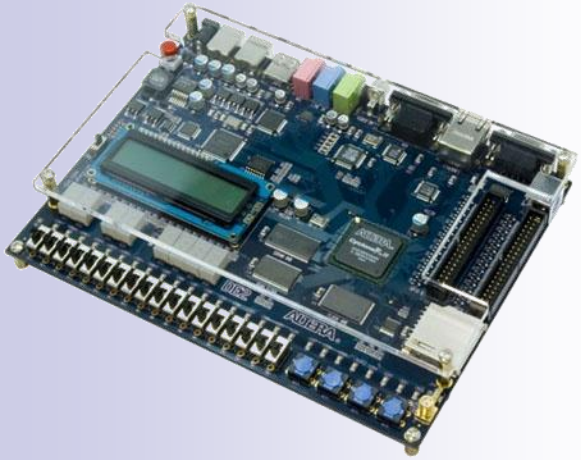
Logic Systems and Processors
cz:Logické systémy a procesory

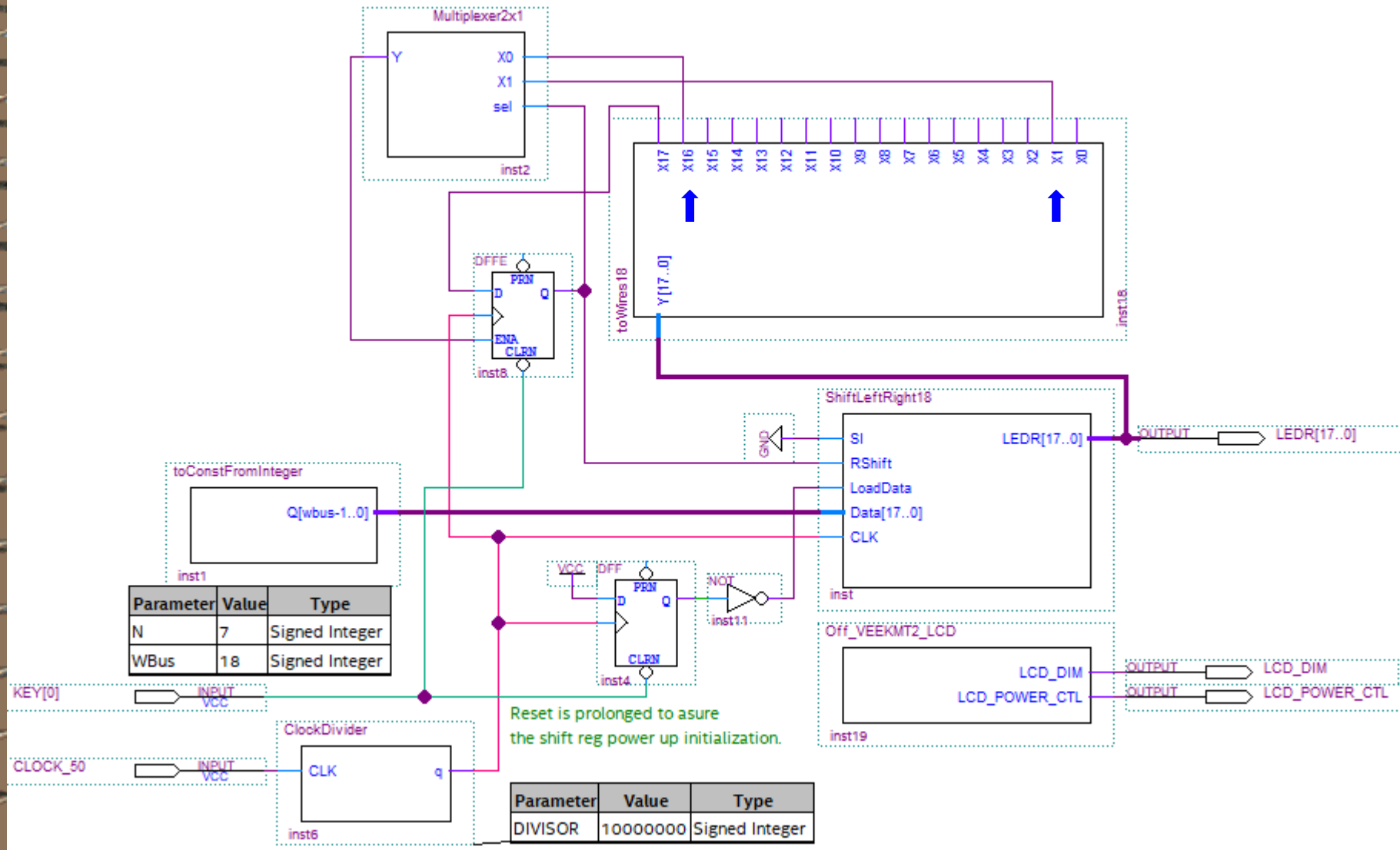


Lecturer: Richard Šusta

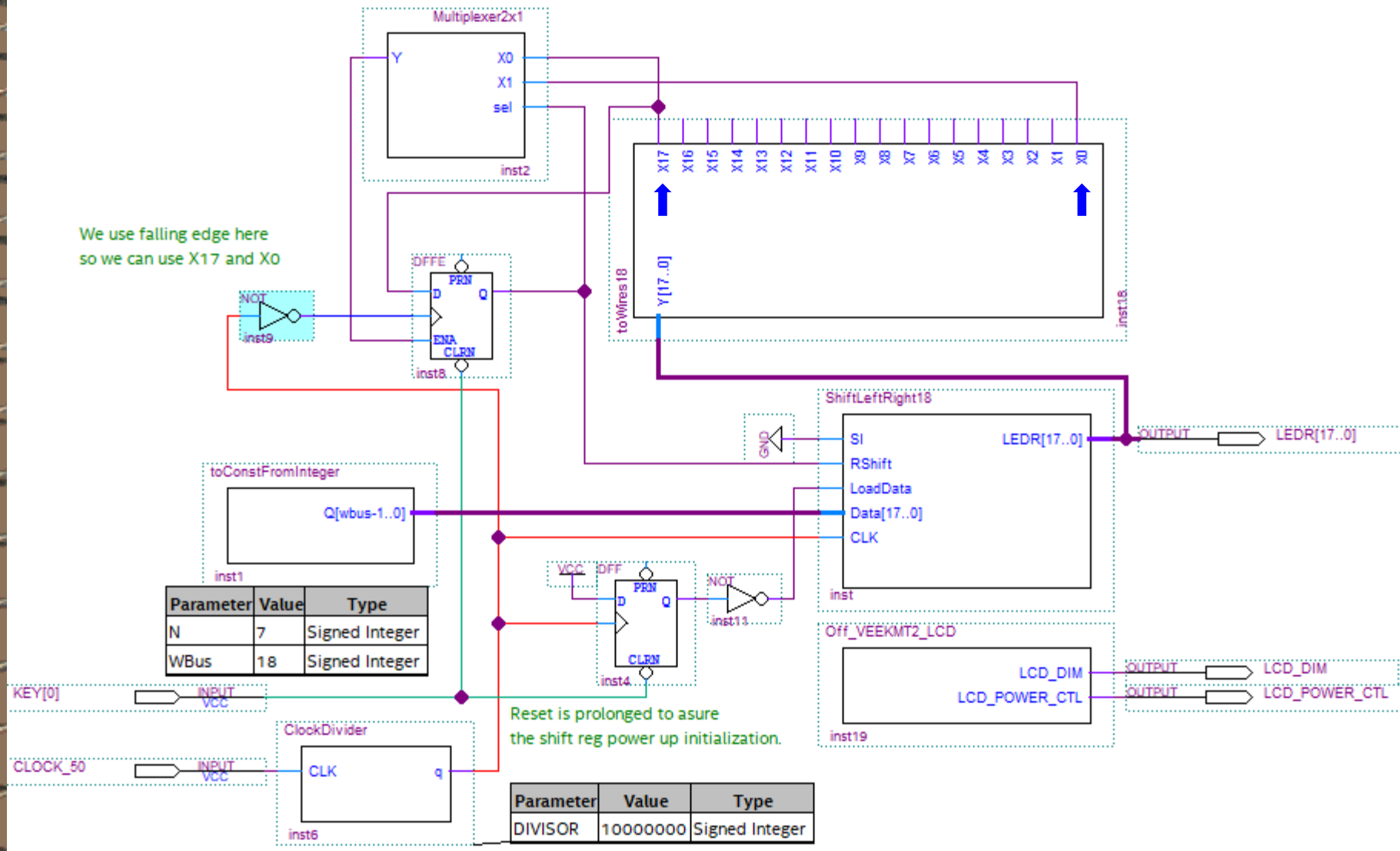
richard@susta.cz, susta@fel.cvut.cz,
+420 2 2435 **7359**

Version V1.0

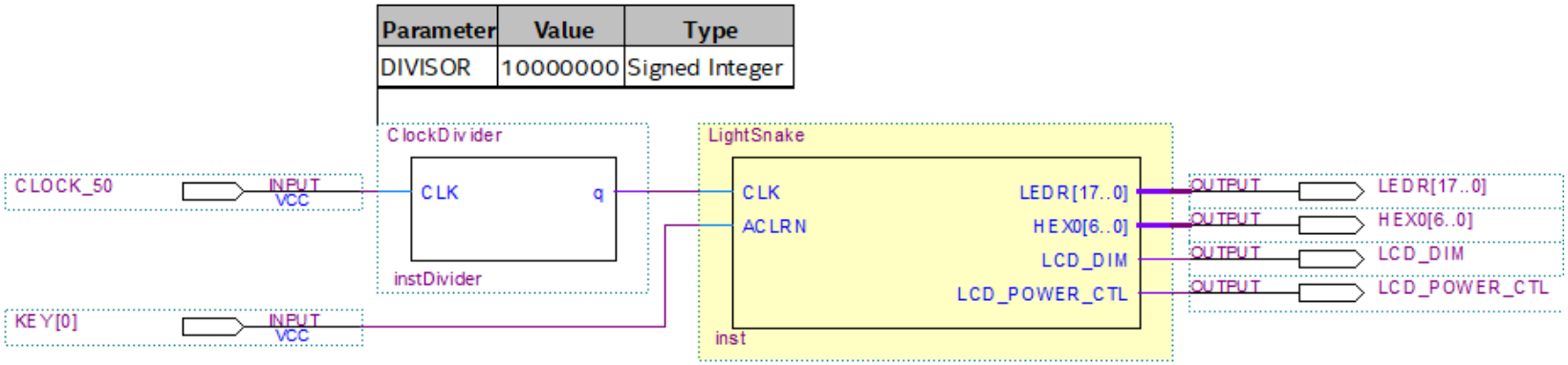
Example: The Light Snake in VHDL



The Light Snake in VHDL - rising/falling edge



BDF




```
entity  LightSnake is
   port( CLK, ACLRN :in std_logic:='0';  -- clock 5 Hz + asyn. clear=KEY[0]
         LEDR: out std_logic_vector(17 downto 0):=(others=>'0');
         HEX0: out std_logic_vector(6 downto 0):=(others=>'0');
         LCD_DIM, LCD_POWER_CTL : out std_logic:='0');
end entity;
```

```
library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all;
entity  LightSnake is
  port( CLK, ACLRN :in std_logic:='0';  -- clock 5 Hz + asyn. clear=KEY[0]
        LEDR: out std_logic_vector(17 downto 0):=(others=>'0');
        HEX0: out std_logic_vector(6 downto 0):=(others=>'0');
        LCD_DIM, LCD_POWER_CTL : out std_logic:='0');
end entity;
architecture rtl0116 of LightSnake is
begin -- architecture
    LCD_POWER_CTL<='0';  -- set LCD power off
    LCD_DIM <= '0';              -- set LCD backlight off
pshift: process(CLK, ACLRN) -- light snake
```

Here, we create process…

```
      end process;
end architecture;
```

Initialization

```
pshift: process(CLK, ACLRN) -- light snake
   variable rg : std_logic_vector(LEDR'RANGE):=(others=>'0');
   variable RUN, RShift : boolean:=false;
   begin
     if ACLRN='0' then RUN:=false; RShift:=false;
     elsif rising_edge(CLK) then
       if RShift then HEX0<="0100001";
                 else HEX0<="1100011";
       end if;
       if not RUN then
          rg:=(0 to 2=>'1', others=>'0');
          RUN:=true;
```

```
variable rg : std_logic_vector(LEDR'RANGE):=(others=>'0');
```

else *- if not RUN*

architecture rtl0116 **of** LightSnake

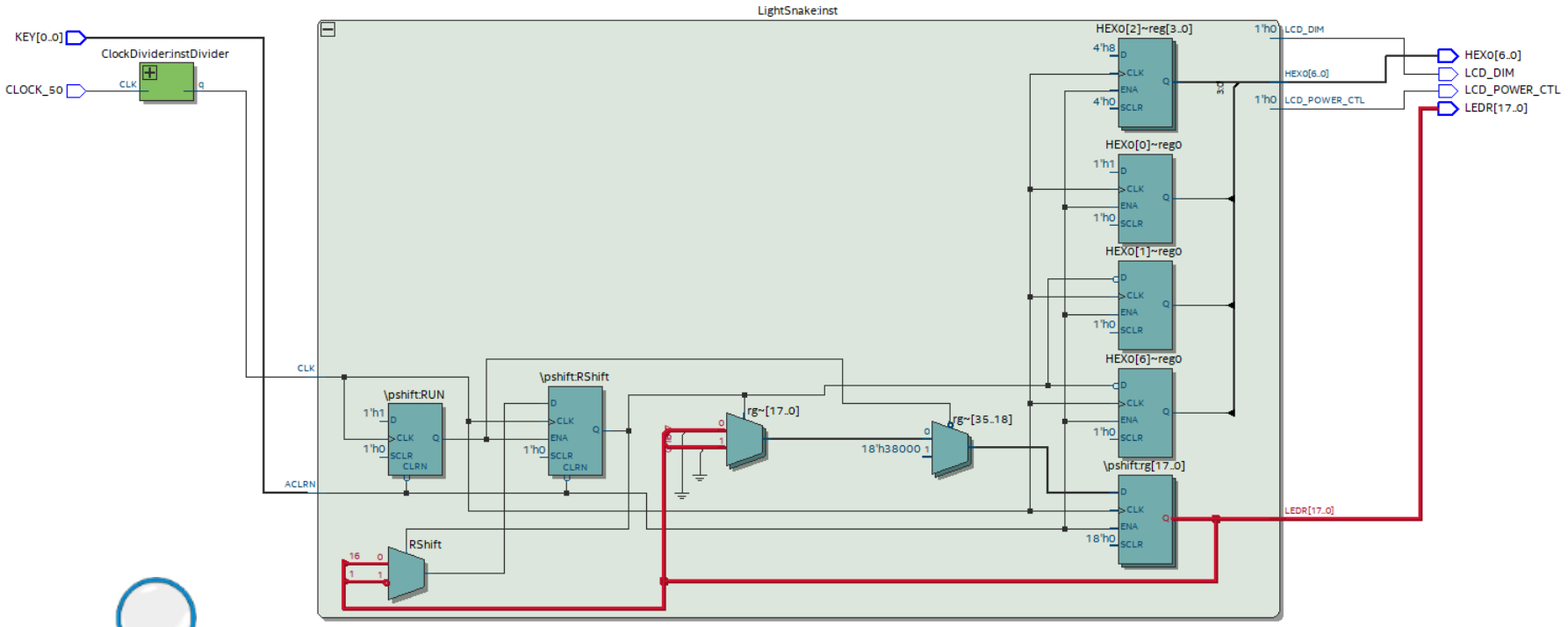
```
case RShift is
    when true=>  RShift:=rg(1)/='1';   rg := '0' & rg(17 downto 1);
    when false=> RShift:=rg(16)='1';   rg := rg(16 downto 0) & '0';
end case;
```

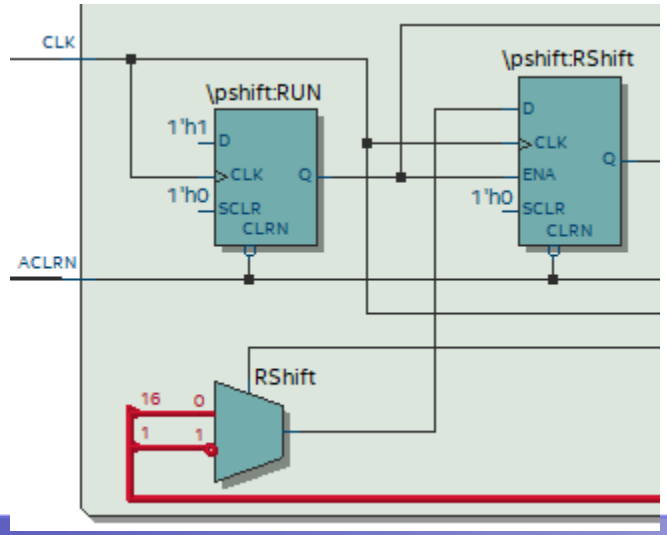
architecture rtl0017 **of** LightSnake

```
case RShift is
    when true=>  rg := '0' & rg(17 downto 1);   RShift:=rg(0)/='1';
    when false=> rg := rg(16 downto 0) & '0';   RShift:=rg(17)='1';
end case;
```

```
    end if; -- if not RUN
  end if;
  LEDR<=rg;
end process;
```

