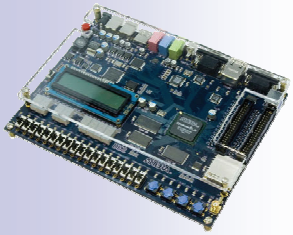
Struktury počítačových systémů

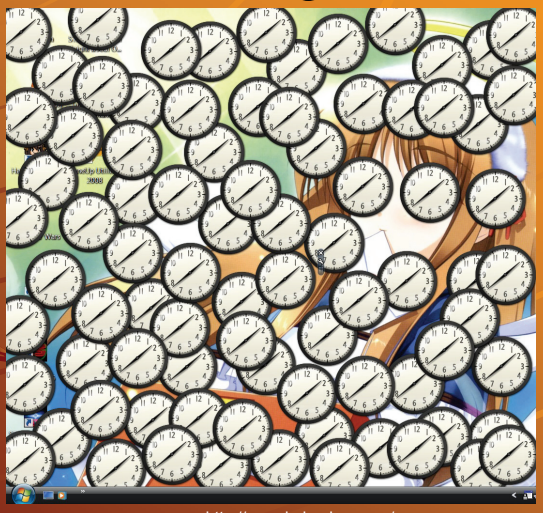


Přednáší: Richard Šusta

27. října 2011 verze 1.0

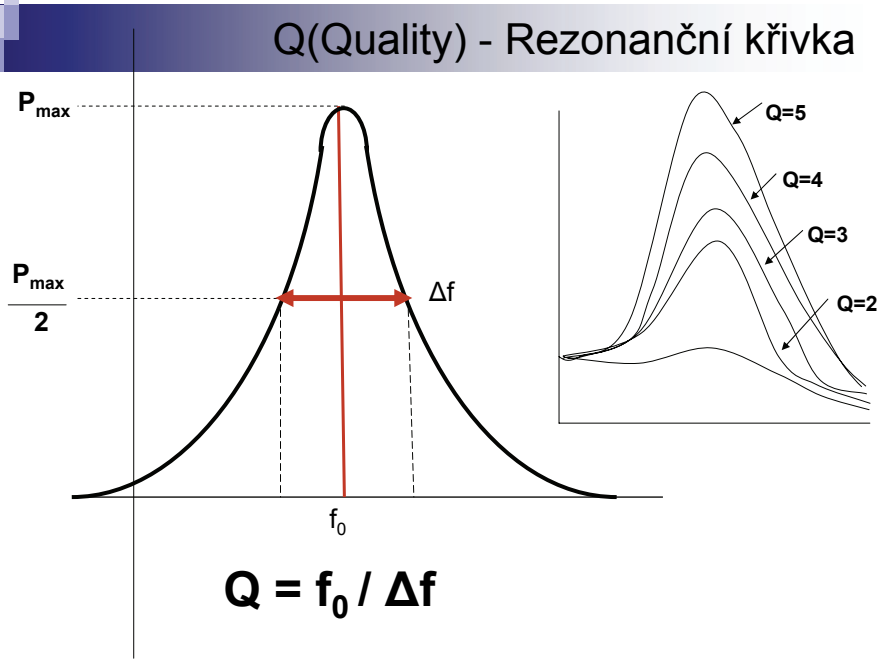
ČVUT-FEL v Praze – kód předmětu A0B35SPS

