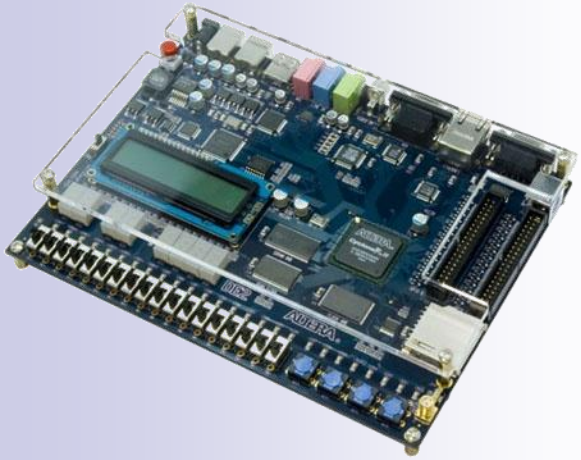
Logic Systems and Processors
cz:Logické systémy a procesory



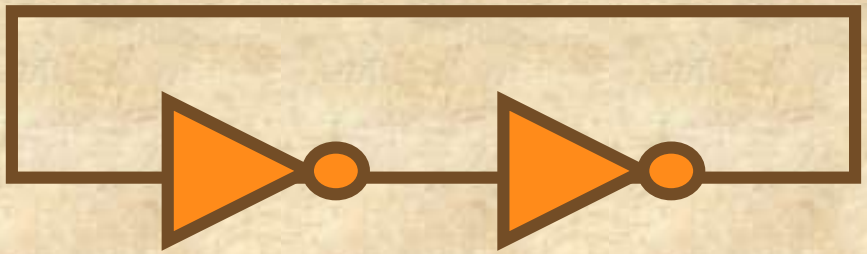
Lecturer: Richard Šusta

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

Version V1.0

DFF - Data Flip-Flop

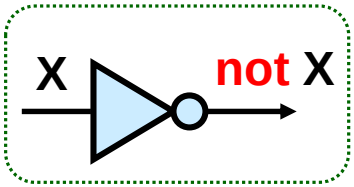
also contains memory loops from invertors

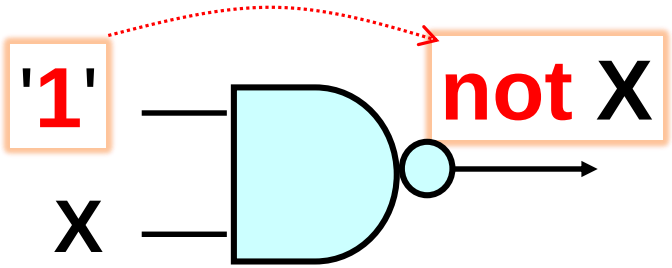


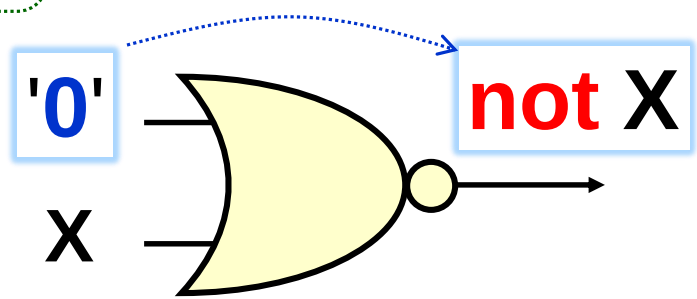
Repeat: *Identity Law and Annulment* Law

Identity Law

Neutral values

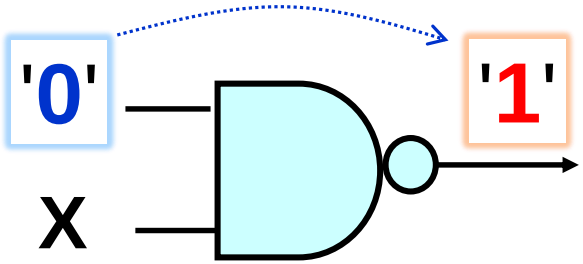


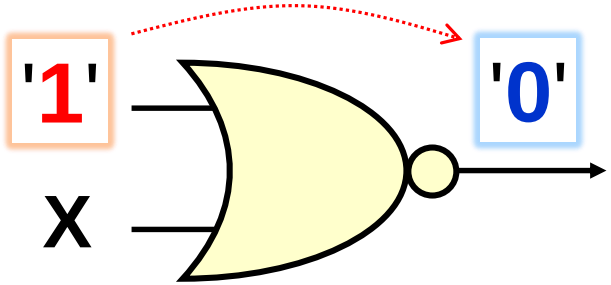




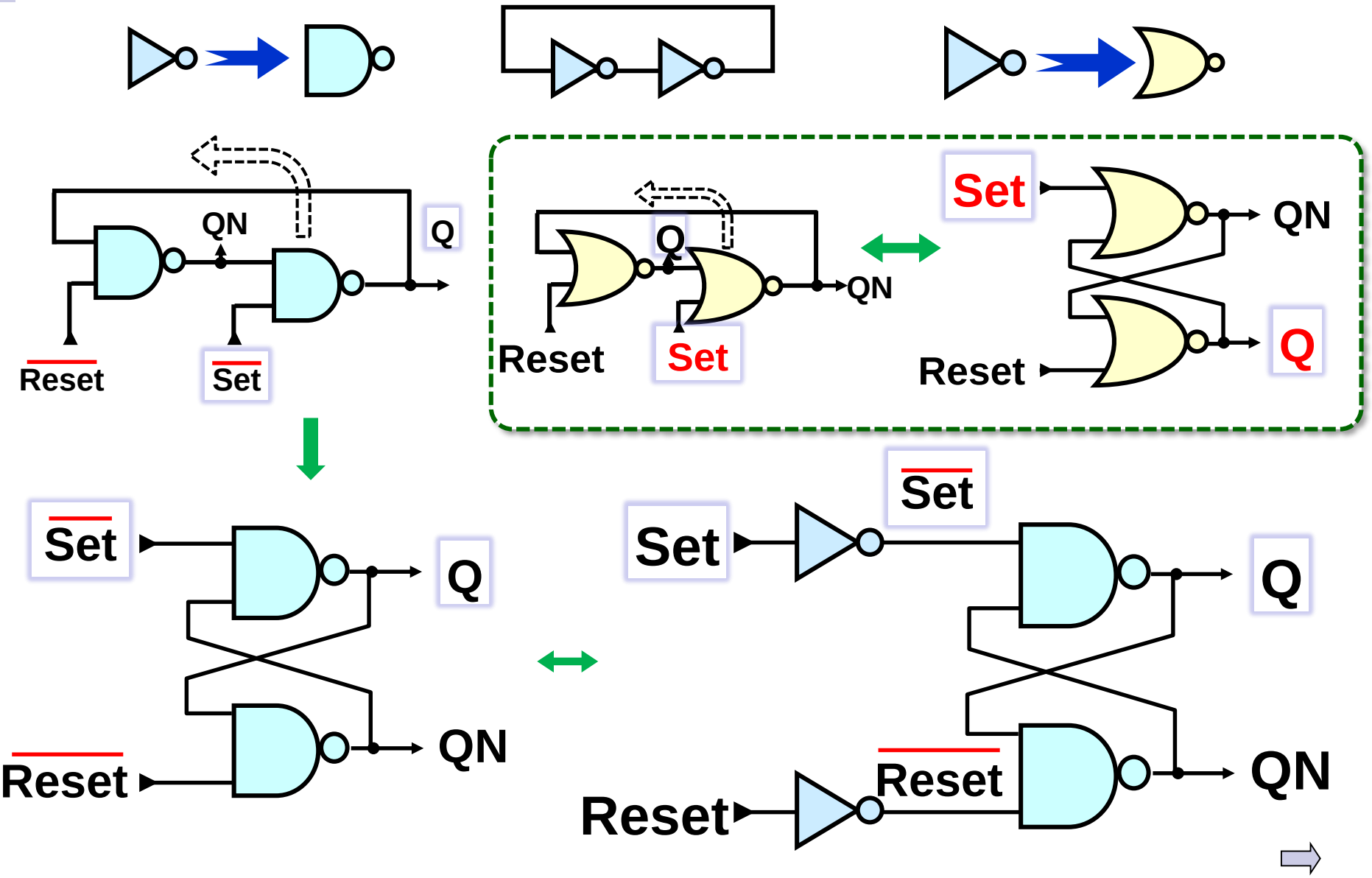
Annulment Law

Aggressive values





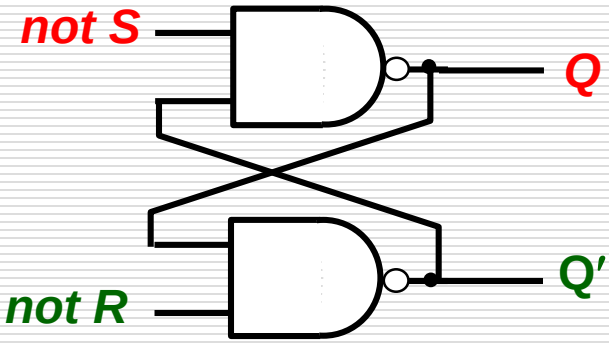
RS-latch

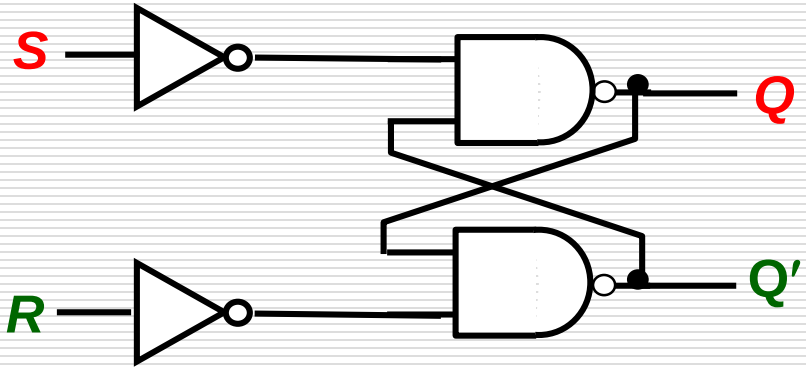

QN
Q
Reset
Set
Q
QN
Reset
Set
Set
QN
Q
Reset
Set
Q
QN
Reset
Set
Set
Q
QN
Reset

How do I remember the flip-flop circuit from NAND gates?

NAND - the *reasoning is symmetric to NOR.*

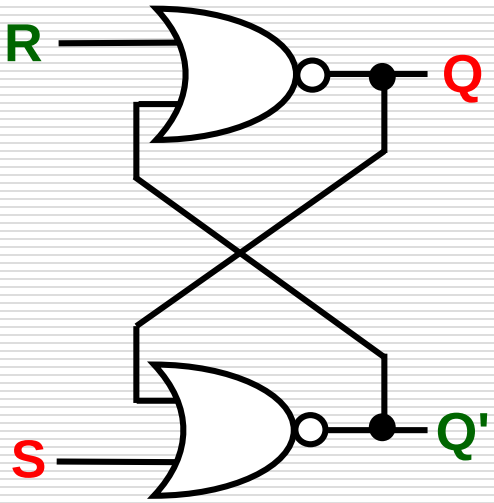
- AND is the selection of the minimum. A NAND (not AND) latch has a pivotal input value of '0' - if any of its inputs are at '0', then the output will definitely be '1' regardless of the values of the other inputs; hence, a NAND latch circuit has active inputs at '0'.
- If the input goes to ´0´, the output of the NAND gate will be ´1´, and therefore Q will be at the same gate as the S input, but S is active here at ´0´ and so will be negated.





How do I remember the flip-flop circuit from NOR gates?

