TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Popis systémů pomocí VHDL

Milan Kolář
Ústav mechatroniky a technické informatiky

INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt ESF CZ.1.07/2.2.00/28.0050
Modernizace didaktických metod
a inovace výuky technických předmětů.

Vývoj VHDL

- **HDL** - **H**ardware **D**escription **L**anguage
- **VHDL** - **V**ery High Speed Integrated Circuits **HDL**
- Vývoj od roku 1983 v rámci projektu **VHSIC**
- **1987 - standard IEEE 1076-1987**
- **1993 - revize IEEE Std 1076-1993**
- 1999 - revize IEEE Std 1076.1-1999
  VHDL-AMS (Analogue & Mixed Signals)
- 2002 – revize IEEE Std 1076-2002
- **2008 – revize IEEE Std 1076-2008**

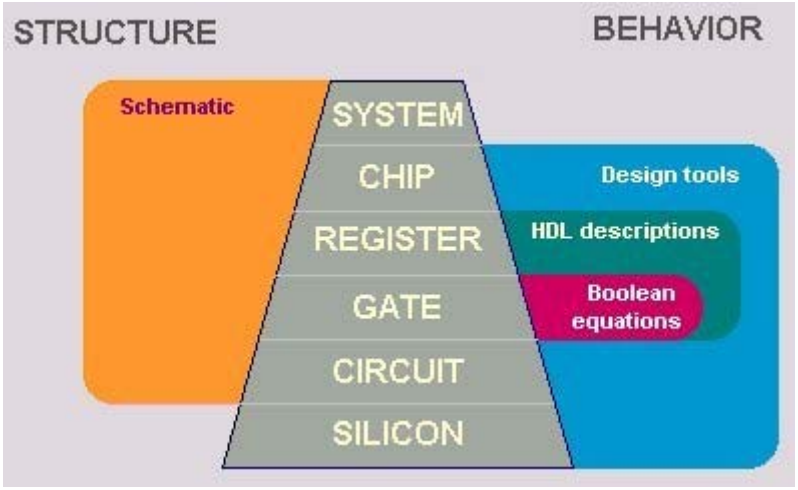
Charakterizace VHDL

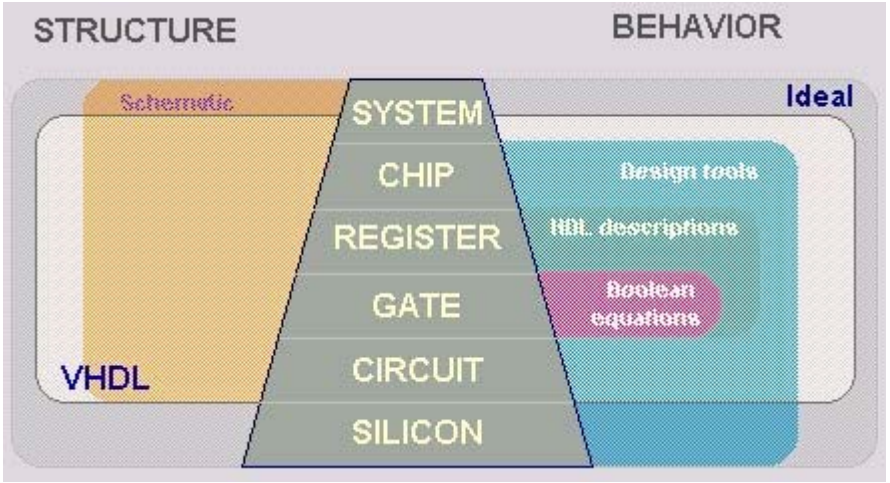
- všeobecně přístupný otevřený standard,
- vhodné pro návrh metodou shora-dolů (top-down),
- nezávislé na budoucí technologii realizace,
- důraz na funkci obvodu (oproštění od detailů),
- umožňuje opakované používání modelů (knihovny),
- využití pro dokumentaci a modelování,
- snadná výměna částí návrhů mezi návrháři (IP core),
- libovolná část návrhu může být osamostatněna,
- model VHDL může být simulován v různých systémech,
- podpora testovatelnosti (Boundary Scan Architecture),
- „upovídaný“ jazyk (opakování bloků, deklarace),
- ne všechny konstrukce jazyka musí být syntetizovatelné.