RTL Viewer





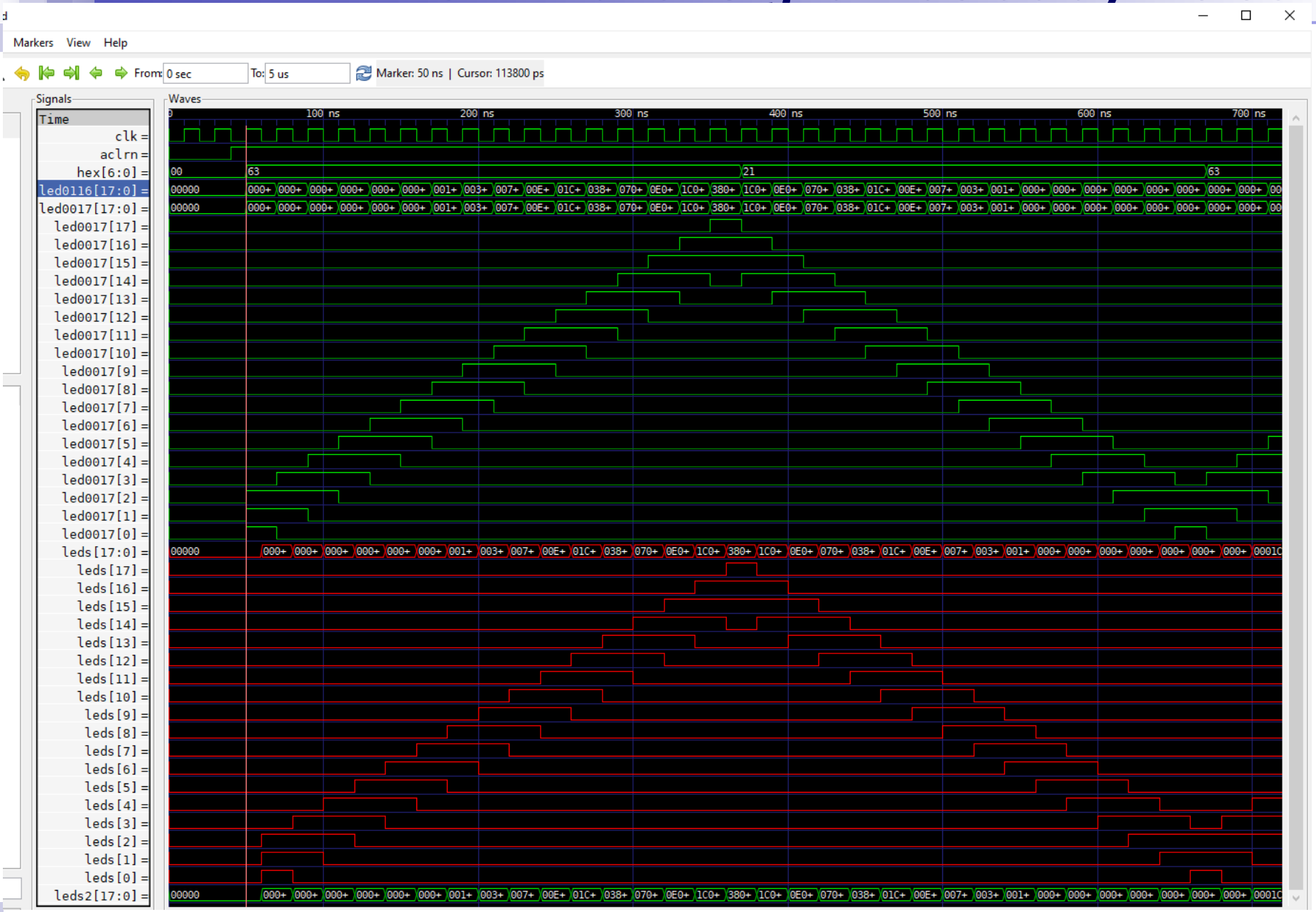
Both variants create the same implementation:-)

Internal Register as a Signal

```
architecture rtlsignal of LightSnake is
signal rg : std_logic_vector(LEDR'RANGE):=(others=>'0');
begin -- architecture
 /* ..................
    begin -- process
     if ACLRN='0' then RUN:=false; RShift:=false;
      elsif rising_edge(CLK) then
       if RShift then HEX0<="0100001"; else HEX0<="1100011"; end if;
        if not RUN then
           rg <= (0 to 2=>'1', others=>'0');  RUN:=true;
        else
         case RShift is
          when true=>  rg <= '0' & rg(17 downto 1); RShift:=rg(1)/='1';
          when false=> rg <= rg(16 downto 0) & '0'; RShift:=rg(16)='1';
         end case;
        end if;
     end if;
    LEDR<=rg;
end process;
```

```
-- The reverse order is also correct
when true=>  RShift:=rg(1)/='1';  rg <= '0' & rg(17 downto 1);
when false=> RShift:=rg(16)='1'; rg <= rg(16 downto 0) & '0';
```

GTKWave: Signal has delay 1 clock



The different behavior follows from

Assignments

blocking :=

versus **non-blocking <=**

We have blocking and non-blocking assignments in the sequential domain

- **Variable assignments (blocking, immediate)**

  variable := *expression;*

- **Signal assignments (non-blocking, delta-cycle)**

  signal <= *expression*;
  signal <= *expression* **after** *delay*;
  signal <= **transport** expression **after** delay;
  signal <= **reject** time **inertial** expression **after** delay;

C language statements are of the blocking type.

By using threads in it we emulate non-blocking mode.

The circuit operates primarily in non-blocking mode.

Transport Delay

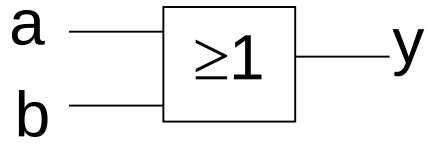
signal <= **transport** expression **after** delay;

The model corresponds to an ordinary wire:

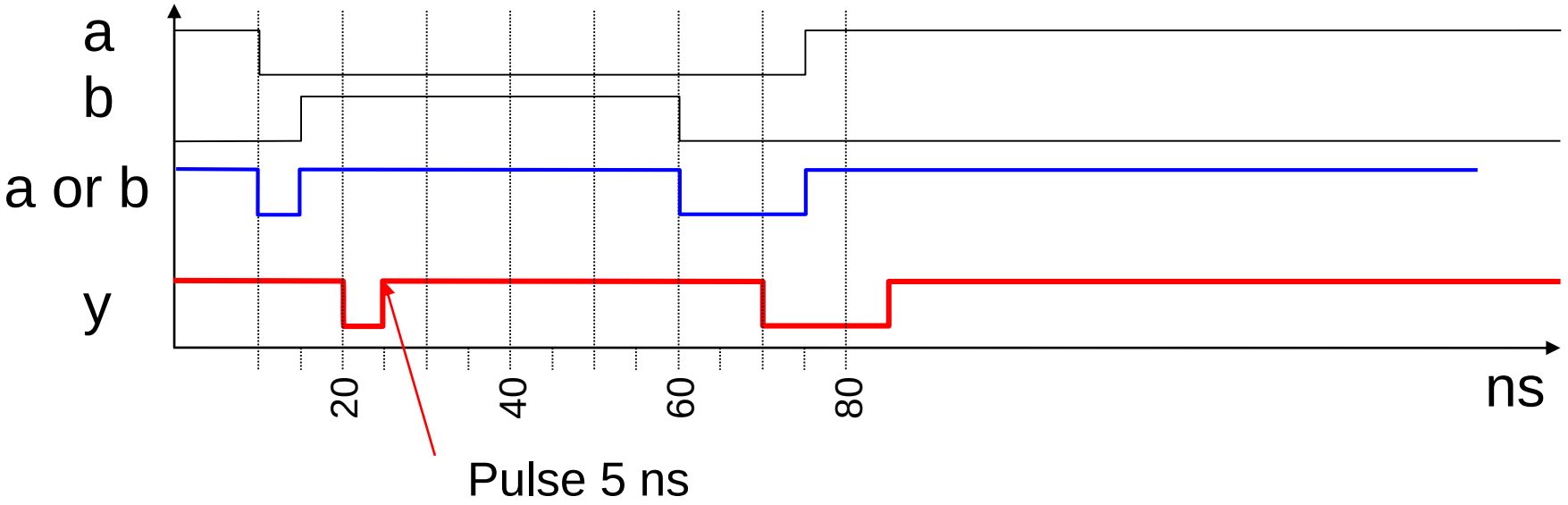
The pulses will pass through regardless of their length - it's just a time delay describing a wire (conductor), but not the behavior of the circuit.

Example:

c <= **transport** a or b **after** 10 ns;



OR gate



Inertial Delay

- The minimum pulse width required to change the gate state is called the inertial delay of the gate.
- Inertial delay means suppressing all "spikes" shorter than the delay time. By default, inertial delay is assumed as the default delay for gate modeling.
  - The gate output only changes when the input pulse has a certain minimum length, i.e., it remains unchanged for a certain period of time = hold time.
  - If the pulse is shorter, then the gate will not change its state.

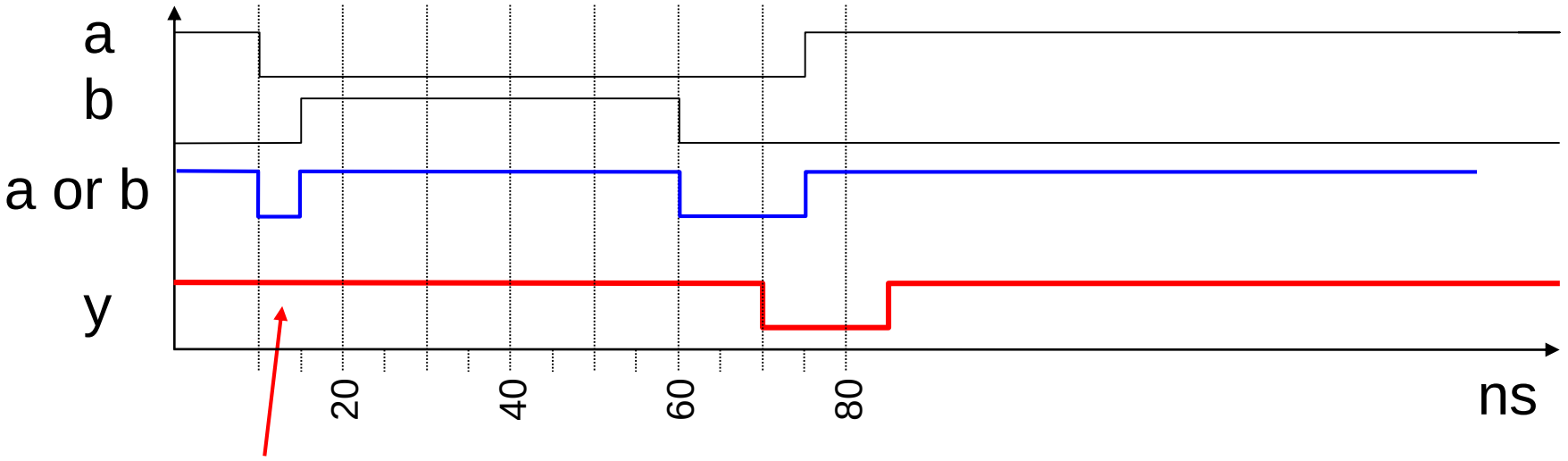
Example

c <= a or b **after** 10 ns;

c <= INERTIAL a or b **after** 10 ns;

c <= REJECT 10 ns INERTIAL a or b **after** 10 ns;

Equivalent commands

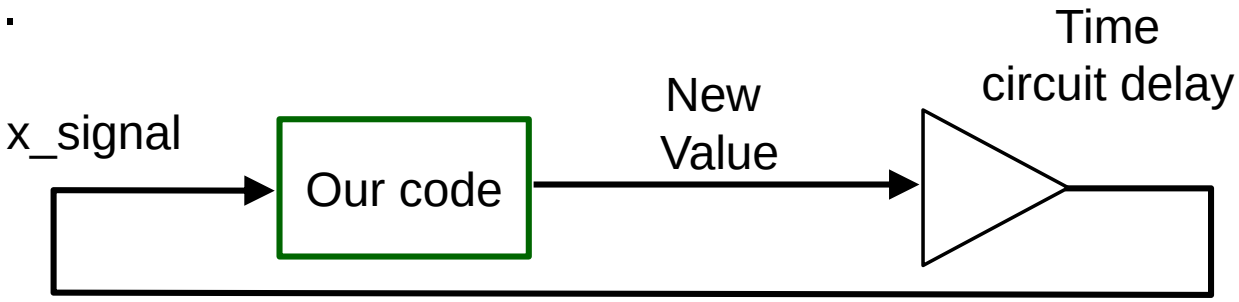


5 ns pulse is suppressed

- VHDL Reference:

  In contrast to variables, the values assigned to signals can be read back only after at least a delta cycle (δ) time, an infinitesimally small advance in time, i.e, in reality, a signal is updated during the next evaluation of the process.

- The situation can be imagined as a circuit where signals (wires) contain input values and in the code we assign output values to these signals.



- *If the signal is used only within a process and is not read by another process, then it can be replaced by a variable. Variables have local validity (scope), which is generally advantageous for both clarity and code safety.*
- *Remember that signals are the only reasonable way to **connect multiple processes or circuits**, i.e. to pass values from one process or circuit to another element.*

VHDL time

Image: https://commons.wikimedia.org

Units of time

Unit	Definition
Base Unit	
fs	femtoseconds (10^{-15} seconds)
Derived Units	
ps	picoseconds (10^{-12} seconds)
ns	nanoseconds (10^{-9} seconds)
us	microseconds (10^{-6} seconds)
ms	miliseconds (10^{-3} seconds)
sec	seconds
min	minutes (60 seconds)
hr	hours (3600 seconds)

Physical data types

- TIME is the only predefined physical data type.
  Note: Types representing physical quantities such as time, voltage, capacitance, etc., are referred to as physical data types in VHDL.
- The value of a physical data type is called a physical literal and can be an integer or a floating-point number.
- The numeric value and dimension MUST be separated by a space. The numeric time value is optional. If not specified, 1 is assumed.
- The smallest resolution available in VHDL is 1 fs.

Examples

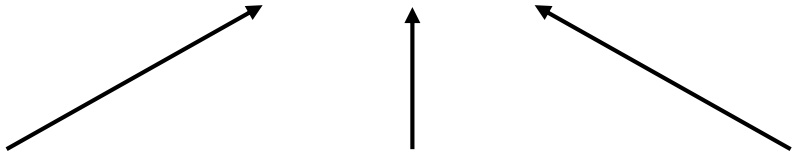
7 ns

1 min

min

12.65 us

978 fs



Value

Mandatory space

Unit (dimension)

Arithmetic Operations with TIME

Examples:

7 ns + 10 ns = 17 ns

1.2 ns - 12.6 ps = 1187400 fs

5 ns * 4.3 = 21.5 ns

20 ns / 5 ns = 4

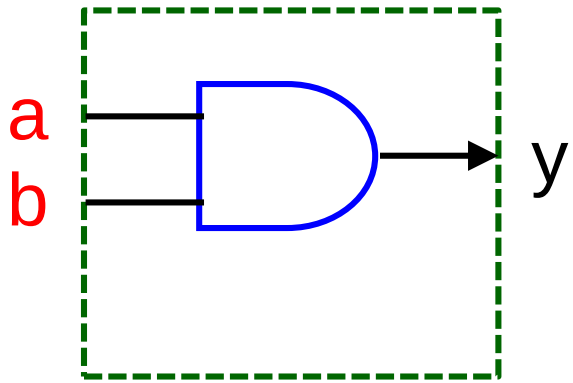

Process with waiting
In event of lift
avoid use of fire
FOCAL SIGNS 0208 687 5300 REF.PS179
DESTINY_3000
WWW.DEVIANTART.COM

"Sensitivity list" shortcut for "wait on!

```
process(a,b)
begin

-- some statements, e.g.
y<=a and b;

end process;
```

⟺

```
process
begin

-- some statements, e.g.
y<=a and b;

wait on a, b;
end process;
```



Meaning of

If the value of a or b does not change of the input, the value of y

← behavior of the combinational circuits

- *Building a circuit from code is not an easy task, so synthesis tools only allow a subset of VHDL commands.*
- ***Quartus requires a sensitivity list here*** *- however, note that the sensitivity list is not analogous to the function parameters, but a different notation for waiting for signals to change, and a clearer relocation of the* wait on *command signals to the beginning of the process.*

```
process(a,b)
begin

-- some statements, e.g.
y<=a and b;

end process;
```

```
process
begin

-- some statements, e.g.
y<=a and b;

wait on a, b;
end process;
```

- *Quartus calculates the incomplete sensitivity list and prints a warning. Correct!*
- *The sensitivity sheet defines the simulation activity. If it is incomplete, it is wrong.*
- *VHDL 2008 allows the sensitivity list: process(all), i.e., all signal in design.*

VHDL Wait statement in simulation

Four wait **commands** can be used in the simulation:

- **wait on** *signal list*;
  - wait for the signal to change;
    e.g.: **wait on** a;
- **wait until** *condition*;
  - wait until the condition is met;
    e.g.: **wait until** c='1';
- **wait for** *duration*;
  - wait for the allotted time;
    e.g.: **wait for** 10 ns;
- **wait;**
  - wait an infinite amount of time.

