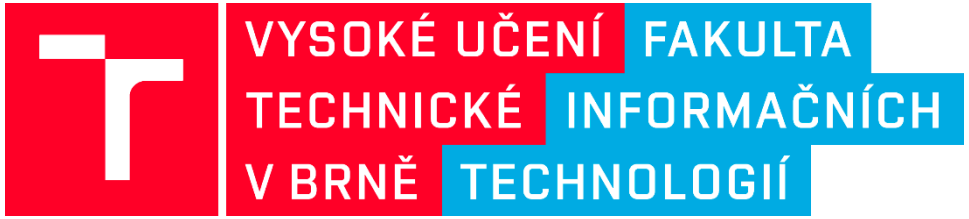
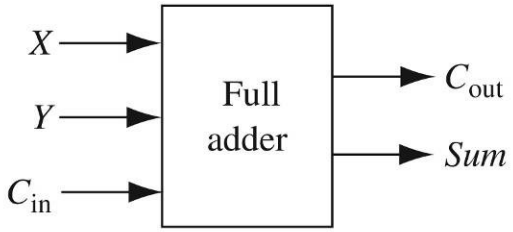
Příklady popisu obvodů ve VHDL

2025

Vojtěch Mrázek
mrazek@fit.vutbr.cz



Úplná sčítačka (dataflow popis)

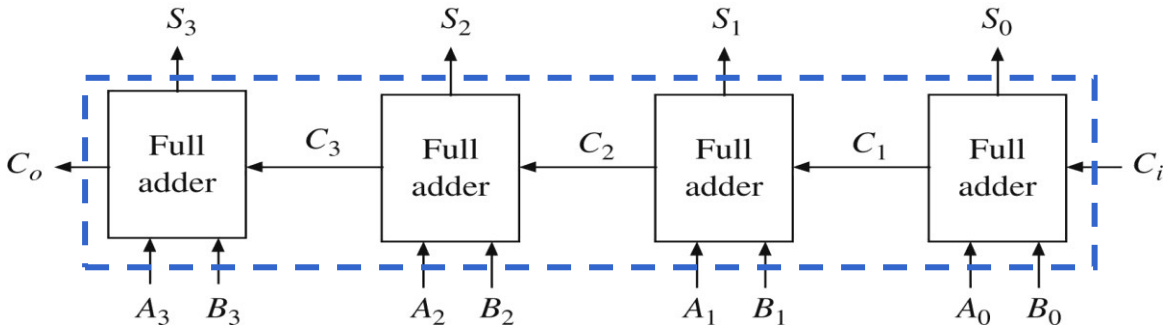



```
entity FullAdder is
  port(X, Y, Cin: in bit;     --Inputs
       Cout, Sum: out bit);   --Outputs
end FullAdder;
```

```
architecture dataflow of FullAdder is
begin

  Sum  <= X xor Y xor Cin after 1 ns;
  Cout <= '1' when (X and Y) or (X and Cin) or (Y and Cin) else
          '0' after 1 ns;

end architecture;
```

- využíváme pouze konstrukce přiřazení signálů
- lze uvést podmínku (when) – vhodné pro popis multiplexoru, dekodéru, ... případně přiřadit zpoždění (after) – pro potřeby simulace

4-bitová sčítačka (strukturní popis)

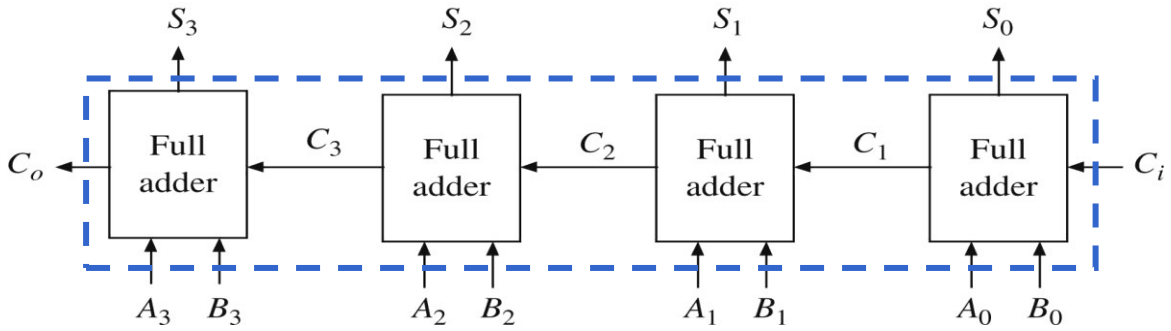



```
entity Adder4 is
  port(
    A, B: in std_logic_vector(3 downto 0);
        Ci: in std_logic;
        S: out std_logic_vector(3 downto 0);
        Co: out std_logic);
end Adder4;
```

```
architecture structure of Adder4 is
  component FullAdder
   port(X, Y, Cin: in std_logic; Cout, Sum: out std_logic);
  end component;
  signal C: std_logic_vector(3 downto 1);
begin

  FA0: FullAdder port map(A(0),B(0),Ci,C(1),S(0));
  FA1: FullAdder port map(A(1),B(1),C(1),C(2),S(1));
  FA2: FullAdder port map(A(2),B(2),C(2),C(3),S(2));
  FA3: FullAdder port map(X <= A(3),Y <= B(3),
                          Cin <= C(3),
                          Sum <= S(3), Cout <= Co);
end structure;
```

- využití kontrukce **component** a **port map** (poziční ne/závislost)
- analogie konstrukce číslicových systémů ze součástek – propojovaní komponent (součástek) signály (dráty)
- komponenta jako blackbox – pracovat může více vývojářů současně

4-bitová sčítačka (behaviorální popis)




```
entity Adder4 is
  port(
    A, B: in std_logic_vector(3 downto 0);
       Ci: in std_logic;
        S: out std_logic_vector(3 downto 0);
        Co: out std_logic);
end Adder4;
```

```
architecture behavioral of Adder4 is
begin
  process (A,B,Ci)
  variable t: std_logic_vector
              (4 downto 0);
  begin
    t := ('0'&A) + ('0'&B) + Ci;
    S <= t(3 downto 0);
    Co <= t(4);
  end process;
end behavioral;
```

```
use IEEE.std_logic_unsigned.all;
```

- využití procesů
- pozor na sensitivity list (v případě kombinační funkce musí být uveden seznam všech signálů ze kterých je odvozen výsledek)

