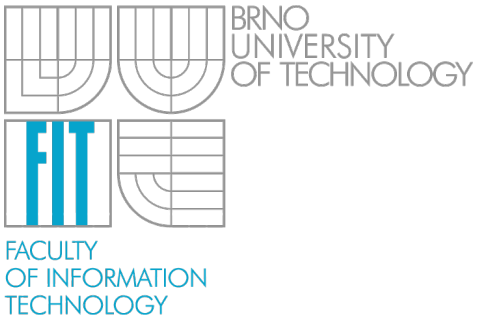
Logická syntéza

Jan Kořenek

Brno University of Technology, Faculty of Information Technology
Božetěchova 2, 612 00 Brno, CZ
www.fit.vutbr.cz/~korenek

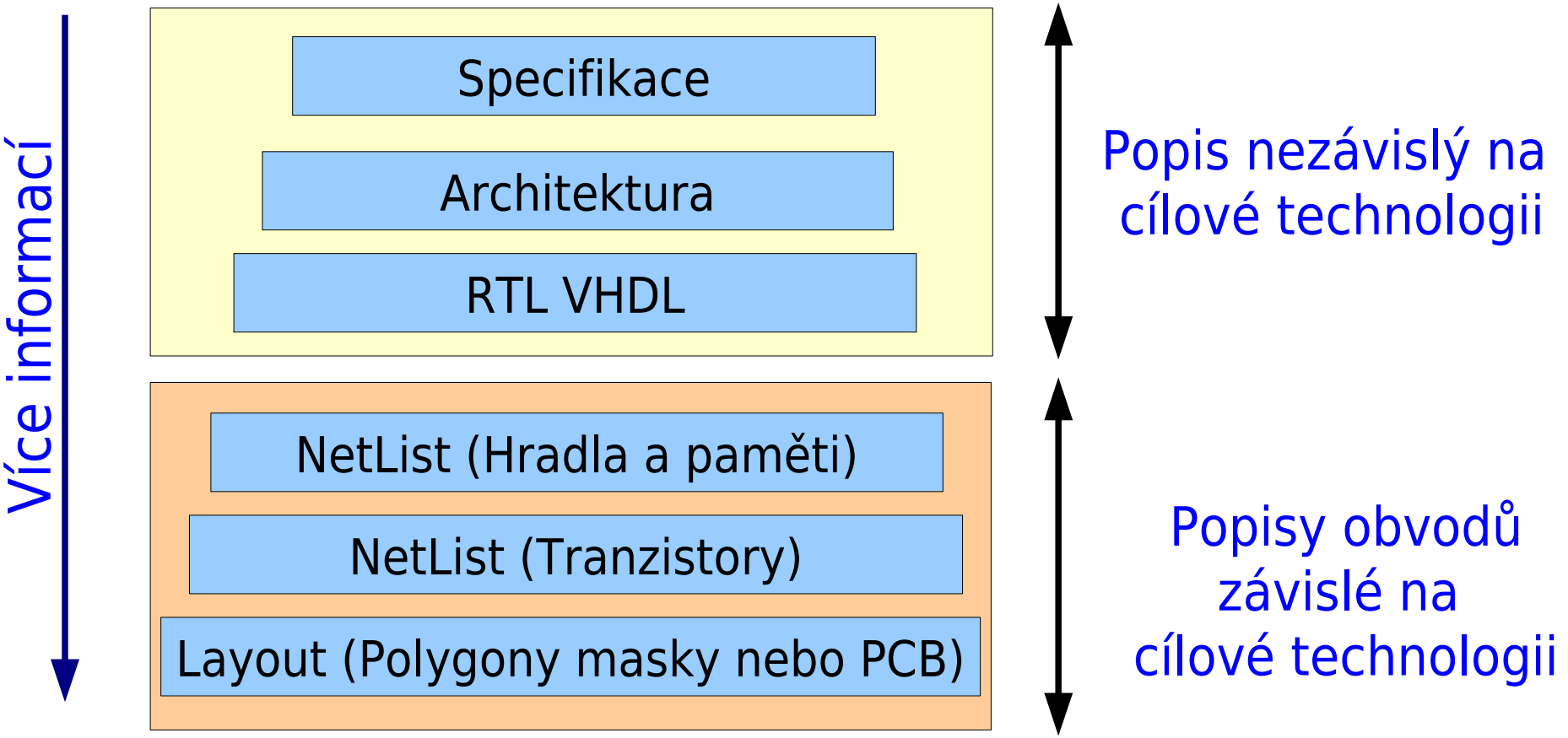


Použitá literatura

N. Frištacký, M. Kolesár, J. Kolenička a J. Hlavatý: „Logické systémy“, SNTL Praha, 1986
M. Eysselt: „Logické systémy“, SNTL Praha, skriptum VUT v Brně, 1985
J. F. Wakerly: „Digital Design. Principles and Practices“, Prentice Hall, ISBN 0-13-769191-2, 2000
V. P. Nelson, H.T.Nagle, B.D.Carroll, J.D.Irwin: „Digital Logic Circuit Analysis & Design“, ISBN 0-13-463894-8, 1995
T.L.Floyd: „Digital Fundamentals“, Prentice Hall, ISBN 0-13-080850-4, 2000

INC 2012

Úrovně abstrakce popisu



- **Syntéza**: Automatická transformace mezi různými úrovněmi popisu