VHDL v různých úrovních abstrakce

VHDL lze použít v různých úrovních abstrakce:

- úroveň behaviorální (popis chování obvodu)
- úroveň RTL (Register Transfer Level)
- úroveň hradel (logická úroveň)





Formální vlastnosti VHDL

- při zápisu se nerozlišují malá a velká písmena (není „case sensitive“),
- každý příkaz je ukončen středníkem (;),
- v zápisu jazyka lze pro lepší čitelnost používat libovolný počet mezer („space insensitive“),
- využívá klíčových slov,
- žádná pravidla pro jména souborů (doporuč. jméno souboru shodné se jménem nejvyšší entity).

Pravidla jmen identifikátorů

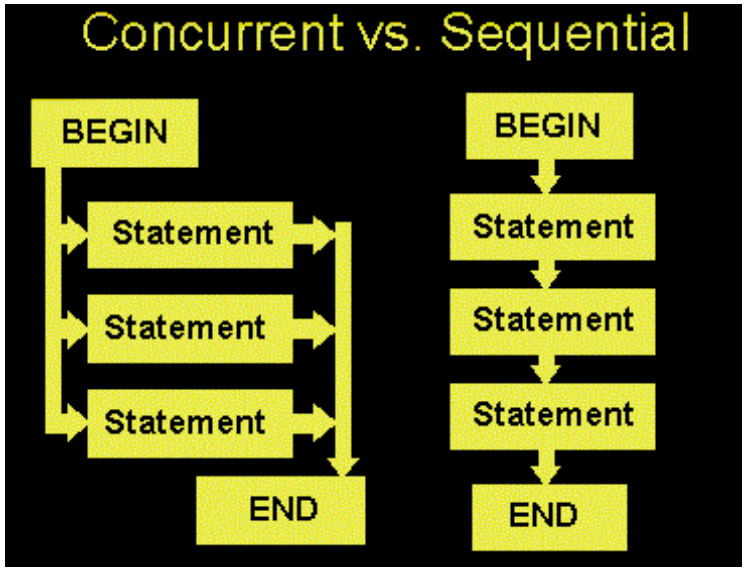
Syntaxe:

```
písmeno{[_]písmeno_nebo_číslice}
```

- nerozlišují se velká a malá písmena,
- musí začínat písmenem,
- mohou obsahovat písmena, číslice a podtržítka,
- jméno nesmí obsahovat mezeru,
- nelze použít dvě podtržítka za sebou,
- podtržítko nesmí být posledním znakem,
- nesmí být totožná s klíčovými slovy,
- musí být unikátní
  - nelze použít signál A a současně sběrnici A(7 downto 0).

Příkazy

- **Declaration statements**
  - definice konstant, typů, objektů, podprogramů;
- **Concurrent statements** – současně probíhající
  - pro popis kombinační logiky;
  - např. block, signal assignment, procedure call, ...
- **Sequential statements**
  - spouští se v napsaném v pořadí;
  - např. příkazy if, case, loop, next, wait, exit, …
  - obdoba SW programu.



Komentář

- začíná dvěma pomlčkami (dash);
- komentář může začínat i za libovolným příkazem;
- komentář končí na konci řádku (nelze jiným způsobem ukončit);
- víceřádkový komentář podporován až od verze VHDL2008 (často ale řešeno již v editorech);
- nedoporučuje se v komentářích používat diakritiku;
- v komentářích se někdy objevují i speciální příkazy návrhového systému (např. pro syntezátor).

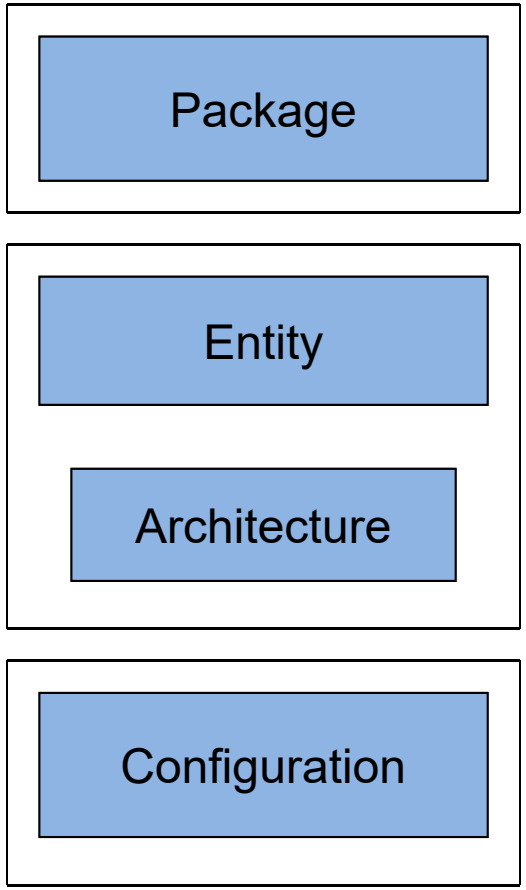
```
-- toto je komentar

c <=  a AND b;    -- toto je take komentar
```

