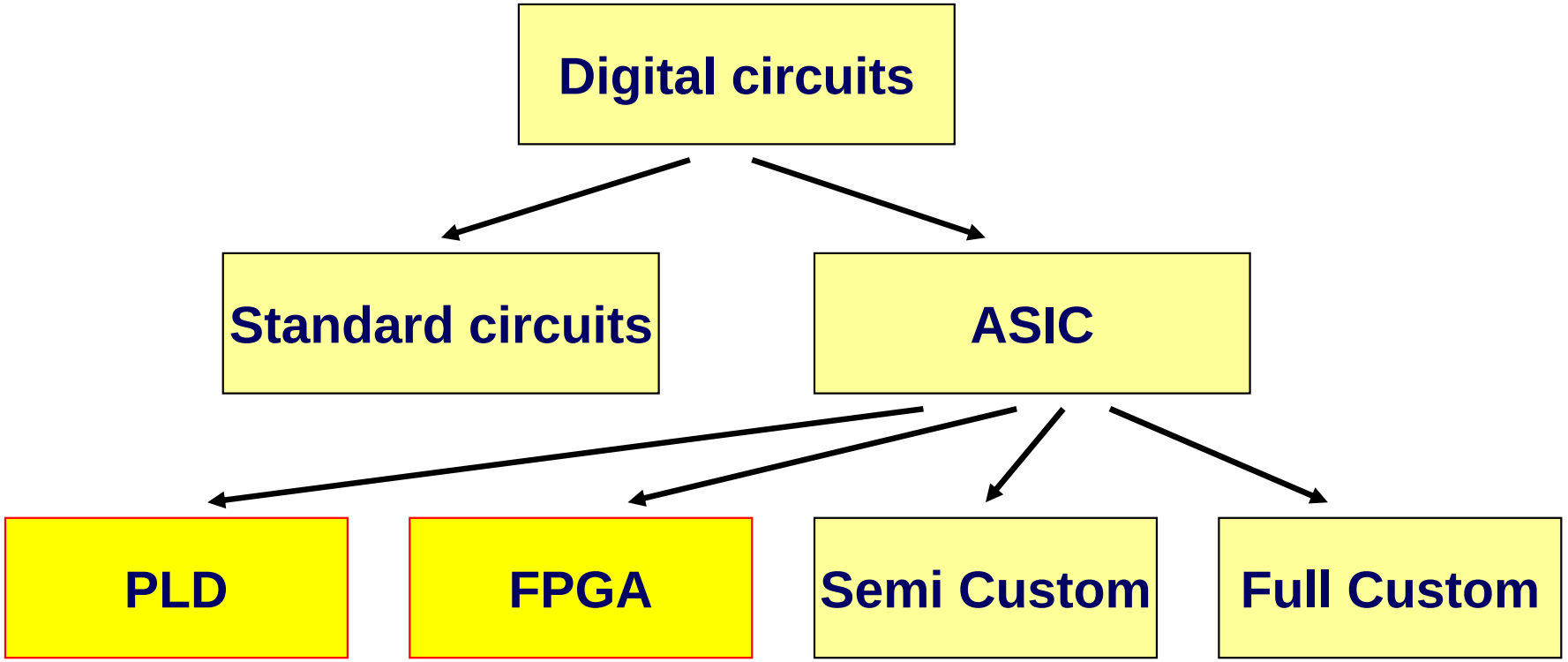
Digital synthesis with PLD



Compiler of Hardware Description Language (ABEL, Verilog, VHDL, …)

Device making the electrical connections inside Programmable Logic Circuits

Digital circuits classification



ASIC - Application Specific Integrated Circuit
PLD - Programmable Logic Device
FPGA - Field Programmable Gate Array
Semi Custom - Integrated Circuits designed by users with library cells
Full Custom - Integrated Circuits dsigned from scratch, designed or orderd by users

Digital circuits description languages

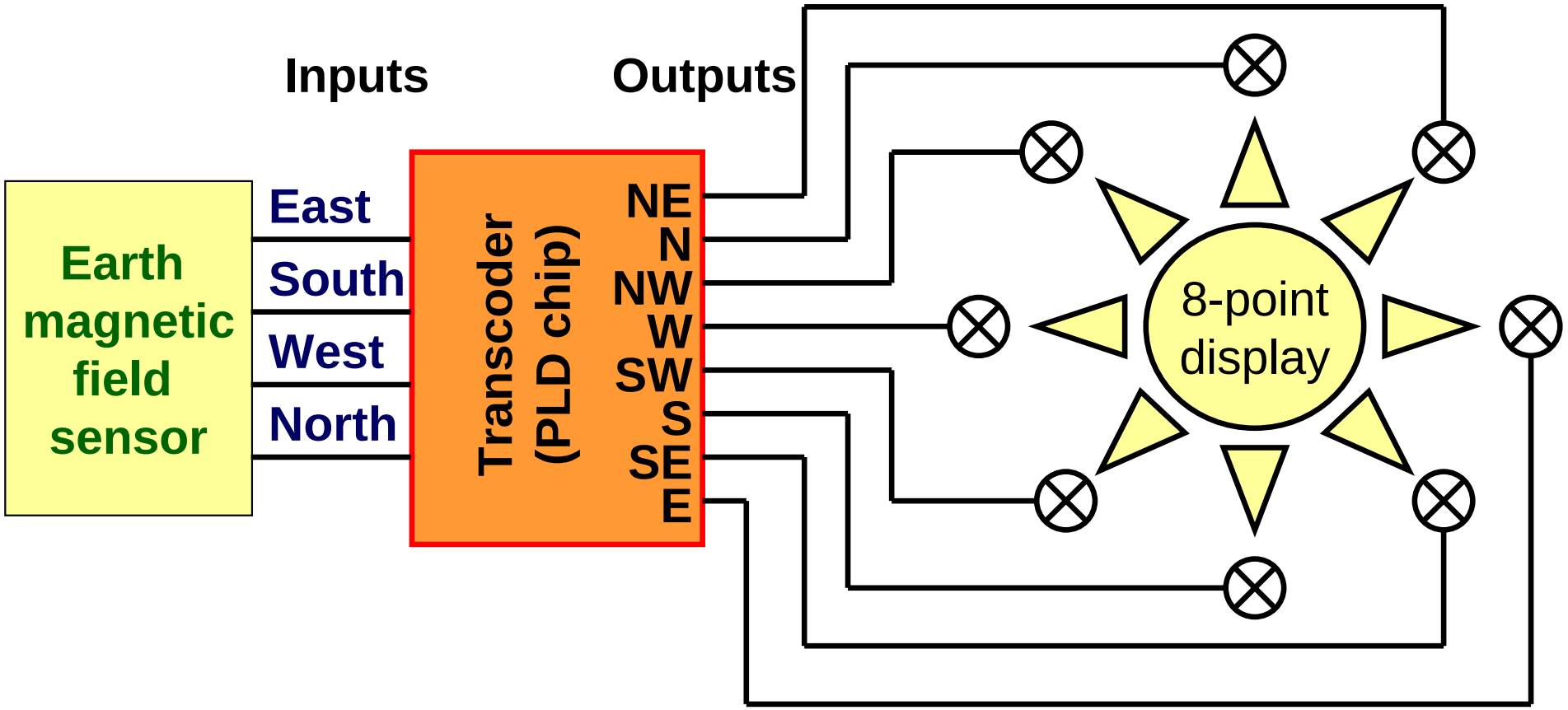
ABEL (Advanced Boolean Expression Language) -
one of the oldest HDL,
applicable for simple and less complex digital circuits,
originally designed for programming certain types of PLDs,
projects can be described by Boolean equations, truth tables or state-diagrams.

VHDL - Very High Speed Integrated Circuit (*VHSIC*) Hardware Description Language
language for general description of complex digital circuits,
acknowledged as IEEE standard (1987),
designed for uniform documentation and simulation of digital projects,
not targeted for any specific architecture,
allows the description of parallel and sequential processing