  - Transformace na jemnější popis s cílem vylepšit parametry zadané uživatelem: rychlost, spotřeba, rozměry, atd.
  - Splnění požadavků (***constraints***) specifikovaných uživatelem (perioda hodin, zpoždění propojovacích vodičů, atd.)

Behaviorální syntéza

- Z behaviorálního popisu algoritmu je vytvořena reprezentace na úrovni struktury (sčítačka, posuvný registr, paměť, řídicí logika)

Logická syntéza

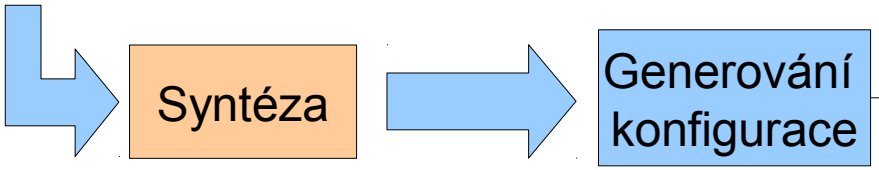
- Z HDL popisu na úrovni RT (meziregistrových přenosů) je vytvořen NetList prvků cílové technologie

Popis obvodu

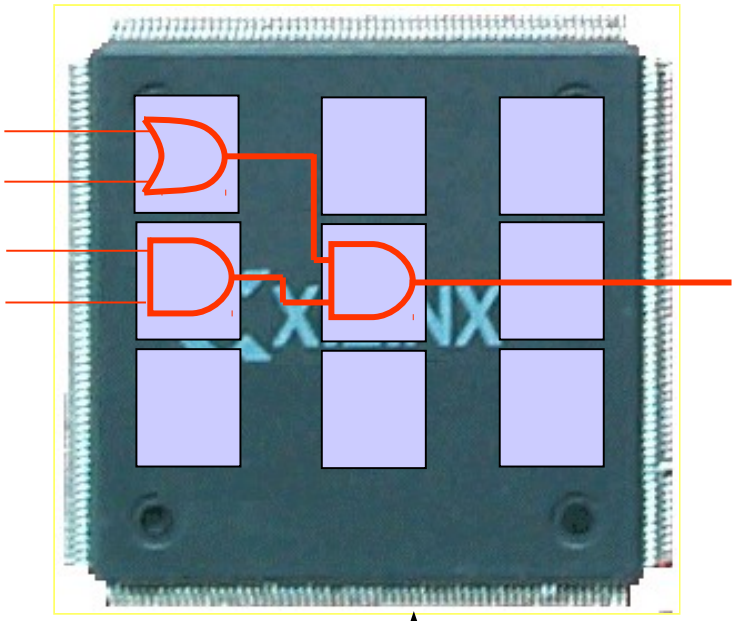
```
library IEEE;
use IEEE.std_logic_1164.all;

entity blk4 is
    port (
        i1: in STD_LOGIC;
        i2: in STD_LOGIC;
        i3: in STD_LOGIC;
        i4: in STD_LOGIC;
        o1: out STD_LOGIC
    );
end blk4;

architecture struc of blk4 is
begin
  o1 <= (i1 or i2) and (i3 and i4);
end struc;
```

