

Návrh obvodů a jazyk VHDL

Jan Kořenek

Brno University of Technology, Faculty of Information Technology

Božetěchova 2, 612 66 Brno

korenek@fit.vutbr.cz

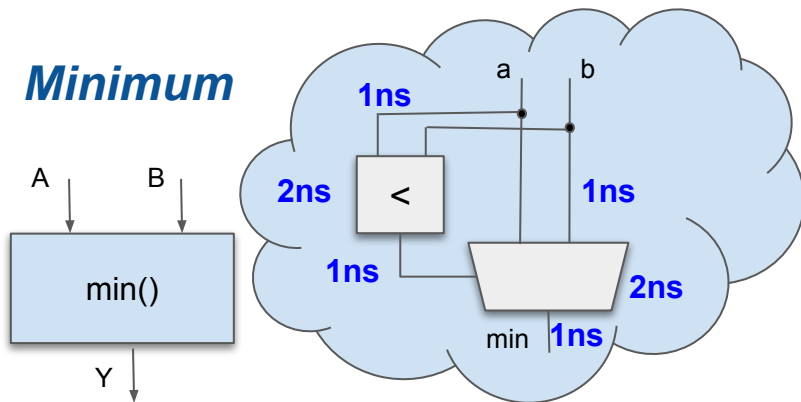


- ***Příklad návrhu obvodů***
- Metodika návrhu
- Jazyk VHDL
- Analýza popisu obvodů

- Navrhnete obvod, který vybere maximální a minimální prvek z N-prvkového pole uloženého v paměti.
 - Kolik taktů hodin bude trvat zpracování 1024 prvků pole?
 - Jaká bude maximální frekvence obvodů, pokud zpoždění všech použitých komponent je shodně 2 ns, zpoždění na každém vodiči shodně 1 ns a setup time registrů je také 1 ns?
 - Zamyslete se nad různými způsoby urychlení při zpracování v hardware.

- Nejdříve si navrhujeme komponenty vybírající minimum a maximum ze dvou prvků

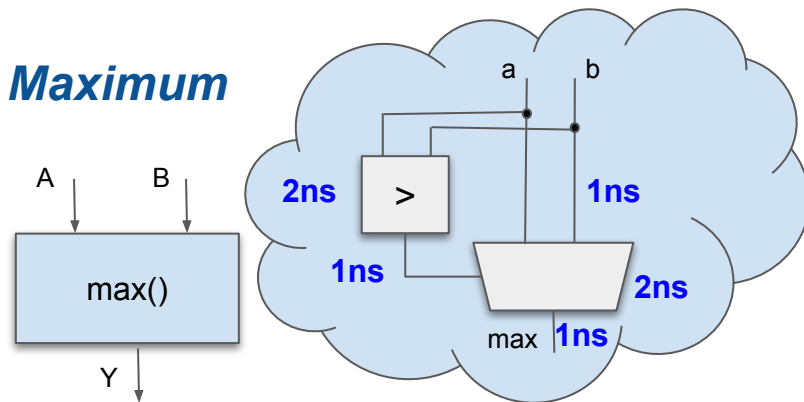
Minimum



```
int min(int A, int B){  
    if (A<B)  
        return A;  
    else  
        return B;  
}
```

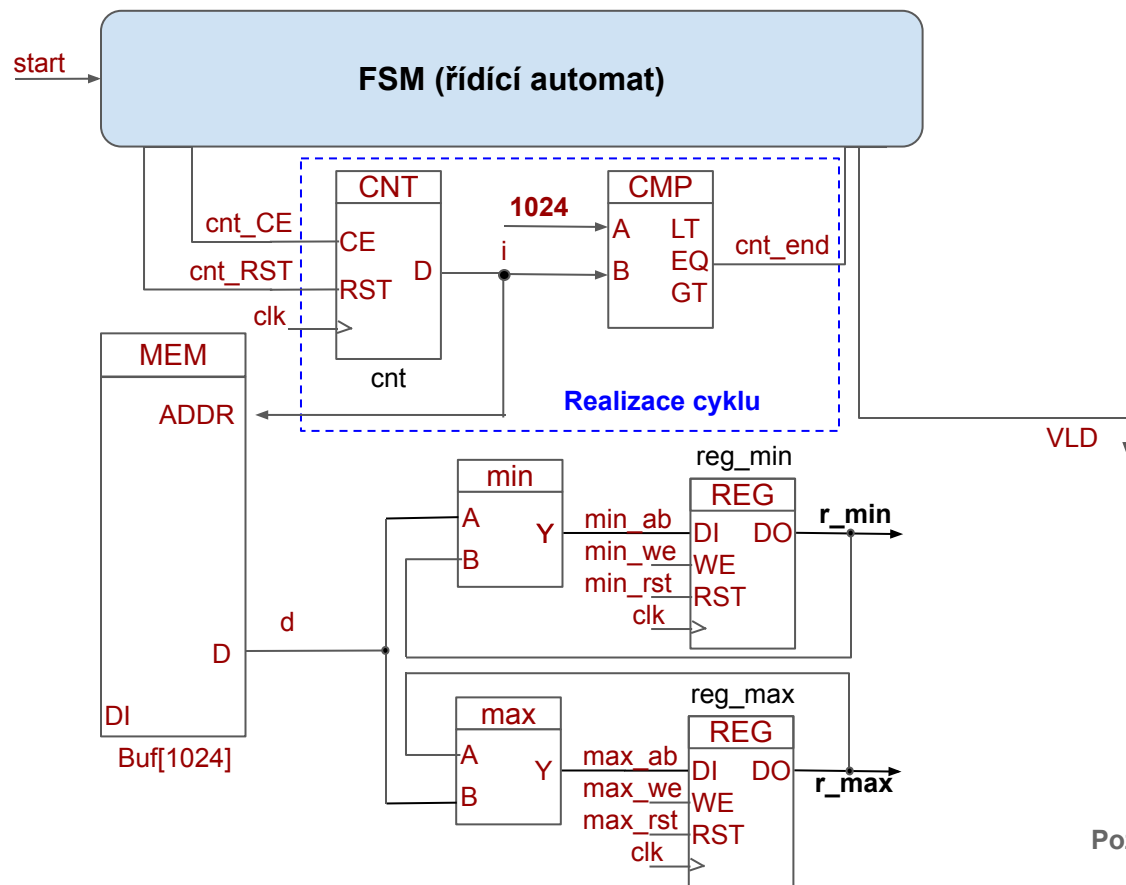
$$T = 1+2+1+2+1 = 7 \text{ ns}$$

Maximum

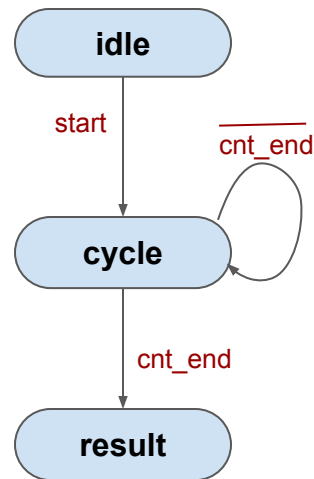


```
int max(int A, int B){  
    if (A>B)  
        return A;  
    else  
        return B;  
}
```

$$T = 1+2+1+2+1 = 7 \text{ ns}$$



```
for (i=0; i<1024; i++) {
    d = Buf[i];
    r_min = min(r_min, d);
    r_max = max(r_max, d);
}
```



idle:

```
cnt_RST=1;
max_rst=1;
min_rst=1;
```

cycle:

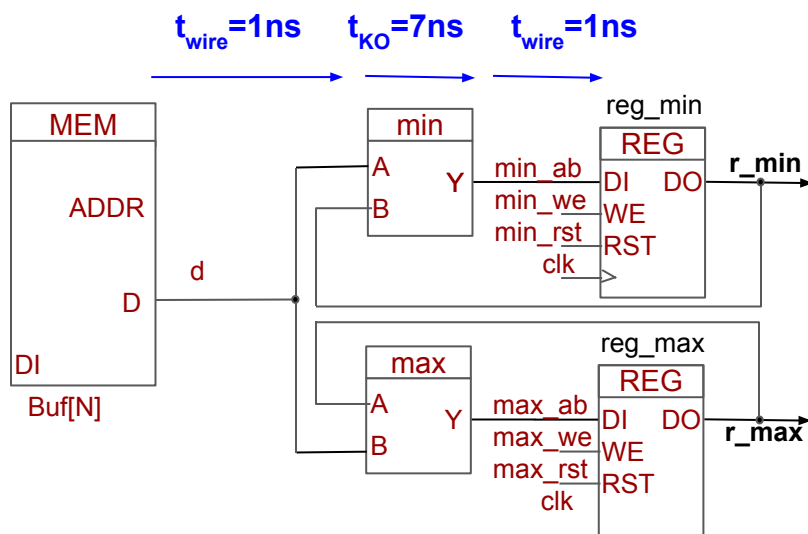
```
cnt_CE=1;
max_we=1;
min_we=1;
```

result:

```
VLD=1;
```

Pozn: Řízení signálů na základě aktuálního stavu. Pokud není některý výstup ve stavu nastaven, má nulovou hodnotu.

- Kolik taktů a času v ns bude trvat zpracování 1024 vzorků dat obvodem při zpoždění vodičů $t_{wire} = 1\text{ ns}$, setup time $t_{setup} = 1\text{ ns}$ a zpoždění každého kombinačního obvodu $t_{KO} = 2\text{ ns}$



$$T = 1 + 7 + 1 + 1 = 10\text{ ns}$$

$$f = 1/10\text{ ns} = 100\text{ MHz}$$

$$t_{1024} = 1024 * 10 = 10240\text{ ns} = 10,240\text{ ms}$$

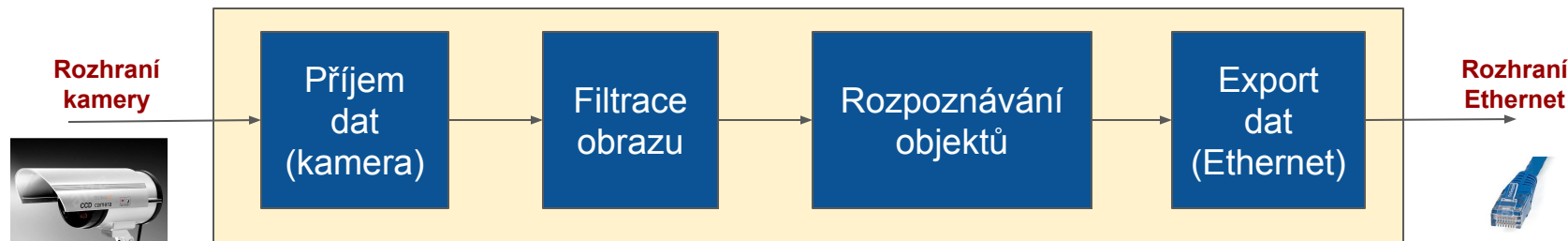
Urychlení je možné dosáhnout

- Méně kroků výpočtu **rozbalením smyčky** a paralelním výpočtem minima a maxima.
- Vyšší frekvence **zřetězením paralelních operací** u rozbalené smyčky

- Příklad návrhu obvodu
- ***Metodika návrhu***
- Jazyk VHDL
- Analýza popisu obvodů

- Začíná se z celkového popisu problému, který se postupně zjemňuje až do nejvyšší míry detailu
 - Specifikace: funkce, rozhraní, cena, výkonnost, atd.
 - Architektura: systémový pohled, definice velkých bloků, atd.
 - Logický návrh: logické prvky a moduly, registry, atd.
 - Návrh obvodu: úroveň tranzistorů, rozvody napájení, plocha, atd.
 - Návrh na fyzické úrovni: fyzický layout, metodologie standardních buňek
- ***Dekompozice*** - postupný rozklad složitého problému na jednodušší úlohy / části.