Delta Delay

- *If we write a command*

  output <= '0';

  will then be processed internally as

  output <= '0' after δ;

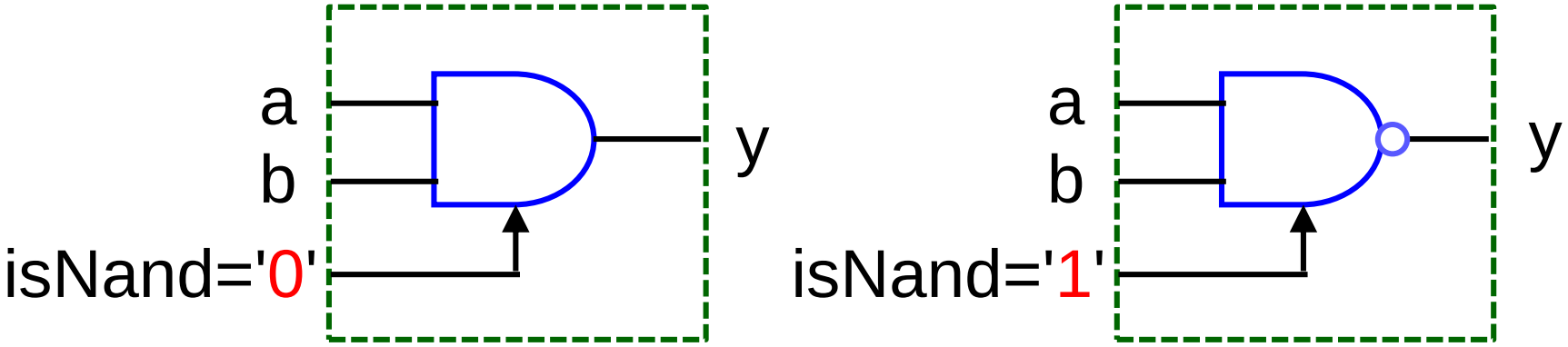
After must not be used in the synthesis code
- exact time delays cannot be implemented,
However, the compiler will compile all "<=" assignments as equivalents with after δ; where delta represents an infinitesimal delay.
Note: δ represents an imaginary internal symbol of the compiler; it cannot be written into the program.

Example

Pat and Mat designing with variables and signals

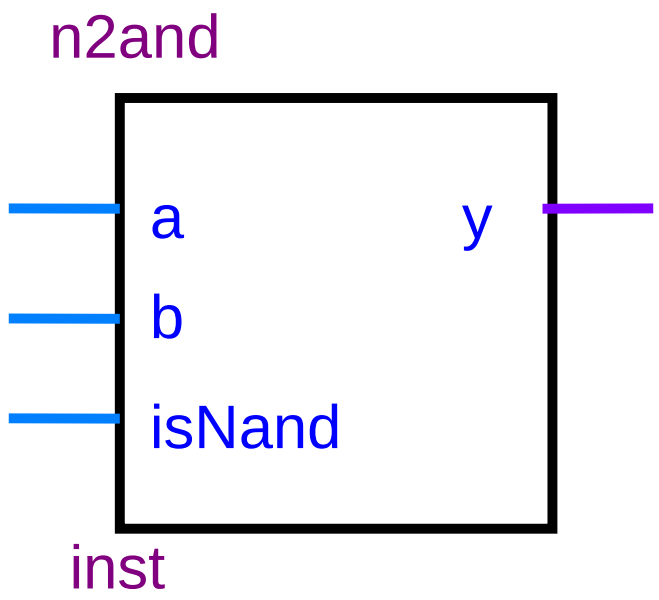
Image credit: http://en.patmat.cz/

- *Pat and Mat build ANDs that can be switched to NAND by isNand input.*

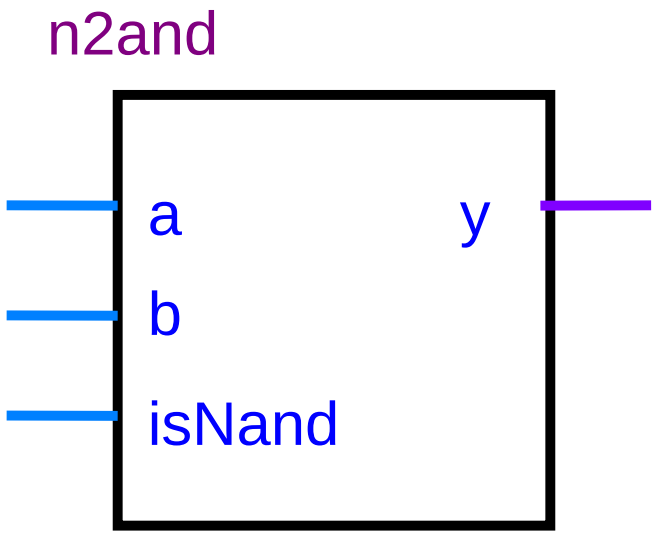


Entity remains the same in all Pat-Mat codes

```
library ieee; use ieee.std_logic_1164.all;

entity n2and is

  port (a, b, isNand: in std_logic;

        y : out std_logic);

 end entity;
```



```
architecture dataflow of n2and is
begin
   y <= a and b when isNand='0'
                else not (a and b);
end architecture;
```



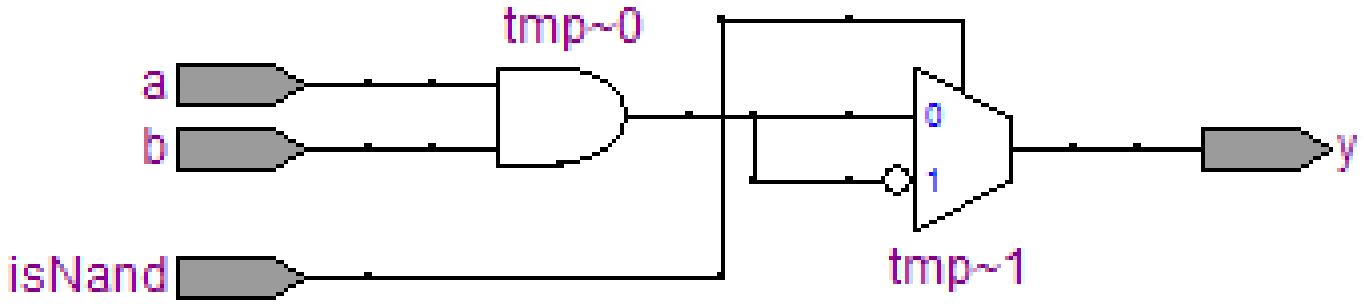
- *but the PAT student thought he'd try the VHDL process...*

Image: http://en.patmat.cz/

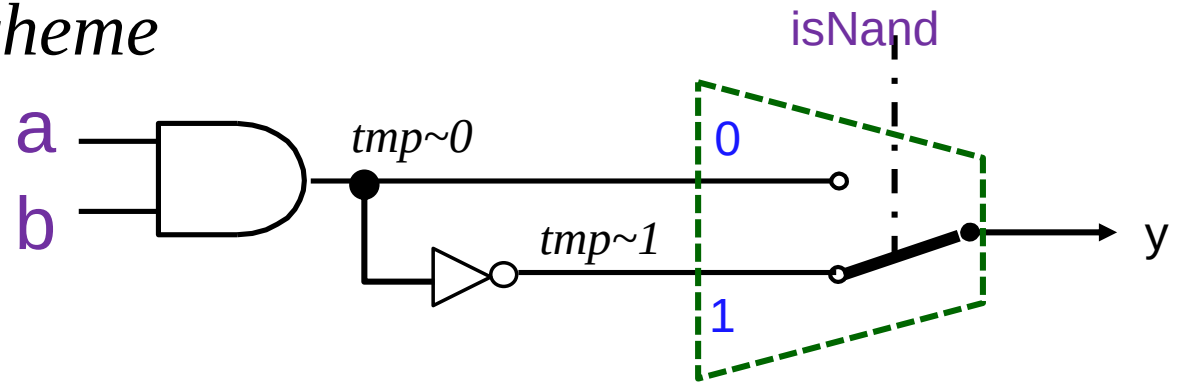
P1-solution with variable

```
architecture dataflow of n2and is
begin
    process(a,b, isNand)
    variable tmp : std_logic;
    begin
      tmp:=a and b;
      if isNand='1' then tmp:=not tmp; end if;
      y<=tmp;
    end process;
end architecture;
```

Result of the compilation

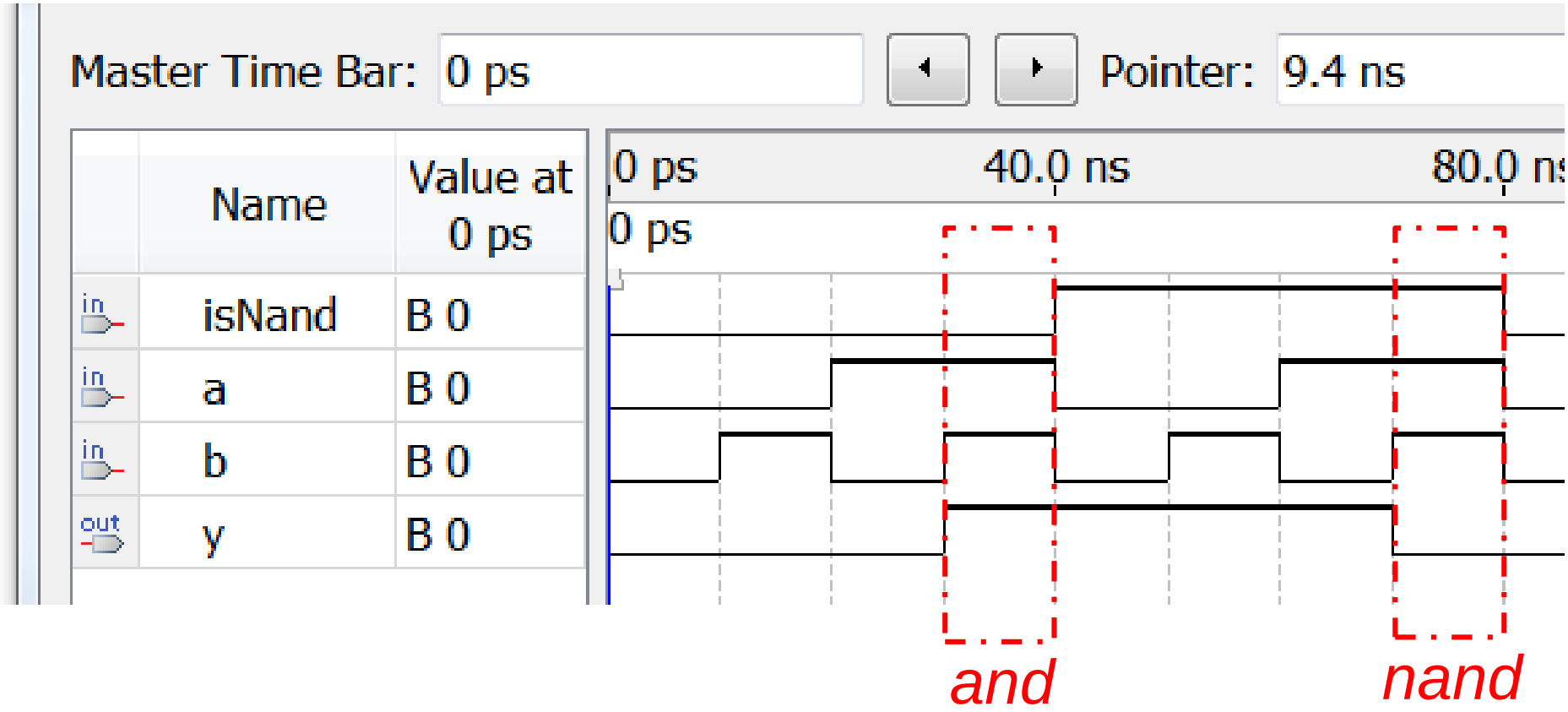


Principal scheme



Comment: Quartus converted multiple variable assignments to unique ones by introducing the auxiliary internal variables tmp~0 and tmp~1.
Addendum: the ~ character is reserved for compiler-generated names only, it cannot be used in code.

Functional simulation result



Mat didn't like Pat's solution,

he changed the variable to signal.

A signal, *that sounds better than some* variable :-)

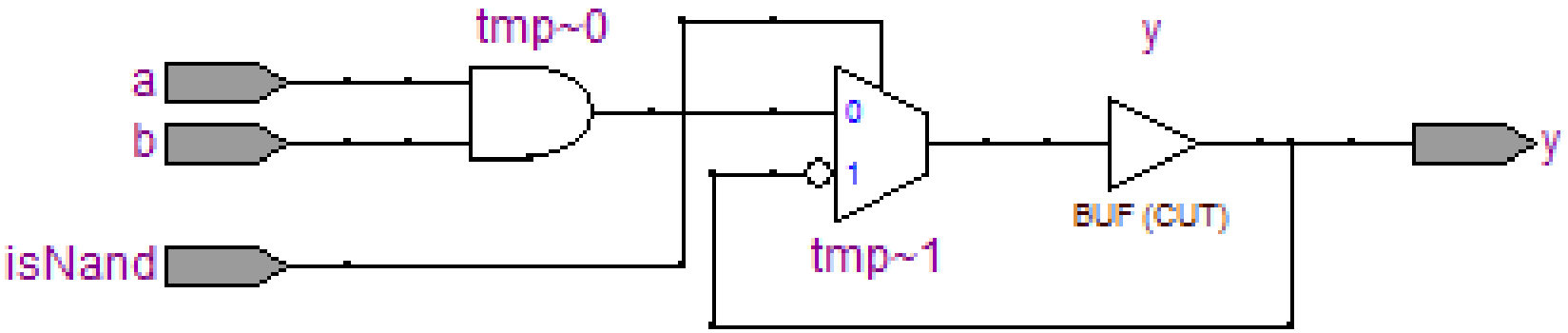
Image: http://en.patmat.cz/

Wrong solution - the circuit got MAT

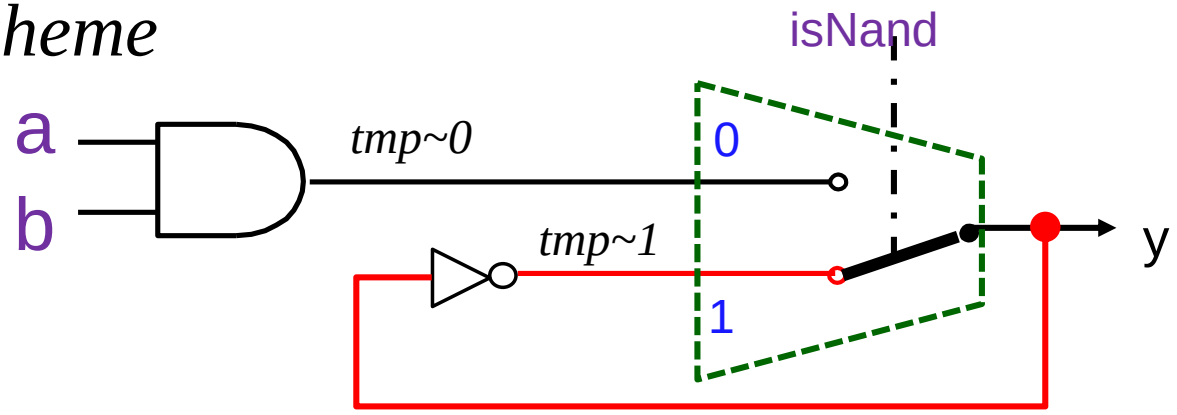
```
architecture dataflow of n2and is
signal tmp : std_logic;
begin
    process(a,b, isNand)
    begin
      tmp<=a and b;
      if isNand='1' then tmp<=not tmp; end if;
      y<=tmp;
     end process;

end architecture;
```

Result of stalemate build

*Warning (332125): Found **combinational loop** of 2 nodes*

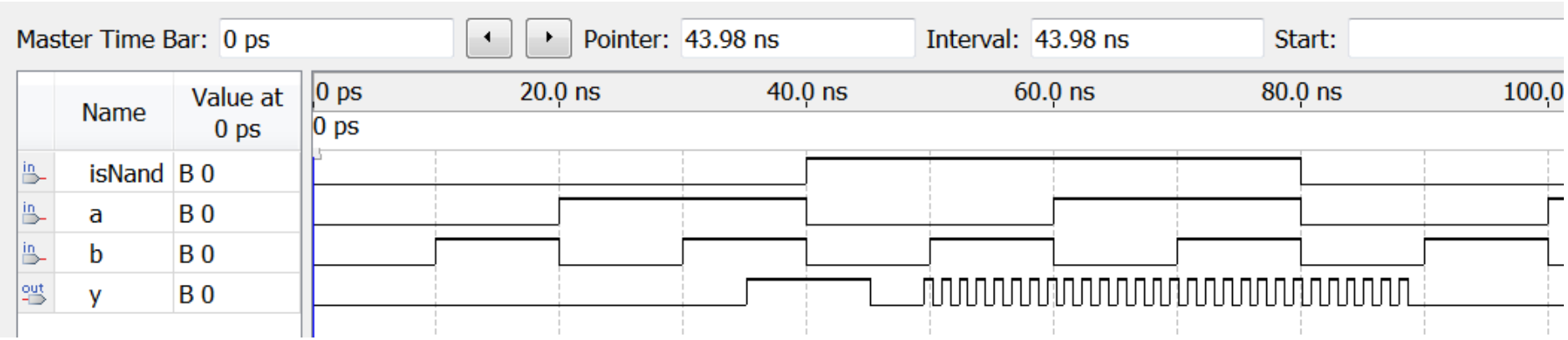


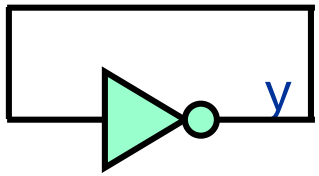
Principal scheme

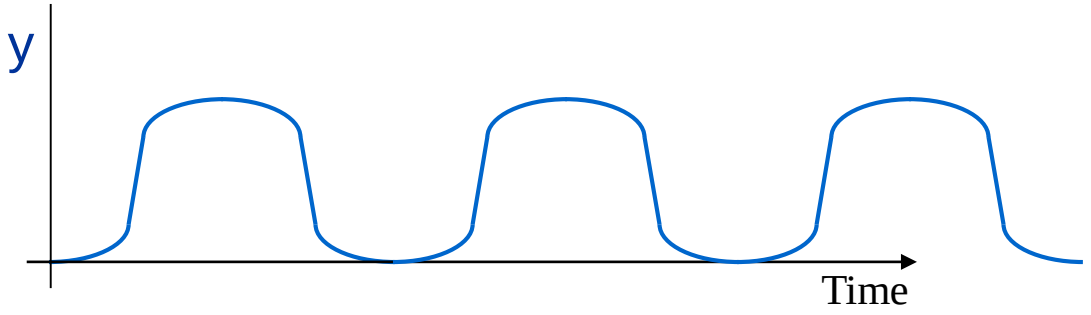


"Functional simulation" fails -> buffer overflow

"Timing-simulation considering gate delays will show a result indicating oscillations