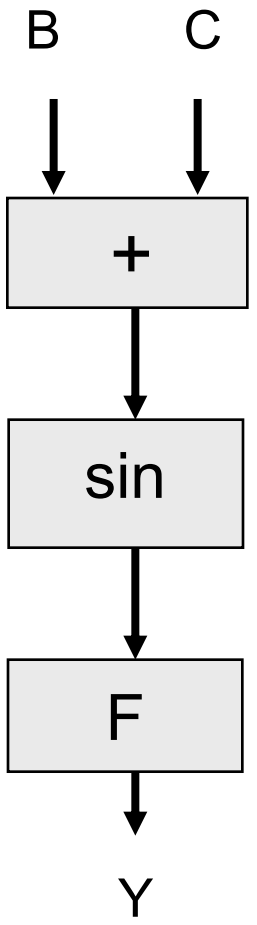
Implementace základních konstrukcí v SW a HW

sekvence (posloupnost operací)

Sekvence příkazů

```
A = B + C
D = sin(A)
Y = F(D)
```

(3 kroky)



pouze datová cesta

Implementace základních konstrukcí v SW a HW

selekce (větvení výpočtu)

Podmíněné vykonávání

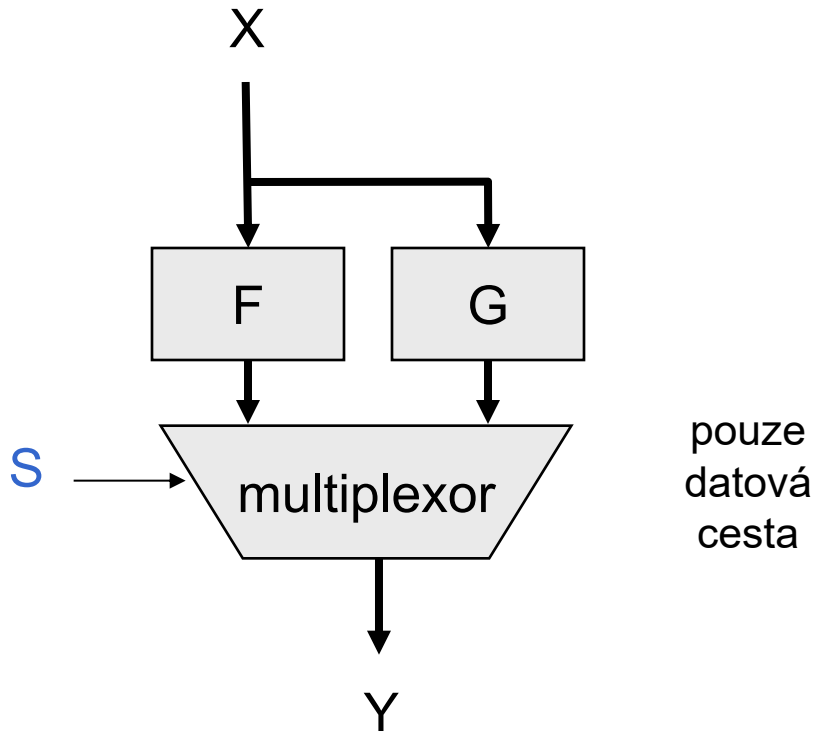
```
If (S == 0):
  Y = F(X)
else:
  Y = G(X)
```

(3~4 instrukce CPU)

vs.

```
Y = G(X)
If (S == 0):
  Y = F(X)
```

(3~5 instrukce CPU)



pouze datová cesta

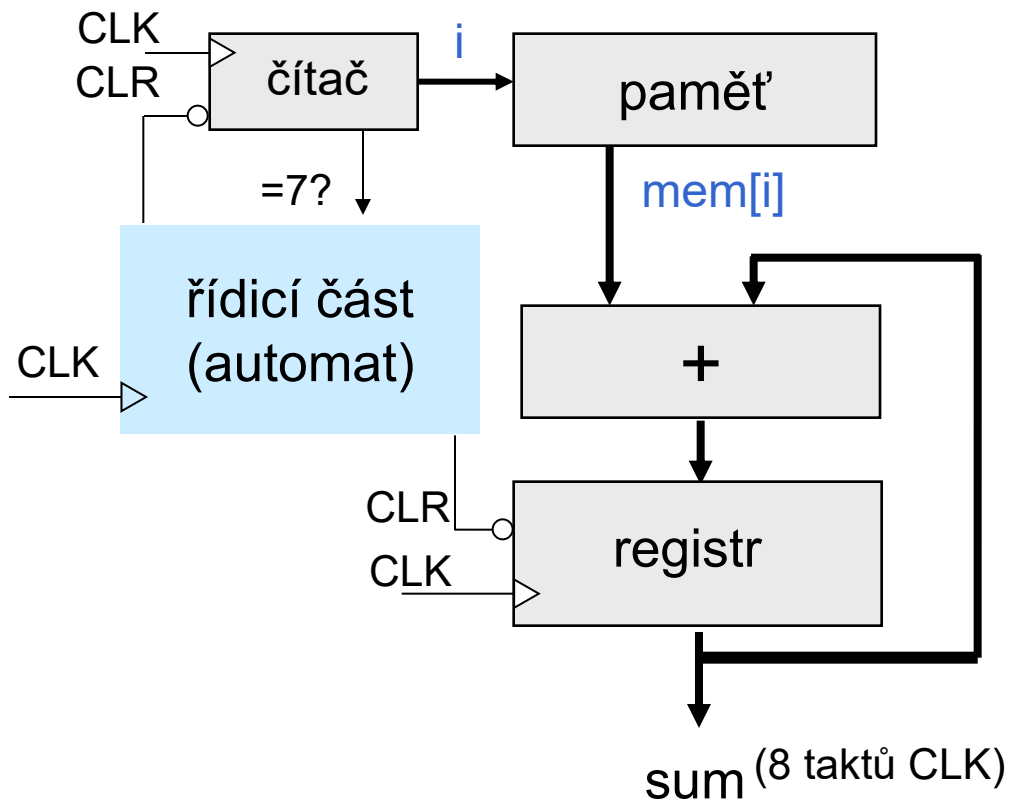
Implementace základních konstrukcí v SW a HW

iterace (opakování výpočtu)

Iterativní zpracování (smyčky)

