دوره مقدماتی آشنایی با FPGA و VHDL

انواع داده

محمدرضا عزیزی

امیرعلی ابراهیمی

بهار ۱۳۹۷

مقدمه

- آشنایی با انواع داده و استفاده مناسب و به جا از آن‌ها برای نوشتن یک کد بهینه و کارا VHDL ضروری است.
- در این بخش، انواع داده‌های از بنیادی، با تاکید بر نوع داده‌های سنتزپذیر بیان خواهد شد.
- در مورد نحوه تعریف نوع داده جدید توسط کاربر و تبدیل انواع داده به یکدیگر نیز بحث خواهد شد.

کتابخانه‌ها و انواع داده (۱)

- زبان VHDL دارای انواع داده از پیش تعریف شده ای است که طبق استانداردهای IEEE 1067 و IEEE 1164 مشخص شده اند.

- کتابخانه ***std***:
  - پکیج ***standard***:
    - BIT
    - BOOLEAN
    - INTEGER
    - REAL

کتابخانه‌ها و انواع داده (۲)

- کتابخانه ***ieee***:
  - پکیج ***std_logic_1164***:
    - STD_LOGIC
    - STD_ULOGIC
  - پکیج ***std_logic_arith***:
    - SIGNED
    - UNSIGNED
    - *conv_integer(p)*
    - *conv_unsigned(p, b)*
    - *conv_signed(p, b)*
    - *conv_std_logic_vector(p, b)*
  - پکیج ***std_logic_siged*** و ***std_logic_unsigned***:
    - پکیج‌هایی که عملیاتی بر روی نوع داده std_logic_vector تعریف می‌کند که گویی نوع‌داده signed یا unsigned هستند.

انواع داده از پیش تعریف شده (۱)

- BIT (و BIT_VECTOR):
  - منطق دو سطحی (’0‘ و ’1‘)
  - مثال:

```
SIGNAL x: BIT;
```

x به عنوان یک سیگنال یک رقمه (one-digit) از نوع BIT تعریف شده است.

```
SIGNAL y: BIT_VECTOR(3 DOWNTO 0);
```

y به عنوان به عنوان یک وکتور ۴ بیتی تعریف شده است. سمت چپ‌ترین بیت = با ارزش‌ترین بیت

```
SIGNAL w: BIT_VECTOR(0 TO 7);
```

w یک بیت وکتور ۸ بیتی است. سمت راست‌ترین بیت = با ارزش‌ترین بیت

انواع داده از پیش تعریف شده (۲)

```
x <= '1';
```

برای **مقداردهی یک بیته**، از ' ' استفاده می‌شود.

```
y <= "0111";
```

برای **مقداردهی وکتورها**، از " " استفاده می‌شود.

```
w <= "01110001";
```

باارزش‌ترین بیت (MSB) = ۱

انواع داده از پیش تعریف شده (۳)

- STD_LOGIC (و STD_LOGIC_VECTOR):
  - منطق ۸ سطحی

- ‘**X**’ Forcing Unknown Impossible to determine this value/result.
- ‘**0**’ Forcing Low logic 0
- ‘**1**’ Forcing High logic 1
- ‘**Z**’ High impedance (synthesizable tri-state buffer)
- ‘**W**’ Weak unknown Weak signal, can't tell if it should be 0 or 1.
- ‘**L**’ Weak low Weak signal that should probably go to 0
- ‘**H**’ Weak high Weak signal that should probably go to 1
- **‘–’** Don’t care

انواع داده از پیش تعریف شده (۴)

- مثال:

```
SIGNAL x: STD_LOGIC;
```

x به عنوان یک سیگنال یک رقمه (one-digit) از نوع STD_LOGIC تعریف شده است.

```
SIGNAL y: STD_LOGIC_VECTOR(3 DOWNTO 0) := “0001”;
```

y به عنوان یک وکتور ۴ بیتی تعریف شده است. سمت چپ‌ترین بیت = با ارزش‌ترین بیت

مقداردهی اولیه (دلخواه) y برابر با "0001" است.

نکته: از عملگر ":=" برای **مقداردهی اولیه** استفاده می‌شود.

- **اتصال چند سیگنال std_logic به یک node:**

Table 3.1
Resolved logic system (STD_LOGIC).