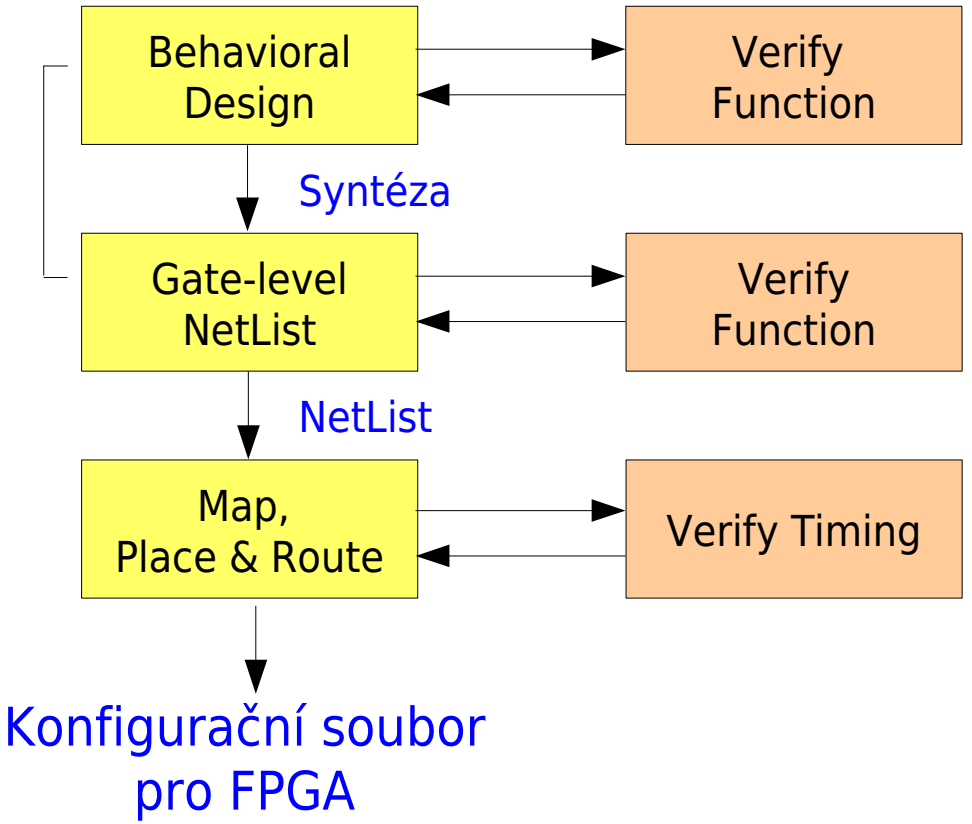


Cílová technologie FPGA



011100101011

Design Flow pro technologii FPGA



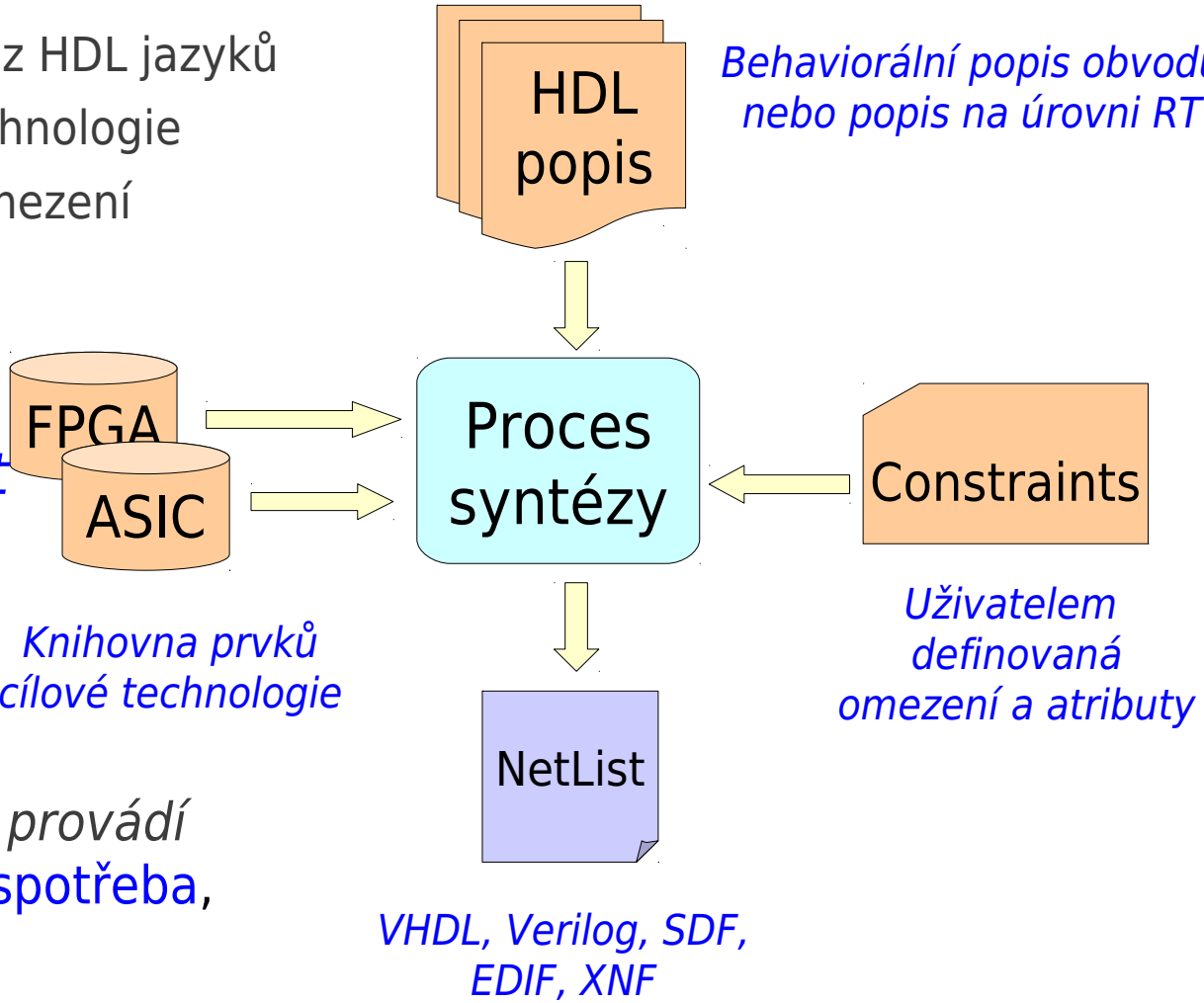
Verifikace na všech úrovních abstrakce

Snaha o zachycení chyb již v prvních fázích vývoje

Proces závislý na cílové technologii

- Rozpoznání prvků cílové technologie a jejich mapování do FPGA
- Výsledkem procesu je konfigurační soubor pro FPGA