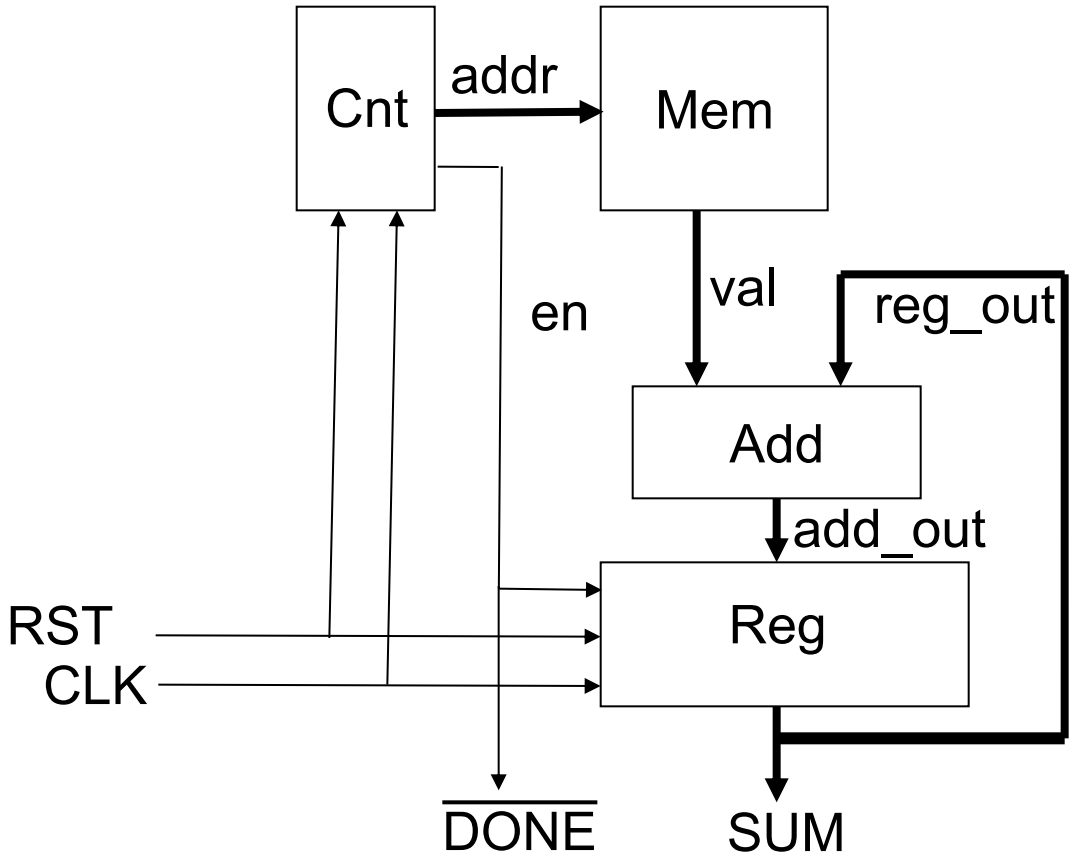
```
sum = 0
i = 0
while (i < 8):
    sum = sum + mem[i]
    i += 1
```

(cca 40 instrukcí CPU)



Obvod realizující iterační součet posloupnosti

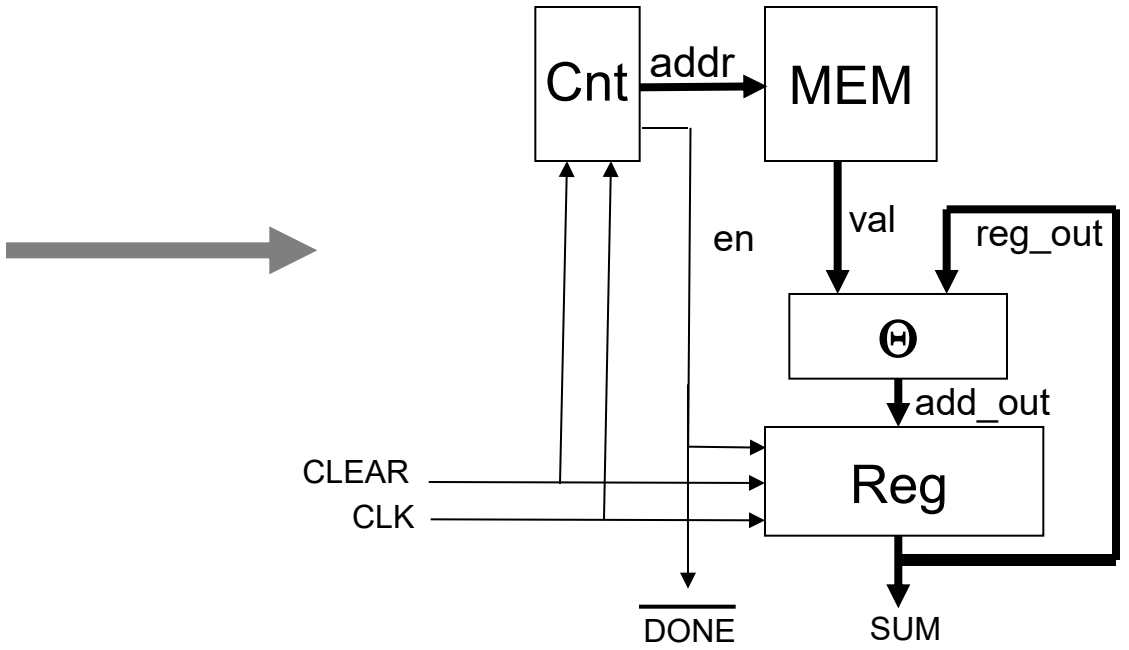
- Čítač (Cnt) generuje adresy celého rozsahu paměti (Mem), zastaví se u adresy 7
- Hodnoty (val) z paměti jsou sčítány sčítačkou (Add) a mezivýsledek uchován v registru (Reg)
- Signál EN, zabraňující přepsání finálního výsledku v Reg, je deaktivován po vygenerování nejvyšší adresy paměti
- Signál RST v log. 1 způsobí nulování registru a čítače
- CLK synchronizuje činnost sekvenčních obvodů – čítače a registru
- Po kompletním průchodu pamětí obsahuje Reg (a jeho výstup – signál SUM) součet všech jejích hodnot a signál DONE je aktivní (v log. 0)



Návrh obvodu realizujícího iterační součet posloupnosti

- Cílem je navrhnout primitivní obvod, který je schopen vypočítat součet hodnot N (např. 8) prvků umístěných v paměti.
- Praktické použití (po rozšíření): kontrola a generování CRC součtu

```
const N = 8
char mem[N] = {1,…};
char i
char sum;

sum = 0
for (i=0;i<N;i++)
   sum = Θ(sum, mem[i])
```



- Jak převedeme pseudo kód na HW implementaci?
  Stačí umět převést sekvenci, selekci, iteraci.

