TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

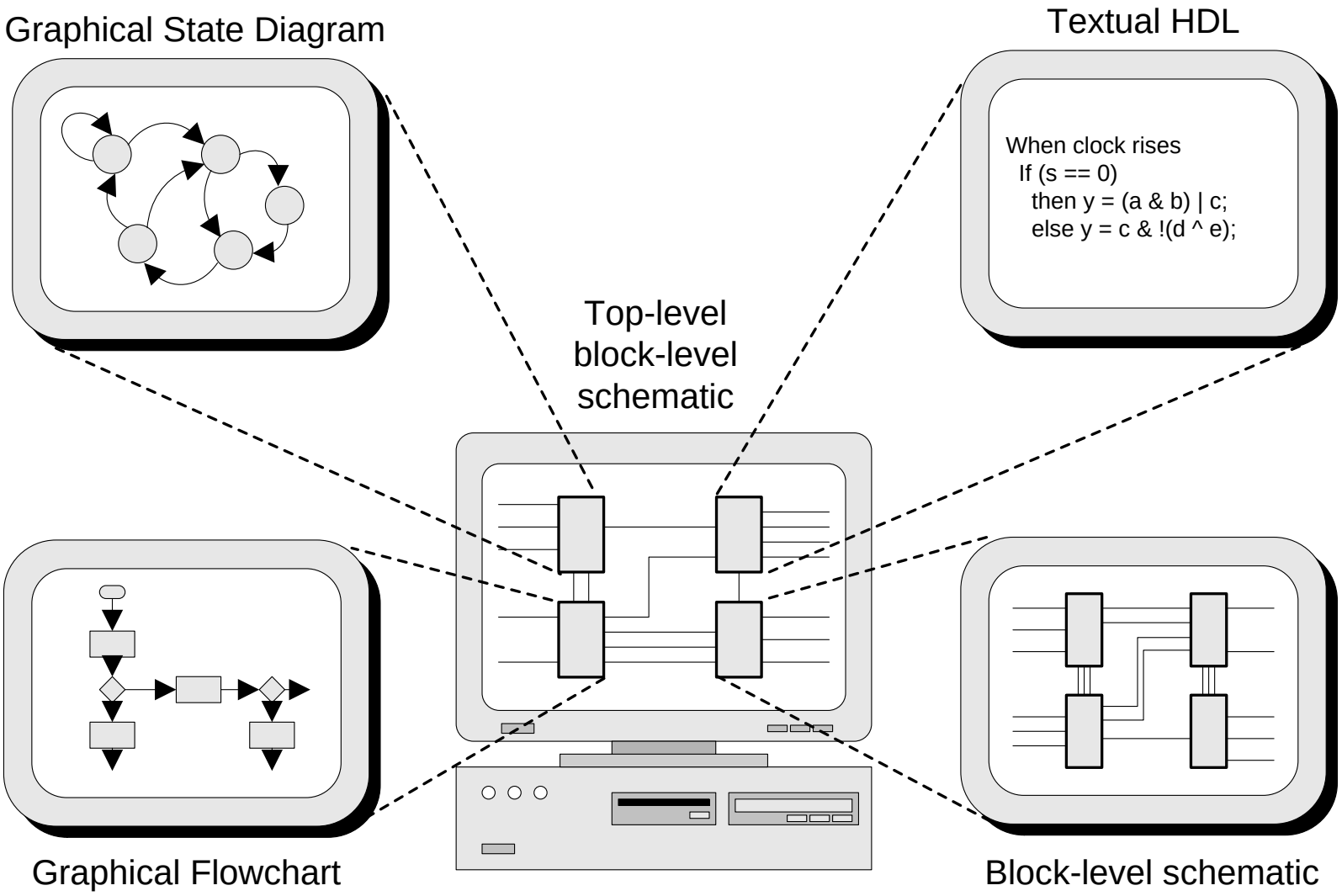
Jazyky popisující hardware

Prof. Ing. Ondřej Novák, CSc.
ITE

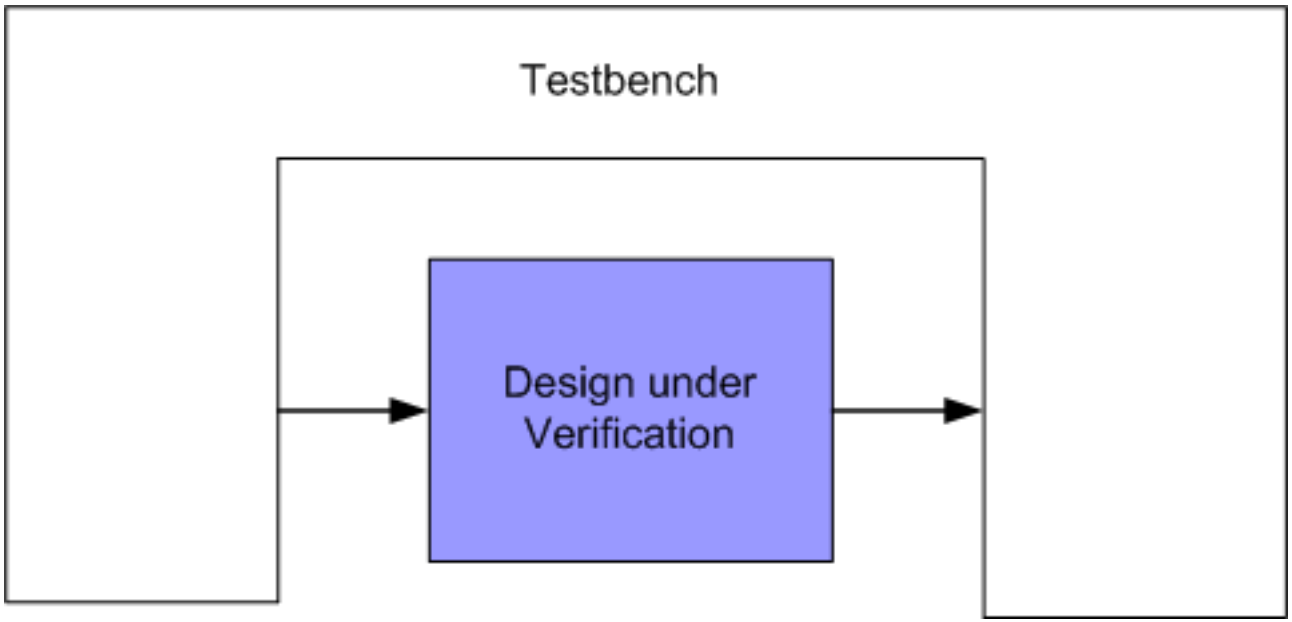
Základní kroky návrhu obvodů

- **Popis**
  - vytvoření modelu obvodu
- **Simulace**
  - Vstupní hodnoty jsou přivedeny na vstupy modelu
  - Kontrola správnosti výstupních hodnot
  - Úspora peněz při ladění návrhu pomocí simulace
- **Syntéza**
  - Transformace HDL kódu do netlistu (seznam hradel a jejich propojení)
- **Překlad**
  - netlist je transformován pro konkrétní technologii