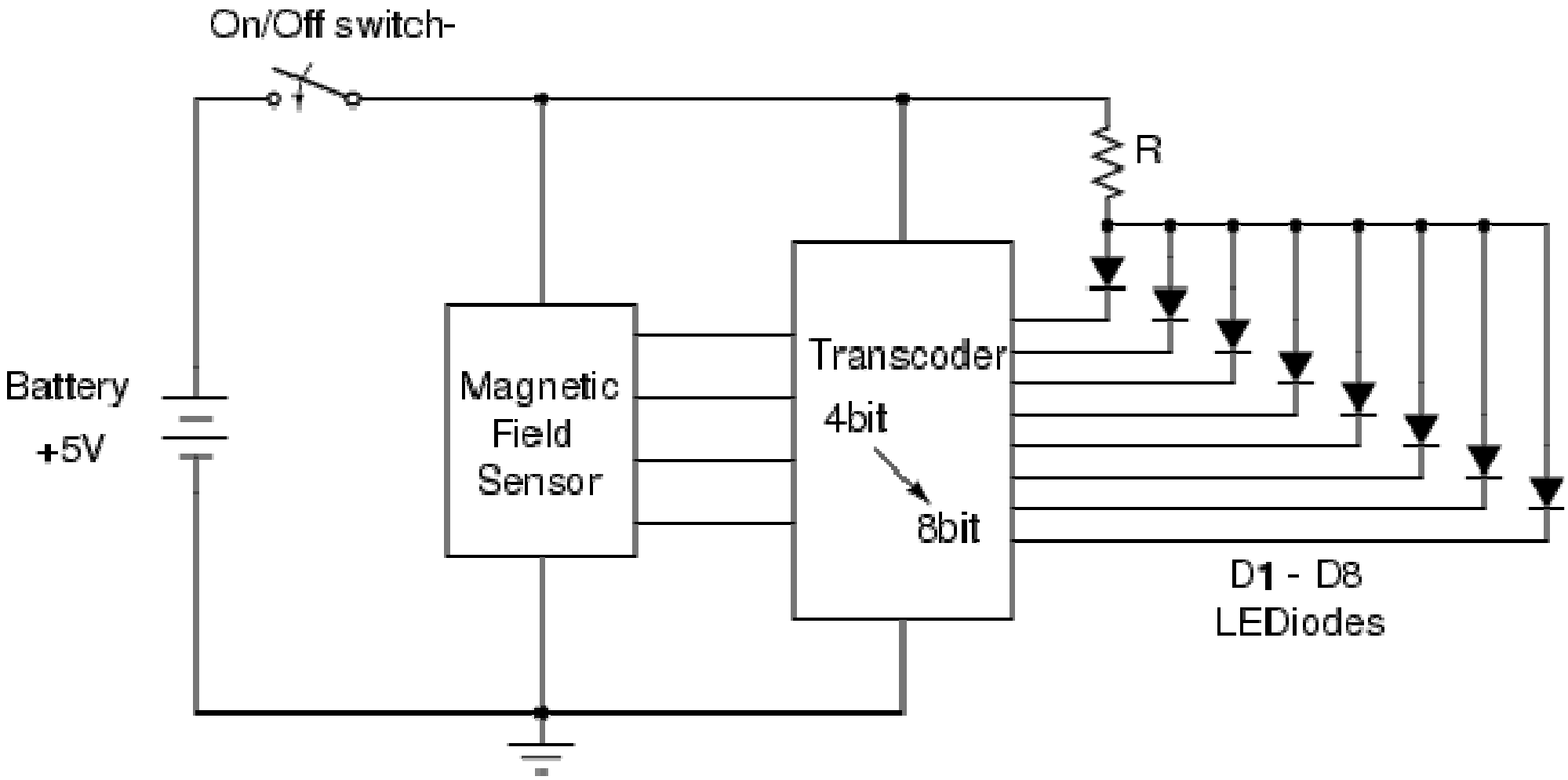
Verilog -
most popular industry standard,
acknowledged as IEEE standard (1995)
less complex than VHDL and easier to learn, based on C-language structure,
best suited for complex FPGA circuits

Example of behavioral description

Electronic Compass



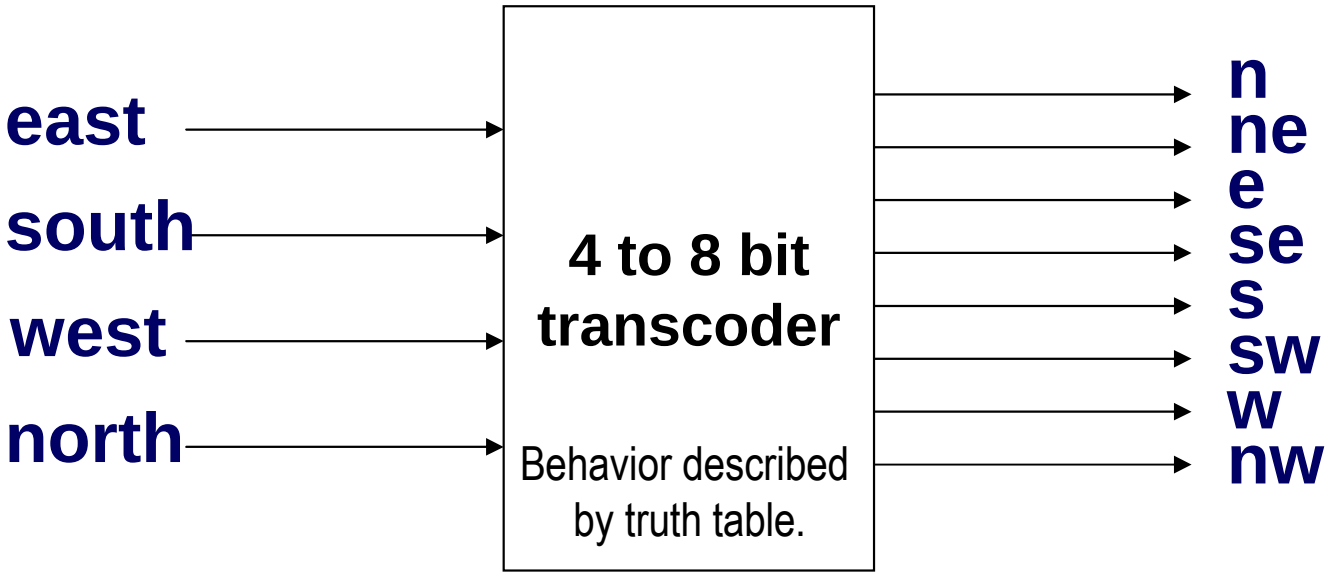
Electronic Compass - schematic



Electronic Compass - transcoder truth-table

Inputs				Outputs							
North	West	East	South	N	NE	E	SE	S	SW	W	NW
0	1	1	1	0	1	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1
1	1	1	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	0	1	1
1	0	1	1	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	0

Electronic Compass – transcoder chip



Input signals **Output signals**

The ABEL code (part I)

```
module transcoder;
title '4 to 8 bit transcoder for electronic comapss'

transcoder device 'P16H8';
```

Beginning of the ABEL program

```
north, west, east, south    pin;
n, ne, e, se, s, sw, w, nw  pin istype 'com';
```

Declarations of input and output signals (similar to variables in programming languages)

```
on  = 0;
off = 1;

yes = 0;
no  = 1;
```

Definitions of constants (in order to improve clarity of the program)

The ABEL code (part II)

```
truth_table
  ([north,west,east,south]->[  n,  ne,  e,  se,  s,  sw,  w, nw])
   [  yes,  no,  no,   no]->    [  on, off, off, off, off, off, off, off];
   [  yes,  no, yes,   no]->    [ off,  on, off, off, off, off, off, off];
   [   no,  no, yes,   no]->    [ off, off,  on, off, off, off, off, off];
   [   no,  no, yes,  yes]->    [ off, off, off,  on, off, off, off, off];
   [   no,  no,  no,  yes]->    [ off, off, off, off,  on, off, off, off];
   [   no, yes,  no,  yes]->    [ off, off, off, off, off,  on, off, off];
   [   no, yes,  no,   no]->    [ off, off, off, off, off, off,  on, off];
   [  yes, yes,  no,   no]->    [ off, off, off, off, off, off, off,  on];

test_vectors
  ([north,west,east,south]->[  n,  ne,  e,  se,  s,  sw,  w,  nw])
   [  yes,  no,  no,   no]->[  on, off, off, off, off, off, off, off];
   [  yes,  no, yes,   no]->[ off,  on, off, off, off, off, off, off];
   [   no,  no, yes,   no]->[ off, off,  on, off, off, off, off, off];
   [   no,  no, yes,  yes]->[ off, off, off,  on, off, off, off, off];
   [   no,  no,  no,  yes]->[ off, off, off, off,  on, off, off, off];
   [   no, yes,  no,  yes]->[ off, off, off, off, off,  on, off, off];
   [   no, yes,  no,   no]->[ off, off, off, off, off, off,  on, off];
   [  yes, yes,  no,   no]->[ off, off, off, off, off, off, off,  on];
end;
```

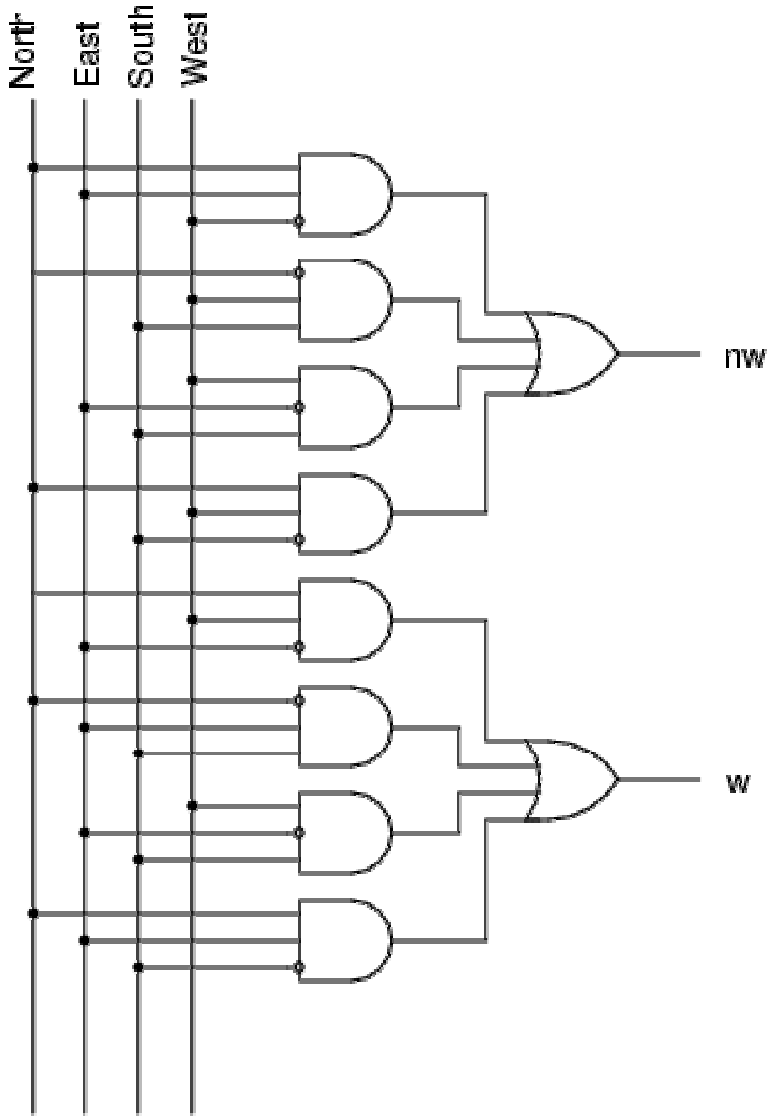
Behavioral description of the device functionality (here in form of truth-table)