Zpracování dat z kamery



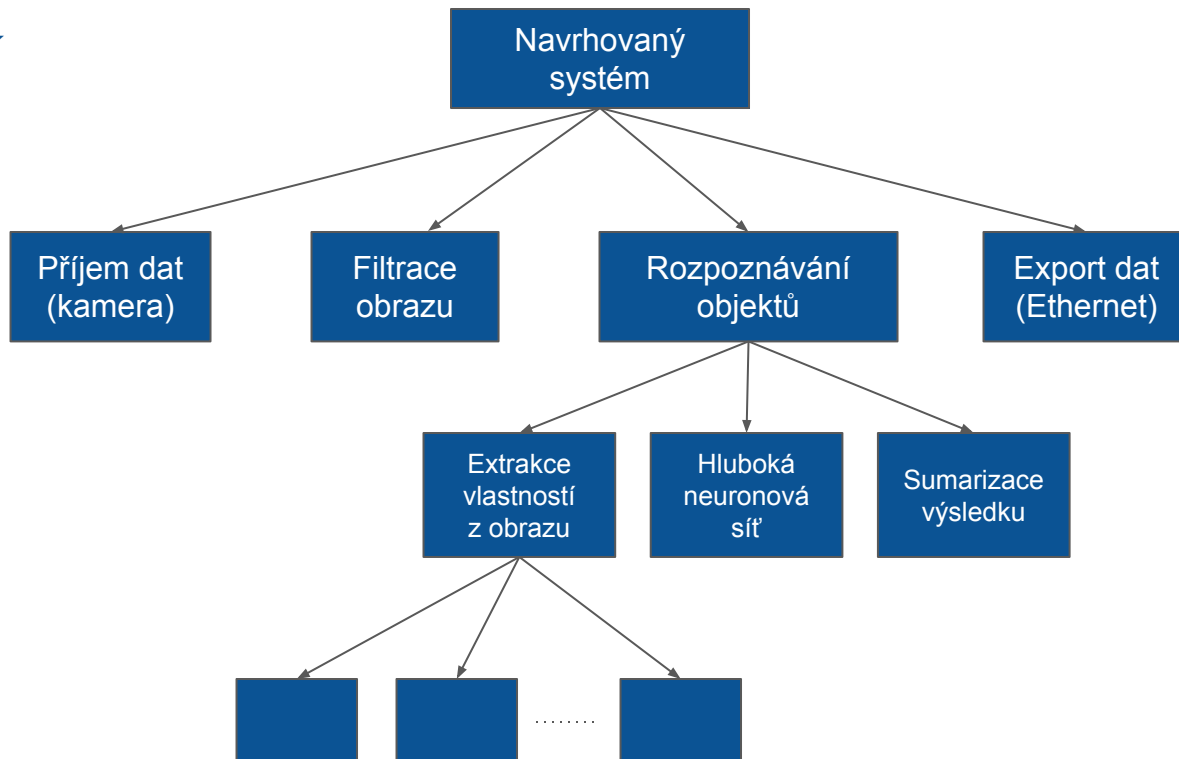
- Každou část můžeme dále dekomponovat na menší bloky
- Postupné zjemnění granularity na snadno řešitelné úlohy

Postupné zjemňování granularity

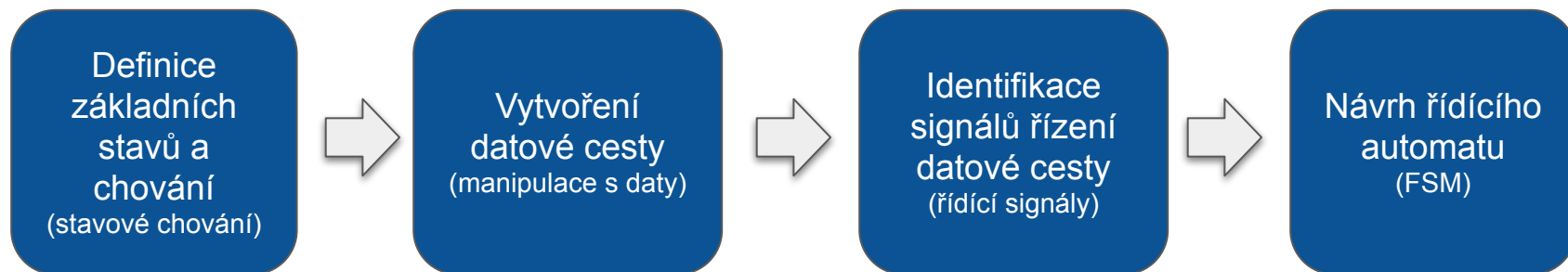
Základní architektura
(bloková struktura)

Dekompozice

**Jednoduše realizovatelné
komponenty**



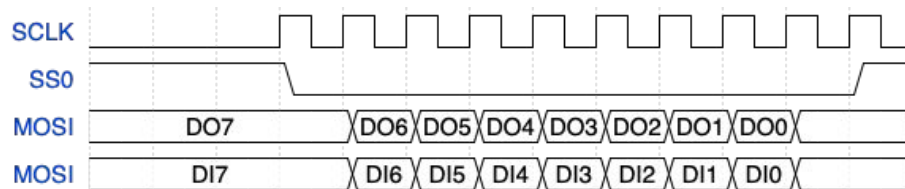
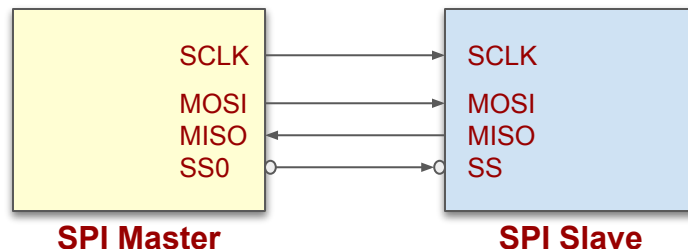
- Návrh je vhodné rozdělit do několika kroků
 - **Popis obvodu na úrovni základního stavového automatu** - automat popisuje základní stavy obvodu, ale neřeší konkrétní nastavení řídicích, vstupních nebo výstupních signálů.
 - **Vytvoření datové cesty (datapath)** - datová cesta provádí manipulaci s daty, a to na základě řízení z obecného automatu.
 - **Identifikace signálů pro řízení datové cesty** - definice signálů, které jsou potřeba pro napojení datové cesty na řídicí blok.
 - **Návrh řídicího automatu** - převedení obecného automatu na Mealyho nebo Moorův automat, který řídí datovou cestu.

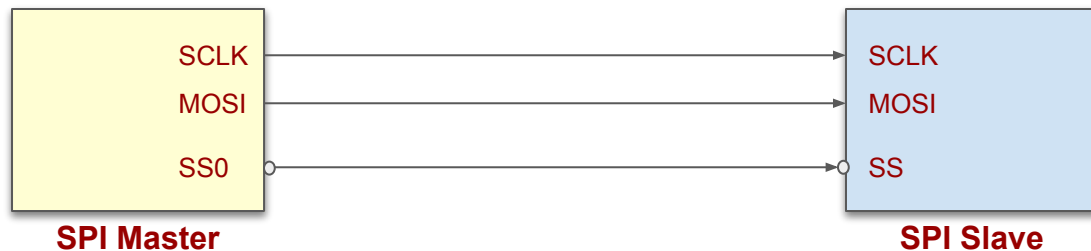


- **Zadání:** Navrhněte obvod pro vysílání dat po sériovém rozhraní SPI. Pro jednoduchost uvažujte pouze jednosměrný přenos s vysíláním 8 bitů dat se zabezpečením pomocí parity (9. bit).

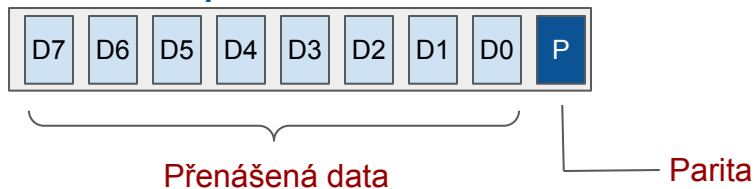
- **Signály SPI rozhraní**

- **SCLK** - hodinový signál
- **MOSI** (Master Out Slave In) sériový datový přenos od zařízení Master k zařízení Slave.
- **MISO** (Master In Slave Out) sériový datový přenos od zařízení Slave k zařízení Master.
- **SS0** (Select Slave) vybírá zařízení, se kterým bude chtít komunikovat. Master může komunikovat s více zařízeními SSx





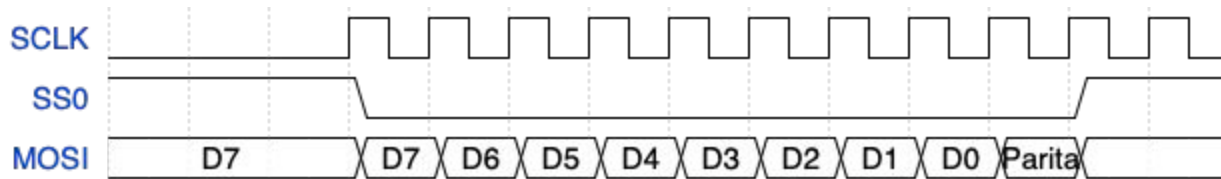
Přenášená zpráva



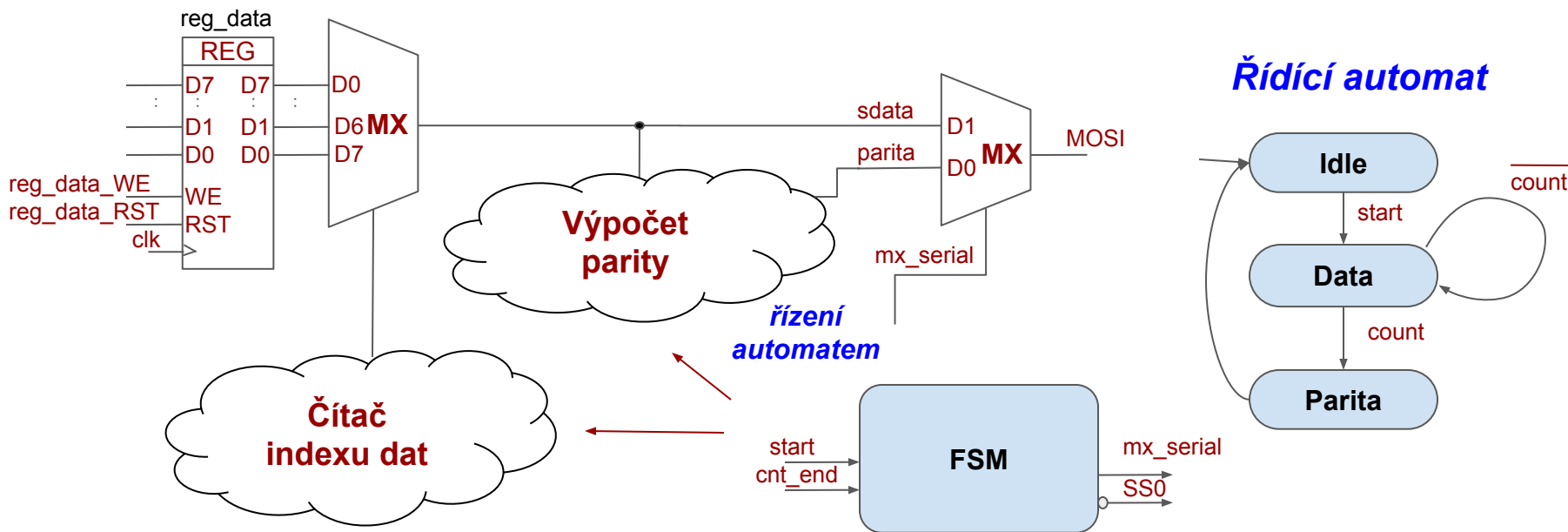
Struktura přenášené zprávy

- Zahájení přenosu** je indikováno signálem $SS0=0$
- Nejprve je přeneseno 8 bitů dat:** $MOSI = Data[i]$, pro $i=7,6,5 \dots 0$
- Pak je přenesen paritní bit:** Pouze zabezpečení komunikace $MOSI = Parita(D7 \text{ až } D0)$
- Ukončení přenosu** je indikováno signálem $SS0=1$