Popis obvodu



Simulace - Testbench

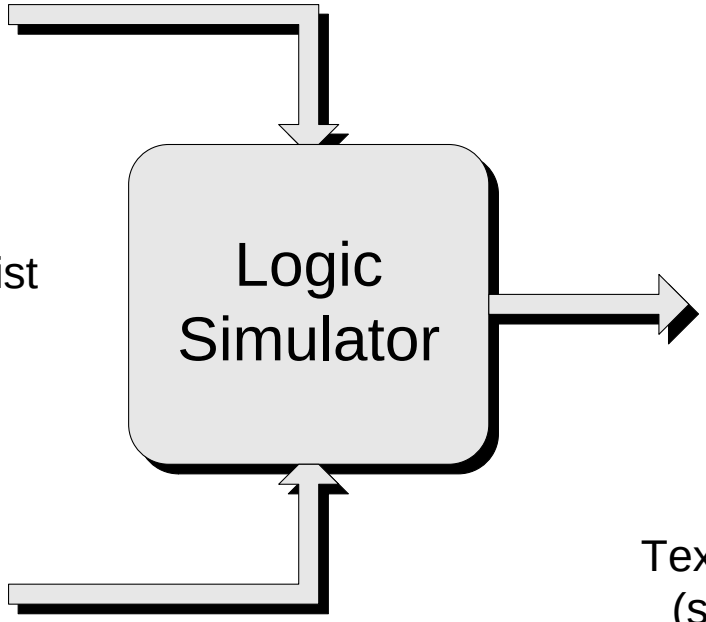


Testbench vytváří virtuální svět testovanému obvodu.
Vstupní stimuly →výstupní odezvy
Vstupní testovací vektory →výstupní data

Logický simulátor

```
BEGIN CIRCUIT=TEST
   INPUT  SET_A, SET-B,
          DATA, CLOCK,
          CLEAR_A, CLEAR_B;
   OUTPUT Q, N_Q;
   WIRE   SET, N_DATA, CLEAR;

   GATE G1=NAND (IN1=SET_A,
                 IN2=SET_B,
                 OUT1=SET);
   GATE G2=NOT  (IN1=DATA,
                 OUT1=N_DATA);
   GATE G3=OR   (IN1=CLEAR_A,
                 IN2=CLEAR_B,
                 OUT1=CLEAR);
   GATE G4=DFF  (IN1=SET, IN2=N_DATA,
                 IN3=CLOCK, IN4=CLEAR,
                 OUT1=Q, OUT2=N_Q);

END CIRCUIT=TEST;
```

Textual gate-level netlist



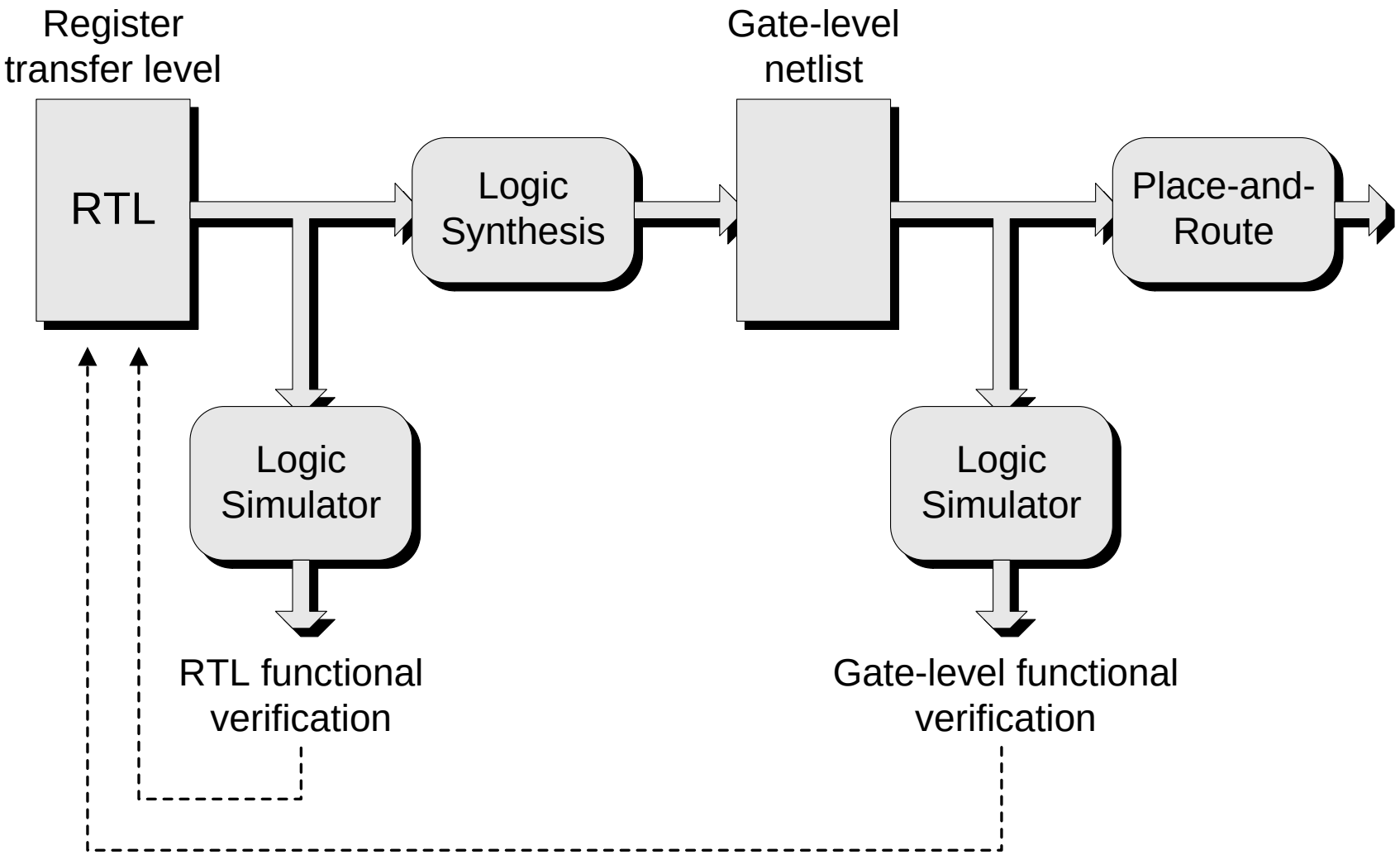

```
              C C
              L L
      S S   C E E
      E E D L A A
      T T A O R R
      _ _ T C _ _
 TIME A B A K A B
----- -----------
    0 1 1 1 0 0 0    ; Set up
  500 1 1 1 1 0 0    ; Rising edge
 1000 1 1 1 0 0 0    ; Falling edge
 1500 1 1 0 0 0 0    ; Set data
 2000 1 1 0 1 0 0    ; Rising edge
 2500 1 1 0 1 0 1    ; Clear active
   :
  etc.
```