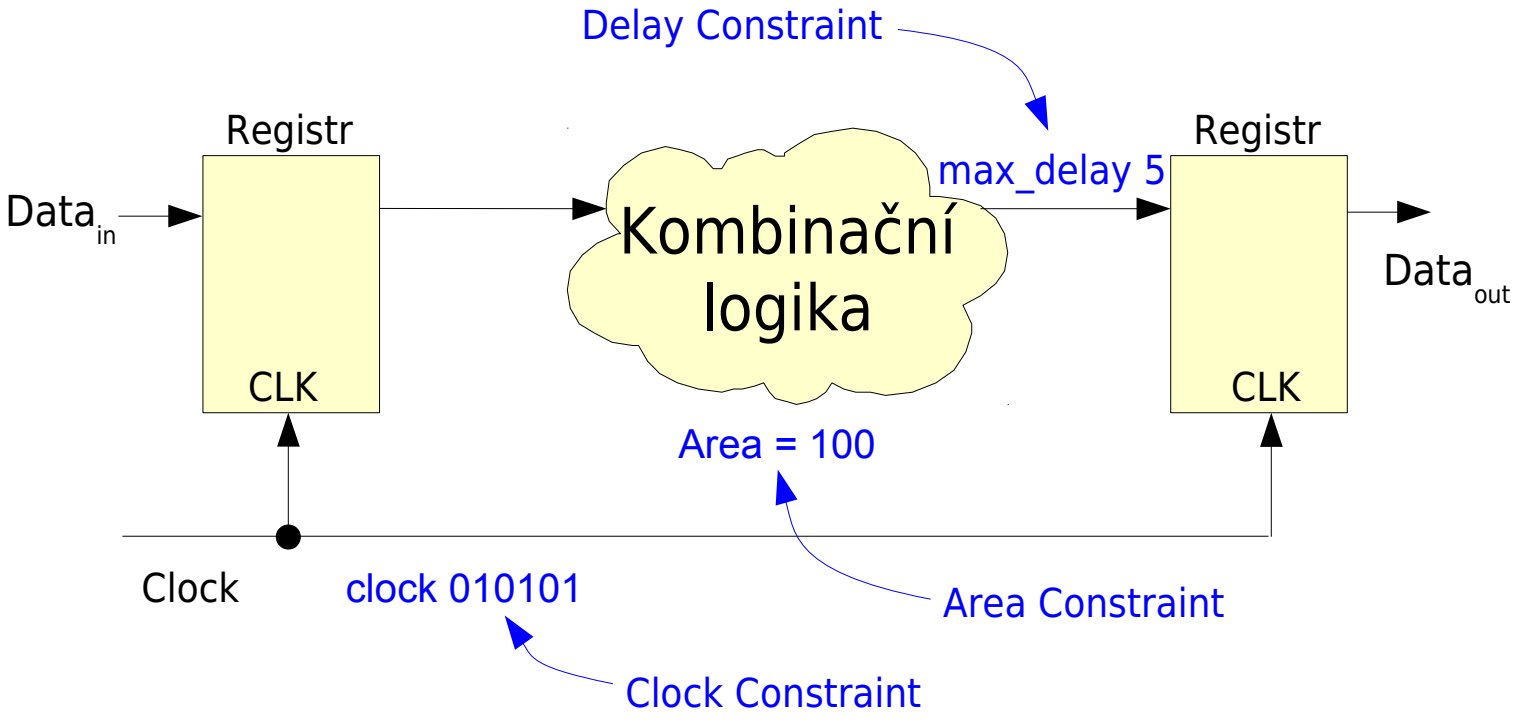
- Vstup syntézy
  - Popis obvodu v některém z HDL jazyků
  - Knihovna prvků cílové technologie
  - Uživatelem definovaná omezení



Proces syntézy vytváří „optimalizovaný“ NetList na úrovni prvků cílové technologie

V průběhu syntézy se provádí optimalizace rychlost, spotřeba, plocha, atd.

- *Constraints* se používají pro řízení optimalizace a mapování
- V současnosti jsou nástroji podporované constraints na čas, spotřebu velikost, umístění na čipu.