Overclocking Clocks



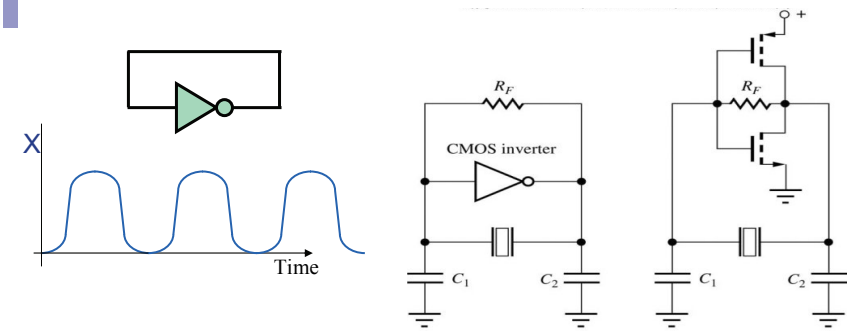
http://www.beige-box.com/

Q(Quality) - Rezonanční křivka



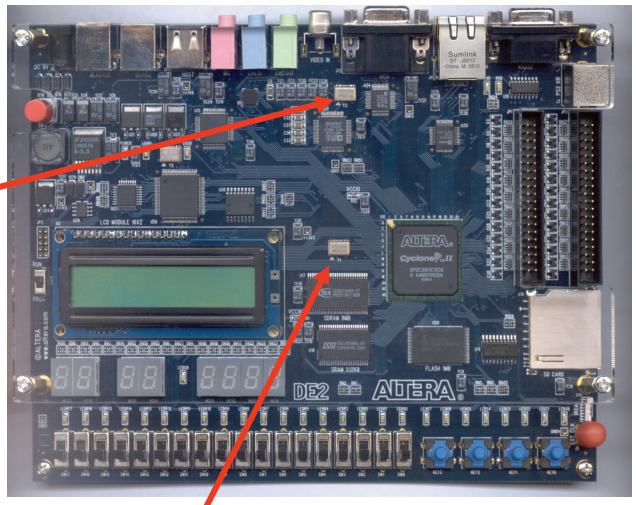
$$Q = f_0 / \Delta f$$

Crystal Oscillator



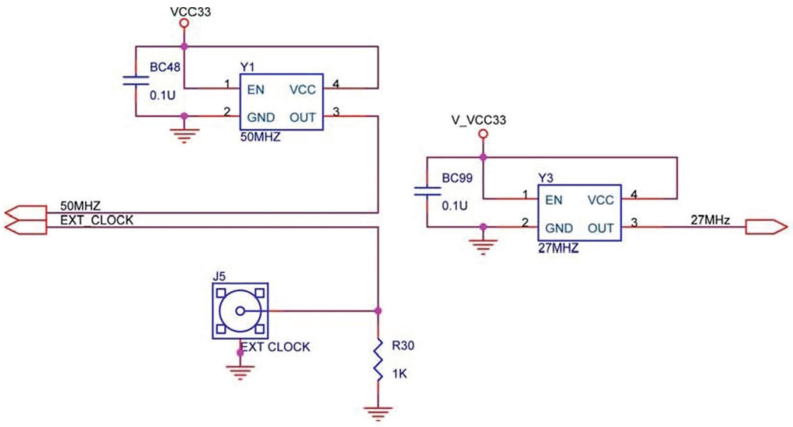
- Working Range **50kHz - 80 MHz**
  - but high frequencies can be based on high overtones!
- Temperature Coefficient k=10 [Hz/MHz/C], $f = k\, f_0\, \Delta T$ (C)
- High Q (1000 – 10,000)
- Slow startup 5-20 ms in MHz range, seconds in kilohertz range

DE2

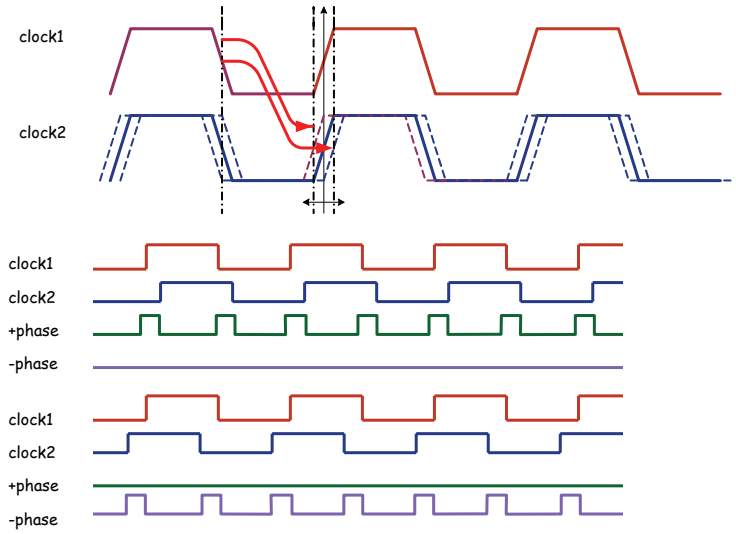


DE oscillators

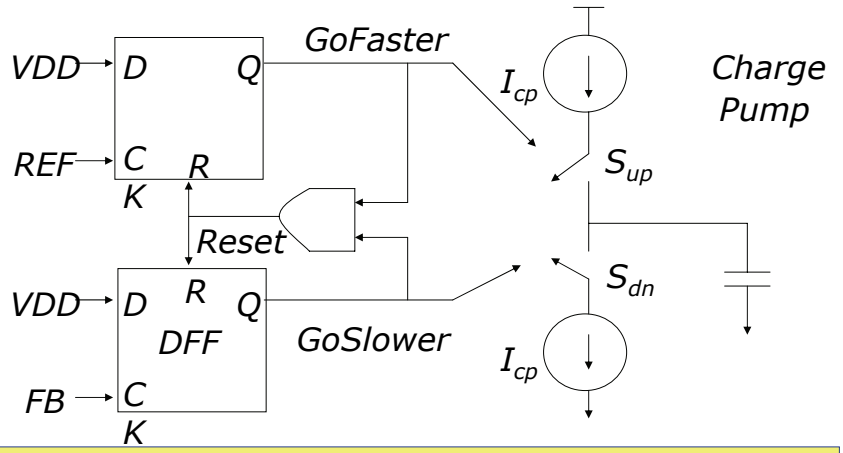
- CLOCK_27 - 27 MHz clock input (with TD_RESET <= '1';)
- CLOCK_50 - 50 MHz clock input