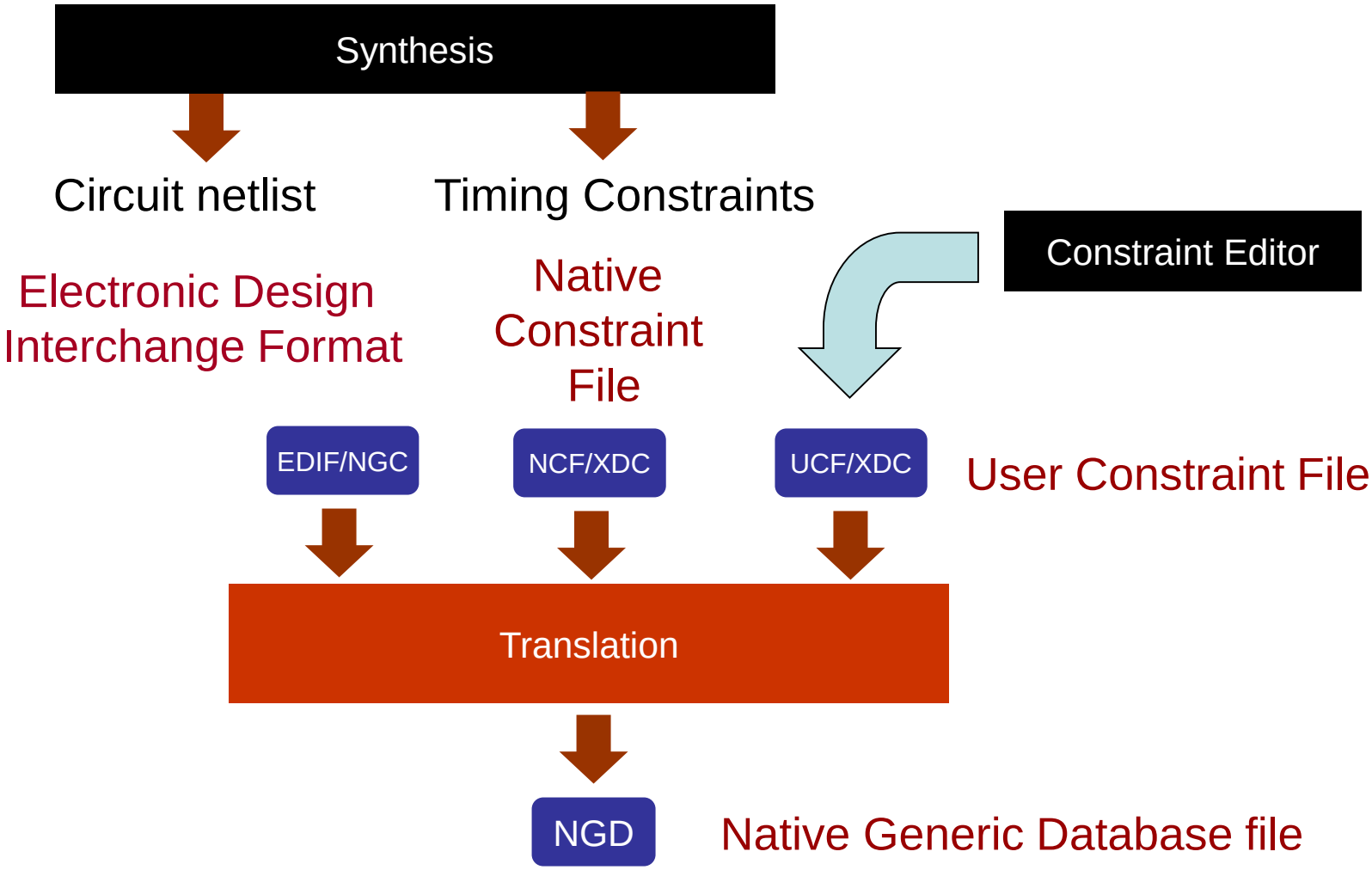
Textual (tabular) stimulus

```
              C C
              L L    N
      S S   C E E    _ C
      E E D L A A    D L
      T T A O R R  S A E    N
      _ _ T C _ _  E T A    _
 TIME A B A K A B  T A R  Q Q
----- -----------  -----  ---
    0 1 1 1 0 0 0  X X X  X X
    5 1 1 1 0 0 0  X 0 X  X X
   10 1 1 1 0 0 0  0 0 0  X X
  500 1 1 1 1 0 0  0 0 0  X X
  520 1 1 1 1 0 0  0 0 0  0 1
 1000 1 1 1 0 0 0  0 0 0  0 1
 1500 1 1 0 0 0 0  0 0 0  0 1
 1505 1 1 0 0 0 0  0 1 0  0 1
 2000 1 1 0 1 0 0  0 1 0  0 1
 2020 1 1 0 1 0 0  0 1 0  1 0
 2500 1 1 0 1 0 1  0 1 0  1 0
 2510 1 1 0 1 0 1  0 1 1  1 0
 2530 1 1 0 1 0 1  0 1 1  0 1
   :
  etc.
```

Textual (tabular) results file
(stimulus and response)

Postup návrhu ASIC




Synthesis
Circuit netlist
Timing Constraints
Electronic Design Interchange Format
Native Constraint File
Constraint Editor
EDIF/NGC
NCF/XDC
UCF/XDC
User Constraint File
Translation
NGD
Native Generic Database file

Výhody HDL jazyků

- Jazyky popisující hardware (Hardware Description Language HDL):
  - dovolí návrháři specifikovat pouze logickou funkci.
  - Z HDL návrhový systém syntetizuje optimalizované zapojení hradel.
  - Popis pomocí HDL je univerzální, nezávisí na realizační platformě (ASIC, hradlové pole, FPGA, PLD).
- Většina návrhových systémů vychází z popisu obvodu pomocí HDL

Popis systému založený na HDL

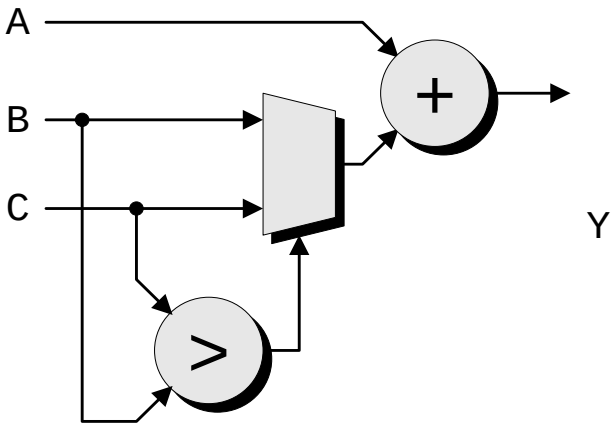
- Umožňuje rychlejší zadání složitých obvodů
- Zrychluje simulaci

Postupně se HDL vyvíjí tak, že logická syntéza je nahrazována syntézou beroucí ohled na architekturu obvodu

Sdílení zdrojů

```
if (B > C)
   then Y = A + B;
   else Y = A + C;
end if;
```

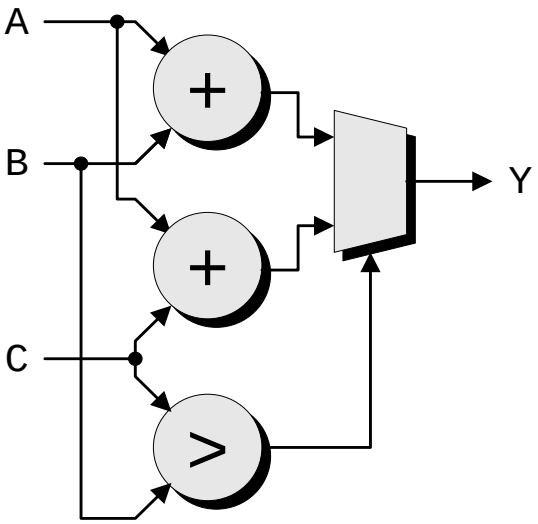
Resource Sharing = ON



Total LUTs = 32

Clock frequency = 87.7 MHz

Resource Sharing = OFF



Total LUTs = 64

Clock frequency = 133.3 MHz (+52% !)