	X	0	1	Z	W	L	H	-
X	X	X	X	X	X	X	X	X
0	X	0	X	0	0	0	0	X
1	X	X	1	1	1	1	1	X
Z	X	0	1	Z	W	L	H	X
W	X	0	1	W	W	W	W	X
L	X	0	1	L	W	L	W	X
H	X	0	1	H	W	W	H	X
-	X	X	X	X	X	X	X	X

انواع داده از پیش تعریف شده (۵)

- STD_ULOGIC (و STD_ULOGIC_VECTOR):
  - منطق ۹ سطحی
    - ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '–')
      - 'U': uninitialized
  - این نوع داده به صورت اتوماتیک multiple assignment را resolve نمی‌کند.
  - std_logic زیرنوع (subtype) ای از این نوع داده است.
- BOOLEAN:
  - True و False
- INTEGER:
  - ۳۲ بیتی
  - ۲/۱۴۷/۴۸۳/۶۴۷- تا ۲/۱۴۷/۴۸۳/۶۴۷+

انواع داده از پیش تعریف شده (۶)

- NATURAL:
  - اعداد صحیح نامنفی
  - ۰ تا ۲/۱۴۷/۴۸۳/۶۴۷+
- REAL:
  - اعداد حقیقی
  - 1.0E38- تا 1.0E38+
  - ***غیرقابل سنتز***
- الفاظ فیزیکی(Physical literals):
  - نمایش خصوصیت‌های فیزیکی مانند زمان، ولتاژ و ...
  - قابل استفاده در شبیه‌سازی و ***غیرقابل سنتز***

انواع داده از پیش تعریف شده (۷)

- الفاظ کاراکتری(Character literals):
  - یک کاراکتر ASCII یا رشته ای از این کاراکترها
  - ***غیرقابل سنتز***
- SIGNED و UNSIGNED:
  - تعریف شده در پکیج *std_logic_arith* از کتابخانه *ieee*
  - دارای ظاهری شبیه به STD_LOGIC_VECTOR ولی اعمال حسابی بر روی آن‌ها قابل انجام است.
  - **← در بخش «نوع‌داده های SIGNED و UNSIGNED» بیشتر مورد بررسی قرار می‌گیرند.**

انواع داده از پیش تعریف شده (۸)

- خلاصه:

Table 3.2
Synthesizable data types.

Data types	Synthesizable values
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	From 0 to +2, 147, 483, 647
INTEGER	From –2,147,483,647 to +2,147,483,647
SIGNED	From –2,147,483,647 to +2,147,483,647
UNSIGNED	From 0 to +2,147,483,647
User-defined integer type	Subset of INTEGER
User-defined enumerated type	Collection enumerated by user
SUBTYPE	Subset of any type (pre- or user-defined)
ARRAY	Single-type collection of any type above
RECORD	Multiple-type collection of any types above

انواع داده از پیش تعریف شده (۹)

- مثال:

```
x0 <= '0';              -- bit, std_logic, or std_ulogic value '0‘

x1 <= "00011111";       -- bit_vector, std_logic_vector,
                        -- std_ulogic_vector, signed, or unsigned

x2 <= "0001_1111";      -- underscore allowed to ease visualization

x3 <= "101111“;         -- binary representation of decimal 47

x4 <= B"101111“;        -- binary representation of decimal 47

x5 <= O"57“;            -- octal representation of decimal 47
```

انواع داده از پیش تعریف شده (۱۰)

```
x6 <= X"2F"                -- hexadecimal representation of decimal 47

n <= 1200;                 -- integer

m <= 1_200;                -- integer, underscore allowed

IF ready THEN...           -- Boolean, executed if ready=TRUE

y <= 1.2E-5;               -- real, not synthesizable

q <= d after 10 ns;        -- physical, not synthesizable
```

انواع داده از پیش تعریف شده (۱۱)

- مثال: عملیات مجاز و غیرمجاز بین انواع داده‌های مختلف

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
…
a <= b(5);      -- legal (same scalar type: BIT)
b(0) <= a;      -- legal (same scalar type: BIT)
c <= d(5);      -- legal (same scalar type: STD_LOGIC)
d(0) <= c;      -- legal (same scalar type: STD_LOGIC)
a <= c;         -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d;         -- illegal (type mismatch: BIT_VECTOR x STD_LOGIC_VECTOR)
e <= b;         -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d;         -- illegal (type mismatch: INTEGER x STD_LOGIC_VECTOR)
```

انواع داده قابل تعریف توسط کاربر (۱)

- انواع داده *integer* قابل تعریف توسط کاربر:

```
TYPE integer IS RANGE -2147483647 TO +2147483647;
-- This is indeed the pre-defined type

TYPE natural IS RANGE 0 TO +2147483647;
-- This is indeed the pre-defined type NATURAL.

TYPE my_integer IS RANGE -32 TO 32;
-- A user-defined subset of integers.

TYPE student_grade IS RANGE 0 TO 100;
-- A user-defined subset of integers or naturals.
```

انواع داده قابل تعریف توسط کاربر (۲)

- انواع داده *شمارشی (enumerated)* قابل تعریف توسط کاربر:

```
TYPE bit IS ('0', '1');
-- This is indeed the pre-defined type BIT

TYPE my_logic IS ('0', '1', 'Z');
-- A user-defined subset of std_logic.