Phase Shift and Detector

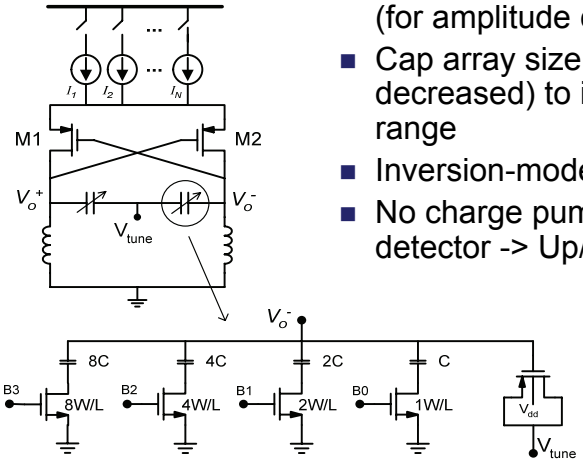


Phase Detector with Charge Pump



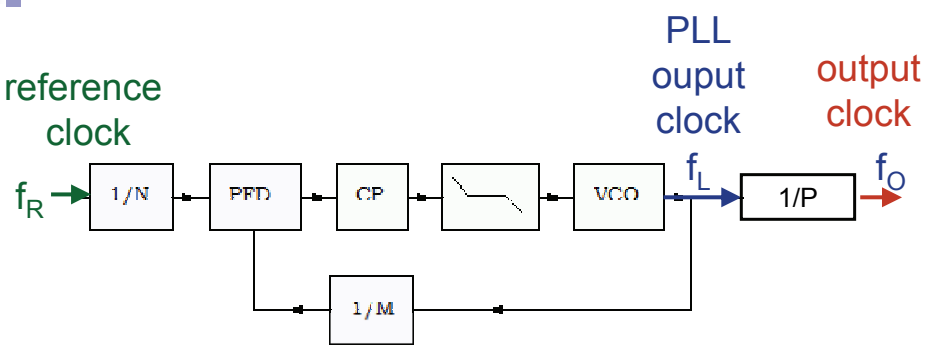
Eng: Charge pump has 2 strictly different meanings it is a part of phase detector in PLL and also a DC converter (cz:kaskádní násobič)

Digital VCO Implementation



- Tail current is digitally controlled (for amplitude calibration)
- Cap array size increased (L decreased) to improve tuning range
- Inversion-mode MOS varactor
- No charge pump: Phase detector -> Up/Down Counter

Basic Block Diagram

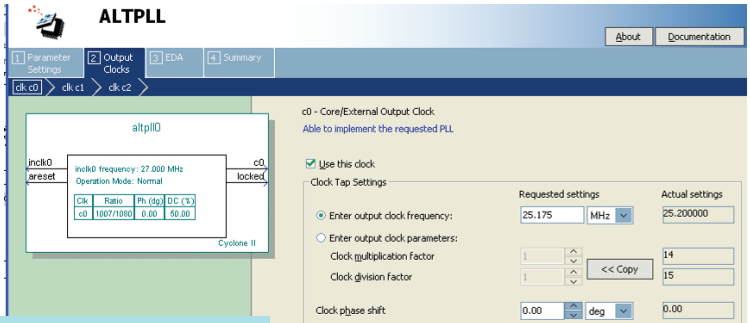


$f_L/M = f_R/N$

$f_L = f_R * (M/N)$

$f_o = f_R * (M/(N * P))$

PLL 27 MHz -> 25,175 Mhz



25,175MHz

25,2 MHz -> error < **0.1%**
25MHz -> error = -0.7%

27MHz crystal:
10[*Hz/C/MHz*] *27[*MHz*]
* 60[*dC°<-20-80C°*] =**+/-0,06%**

Example:
Priority Encoder 16:16 bits

Truth Table

```
Inputs x(15 downto 0):outputs y(15 downto 0)
1--- ---- ---- ----  :  1000 0000 0000 0000
01-- ---- ---- ----  :  0100 0000 0000 0000
001- ---- ---- ----  :  0010 0000 0000 0000
0001 ---- ---- ----  :  0001 0000 0000 0000
                     ***
0000 0000 0000 1---  :  0000 0000 0000 1000
0000 0000 0000 01--  :  0000 0000 0000 0100
0000 0000 0000 001-  :  0000 0000 0000 0010
0000 0000 0000 0001  :  0000 0000 0000 0001
0000 0000 0000 0000  :  0000 0000 0000 0000
```

Note: Symbol - means don't care

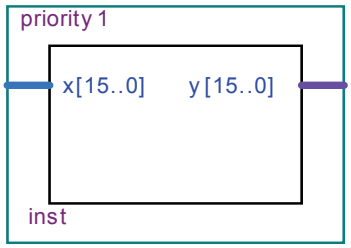
Note: Array versus std_logic_vector

```
-- library type
TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;
-- library type
TYPE MojePole IS ARRAY (7 downto 0) of std_logic;

signal reg : MojePole;
signal reg2 : std_logic_vector(7 downto 0);

...
reg2<=reg; -- nelze vzájemně převést žádným přetypováním

...
reg2(0)<=reg(0); reg2(1)<=reg(1); -- jediný způsob konverze
```

