Zložené údajové typy

Syntax deklarácie zloženého údajového typu:

composite_type_definition ::= array_type_definition
| record_type_definition

Syntax deklarácie typu pole:

array_type_definition ::= unconstrained_ array_definition
| constrained_ array_definition

unconstrained_ array_definition ::=
array (index_subtype_definition{, index_subtype_definition})
of element_subtype_indication
index_subtype_definition ::= type_mark **range** <>
type_mark ::= *type*_name
| *subtype*_name

constrained_ array_definition ::=
array index_constraint **of** element_subtype_indication
index_constraint ::= (discrete_range{, discrete_range})
discrete_range ::= discrete_subtype_indication
| range
range ::= range_attribute_name
| simple_expression direction simple_expression

Syntax deklarácie typu záznam:

record_type_definition ::=
record
element_declaration
{ element_declaration }
end record [record_type_simple_name]
element_declaration ::= identifier_list: element_subtype_definition;

Polia

```
-- definície poľa s hranicami rozmeru určenými špecifikáciou rozsahu
type Word is array (15 downto 0) of Bit;
type Severity_level_stats is array (Note to Failure) of Integer;

--definície poľa s indikáciou podtypu (subtype indication)
 -- definície podtypov
type Column is range 1 to 80 ;
type Row is range 1 to 24 ;

-- definície dvojrozmerného poľa s hranicami rozmerov určenými indikáciou podtypu
type Matrix is array (Row, Column) of Boolean ;
type Reduced_matrix is array
        (Row range 1 to 10, Column range 1 to 40) of Boolean ;

-- definícia neohraničeného poľa
type Screen is array (Integer range <>, Integer range <>) of Pixel;
-- typ Pixel je definovaný na inom mieste

-- preddefinované neohraničené typy polí:
type Bit_vector is array (Natural range <>) of Bit;
type String is array (Positive range <>) of Character;
```

Agregáty a reťazcové literály

```
-- Declaration of typ Conversion_array:
type Clock_level is (Low, Rising, High, Falling);
type Conversion_array is array (Clock_level) of Bit;
-- Agregát typu Conversion_array – pozičné priradenie
  ('0','1','1','0')

-- menovité priradenie
 (Low => '0', Rising => '1', High => '1', Falling => '0')
 (Rising => '1', High => '1', Falling => '0', Low => '0')

-- výbery oddelené zvislou čiarou:
 (Low | Falling => '0', Rising | High => '1')

type Hex_letter is ('0','1','2','3','4','5','6','7','8','9', 'A','B','C','D','E','F');
type Hex_index is array (Hex_letter) of Boolean;
-- špecifikácia rozsahu
constant Is_decimal_digit : Hex_index := ('0' to '9' => True, 'A' to 'F'=> False);
type Eight is range 1 to 8;
type Eight_digit is array (Eight) of Hex_letter;
-- indikácia podtypu
variable Hex_value: Eight_digit := (Eight => '0');

-- použitie vyhradeného slova others:
Constant Is_decimal_digit : Hex_index := ('0' to '9' => True, others => False);
variable Hex_value : Eight_digit := (others => '0');
```

```
-- použitie reťazcových literálov
type Four_level_logic is ('X', '0', '1', 'Z');
type Pair is array (1 downto 0) of Four_level_logic;
constant Reset : Pair := "0Z";
-- equivalent to (1 => '0', 0 => 'Z');
constant Clear : Pair := "Z0";
-- equivalent to (1 => 'Z', 0 => '0');
```

Agregáty sa dajú použiť aj pri viacrozmerných poliach:

```
type REG_32_BIT is array (31 downto 0) of Bit;
type ROM_TYPE is array (Natural  range <>) of type REG_32_BIT;
type ARRAY_2D is array (Natural  range <>, Natural  range <>) of Bit;

signal FOUR_WORDS: ROM_TYPE(1 to 4) := (X”2F3C_5456”, X”FF22_A5B9”,
X”9900_AD51”, X”FFFF_FFFF”);
signal tabulka: ARRAY_2D(1 to 5, 1 to 10) :=
                                        ((‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’),
                                        (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’),
                                        (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’),
                                        (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’),
                                        (‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’, ‘1’));
```