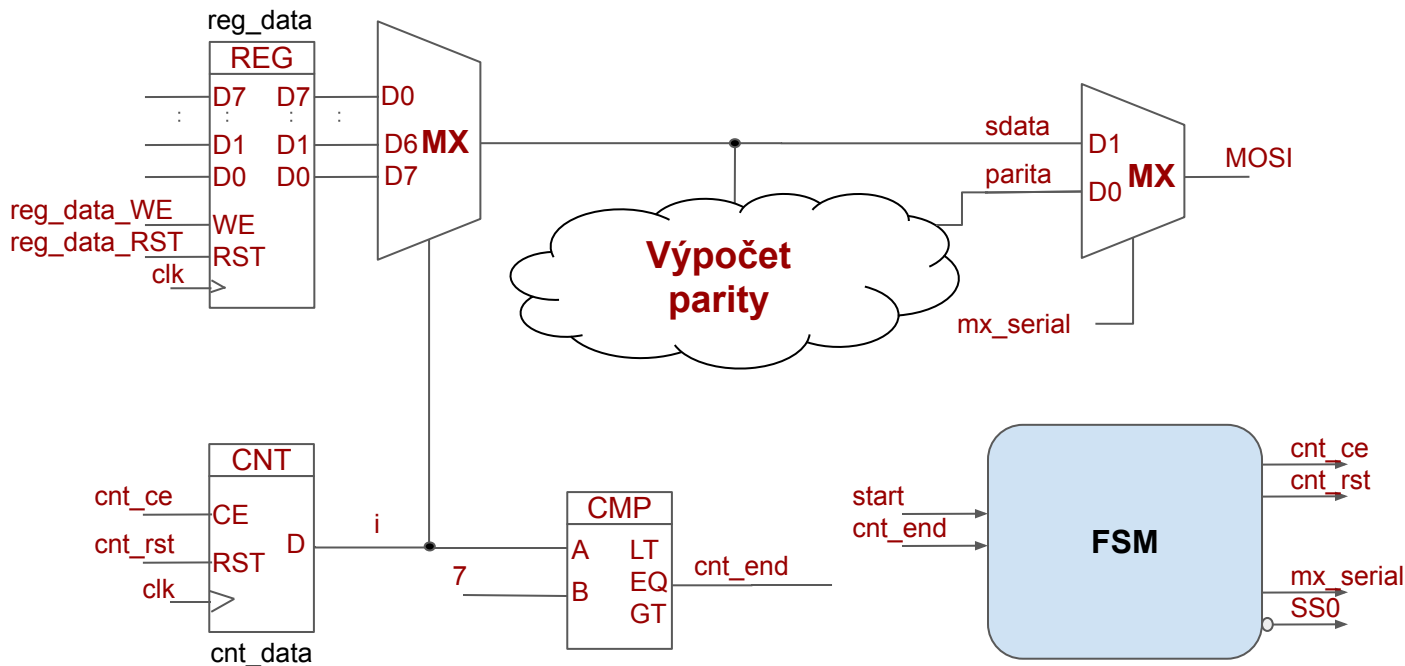
Časový průběh vysílání dat



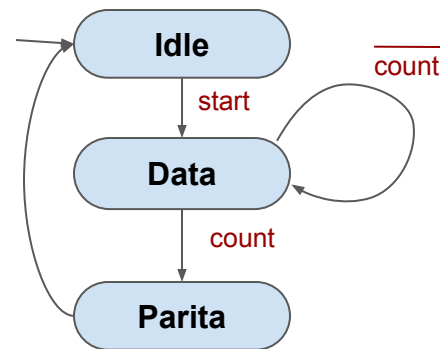
- Schéma zapojení tvoří multiplexory přepínající jednotlivé bity dat a paritu na sériový kanál MOSI sběrnice SPI
- Základní stavy automatu popisují fáze vysílání dat na sběrnici



- Čítače inkrementuje index odesílaných a řídí multiplexor dat
- Komparátor detekuje ukončení odesílání dat (čítač dosáhne hodnoty 7)

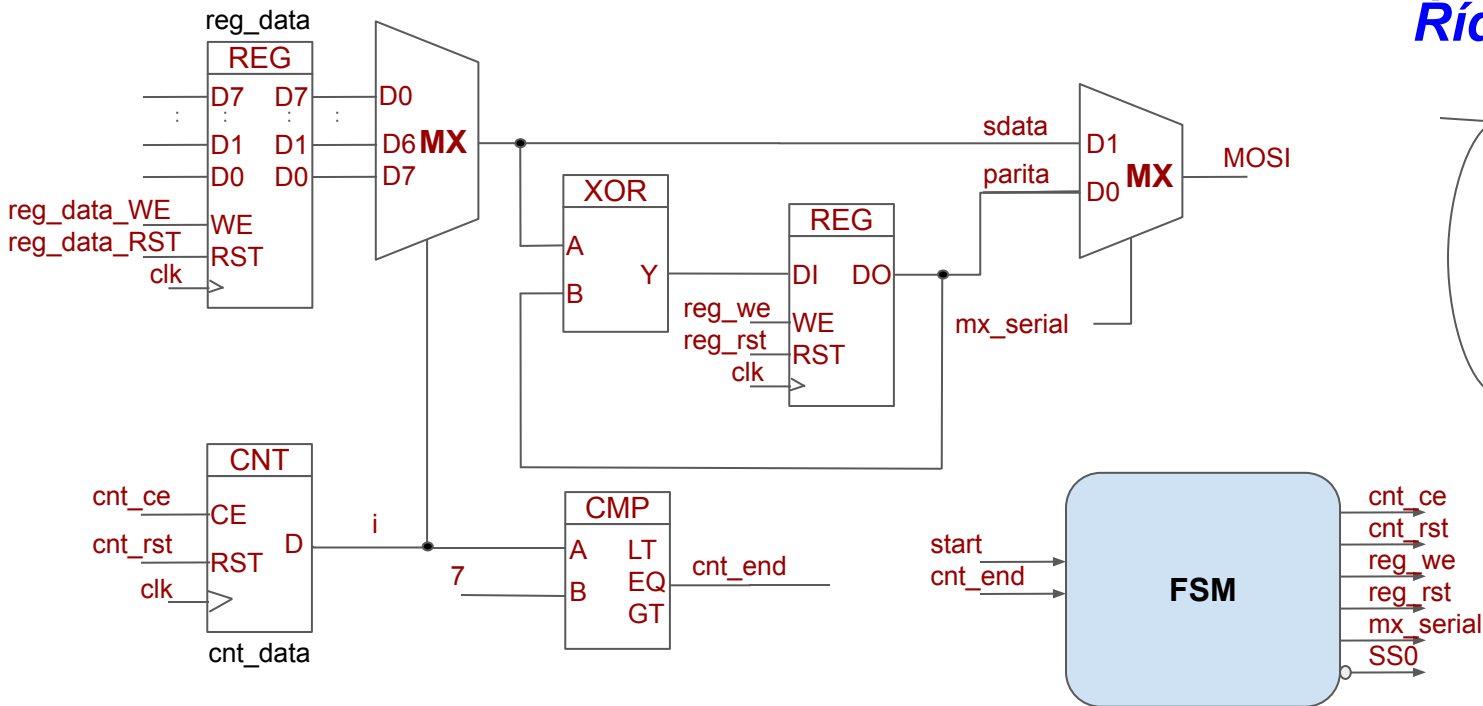
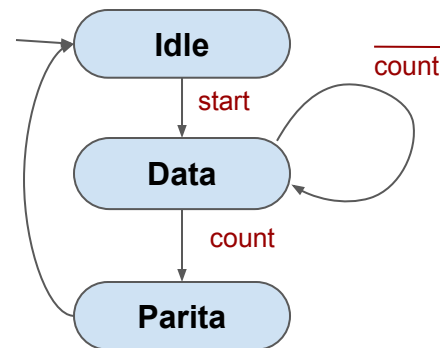