Hlavní komponenty VHDL

dvě povinné komponenty: Entity a Architecture

Příklad - hradlo XOR:

```
ENTITY  hr_xor  IS
      PORT ( a, b : IN BIT;
                  y : OUT BIT );
END hr_xor;
ARCHITECTURE  ar_hr_xor  OF  hr_xor  IS
BEGIN
      y <= a XOR b;
END ar_hr_xor;
```



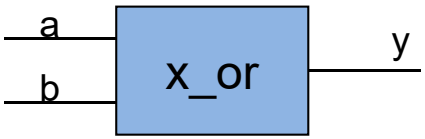
Entita a architektura

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

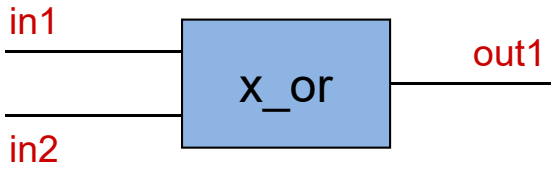
- **Architektura** - určuje chování entit
- tělo architektury má dvě části:
  - deklarační část (např. definice signálů),
  - příkazová část (uzavřeno do ***begin - end***);
- architektura musí být spojena se specifikovanou entitou.

Příklad: XOR (popis chování)

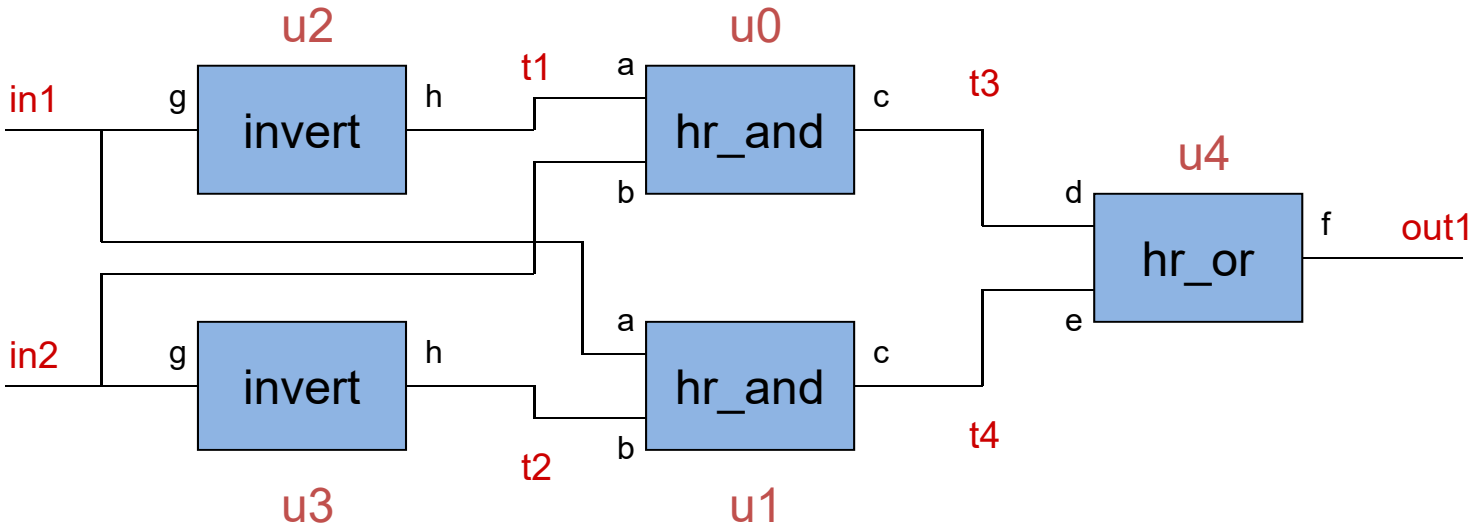
```
ENTITY  x_or  IS
   PORT ( a, b : IN BIT;
                 y : OUT BIT );
END x_or;
ARCHITECTURE  behavior  OF  x_or  IS
BEGIN
   PROCESS (a, b)
     BEGIN
        IF ( a = b ) THEN  y <=  '0';
        ELSE y <=  '1';
        END IF;
     END PROCESS;
END behavior;
```