Verification of correctness of generated equations

The code implementation

Automatically generated equations:

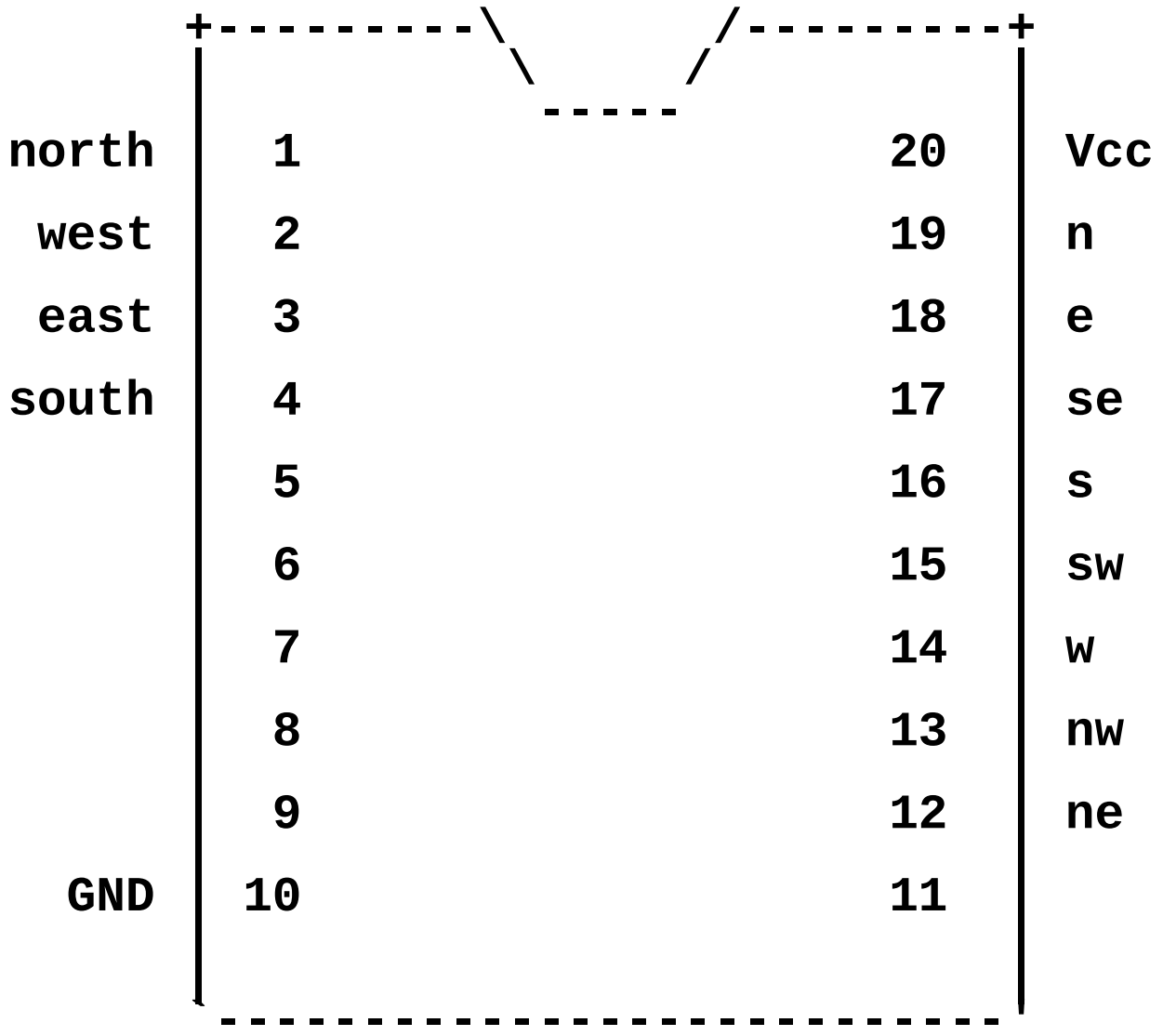
```
nw = (north & !west & east
    # !north & west & south
    # west & !east & south
    # north & west & !south);

w = (north & west & !east
    # !north & east & south
    # west & !least & south
    # north & east & !south);
…
…
...
```



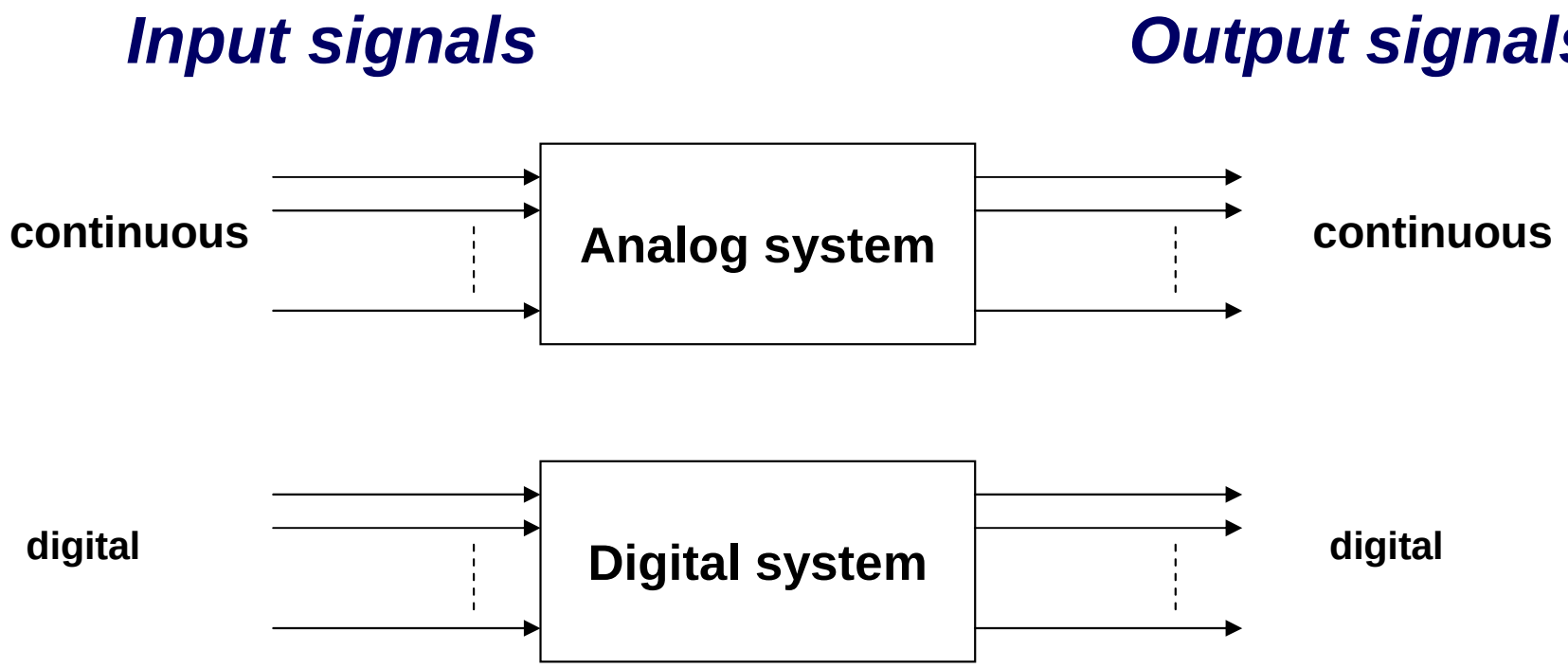
Automatically generated logical structure

P16H8 Chip Diagram

4 to 8 bit transcoder for electronic comapss



Digital vs Analog Systems

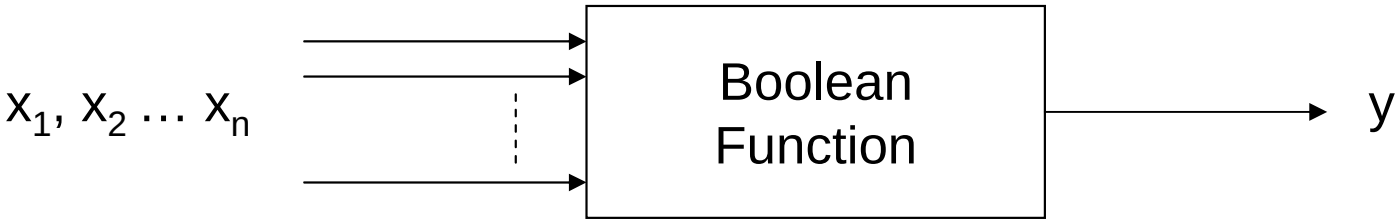


Boolean Algebra

Identity element	a+0=a	a*1=a
Commutativity	a+b=b+a	a*b=b*a
Associativity	a+(b+c)=(a+b)+c	a*(b*c)=(a*b)*c
Distributivity	a+(b*c)=(a+b)*(a+c)	a*(b+c)=(a*b)+(a*c)
Complement	a+ā=1	a*ā=0

Idempotency	a+a=a	a*a=a
Complement	a+1=1	a*0=0
Absorption	a+a*b=a	a*(a+b)=a
Element Elimination	a+ā*b=a+b	a*(ā+b)=a*b
De Morgan's Laws	$\overline{a+b}=\bar{a}*\bar{b}$	$\overline{a*b}=\bar{a}+\bar{b}$

Boolean Functions



$$y = B(x_1, x_2 \ldots x_n)$$

E.g.