Řídící automat

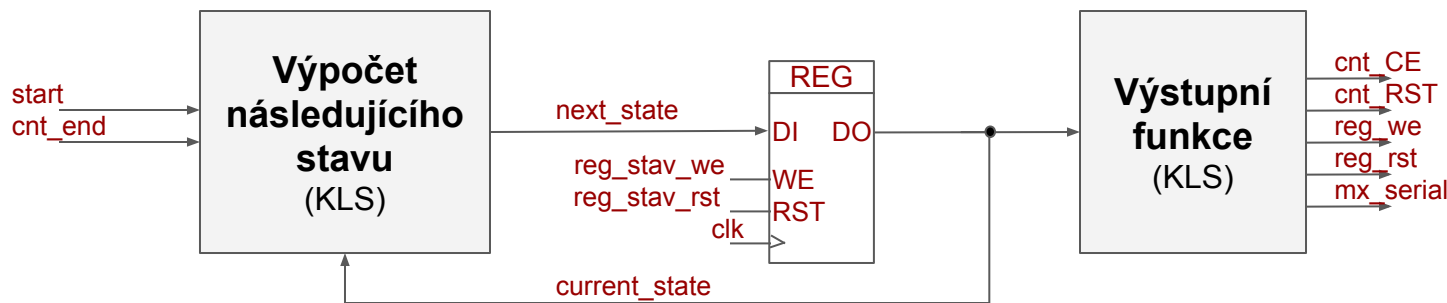


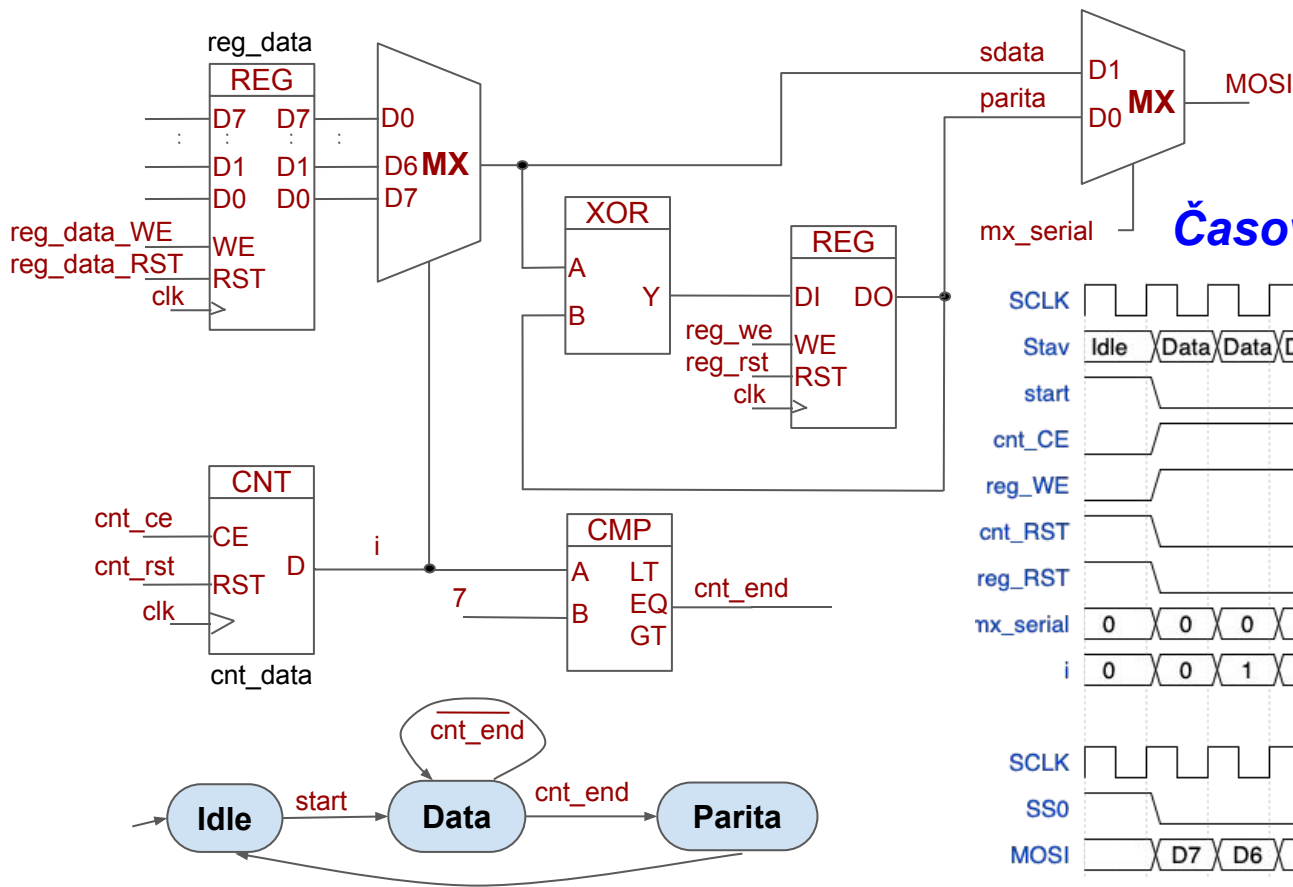
- Parita je počítána postupně, jak jsou data vysílána po sériové lince

Řídící automat

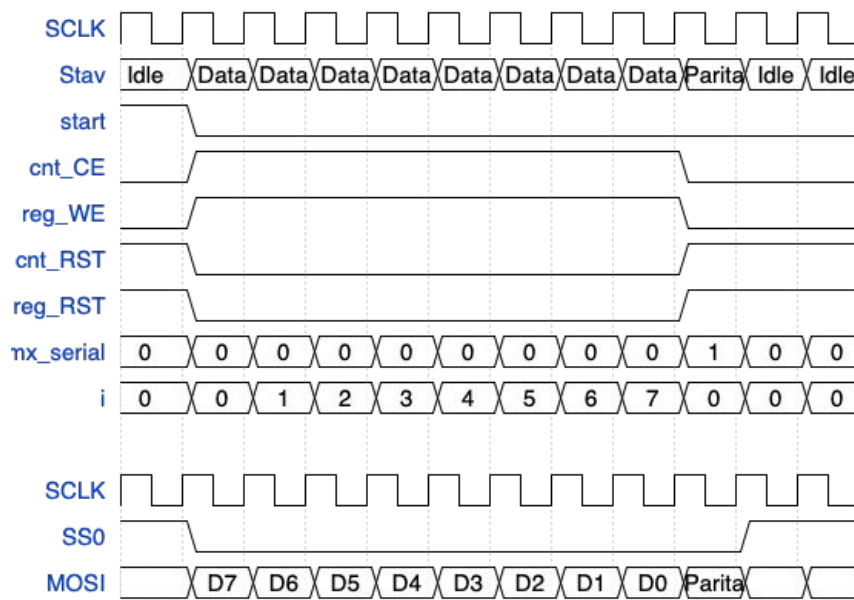


- Automat přechází do dalšího stavu při splnění podmínky a při náběžné hraně hodin clk
 - Aktuální stav je uchováván v registru current_state
 - Kombinační logika vypočítá na základě vstupu a aktuálního stavu následující stav, který se uloží s náběžnou hranou hodin
 - Výstupy jsou generovány kombinační logickou sítí pouze na základě aktuálního stavu

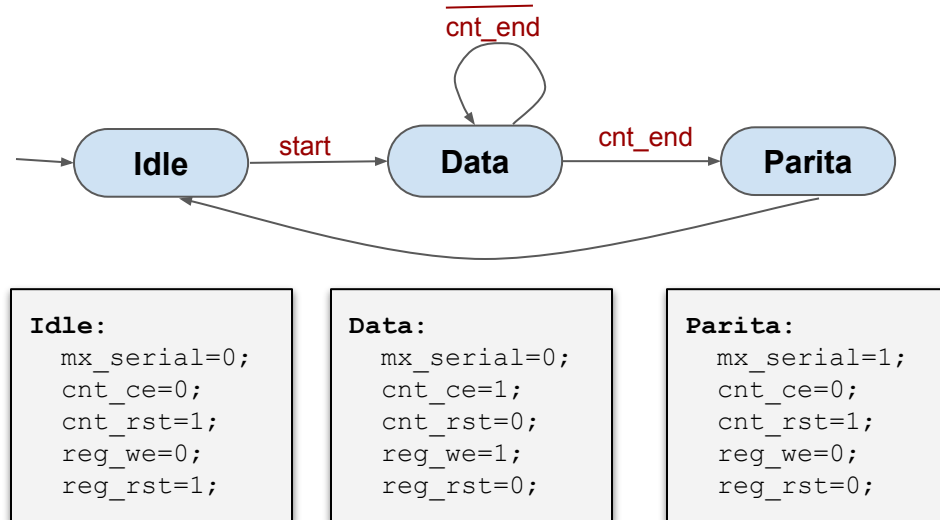




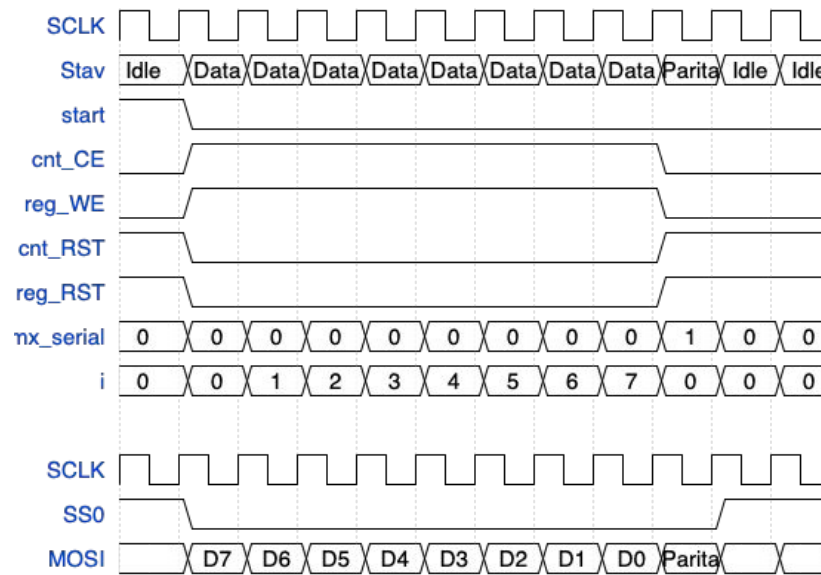
Časový diagram vysílání dat



Řízení signálů automatem (Moorovy výstupy)



Časový diagram vysílání dat



- Příklad návrhu obvodu
- Metodika návrhu obvodů
- ***Jazyk VHDL***
- Analýza popisu obvodů

- Slovně

- Slovní vyjádření toho co má obvod dělat je pro člověka přirozené, avšak vyrobit podle něj obvod není možné

„Navrhněte (číslcový) obvod, který spočítá sumu všech členů zadané posloupnosti“

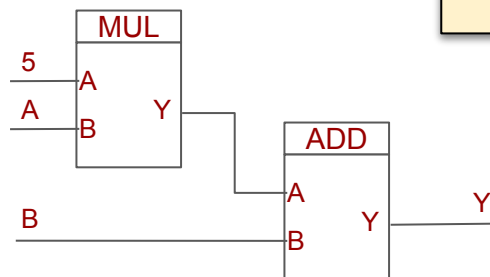
- Matematicky

- V současné době nejsou vhodné nástroje, které by umožnily automatizovaně bez úzké asistence člověka (návrháře) mapování do fyzického hardware

$$S = \sum_{i=1}^N v_i$$

- Graficky pomocí schématu

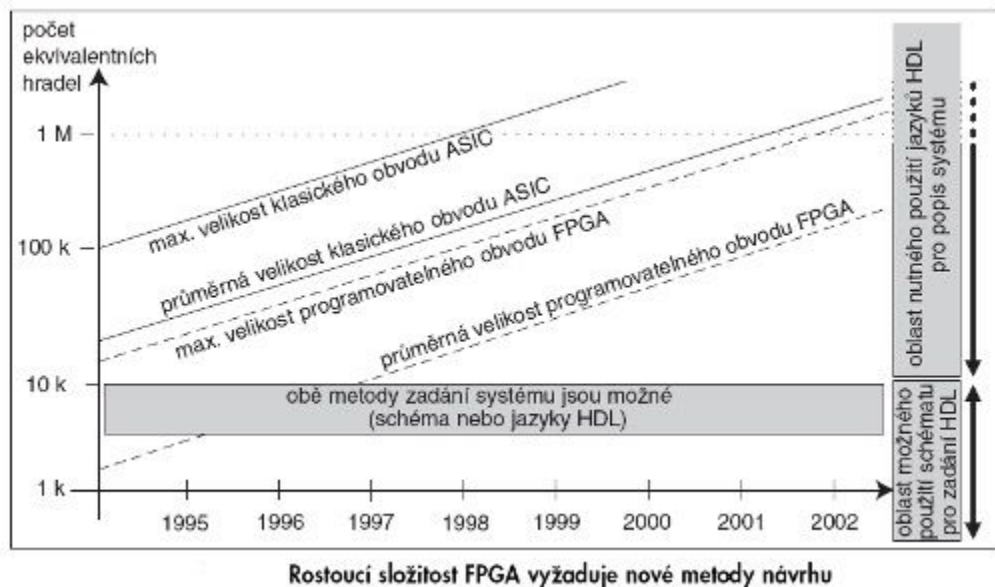
- Funkčních bloky a jejich propojení
- Pro velké obvody pracné a nepřehledné



- Programovacím jazykem

- Lze vytvořit popis chování obvodu v programovacím jazyku
- Vhodné zejména pro velké obvody