VHDL má striktně oddělené typy

Priority Encoder Entity

```
library ieee;use ieee.std_logic_1164.all;
entity priority1 is      port  (
        x  : in std_logic_vector(15 downto 0);
        y  : out std_logic_vector(15 downto 0)     );
end entity;
```



Statements used for modeling

- Concurrent statements
  - ...<=.... ;
  - with ... select
    ...<=... when ...,
    ...when ...,
    ...when others;
  - ...<= ... when ... else
    ... when ... else
    else ... ;

With ... select solution

```
architecture dataflow1 of priority1 is
begin  with x select
      y <= X"8000" when X"8000" to X"FFFF",
           X"4000" when X"4000" to X"7FFF",
           X"2000" when X"2000" to X"3FFF",
           X"1000" when X"1000" to X"1FFF",
           X"0800" when X"0800" to X"0FFF",
           X"0400" when X"0400" to X"07FF",
           X"0200" when X"0200" to X"03FF",
           X"0100" when X"0100" to X"01FF",
           X"0080" when X"0080" to X"00FF",
           X"0040" when X"0040" to X"007F",
           X"0020" when X"0020" to X"003F",
           X"0010" when X"0010" to X"001F",
           X"0008" when X"0008" to X"000F",
           X"0004" when X"0004" to X"0007",
           X"0002" when X"0002" | X"0003",
           X"0001" when X"0001",  X"0000" when others;
end dataflow1;
```

Compile and locate priority1 in Quartus RTL Viewer

Config.vhd

```
configuration config of priority1 is
   for dataflow2
   end for;
end config;
```

for switching architectures

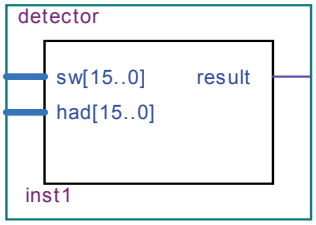
When - else solution

```
architecture dataflow2 of priority1 is
begin
    y <= X"8000" when x(15)='1' else
        X"4000" when x(14)='1' else
        X"2000" when x(13)='1' else
        X"1000" when x(12)='1' else
        X"0800" when x(11)='1' else
        X"0400" when x(10)='1' else
        X"0200" when x(9)='1' else
        X"0100" when x(8)='1' else
        X"0080" when x(7)='1' else
        X"0040" when x(6)='1' else
        X"0020" when x(5)='1' else
        X"0010" when x(4)='1' else
        X"0008" when x(3)='1' else
        X"0004" when x(2)='1' else
        X"0002" when x(1)='1' else
        X"0001" when x(0)='1' else "0000000000000000";
end dataflow2;
```

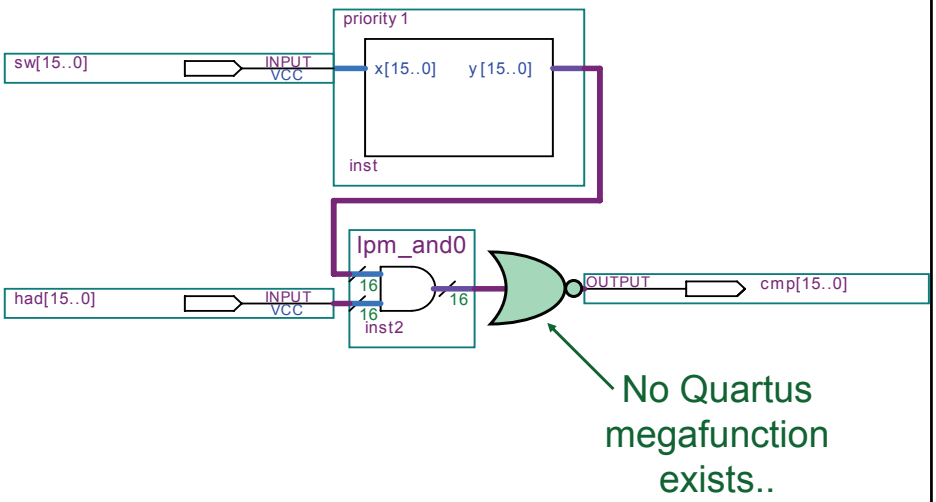
Compile and locate priority1 again in Quartus RTL Viewer

Detector for Snake

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
LIBRARY work;

ENTITY detector IS
   PORT
   (
        sw, had :  IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
        result :  OUT  STD_LOGIC
   );
END detector;
```



Overview of Detector



Convert Symbol to VHDL prototype by
Create/Update -> Create HDL design file... and edit it.

VHDL detector