Použitie objektov zloženého typu.

```
type Column is range 1 to 80 ;
type Row is range 1 to 24 ;
type Matrix is array (Row, Column) of Boolean ;

signal S : Matrix;
-- References (indexed names) to elements of signal S
S (1, 1) -- type Boolean
S (3, 14) -- type Boolean

type Byte is array (7 downto 0) of Bit;
type Memory is array (0 to 2**16-1) of Byte;

signal S_byte : Byte;
signal S_Memory : Memory;

-- References (slice name) to contiguous elements of array signals:
-- slice of 3 elements
S_byte (3 downto 1)
-- slice of 2**15 elements
S_Memory (2**15 -1 to 2**16-1)
-- slice of 1 element
S_Memory  (0 downto 0)

-- bázový typ
-- type anonymous is array (Integer range <>) of Bit;
```

Záznamy

```
type Opcode is   (Add, Add_with_carry, Sub, Sub_with_carry, Complement);
type Address is range 16#0000# to 16#FFFF#; -- čiže 2^16
 type Instruction is record
   Opcode_field: Opcode;
   Operand1 : Address;
   Operand2 : Address;
 end record;
type Register_bank is record
 F0, F1: Real;
 R0, R1: Integer;
 A0, A1: Address;
 IR: Instruction;
end record;
```

Agregáty

```
type Rational is record
 Numerator : Integer;
 Denominator: Integer;
end record
-- pozičné priradenie elementov
 (155, 2077)
-- menovité priradenie
 (Numerator => 155, Denominator => 2077)
 (Denominator => 2077, Numerator => 155)
 (Low => '0', Rising => '1', High => '1', Falling => '0')
 (Rising => '1', High => '1', Falling => '0', Low => '0')
(Numerator | Denominator => 1)
Constant One : Rational := (others =>1);
signal S : Rational;
-- References (selected names) to element of signal S
S.Numerator -- type is Integer
S.Denominator -- type is Integer
```

Podtypy, rezolučné podtypy a rezolučné signály

Syntax deklarácie podtypu:

subtype identifier **is** subtype_indication;

subtype_indication ::= [resolution_function_name] type_mark [constraint]
type_mark ::= *type*_name | *subtype*_name
constraint ::= range_constraint | index_constraint
range_constraint ::= **range** range
index_constraint ::= (discrete_range{, discrete_range})

-- Range constraint:

subtype Lowercase_letters **is** Character **range** 'a' **to** 'y';

-- Index constraint:

subtype Register **is** Bit_vector (7 **downto** 0);

-- Preddefinované podtypy

subtype Positive **is** Integer **range** 1 **to** Integer'High ;

subtype Natural **is** Integer **range** 0 **to** Integer'High ;

Rezolučná funkcia musí spĺňať nasledujúce podmienky:

- Funkcia má iba jeden parameter, ktorého typ je neohraničené jednorozmerné pole hodnôt spomenutého typu.
- Typ hodnoty, ktorú funkcia vracia musí byť zhodný s typom elementu jej parametra.

Deklarácia rezolučných signálov:

type Bit4 **is** ('X', '0', '1', 'Z');

type Bit4_Vector **is array** (Integer **range** <>) **of** Bit4;

-- A resolution function declaration:

function Wired_Or (Input : Bit4_Vector) **return** Bit4;

-- Available resolved signal declarations:

signal Resolved_S1 : Wired_Or Bit4;

subtype Resolved_Wire **is** Wired_Or Bit4;

signal Resolved_S2 : Resolved_Wire ;

-- S3 is not a resolved signal
signal S3 : Bit4 ;

```
function Wired_Or (Input: Bit4_Vector) return Bit4 is
    variable Result: Bit4 := '0';
begin
    for i in Input'Range loop
        if Input(i) = '1' then
            Result := '1';
            exit;
        elsif Input(i) = 'X' then
            Result := 'X';
        else -- Input(i) = 'Z' or '0'
            null;
        end if;
    end loop;
    return Result;
end Wired_Or;

Source_1:       Resolved_S1 <= '1';
Source_2:       Resolved_S1 <= '0';
```

Atribúty

Atribút (attribute) je ***charakteristika prvku*** jazyka VHDL, ktorá má svoje meno.