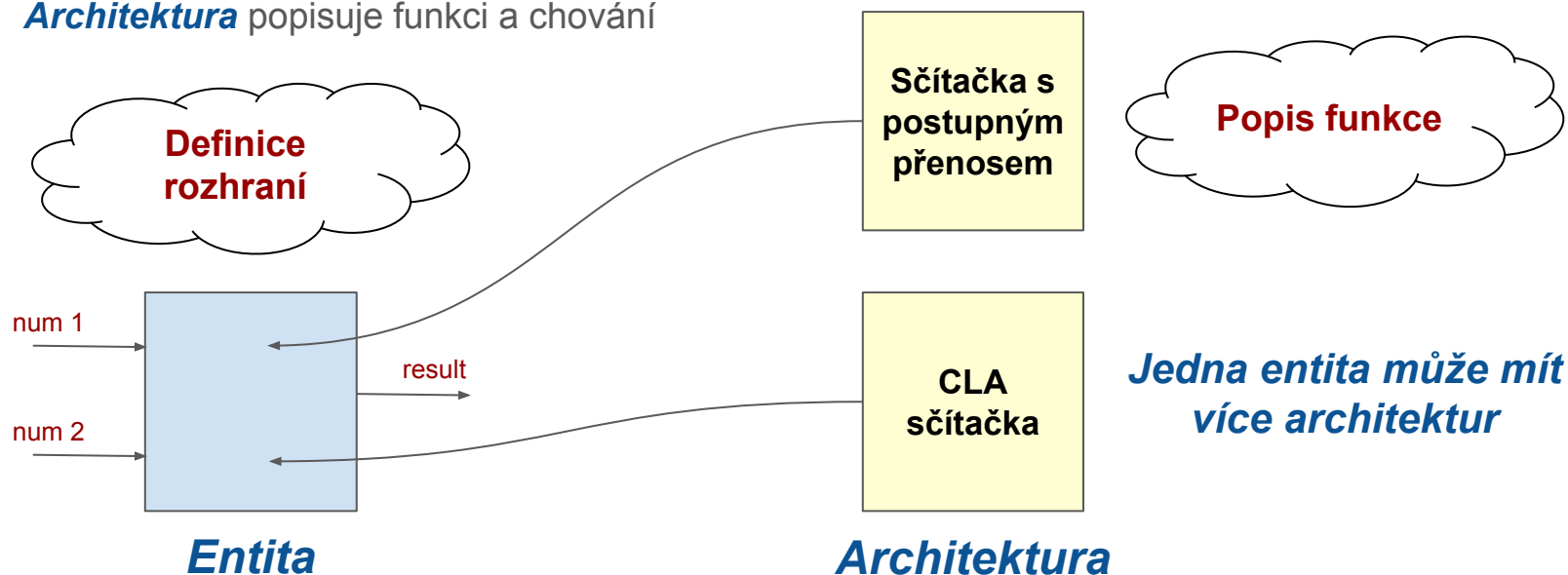
- Grafická reprezentace pomocí logického schématu
- Zvyšující se složitost číslicových zařízení vedla ke vzniku HDL (Hardware Description Language) jazyků



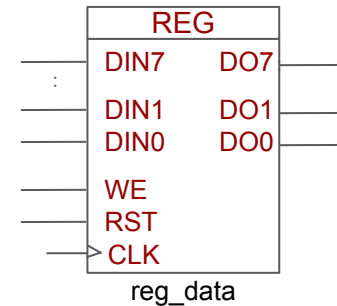
- Na rozdíl od schématu návrhář popisuje funkci obvodu
 - Zařízení je možné **modelovat** a **simulovat**
 - **Proces syntézy umožňuje transformovat HDL popis do prvků cílové technologie** – syntéza je proces analogický kompilaci používané u programovacích jazyků
- V praxi se používají zejména jazyky **VHDL** a **Verilog**
 - Oba jazyky mohou být použity pro mapování popisu do cílové technologie (logická syntéza)
 - VHDL dominuje v Evropě, Verilog v USA

- VHDL popisuje číslicová zařízení a jednotlivé části zařízení pomocí ***komponent***

- Entita*** definuje rozhraní komponenty
- Architektura*** popisuje funkci a chování



- Popisuje rozhraní mezi komponentou a okolím
- Rozhraní komponenty se skládá ze signálů rozhraní (**port**) a generických parametrů (**generic**)
- Signály rozhraní mohou být podle směru šíření dat v módu **IN**, **OUT** nebo **INOUT**



Příklad:

```
entity register is
  generic (DATA_WIDTH : integer :=8
  );
  port (CLK      : in  std_logic;
        RST      : in  std_logic;
        WE       : in  std_logic;
        DIN       : in  std_logic_vector(DATA_WIDTH-1 downto 0);
        DO        : out std_logic_vector(DATA_WIDTH-1 downto 0);
  );
```

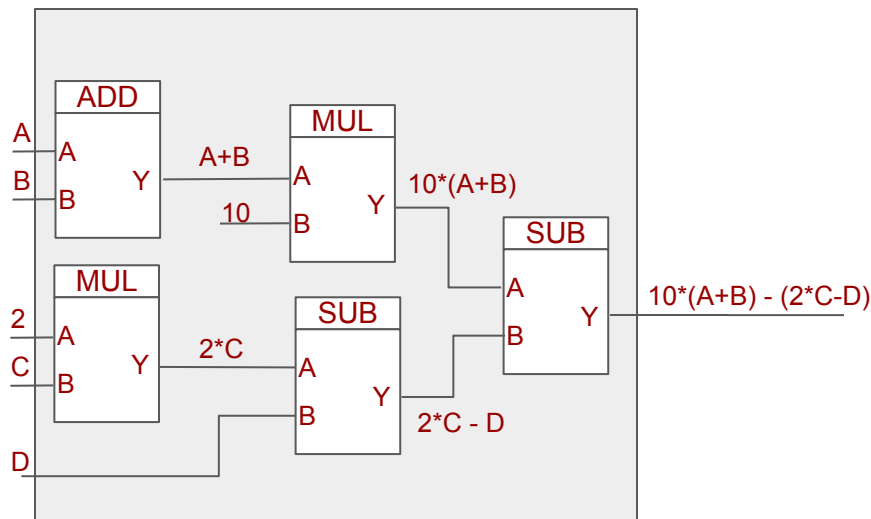
Generické
parametry

Signály rozhraní

- Definuje chování nebo strukturu komponenty
- **Architektura je vždy svázána s entitou**, která definuje rozhraní komponenty (interakci s okolím)
- Každá komponenta může být popsána na úrovni **struktury**, **chování** nebo **dataflow popisu**
- Různé způsoby popisu je možné kombinovat



Architektura



**Každá komponenta
neustále generuje
výstup na základě
vstupů**

**Komponenty pracují
zcela paralelně
(nezávisle na sobě)**

- Popis architektury obvodu musí reflektovat implicitně paralelní zpracování v hardware

Syntax

```
Architecture arch_name of ent_name is
    -- Deklarační část
begin
    -- Sekce paralelních příkazů
end architecture name;
```

Jméno entity

určuje signály rozhraní

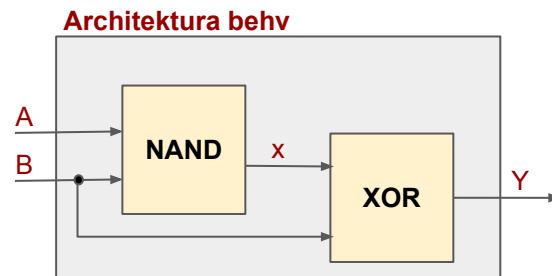
**Deklarační část
architektury**

je vyhrazena pro deklaraci
signálů, konstant nebo typů
použitých uvnitř architektury.

- Součástí sekce paralelních příkazů mohou být instance komponent nebo procesy vzájemně propojené signály
 - **Behaviorální popis** – architektura je složena z jednoho nebo více procesů
 - **Strukturní popis** – architektura obsahuje pouze instance komponent
- V praxi se behaviorální a strukturní popis často kombinují

- Architektura je složena z jednoho nebo více procesů
- Proces je program, který určuje, jak se mají nastavit výstupní signály v závislosti na změnách vstupních signálů
- Z popisu nemusí být zřejmá hardwarová realizace

```
Architecture behv of ent_obvodu is
    signal x : std_logic;
begin
    process (A,B)
    begin
        x <= not (A AND B);
    end process;
    process (x,B)
    begin
        Y <= (x XOR B);
    end process;
end behv;
```



Proces přepočítá výstup Y vždy při změně signálu A nebo B

- Modeluje datové závislosti vstupů a výstupů
- Zkrácený zápis chování pomocí paralelních příkazů v architektuře

Přiřazovací příkaz

```
Y <= NOT (A AND B);
```

Podmíněný přiřazovací příkaz

```
Y <= B when (A='1') else '0';
```

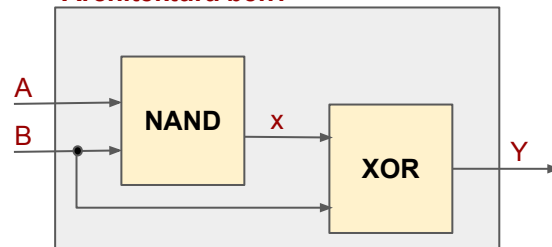
Výběrový přiřazovací příkaz

```
with S select    Y <= A when "0",  
                  B when "1";
```

Příklad

```
architecture dataflow of ent_obvodu is  
    signal x : std_logic;  
begin  
    x <= not (A AND B);  
    Y <= (x XOR B);  
end behv;
```

Architektura behv



Syntax

```
name: process (sensitivity list)  
    declarations  
begin  
    sequential statements  
end process name;
```

Seznam vstupních signálů

Kdykoliv se změní signál na sensitivity listu, je spuštěn proces a vypočítají se nové hodnoty signálů

Deklarační část procesu

je vyhrazena pro deklaraci proměnných, konstant nebo typů použitých uvnitř procesu

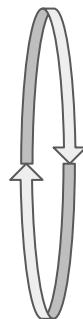
Sekvenční příkazy

Program, který popisuje chování dané komponenty nebo její části. Na základě vstupních signálů a vnitřních proměnných program vypočítá hodnoty výstupních signálů

- Proces popisuje chování celé komponenty nebo pouze její části
- Architektury může obsahovat více procesů komunikujících vzájemně pomocí signálů

- Hardwarové bloky souvisle generují výstupy na základě vstupů
- VHDL modeluje chování HW bloků spouštěním procesů v nekonečné smyčce

```
mu_j_process: process
begin
    příkaz 1;
    příkaz 2;
    příkaz 3;
    příkaz 4;
end process mu_j_process;
```

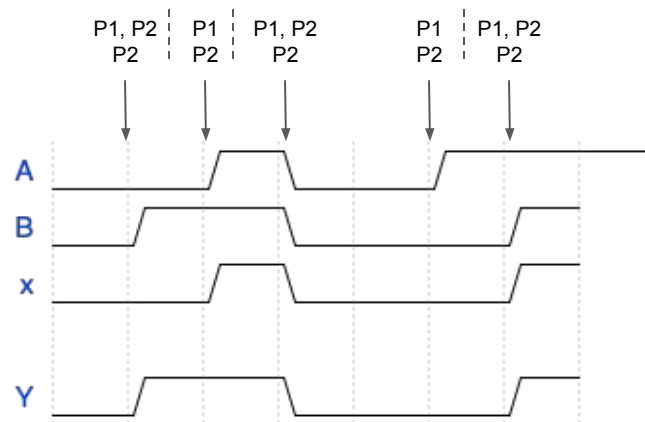


*Proces ve VHDL se
neustále opakuje, nemůže
být nikdy ukončen, ale
může být
pozastaven / uspán*

- Jak analyzovat chování VHDL implementace v simulátoru?
 - Aby nedošlo k zacyklení v nekonečné smyčce procesu, je nutné pro simulaci obvodu definovat **sensitivity list** nebo používat **příkazy WAIT**