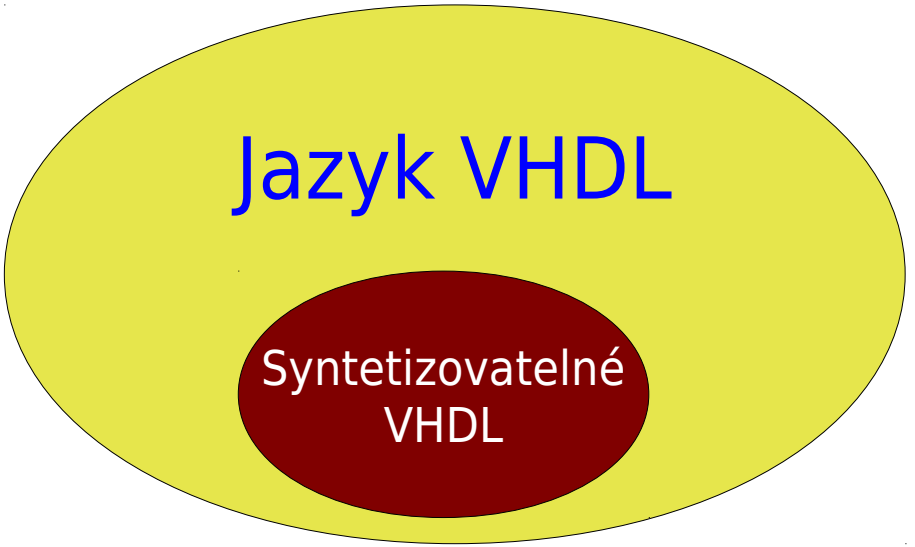
- Obsahuje informace o prvcích cílové technologie – nezbytná informace pro syntézu
- Součástí knihovny je:
  - Logická funkce buňky
  - Zabraná plocha na čipu
  - Časování průchodu signálu ze vstupu na výstup
  - Omezení na fanout
  - Časová omezení dané buňky

Příklad: AND gate

```
Library (x,y,z) {
cell (and2) {
 area : 5;
 pin (a1, a2) {
   direction   : input;
   capacitance : 1;
 }
 pin (o1) {
   direction   : output;
   function    : "a1 & a2";
   timing () {
      intrinsic_rise : 0.37;
      intrinsic_fall : 0.56;
      rise_resistance: 0.1234;
      fall_resistance: 0.4567;
      related_pin    : "a1 a2"
   }
 }
}
}
```

- Syntéza dokáže pracovat pouze s podmnožinou jazyka VHDL – *syntetizovatelné VHDL*

- Problematické konstrukce
  - Čtení nebo zápis ze souboru
  - Rozsah smyček a generických instancí musí být konstantní
  - Zpoždění definované pomocí příkazu wait
  - Příkazy pro ověřování funkce komponent – assert, report, ...
  - Je potřeba dávat pozor i na způsob zápisu



Ne všechny konstrukce napsané ve VHDL je možné syntézou převést na obvodovou realizaci z prvků cílové technologie!

- Nežádoucí vznik registrů typu Latch - ***Inferring latches***
  - Není definovaná hodnota výstupu pro některou z kombinací na vstupu vede na zachování původní hodnoty pomocí latch registru

```
process (sel, a, b, c)
begin
   case sel is
     when "00" => mux_out <= a;
     when "01" => mux_out <= b;
     when "10" => mux_out <= c;
     when others => null;
   end case;
end process;
```

Neošetřené kombinace “11“ na vstupu

Vytvoří se nežádoucí Latch registr!

```
process (sel, a, b, c)
begin
   mux_out <= a;
   case sel is
     when "00" => mux_out <= a;
     when "01" => mux_out <= b;
     when "10" => mux_out <= c;
     when others => null;
   end case;
end process;
```

Správně napsaný kód

- Na senzitivity listu procesu nejsou umístěny všechny vstupní signály procesu (***Incomplete sensitivity list***)
  - Jiné chování obvodu v simulacích a v hardware – chyba se špatně ladí

Chybí signály a,b,c,d

```
process (sel)
begin
   case sel is
      when "00" => mux_out <= a;
      when "01" => mux_out <= b;
      when "10" => mux_out <= c;
      when "11" => mux_out <= d;
      when others => null;
   end case;
end process;
```

V hardware bude multiplexor fungovat, v ModelSimu nebude

- Užitečné pro implementaci generických komponent
- Rozsah cyklů nebo příkazu generate musí být znám v okamžiku syntézy
  - Rozsah logiky asociované ke smyčce nelze v čipu dynamicky měnit
  - Řada syntézních nástrojů podporuje v definici rozsahu pouze typ integer

Příklad:

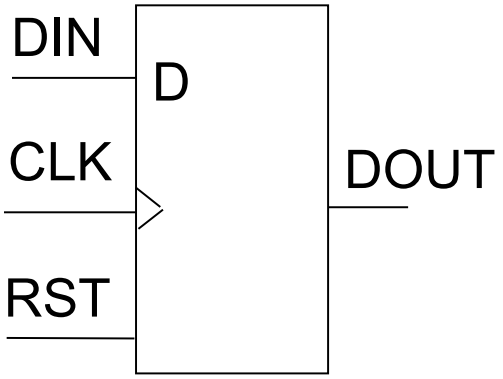
```
process(cnt_bin)
begin
   DO  <= (others => '0');
   for i in 0 to 7 loop
      if (conv_std_logic_vector(i, 3) =
          cnt_bin) then
         DO(i) <= '1';
      end if;
   end loop;
end process;
```

Převod binárního čísla do kódu 1 z n

- Je zaručeno rozpoznání komponent syntézou
- Použití šablon dává vývojáři kontrolu nad syntézním nástrojem a dává prostor pro optimalizace
- Šablony pokrývají základní prvky z úrovně RT
  - Klopný obvod typu D
  - Multiplexor a dekodér
  - Čítač, sčítačka, případně násobička nebo dělička
  - Automat (FSM)

Klopný obvod typu D

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dffx is
port (
   CLK   : in  std_logic;
   RST   : in  std_logic;
   DIN   : in  std_logic;
   DOUT  : out std_logic );
end dffx;
architecture behav of dffx is
begin
   process (CLK,RST)
   begin
      if (CLK'event and CLK = '1') then
         if (RST = '1') then
            DOUT <= '0';
         else
            DOUT <= DIN;
         end if;
      end if;
   end process;
end  behav;
```



CLK (clock) – hodinový vstup

RST(reset) – asynchronní reset

DIN (data in) – data přivedená na vstup registru

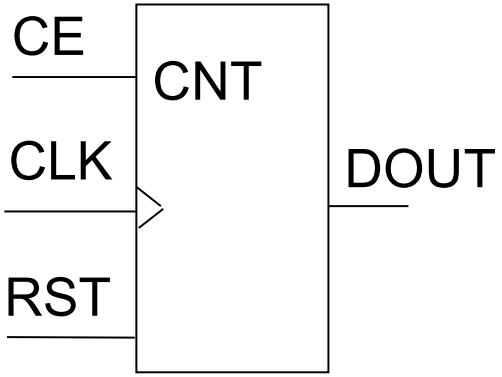
DOUT (data output) – hodnota uložená v registru

Synchronní reset

(Nulování synchronizováno s hodinovým signálem)

Čítač