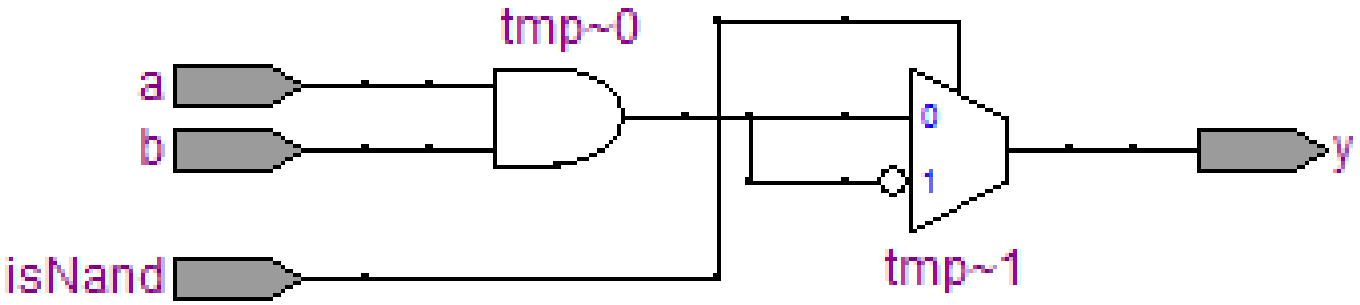
Note: GHDL cannot simulate because it only does functional simulations.

Pat finally made a correction

```
architecture dataflow of n2and is
signal tmp : std_logic;
begin
    process(a,b, isNand)
    begin
        tmp<=a and b;
        if isNand='1' then tmp<=not (a and b); end if;
        y<=tmp;
    end process;
end architecture;
```

The P-code no longer uses the values assigned to the tmp signal inside the process - its new value will appear only next time.

The circuit is again correct



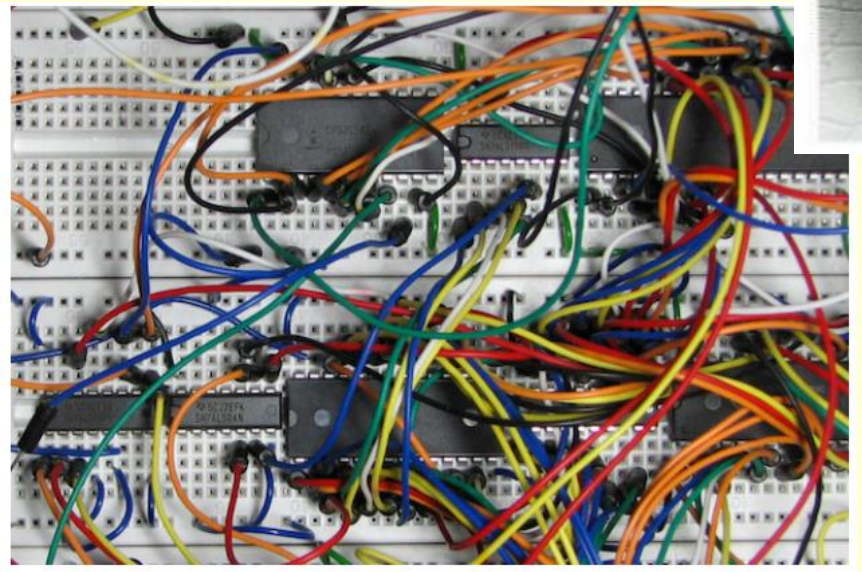
How would n2and be written by others?

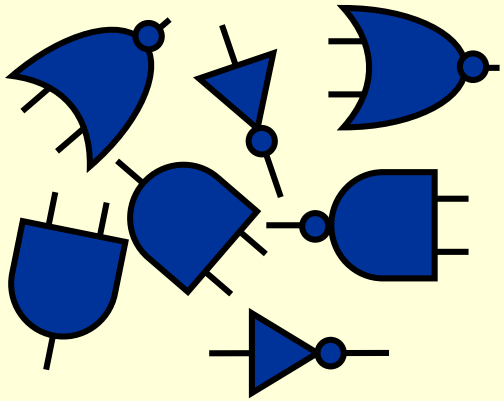
```
architecture profi of n2and is -- without process
begin
      y<= (a and b) xor isNand; -- xor as a controlled inverter
end architecture;
```

Inside process, we can use if then else.

```
architecture proc of n2and is
begin
   process(a, b, isNand)
   begin
      if isNand='1' then y<=not (a and b);
                    else y<= a and b;
      end if;
   end process;
end architecture;
```

Causes for Gates Delays are Hidden Inside Chips



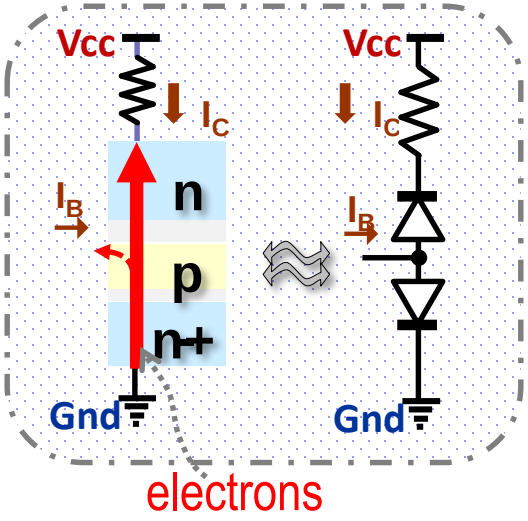


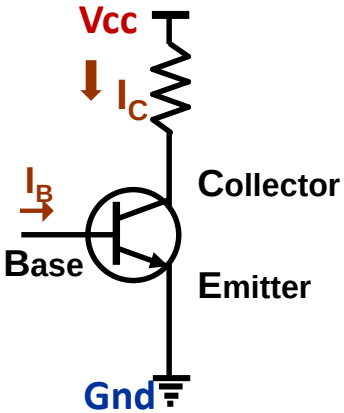
[Pictures sources: radio411.com, freepik.com, apple.com]

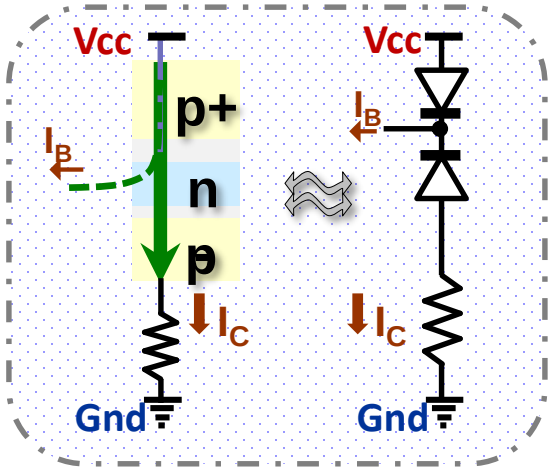
Bipolar Transistors

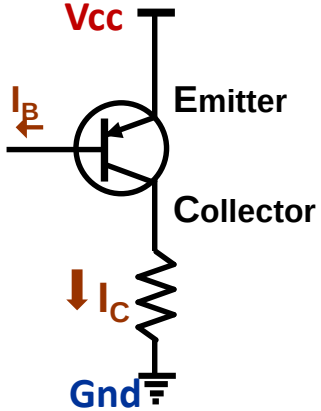
are used on the external input and output pins of the Cyclone family FPGA.

$I_C = \beta * I_B$









NPN

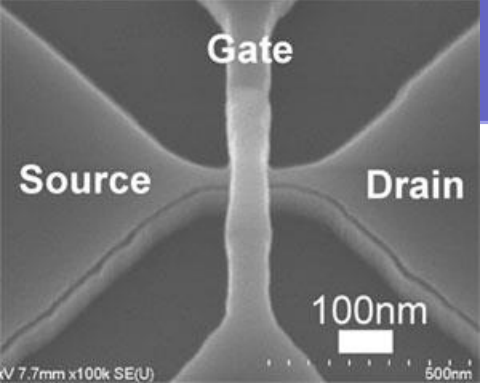
PNP

FPGA Logical Elements: CMOS

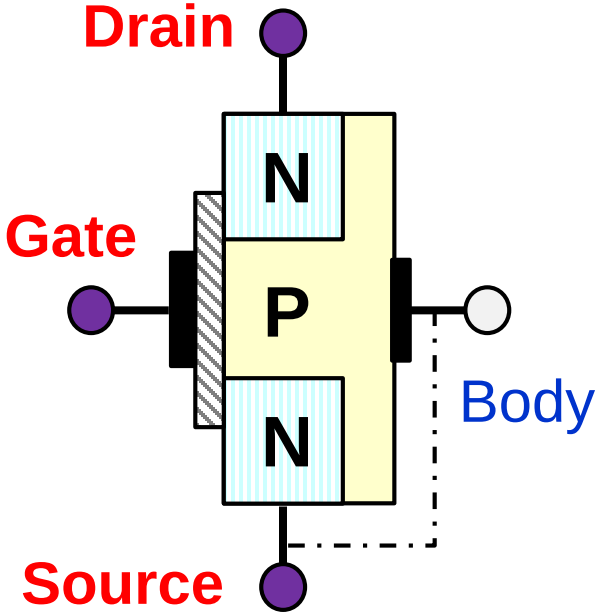
Complementary **M**etal-**O**xide **S**emiconductor Field Effect Transistors, *pronounced "sea-moss."*

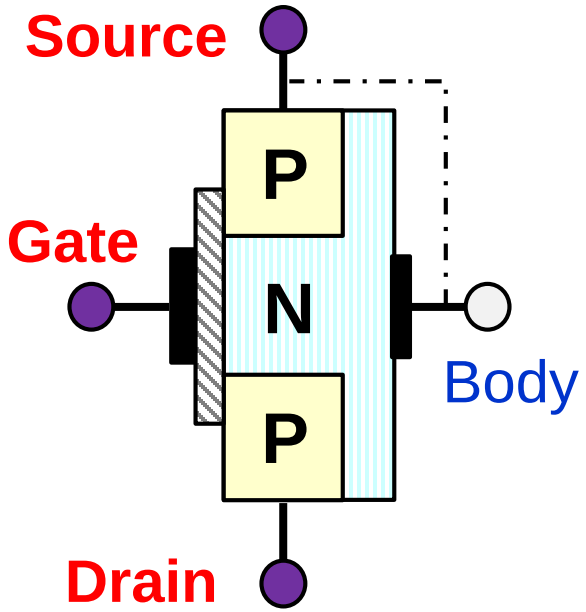
The lecture will present only a brief overview.
See Logic Circuits on FPGA pages from 56 to 65
for the extended description.

FPGA Logic Circuits: CMOS



[Top view SEM photo: AIST]
SEM =
scanning electron microscope

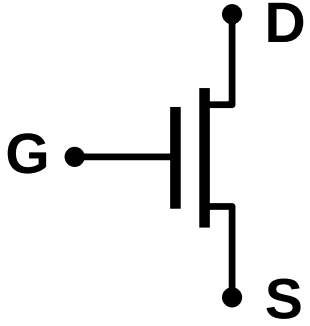


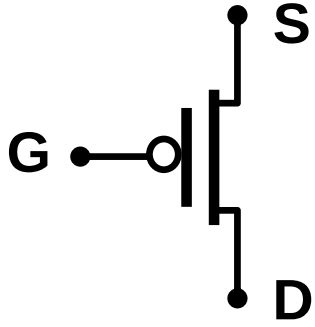


N-channel MOSFET

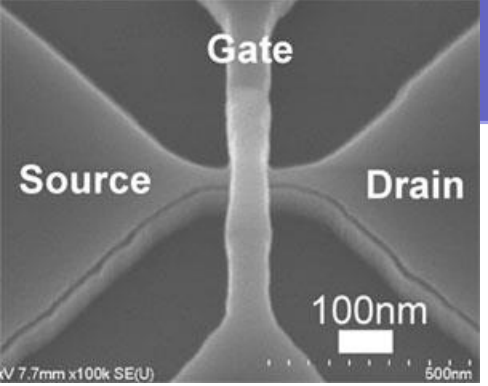
P-channel MOSFET

Simplified Schematic Symbols



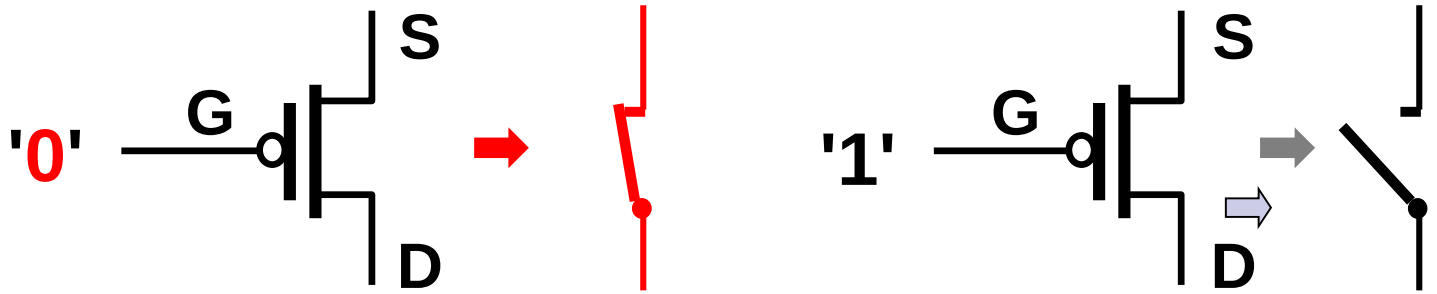


CMOS as Switches



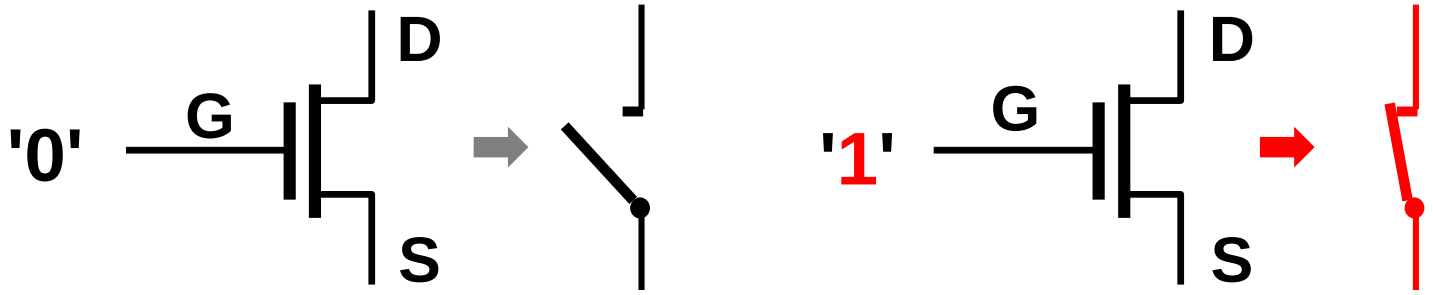
S - *is the source of major carriers in a semiconductor*
NMOS - free electrons, PMOS - holes
G - *gate controlling conductivity by its electric field*
D - *drain*