- ***Sensitivity list procesu definuje seznam vstupních signálů***, které ovlivňují některý z výstupů procesu
- ***Pokud má process sensitivity list, vykoná se pouze při změně některého ze signálu na sensitivity listu*** tak, aby se vždy při změně vstupu generovaly nové výstupy
- Simulace je pak rychlá, reaguje pouze na změny vstupních signálů

```
Architecture behv of ent_obvodu is
  signal x : std_logic;
begin
  P1: process (A,B)
  begin
    x <= not (A AND B);
  end process;
  P2: process (x,B)
  begin
    Y <= (x OR B);
  end process;
end behv;
```



- Podmíněné vykonání příkazů (*if ... then ...*)

```
IF <condition> THEN <statements> END IF;
```

- Podmíněné vykonání příkazů s alternativou (*if ... then ... else ...* nebo *if ... then ... elsif ...*)

```
IF      <condition> THEN  <statements>  
[ELSIF  <condition> THEN  <statements>]  
[ELSE   <condition> THEN  <statements>]  
END IF;
```

- Výběr více příkazů (*case ...*)

```
CASE <expression> IS WHEN <value1> => <statements>  
                        [WHEN <value2> => <statements>]  
                        [WHEN <value3> => <statements>]  
END CASE;
```

- Cykly umožňující opakované vykonání sekvence příkazů

- *while* ... *do* ...

```
WHILE <condition> LOOP <statements> END LOOP;
```

- *for* ... *loop* ...

```
FOR <range> LOOP <statements> END LOOP;
```

- Příkazy pro přerušení běhu smyčky

- *NEXT* – skok do další iterace

```
next when <condition>;
```

- *EXIT* – ukončení celé smyčky

```
exit when <condition>;
```

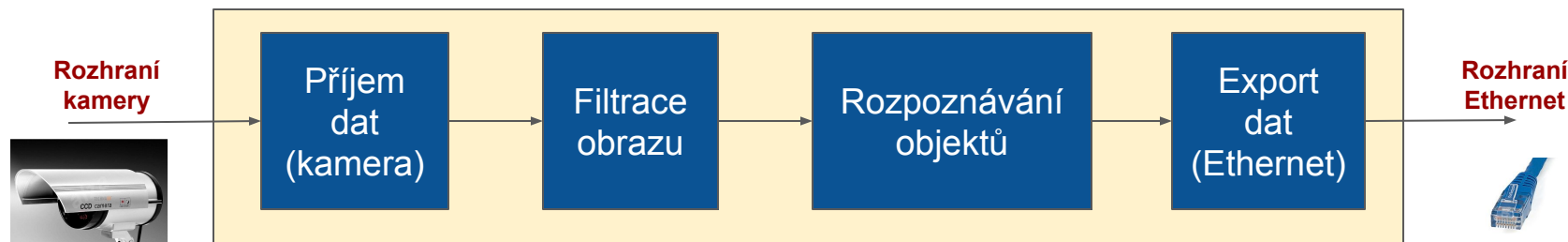
*Součet jedniček v bitovém vektoru **bus_in***

```
process (bus_in)
    variable count : std_logic_vector(3 downto 0);
begin
    count := "0000";
    for i in 0 to 15 loop
        if bus_in(i) = '1' then
            count := count + '1';
        end if;
    end loop;
    N_ONE <= count;
end process;
```

- Při každé změně signálu **bus_in** se vyvolá proces, sečtou se jedničky ve vektoru **bus_in** a výsledek se uloží do **N_ONES**.
- Pro akumulaci počtu jedniček je využita proměnná

- Popisuje strukturu systému, tedy z čeho se daný systém (zařízení nebo komponenta) skládá

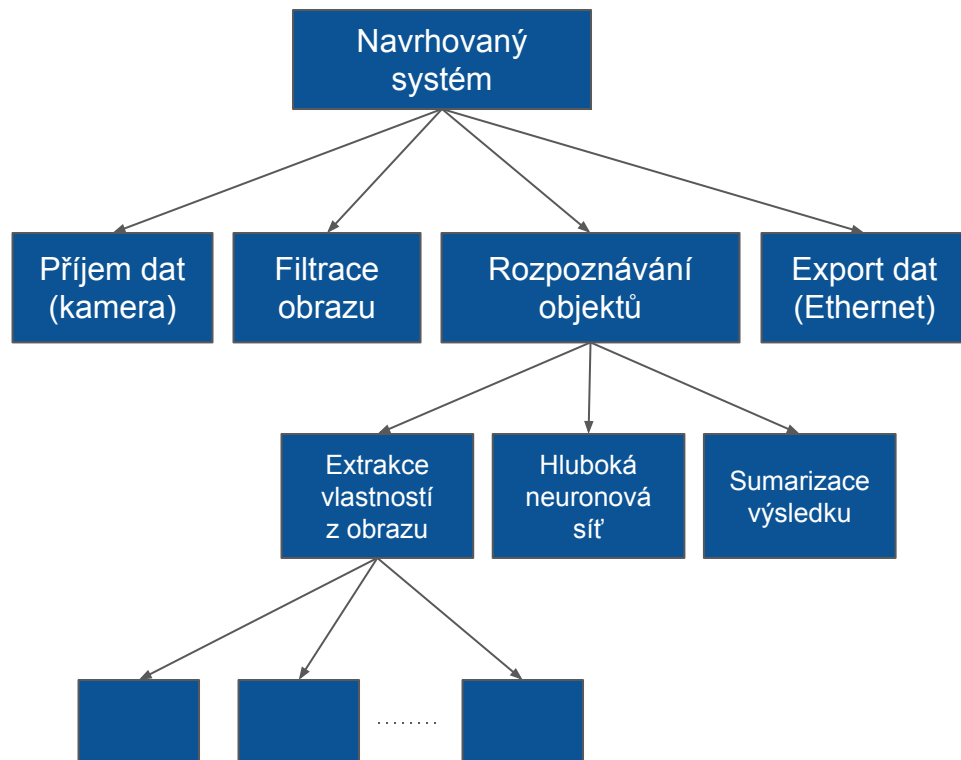
Zpracování dat z kamery



- Strukturní popis odpovídá dekompozici systému na dílčí části (komponenty)

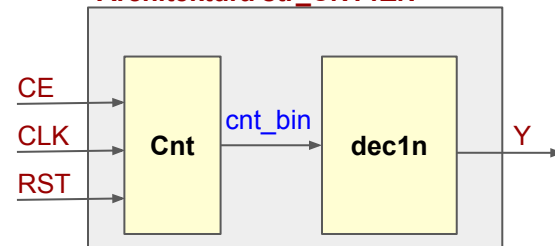
- Strukturní popis může mít více úrovní hierarchie
 - Každá dílčí komponenta může být popsána opět na úrovni struktury nebo na úrovni chování
 - Komponenty na nejnižší úrovni jsou vždy popsány behaviorálně

Komponenty na nejnižší úrovni jsou popsány behaviorálně nebo se jedná o knihovní prvky.



- Je nutno specifikovat knihovnu, ve které se entita nachází
- Přiřazení signálů je možné realizovat pořadím nebo přiřazením jména
- Nepřipojené signály se označují klíčovým slovem open

Architektura str_CNT1ZN

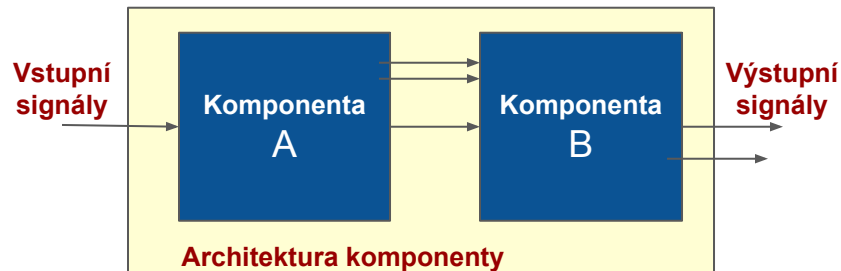


```
entity CNT1ZN is
  port (
    CLK  : in  std_logic;
    RST  : in  std_logic;
    CE   : in  std_logic;
    DO   : out std_logic_vector(7 downto 0)
  );
end CNT1ZN;
architecture str_CNT1ZN of CNT1ZN is
  signal cnt_bin : std_logic_vector(2 downto 0);
begin
  cnt_i: entity work.cnt
    port map ( CLK=>CLK, RST=>RST, CE=>CE,
              DOUT => cnt_bin);
  decln_i: entity work.decln
    port map ( addr=>cnt_bin, y=>DO);
end struct;
```

```
entity cnt is
  port (
    CLK  : in  std_logic;
    RST  : in  std_logic;
    CE   : in  std_logic;
    DOUT : out std_logic_vector(2 downto 0)
  );
end cnt;
```

```
entity decln is
  port (
    addr: in  std_logic_vector(2 downto 0);
    y   : out std_logic_vector(7 downto 0)
  );
end decln;
```

- Signály slouží pro komunikaci mezi komponenty nebo procesy
- Mohou být implementovány pomocí *vodiče* nebo *sběrnice* (více vodičů)
- Signálu je možné přiřadit libovolný datový typ, který definuje charakter přenášených hodnot
 - Pro reprezentace vodiče se používá typ `std_logic`
 - Pro sběrnice se používá typ `std_logic_vector()`. Šířka sběrnice je definována šířkou pole.
- Význam bitů sběrnice je dán rozsahem pole



```
std_logic_vector(7 downto 0);  
std_logic_vector(0 to 7);
```

7 je MSB bit

0 je MSB bit

- Vnitřní signály komponenty jsou deklarovány v deklarční části architektury (signály lze použít pouze v rámci architektury)

```
Architecture name of processor is
  -- Deklarační část
begin
  -- Sekce paralelních příkazů
end architecture name;
```

***Zde je možné
deklarovat vnitřní
signály architektury***

***Signál nelze deklarovat
uvnitř procesu!!!***

- Deklarace signálu

```
signal <jmeno> : <typ> [:= imp_hodnota];
```

- Jméno jednoznačně identifikuje signál, typ definuje charakter přenášených dat
- Signálu je možné nastavit počáteční (implicitní) hodnotu

- Nastavení signálu se provádí až v okamžiku pozastavení / uspání procesu
- Pokud je v procesu spuštěno více příkazů přiřazení hodnoty do signálu, je provedeno pouze poslední přiřazení, a to až v okamžiku dokončení procesu (pozastavení / uspání).
 - **Příkaz přiřazení pouze naplánuje akci přiřazení nové hodnoty do signálu**, obsah signálu ale zůstává do pozastavení / uspání procesu nezměněn
 - Samotná **změna hodnoty signálu se provede až v okamžiku pozastavení / uspání procesu**, kdy zpracování programu uvnitř procesu dojde do konce procesu

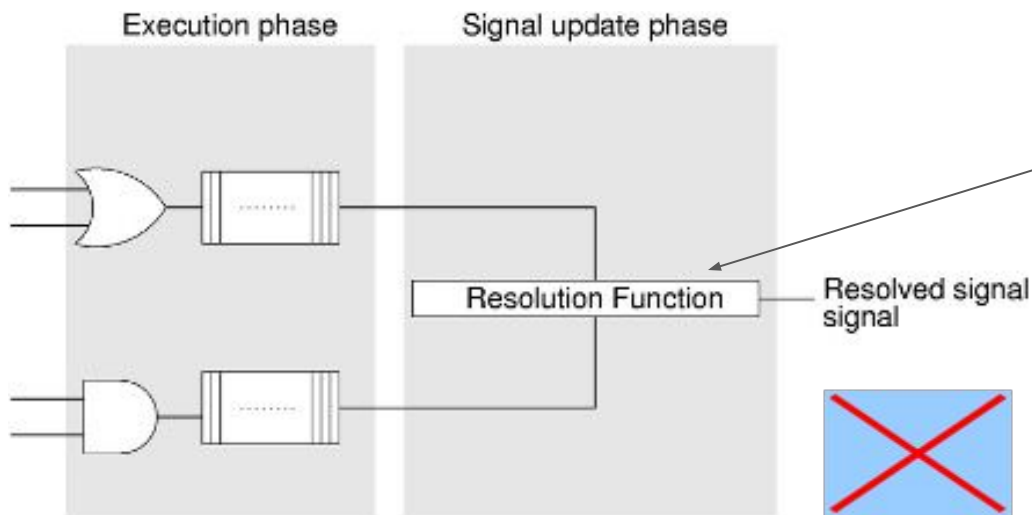
```
example : process (A, B)
begin
  C <= A;
  C <= B+1;
end process example;
```

Naplánováno přiřazení hodnoty A do signálu C

Přepsání plánu přiřazení do signálu C: místo A se přiřazení změní na hodnotu B+1

Pozastavení / uspání procesu a provedení posledního přiřazení signálu C na novou hodnotu (B+1)

- Pokud má signál jediný zdroj, je jednoduché určit hodnotu signálu
- Signály jsou nastavovány z více zdrojů v řadě aplikací – například datová sběrnice počítače
 - Co se stane v případě, že je nastavován signál z více zdrojů současně?
 - Jaké hodnoty bude výsledný signál nabývat?



