Datové typy signed a unsigned

- datové typy obdobné std_logic_vector
- umožňují definovat znaménko
- slouží pro práci s aritmetickými operátory
- využívá se při volání balíků numeric_bit, numeric_std, …
- Signed: první bit znaménko
- Unsigned: kladná celá čísla

Příklad:

A má hodnotu “1010”, B má hodnotu “0100”
A a B jsou typu signed, potom A < B je “pravdivé”
A a B jsou typu unsigned, potom A < B je “nepravdivé”

Jednoduché přiřazení signálu (simple)

- paralelní příkaz, vykazuje určité setrvačné zpoždění
- datové typy na obou stranách musí být stejné

Syntaxe:

jméno_signálu <= výraz ;

Příklad:

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

Přiřazování 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) <= a(0) ;    c(0) <= a(1) ;
q <= a NOR b ;     --  q(0) <= a(0) NOR b(0) ;  q(1) <= a(1) NOR b(1) ;
a(0) <= b(1) ;
```

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' ;
```

Příkaz WAIT

- příkaz pro spuštění (pozastavení) procesu nebo procedury
- sekvenční příkaz

Syntaxe:

WAIT [**ON** seznam_signálů][**UNTIL** výraz][**FOR** čas] ;

Příklady:

```
WAIT ON s1, s2 ;        -- čekáme, dokud nenastane změna signálů
WAIT FOR 50 ns ;        -- čekáme daný čas (nesyntetizovatelné)
WAIT UNTIL enable = '1' ;       -- čekáme na pravdivost podmínky
WAIT ON a, b UNTIL clk = '1' ;          -- čekáme, dokud nenastane
      změna jednoho ze signálů a nebo b, ale současně musí clk = 1
```

Příkaz Process

- představuje nezávislý děj, který se provede při aktivaci
- užívá se zejména pro popis sekvenčních dějů
- Příkazy uvnitř procesu se vykonávají sekvenčně, ale více procesů v architektuře se vykonává paralelně

```
[jméno] : PROCESS [(clock, reset)]  -- seznam citlivých proměnných
-- deklarace procesu
BEGIN
-- příkazy procesu
END PROCESS [jméno] ;
```

- seznam citlivých proměnných je při syntéze ignorován, ale je významný pro správné provádění simulace

Spouštění procesu

Každý proces je spuštěn automaticky na začátku simulace.

Další spouštění je možné jedním ze dvou možností:

```
PROCESS (a)             -- spouštění na základě citlivostního seznamu
BEGIN
      y <= a ;
END PROCESS ;

PROCESS                 -- spouštění příkazem WAIT
BEGIN
      WAIT on a:
      y <= a ;
END PROCESS ;
```

Process – použití signálů

```
SIGNAL a, b, c : bit ;
PROCESS (b)
BEGIN
a <= b ;
c <= a ;
END PROCESS ;
```

Signál	Předešlý stav	Následující stav
b	1	0
a	1	0
c	1	**1 !**

```
PROCESS (a, b)
BEGIN
a <= b ;
c <= a ;
END PROCESS ;
```

Signál	Předeslý stav	Iterace č.1	Iterace č.2
b	1	0	0
a	1	0	0
c	1	1	0

Process – použití proměnné

```
SIGNAL b, c : bit ;
PROCESS (b)
VARIABLE a : bit ;
BEGIN
a := b ;
c <= a ;
END PROCESS ;
```

Signál	Předešlý stav	Následující stav
b	1	0
a	1	0
c	1	**0**

- proměnné lze deklarovat a používat pouze v procesu
- přiřazování je nahrazeno **:=** (např.: y := a AND b ;)
- signály lze přiřazovat do proměnných a naopak

Příkaz IF

- lze používat pouze uvnitř procesu (sekvenční příkaz)
- má charakter prioritního přiřazení => může vést na složitější obvodové zapojení (volit raději CASE – WHEN)
- podmínka je výraz vracející hodnotu typu boolean

Syntaxe:

IF podmínka1 **THEN** {sekvence_příkazů1}
[{ ELSIF podmínka2 **THEN** {sekvence_příkazů2} }]
[ELSE {sekvence_příkazů}]
END IF ;

Příkaz CASE

- lze používat pouze uvnitř procesu (sekvenční příkaz)
- musí být specifikovány všechny možnosti výrazu
- hodnoty výrazu se nesmí překrývat
- nemá charakter prioritního přiřazení (obdoba příkazu WITH – SELECT – WHEN)

Syntaxe:

CASE výraz **IS**
WHEN hodnota_1 => příkaz ;
WHEN hodnota_2 | hodnota_4 => příkaz ; -- nebo
WHEN hodnota_m **TO** hodnota_n => příkaz ;
WHEN OTHERS => příkaz ; -- příp. NULL
END CASE ;