```
ARCHITECTURE bdf_type OF detector IS

COMPONENT priority1
   PORT(x : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
         y : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
   );
END COMPONENT;
SIGNAL       WIRE_0 :  STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN

b2v_inst : priority1
PORT MAP(x => sw, y => WIRE_0);
  result <= '0' when (WIRE_0 and had)= X"0000" else '1';
END bdf_type;
```

Processes

for behavioral modeling

Process Format

```
[label:]  process (sensitivity list)
              declarations
          begin
              sequential statements
          end process;
```

- **Process allows conventional programming language methods to describe circuit behavior**, but the behavior need not always be synthesizable in an ASIC
- **Its behavior in a simulation**:
  1. Process statements executed once at start of simulation
  2. Process halts at "end" until an event occurs on a signal in the "sensitivity list"
  3. Process statements executed sequentially (sequential statements)

VHDL

```
architecture behav1 of priority1 is
begin
   process(x)
   variable   ix : integer;
   variable   enable : boolean;
   begin
     enable:=true; -- inicializations should be in code
     for ix in x'RANGE loop
       if( enable and (x(ix)='1') ) then
         y(ix)<='1'; enable:=false;
         else   y(ix)<='0';
       end if;
     end loop;
   end process;
end behav1;
```

Compile and locate priority1 again in Quartus RTL Viewer

Wrong inicialization in VHDL

```
architecture behav1 of priority1 is
begin
   process(x)
   variable   ix : integer;
   variable   enable : boolean:=true;
   begin
     for ix in x'RANGE loop
       if( enable and (x(ix)='1') ) then
         y(ix)<='1'; enable:=false;
         else   y(ix)<='0';
       end if;
     end loop;
   end process;
end behav1;
```

This code does not work
<- all variable have static durations.

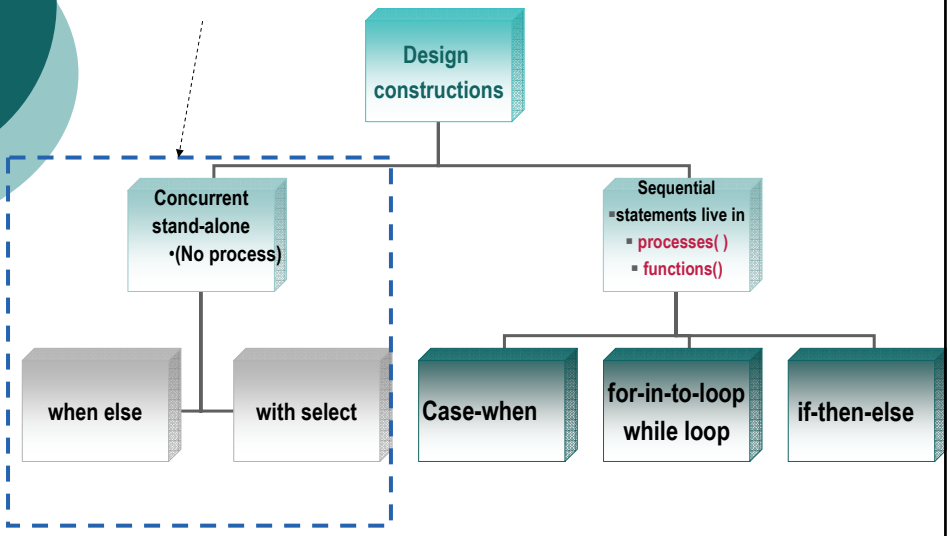
Wrong style of behavior

```
architecture behav2 of priority1 is
begin
   process (x)
   variable   ix, jx : integer; variable ones  : integer;
   begin
     for ix in 15 downto 0 loop
       ones:=0; jx:=ix+1;
       while jx<=15 loop
          if x(jx)='1' then ones:=ones+1; end if;
          jx:=jx+1;
       end loop;
      if ones=0 and x(ix)='1' then y(ix)<= '1'; else y(ix)<='0'; end if;
    end loop;
  end process;
end behav2;
```

Behavior explanation too complex for RTL

Compile and locate *priority1* again in Quartus RTL Viewer to see why it is terrible style

Design constructions
Concurrent statements



Variable assignment := execution

- Variables can only be declared and used in the sequential part (inside process or function) of VHDL local to a process.
- variable := variable assignment.
- A2 :=B2 and C2
- similar to signal assignment but A2 must be a variable.

Latches and Flip-flops

In VHDL

Latches

Latches can be easily described by using the concurrent signal assignment statement

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity JK_LATCH is
port ( J, K : in std_logic;
       signal Q : out std_logic);
end JK_LATCH;

architecture TRUTH_TABLE of JK_LATCH is
signal Q_MEM : std_logic;
begin
-- Map truth table into conditional concurrent statements
 Q_MEM <= Q_MEM when (J = '0' and K = '0') else
              '0' when (J = '0' and K='1') else
              '1' when (J='1' and K = '0') else
              not Q_MEM;
 Q <= Q_MEM;
end TRUTH_TABLE;
```