$$f(a, b, c) = a*(b+\bar{c}) + (\bar{a}+b)*c$$

Truth Table notation

a	b	c	f(a, b, c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

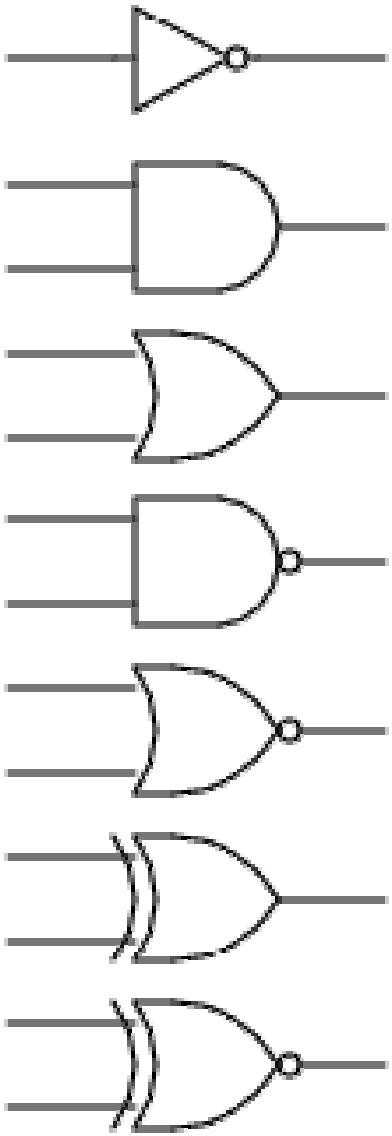
Minterm notation

$$f(a, b, c) = \bar{a}\bar{b}c + \bar{a}bc + a\bar{b}\bar{c} + ab\bar{c} + abc$$

Maxterm notation

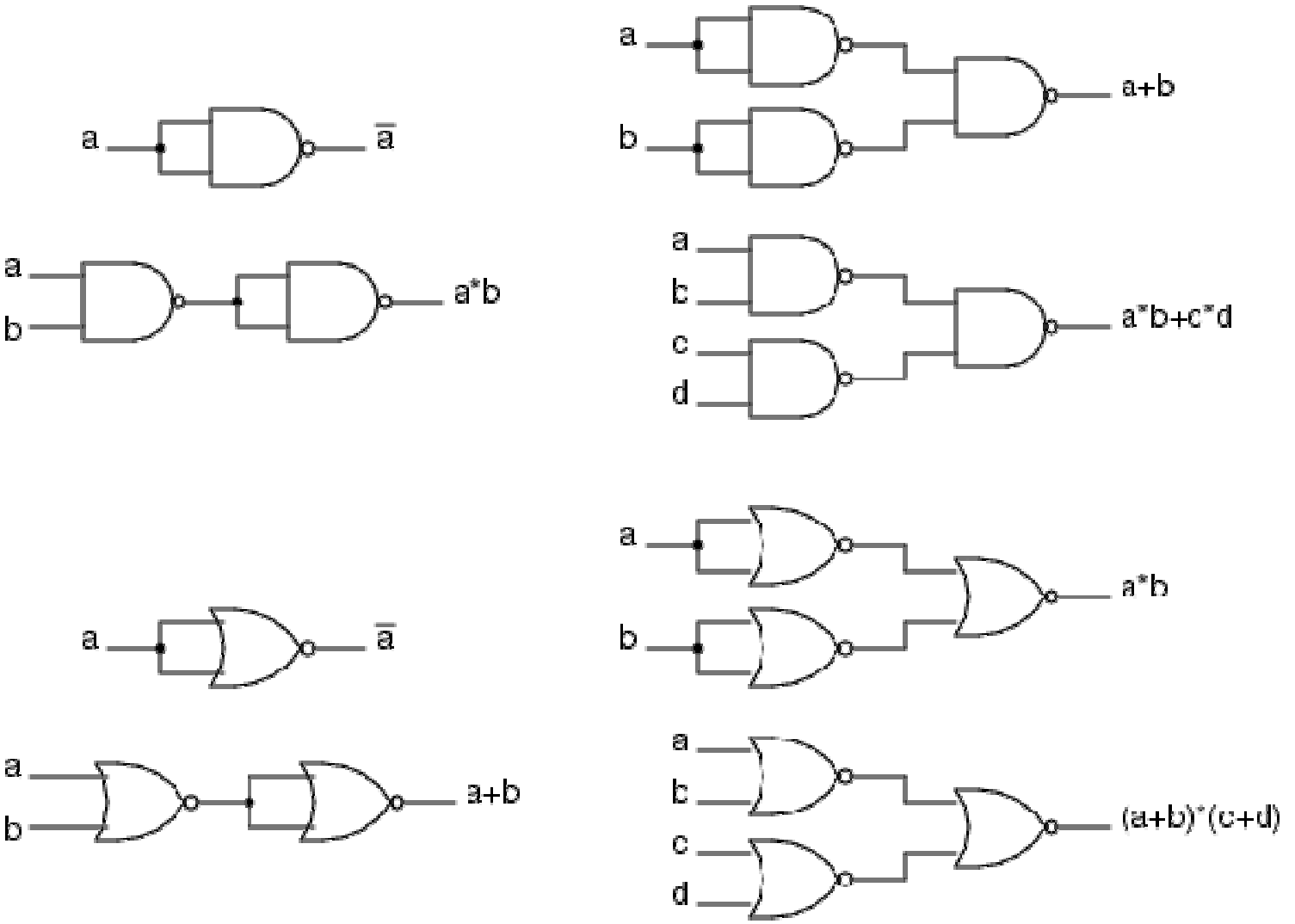
$$f(a, b, c) = (a+b+c)\,(a+\bar{b}+c)\,(\bar{a}+b+\bar{c})$$

Logic Gates



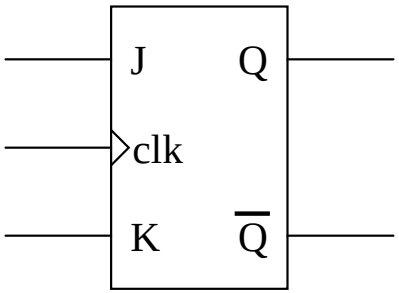
NOT	$F = \overline{a}$
AND	$F = a \cdot b$
OR	$F = a + b$
NAND	$F = \overline{a \cdot b}$
NOR	$F = \overline{a + b}$
XOR	$F = a \oplus b$
XNOR	$F = \overline{a \oplus b}$

Boolean Function Realization - examples



Memory Elements (flip-flops)

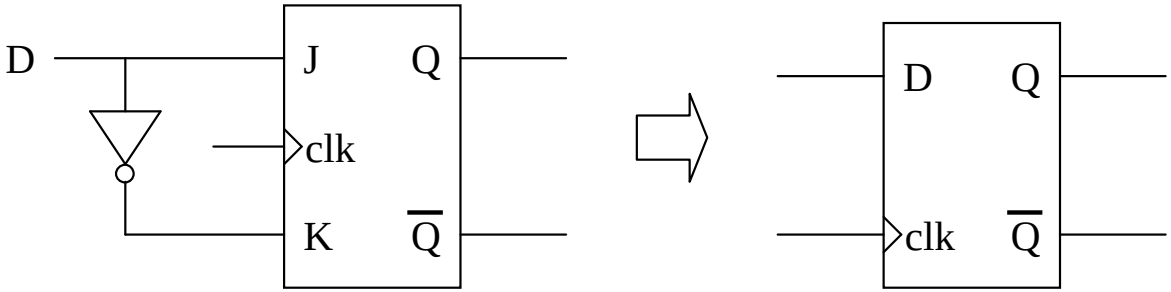
JK flip-flop



J(t)	K(t)	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}$(t)

$$Q(t+1) = J(t)\,\overline{Q}(t) + \overline{K}(t)\,Q(t)$$

Data flip-flop

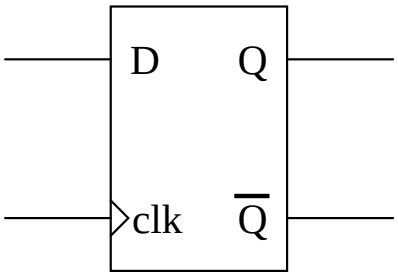


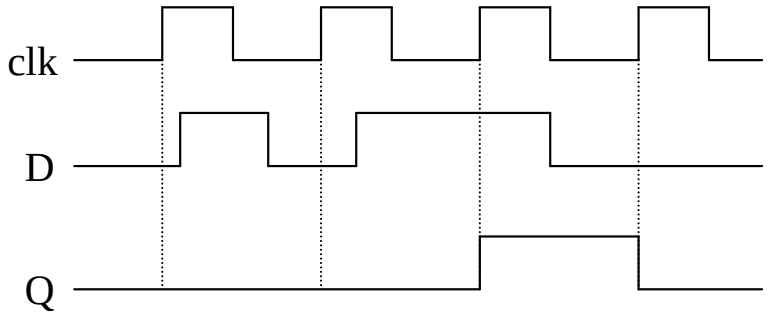
D(t)	Q(t+1)
0	0
1	1

$$Q(t+1) = D(t)$$

Memory Elements (flip-flops)