```
library IEEE;
use IEEE.std_logic_1164.all;
entity counter is
port (
   CLK    : in  std_logic;
   RST    : in  std_logic;
   CE     : in  std_logic;
   DOUT   : out std_logic_vector(7 downto 0)
end counter;
architecture behav of counter is
begin
   process (CLK,RST,CE)
   begin
      if (RST = '1') then
         DOUT <= (others => '0');
      elsif (CLK'event and CLK = '1') then
         if CE='1' then
            DOUT <= DOUT + '1';
         end if;
      end if;
   end process;
end  behav;
```



CLK (clock) – hodinový vstup

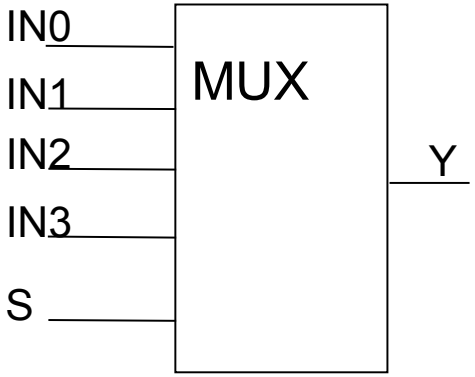
RST(reset) – reset

CE (count enable) – povolení čítání

DOUT (data output) – hodnoty čítače

Multiplexor

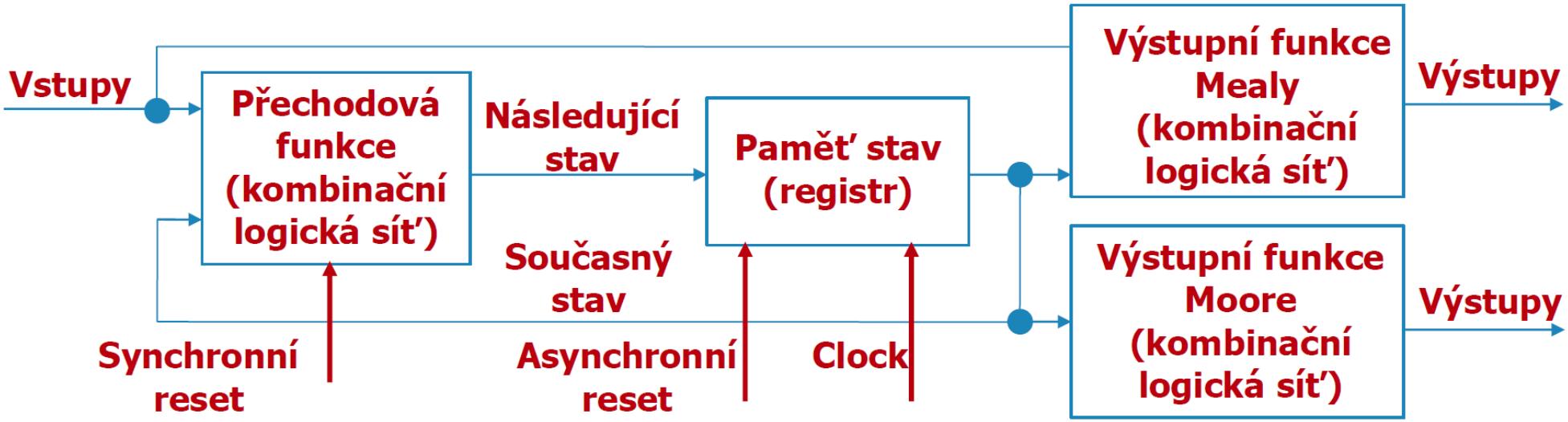
```
library ieee;
use ieee.std_logic_1164.all;
entity Mux is
port( I3: in  std_logic_vector(2 downto 0);
      I2: in  std_logic_vector(2 downto 0);
      I1: in  std_logic_vector(2 downto 0);
      I0: in  std_logic_vector(2 downto 0);
      S : in  std_logic_vector(1 downto 0);
      O : out std_logic_vector(2 downto 0));
end Mux;
architecture behv1 of Mux is
begin
   process(I3,I2,I1,I0,S)
   begin
      case S is
         when "00" =>    O <= I0;
         when "01" =>    O <= I1;
         when "10" =>    O <= I2;
         when "11" =>    O <= I3;
         when others =>  O <= "ZZZ";
      end case;
   end process;
end behv1;
```



IN0, IN1,
IN2, IN3 – přepínané vstupy
S – řídící hradlo
Y – výstup multiplexoru

- Konečný automat tvoří tři základní části:
  - registr pro uložení aktuálního stavu
  - logika následujícího stavu
  - logika pro generování výstupu (Moore, Mealy)

- Každé této části odpovídá ve VHDL jeden proces



Implementace automatu

```
proc_cstate : process (CLK, RST,
                       next_state)
begin
    if RST = '1' then
        cur_state <= s_idle;
    elsif CLK'event AND CLK='1' then
        cur_state <= next_state;
    end if;
end process proc_cstate;
```

```
output_logic : process (cur_state)
begin
   DNEXT <= '0';
   ACK   <= '0';
   case cur_state is
      when s_data =>
         ACK   <= '1';
      when s_next =>
         DNEXT <= '1';
      when others =>
         null;
   end case;
end process output_logic;
```

```
nstate_logic : process (cur_state, RQ,
                        DRDY)
begin
   next_state <= s_idle;

   case cur_state is
      -- ----- stav IDLE -----
      when s_idle =>
         if RQ='1' then
            next_state <= s_wait;
         else
            next_state <= s_idle;
         end if;
      -- ----- stav WAIT -----
      ...
      -- ----- stav DATA -----
      ...
      -- ----- stav NEXT -----
      when s_next =>
         next_state <= s_idle;
      when others =>
         null;
   end case;
end process nstate_logic;
```