PMOS

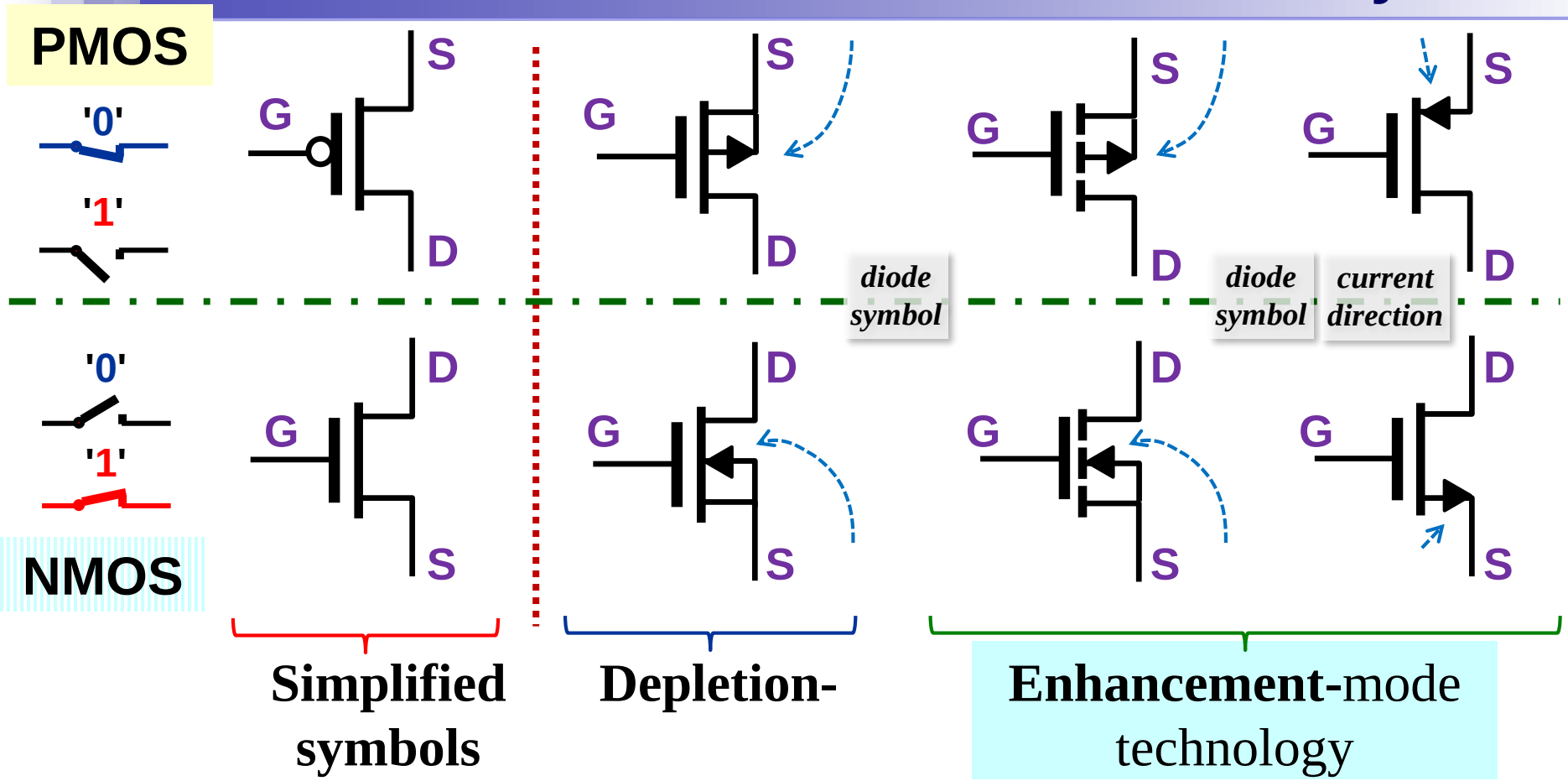


[Foto: Skyspring Nanomaterials]

NMOS



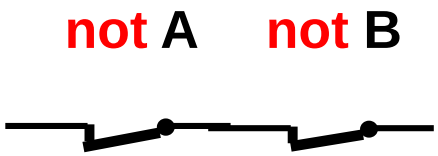
Overview of Used CMOS Symbols



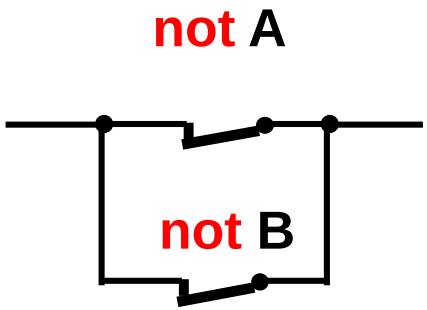
CMOS Logic PMOS only or NMOS only

PMOS

not (A or B) = not A and not B

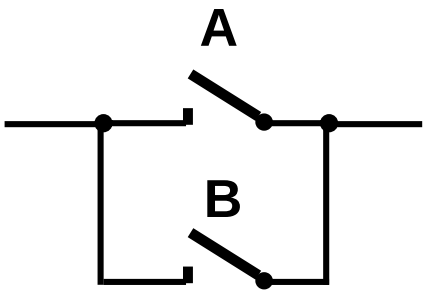


not (A and B) = not A or not B

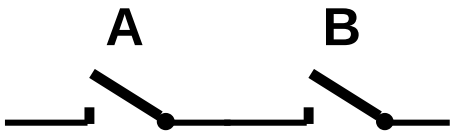


NMOS

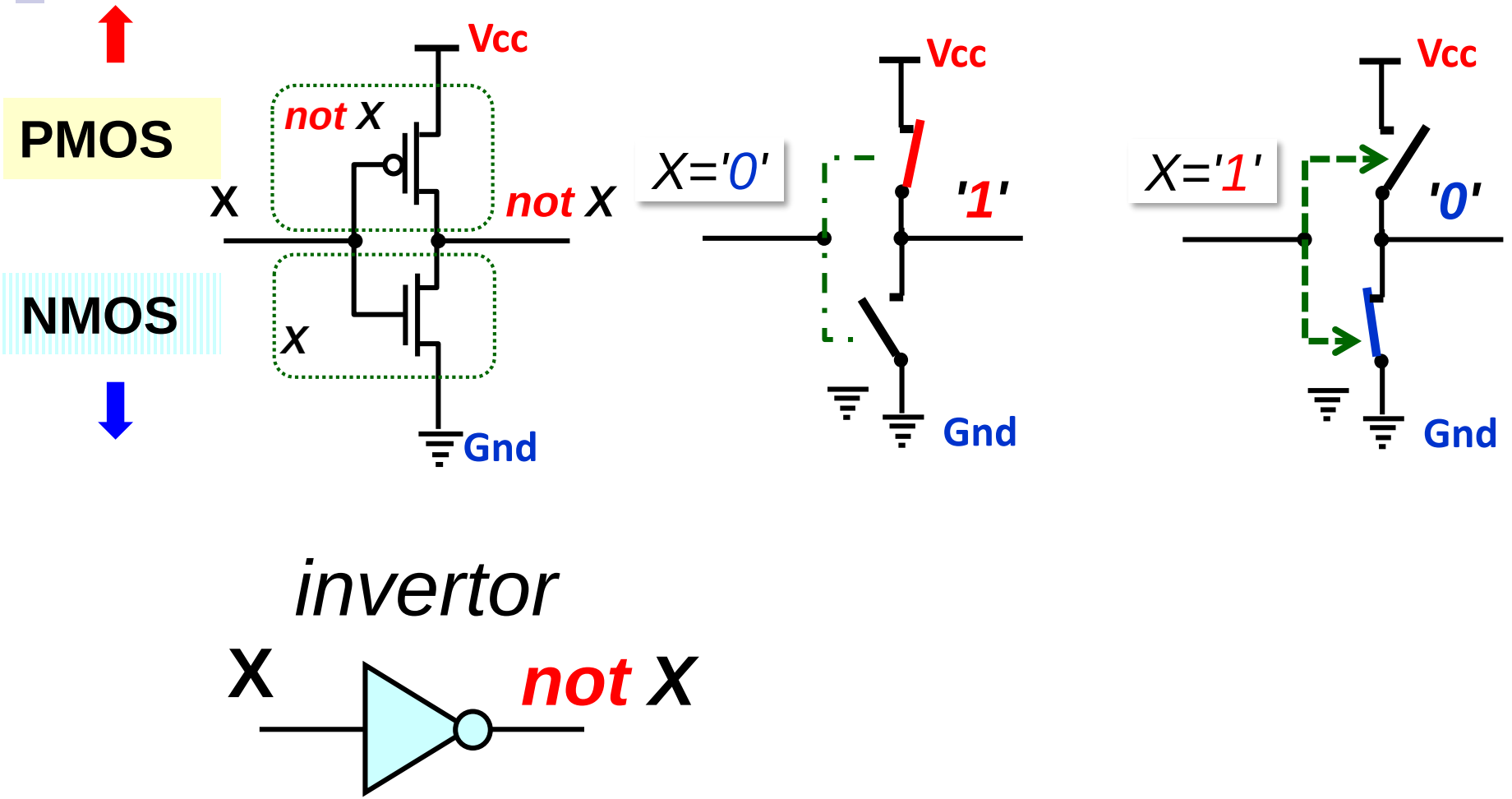
A or B



A and B



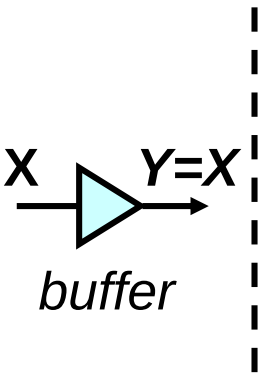
CMOS Invertor

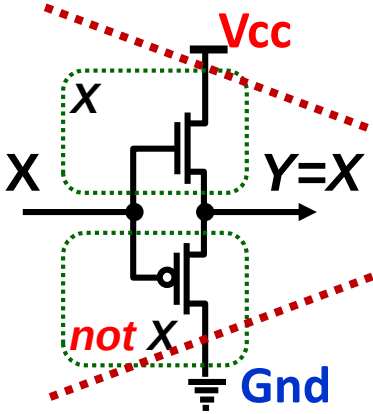


invertor

Beware of CMOS Operating Conditions

Slower Buffer

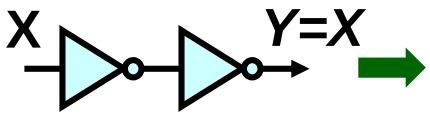


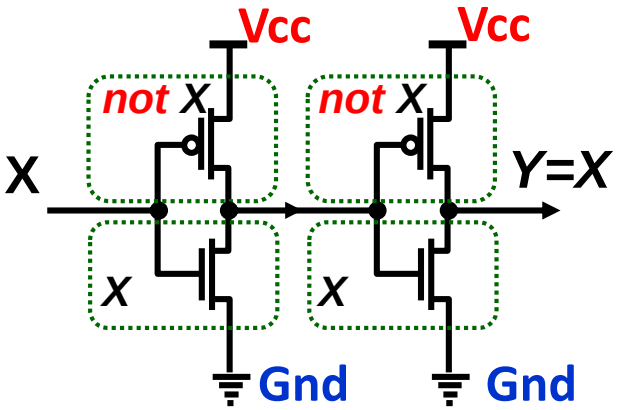


NMOS *top*

PMOS *bottom*

Faster Buffer

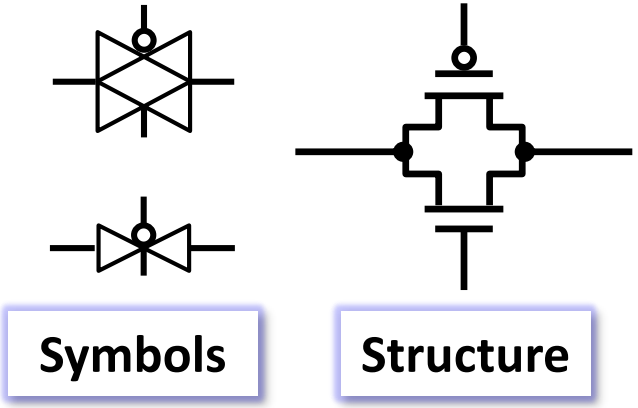


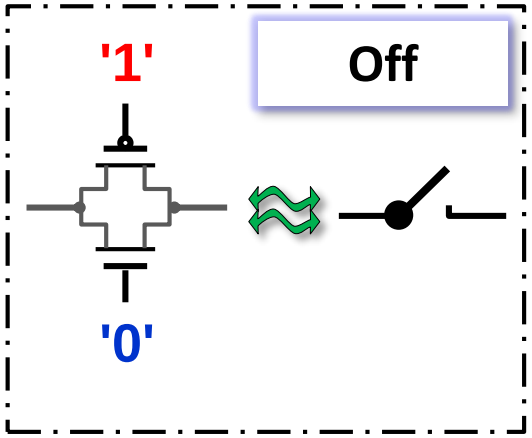


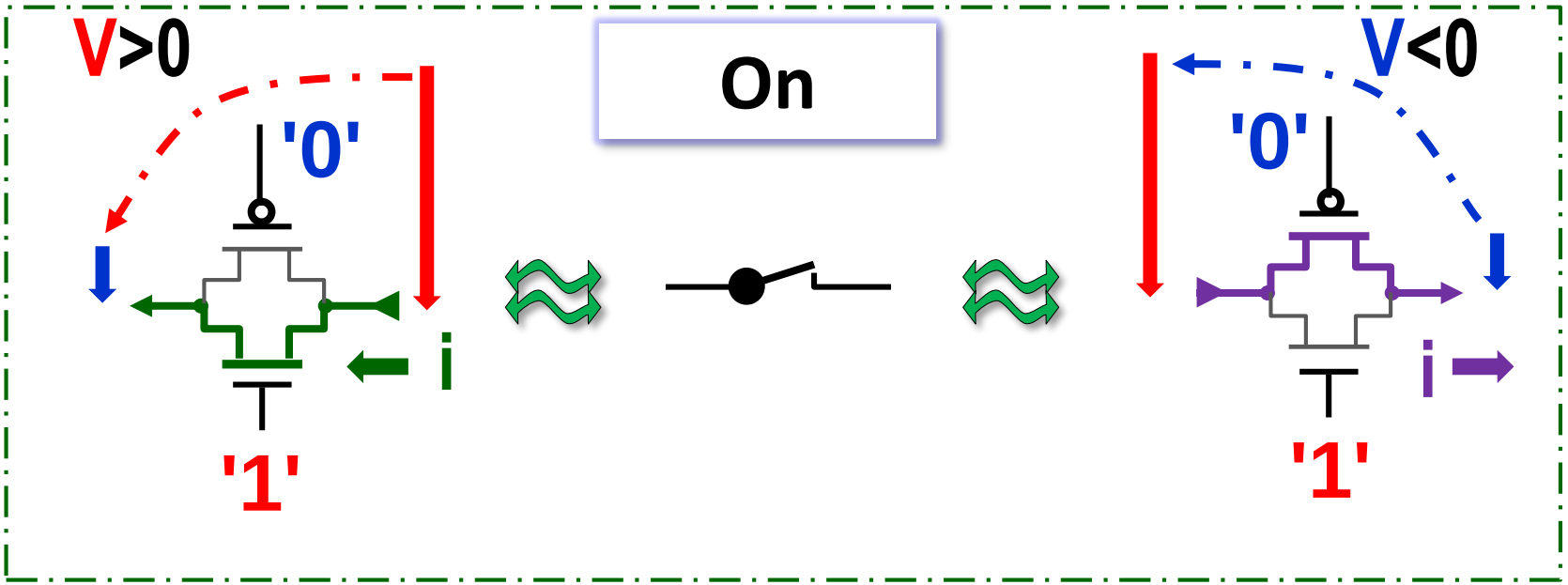
PMOS *top*

NMOS *bottom*

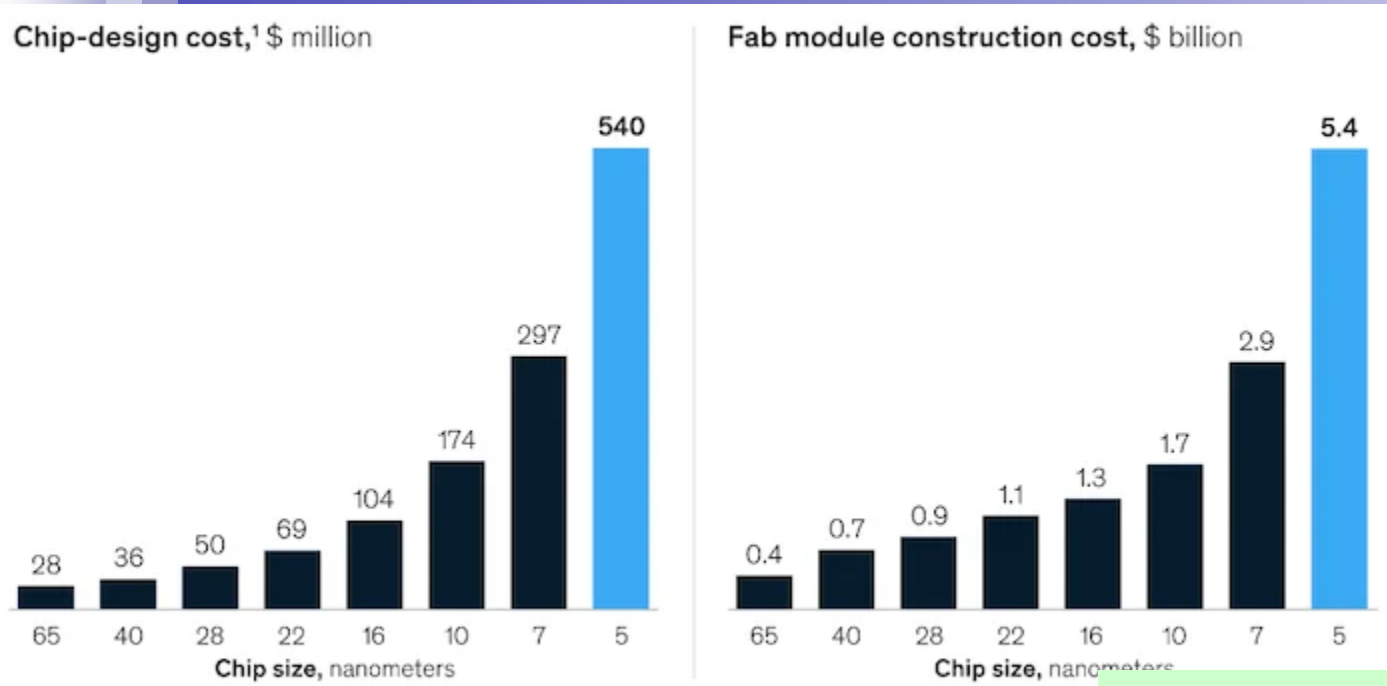
CMOS Switches are also Pairs







Design Price Versus nm Technology



[1]Major components include IP qualification, architecture, verification, physical, software, prototyping, and validation.
Source: IBS; McKinsey