HDL

Verilog

- Vyvinut v roce 1984 Gateway Design Automation
- IEEE standard (1364) v roce 1995,standard IEEE 1800 v roce 2009

VHDL

- Vyvinut v roce 1981 Department of Defense, USA
- IEEE standard (1076) v roce1987, revize 2008

System Verilog

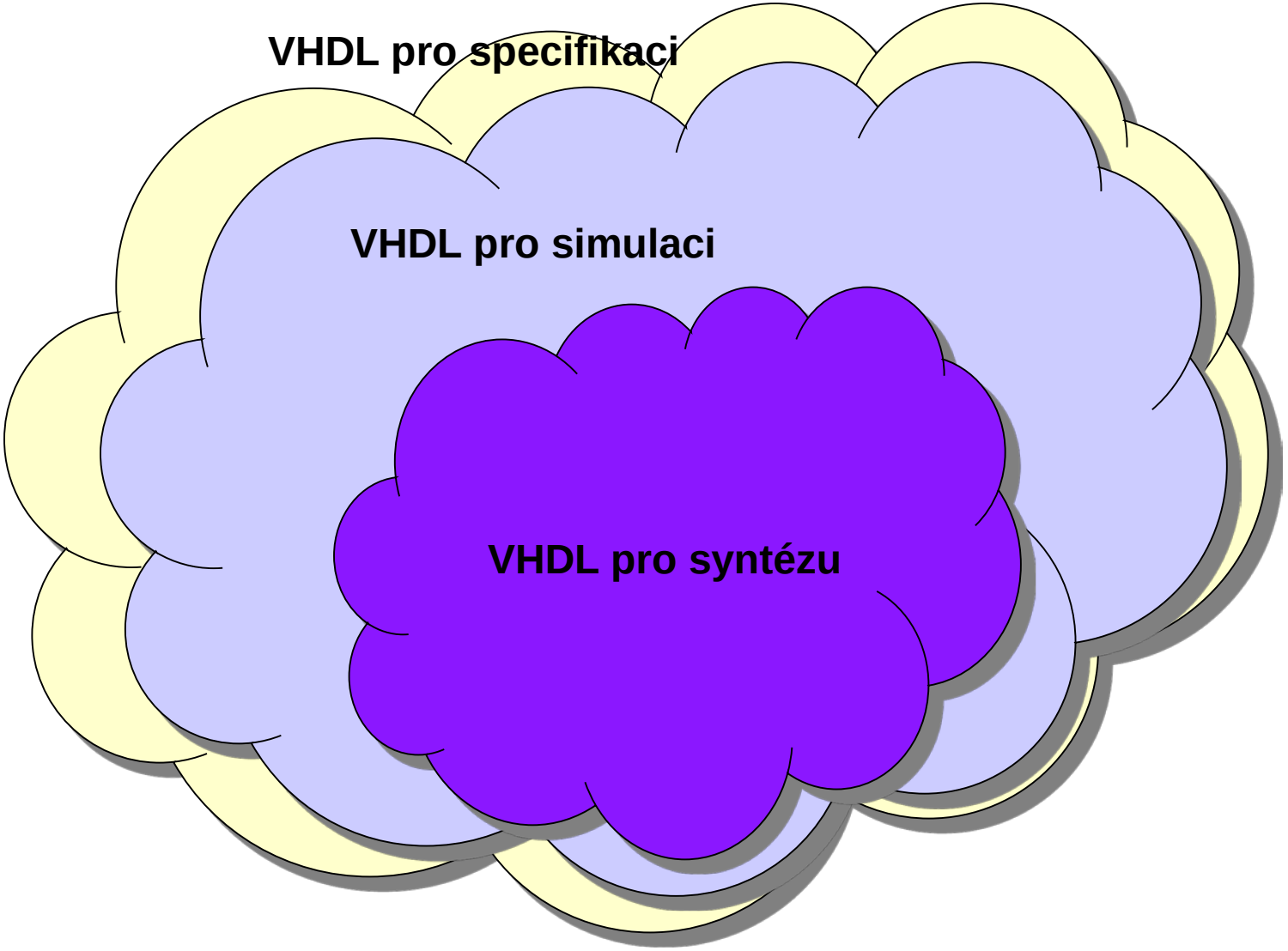
- Vyvinut v roce 2002
- IEEE standard (1800) v roce 2005

System C

- Vyvinut v roce 1999
- IEEE standard (1666) v roce 2005

Většina návrhů obvodů je prováděna pomocí Verilogu nebo VHDL

VHDL



Entita a architektura

- **Entita** - „černá skříňka“ se vstupy a výstupy (obdoba grafického symbolu)
  - Entita nepopisuje chování modulů (nedefinuje funkci)

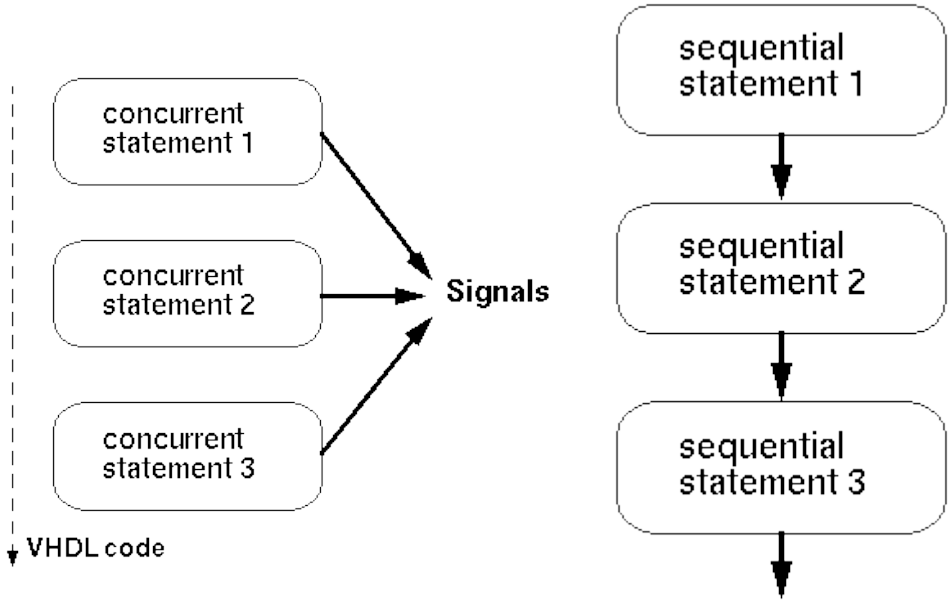
- **Architektura** - určuje chování entit
  - tělo architektury má dvě části:
    - deklarační část (např. definice signálů)
    - příkazová část (uzavřeno do ***begin - end***)
  - architektura musí být spojena se specifikovanou entitou
  - rozdílné architektury definují rozdílné pohledy na entity
  - ke každé entitě lze definovat více architektur