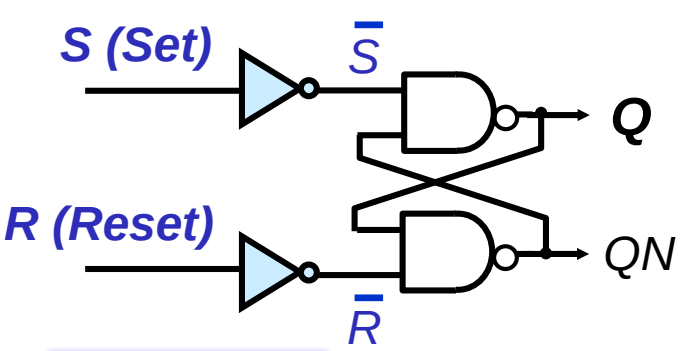
- The OR gate is the maximum selection.
- The NOR (not OR) gate has a key input value of '1' - if any of its inputs is at '1', then the output will be ´0´ regardless of the values of the other inputs. It follows that the NOR flip-flop circuit has active inputs R and S at ´1´.
- When R and S are '1', both outputs will be '0'. This condition is considered forbidden only if it is requiredby an external constraint, Q = not Q'. The circuit itself does not impose any forbidden states.
- Since the output of the NOR gate goes to '0' when the input is '1', the output Q cannot be connected to the gate with input S; instead, it will be connected to the other gate.



Latch

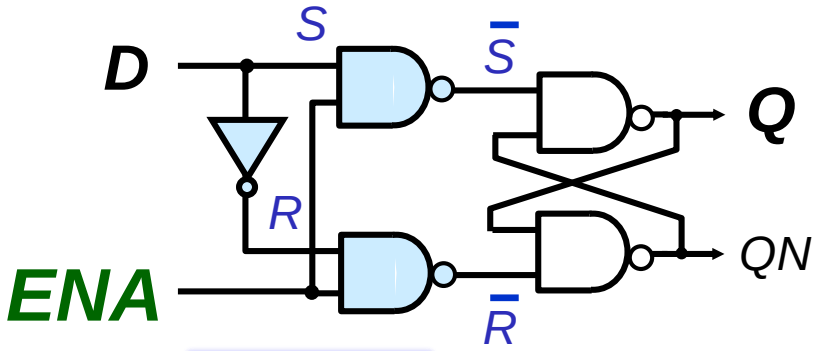
Image: https://simple.wikipedia.org/wiki/Latch

D-latch

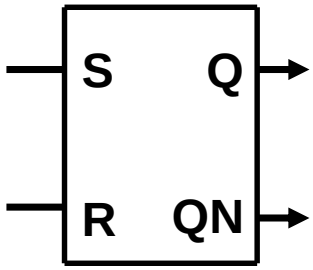


RS Latch

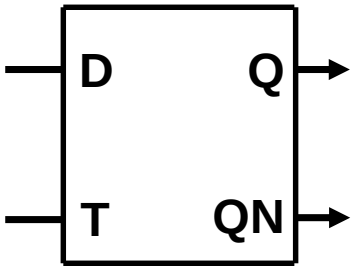
≈

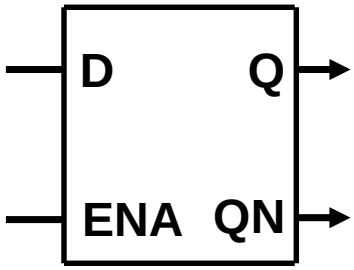


D Latch



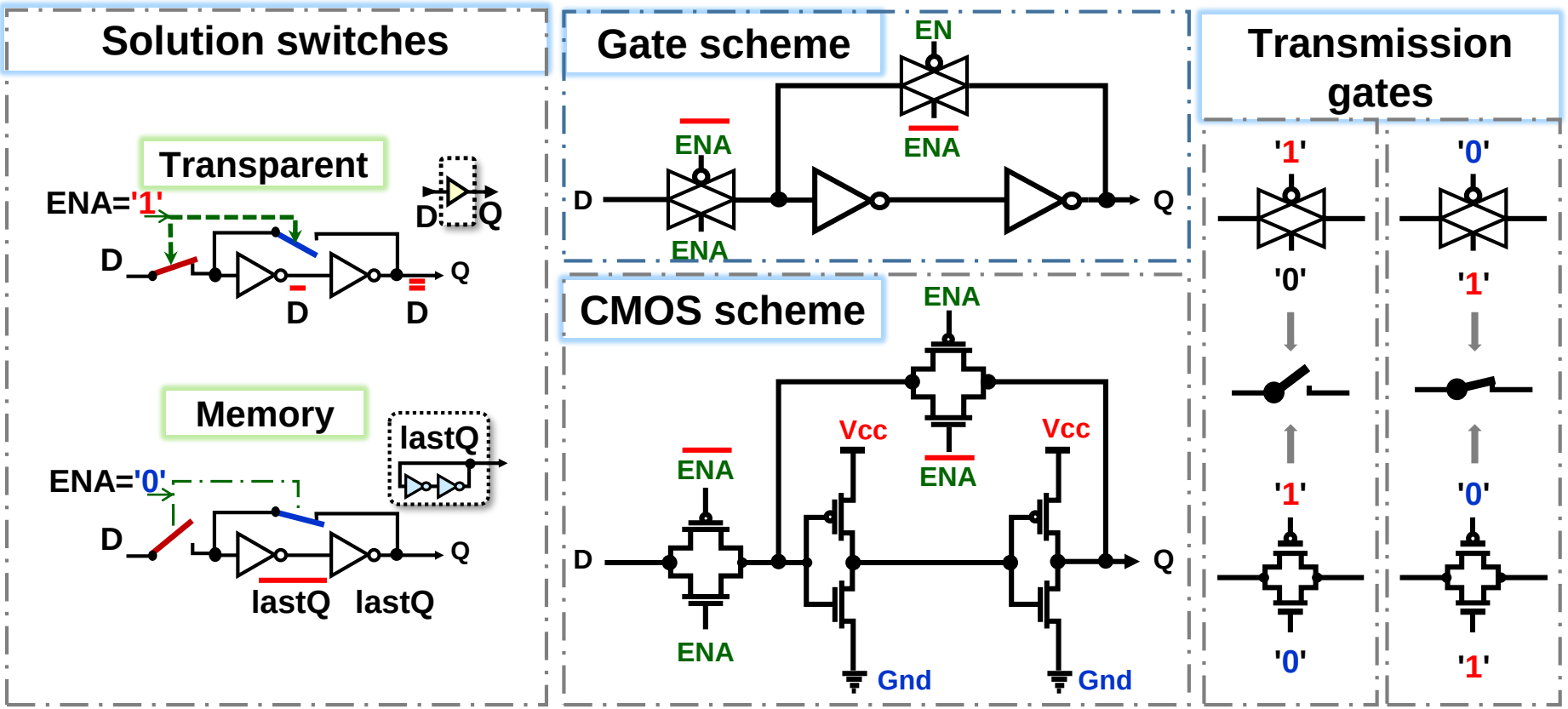
RS Latch Symbol

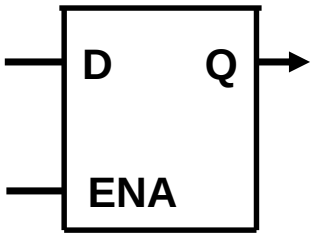




D Latch Symbols

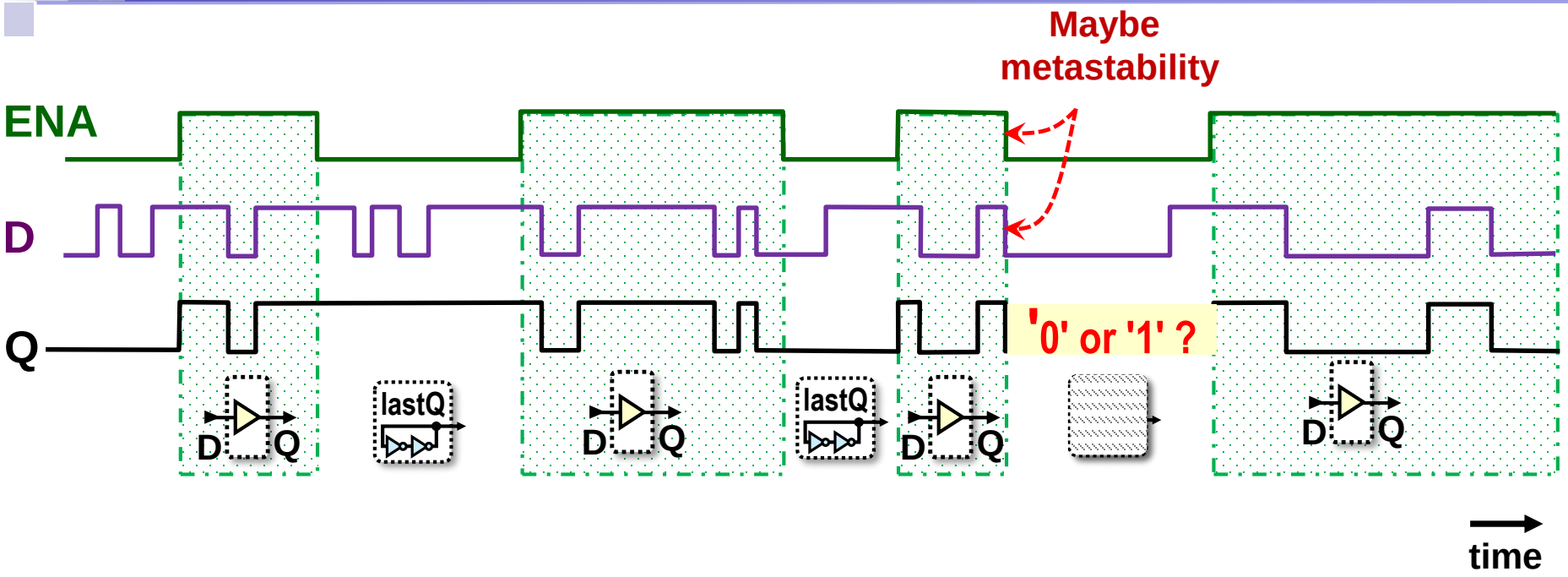
Transparent Latch(D-latch) in CMOS

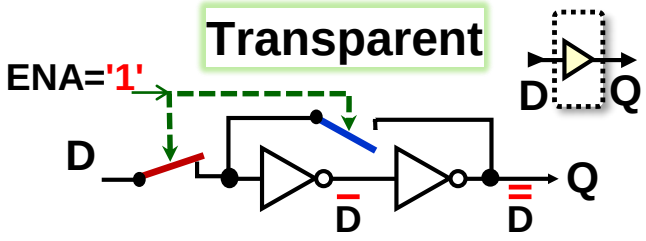


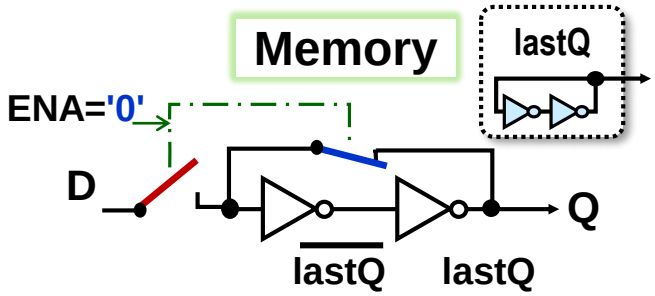


It has two states:
Transparent and **Memory**

D-latch behaviour







We will discuss the metastability in the next lecture.