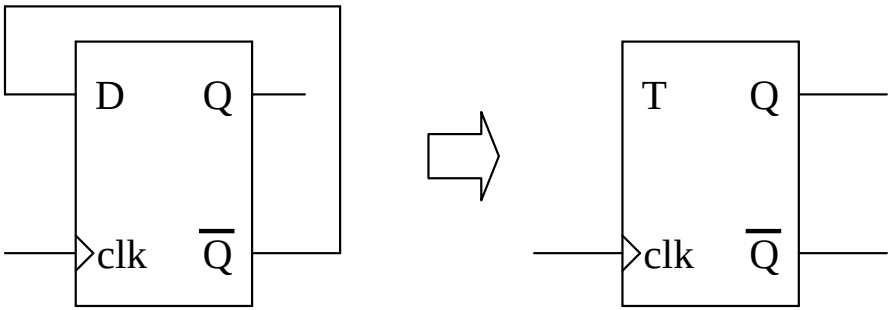
Data flip-flop

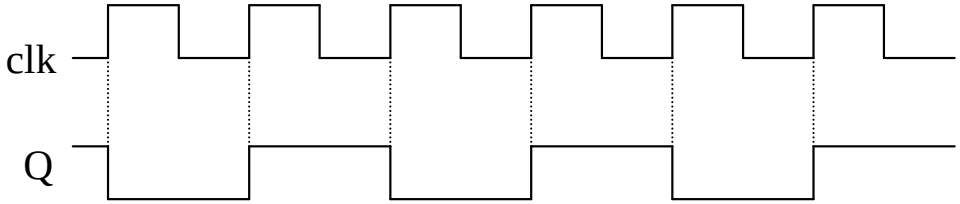




$$Q(t+1) = D(t)$$

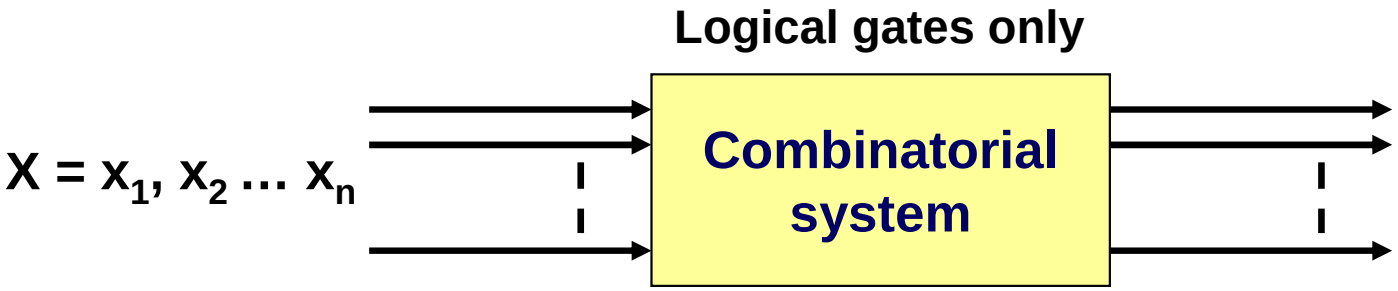
Toggle flip-flop





$$Q(t+1) = \overline{Q}(t)$$

Digital Systems



$Y = y_1, y_2 \dots y_m$

$y_1 = B_1(x_1, x_2 \dots x_n)$
$y_2 = B_2(x_1, x_2 \dots x_n)$
…
$y_m = B_m(x_1, x_2 \dots x_n)$

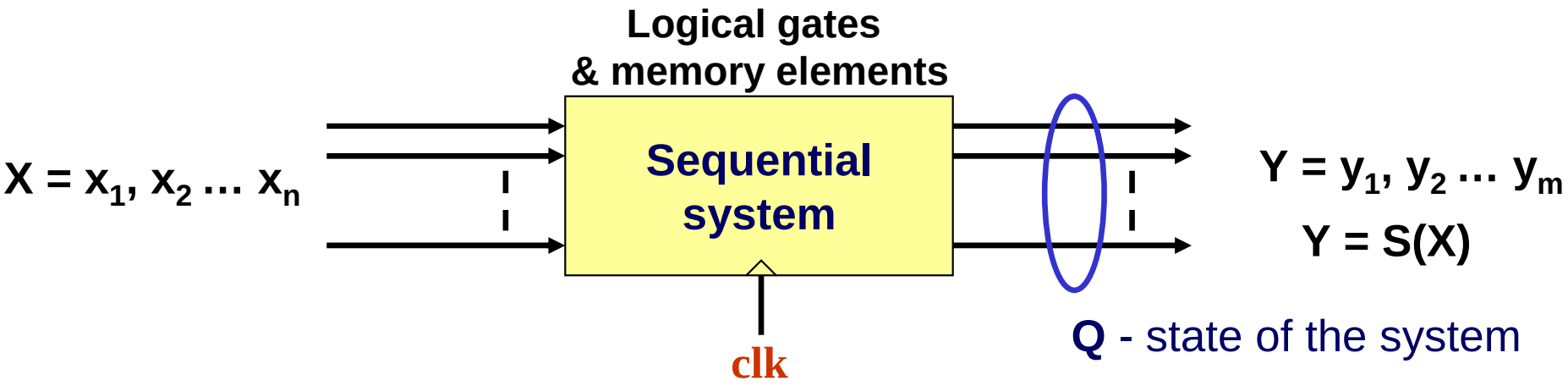
$Y = B(X)$

B_x - boolean function

Any change of input signals causes the appropriate change of output signals, as fast as it is possible, i.e after the propagation time of logical gates.

Output signals in combinatorial circuits depends only on the present input signals.

Digital Systems



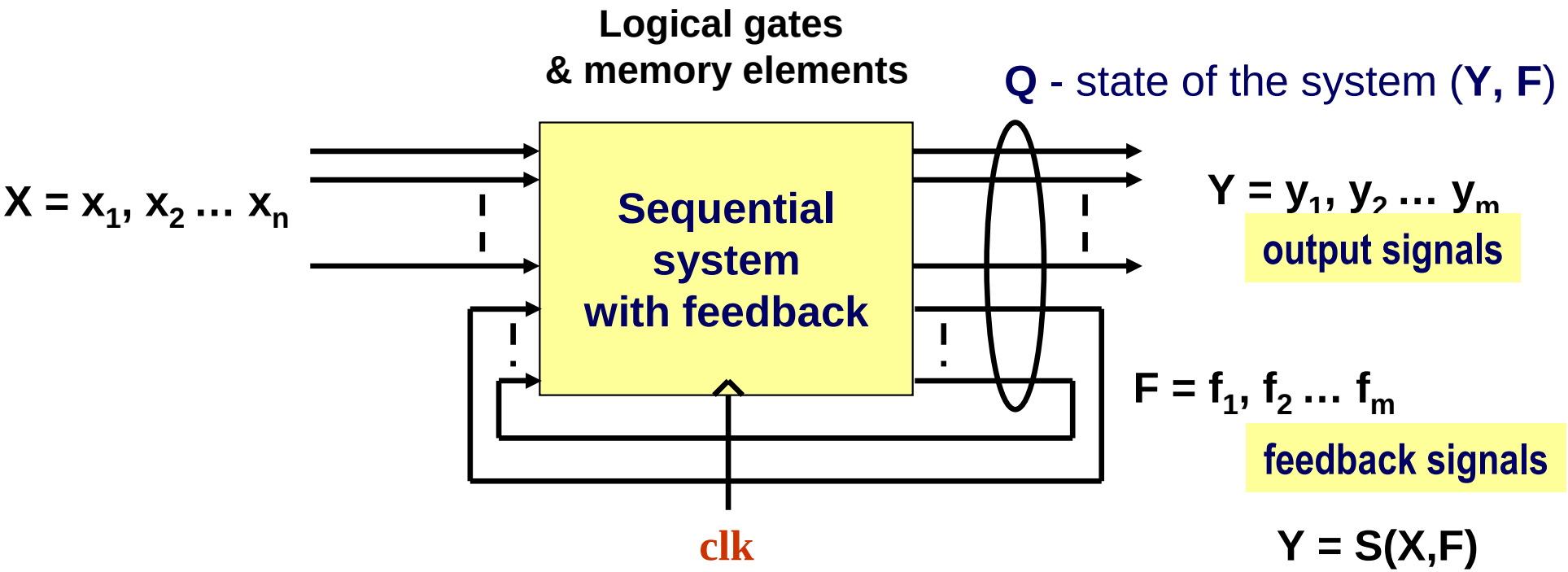
Output signals are the outputs of memory elements and they may change only after a new write operation is performed.

Output signals are Boolean functions (S) of input signals, but they may change only at prescribed moments.

When there is no write (store) operation, the output remains stable, regardless the state of the input signals.

The sufficient description of the sequential system is given by the set of signals at the output of the memory elements, i.e. *state of the system (Q)*

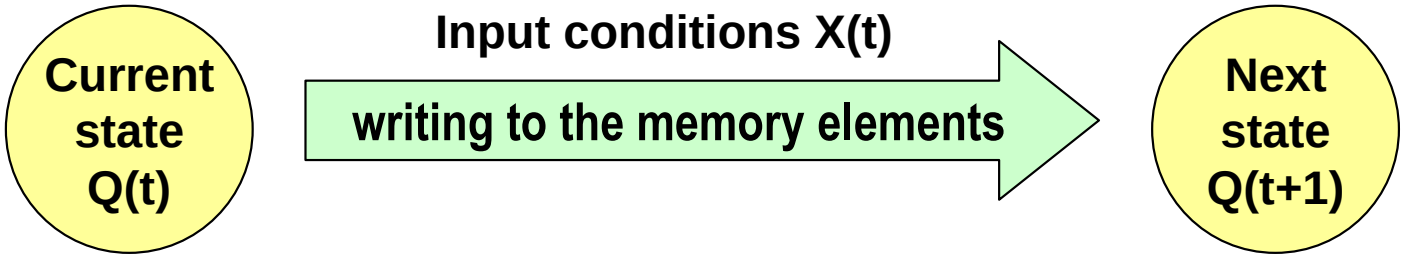
Digital Systems



Output signals are boolean functions (S) of input signals and feedback signals, but they may change only at prescribed moments.