Entita a architektura

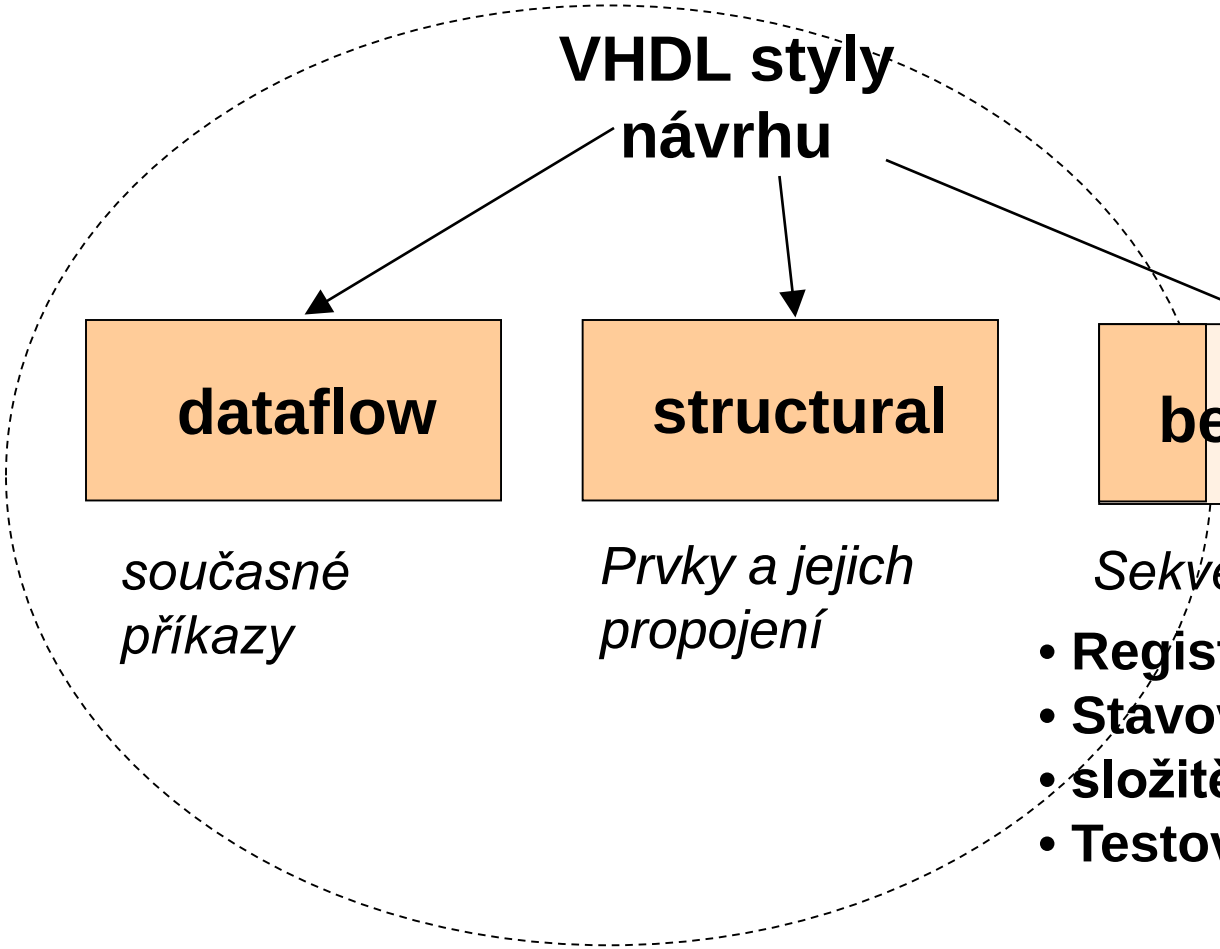
```
use IEEE;
use IEEE.std_logic_1164.all;

entity AND2_OP is
    port(
       A, B: in std_logic;
       Z   : out std_logic);
end AND2_OP;

architecture MODEL of AND2_OP is
begin
    Z <= A and B;
end MODEL;
```

Příkazy VHDL

- **Deklarace**:
  - definice konstant, typů, objektů
- **Současné** (concurrent, dataflow) příkazy
  - popis kombinačních obvodů
- **Sekvenční** (sequential, behavioral) příkazy:
  - if, case, loop, next, wait …



VHDL návrhové styly



Podmnožina příkazů vhodných pro syntézu

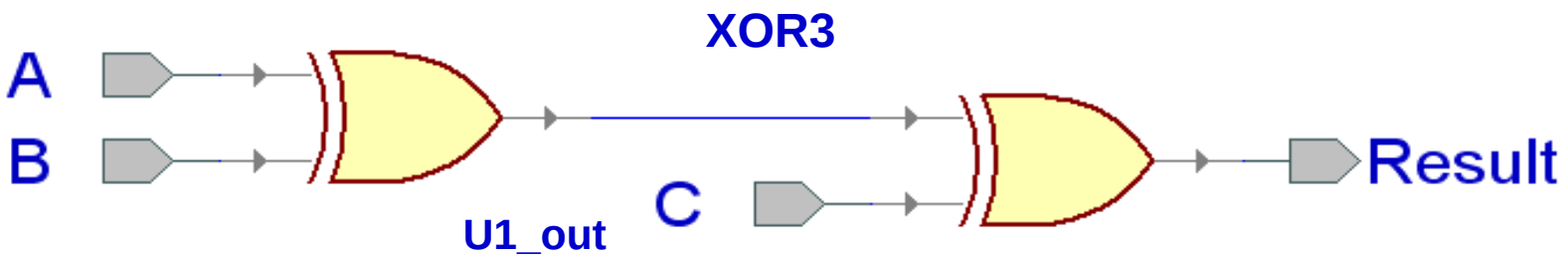
Příklad: XOR3

Entity (XOR3 Gate)

```
entity XOR3 is
   port(
      A : in STD_LOGIC;
      B : in STD_LOGIC;
      C : in STD_LOGIC;
      RESULT : out STD_LOGIC
         );
end XOR3;
```

Dataflow Architektura XOR3 hradlo

```
architecture XOR3_DATAFLOW of XOR3 is
signal U1_OUT: STD_LOGIC;
begin
        U1_OUT<=A xor B;
        RESULT<=U1_OUT xor C;
end XOR3_DATAFLOW;
```



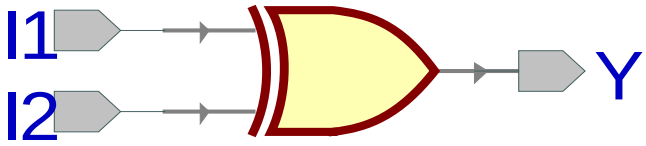
Další dataflow operátory: not, and, or, xor, nand,nor

Dataflow popis

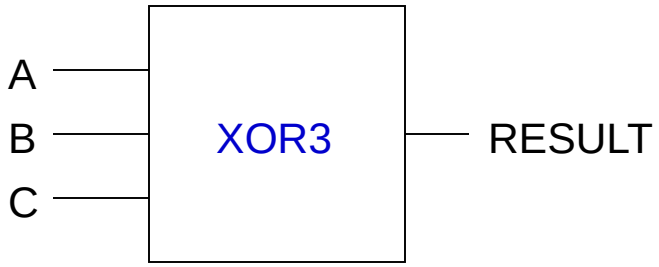
- Popisuje , jak se data přesouvají v systému
- Dataflow popis používá sérii současných příkazů pro realizaci logických funkcí. Všechny příkazy jsou vyhodnoceny v jeden okamžik, nezáleží tedy na jejich pořadí.
- Je užitečný, když je možné logiku obvodu representovat Boolovskými rovnicemi.