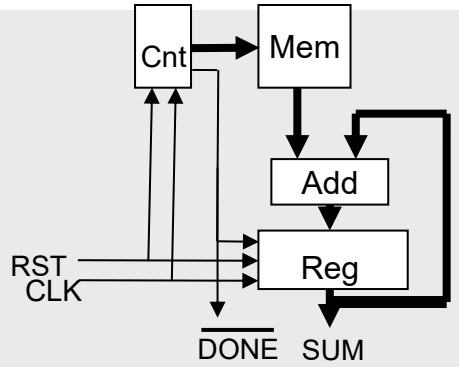
Příklad na papír

Sekvenční sčítačka s využitím komponent

(strukturní popis)

```
entity Acc is
port (
   CLK: in std_logic;
   RST: in std_logic;
   SUM: out std_logic_vector(7 downto 0);
  DONE: out std_logic
);
end Acc;

architecture struct of Acc is

component Add is
port (
   A: in std_logic_vector(7 downto 0);
   B: in std_logic_vector(7 downto 0);
   RES: out std_logic_vector(7 downto 0)
);
end component;

component Cnt is
port (
   CLK: in std_logic;
   RST: in std_logic;
   ADDR: out std_logic_vector(2 downto 0);
   STOP: out std_logic
);
end component;

component Mem is
port (
   ADDR: in std_logic_vector(2 downto 0);
   VAL: out std_logic_vector(7 downto 0)
);
end component;
```

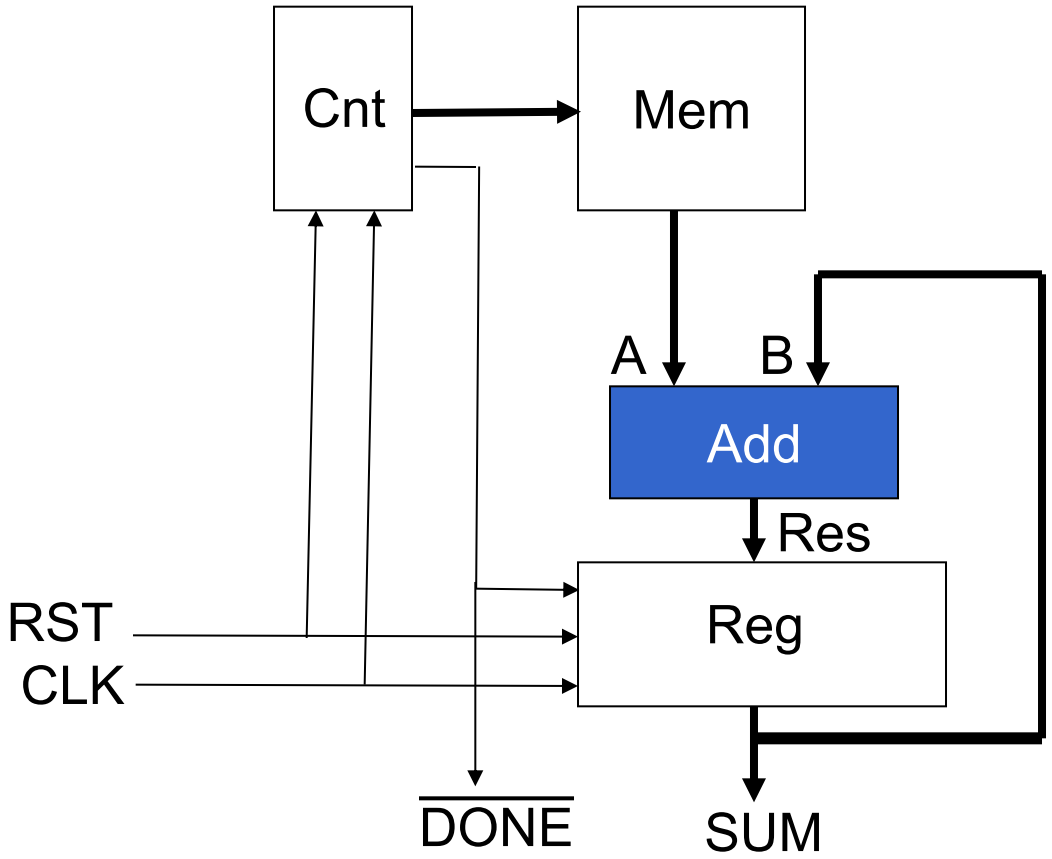
```
component Reg is
port (
   CLK: in std_logic;
   RST: in std_logic;
   EN: in std_logic;
   DIN: in std_logic_vector(7 downto 0);
   DOUT: out std_logic_vector(7 downto 0)
);
end component;

signal addr: std_logic_vector(2 downto 0);
signal val: std_logic_vector(7 downto 0);
signal add_out: std_logic_vector(7 downto 0);
signal reg_out: std_logic_vector(7 downto 0);
signal en: std_logic;

begin

   add_inst: Add port map(A => val, B => reg_out, RES => add_out);
   mem_inst: Mem port map(ADDR => addr, VAL => val);
   cnt_inst: Cnt port map(CLK, RST, addr, en);
   reg_inst: Reg port map(CLK, RST, en, add_out, reg_out);

   SUM <= reg_out;
   DONE <= en;

end struct;
```



Model sčítačky (komponenta Add)

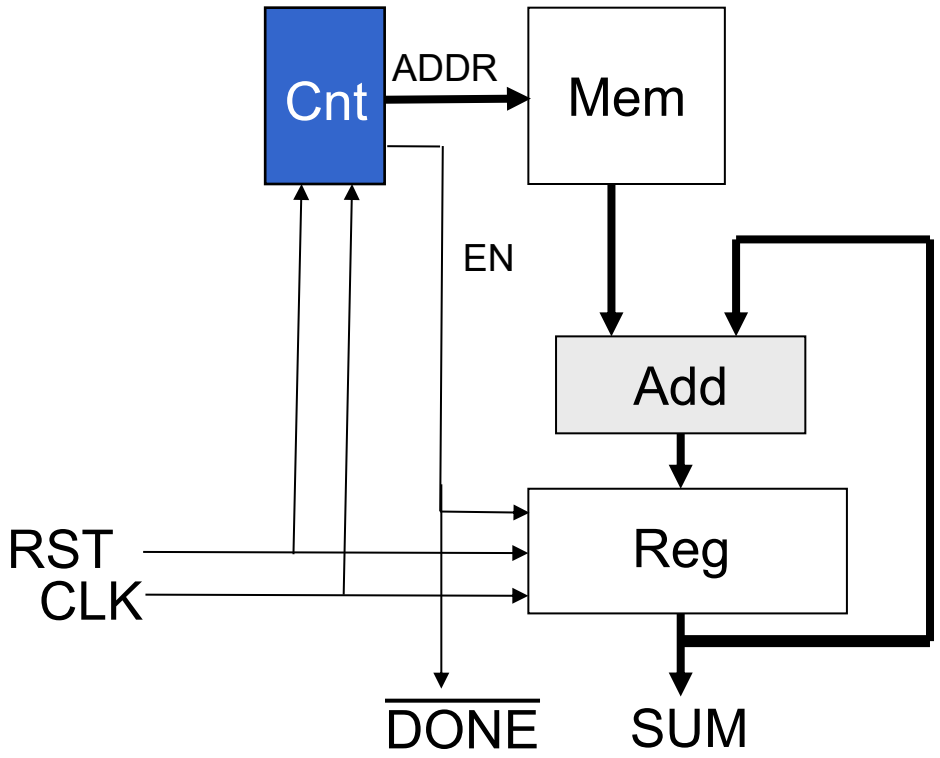
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
```