Všeobecná forma zápisu mena atribútu (attribute_name)

attribute_name ::= name ' attribute_identifier [(parameter)]

Preddefinované atribúty pre všetky skalárne podtypy

Left, Right, High a ***Low*** – reprezentujú hodnotu

type Bit_position **is range** 15 **downto** 0 ;
type Fraction **is range** -0.999999 **to** 0.999999 ;
type Opcode **is** (Add, Add_with_carry, Sub, Sub_with_carry, Complement);
subtype Adding_opcode **is** Opcode **range** Add **to** Add_with_carry ;

Bit_position'Left = 15
Bit_position'Low = 0
Fraction'Right = 0.999999
Fraction'High = 0.999999

Opcode'Left = Add
Opcode'High = Complement
Adding_opcode'Right = Add_with_carry

Preddefinované atribúty pre diskrétne a fyzikálne podtypy

Pos, Val, Succ, Pred, Leftof, Rightof - reprezentujú hodnotu

Bit_position'Pos(15) = 15
Bit_position'Val(15) = 15
Time'Pos(1 fs) = 1
Time'Pos(1 ps) = 1000
Opcode'Pos(Add) = 0

Opcode'Pos(Complement) = 4
Opcode'Val(2) = Sub
Opcode'Val(-1) is undefined
Opcode'Val(5) is undefined

Vo všeobecnosti platia rovnice:
Pre vymenovaný typ T
T'Pos(T'Left) = 0
Pre celočíselný typ T
T'Pos(T'Left) = T'Left
Pre fyzikálny a diskrétny typ T
T'Succ(X) = T'Val(T'Pos(X)+1)
T'Pred(X) = T'Val(T'Pos(X)-1)

Pre fyzikálny a diskrétny typ T so stúpajúcim rozsahom

T'Rightof(X) = T'Succ(X)

T'Leftof(X) = T'Pred(X)

Pre fyzikálny a diskrétny typ T s klasajúcim rozsahom

T'Rightof(X) = T'Pred(X)

T'Leftof(X) = T'Succ(X)

Ďalšie príklady:

Bit_position'Pos(3) = 3

Bit_position'Pred(14) = 13

Bit_position'Leftof (14) = 15

Time'Pred(1 ps) = 999

Opcode'Pos(Sub) = 2

Opcode'Succ(Add) = Add_with_carry

Opcode'Rightof(Sub) = Sub_with_carry

Preddefinované atribúty pre typ ohraničené pole A a objekty typu A

Left, Right, High, Low, Length - reprezentujú hodnotu

Range a ***Reverse_range*** - reprezentujú rozsah

Pre ľubovoľný typ ohraničené pole alebo objekt typu pole A a podtyp T N-tého rozmeru A platia nasledujúce rovnosti:

A'Right[(N)] = T'Right

A'Left[(N)] = T'Left

A'High[(N)] = T'High

A'Low[(N)] = T'Low

A'Length[(N)] = T'Pos(A'High(N)) – T'Pos(A'Low(N)) + 1

```
type Window is array (1 to 12, 1 to 40) of Pixel ;
-- used as a discrete range:
type Row_info is array (Window'Range(1)) of Boolean ;
type Columm_info is array (Window'Range(2)) of Boolean ;
-- used in a range constraint:
subtype Columm_number is Integer range Window'Range(2) ;
```

Window'Right(1) = 12 = Window'High

Window'Left = 1 = Window'Low

Window'Length = 12

Window'Length(2)= 40

Preddefinované atribúty signálov

1) Atribúty, ktoré definujú signály: ***Delayed, Stable, Quiet, Transaction.***
2) Atribúty, ktoré reprezentujú funkcie, poskytujúce informácie o signáloch: ***Active, Event, Last_Active, Last_Event, Last_Value.***

Nech S je signal.

S'Delayed(T) – reprezentuje signál rovnakého typu, ako S, oneskorený o čas T.

S'Stable[(T)] – reprezentuje signál typu Boolean, ktorý nadobudne hodnotu TRUE, ak počas posledných *T* časových jednotiek nenastala udalosť na signáli S.

S'Quiet[(T)] – reprezentuje signál typu Boolean, ktorý nadobudne hodnotu TRUE, ak počas posledných *T* časových jednotiek nenastala transakcia na signáli S.

S'Transaction – reprezentuje signál typu Bit, ktorého hodnota sa mení pri každej transakcii na signále S.

S'Active – reprezentuje funkciu, ktorá vracia hodnotu TRUE, ak v danom simulačnom čase nastala transakcia na signáli S.

S'Event – reprezentuje funkciu, ktorá vracia hodnotu TRUE, ak v danom simulačnom čase nastala udalosť na signáli S.

S'Last_Active – reprezentuje funkciu, ktorá vracia čas, ktorý uplynul od poslednej transakcie na signáli S.

S'Last_Event – reprezentuje funkciu, ktorá vracia čas, ktorý uplynul od poslednej udalosti na signáli S.

S'Last_Value – reprezentuje funkciu, ktorá vracia hodnotu signálu S pred poslednou udalosťou.