Flip-Flop Circuits

"Computer flip-flops"

Flip-flop = a) flip-flops b) reverse direction

Latch - Flip-flop in English

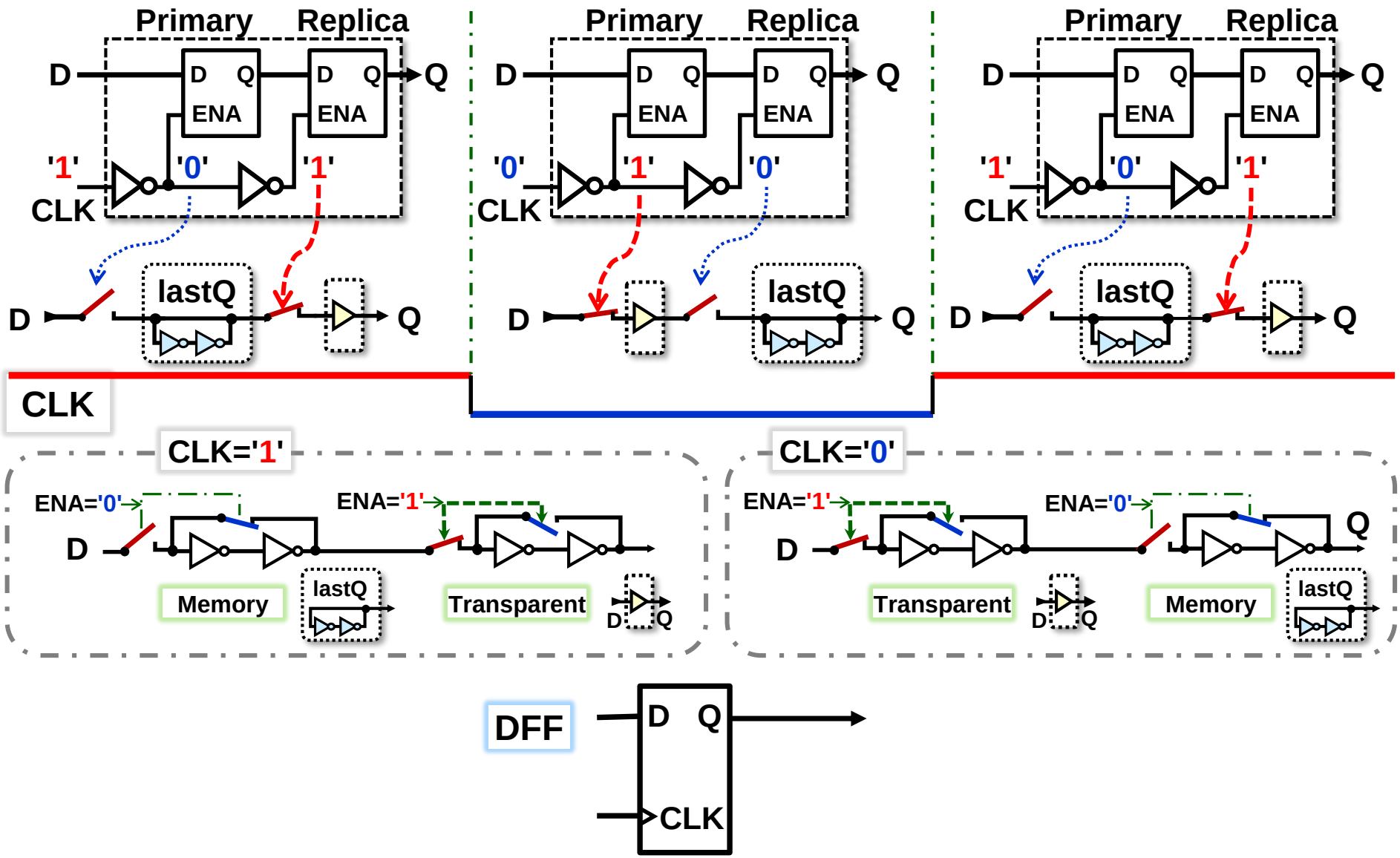
We will use English terminology of circuit development tools:

- **Latch** defines only asynchronous latch circuits, such as RS latch and D-latch.
- ***Transparent* latch** - although it is a precise technical term, it is more often written as **D-latch** instead.
- **Edge-triggered latch** includes a whole category of latch circuits that change their output only when an edge of a clock signal arrives.
- ***Flip-flop*** - (original term for a back flip, 180-degree turn)
  = In circuits, it denotes the most commonly used implementation known as *edge-triggered* latch, with the established abbreviation DFF, **Data Flip-Flop**.

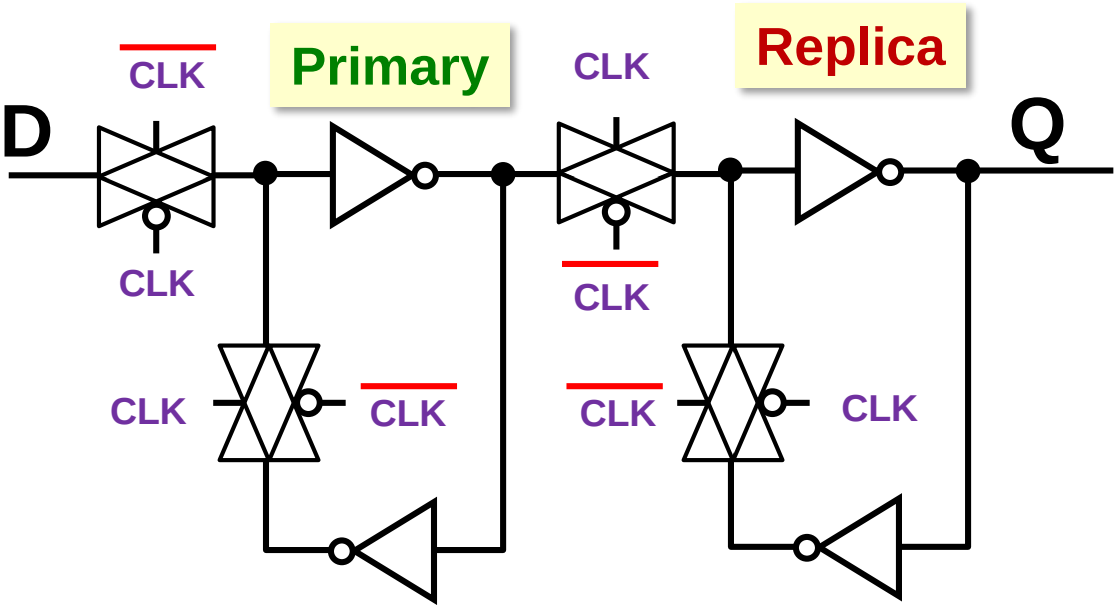
For more details see the textbook Logic Circuits, chapter 7.1 on page 94

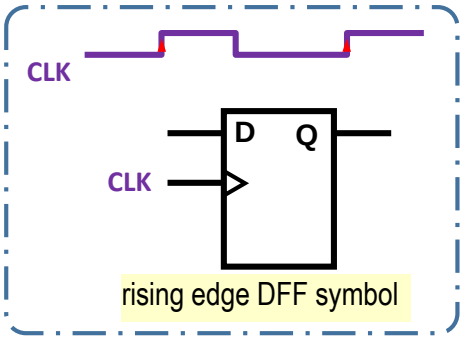
Note: Czech language does not distinguish between latches and gates.