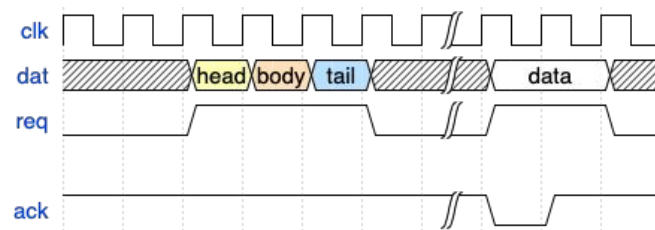
Rezoluční funkce definuje prostřednictvím tabulky hodnotu signálu v případě, že je nastavován signál ze dvou různých zdrojů současně

Dvě hodnoty jsou na reprezentaci výsledku málo!

Jaký bude výsledek rezoluční funkce – hodnota '0' nebo '1' ?

- Každý signál má svůj průběh v čase, nikoliv pouze aktuální hodnotu
- **Historie signálů je ve VHDL přístupná pomocí atributů**
- Syntaxe použití atributu

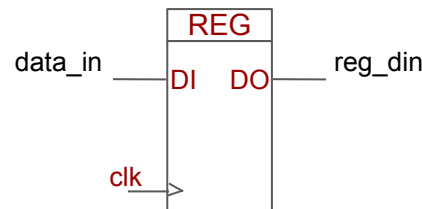
```
<jmeno_signalu>'<jmeno_atributu>
```



- Příklady používaných atributů
 - **event** – boolean – atribut je hodnoty true v případě změny hodnoty signálu
 - **last_value** – hodnota signálu před poslední změnou hodnoty
- Používá se zejména při **detekci náběžné hrany hodin**

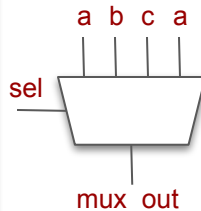
```
process (clk, data_in)
begin
  if clk'event AND clk='1' then
    reg_din <= data_in;
  end if;
end process;
```

**Detekce
náběžné hrany**



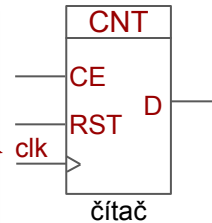
Kombinační obvod (multiplexor)

```
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;
```



Sekvenční obvod (čítač)

```
process (clk, RST, CE)
begin
    if (RST = '1') then
        DOUT <= (others => '0');
    elsif (clk'event and clk = '1') then
        if CE='1' then
            D <= D + '1';
        end if;
    end if;
end process;
```



- **Sekvenční obvod obsahuje synchronizaci na náběžnou hranu hodin ($clk'event$ and $clk=1$)**
- Kombinační obvod vypočítává změnu výstupních signálů na základě vstupu bez synchronizace (není využit hodinový signál)

- Signály slouží pro komunikaci mezi procesy – připojení ***vstupů a výstupů procesů***
- Vlastnosti signálů
 - Signály nemohou být deklarovány uvnitř procesů
 - ***Přiřazení signálů je fyzicky provedeno až v okamžiku pozastavení / uspání procesu.*** Dokud není proces uspán, má signál původní hodnotu
 - ***V procesu je provedeno pouze poslední přiřazení signálů.*** Ostatní přiřazení jsou nejsou realizovány.
- V procesu je možné přechodně uchovat hodnotu pomocí proměnných (variable)

- Proměnné mohou být deklarovány a použity pouze v procesu
- Příklad deklarace a použití proměnné

```

Process (b)
variable a : integer := 3;
begin
    a := a + 2;
    output <= a * b;
end process;
    
```

Deklarace může obsahovat **jméno**, **typ** a **implicitní hodnotu**

Signál output je nastaven již podle aktualizované proměnné **a**
(**output <= 5 * 4**)

Přiřazení do proměnné se provede okamžitě
(**a = 3 + 2 = 5**)

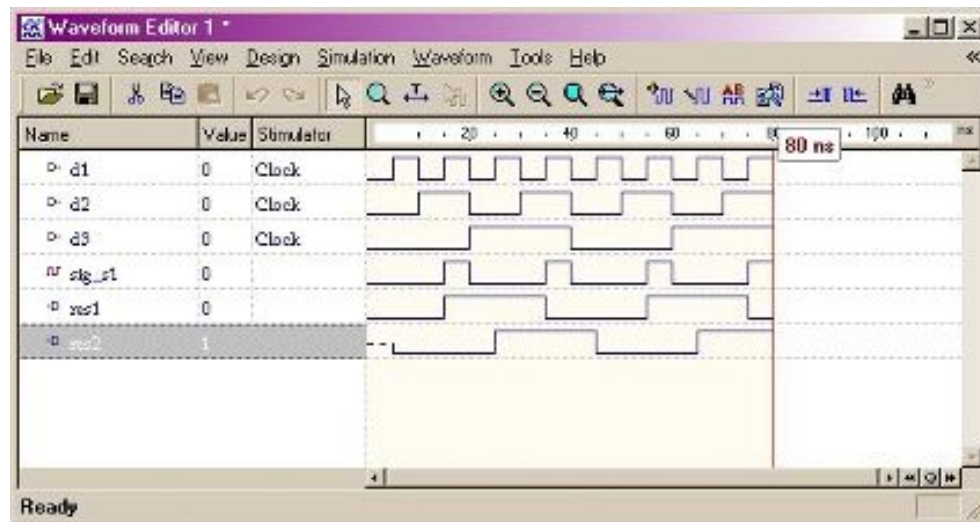
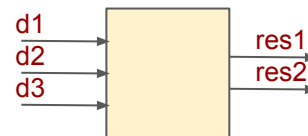
Hodnota proměnné zůstává zachována i po uspání procesu

```
entity sig_var is
port(d1, d2, d3:      in  std_logic;
     res1, res2:      out std_logic);
end sig_var;
```

```
architecture behv of sig_var is
signal sig_s1: std_logic;
begin
```

```
    proc1: process(d1, d2, d3)
        variable var_s1: std_logic;
    begin
        var_s1 := d1 and d2;
        res1 <= var_s1 xor d3;
    end process;
```

```
    proc2: process(d1,d2,d3)
    begin
        sig_s1 <= d1 and d2;
        res2 <= sig_s1 xor d3;
    end process;
end behv;
```



Rozdíly mezi výstupy res1 a res2

- Komentář je uvozen dvojicí znaků `--`

```
-- Toto je komentář
```

- Znak nebo bit se zadává pomocí apostrofů

```
sig_bit <= '1';
```

- Řetězec nebo bitový vektor se zadává pomocí uvozovek

```
sig_bit_vector <= "0001";
```

- Jména signálů a proměnných musí začínat písmenem a dále mohou obsahovat písmena a číslice

- Výčtový datový typ

```
TYPE muj_stav IS (reset, idle, rw, io);  
signal stav: muj_stav;  
...  
stav <= reset;      -- nelze stav <="00";  
...
```

- Pole

```
TYPE data_bus IS ARRAY (0 TO 15) OF BIT;  
variable x: data_bus;  
variable y: bit;  
y := x(12);
```

- Uvažujme deklaraci signálů

```
signal a, b: bit_vector (3 downto 0);  
signal c   : bit_vector (7 downto 0);  
signal clk : bit;
```

- Konkatenace signálů:

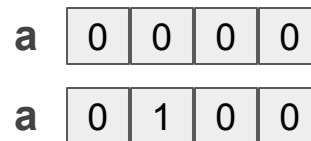
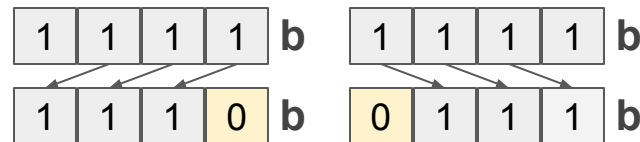
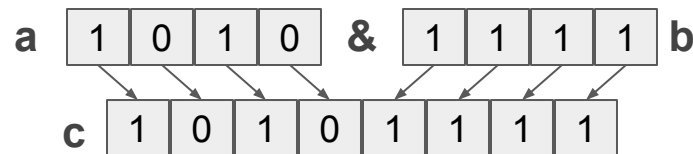
```
c <= a & b;
```

- Bitový posun doleva a doprava

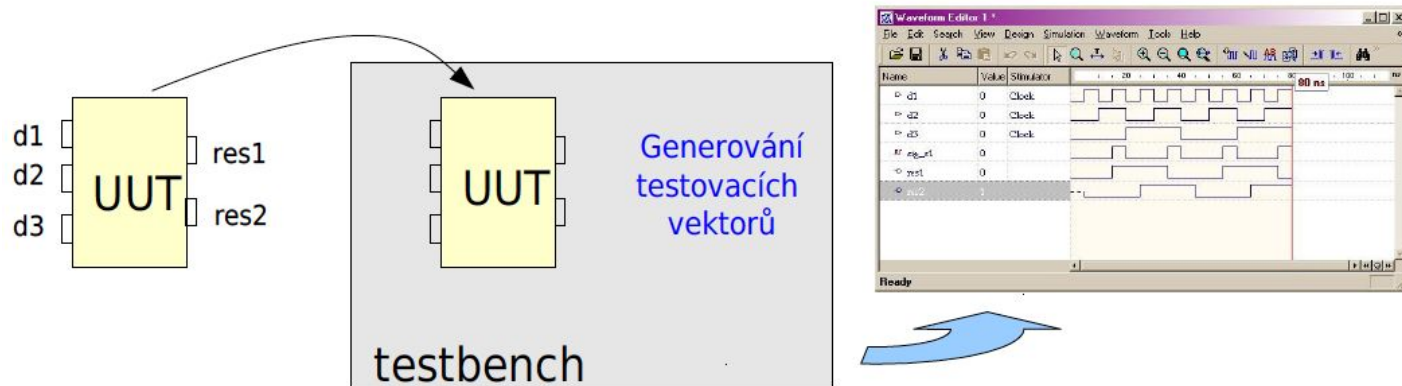
```
b <= b(2 downto 0) & '0'; -- posun doleva  
b <= '0' & b(3 downto 1); -- posun doprava
```