```
entity Add is
port (
   A: in std_logic_vector(7 downto 0);
   B: in std_logic_vector(7 downto 0);
   RES: out std_logic_vector(7 downto 0)
);
end Add;

architecture behav_add of Add is
begin
   RES <= A + B;
end behav_add;
```



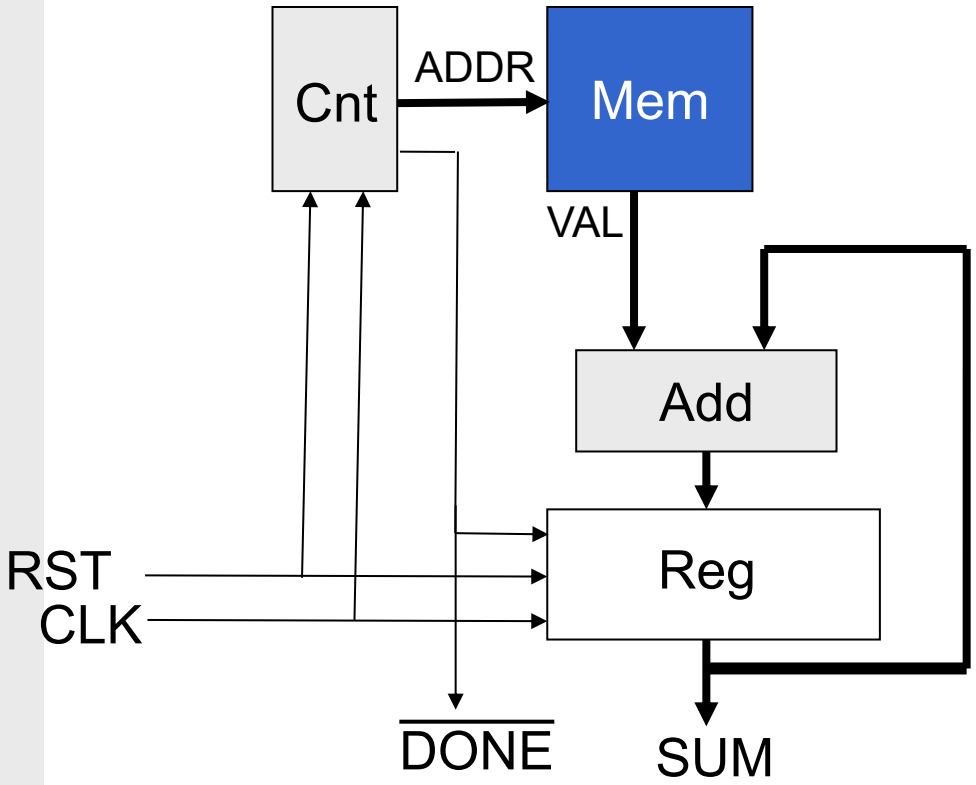
Model čítače (komponenta Cnt)

```
entity Cnt is
port (
   CLK: in std_logic;
   RST: in std_logic;
   ADDR: out std_logic_vector(2 downto 0);
   EN: out std_logic
);
end Cnt;

architecture behav of Cnt is
   signal cnt: std_logic_vector(2 downto 0);
begin
   cnt_proc: process(CLK, RST)
   begin
      if RST = '1' then
         cnt <= "000";
         EN <= '1';
      elsif CLK'event and CLK = '1' then
         EN <= '1';
         if cnt = "111" then
            EN <= '0';
         else
            cnt <= cnt + 1;
         end if;
      end if;
   end process;

   ADDR <= cnt;
end behav;
```



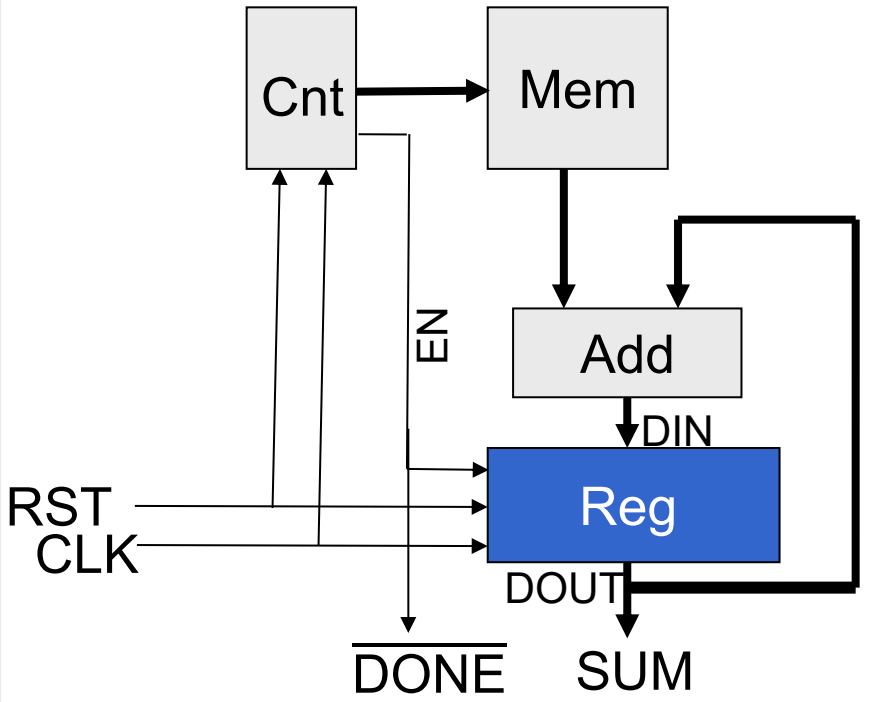
Model ROM paměti (komponenta Mem)

```
entity Mem is
port (
   ADDR: in std_logic_vector(2 downto 0);
   VAL: out std_logic_vector(7 downto 0)
);
end Mem;