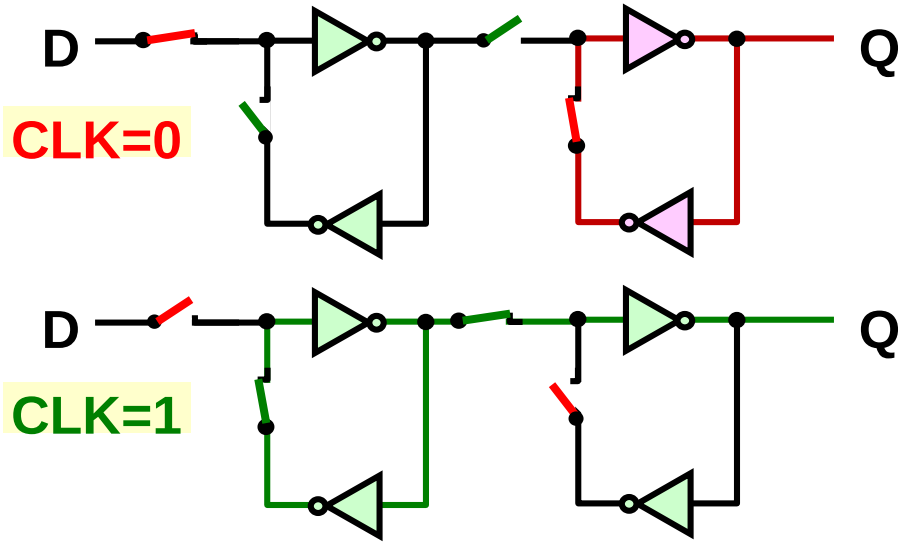
Primary-Replica DFF



Actual wiring Primary - Replica

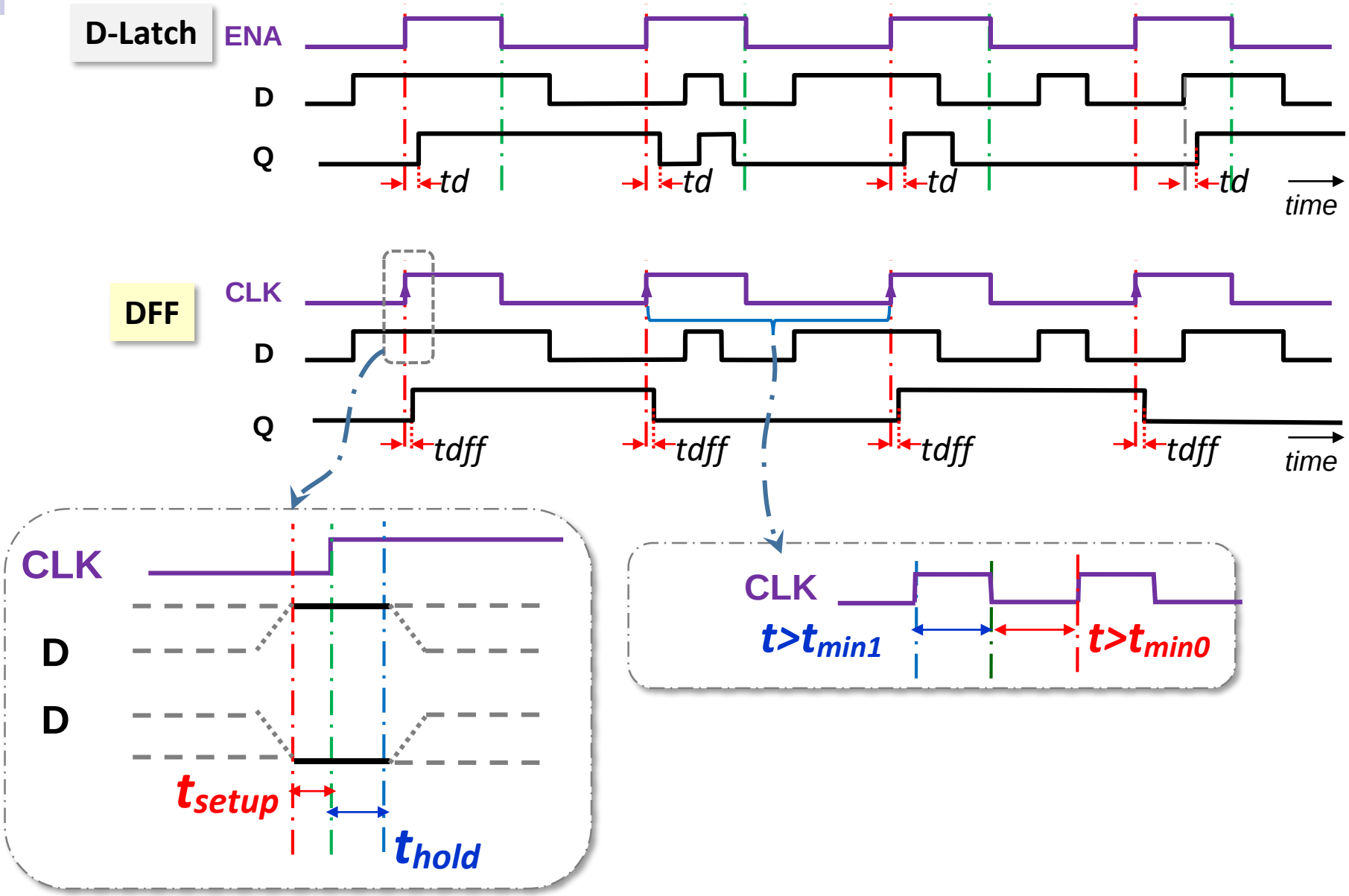


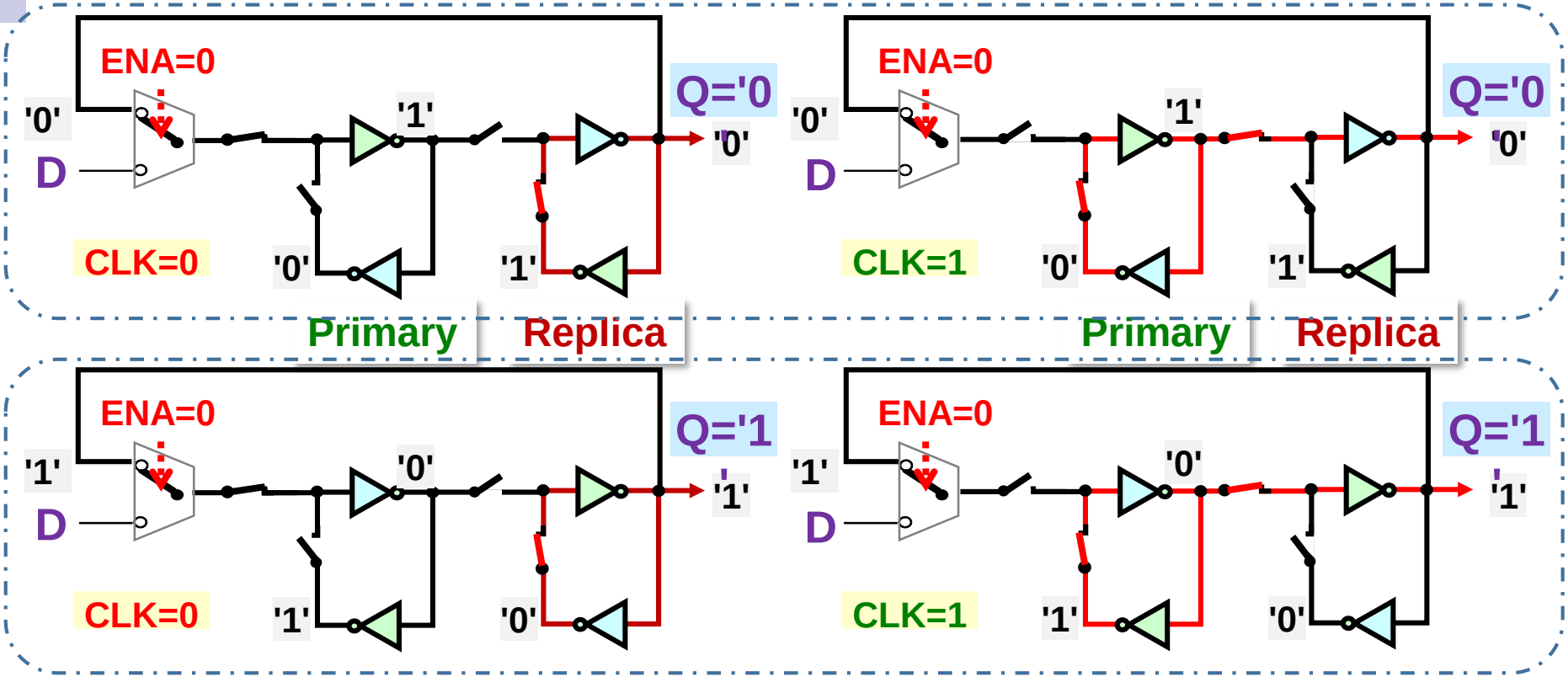


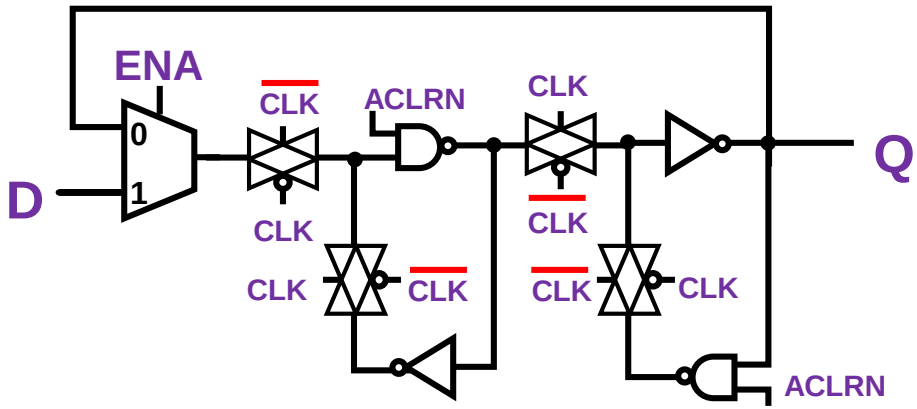


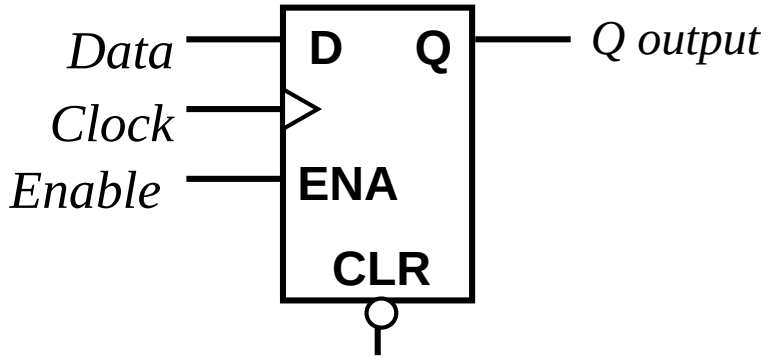
By alternating the clocks, the data flow is interrupted in a similar way to the transition chamber, only one door opens.

DFF activity chart



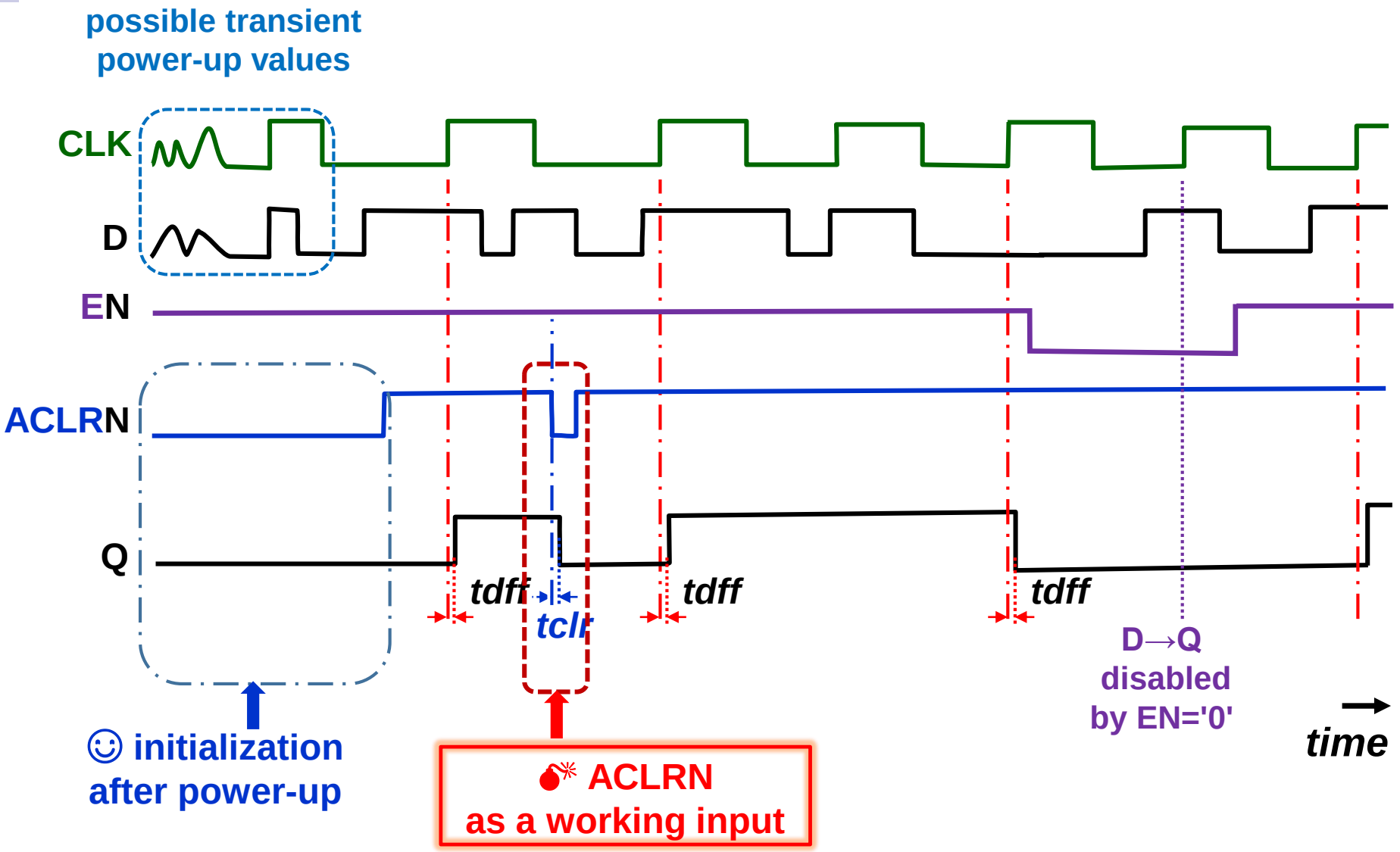






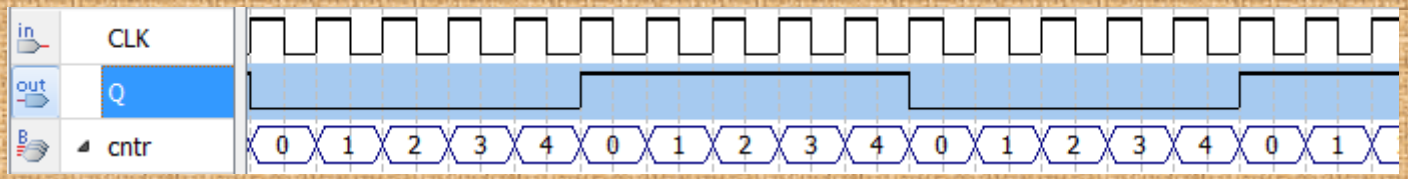
Asynchronous clear (negative logic) after power-up

DFFE activity



Divider of ten

with 50% duty cycle)



Counters and frequency dividers

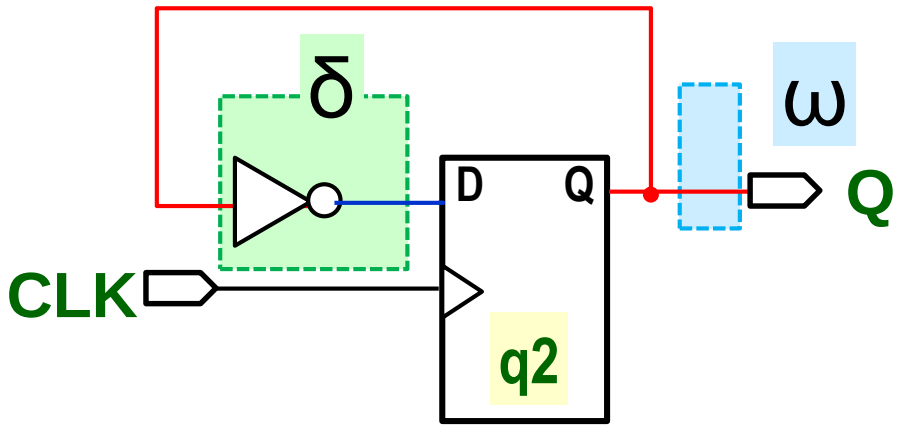
- Let us denote the clock signal frequency at the CLK input as f_{clk}.
- The highest active bit of the counter logically changes from ´0´ to ´1´ and from ´1´ to ´0´. Let us denote the frequency of this signal as f_{out}.
- If the binary counter counts by 1's in the range 0 to M-1, where M is any natural number greater than 1, then the counter also divides the input frequency by M:

$$f_{out} = \frac{f_{clk}}{M}$$

In the literature, the word divider (frequency) is sometimes used as a synonym for a binary counter in increments of +1, since its design is virtually identical. The two circuits differ only in the resulting application and the range of functions. Divisors typically only count up and have only a 1-bit output.