J K	Q+
0 0	Q
0 1	0
1 0	1
1 1	Q'

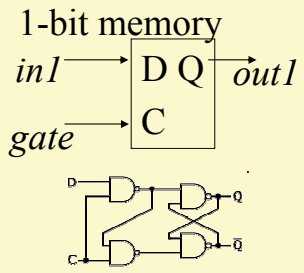
Hence, the JK latch is identical to an SR latch that is made to toggle its output when passed the restricted combination

Gated D-Latch

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity latch_ex is
  port (gate, in1 : in std_logic;
        out1 : out std_logic);
  end latch_ex;

architecture latch_ex_arch of latch_ex is
  begin
  process (gate, in1)
  begin
    if (gate = '1') then
       out1 <= in1;
    end if;
  end process;
end latch_ex_arch;
```

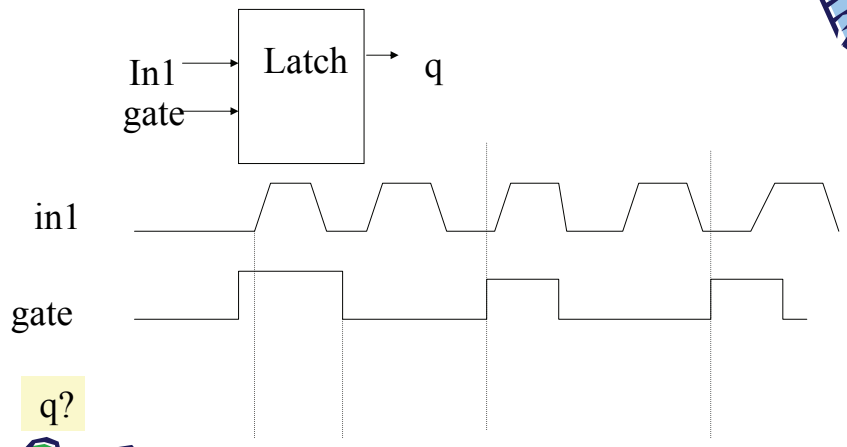
Latch

1-bit memory



sensitivity list
- the process executes once when the gate or in1

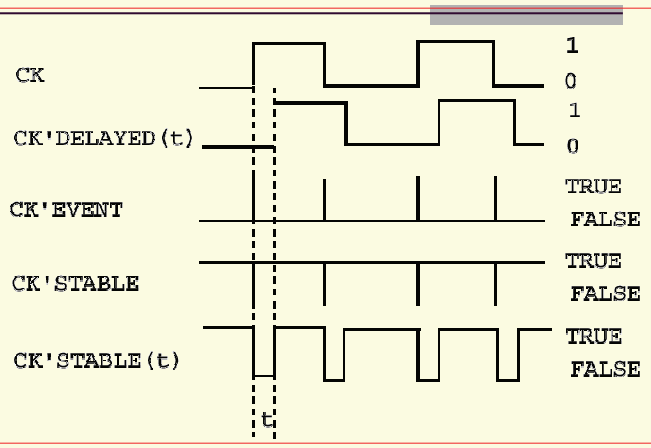
Miniotázky: nakreslete q



Signal -Related Attributes

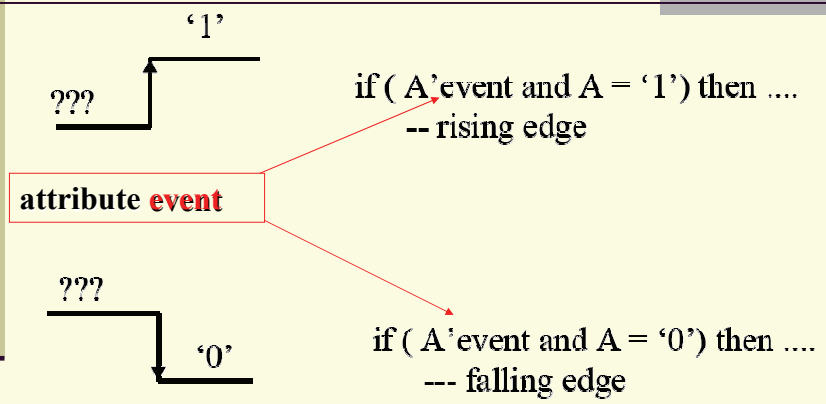
VHDL contains a number of predefined attributes, which are related to signals. They can be divided into two classes:

- attributes which define signals themselves
- attributes which are functions to provide information about signals.



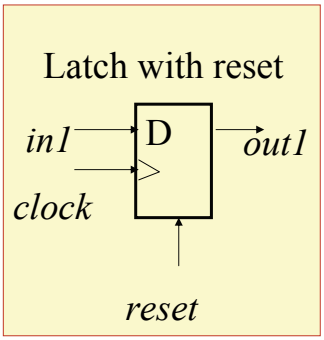
Attributes are also signals

Detecting Edges

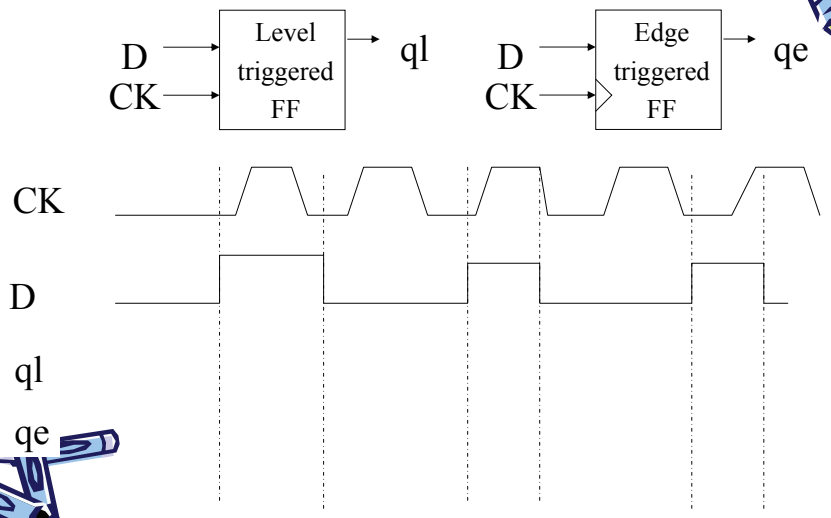


Flip-flop with asynchronous reset