The sufficient description of the sequential system is given by the set of signals at the output of the memory elements, i.e. *state of the system (Q) - output & feedback*

Description of Sequential Digital Systems

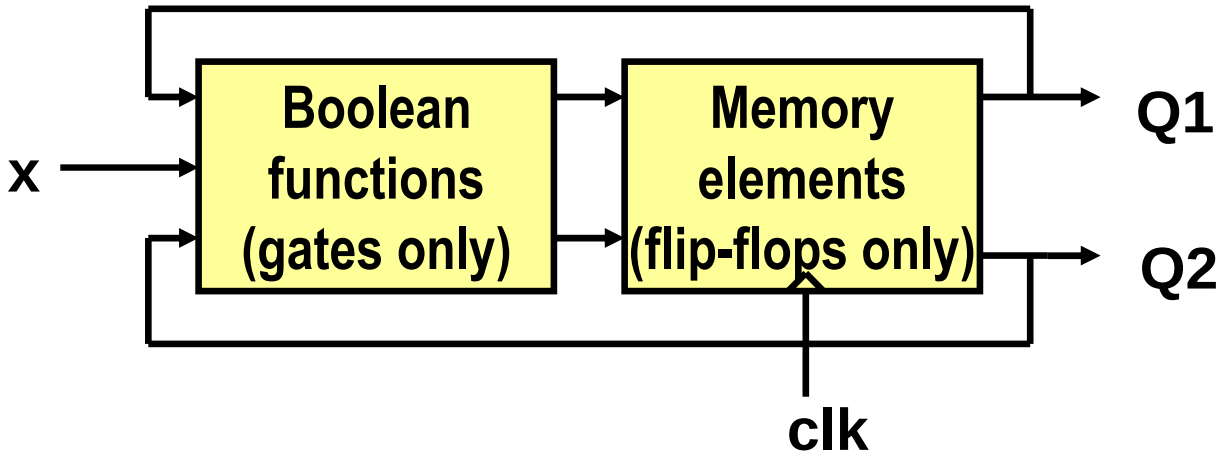


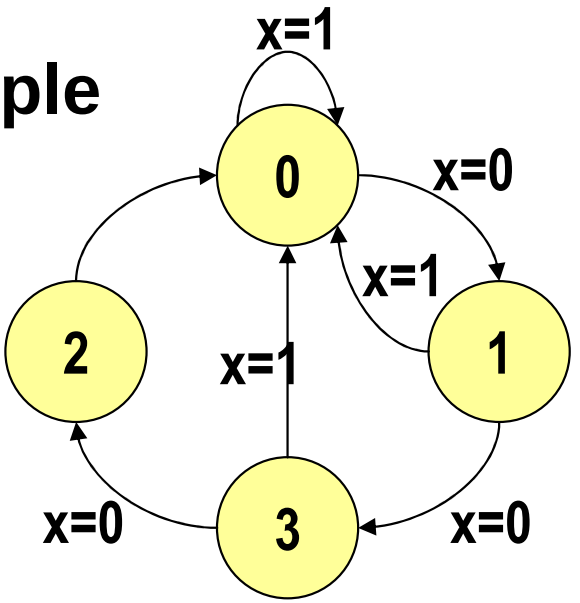
What the next state will be after the transition depends on

1) the current state (values of the memory element signals Y, F)