1-bit counter / Divider 2 (1/2)

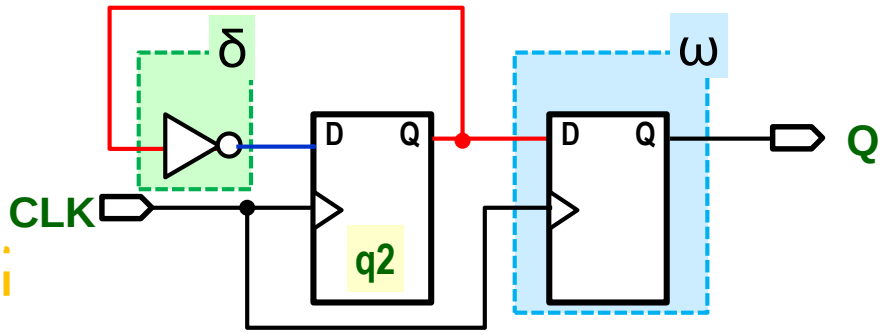
The output will be 0, 1, 0, 1... The δ function of the next state is just an inverter.




```
library ieee;
use ieee.std_logic_1164.all;
entity DivBy2 is
 port (CLK : in std_logic;
       Q : out std_logic);
end entity;
architecture rtl1 of DivBy2 is
begin
 process(CLK)
 variable q2 : std_logic:='0'; -- simulations require initialization!
 begin
    if rising_edge(CLK) then
       q2:=not q2; -- the delta function - the next value = 1 bit adder +1
    end if;
    Q<=q2; -- the omega function is only a wire
 end process;
end architecture;
```

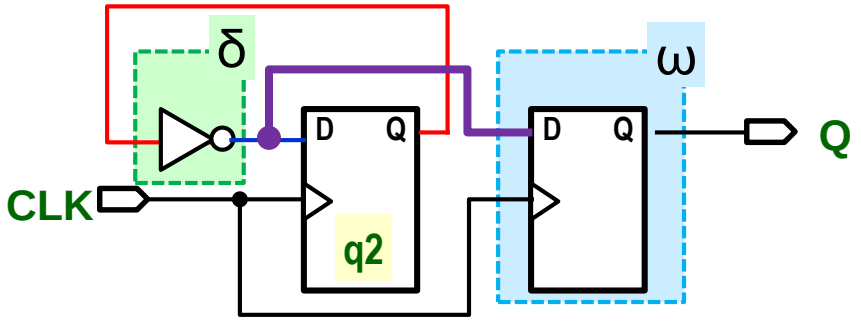
1-bit counter / Divider 2 (2/2)

```
architecture rtl2reg of DivBy2 is begin
    process(CLK)
    variable q2 : std_logic:='0';
    begin
        if rising_edge(CLK) then
            Q<=q2; -- here, omega adds 1 register
            q2:=not q2; -- the delta function
        end if;
    end process;
end architecture;
```



The placement of omega section affects the circuit!

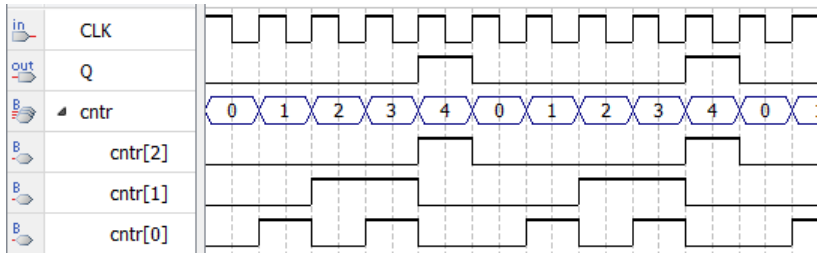
```
architecture rtl3wrong of DivBy2 is begin
    process(CLK)
    variable q2 : std_logic:='0';
    begin
        if rising_edge(CLK) then
            q2:=not q2; -- the delta function
            Q<=q2; -- now, omega uses the next value
        end if;
    end process;
end architecture;
```

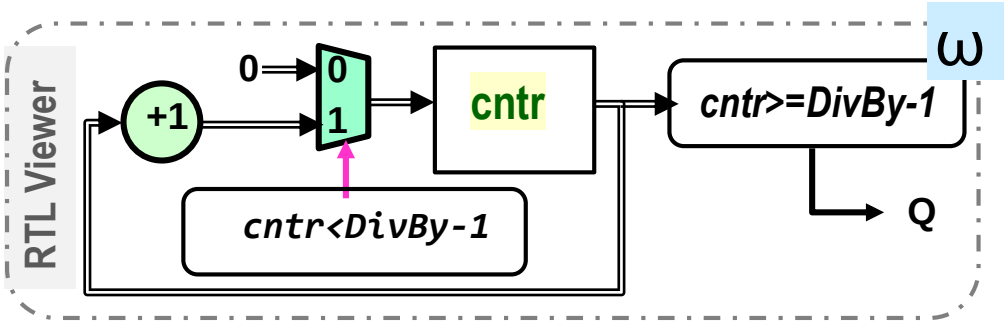


Counter 5 / Divider 5 (1/2)

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity DivBy5 is
 port ( CLK : in std_logic; Q : out std_logic);
 end entity;
architecture rtl1 of DivBy5 is
begin

 process(CLK)
 constant DivBy: integer:=5;
 variable cntr : integer range 0 to DivBy-1:=0;
 begin
  if rising_edge(CLK) then
     if cntr<DivBy-1 then cntr:=cntr+1; else cntr:=0;
 end if;
  end if;
  if cntr<DivBy-1 then Q<='0'; else Q<='1';
  end if;
 end process;
end architecture;
```



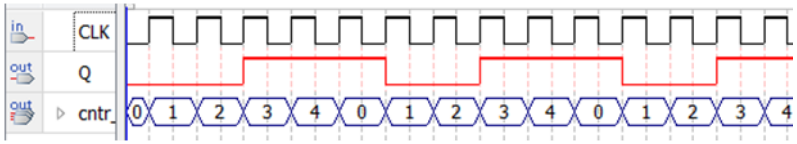


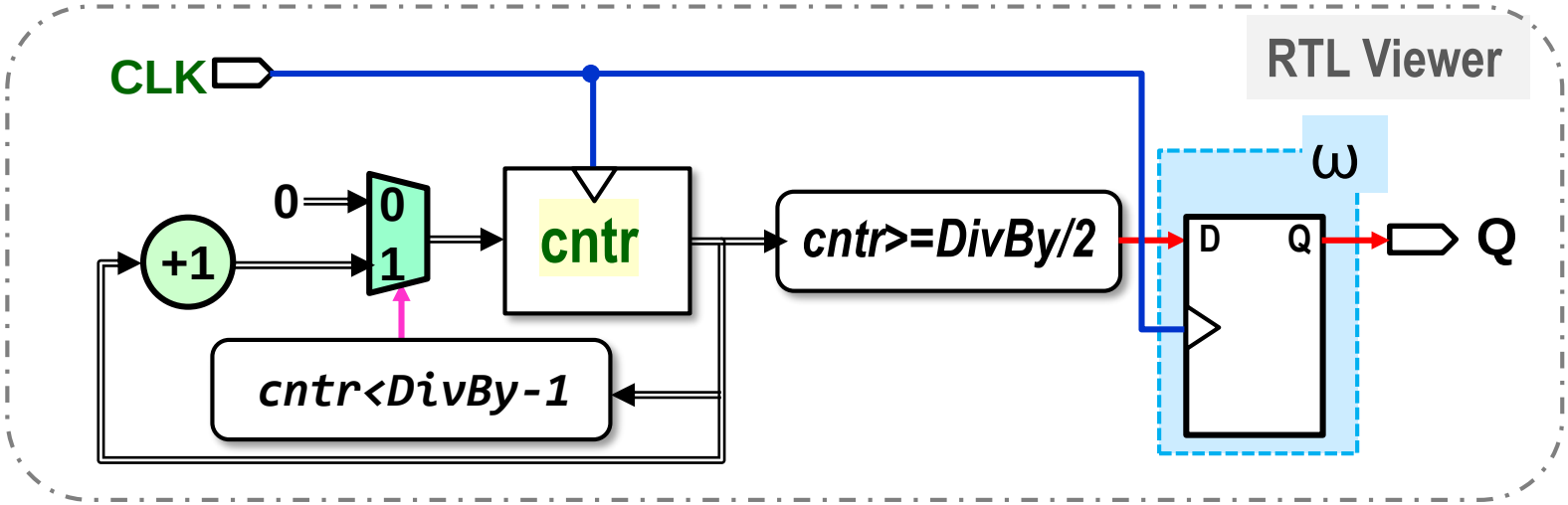
δ

ω - implementation?
It delays Q output and generate glitches.

Counter 5 (2 of 2)

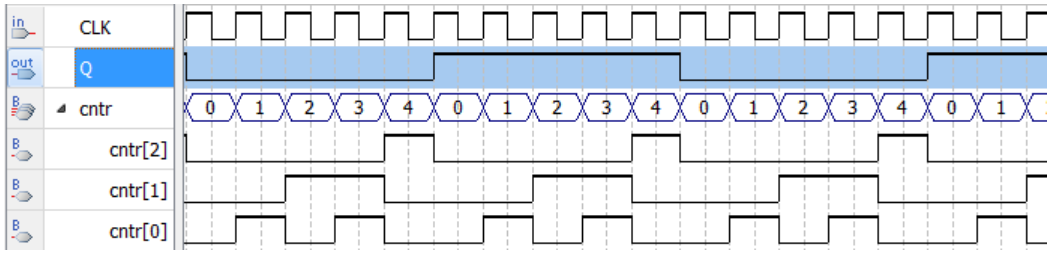
```
architecture rtl3universal of DivBy5 is
begin
      process(CLK)
      constant DivBy: integer:=5;
      variable cntr : integer range 0 to DivBy-1:=0;
      begin
           if rising_edge(CLK) then
                if cntr<DivBy/2 then Q<='0'; else Q<='1'; end if; -- more robust ω output function
                if cntr<DivBy-1 then cntr:=cntr+1; else cntr:=0; end if; -- δ function
           end if;
      end process;
end architecture;
```

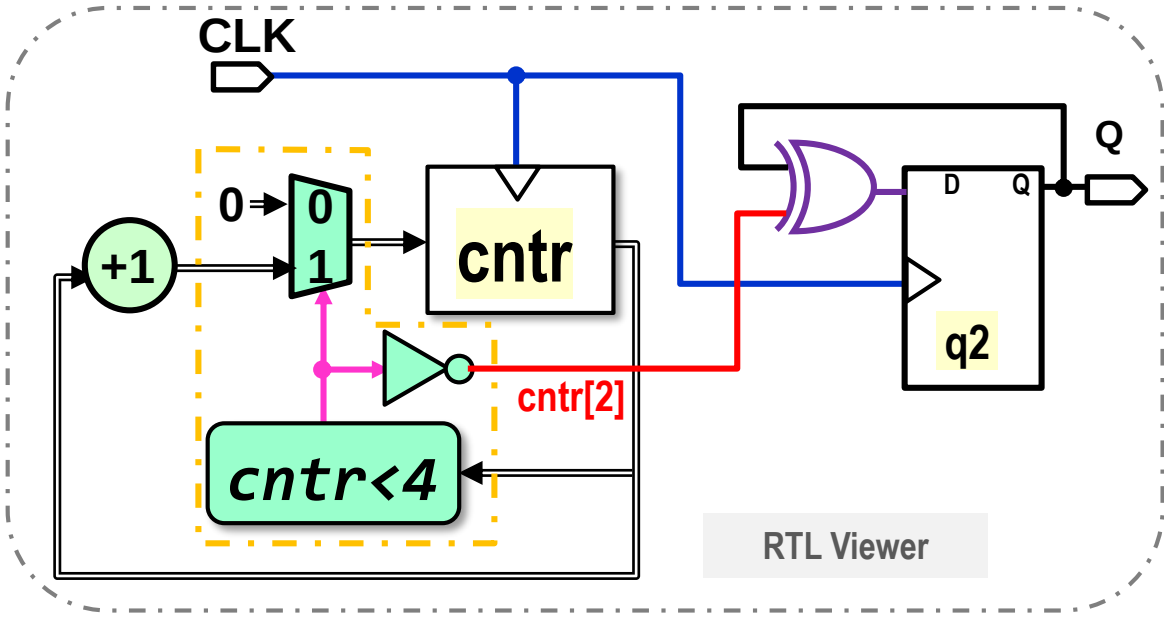




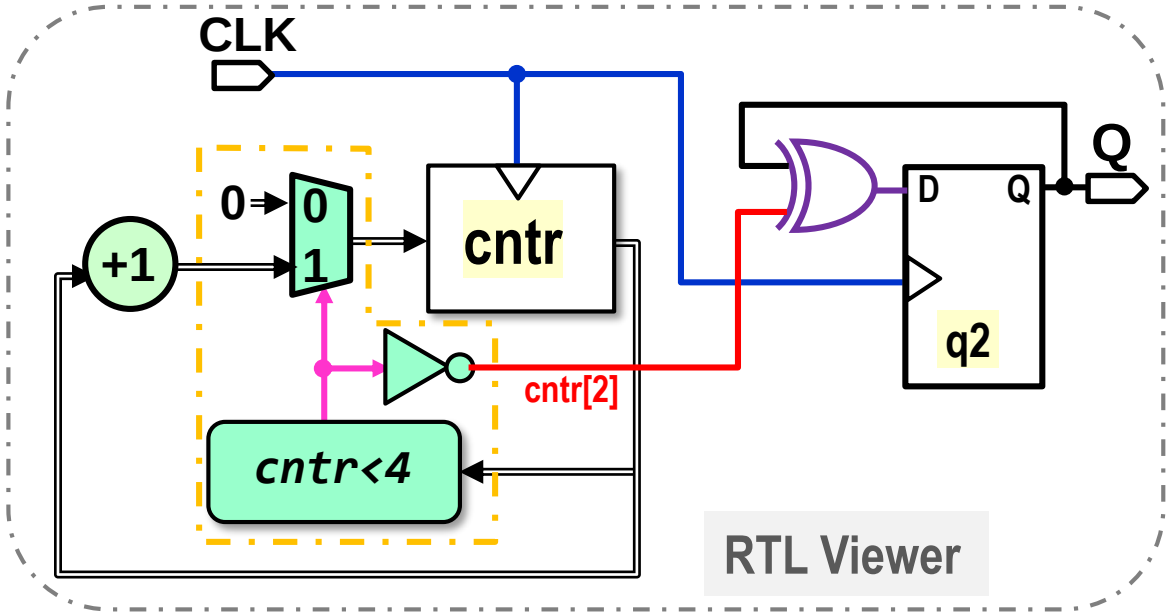
Divider by ten with 50% duty cycle

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity DivBy10 is
   port ( CLK : in std_logic; Q : out std_logic);
end entity;
architecture rtl1 of DivBy10 is
begin
   process(CLK)
   constant DivBy: integer:=5;
   variable cntr : integer range 0 to DivBy-1:=0;
   variable q2 : std_logic:='0';
   begin
      if rising_edge(CLK) then
         if cntr <DivBy-1
              then cntr:=cntr+1;
              else cntr:=0;
                   q2:=not q2; -- DivBy2
         end if;
      end if;
      Q<=q2;-- the omega - only a wire
   end process;
end architecture;
```