```
architecture dff_asyn_arch of dff_asyn is
begin
  process(clock, reset)
  begin
    if (reset = '1') then
      out1 <= '0';
    --rising_edge(clock)
     elsif clock = '1' and clock'event then
      out1 <= in1;
    end if;
 end process;
end dff_asyn_arch;
```



Miniotázky: nakreslete ql, qe



Miniotázky

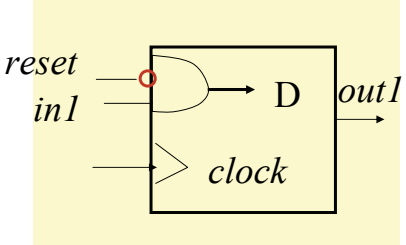
Co se stane, při náběžné hraně clock, pokud bude reset='1' ?

```
1 architecture dff_asyn_arch of dff_asyn is
2 begin
3   process(clock, reset)
4   begin
5     if (reset = '1') then
6       out1 <= '0';
7     elsif clock = '1' and clock'event then
8       out1 <= in1;
9     end if;
10  end process;
11 end dff_asyn_arch;
```

clock and reset must be in sensitivity list

Flip-flop with synchronous reset

```
architecture dff_syn_arch of dff_syn is
begin process(clock,reset) -- reset can be removed,
 begin–                     -- but allowed
   if clock = '1' and clock'event then
     if (reset = '1') then
       out1 <= '0';
     else
       out1 <= in1;
      end if;
   end if;
 end process;
end dff_syn_arch;
```



Why reset is not needed in the sensitivity list?

Difference between Syn. & Asyn. flip-flops

The order of the statements inside the process determines Syn. or Asyn. reset

1

```
if clock = '1' and clock'event then
  if (reset = '1') then
```

2

```
if (reset = '1') then
  q <= '0';
elsif clock = '1' and clock'event then
```

Which flip-flop has synchronous and asynchronous reset?

Miniquestions

- What is the difference between latches and flip-flops (level triggered flip-flops and edge triggered flip-flops)?

  Note: **In some FPGA all flip-flops are treated as 50% edge triggered flip-flops.

- What is the difference between
  - synchronous reset (syn-reset) flip-flops and
  - asynchronous reset (asyn-reset) flip-flops?

Example: Divider by 50 milions with 50% duty cycle

Synchronous Divider

```
-- pridame delic 2 k delici 25000000 -> delic50000000
library ieee; use ieee.std_logic_1164.all; use
   ieee.std_logic_unsigned.all;
-- delic 50e6 se symetrickym vystupem
entity delic50M is
  port(clk  : in std_logic;
        q  : out std_logic);
end entity;

architecture behav1 of delic50M is
constant MAX:integer :=24999999;
begin
 process (clk)
  --processes body on the next slide
 end process;
end behav1;
```


Synchronous Divider