2) the values of input signals at the moment of the transition

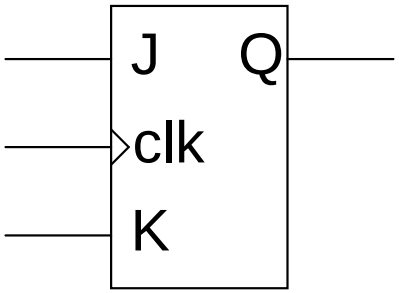
State Diagrams - example

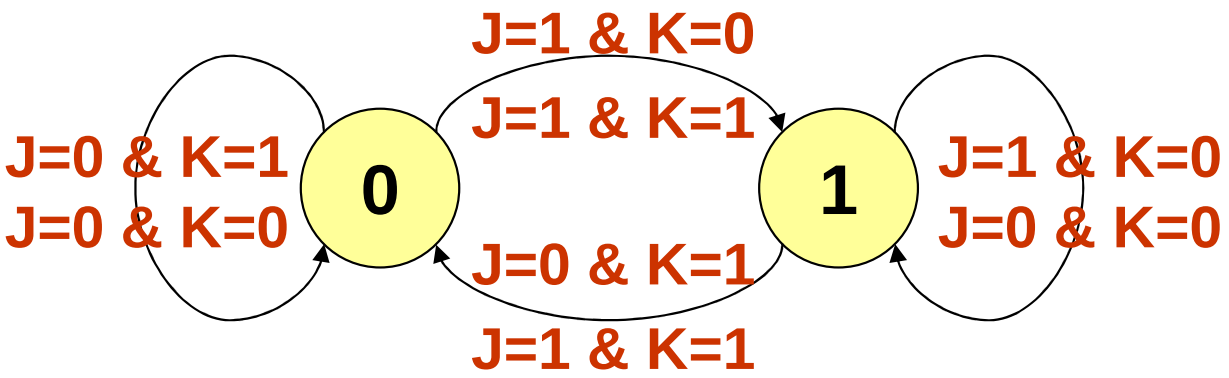




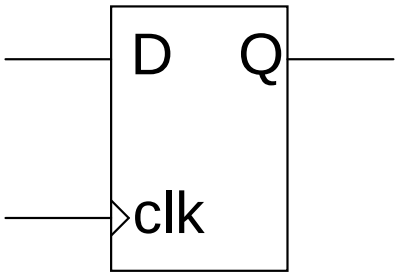
State Diagrams – memory elements

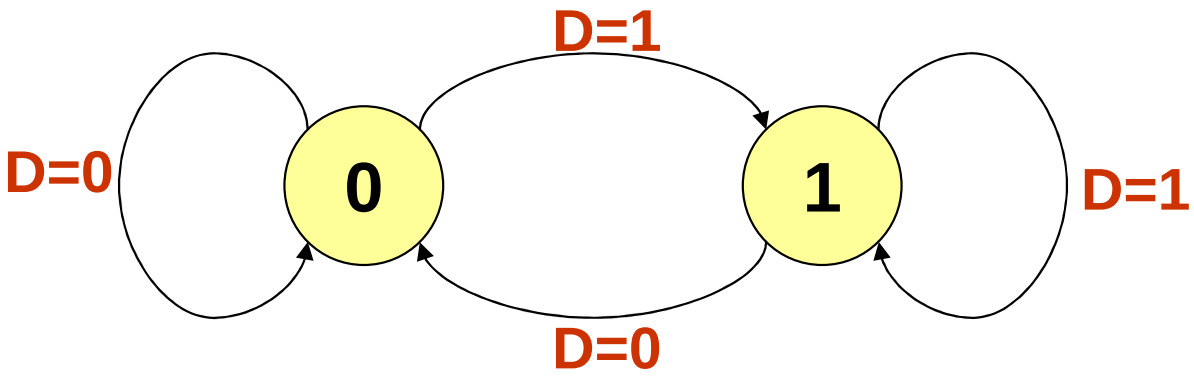
JK flip-flop



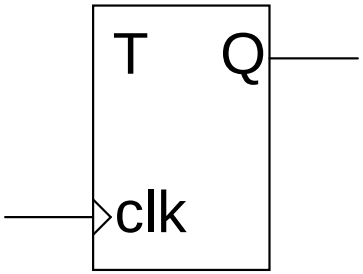


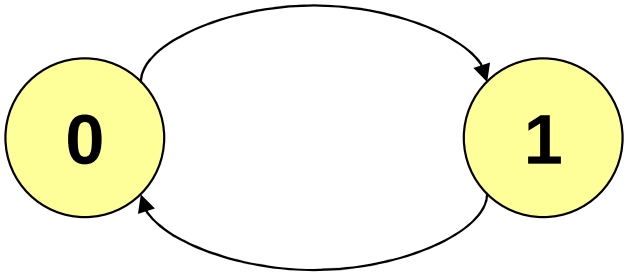
Data flip-flop



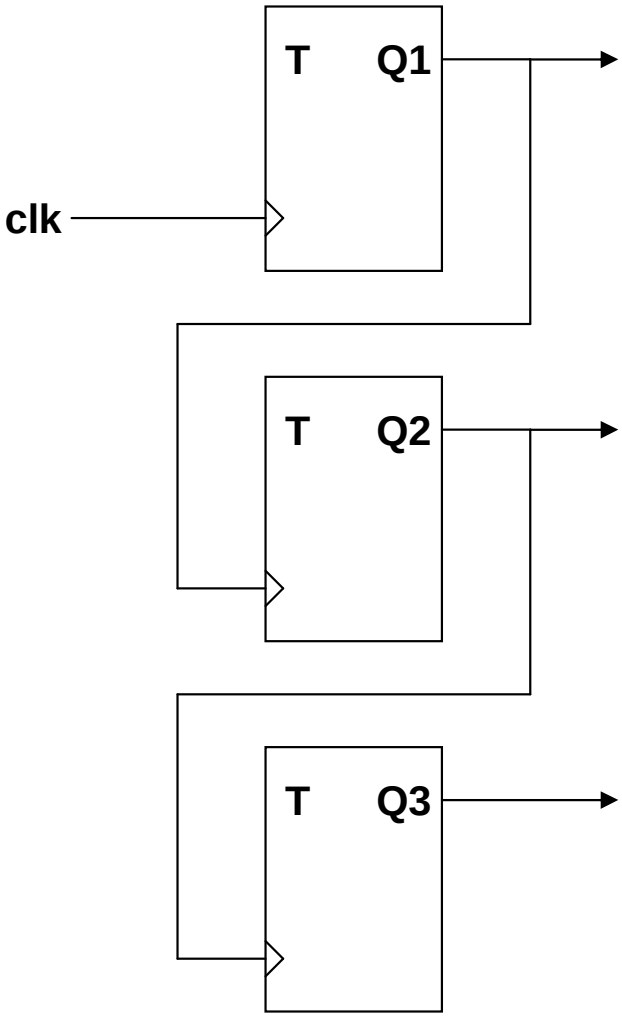


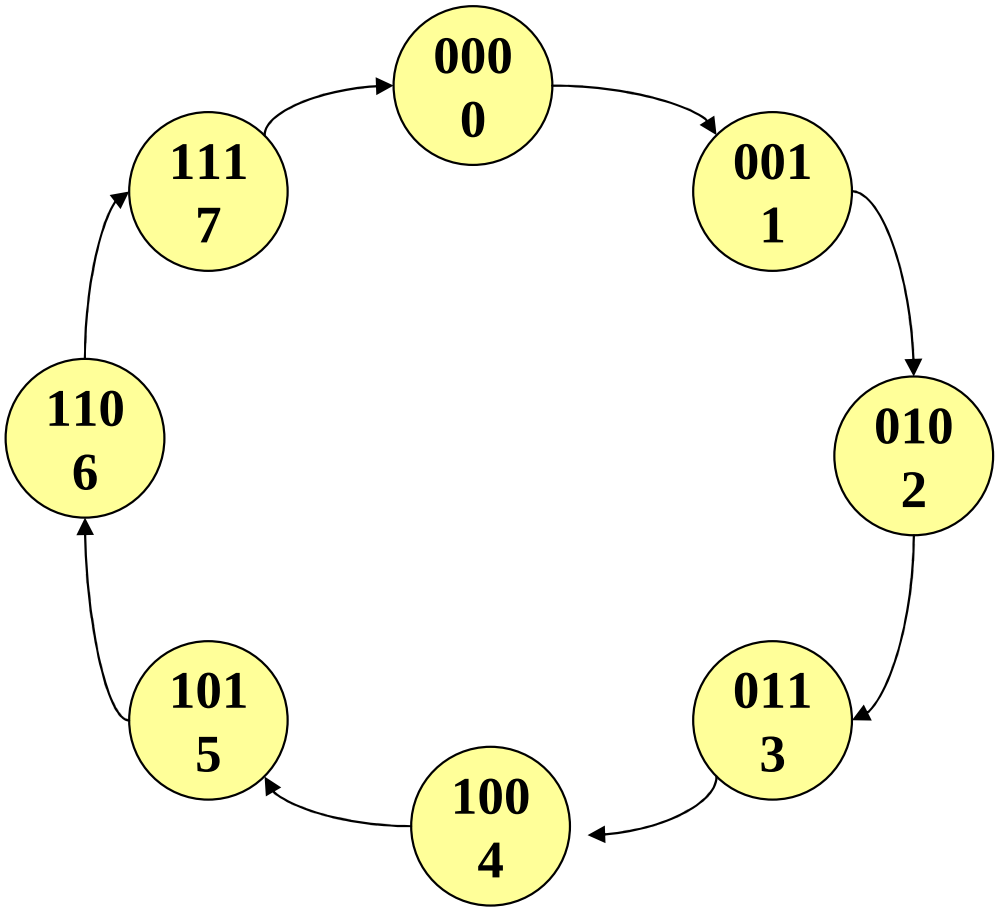
Toggle flip-flop

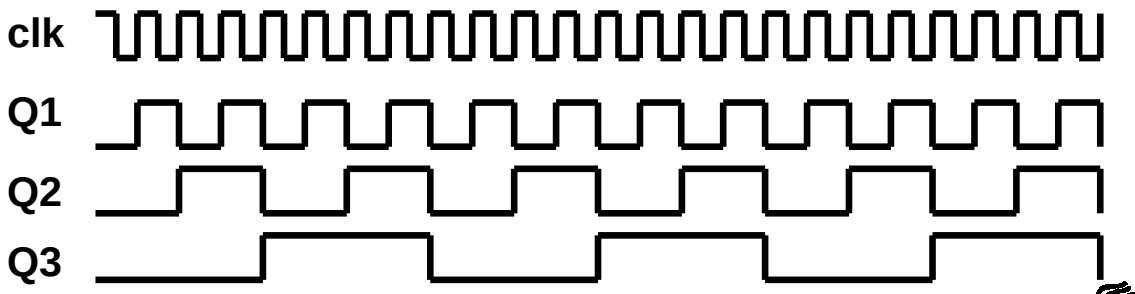




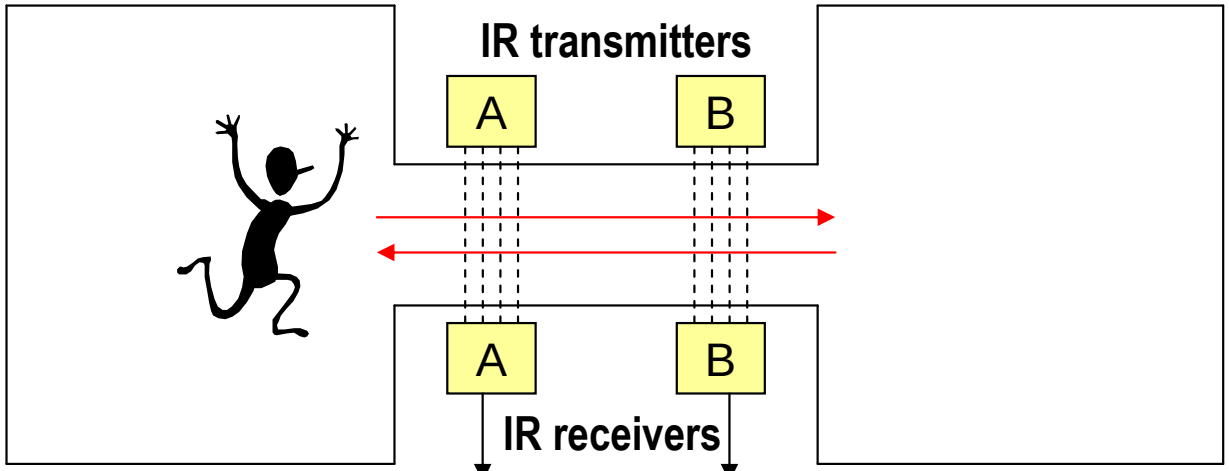
State Diagram – counter

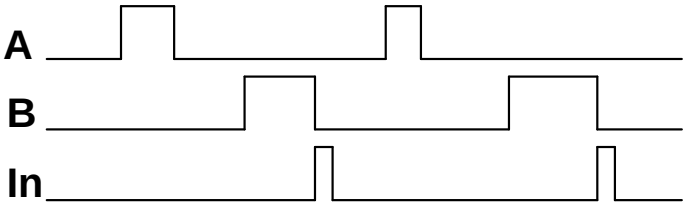


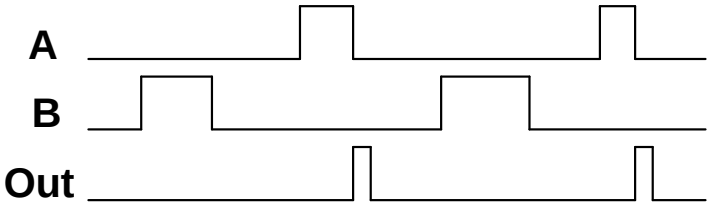


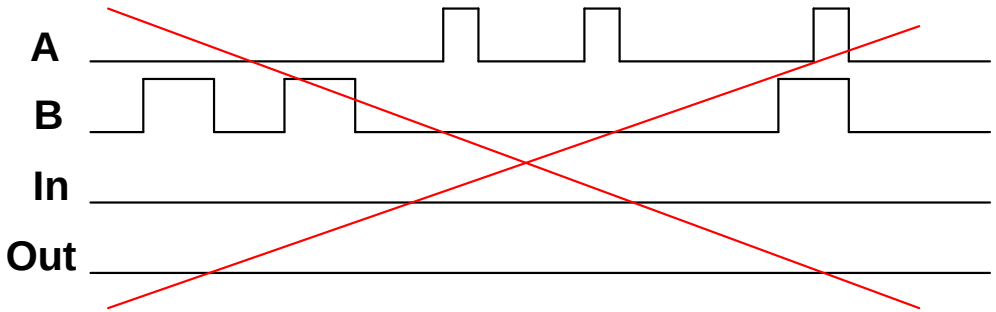


State Diagram example - bi-directional person counter

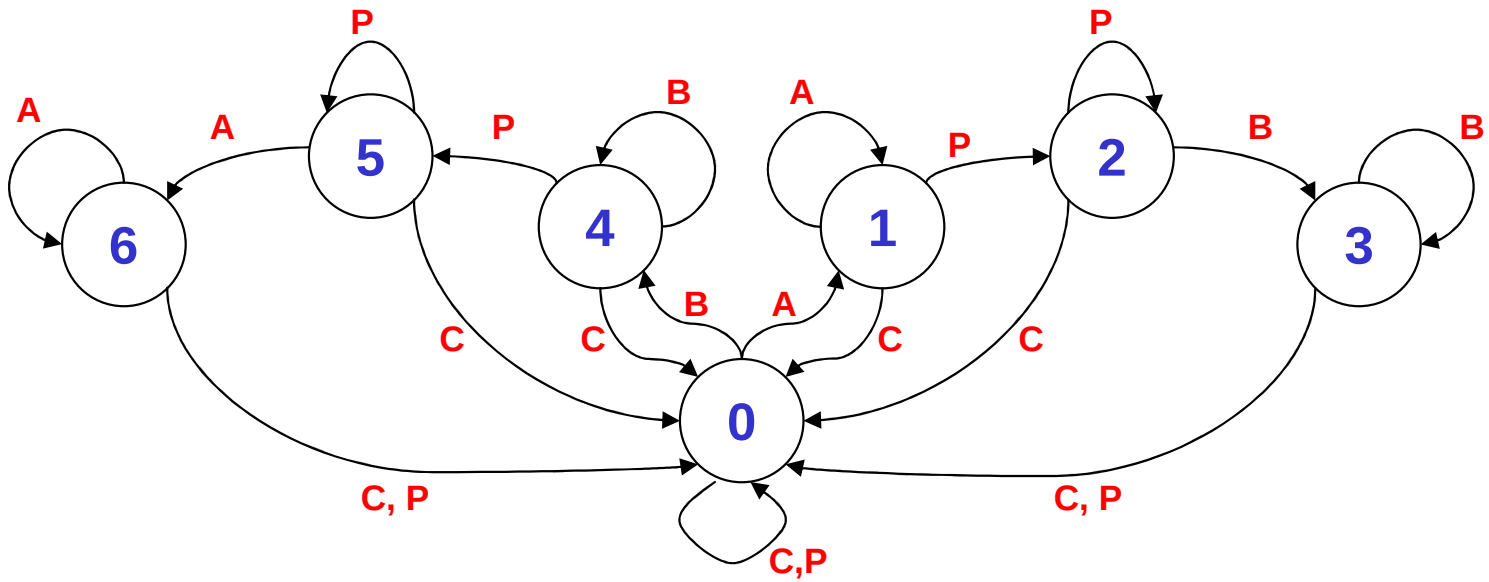




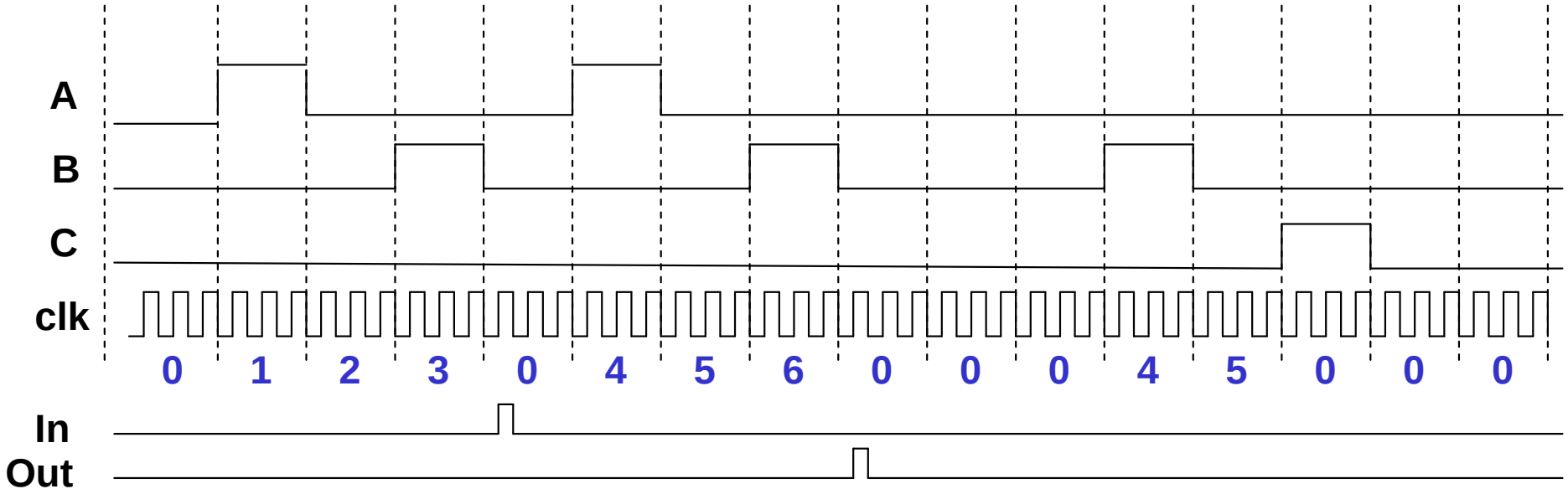




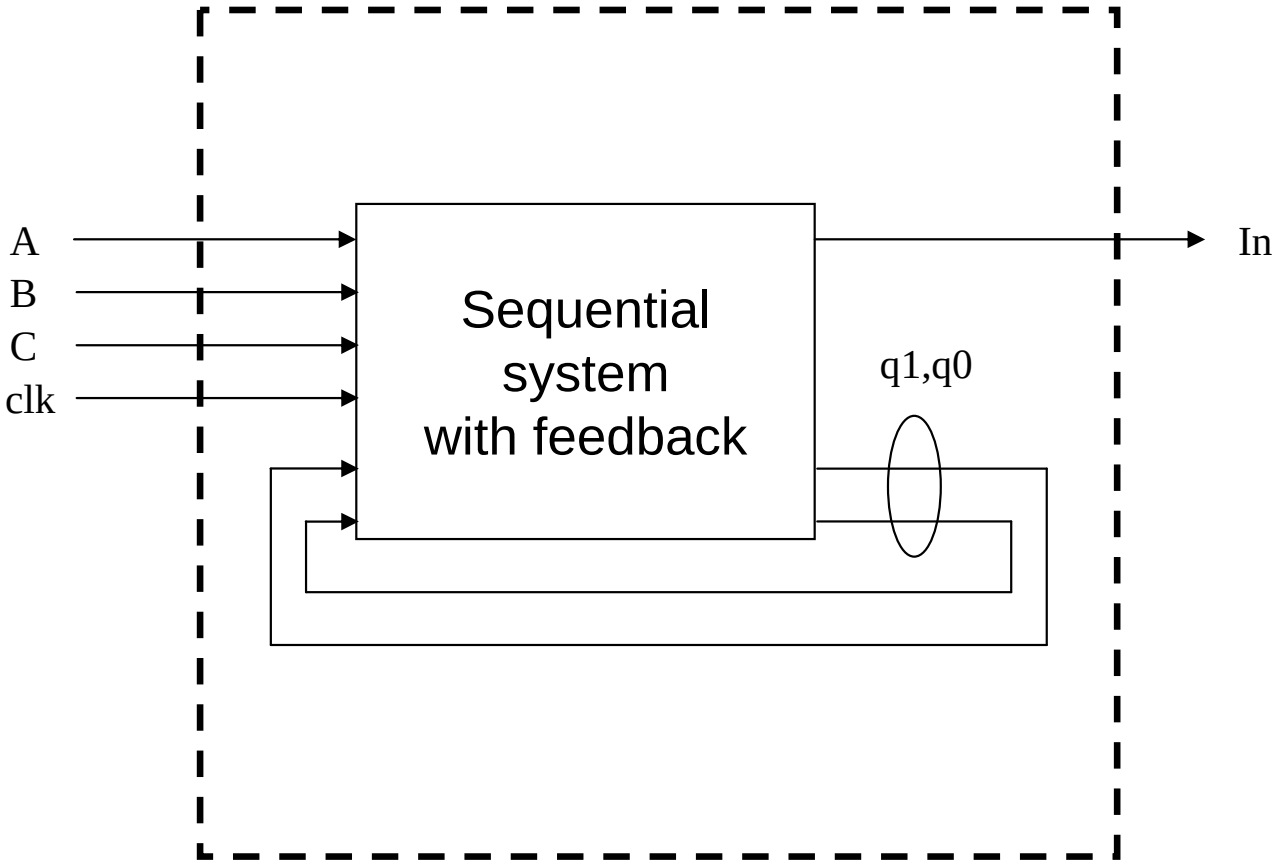
State Diagram example - bi-directional person counter



A,B,C
A = [1,0,0]
B = [0,1,0]
P = [0,0,0]
C = [x,x,1]



Realization - uni-directional person counter



Integrated circuit for person counting system

ABEL - bi-directional person counter

```
module pcounter;
title 'Bi-direction person counter'

        clk pin;  "clock signal
        A,B pin;  "signals from IR sensors
        C   pin;  "clear signal

        In_ pin istype 'reg';  "output line

        q1,q0  node istype 'reg';   "state counter

        c = .c.;