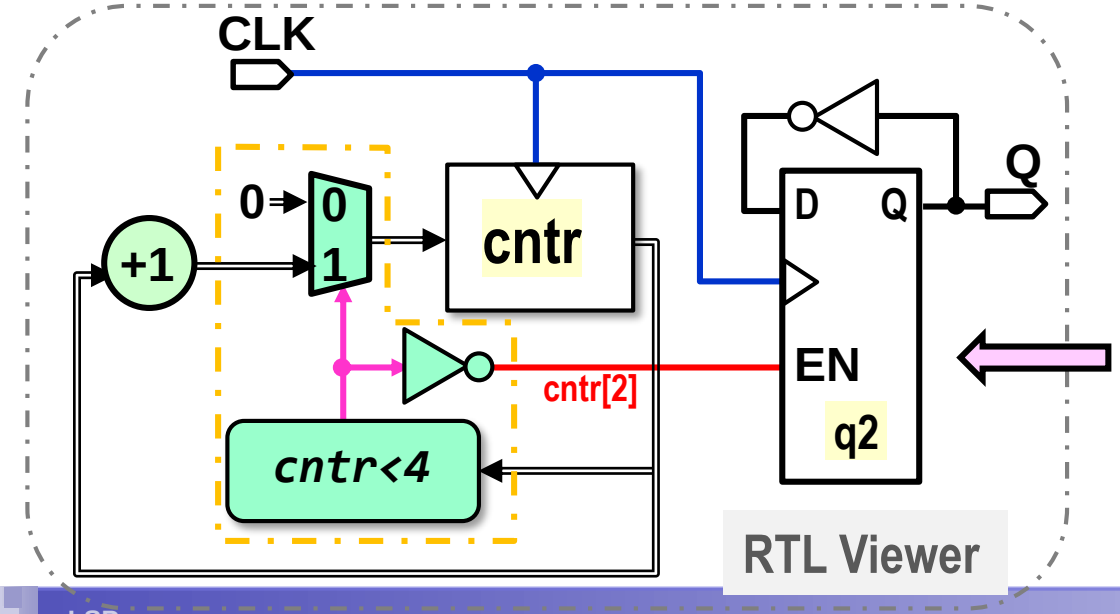




Divider 10 without and with Enable



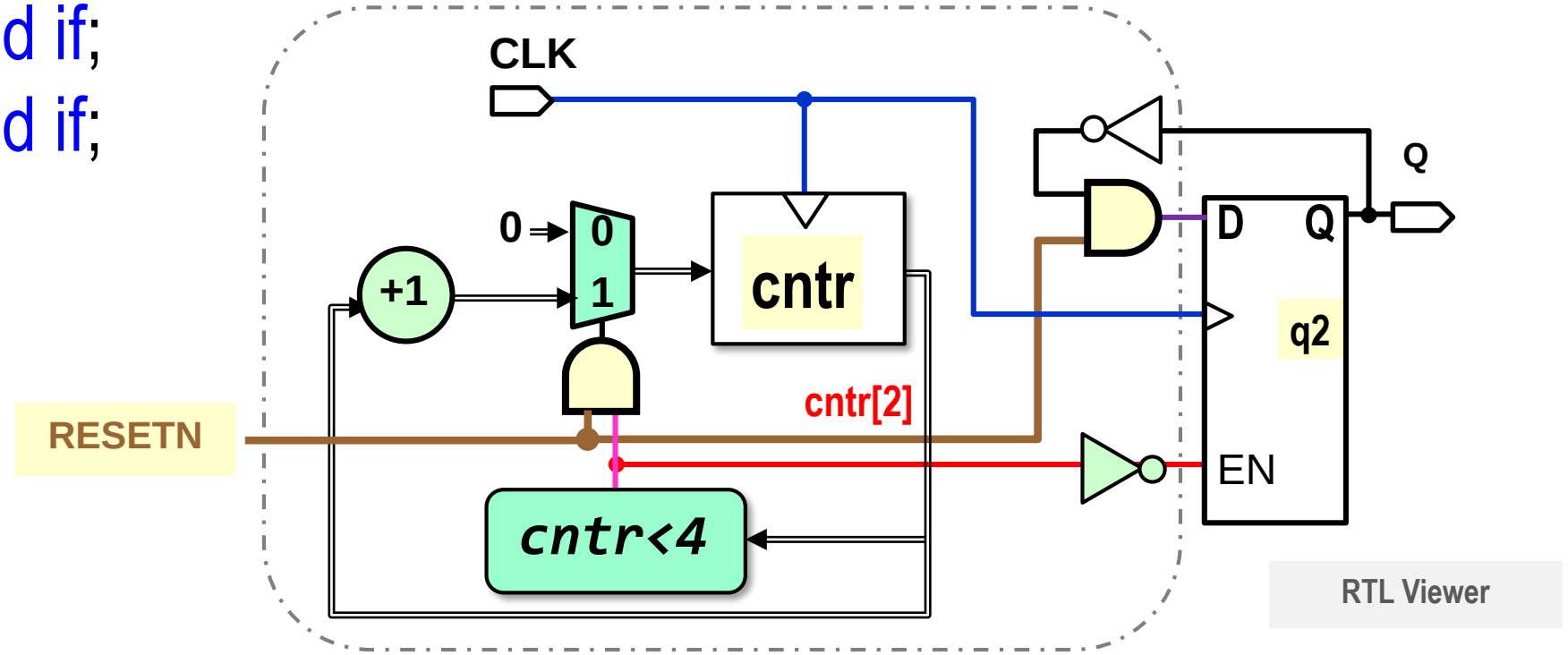
Both solutions are completely equivalent. The compiler selects the one it considers more appropriate after considering the necessary couplings and other parameters.



EN = ENABLE
But how do we plug it in?
We don't have to, the Cyclone FPGA family DFF has it.

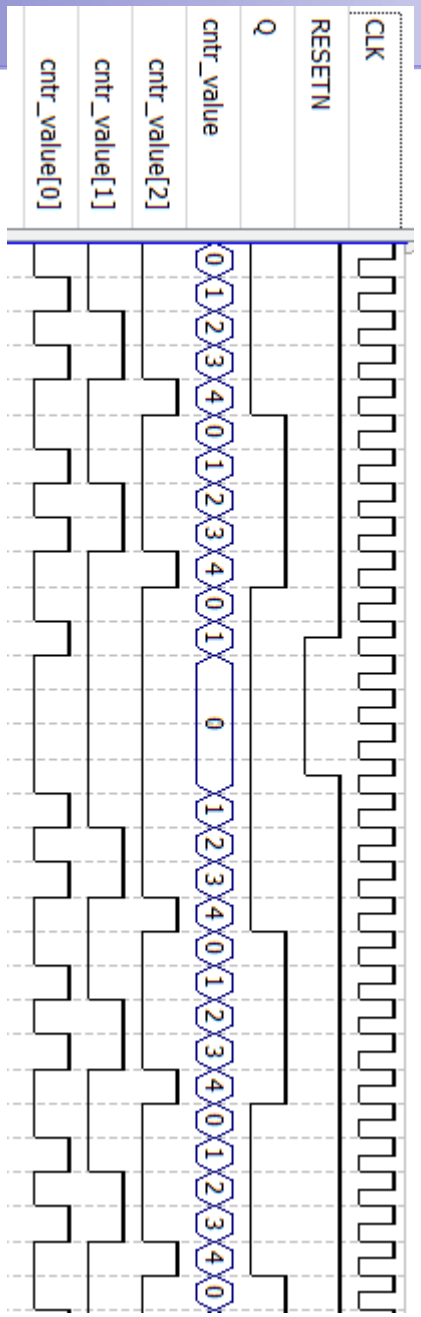
Adding synchronous RESETN

```
port ( CLK, RESETN : in std_logic; Q : out std_logic);
-- ...
if rising_edge(CLK) then
  if cntr <DivBy-1 and RESETN='1'
  then cntr:=cntr+1; else cntr:=0; q2:=not q2 and RESETN;
end if;
end if;
```