Popis KLO procesem

- v seznamu citlivých proměnných uvést všechny vstupy
- obvod nutno plně specifikovat

Příklad KLO:

```
PROCESS (vstupy) BEGIN
  CASE vstupy IS
    WHEN "000" => segment <= "0000001";
    WHEN "001" => segment <= "1001111";
    WHEN "010" => segment <= "0010010";
    WHEN "011" => segment <= "0000110";
    WHEN "100" => segment <= "1001100";
    WHEN "101" => segment <= "0100100";
    WHEN OTHERS => segment <= "-------";   -- HW výhodnější než uvést
  END CASE;                                -- konkrétní logické hodnoty
END PROCESS;
```

Položky LIBRARY a USE

- v systému VHDL je standardně přístupná pouze knihovna „Std“ s package „standard“ - není třeba připojovat (bývá přístupná i knihovna „work“); ostatní knihovny je třeba připojit (zviditelnit) příkazem LIBRARY
- package se připojují příkazem USE (zviditelňuje specifikované položky v daném package)
- obě položky nutno v návrhu opakovat pro každou entitu a package
- pokud je třeba zpřístupnit více package z jedné knihovny, užijeme příkaz USE několikrát za sebou

Syntaxe:

LIBRARY jméno_knihovny ;

USE jméno_knihovny.jméno_package.položka ; -- příp. all

Package (knihovní balík)

- hierarchicky nad entitou a architekturou
- položky deklarované v package jsou viditelné v celém návrhu (v package nelze uvést deklaraci entity nebo architektury)
- Package obsahuje dvě části:
  - deklarační část (příkaz **PACKAGE**) – pro deklarace hlaviček funkcí a procedur, typů a podtypů, konstant a signálů, komponent (vlastní popis musí být mimo sekci package), atributů, aj.
  - vlastní tělo (příkaz **PACKAGE BODY**) – pro definice podprogramů, funkcí, aj.)

Package - příklad

```
PACKAGE system IS
   CONSTANT pocet : INTEGER := 2 ;
   PROCEDURE add (SIGNAL a, b : IN BIT ; suma : OUT BIT ) ;
END system ;

PACKAGE BODY system IS
   PROCEDURE add (SIGNAL a, b : IN BIT ; suma : OUT BIT ) ;
   BEGIN
        suma <= a  XOR  b ;
   END add ;
END system ;
```

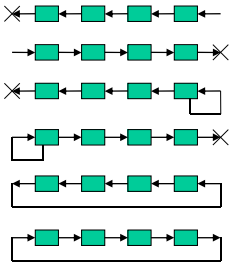
Standardizované package IEEE

Jsou uloženy v knihovně IEEE;

- **std_logic_1164** – definice vícehodnotovou logiku, vybrané konverzní funkce a logické operátory s std logikou
- **numeric_bit, numeric_std** – definice aritmetických a logických operací nad typy unsigned a signed (nahrazují starší package std_logic_arith, std_logic_unsigned a std_logic_signed)
- **math_real** – definuje operace v plovoucí řádové čárce (zatím pouze pro simulace)
- **fixed_pkg, fphdl_pkg** – připravují se syntetizovatelné balíky pro podporu matematických operací v pevné a plovoucí řádové čárce

Operátory

- logické: **NOT, AND, OR, NAND, NOR, XOR, XNOR**
- relační: **=, /=, >, <, >=, <=**
- aritmetické: **+, -, *, /, mod** (Modulus), **rem** (Remainder), **abs** (Absolute Value), ** (exponent)
- posuvu a rotace
  - **SLL** (Shift Left Logical)
  - **SRL** (Shift Right Logical)
  - **SLA** (Shift Left Arithmetic)
  - **SRA** (Shift Right Arithmetic)
  - **ROL** (Rotate Left Logical)
  - **ROR** (Rotate Right Logical)
- slučující operátor &



Operátory – priorita

1) **, ABS, NOT -- nejvyšší priorita
2) *, /, MOD, REM
3) znaménka +, -
4) +, -, &
5) SLL, SRL, SLA, SRA, ROL, ROR
6) =, /=, <, <=, >, >=
7) AND, OR, NAND, NOR, XOR, XNOR -- nejnižší priorita

Prioritu operátorů lze měnit oblými závorkami, jinak se vyhodnocují zleva doprava

Pozn.: operátory NAND a NOR nejsou asociativní !