Note: Silicon's atomic size is about 0.2 nm (nanometers) = 2 Å (Ångströms). GaAs ~ 2.5 Å

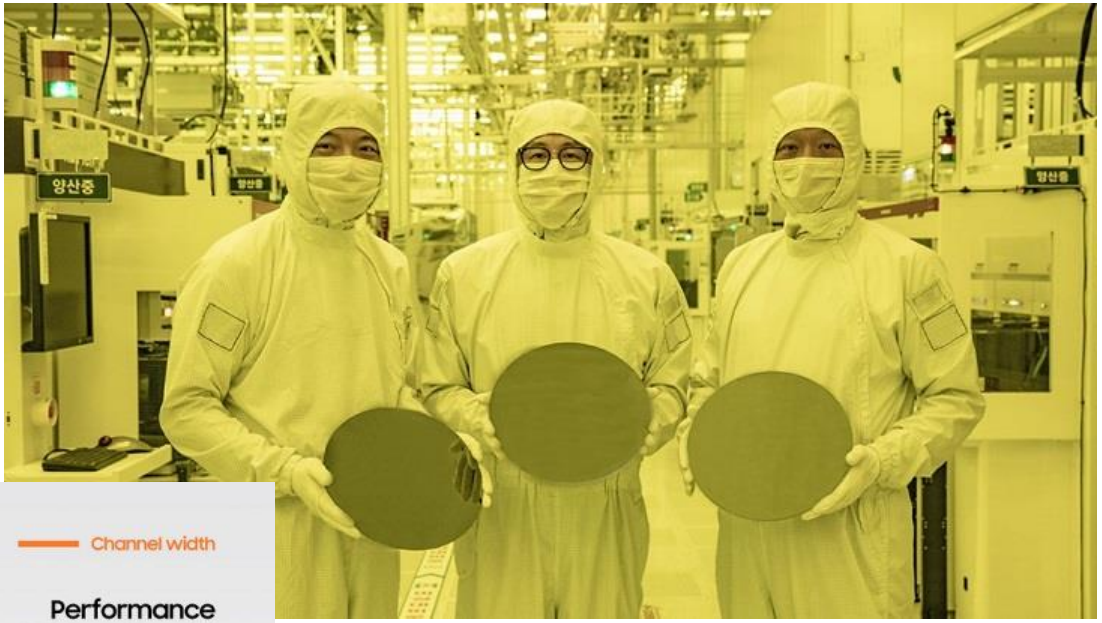
When a chip design is debugged, its wafers are cheaper.

3nm has a rough guess to be about $10 billions.

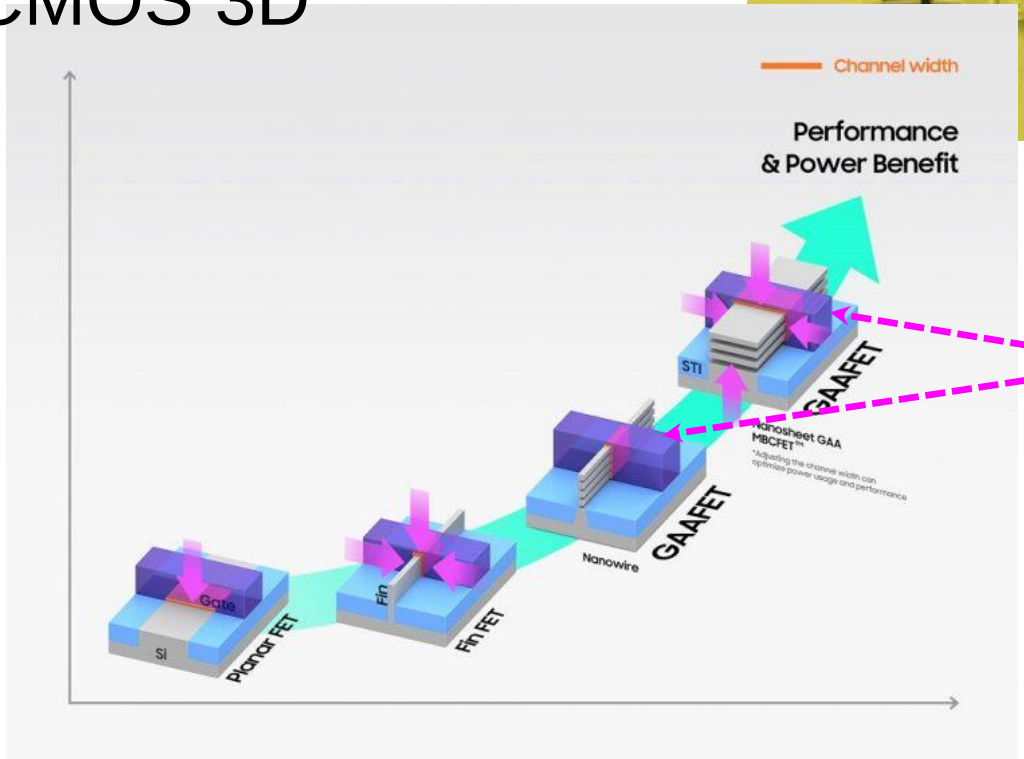
Source: ***McKinsey and Company***

Microsoft spent *?probably?* $**20B/yr** in the fabrication of its 18Å (=1.8 nm) technology.

3nm Technology Wafers



CMOS 3D



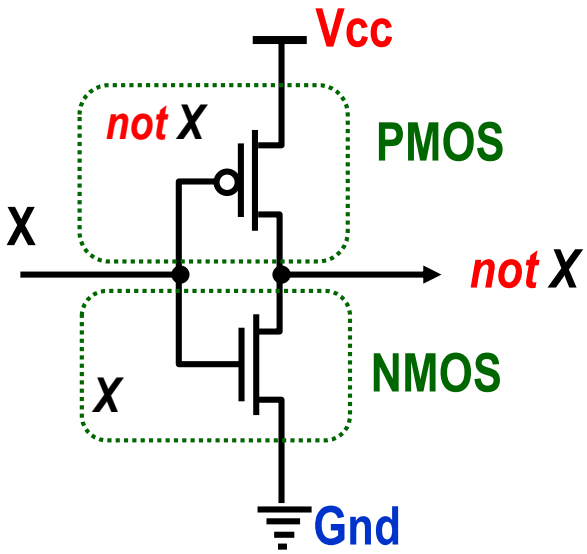
Gate

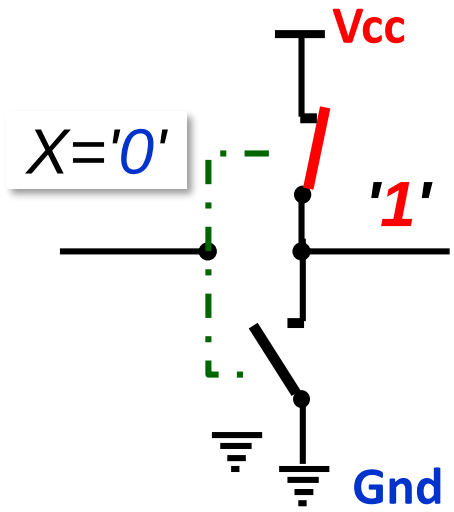
Images: Samsung

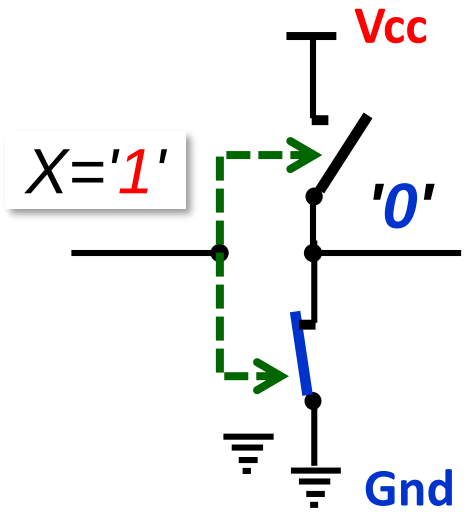
Delay

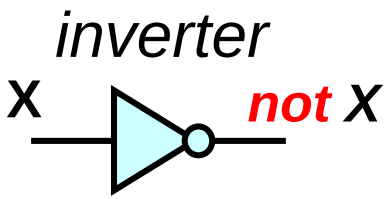
Image: Corel Cliparts

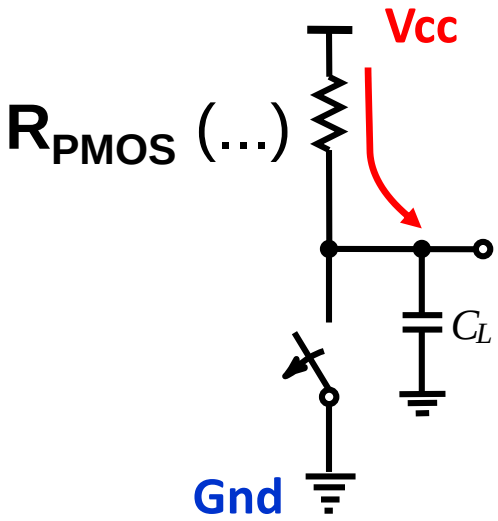
Repeat: CMOS inverter



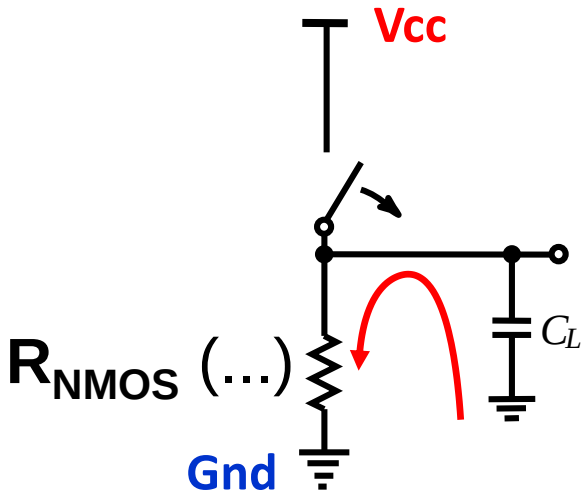








Charging

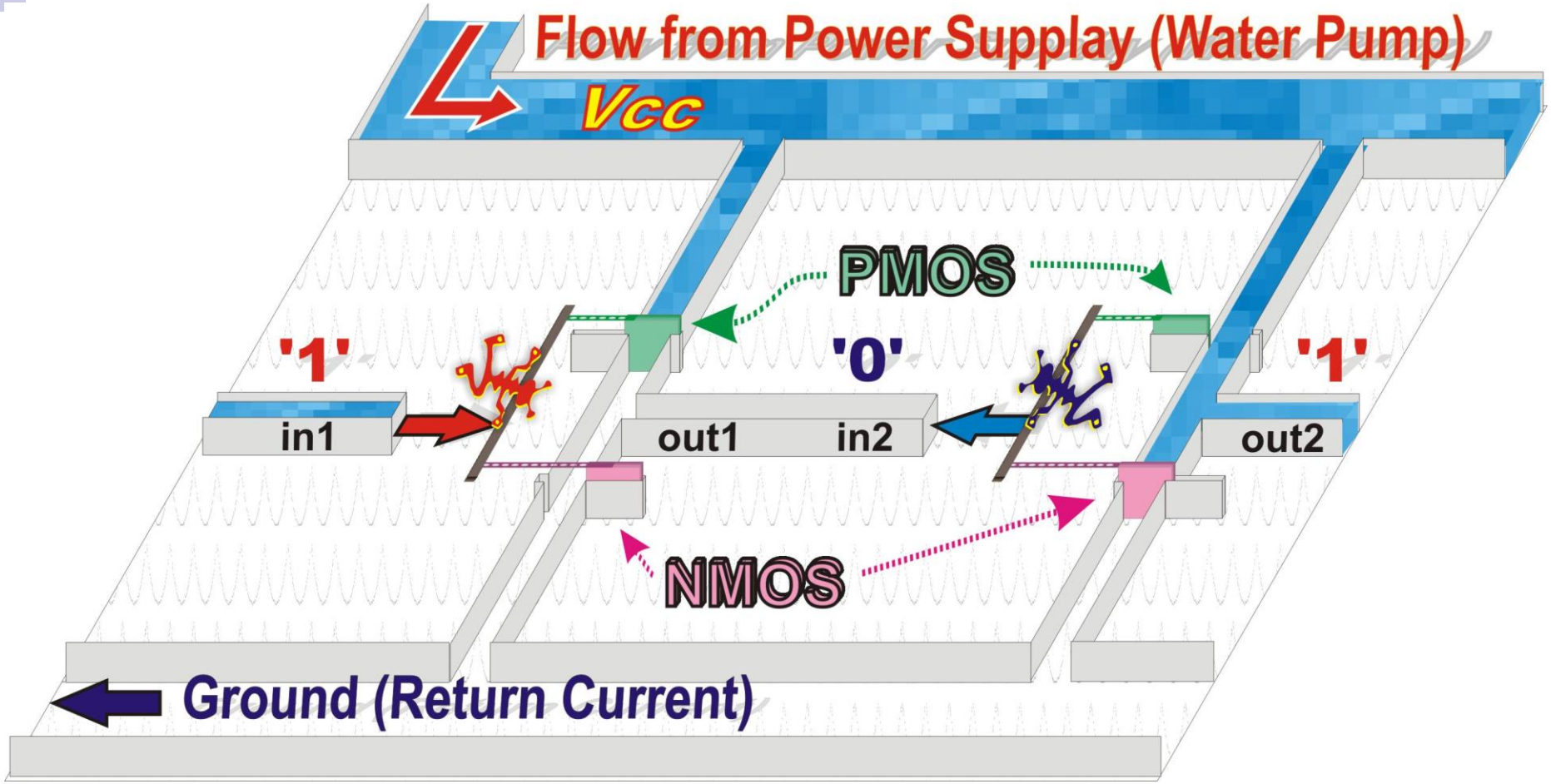


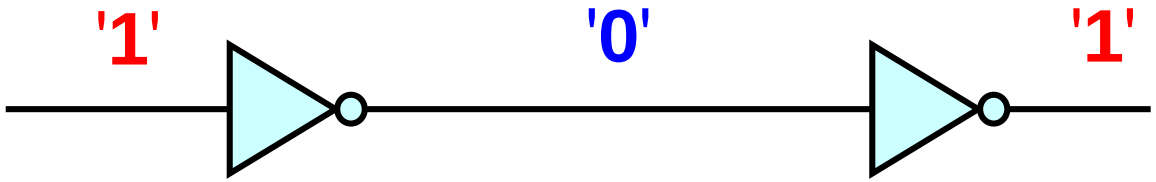
Discharging

We will emulate delays by lock-chambers

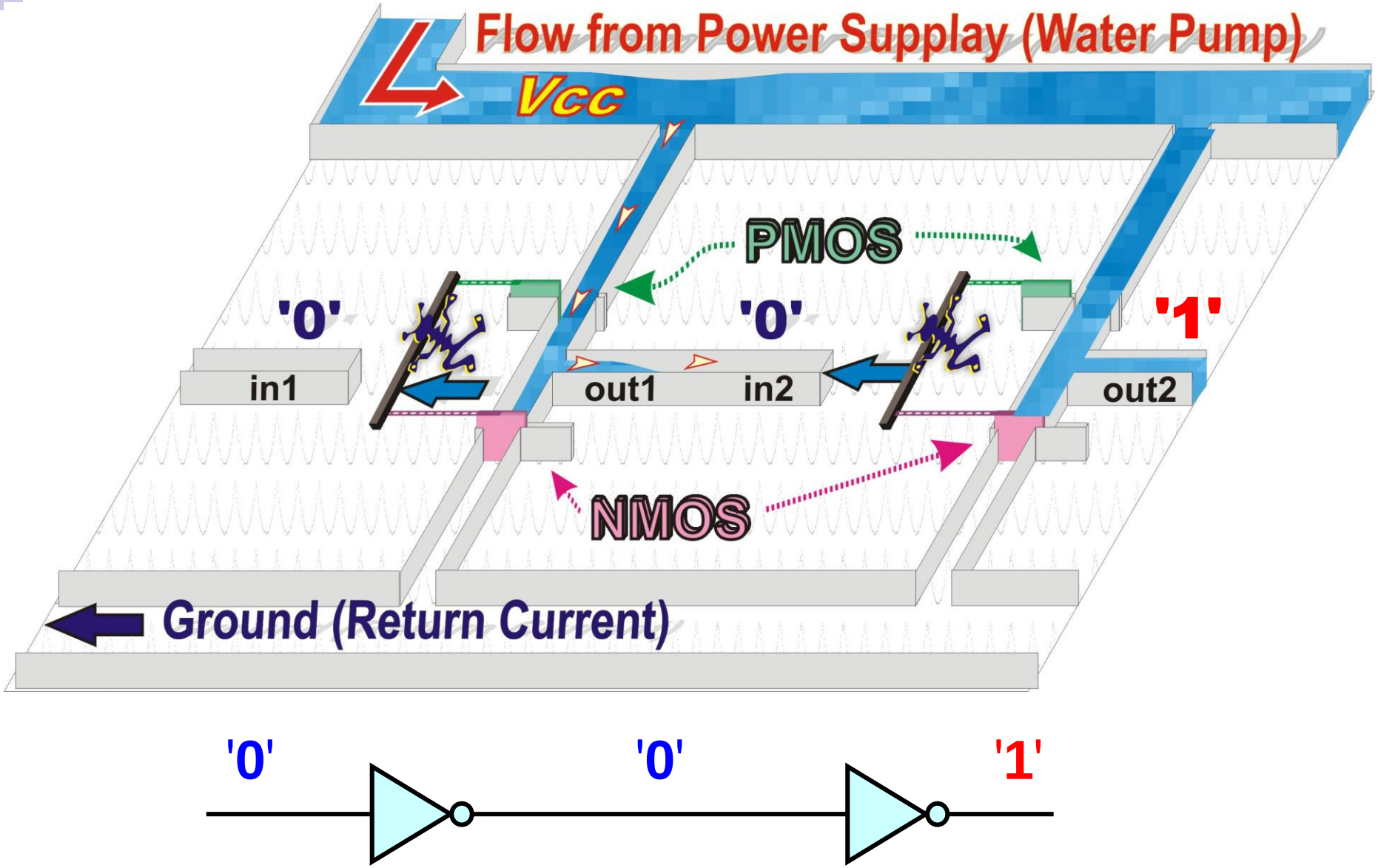
[Photo: R. Šusta, Crinan Chanel, West of Scotland]