Uvažujme signál X1 typu Bit, ktorého časový priebeh je definovaný nasledovne:

```
X1<=  '1' after 5ns,
      '1' after 7ns,
      '0' after 10ns,
      '0' after 15ns,
      '1' after 20ns,
      '0' after 25ns;
```

V simulačnom čase **15 ns** majú atribúty signálu X1 nasledovné hodnoty:

X1'Delayed(8 ns) = '1'
X1'Delayed(2 ns) = '0'
X1'Stable(8 ns) = FALSE
X1'Stable(2 ns) = TRUE
X1'Quiet (3 ns) = FALSE
X1'Quiet = FALSE
X1'Active = TRUE
X1'Event = FALSE
X1'Last_Active = 0 ns
X1'Last_Event = 5ns
X1'Last_Value = '1'

Preddefinované operátory

Skupina	Symbol	Funkcia	typ operandu	typ operandu P	typ výsledku
aritmetické (binárne)	+	**sčítanie**	numerický	rovnaký	rovnaký
	-	**odčítanie**			
	*	**násobenie**	celočíselný	rovnaký	rovnaký
			reálny	rovnaký	rovnaký
	/	**delenie**	fyzikálny	INTEGER alebo REAL	ako ľavý
			fyzikálny	rovnaký fyzikálny	celočíselný
	mod	**modulo**	celočíselný	rovnaký	rovnaký
	rem	**zvyšok**			
	**	**umocnenie**	celočíselný	NATURAL	ako ľavý
			reálny	INTEGER	ako ľavý
aritmetické (unárne)	+	**plus**	numerický		rovnaký
	-	**mínus**			
	abs	**abs. hodnota**			

relačné (binárne)	=	**rovnosť**	ľubovoľný, okrem súboru	rovnaký	BOOLEAN
	/=	**nerovnosť**			
	<	**menší**	skalárny	rovnaký	BOOLEAN
	>	**väčší**	jednorozmerné pole diskrétnych prvkov	rovnaký	BOOLEAN
	<=	**menší rovný**			
	>=	**väčší rovný**			
logické (binárne)	**and**	**logický súčin**	BIT	rovnaký	rovnaký
	or	**logický súčet**	BOOLEAN	rovnaký	rovnaký
	nand	**negácia log. súčinu**	jednorozmerné pole prvkov typu BIT alebo BOOLEAN	rovnaký	rovnaký
	nor	**negácia log. súčtu**			
	xor	**vylučovacie or**			
logické(unárne)	**not**	**negácia**	ako binárne logické		rovnaký
pripojenie	**&**	**concatenation**	jednorozmerné pole alebo element jednorozmerného poľa	rovnaký alebo element typu druhého operandu	jednorozmerné pole

A mod B má znamienko B a platí:

$$|A \bmod B| < |B| \wedge \exists N \in Integer; A \bmod B = A - (B * N)$$

A rem B má znamienko A a platí:

$$|AremB| < |B| \wedge AremB = A - (A / B) * B$$

5 mod 3 = 2	5 rem 3 = 2
(-5) mod 3 = 1	(-5) rem 3 = -2
(-5) mod (-3) = -2	(-5) rem (-3) = -2
5 mod (-3) = -1	5 rem (-3) = 2

Precedencia operátorov

zmiešané	**, abs, not
násobenie	*, /, mod, rem
znamienkové	+, -
sčítanie	+, -, &
posuvy a rotácie	sll, srl, sla, sra, rol, ror
relačné	=, /=, <, >, <=, >=
logické	and, or, nand, nor, xor, xnor

Základné konštrukcie VHDL

Medzi základné konštrukcie jazyka VHDL patrí: *deklarácia entity* (entity declaration), *telo architektúry* (architecture body), *podprogram* (subprogram), *deklarácia balíka* (package declaration) a *telo balíka* (package body).