Části jednorozměrných polí (slice)

```
SIGNAL a, b : bit_vector (3 DOWNTO 0) ;
SIGNAL c : bit_vector (0 TO 3) ;

b (3 DOWNTO 2) <= " 01" ; -- část pole b (array slicing)
b (2 TO 3)   -- nelze, je-li b definováno jako (3 DOWNTO 0)
c (1 TO 2) <= a (3 DOWNTO 2) ;
c <= a ;
c (2 DOWNTO 1) <= a (2 TO 3) ;  -- nelze
```

Slučující operátor (concatenation) &

Slučuje jednorozměrná pole (včetně řetězců):

Příklad – použití pro sloučení dvou signálů:

```
a, b : IN bit_vector (1 DOWNTO 0) ;
c : OUT bit_vector (3 DOWNTO 0) ;
c <= a & b ;
-- c(3) <= a(1);  c(2) <= a(0); c(1) <= b(1);  c(0) <= b(0);
```

Příklad – použití při definici posuvného registru:

```
VARIABLE shifted, shiftin : bit_vector (0 TO 3) ;
shifted := shiftin (1 TO 3) & '0' ;
```

Příklad – použití pro násobení a dělení (Q = C / 16 + C * 4):

```
SIGNAL C, Q : std_logic_vector (7 DOWNTO 0) ;
Q <= „0000“ & C(7 DOWNTO 4) + C(5 DOWNTO 0) & „00“ ;
```

Náběžné a sestupné hrany

clk'event AND clk = '1' -- náběžná hrana
clk'event AND clk = '0' -- sestupná hrana

SIGNAL clock : boolean ;
IF NOT clock AND clock'event -- sestupná hrana

v knihovně IEEE 1164 existují funkce **rising_edge ( )** a **falling_edge ( )**
(knihovnu IEEE 1164 nutno deklarovat – příkazy LIBRARY a USE)

- tyto funkce nelze užívat v log. operacích: IF NOT falling_edge (clk)
- lze psát: enclk <= clk AND clk_en ; … IF rising_edge(enclk) THEN

WAIT UNTIL (clk'event AND clk'last_value = '0');
IF (num'event AND num = '0') THEN q <= c(5) ; END IF ;

Použití hodinového signálu v procesu

```
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                          -- použitím příkazu WAIT
BEGIN
WAIT ON (hodinový_signál) UNTIL (specifikace_hrany_hod_signálu)
výstupní_signál <= vstupní signál ;
… další sekvenční příkazy …
END PROCESS ;
```

Použití synchronního resetu (1)

```
PROCESS (hodinový_signál)        -- použitím citlivých proměnných
BEGIN
   IF (podmínka_hrany_hod_signálu) THEN
         IF (podmínka_resetu) THEN
         výstupní signál <= resetovací_hodnota ;
         ELSE
         výstupní_signál <= vstupní signál ;
         … další sekvenční příkazy …
         END IF ;
   END IF ;
END PROCESS ;
```

- nepoužívat ELSE ve vnějším IF s podmínkou hrany hodinového signálu
- nepoužívat v procesu více příkazů IF (pouze vložené)

Použití synchronního resetu (2)

```
PROCESS                                    -- použitím příkazu WAIT
BEGIN
   WAIT ON (hodinový_signál) UNTIL (specifikace_hrany_hod_signálu)
        IF (podmínka_resetu) THEN
        výstupní_signál <= resetovací_hodnota ;
        ELSE
        výstupní_signál <= vstupní signál ;
        … další sekvenční příkazy …
        END IF ;
END PROCESS ;
```

Použití asynchronního resetu (1)

Použitím citlivých proměnných:

```
   PROCESS (hodinový_signál, resetovací_signál)
   BEGIN
        IF (resetovací_podmínka) THEN
        výstupní signál <= resetovací_hodnota ;
        ELSIF (podmínka_hrany_hod_signálu) THEN
                výstupní_signál <= vstupní signál ;
                … další sekvenční příkazy …
        END IF ;
   END PROCESS ;
```

- nepoužívat příkaz ELSE po detekci hrany hodinového signálu
- nepoužívat v procesu více příkazů IF (pouze vložené)

Použití asynchronního resetu (2)

Použitím příkazu WAIT:

```
   PROCESS
   BEGIN
        WAIT ON hodinový_signál, resetovací_signál ;
        IF (resetovací_podmínka) THEN
        výstupní signál <= resetovací_hodnota ;
        ELSIF (podmínka_hrany_hod_signálu) THEN
                výstupní_signál <= vstupní signál ;
                … další sekvenční příkazy …
        END IF ;
   END PROCESS ;
```

Klopný obvod řízený hladinou (latch)

- v návrzích raději nepoužívat (vliv střídy hod. signálu)
- vedou na jednodušší obvodové zapojení
- někdy se generují v důsledku špatného popisu kombinační logiky (chybějící ELSE, neúplný příkaz CASE)
- v seznamu citlivých proměnných nutno uvést i datové vstupy

Příklad: (při clk = ‘0’ zachovává předchozí stav)

```
   PROCESS (clk, a, b)
   BEGIN
        IF (clk = ‘1’) THEN
            y <= a AND b ;   -- u KLO by bylo: ELSE y <= ‘0’ ;
        END IF ;
   END PROCESS ;
```