Příklad: XOR (strukturální popis)



$$(Y = \overline{A}B + A\overline{B})$$



Příklad: XOR (strukturální popis)

```
ENTITY  x_or  IS
    PORT ( in1, in2 : IN BIT;
                out1 : OUT BIT );
END x_or;
```

```
ENTITY  hr_and  IS
    PORT ( a, b : IN BIT;
                c : OUT BIT );
END hr_and;
ARCHITECTURE  behavior  OF
    hr_and  IS
BEGIN
    PROCESS (a, b)
      BEGIN
          c <=  a AND b;
    END PROCESS;
END behavior;
```

```
ENTITY  hr_or  IS
    PORT ( d, e : IN BIT;
                f : OUT BIT );
END hr_or;
ARCHITECTURE  behavior OF hr_or IS
BEGIN
    PROCESS (d, e)
      BEGIN
          f <=  d OR e;
    END PROCESS;
END behavior;
```

Příklad: XOR (strukturální popis)

```
ENTITY  invert  IS
  PORT ( g : IN BIT;
         h : OUT BIT );
END invert;
ARCHITECTURE behavior OF invert IS
BEGIN
  PROCESS (g)
    BEGIN
      h <=  NOT g;
  END PROCESS;
END behavior;
```

```
ARCHITECTURE structural OF x_or IS
SIGNAL t1, t2, t3, t4 : BIT;
COMPONENT hr_and
  PORT (a, b : IN BIT;
        c : OUT BIT );
END COMPONENT;
```

```
COMPONENT  hr_or
  PORT (d, e : IN BIT;
        f : OUT BIT );
END COMPONENT;
COMPONENT  invert
  PORT (g : IN BIT;
        h : OUT BIT );
END COMPONENT;

BEGIN
    u0: hr_and PORT MAP (a=>t1, b=>in2, c=>t3);
    u1: hr_and PORT MAP (a=>in1, b=>t2, c=>t4);
    u2: invert PORT MAP (g=>in1, h=>t1);
    u3: invert PORT MAP (g=>in2, h=>t2);
    u4: hr_or PORT MAP (d=>t3, e=>t4, f=>out1);
END structural;
```

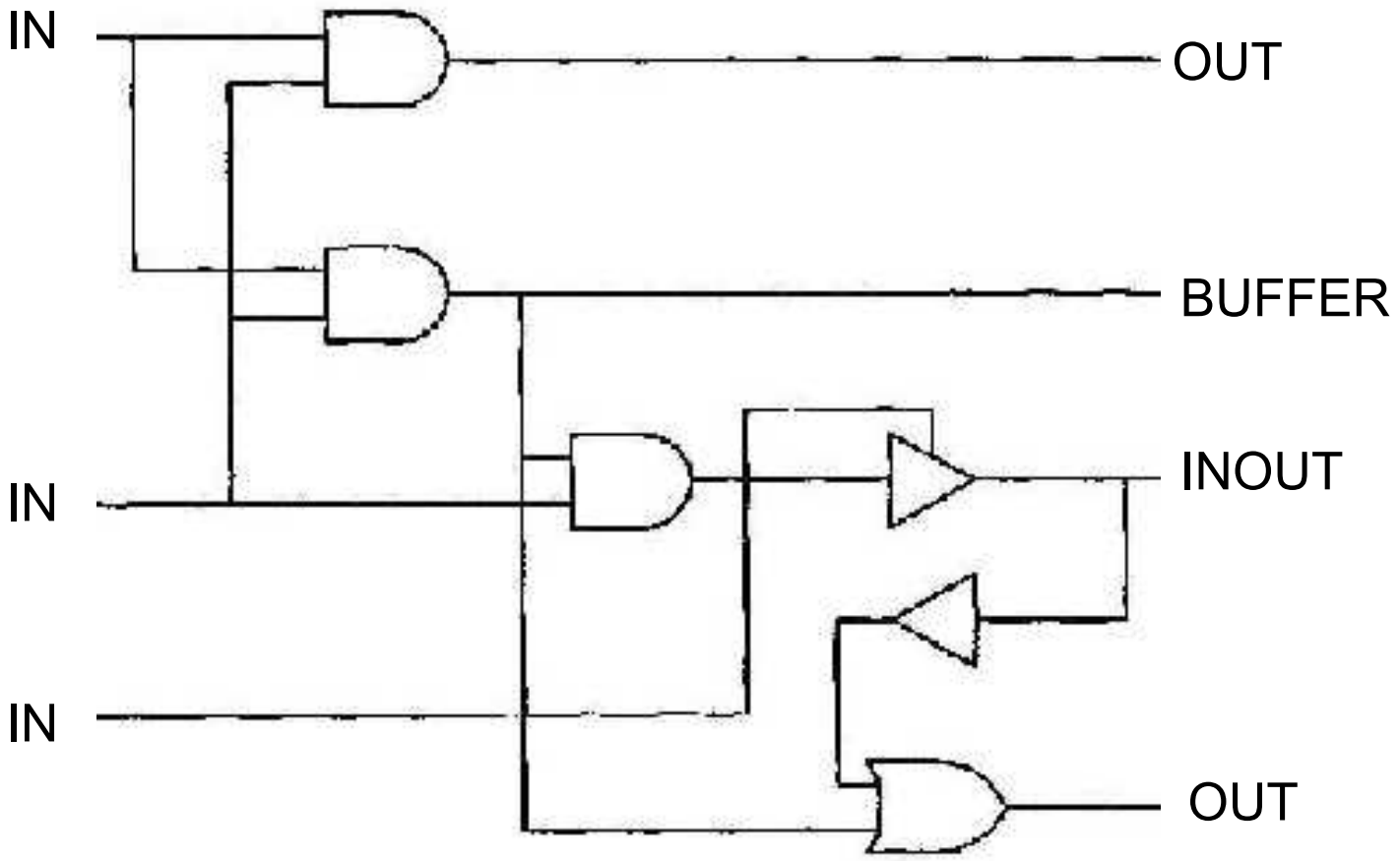
Porty (brány)