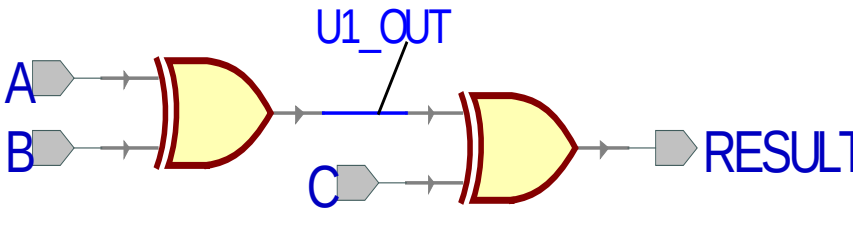
Strukturní architektura hradla XOR3

```
architecture XOR3_STRUCTURAL of XOR3 is
signal U1_OUT : STD_LOGIC;

component XOR2 is
    port(
        I1 : in  STD_LOGIC;
        I2 : in  STD_LOGIC;
        Y  : out STD_LOGIC
    );
end component;

begin
U1 : XOR2
port map(I1 => A,
         I2 => B,
         Y  => U1_OUT);

U2 : XOR2
port map(I1 => U1_OUT,
         I2 => C,
         Y  => RESULT);
end XOR3_STRUCTURAL;
```



XOR2

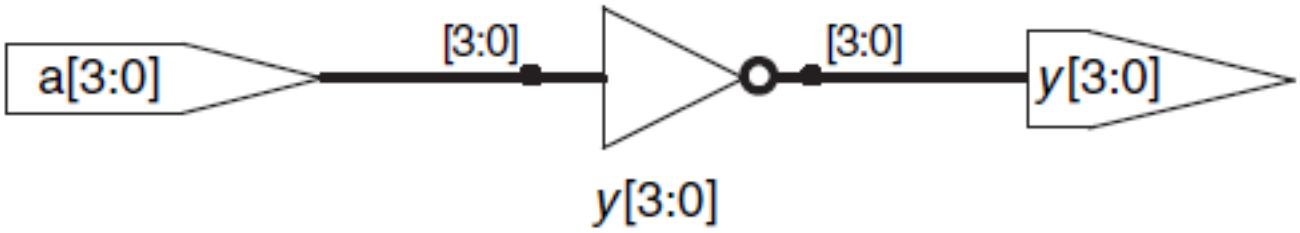




XOR3

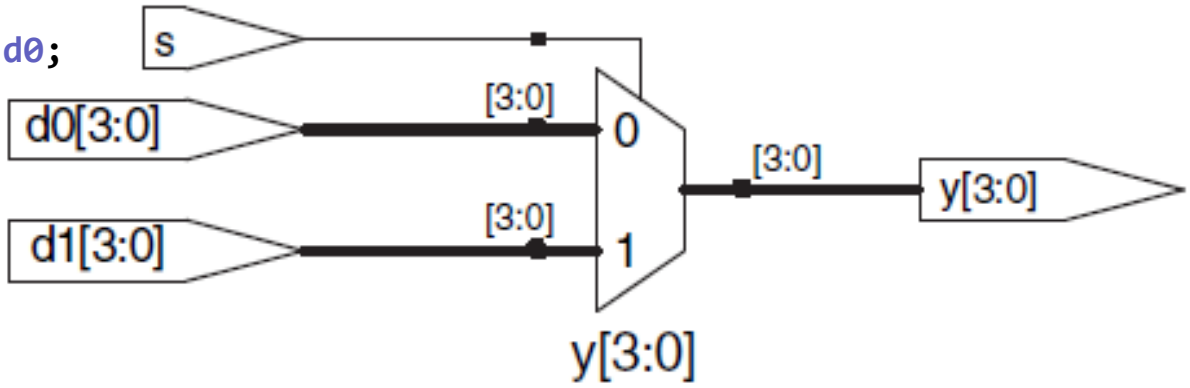
Práce s vektory

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

entity inv is
port(a : in  STD_LOGIC_VECTOR(3 downto 0);
     y : out STD_LOGIC_VECTOR(3 downto 0));
end;
architecture synth of inv is
begin
y <= not a;
end;
```



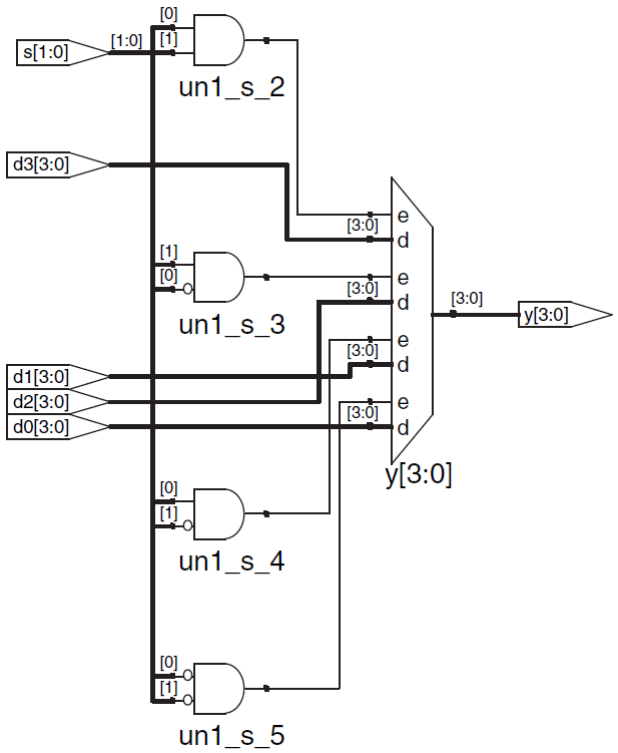
podmíněné přiřazení signálů – vícebitová výhybka

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

entity mux2 is
    port(
        d0, d1 : in  STD_LOGIC_VECTOR(3 downto 0);
        s      : in  STD_LOGIC;
        y      : out STD_LOGIC_VECTOR(3 downto 0)
    );
end;

architecture synth of mux2 is
begin
    y <= d1 when s='1' else d0;
end;
```



Multiplexor

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

entity mux4 is
    port(
        d0, d1, d2, d3 : in  STD_LOGIC_VECTOR(3 downto 0);
        s              : in  STD_LOGIC_VECTOR(1 downto 0);
        y              : out STD_LOGIC_VECTOR(3 downto 0));
    end;