Full code

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity DivBy10Resetn is
   port ( CLK, RESETN : in std_logic; Q : out std_logic);
end entity;
architecture rtl1 of DivBy10Resetn is
begin
   process(CLK)
   constant DivBy: integer:=5;
   variable cntr : integer range 0 to DivBy-1:=0;
   variable q2 : std_logic:='0';
   begin
      if rising_edge(CLK) then
         if cntr <DivBy-1 and RESETN='1'
              then cntr:=cntr+1;
              else cntr:=0;
                   q2:=not q2 and RESETN; -- DivBy2
         end if;
      end if;
      Q<=q2;-- the omega - only a wire
   end process;
end architecture;
```

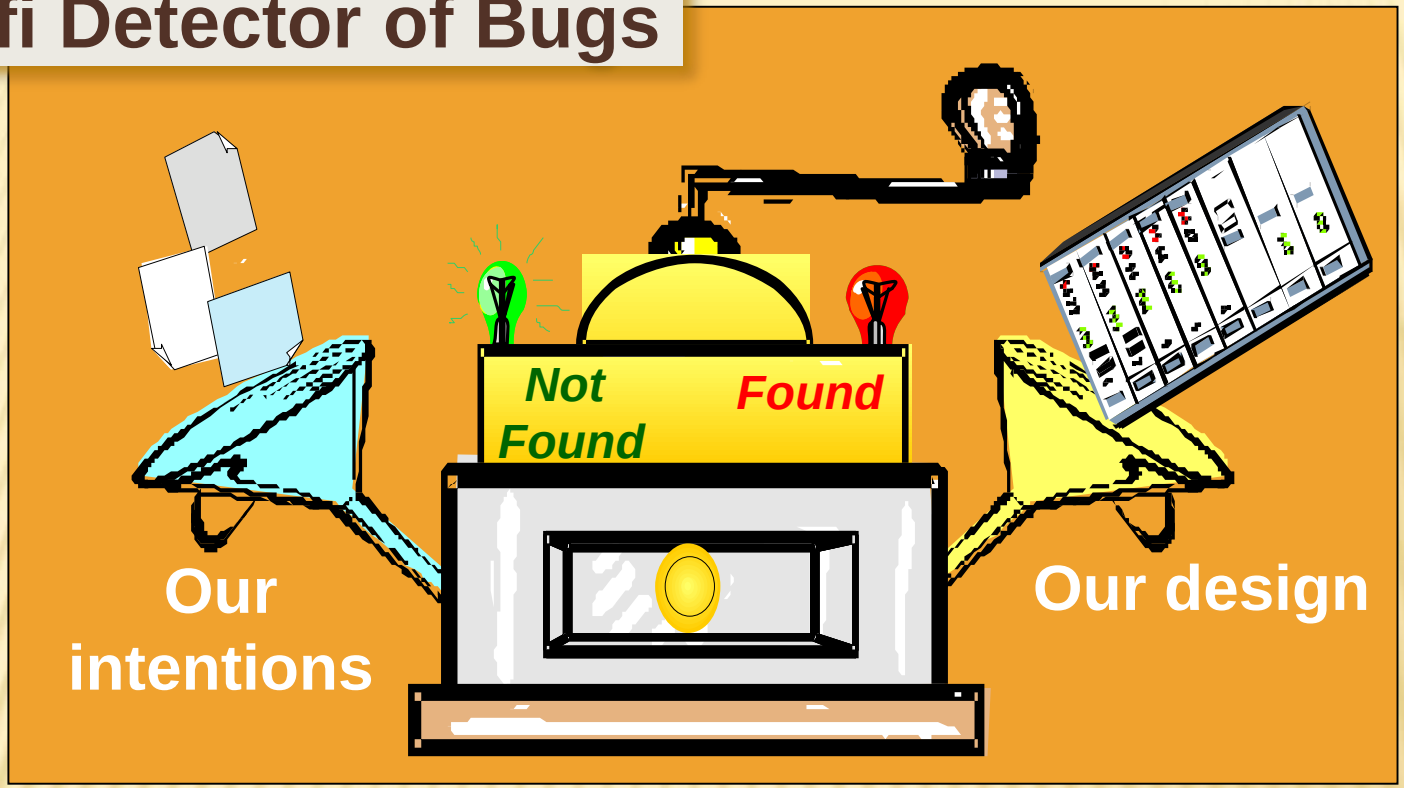


WE'RE TESTING OUR DESIGN...

by uploading it to the technology

Vladimít Wagner, Steel City, 2010

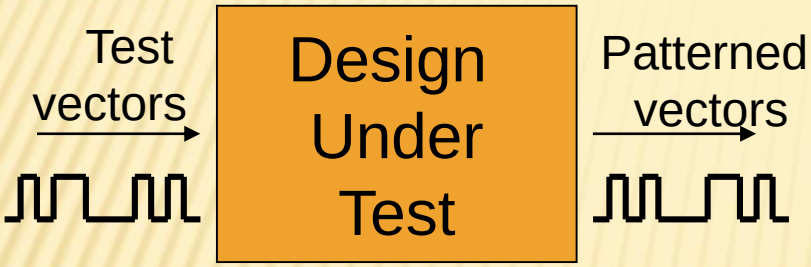
Sci-fi Detector of Bugs



Testing our design by

TESTBENCHES IN VHDL

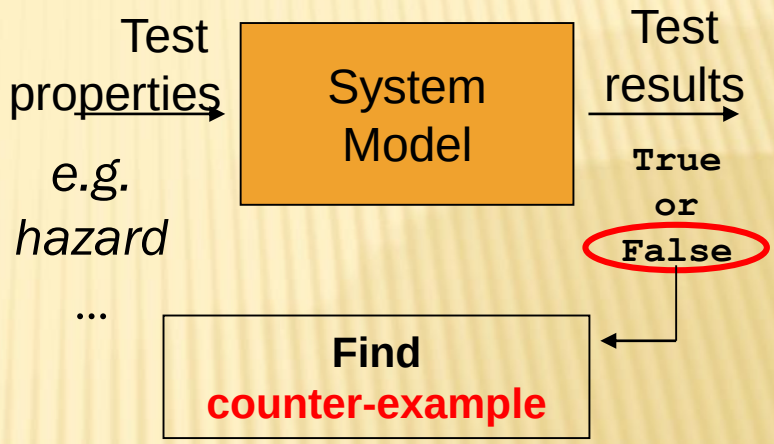
Testing & Simulation



- Cannot cover all possible cases
- Possibility of surviving subtle (corner case) bugs

- It is always possible

Formal Verification



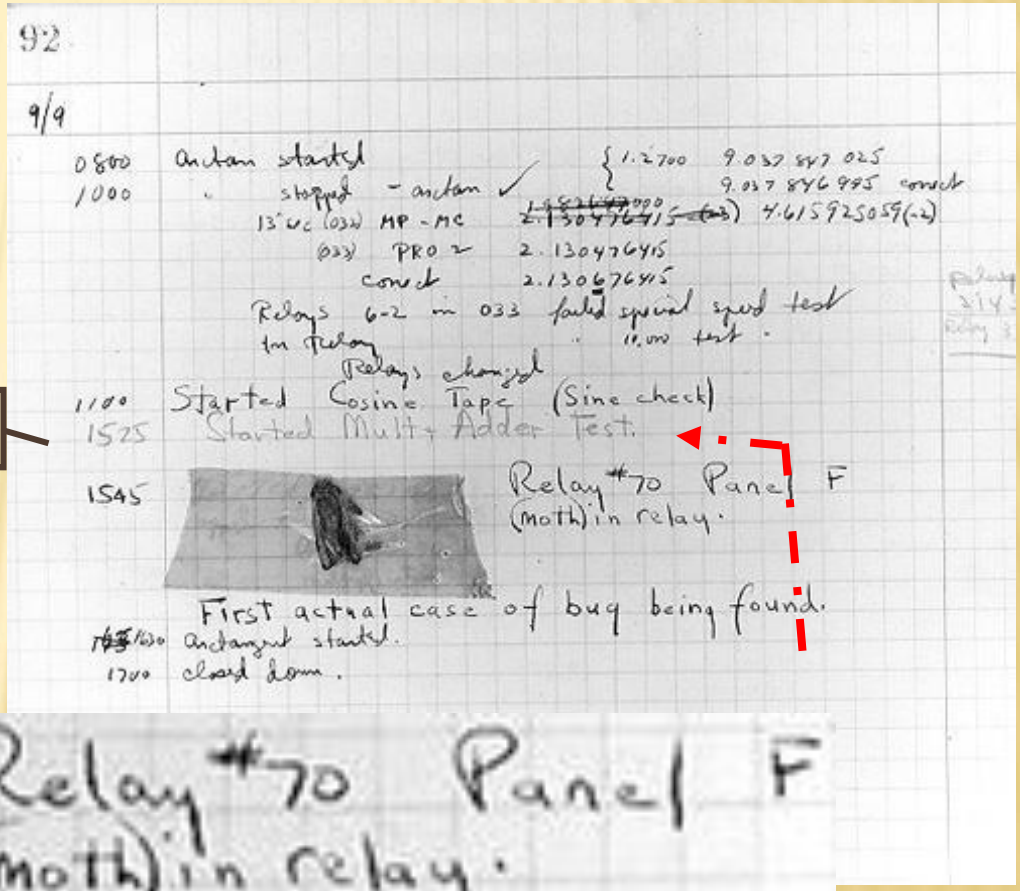
- Equivalent to simulating all cases in simulation
- No **bug** according to the specified property

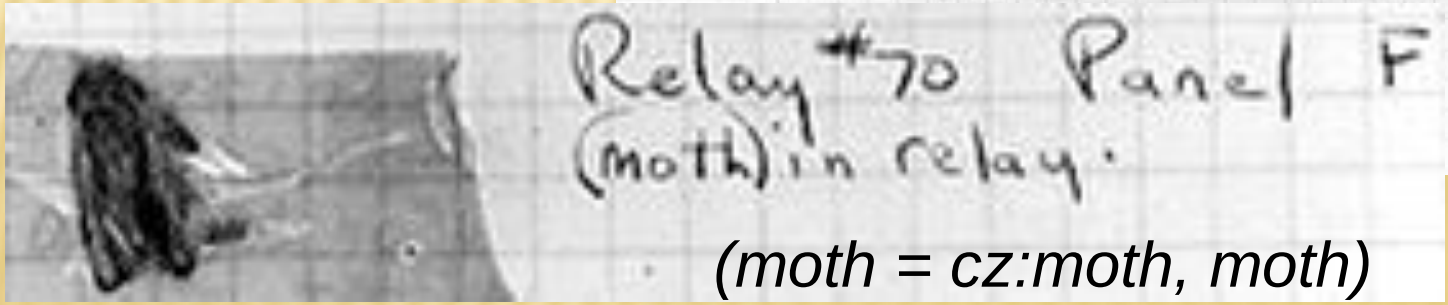
- Sometimes it is possible

TECH URBAN LEGENDS

- The first "Bug" of the modern era is traditionally attributed to Grace Murray Hopper for finding a bug in the contacts of the Mark I computer based on relay logic.

Log!

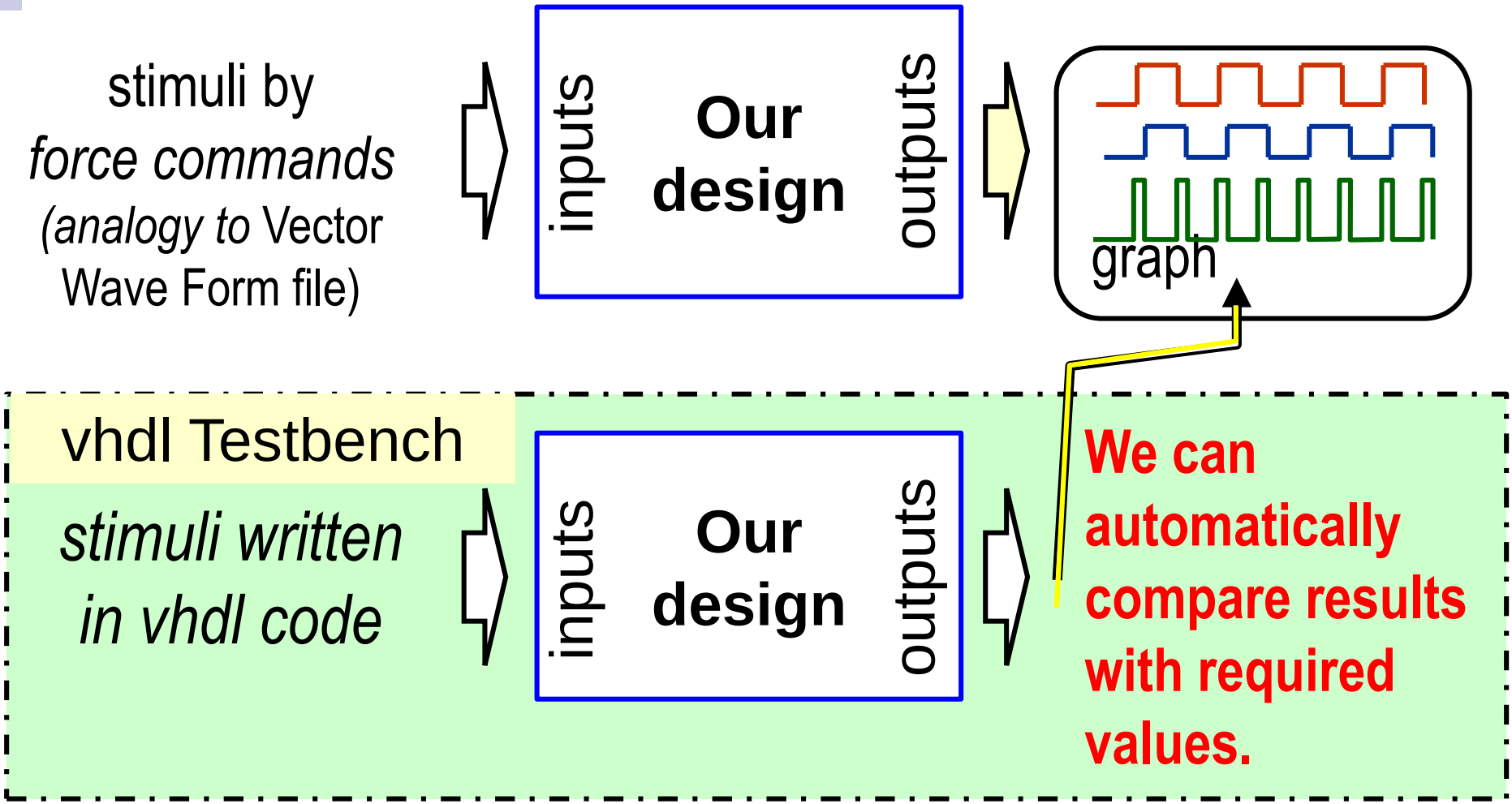




(moth = cz:moth, moth)

Simulation in VHDL

Testbench



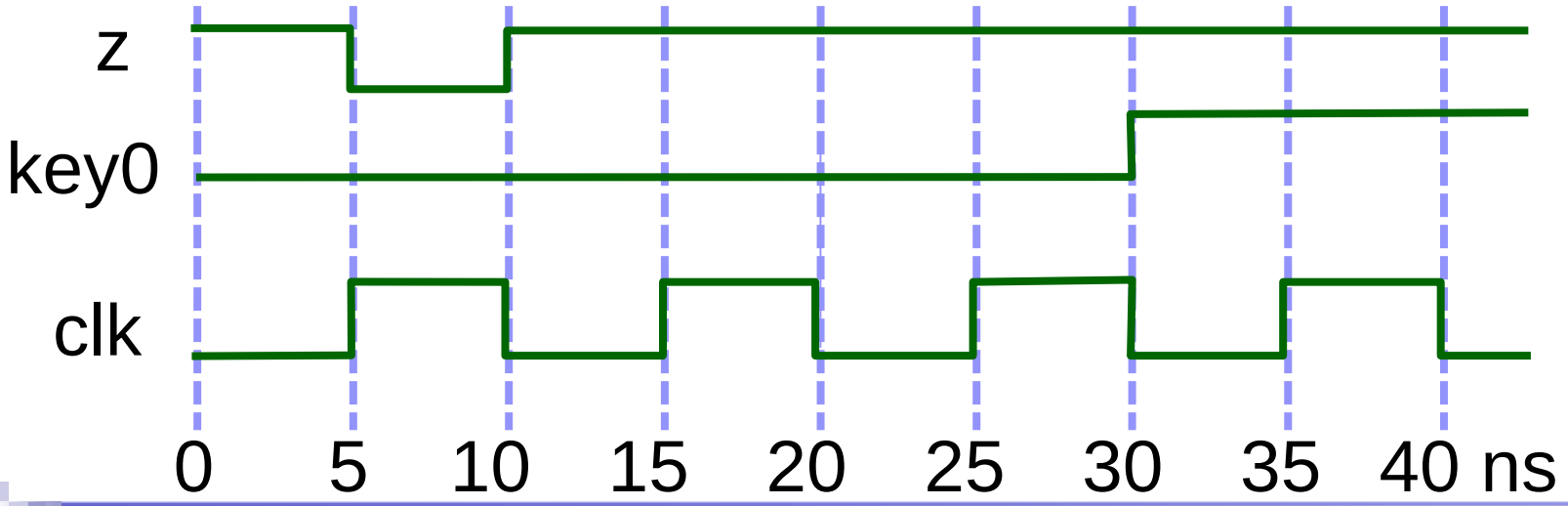
stimulus, pl. stimuli (from Latin stilus, stylus->style) - something that rouses the mind or spirits or incites to activity - en: stimulus, stimulus

The after command in the simulation