- popisují vnější signály entity
- jsou charakterizovány:
  - jménem (libovolná skupina znaků začínající písmenem)
  - módem (určuje směr toku dat):
    IN, OUT, BUFFER, INOUT
  - datovým typem (lze spojovat porty stejného typu)

Módy portů

- **IN** – data lze z portu pouze číst;
- **OUT** – data vycházejí z portu (výstupní signál nemůže být použit jako vstup uvnitř entity);
- **INOUT** – obousměrný tok (obousměrné vstupy/výstupy) – slouží pro připojení k třístavové sběrnici.
- **BUFFER** – výstup se zpětnou vazbou (může být buzen pouze z vnitřku entity - data mohou z entity pouze vystupovat, lze zpětně číst) – v nadřazené struktuře lze spojovat pouze s módem IN (ne OUT nebo INOUT);

Příklad portů



Porty (pokračování)

Syntaxe:

PORT (jméno_signálu : mód datový_typ) ;

Příklad:

```
PORT ( vstup : IN  BIT ;
          a1, b1 : IN  BIT_VECTOR (3 DOWNTO 0) ;
          vystup : OUT  BIT ) ;

soucet : OUT BIT_VECTOR (0 TO 7) ;          -- 0 je MSB, 7 je LSB
operand : IN BIT_VECTOR (4 DOWNTO 0) ;-- 4 je MSB, 0 je LSB
   -- nelze:  BIT_VECTOR (4 TO 1);
              BIT_VECTOR (0 DOWNTO 5);
```

Položka Generic

- obdoba portů, ale nepředstavuje žádný signál;
- obdoba konstanty (viditelná v entitě a přiřazených architekturách);
- používá se jako parametr (neměnící se v čase);
- použití pro lepší čitelnost, správu a konfiguraci.

Syntaxe:

GENERIC (generic_jméno : datový_typ [:= hodnota] ;

Příklad:

```
GENERIC (Delay : integer := 5) ; -- časový parametr
GENERIC (BusWidth : integer := 8 ); -- velikost objektu
GENERIC (Loop : integer := 3); -- proměnný počet smyček
```

Datové objekty

Ve VHDL jsou 4 třídy datových objektů:

- **Konstanty** (constants) – mají neměnnou hodnotu (nelze dále měnit ⇒ lze psát pouze na pravé straně přiřazení) – deklarují se v entitách, architekturách, procesech, funkcích, procedurách a slohách (package).
- **Proměnné** (variables) – používají se jako pomocné objekty (nepředstavují skutečné signály, nelze je použít jako porty) – deklarují se v procesech, funkcích a procedurách.
- **Signály** (signals) – většinou jsou fyzicky přítomné ve formě elektrických signálů – deklarují se pouze v deklarační části architektury.
- **Soubory** (files) – používají se převážně pro uložení vstupních a výstupních dat při simulaci.