- Agregace

```
a <= (others => '0');           -- vše do nuly  
a <= ('0', '1', others=>'0'); -- MSB = "01"
```



- Testování VHDL komponent v prostředí VHDL



- Testbench obvykle obsahuje
 - Instanci vyvíjené komponenty označenou jako UUT (Unit Under Test)
 - Generátor testovacích vektorů
 - Monitorování a ověřování reakcí UUT

```
entity testbench is  
end entity testbench;
```

Rozhraní entity Testbench

```
architecture testbench_arch of testbench is
```

```
    signal a, b : std_logic;  
    signal ...
```

*Testovací vektory
připojené k UUT*

```
begin
```

```
    UUT : entity work.and_gate  
    port map( ...  
              ...  
            );
```

*Instance UUT
(připojení
testovacích vektorů)*

```
    test : process  
    begin  
        a <= ...;  
        b <= ...;  
        wait for ...;  
        ...  
        wait for ...;  
        ...
```


*Postupné přikládání
testovacích vektorů*

```
        wait;  
    end process test;
```

```
end architecture testbench_arch;
```

- **Proces může být pozastaven příkazem WAIT** v kterékoliv fázi zpracování procesu → je možné **definovat posloupnost kroků při testování komponent**
- Proces obsahující sensitivity list nemůže obsahovat příkaz WAIT
- **Sensitivity list je možné převést na příkaz WAIT on**

```
mu_j_process: process (A, B)
begin
    příkaz 1;
    příkaz 2;
    příkaz 3;
end process mu_j_process;
```



```
mu_j_process: process
begin
    WAIT on A, B;
    příkaz 1;
    příkaz 2;
    příkaz 3;
end process mu_j_process;
```

- **WAIT FOR** čas – pozastaví proces na specifikovaný čas

```
wait for 10 ns;  
wait for clk_per/2;
```

- **WAIT UNTIL** podmínka - pozastaví proces dokud není podmínka pravdivá

```
wait until Clk='1';  
wait until CE and (not RESET);
```

- **WAIT ON** sensitivity list - pozastaví proces dokud není detekována nějaká událost na některém ze signálu uvedených na senzitivity listu

```
wait on Enable;
```

```
entity and_gate is
  port (
    A : in  std_logic;
    B : in  std_logic;
    Y : out std_logic
  );
end entity and_gate;

architecture behavioral of and_gate is

begin
  p_and : process (A, B)
  begin
    Y <= A and B;
  end process p_and;

end architecture behavioral;
```

*Test všech kombinací
na vstupu hradla AND*



```
entity testbench is
end entity testbench;

architecture tb_arch of testbench is
  signal a, b, y : std_logic;
begin
  UUT : entity work.and_gate
  port map( A => a,
            B => b,
            Y => y
  );
  test : process
  begin
    a <= 0; b <= 0;
    wait for 10 ns;
    a <= 0; b <= 1;
    wait for 10 ns;
    a <= 1; b <= 0;
    wait for 10 ns;
    a <= 1; b <= 1;
    wait;
  end process test;
end architecture tb_arch;
```

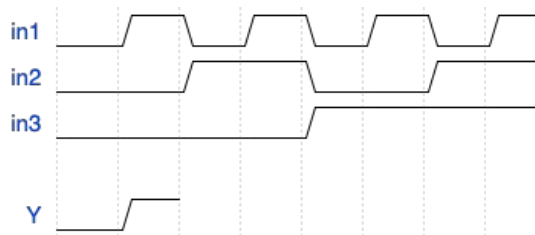

- Automaticky ověří při simulacích platnost zadané podmínky
 - Výsledky jsou zobrazeny ve Waweform dialogu nebo jsou zapsány do logu v rámci simulace
- **Asssertion** se skládá ze tří částí
 - **assert** – specifikuje podmínku, která by měla být splněna
 - **report** – definuje zprávu, která se zobrazí v případě nesplnění podmínky
 - **severity** – určuje jak se simulátor zotaví z chybové situace (general note, warning, system failure)
- Příklad použití

```
assert (left>right)
  report  "Right number greater than left number"
  severity warning;
```

- Příklad návrhu obvodu
- Metodika návrhu
- Jazyk VHDL
- *Analýza popisu obvodů*

- Vytvoření schématu architektury
 - Vytvoření základního schématu tak, že **každý proces tvoří jednu HW jednotku** (blok) a **signály HW jednotky** (procesy) **vzájemně propojují**
 - Vstupní signály HW bloků (procesů) jsou dány sensitivity listem, výstupní signály odpovídají signálům nastavovaným v rámci procesů
- Analýza průchodu signálu obvodem
 - Schéma propojení procesů (HW jednotek) modeluje vzájemné datové závislosti → schéma můžeme využít při analýze šíření změn signálů obvodem
 - Pro každou změnu signálu na vstupu procesu je nutné přepočítat výstupy a nové změny signálu propagovat do dalších procesů podle schématu propojení (datových závislostí)
- Automatizovaně pomocí simulátoru (ModelSim, vsim, ...)
 - Simulace časových průběhů pro kontrolu správné funkce obvodu

```
process(in1, in2, in3)
begin
  case in3 is
    when '0' => y <= in1;
    when '1' => y <= in2;
    when others => null;
  end case;
end process;
```

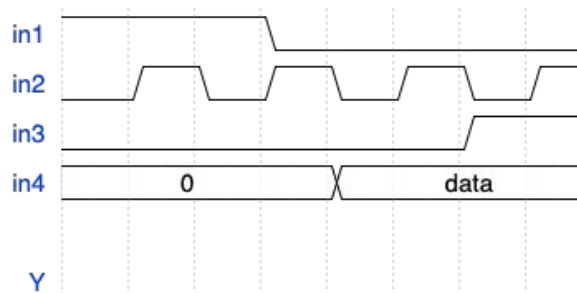


Vstupy

Výstupy

Jaký základní prvek proces
ve VHDL popisuje?

```
process(in1, in2, in3, in4)
begin
  if in1='1' then
    out1='0';
  elsif (in2'event and in2='1') then
    if in3='1' then
      out1=in4;
    end if;
  end if;
end process;
```

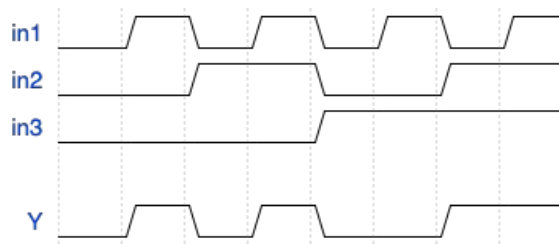


Vstupy

Výstupy

Jaký základní prvek proces
ve VHDL popisuje?

```
process(in1, in2, in3)
begin
  case in3 is
    when '0' => y <= in1;
    when '1' => y <= in2;
    when others => null;
  end case;
end process;
```



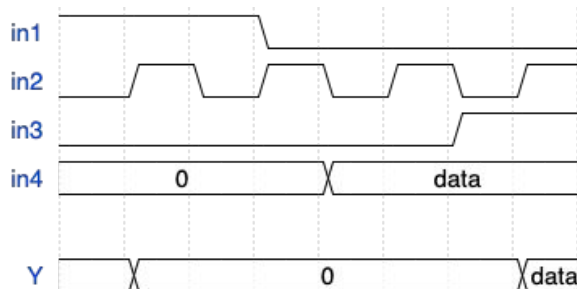
Vstupy ... in1, in2, in3 ...

Výstupy ... y ...

Jaký základní prvek proces
ve VHDL popisuje?

multiplexor

```
process(in1, in2, in3, in4)
begin
  if in1='1' then
    out1='0';
  elsif (in2'event and in2='1') then
    if in3='1' then
      out1=in4;
    end if;
  end if;
end process;
```



Vstupy ...in1, in2, in3, in4 ...

Výstupy ...out1....

Jaký základní prvek proces
ve VHDL popisuje?

registr

```

Architecture behv of ent_obvodu is
    signal sel : std_logic_vector(1 downto 0);
begin

    process (CLK, RST)
    begin
        if RST = '1' then
            sel <= "00";
        elsif CLK'event AND CLK='1' then
            sel <= sel + '1';
        end if;
    end process;

    process (D3,D2,D1,D0,sel)
    begin
        case sel is
            when "00" =>    O <= D0;
            when "01" =>    O <= D1;
            when "10" =>    O <= D2;
            when "11" =>    O <= D3;
            when others =>  O <= D0;
        end case;
    end process;

end behv;
    
```

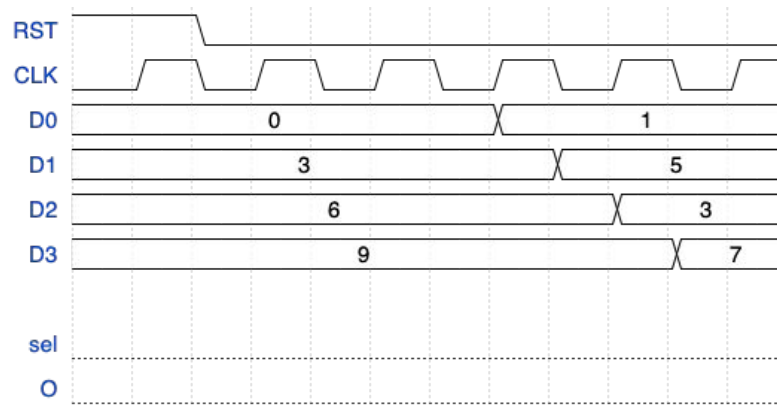
Schéma zapojení

D0
D1
D2
D3

O

RST
CLK

Časový průběh



```

Architecture behv of ent_obvodu is
  signal sel : std_logic_vector(1 downto 0);
begin

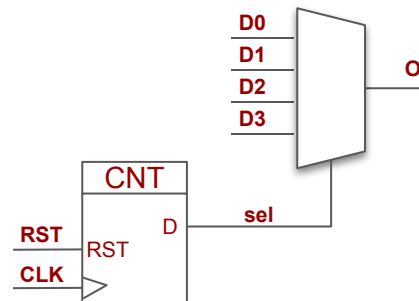
  process (CLK, RST)
  begin
    if RST = '1' then
      sel <= "00";
    elsif CLK'event AND CLK='1' then
      sel <= sel + '1';
    end if;
  end process;

  process (D3,D2,D1,D0,sel)
  begin
    case sel is
      when "00" =>    O <= D0;
      when "01" =>    O <= D1;
      when "10" =>    O <= D2;
      when "11" =>    O <= D3;
      when others =>  O <= D0;
    end case;
  end process;

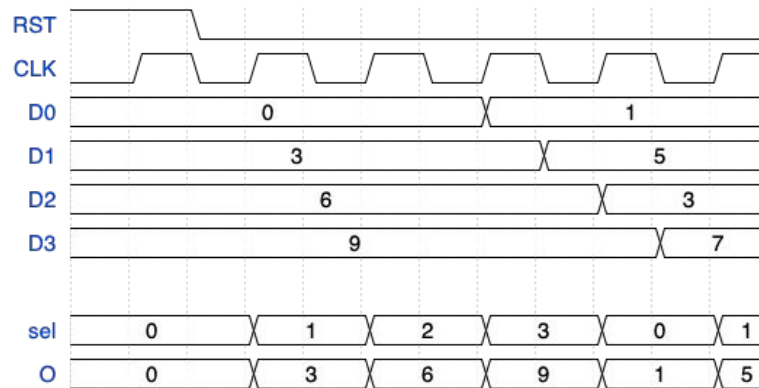
end behv;

```

Schéma zapojení



Časový průběh



Děkuji za pozornost