architecture behav_mem of Mem is

begin
   mem: process(ADDR)
   begin
      case ADDR is
         when "000" => VAL <= "00000001";
         when "001" => VAL <= "00000010";
         when "010" => VAL <= "00000011";
         when "011" => VAL <= "00000111";
         when "100" => VAL <= "00001111";
         when "101" => VAL <= "00011111";
         when "110" => VAL <= "00111111";
         when "111" => VAL <= "01111111";
         when others => VAL <= "00000000";
      end case;
   end process;
end behav_mem;
```



Model registru (komponenta Reg)

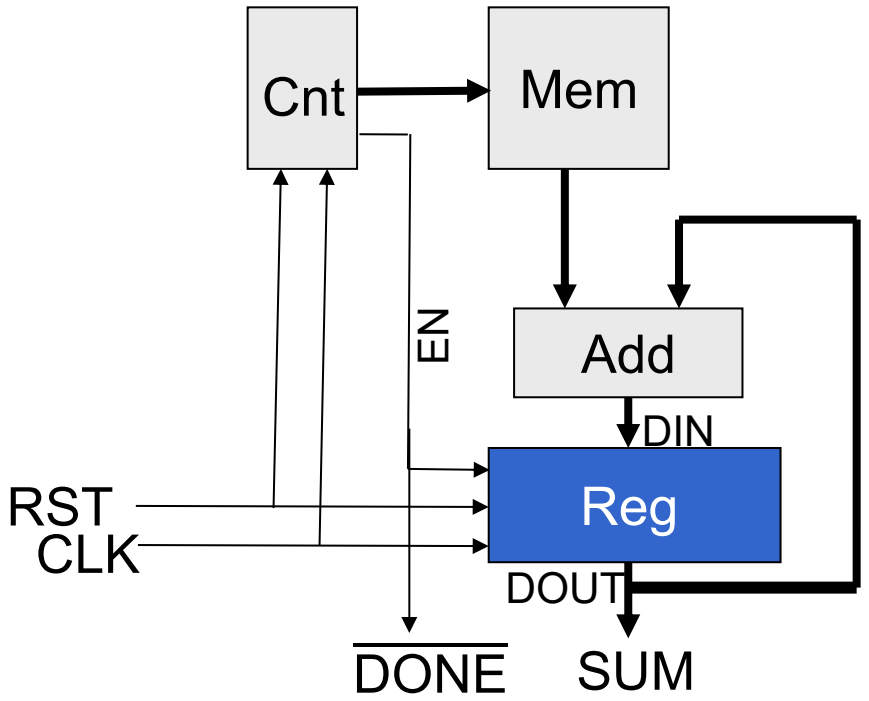
```
entity Reg is
port (
   CLK: in std_logic;
   RST: in std_logic;
   EN: in std_logic;
   DIN: in std_logic_vector(7 downto 0);
   DOUT: out std_logic_vector(7 downto 0)
);
end Reg;

architecture behav of Reg is

begin
   reg: process(CLK, RST)
   begin
      if RST = '1' then
         DOUT <= "00000000";
      elsif CLK'event and CLK = '1' then
         if EN='1' then
            DOUT <= DIN;
         end if;
      end if;
   end process;
end behav;
```



Model registru (komponenta Reg)

Příklad využití generických proměnných

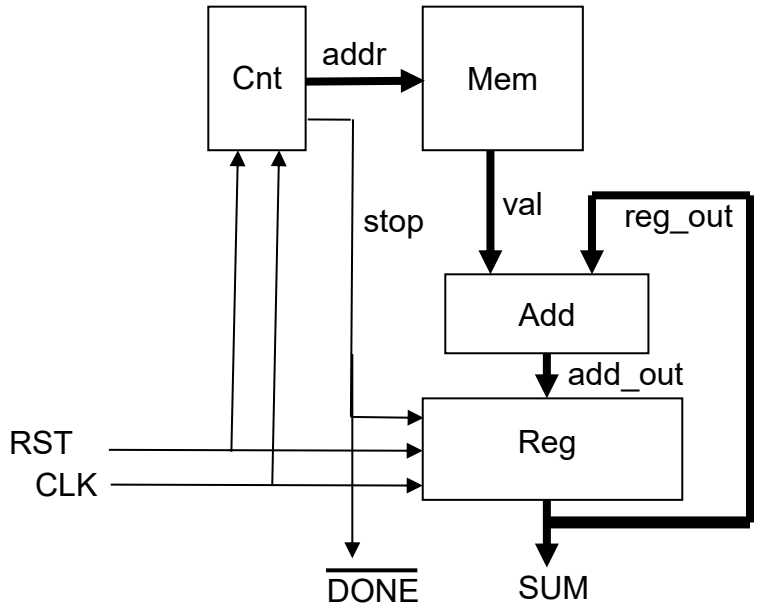
```
entity Reg is
generic (
   DATA_WIDTH : integer := 8
)
port (
   CLK: in std_logic;
   RST: in std_logic;
   EN: in std_logic;
   DIN: in std_logic_vector(DATA_WIDTH-1 downto 0);
   DOUT: out std_logic_vector(DATA_WIDTH-1 downto 0)
);
end Reg;

architecture behav of Reg is

begin
   reg: process(CLK, RST)
   begin
      if RST = '1' then
         DOUT <= (others => '0');
      elsif CLK'event and CLK = '1' then
         if EN='1' then
            DOUT <= DIN;
         end if;
      end if;
   end process;
end behav;
```



Sekvenční sčítačka popsaná behaviorálně

```
entity Acc is
port (
   CLK: in std_logic;
   RST: in std_logic;
   SUM: out std_logic_vector(7 downto 0);
   DONE: out std_logic
);
end Acc;

architecture behav of Acc is

signal addr: std_logic_vector(2 downto 0);
signal val: std_logic_vector(7 downto 0);
signal add_out: std_logic_vector(7 downto 0);
signal reg_out: std_logic_vector(7 downto 0);
signal stop: std_logic;

begin

   cnt: process(CLK, RST)
   begin
      if RST = '1' then
         addr <= "000";
         stop <= '0';
      elsif clk'event and clk = '1' then
         if addr = "111" then
            stop <= '1';
         else
            addr <= addr + 1;
            stop <= '0';
         end if;
      end if;
   end process;
```

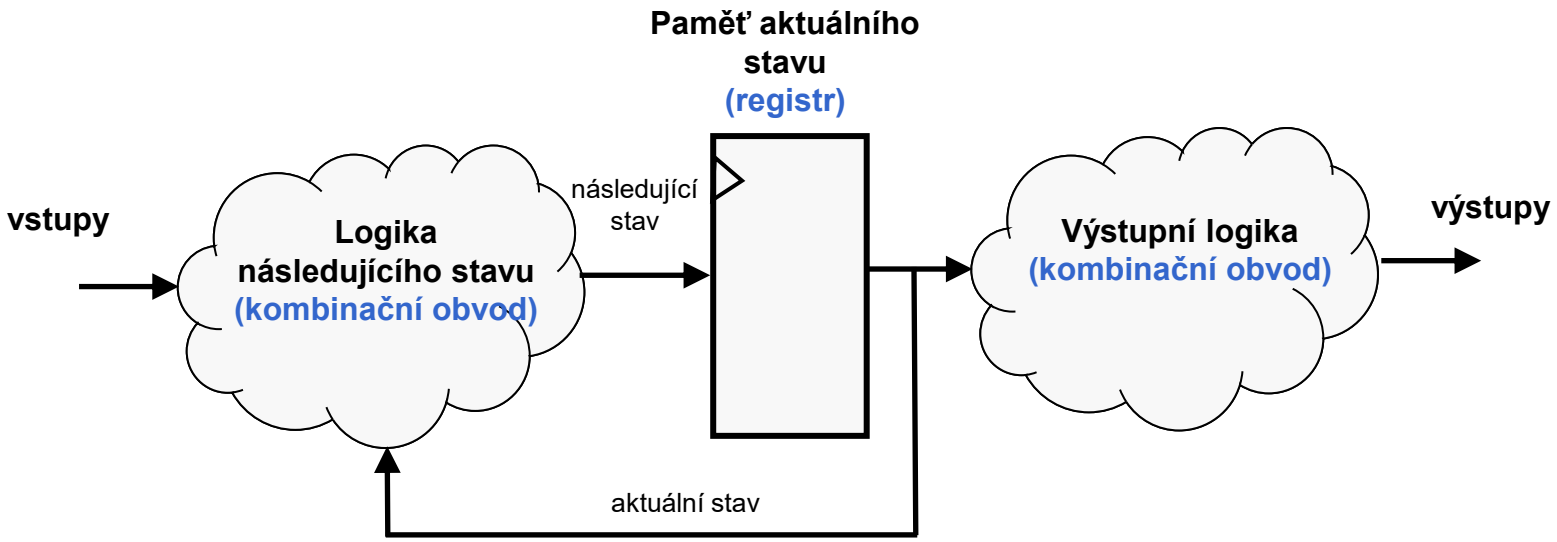
```
  reg: process(CLK, RST)
    begin
       if RST = '1' then
          reg_out <= X"00";
       elsif clk'event and
             clk = '1'
       then
          if stop = '0' then
             reg_out <= add_out;
          end if;
       end if;
    end process;