Datové objekty (pokračování)

Syntaxe:

CONSTANT jméno_konstanty : datový_typ := hodnota ;
VARIABLE jméno_proměnné : datový_typ [:= hodnota] ;
SIGNAL jméno_signálu : datový_typ [:= hodnota] ;

- **Datový_typ** použijeme standardní nebo je nutné jej definovat příkazem TYPE ;
- nastavení hodnoty není podporováno syntézou (slouží jen k simulaci)

Příklady:

CONSTANT pi : **REAL** := 3.14 ;
CONSTANT rychlost : **INTEGER** ; -- defaultní hodnota: 0
VARIABLE suma : **BIT_VECTOR** (0 **TO** 3) := “0010” ;
SIGNAL select : **STD_LOGIC** ;

Viditelnost datových objektů

Datový objekt deklarovaný:

- v procesu je viditelný pouze uvnitř tohoto procesu;
- v architektuře je viditelný ve všech příkazech této architektury;
- v entitě je viditelný ve všech architekturách přidělených této entitě;
- v „package“ je viditelný ve všech návrzích užívajících tohoto „package“.

Jednoduché přiřazení signálu (simple)

- paralelní příkaz;
- vykazuje určité setrvačné zpoždění (neprovede se bezprostředně);
- datové typy na obou stranách musí být stejné.

Syntaxe:

jméno_signálu <= výraz ;

Příklad:

```
SIGNAL a, b, q : bit ;
q <= a XOR b ;
```

Přiřazovaní polí

- velikost polí na levé i pravé straně přiřazení musí být stejná
- jednotlivé elementy jsou přiřazovány podle pozice (ne podle indexu)

Příklady:

```
SIGNAL a, b, q : std_logic_vector (0 TO 1);
SIGNAL c : std_logic_vector (1 DOWNTO 0);
a <= b ;            --  a(0) <= b(0) ;   a(1) <= b(1) ;
c <= b ;            --  c(1) <= b(0) ;    c(0) <= b(1) ;
q <= a NOR b ;  --  q(0) <= a(0) NOR b(0) ;  q(1) <= a(1) NOR b(1);
a(0) <= b(1) ;
```

Polohové a jmenné přiřazování

Příklady:

```
b, e : bit;
a : bit_vector(3 DOWNTO 2);
d : bit_vector(0 TO 2);
c : bit_vector(8 DOWNTO 1);

c <= (‘0’, ‘1’, OTHERS => ‘0’);        -- polohové
d <= (‘0’, ‘1’, ‘0’);                  -- polohové
a <= (b, ‘0’);                         -- polohové
c <= (8 => ‘1’, 7 => b, 5 DOWNTO 2 => ‘1’, OTHERS => ‘0’);
                                       -- jmenné
d <= (0 => b nand e, 1 to 2 => a);  -- jmenné d(1) <= a(3), d(2) <= a(2)
c <= ”00000000”;              -- nevhodné, raději:  c <= (OTHERS => ‘0’);
```

Výběrové přiřazení signálu (selected)

Syntaxe:

WITH výběrový_signál **SELECT**
  jméno_signalu <= hodnota_1 **WHEN** hodnota_1_výběrového_signálu,
  hodnota_2 **WHEN** hodnota_2_výběrového_signálu,
  …
  hodnota_n **WHEN** hodnota_n_výběrového_signálu;

- nemá charakter prioritního přiřazení (obdoba CASE – WHEN)

Příklad:

```
WITH sel SELECT
    hd1 <= i0 WHEN “0000” TO “0100”,   --  od – do
           i1 WHEN “0101” | “0111“,    --  nebo
           i2 WHEN “1010”,
           i3 WHEN OTHERS ;            --  ostatní hodnoty
```

Podmíněné přiřazení signálu (conditional)

Syntaxe:

```
jméno_signalu <=  hodnota_1 WHEN podmínka_1 ELSE
                  hodnota_2 WHEN podmínka_2 ELSE
                  ...
                  hodnota_n WHEN podmínka_n ELSE
                  hodnota_x ;
```

- má charakter prioritního přiřazení => může vést na složitější obvod

Příklad:

```
hd1 <= i0 WHEN w = '0' ELSE
       i1 WHEN x  = '1' ELSE
       i2 WHEN y  = '1' ELSE '0';
```