Syntax deklarácie entity (Entity Declaration)

entity_declaration ::=
    entity identifier **is**
        [**generic** interface-list;]
        [**port** interface-list;]
        [entity_declarations]
    [**begin**
        entity_statements]
    end [**entity**] [identifier];

Syntax tela architektúry (architecture body)

architecture_body ::=
    architecture identifier **of** entity_name **is**
        [architecture_declarations]
    begin
        [architecture_statements]
    end [**architecture**] [identifier];

základné deklarácie (basic declarations):
    deklarácia typu
    deklarácia podtypu
    deklarácia konštanty
    deklarácia súboru
    deklarácia aliasu
    deklarácia podprogramu
    use klauzula

Ďalšie deklarácie povolené v deklarácii entity:
    deklarácia signálu
    telo podprogramu
    deklarácia atribútu
    špecifikácia atribútu
    špecifikácia odpojenia

V deklarácii entity sú povolené príkazy (pasívne):
    paralelný príkaz tvrdenia (concurrent assertion statement)
    paralelné volanie procedúry
    príkaz proces

V tele architektúry sú povolené všetky paralelné príkazy:

paralelný príkaz priradenia signálu (concurrent signal assignment statement)
paralelný príkaz tvrdenia (concurrent assertion statement)
paralelné volanie procedúry
príkaz proces
príkaz vytvorenia inštancií komponentov
príkaz bloku
príkaz generate

Podprogramy

deklarácia podprogramu - syntax:

subprogram_declaration ::= subprogram_specification;

subprogram_specification ::=

procedure identifier [interface_list]

|**function** identifier [interface_list] **return** type-mark

--bez parametrov – najjednoduchší tvar

procedure P;

function Limit **return** Real;

--objektová trieda parametra je implicitná

procedure Mod(X:**inout** Integer); --variable

function Mod(X:Integer) **return** Byte; --constant

--objektová trieda param. je explicitná

procedure Mod

(**variable** X:**inout** Integer);

function Mod

(**constant** X:**in** Integer)

return Byte;

--parametre sú obj. triedy signal

procedure Xor2(**signal** In1, In2: Bit; --implicitný režime **in**

signal Out1: **out** Bit);

Telo podprogramu - syntax:

subprogram_body ::= subprogram_specification **is**
[subprogram_declarations]
begin
[subprogram_statements]
end [**procedure** | **function**] [identifier];

V tele podprogramu sa môžu nachádzať:

- základné deklarácie
- deklarácia premennej
- telo podprogramu
- deklarácia atribútu
- špecifikácia atribútu

Príklad tela podprogramu:

```
function Min(X,Y:Integer) return Integer is
begin
        if X<Y then return X;
        else return Y;
end if;
end Min;
```

Balíky a klauzula USE

Deklarácia balíka - syntax:

package_declaration ::= **package** identifier **is**
[package_declarations]
end [**package**] [identifier];

v deklarácii balíka sa môžu nachádzať:

- základné deklarácie
- deklarácia signálu
- deklarácia atribútu
- špecifikácia atribútu
- špecifikácia odpojenia
- deklarácia komponentu

```
package Logic is
        type Three_level_logic is ('0','1','Z');
        constant Unknown_value: Three_level_logic :='0';
        function Invert (Input: Three_level_logic) return Three_level_logic;
end Logic;
```

Telo balíka - syntax:

```
package_body ::=    package body identifier is
                            [package_body_declarations]
                    end [package body] [identifier];
```

V tele balíka sa môžu nachádzať:

- základné deklarácie
- telo podprogramu.

```
package body Logic is
        --telo podprogramu funkcie Invert
        function Invert (Input: Three_level_logic) return Three_level_logic is
        begin
                case Input is
                        when '0'=> return '1';
                        when '1'=> return '0';
                        when 'Z'=> return 'Z';
                end case;
        end Invert;
end Logic;
```

Príklad použitia **use** klauzuly:

```
use Logic.Three_level_logic, Logic.Invert;
entity Inverter is
        port(X:in Three_level_logic;Y:out Three_level_logic);
end Inverter;
architecture Inverter_body of Inverter is
begin
```

```
        process
        begin
                Y<=Invert(X) after 10ns;
                wait on X;
        end process;
end Inverter_body;
```

Väzby medzi konštrukciami

Syntax zoznamu prepojení:

interface_list::= interface_element {; interface_element}
interface_element::= interface_constant_declaration
| interface_signal_declaration
| interface_variable_declaration
interface_signal_declaration ::=
[**signal**] identifier_list: [mode] subtype_indication [**bus**] [:= *static*_expression]
mode::= **in** | **out** | **inout** | **buffer | linkage**

Entity, Komponenty, Bloky		
Objekt	Trieda	Režimy
port	signal	**in,out,inout,buffer, linkage**
generic	constant	**in**

Podprogramy		
Objekt	Trieda	Režimy
parameter	signal	**in,out,inout**
parameter	constant	**in**
parameter	variable	**in,out,inout**

Syntax asociačného zoznamu:

association_list ::= association_element {, association_element}
association_element ::= [formal_part =>] actual_part