        x = .x.;

Equations
        [q1..q0].clk=clk;
        In_.clk=clk;

State_Diagram [q1..q0]
        state 0:if ((A==1) & (B==0) & (C==0)) then 1 with In_:=0
                else 0 with In_:=0;

        state 1:if (C==1) then 0
                else
                if ((A==1) & (B==0)) then 1
                else
                if ((A==0) & (B==0)) then 2;

        state 2:if (C==1) then 0
                else
                if ((A==0) & (B==0)) then 2
                else
                if ((A==0) & (B==1)) then 3;

        state 3:if ((A==0) & (B==1)) then 3
                else 0 with In_:=1;

end pcounter;
```

ABEL - bi-directional person counter

```
test_vectors ([clk,A,B,C] -> [q1,q0,In_])
        [  c,0,0,0] -> [ 0, 0,  0];
        [  c,1,0,0] -> [ 0, 1,  0];
        [  c,0,0,0] -> [ 1, 0,  0];
        [  c,0,1,0] -> [ 1, 1,  0];
        [  c,0,0,0] -> [ 0, 0,  1];
        [  c,0,0,0] -> [ 0, 0,  0];
        [  c,1,0,0] -> [ 0, 1,  0];
        [  c,0,0,0] -> [ 1, 0,  0];
        [  c,x,x,1] -> [ 0, 0,  0];
        [  c,0,0,0] -> [ 0, 0,  0];
        [  0,1,0,0] -> [ 0, 0,  0];
        [  c,1,0,0] -> [ 0, 1,  0];
```

Equations:

```
In_ := (!B & q1.FB & q0.FB
    # A & q1.FB & q0.FB);

In_.C = (clk);

q0 := (!A & B & q1.FB & q0.FB
    # !A & B & !C & q1.FB
    # A & !B & !C & !q1.FB);

q0.C = (clk);

q1 := (!A & B & q1.FB & q0.FB
    # !A & !C & q1.FB & !q0.FB
    # !A & !B & !C & !q1.FB & q0.FB);

q1.C = (clk);
```