Model of two inverters 1/4

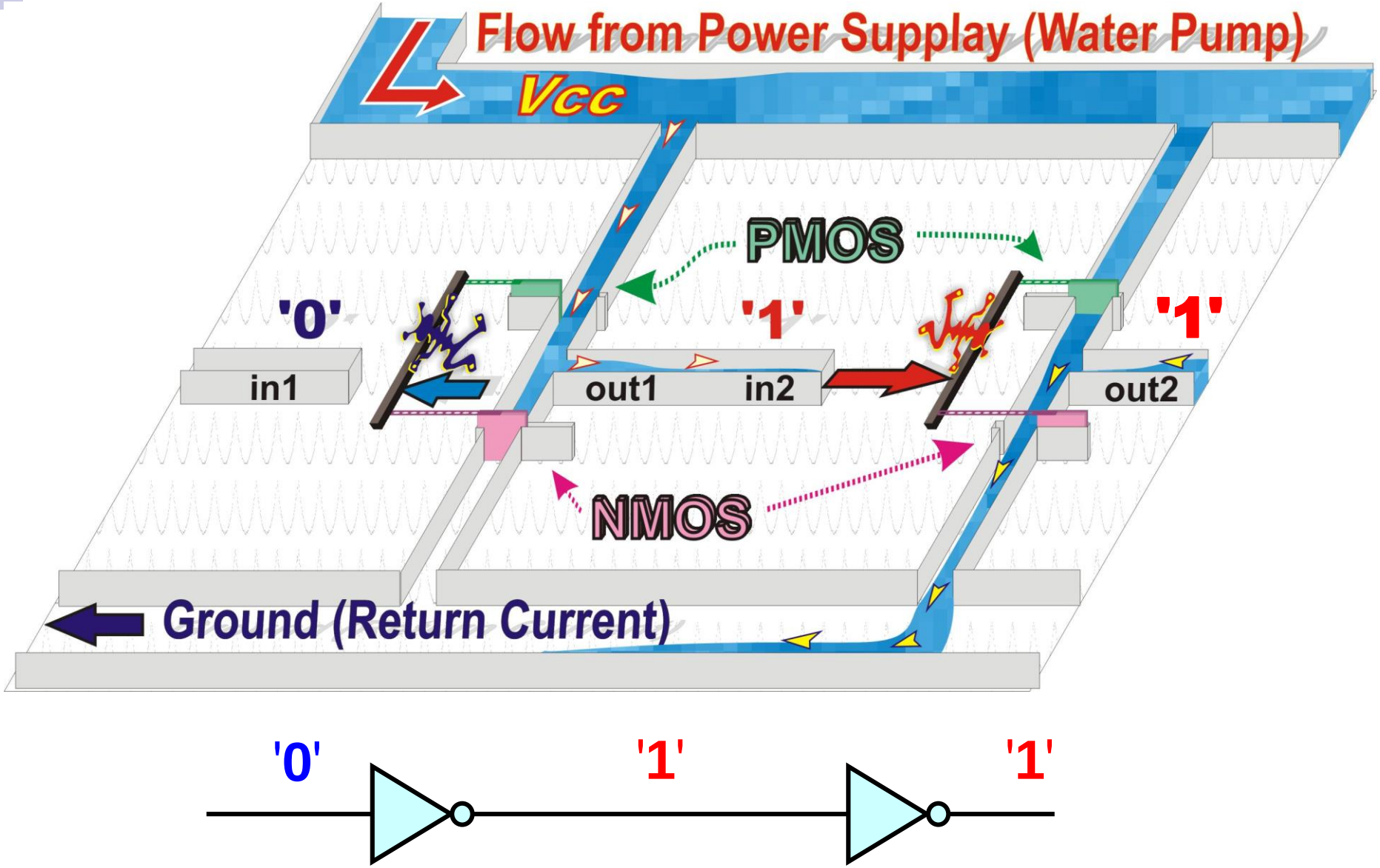




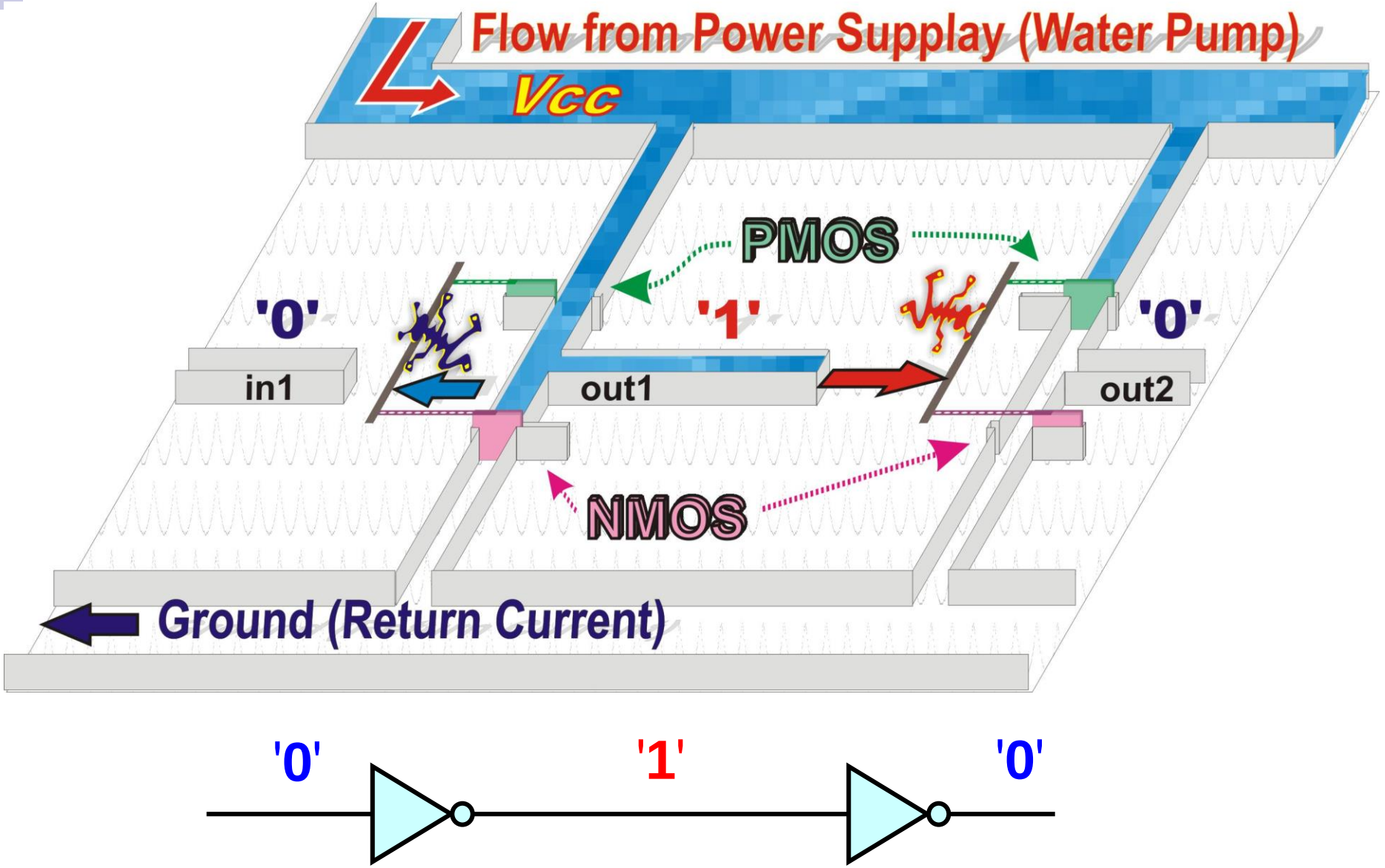
Model of two 2/4 inverters



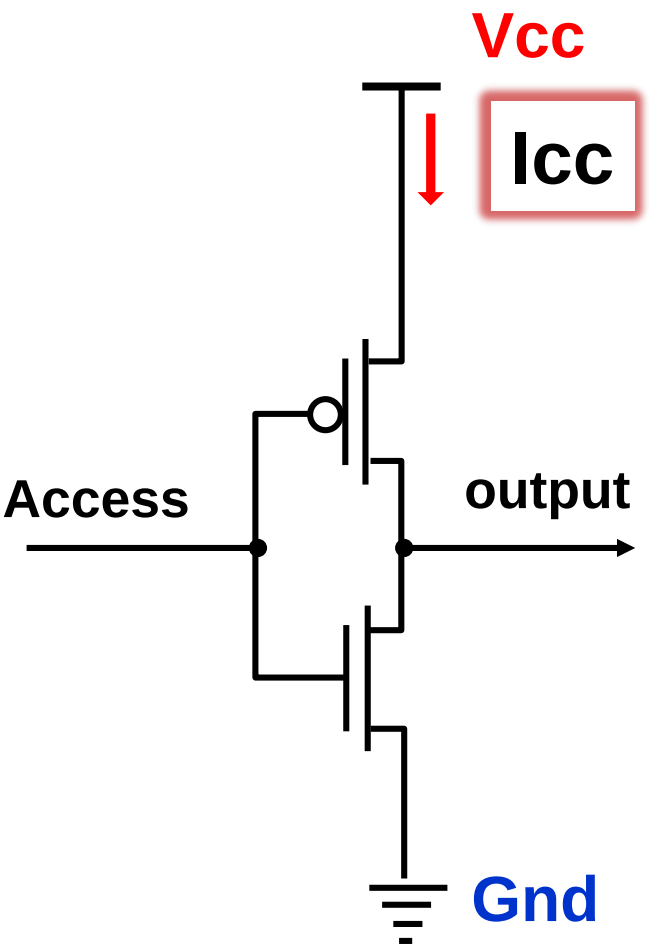
Model of two inverters 3/4

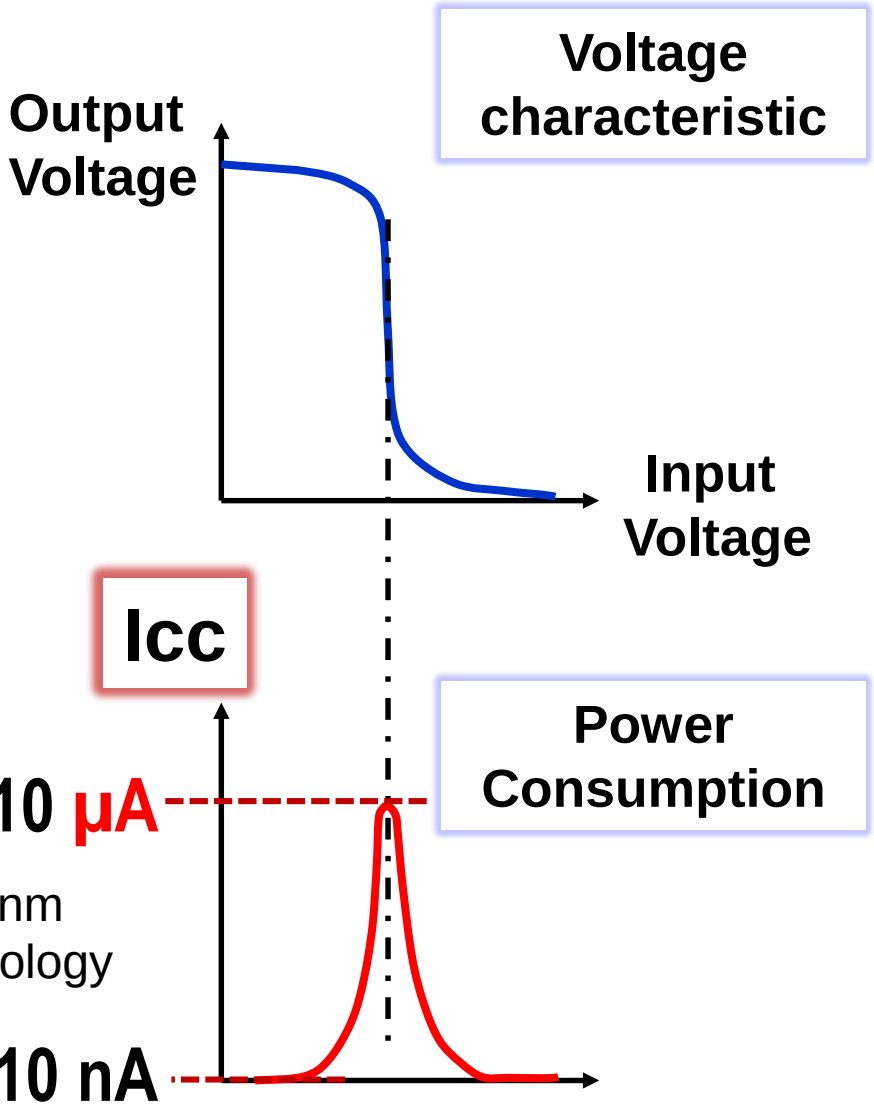


Model of two 4/4 inverters

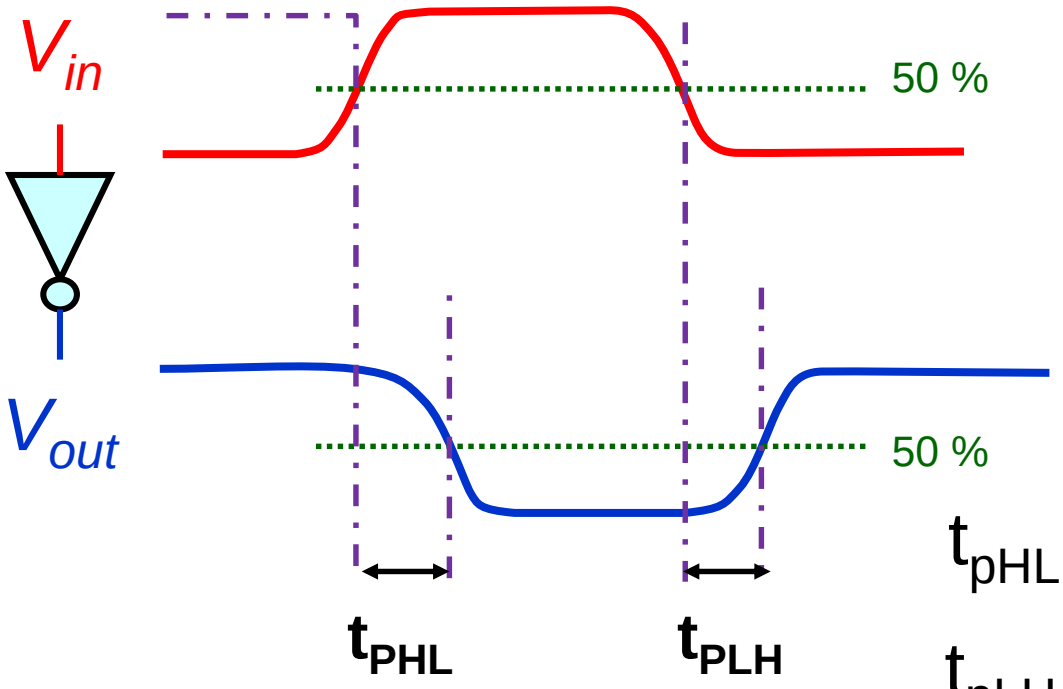


Current Peeks





Propagation Delay Times



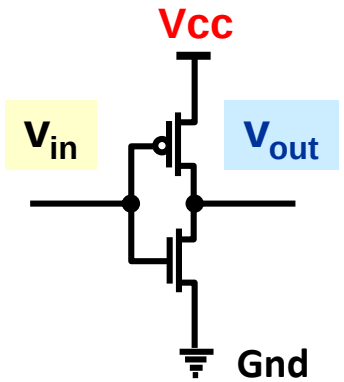
t_{pHL} - from a high to a low

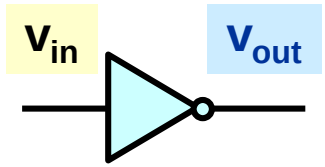
t_{pLH} - from a low to a high

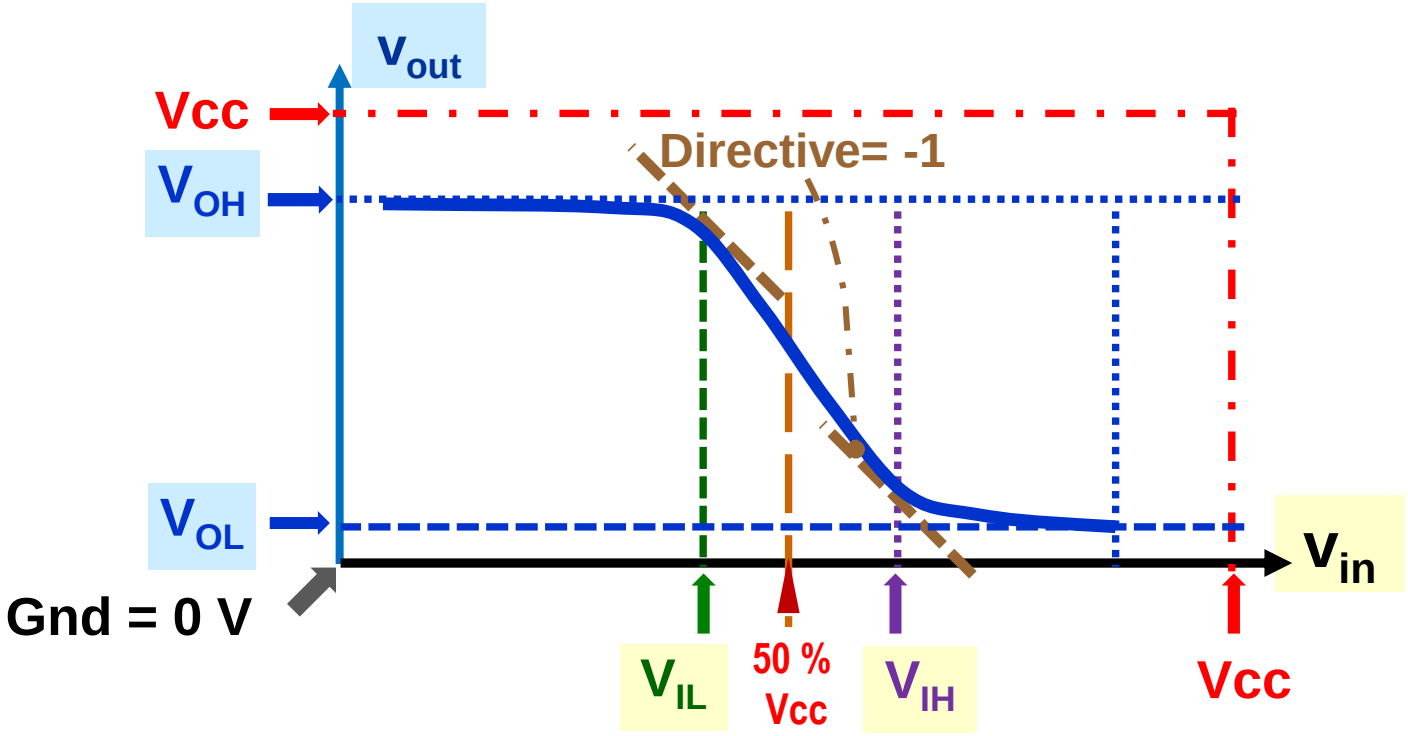
Average Propagation Delay

$$t_{pd} = (t_{pHL} + t_{pLH}) / 2$$

Characteristic voltages





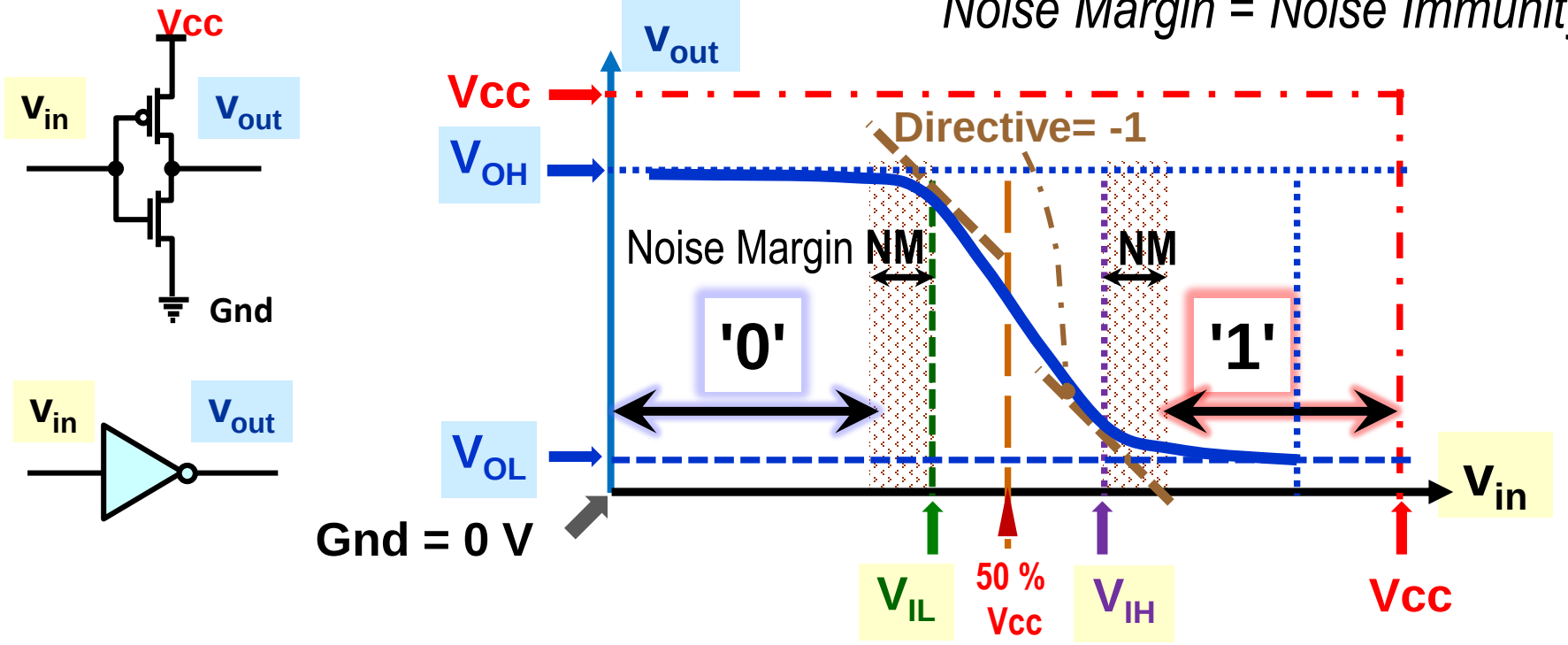


Manufacturers' catalogues always st

- V_{IL} , V_{IH} input low, high
- V_{OL} and V_{OH} output low, high

Logical '0' and '1'

Noise Margin = Noise Immunity



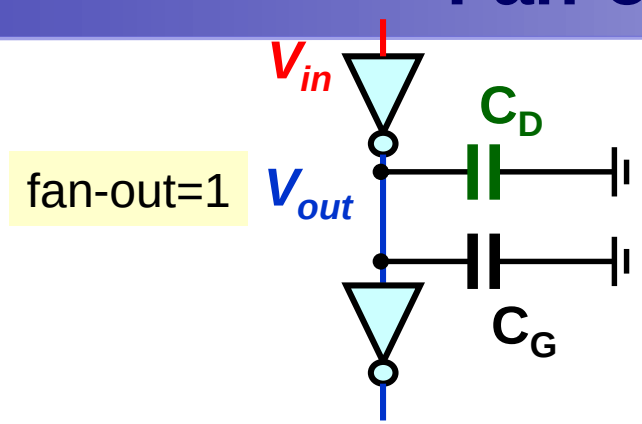
Cyclone IV E in DE2-115 and VEEK-MT2

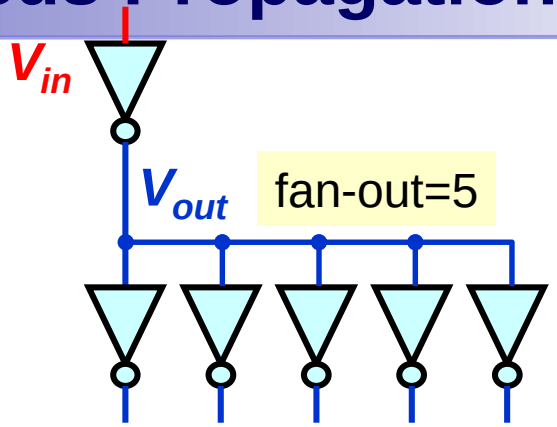
Vcc	V_{OL} [V]	V_{IL} [V]	V_{IH} [V]	V_{OH} [V]
1.2 V (FPGA core)	**0.3**	**0.42**	**0.78**	**0.9**
3.3 V (In/Out Pins)	**0.33**	**1.0**	**1.65**	**3.0**

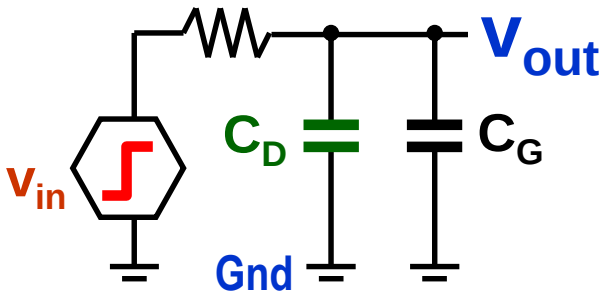
Noise margin

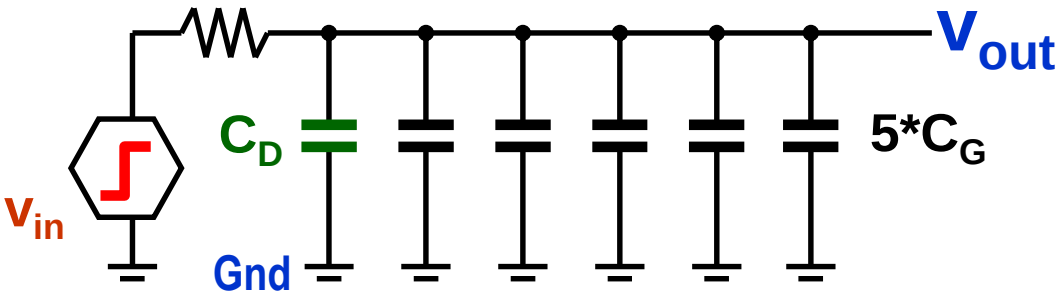