architecture synth1 of mux4 is
begin
y <= d0 when s = "00"
     else d1 when s = "01"
     else d2 when s = "10"
     else d3;
end;
```



Operátory – priority provádění operací

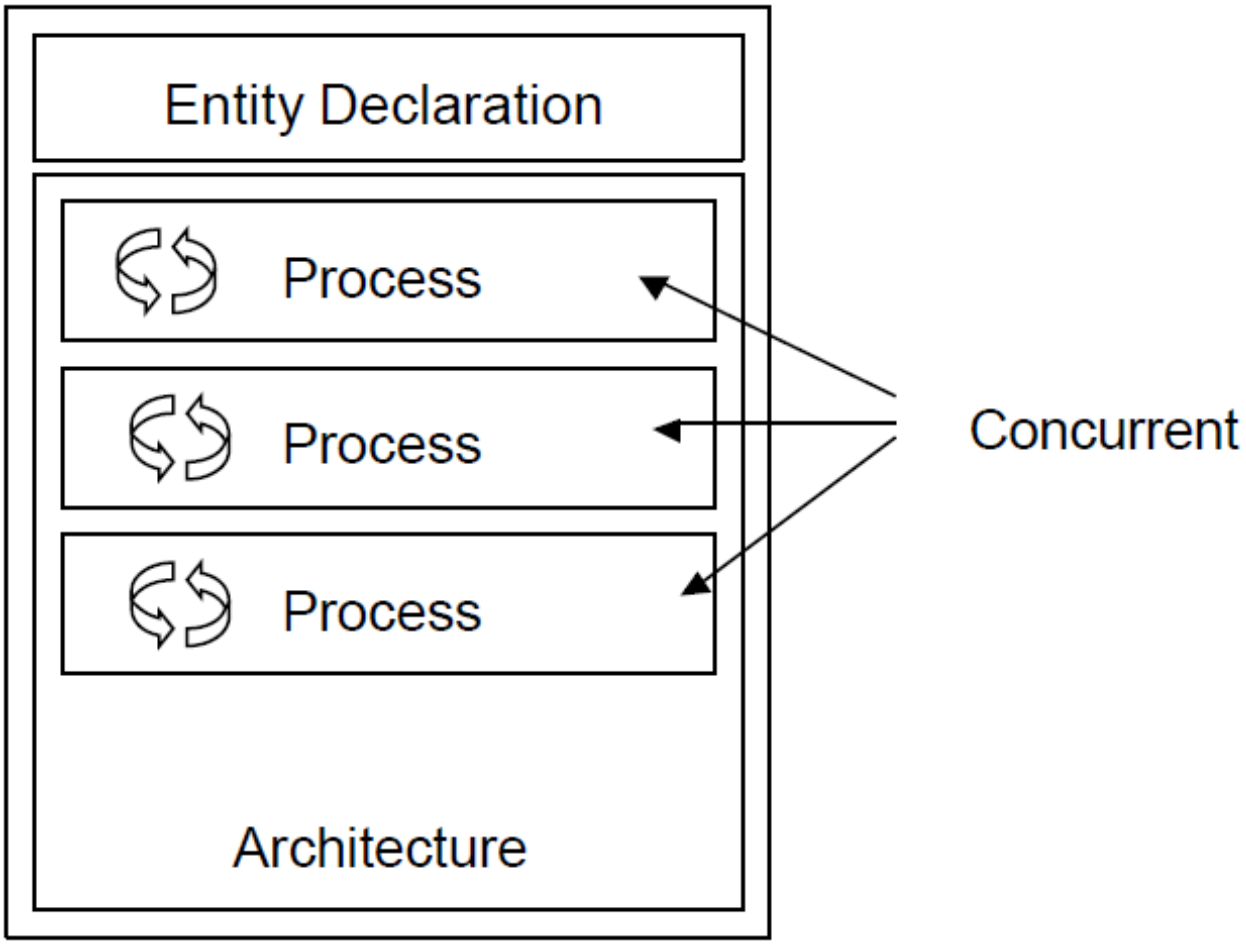
	Op	Meaning
Highest	`not`	NOT
	`*`, `/`, `mod`, `rem`	MUL, DIV, MOD, REM
	`+`, `-`	PLUS, MINUS
	`rol`, `ror`, `srl`, `sll`	Rotate, Shift logical
	`<`, `<=`, `>`, `>=`	Relative Comparison
	`=`, `/=`	Equality Comparison
Lowest	`and`, `or`, `nand`, `nor`, `xor`, `xnor`	Logical Operations

Použití hodinového signálu v sekvenčním příkazu

```
process(hodinový_signál)    -- použitím citlivých proměnných
begin
    if (podmínka_hrany_hod_signálu) then
        výstupní_signál <= vstupní signál;
        … další sekvenční příkazy …
        end if;
end process;
```

```
process
begin
wait on (hodinový_signál) until (specifikace_hrany)
        výstupní_signál <= vstupní signál;
        … další sekvenční příkazy …
end process;
```

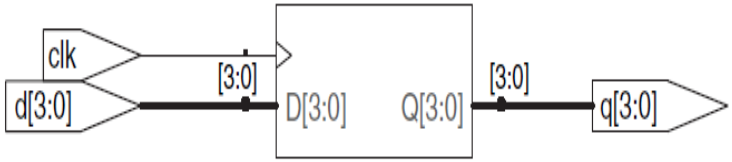
Více procesů

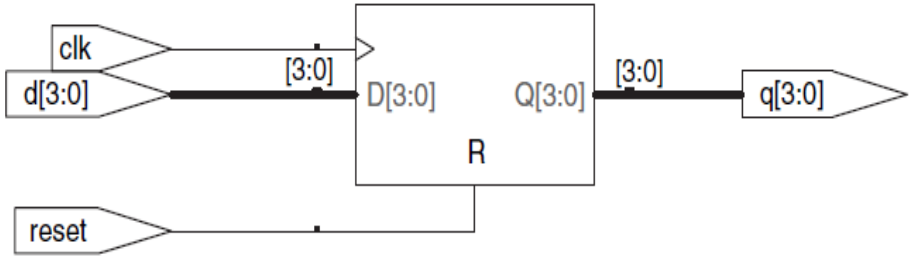


DFF