    with addr select
       val <= "00000001" when "000",
              "00000010" when "001",
              "00000011" when "010",
              "00000111" when "011",
              "00001111" when "100",
              "00011111" when "101",
              "00111111" when "110",
              "01111111" when "111",
              "00000000" when others;

    add_out <= reg_out + val;

    SUM <= reg_out;
    DONE <= not stop;

end behav;
```



Připomenutí principů modelování konečných automatů ve VHDL

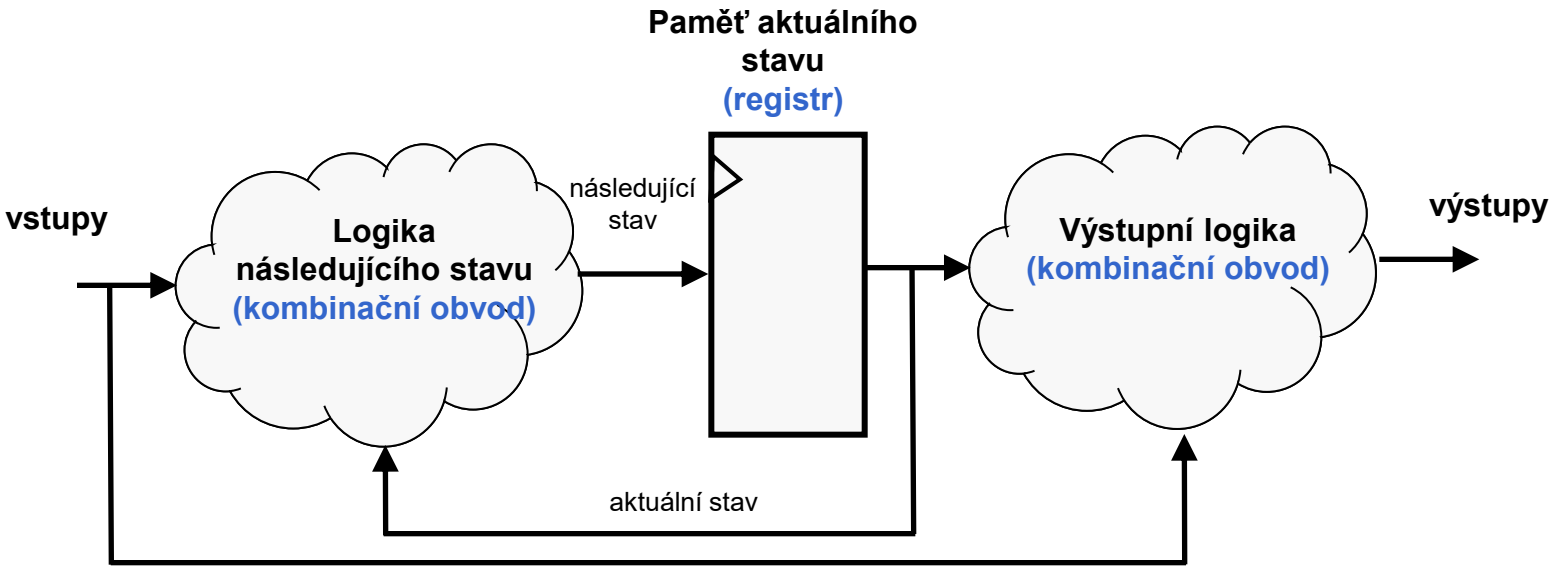
Typy konečných automatů

- Obvod mající paměť lze chápat jako konečný stavový automat (FSM)
- Návrh automatu vyžaduje
  - definovat stavy automatu, přechody mezi stavy a výstupní funkci
- Podle způsobu realizace výstupu rozlišujeme dva základní FSM
  - Moore (výstup je pouze funkcí stavu),
  - Mealy (výstup je funkcí stavu a vstupů)



Typy konečných automatů

- Obvod mající paměť lze chápat jako konečný stavový automat (FSM)
- Návrh automatu vyžaduje
  - definovat stavy automatu, přechody mezi stavy a výstupní funkci
- Podle způsobu realizace výstupu rozlišujeme dva základní FSM
  - Moore (výstup je pouze funkcí stavu),
  - Mealy (výstup je funkcí stavu a vstupů)



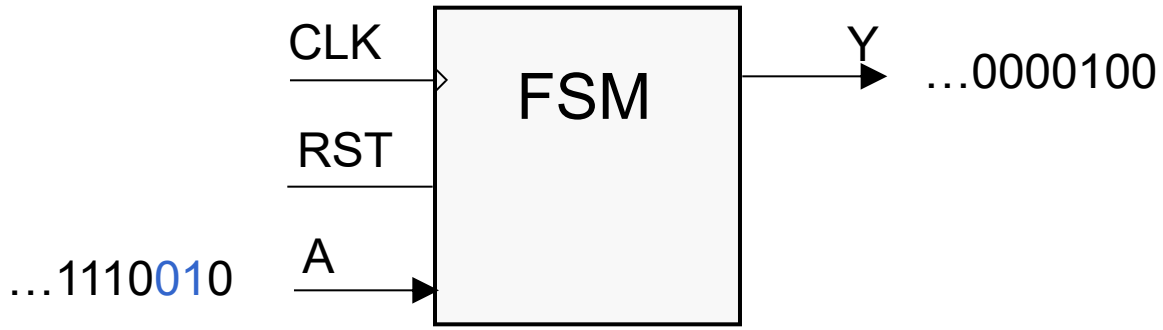
Implementace konečných automatů ve VHDL

Příklad: detektor posloupnosti 10

- VHDL implementace konečných automatů
  - FSM lze ve VHDL popsat pomocí dvou nebo tří procesů

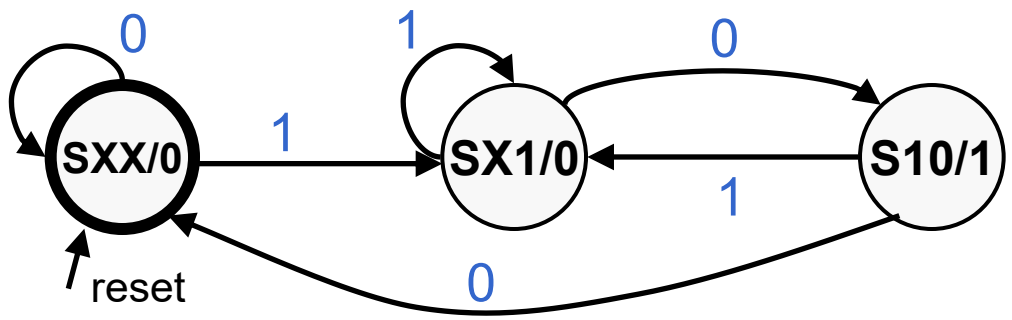
- Příklad FSM: detektor posloupnosti 10 v řetězci
  - výstupem je 1 pokud byla detekována posloupnost 10

příklad vstupu: 0100111011111010100110 11



odpovídající výstup: 0010000100000101010000100

Detektor – Moore FSM – 2 procesy



Deklarace signálů