```
process (clk)
 variable   delic2 : std_logic := '0';
 variable   cnt : integer:=0;
begin
   if (clk'event and clk='1') then
      if cnt=MAX then
        cnt:=0; delic2:=not delic2;
      else cnt := cnt + 1;
     end if;
   end if;
   q<=delic2;
 end process;
```

Too high gate count
56 LE / 33 Reg
LE=Logic elements
Reg - registers

Information

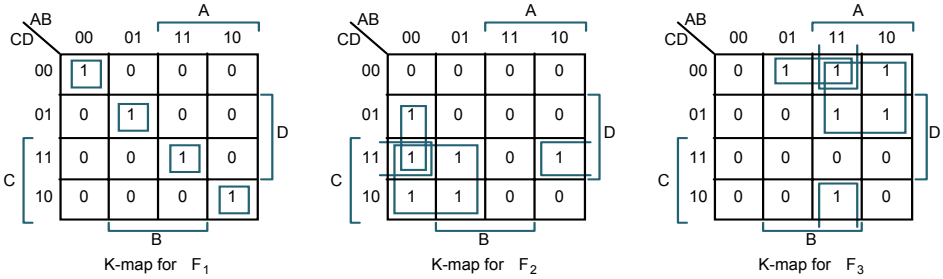
- If we supply detailed information about circuit limit VHDL compiler will create more optimal circuit
  - Is it required 32 bit cnt variable ? -> No
  - Is it necessary perform equality comparison? -> No

Improved Behavior Description

```
process (clk)
  variable   delic2 : std_logic := '0';
  variable   cnt : integer range 0 to MAX :=0;
begin
   if (clk'event and clk='1') then
      if cnt=MAX then
        cnt:=0; delic2:=not delic2;
      else cnt := cnt + 1;
     end if;
   end if;
   q<=delic2;
  end process;
```

46LE/26Reg
by adding range 0 to MAX information we simplified compilation

Remeber Two-Level Simplification



Equality is expensive operation

F1 = A' B' C' D' + A' B C' D + A B C D + A B' C D'

F2 = A' B' D + A' C

F3 = B C' D' + A C' + A B D'

F1 = F2' . F3'

Synchronous Divider

```
process (clk)
  variable   delic2 : std_logic := '0';
  variable   variable   cnt : integer range 0 to MAX :=0;
begin
   if (clk'event and clk='1') then
      if cnt>=MAX then
        cnt:=0; delic2:=not delic2;
      else cnt := cnt + 1;
     end if;
   end if;
   q<=delic2;
  end process;
```

37LE/26Reg
compare by >=
we added another information

Synchronous Divider

```
process (clk)
  variable   delic2 : std_logic := '0';
  variable   variable   cnt : integer range 0 to MAX :=0;
begin
   if (clk'event and clk='1') then
      if cnt>MAX-1 then
        cnt:=0; delic2:=not delic2;
      else cnt := cnt + 1;
     end if;
   end if;
   q<=delic2;
  end process;
```

36LE/26Reg
compare by >

Information

By inserting detailed information
about circuit behavior,
we have 36LE/26Reg
i.e. we saved 20LE/9Reg
from original 56LE/33Reg

Universal divider

```
library ieee;use ieee.std_logic_1164.all;use ieee.std_logic_unsigned.all;
entity FreqDivByEven is
  generic( Even_Divisor: integer := 50 );
  port(CLK  : in std_logic;
       q  : out std_logic);
end entity;
architecture behav of FreqDivByEven is
constant Divider:integer :=Even_Divisor/2;
begin
 -- FreqDivByEven requires EVEN number
 assert Even_Divisor mod 2=0 -- tested in compile time
 report "FreqDivByEven requires EVEN divisor"
        severity severity_level(error);
 process (CLK)
```

see the next slide

```
 end process;
end behav;
```


Universal divider

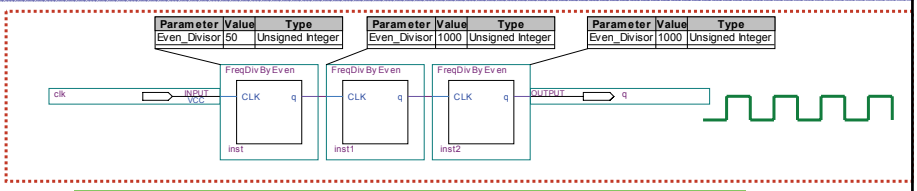
```
process (CLK)
 variable   cnt : integer range 0 to Divider-1:=0;
 variable   q2 : std_logic := '0';
 begin
 if (CLK'event and CLK='1') then
   if Even_Divisor=2 then q2:=not q2;
   elsif cnt>Divider-2 then cnt:=0; q2:=not q2;
   else cnt := cnt + 1;
   end if;
  end if;
 q<=q2;
 end process;
```

One divider with parameter 50milions

Quartus info for divider by 50 milions created as one circuit:

- **36LE/26Reg**
- Info: Clock "clk" has Internal fmax of 263.99 MHz between source register

The better divider by 50 milions



Gate count only 33 LE/ 26Reg

Info: Clock "clk" has Internal fmax of **338.29 MHz** between source register "FreqDivByEven:inst1|cnt[0]" and destination register "FreqDivByEven:inst1|q2" (period= 2.956 ns)

Decomposition

We can obtain more optimal
and effective circuit by decomposing
huge designs to several smaller units

- e.g. by proper splitting
one 50 million divider
to 3 dividers 50,1000, and 1000