```
architecture synth of flop is

begin
    process(clk)
    begin
        if rising_edge(clk) then
            q <= d;
        end if;
    end process;
end;
```

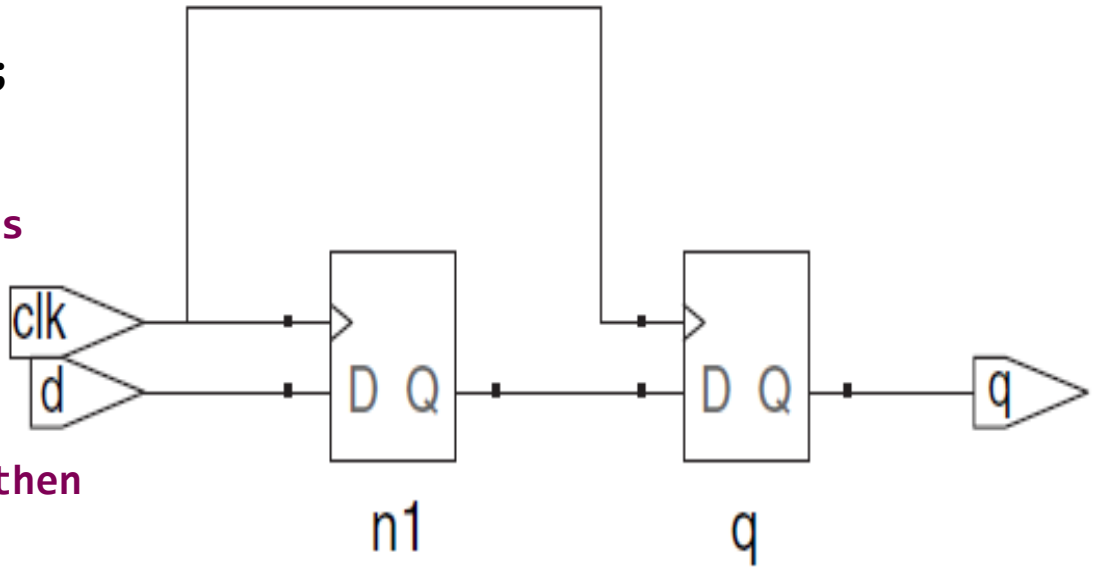
```
architecture asynchronous of flop is
begin
    process(clk, reset)
    begin
        if reset then
            q <= "0000";
        elsif rising_edge(clk) then
            q <= d;
        end if;
    end process;
end;
```





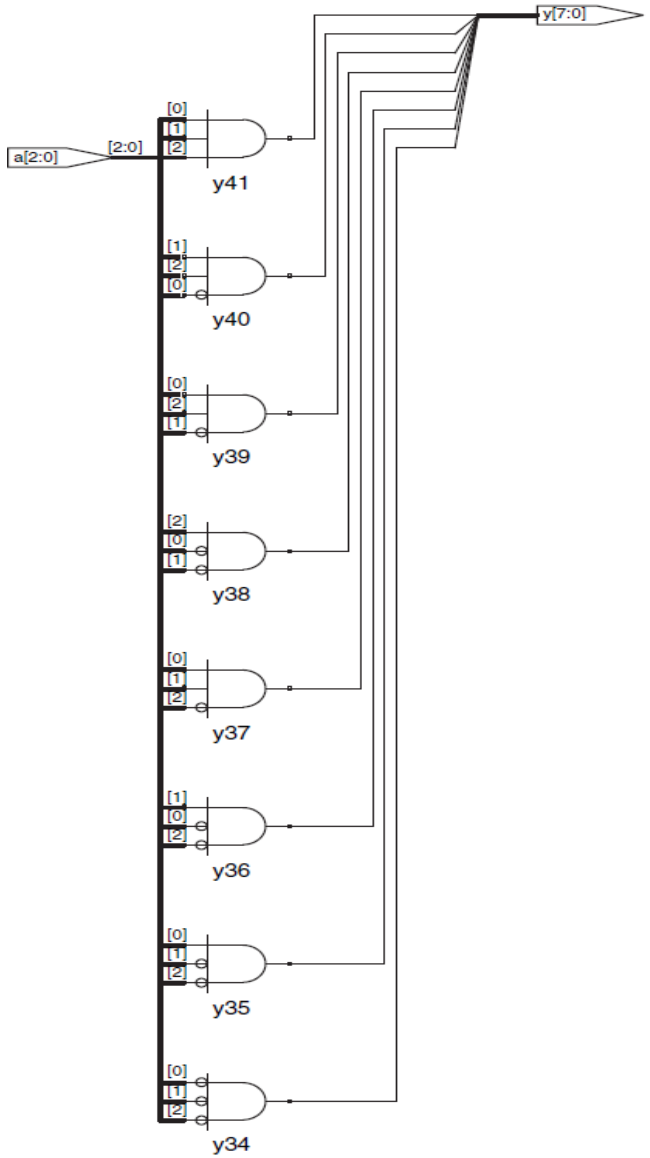
Posuvný registr

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

entity sync is
    port(clk : in  STD_LOGIC;
         d   : in  STD_LOGIC;
         q   : out STD_LOGIC);
end;

architecture good of sync is
signal n1 : STD_LOGIC;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            n1 <= d;
            q  <= n1;
        end if;
    end process;
end;
```



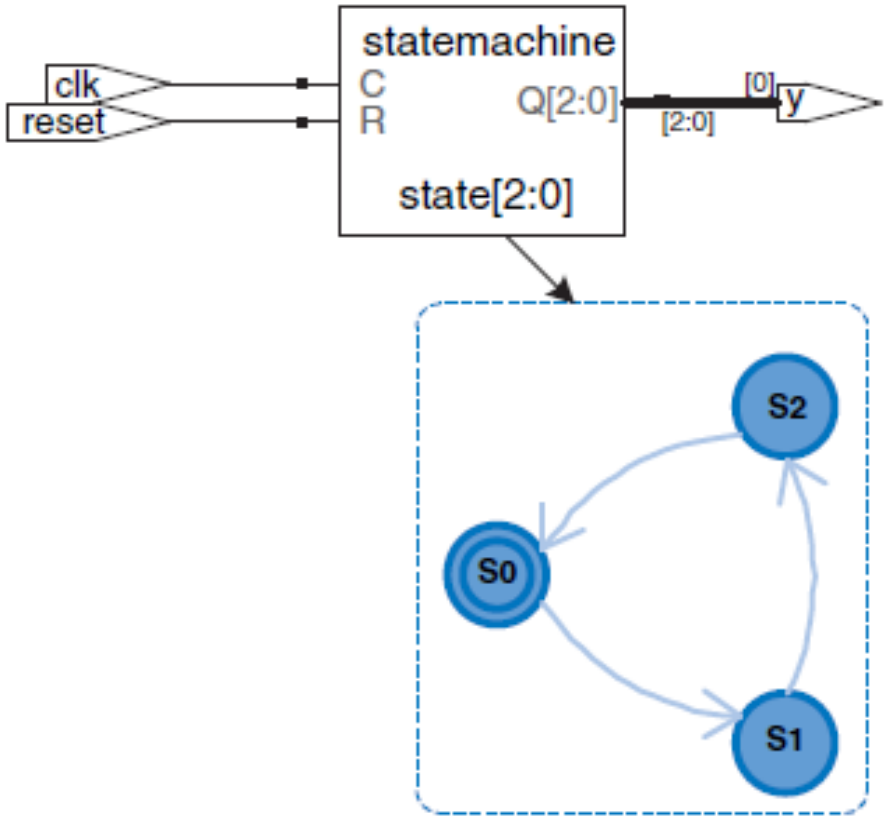
dekodér

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity decoder3_8 is
port(
    a : in  STD_LOGIC_VECTOR(2 downto 0);
    y : out STD_LOGIC_VECTOR(7 downto 0)
);
end;

architecture synth of decoder3_8 is
begin
    process(a)
    begin
        case a is
            when "000"  => y <= "00000001";
            when "001"  => y <= "00000010";
            when "010"  => y <= "00000100";
            when "011"  => y <= "00001000";
            when "100"  => y <= "00010000";
            when "101"  => y <= "00100000";
            when "110"  => y <= "01000000";
            when "111"  => y <= "10000000";
            when others => y <= "XXXXXXXX";
        end case;
    end process;
end;
```



Návrh automatu

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity divideby3FSM is
    port(
        clk, reset : in  STD_LOGIC;
        y          : out STD_LOGIC);
end;
architecture synth of divideby3FSM is
    type statetype is (S0, S1, S2);
    signal state, nextstate : statetype;
begin
    --state register
    process(clk, reset)
    begin
        if reset = '1' then
            state <= S0;
        elsif rising_edge(clk) then
            state <= nextstate;
        end if;
    end process;

    --next state logic
    nextstate <= S1 when state = S0
            else S2 when state = S1
            else S0;
    --output logic
    y         <= '1' when state = S0 else '0';
end;
```



Rozpoznání vzorku – Mealy automat

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

entity patternMealy is
    port(
        clk, reset : in  STD_LOGIC;
        a          : in  STD_LOGIC;
        y          : out STD_LOGIC
    );
end;

architecture synth of patternMealy is
    type statetype is (S0, S1);
    signal state, nextstate : statetype;
begin
    --state register
    process(clk, reset)
    begin
        if reset = '1' then
            state <= S0;
        elsif rising_edge(clk) then
            state <= nextstate;
    end if;
    end process;
```

```
    --next state logic
    process(a, state)
    begin
        case state is
            when S0 =>
                if a = '1' then
                    nextstate <= S0;
                else
                    nextstate <= S1;
                end if;
            when S1 =>
                if a = '1' then
                    nextstate <= S0;
                else
                    nextstate <= S1;
                end if;
            when others =>
                nextstate <= S0;
        end case;
    end process;

    --output logic
    y <= '1' when (a = '1' and state = S1)
            else '0';

end;
```

Syntetizovaný obvod