```
type FSMstate is (SXX, SX1, S10);
signal pstate : FSMstate;
signal nstate : FSMstate;
```

```
--Present State register
pstatereg: process(RST, CLK)
begin
   if (RST='1') then
      pstate <= SXX;
   elsif (CLK'event) and (CLK='1') then
      pstate <= nstate;
   end if;
end process;
```

```
--Next State logic + output logic
nstate_logic: process(pstate, A)
begin
   nstate <= SXX;
   Y <= '0';
   case pstate is
      when SXX =>
           if (A = '1') then
              nstate <= SX1;
           end if;
      when SX1 =>
           nstate <= SX1;
           if (A = '0') then
              nstate <= S10;
           end if;
      when S10 =>
           nstate <= SX1;
           Y <= '1';
           if (A = '0') then
              nstate <= SXX;
           end if;
      when others => null;
   end case;
end process;
```

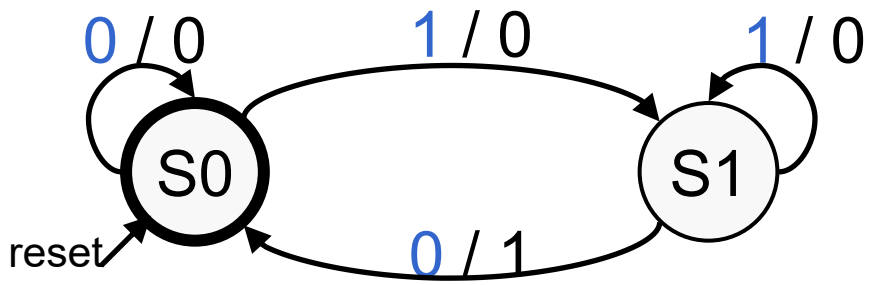
Detektor – Moore FSM – 3 procesy

```
--Present State register
pstatereg: process(RST, CLK)
begin
   if (RST='1') then
      pstate <= SXX;
   elsif (CLK'event) and (CLK='1') then
      pstate <= nstate;
   end if;
end process;
```

```
--Output logic
output_logic: process(pstate)
begin
   Y <= '0';
   case pstate is
      when S10 =>
           Y <= '1';
      when others => null;
   end case;
end process;
```

```
--Next State logic
nstate_logic: process(pstate, A)
begin
   nstate <= SXX;
   case pstate is
      when SXX =>
           if (A = '1') then
              nstate <= SX1;
           end if;
      when SX1 =>
           nstate <= SX1;
           if (A = '0') then
              nstate <= S10;
           end if;
      when S10 =>
           nstate <= SX1;
           if (A = '0') then
              nstate <= SXX;
           end if;
      when others => null;
   end case;
end process;
```

Detektor – Mealy FSM – 2 procesy



Deklarace signálů

```
type FSMstate is (S0, S1);
signal pstate : FSMstate;
signal nstate : FSMstate;
```

```
--Present State register
pstatereg: process(RST, CLK)
begin
   if (RST='1') then
      pstate <= S0;
   elsif (CLK'event) and (CLK='1') then
      pstate <= nstate;
   end if;
end process;
```

```
--Next State logic + output logic
nstate_logic: process(pstate, A)
begin
   nstate <= S0;
   Y <= '0';
   case pstate is
      when S0 =>
           if (A = '1') then
              nstate <= S1;
           end if;
      when S1 =>
           nstate <= S1;
           if (A = '0') then
              Y <= '1';
              nstate <= S0;
           end if;
      when others => null;
   end case;
end process;
```

Detektor – Mealy FSM – 3 procesy

```
--Present State register
pstatereg: process(RST, CLK)
begin
   if (RST='1') then
      pstate <= S0;
   elsif (CLK'event) and (CLK='1') then
      pstate <= nstate;
   end if;
end process;
```

```
--Output logic
output_logic: process(pstate, A)
begin
   Y <= '0';
   if (pstate = S1) and (A = 0) then
      Y <= '1';
   end if;
end process;
```

```
--Next State logic + output logic

nstate_logic: process(pstate, A)
begin
   nstate <= S0;
   case pstate is
      when S0 =>
           if (A = '1') then
              nstate <= S1;
           end if;
      when S1 =>
           nstate <= S1;
           if (A = '0') then
              nstate <= S0;
           end if;
      when others => null;
   end case;
end process;
```

Pozor na sensitivity list u výstupní logiky

Detektor – Mealy FSM – 3 procesy

```
--Present State register
pstatereg: process(RST, CLK)
begin
   if (RST='1') then
      pstate <= S0;
   elsif (CLK'event) and (CLK='1') then
      pstate <= nstate;
   end if;
end process;
```

Výstupní logika nemusí být nutně popsána pomocí procesu

```
--Output logic
Y <= '1' when (pstate = S1) and (A = 0) else
     '0';
```

```
--Next State logic + output logic

nstate_logic: process(pstate, A)
begin
   nstate <= S0;
   case pstate is
      when S0 =>
           if (A = '1') then
              nstate <= S1;
           end if;
      when S1 =>
           nstate <= S1;
           if (A = '0') then
              nstate <= S0;
           end if;
      when others => null;
   end case;
end process;
```