```
signal clk : std_logic:='0';
signal z, key0 : std_logic;
```

Note: Initialized signals (wires) are suitable for simulation only !!! Their synthesis is very difficult. Quartus often ignores a shows warning.

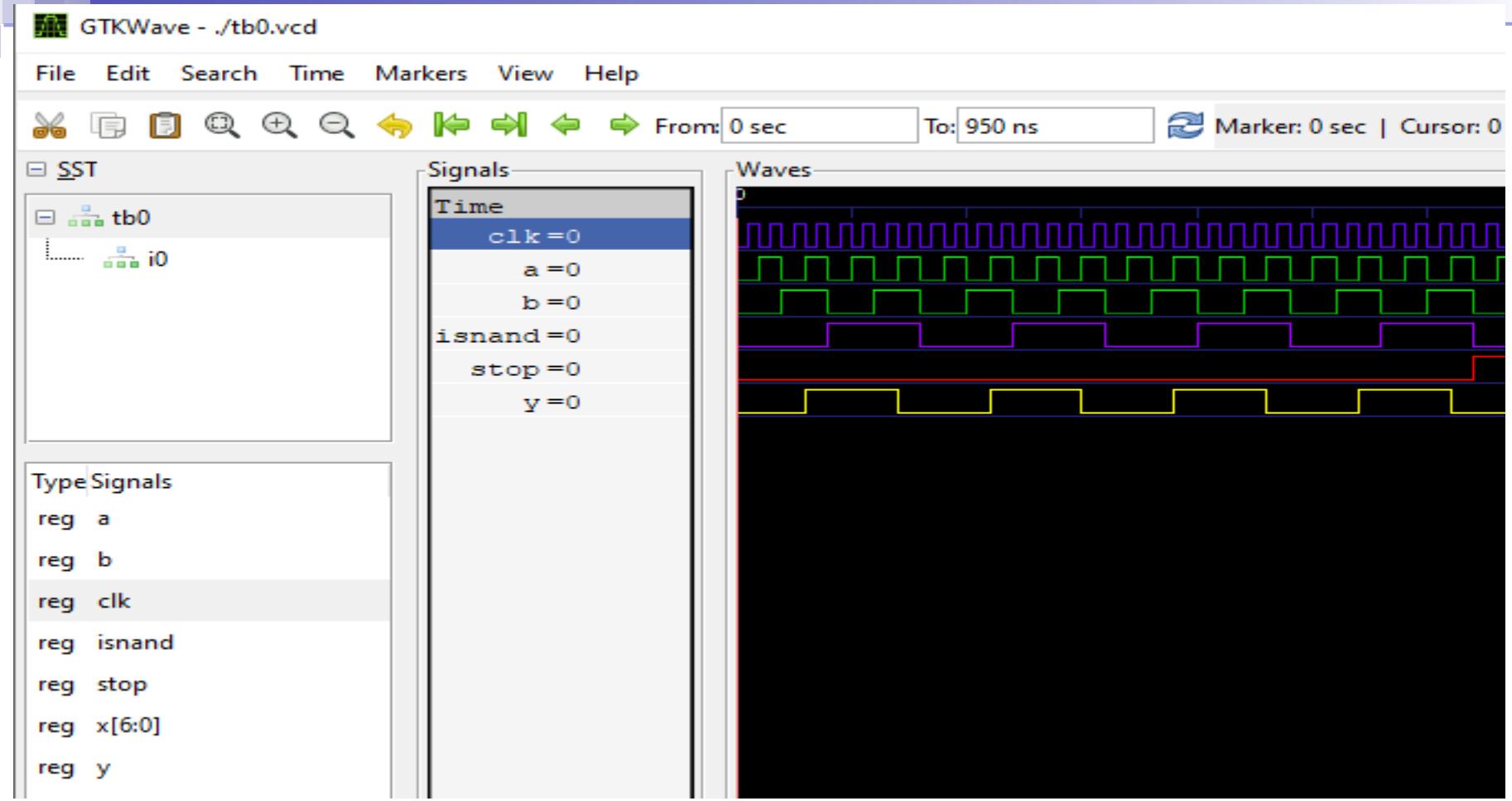
```
z <= '1', '0' after 5 ns, '1' after 10 ns;
key0 <= '0', '1' after 30 ns; -- initialization ...
clk <= not clk after 5 ns; -- clock
```



Pat-Mat

```
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 arch0 of n2and is
begin y <= a and b when isNand='0' else not (a and b);
end architecture; ------------------------------------------------------------
architecture arch1 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; ------------------------------------------------------------
architecture arch2 of n2and is
signal tmp : std_logic;
begin process(a,b, isNand, tmp)
        begin tmp<=a and b; if isNand='1' then tmp<=not tmp; end if;
                y<=tmp;
       end process;
end architecture; ------------------------------------------------------------
```

Let's sketch the waveforms




```
signal x: unsigned(6 downto 0):=(others=>'0');
signal clk, a, b, isNand, stop, y : std_logic:='0';
begin -- architecture
  clk<=x(0); a<=x(1); b<=x(2); isNand<=x(3); stop<=x(6); x<= x+1 after 10 ns;
```

Testbench

```
library ieee; use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity tb0 is end entity; -- entity for testbench does not contain
declarations
architecture testbench0 of tb0 is
signal x: unsigned(6 downto 0):=(others=>'0');
signal a, b, isNand, stop : std_logic:='0';
signal y : std_logic:='0';
begin ------------------------------------------------------------
        a<=x(1); b<=x(2); isNand<=x(3); stop<=x(6);
       x<= x+1 after 10 ns;
i0 : entity work.n2and(arch0) port map(a, b, isNand, y);
    assert stop/='1' report LF&LF&":-) Done!"&LF severity failure;
end architecture;
```