- The noise immunity is chosen to increase the chances of the circuit functioning correctly under the specified operating conditions.
- Sources of noise include the main power supply, where transistor switching generates unwanted noise on the ground and supply wires.
- Noise is also caused by environmental effects such as electric and magnetic fields and radiation waves.

Fan-out versus Propagation Delay

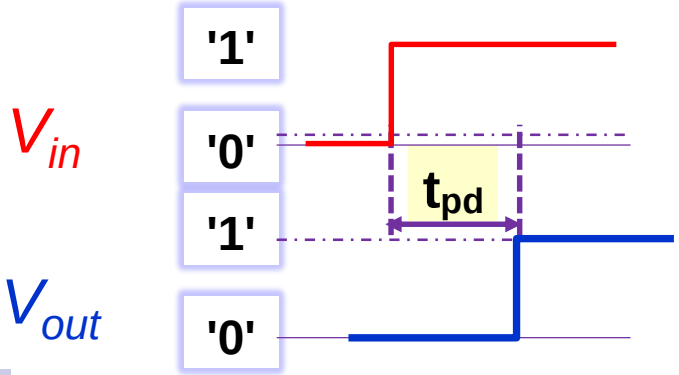


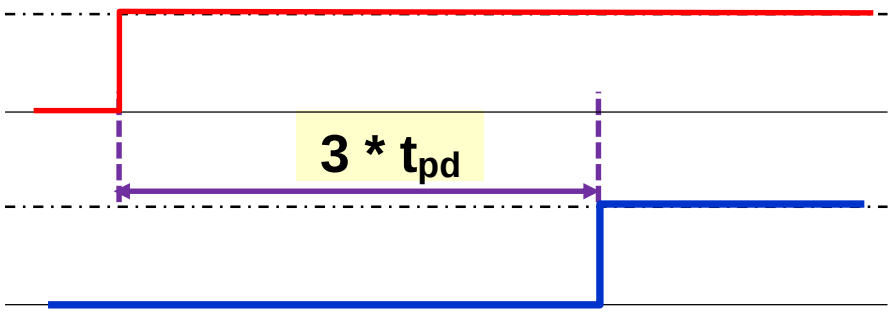






if $C_D \approx C_G$ $t_{pd} = 0.7R(C + C)_{DG}$ $t_{pd2} = 0.7R(C + 5*C_{DG}) = 3*t_{pd}$





Hazard

Medieval English took the word ***hazard*** *from the French* ***hasard****, which in turn assimilated it from Arabic* ***az-zahr*** *= dice.*

In digital logic, a hazard is an undesirable effect caused either by a functional deficiency or by external influences.

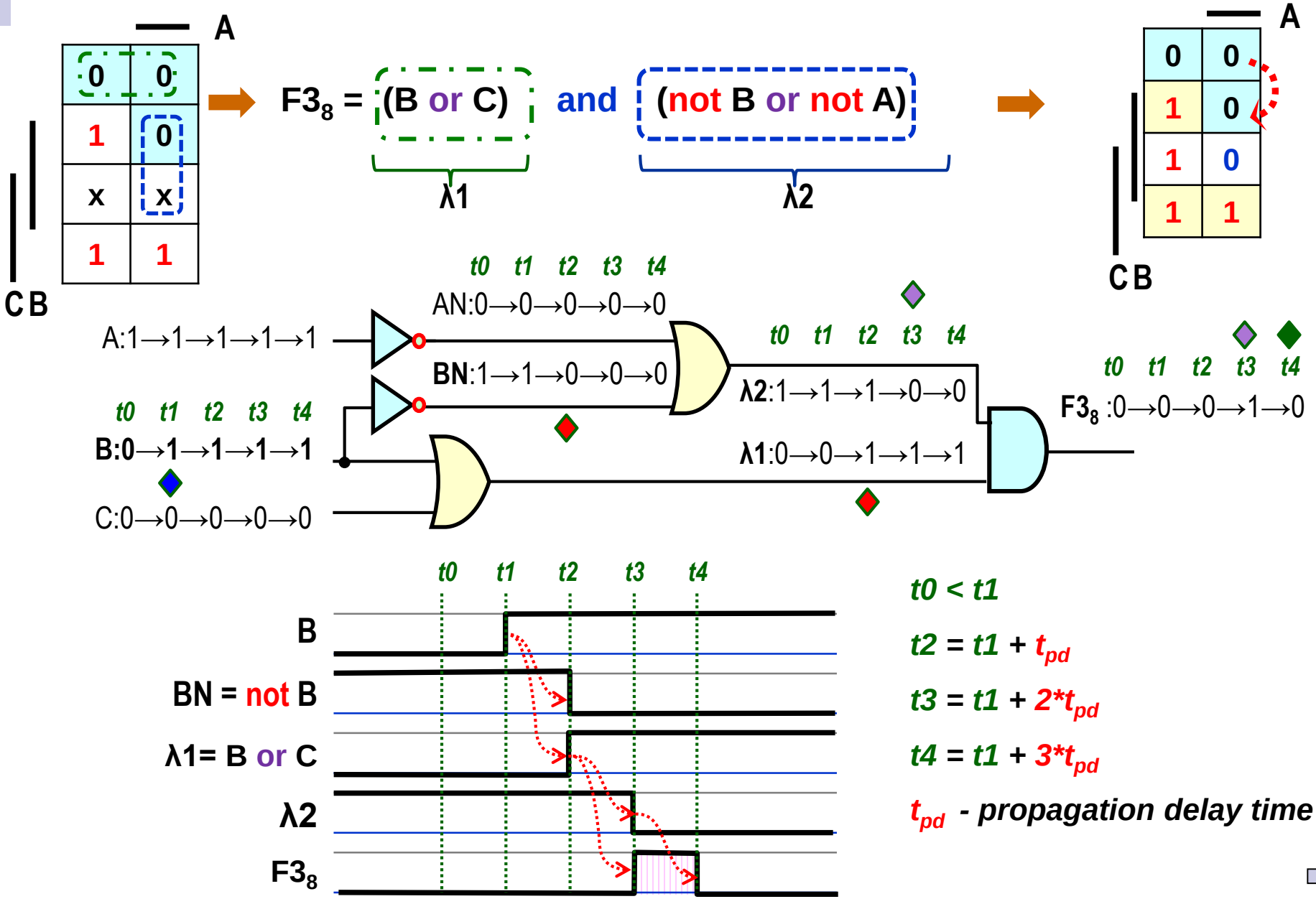
Steady-state behavior - output value at long-term stable inputs.

Transient behavior - temporary values of outputs after a change of inputs.

Glitch - *glitch, malfunction - comes from German glitschig (slippery)*
In circuits: a **short unwanted pulse during transients**.

If the circuit can roll out **Glitch**, then he has a **hazard**.

Different lengths of logic paths can cause hazards!



How do we eliminate hazards?