TYPE state IS (idle, forward, backward, stop);
-- An enumerated data type, typical of finite state machines.

TYPE color IS (red, green, blue, white);
-- Another enumerated data type. Encoding: “00”,“01”,“10”,“11”
```

زیرنوع‌ها (SUBTYPES) (۱)

- زیرنوع، یک نوع‌داده است که محدودیت‌هایی را داراست.
- دلیل اصلی استفاده از زیرنوع این است که عملیات بین زیرنوع و نوع پایه آن مجاز است، در حالی که عملیات بین دو نوع مختلف مجاز نیست.
- مثال: زیرنوع های زیر با توجه به انواع‌داده اسلایدهای قبل تعریف شده است:

```
SUBTYPE natural IS INTEGER RANGE 0 TO INTEGER'HIGH;
-- As expected, NATURAL is a subtype (subset) of INTEGER.

SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO 'Z';
-- Recall that STD_LOGIC=('X','0','1','Z','W','L','H','-').
-- Therefore, my_logic=('0','1','Z').
```

زیرنوع‌ها (SUBTYPES) (۲)

```
SUBTYPE my_color IS color RANGE red TO blue;
-- Since color=(red, green, blue, white), then
-- my_color=(red, green, blue).

SUBTYPE small_integer IS INTEGER RANGE -32 TO 32;
-- A subtype of INTEGER.
```

زیرنوع‌ها (SUBTYPES) (۳)

- مثال: عملیات مجاز و غیرمجاز بین نوع‌داده ها و زیرنوع ها:

```
SUBTYPE my_logic IS STD_LOGIC RANGE '0' TO '1';
SIGNAL a: BIT;
SIGNAL b: STD_LOGIC;
SIGNAL c: my_logic;
...
b <= a; --illegal (type mismatch: BIT versus STD_LOGIC)
b <= c; --legal (same "base" type: STD_LOGIC)
```

آرایه‌ها (ARRAYS) (۱)

- مجموعه داده‌های هم‌نوع.
- یک بعدی(1D)، دوبعدی(2D) یا یک بعد در یک بعد(1D*1D)
- آرایه با ابعاد بیشتر نیز می‌تواند وجود داشته باشد ولی به صورت عمومی سنتزپذیر نیستند.

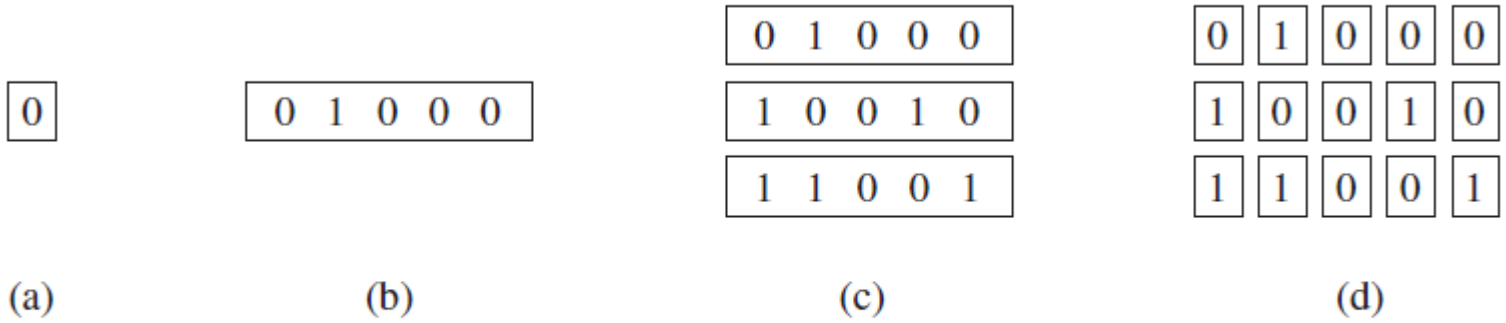


Figure 3.1
Illustration of (a) scalar, (b) 1D, (c) 1Dx1D, and (d) 2D data arrays.

آرایه‌ها (ARRAYS) (۲)

- دسته بندی انواع داده‌های از پیش تعریف شده سنتزپذیر از نظر ابعاد:
  - **اسکالر**: BIT، STD_LOGIC، STD_ULOGIC، BOOLEAN
  - **وکتور 1D**: BIT_VECTOR، STD_LOGIC_VECTOR، STD_ULOGIC_VECTOR، INTEGER، SIGNED و UNSIGNED
- ساختار تعریف یک آرایه:

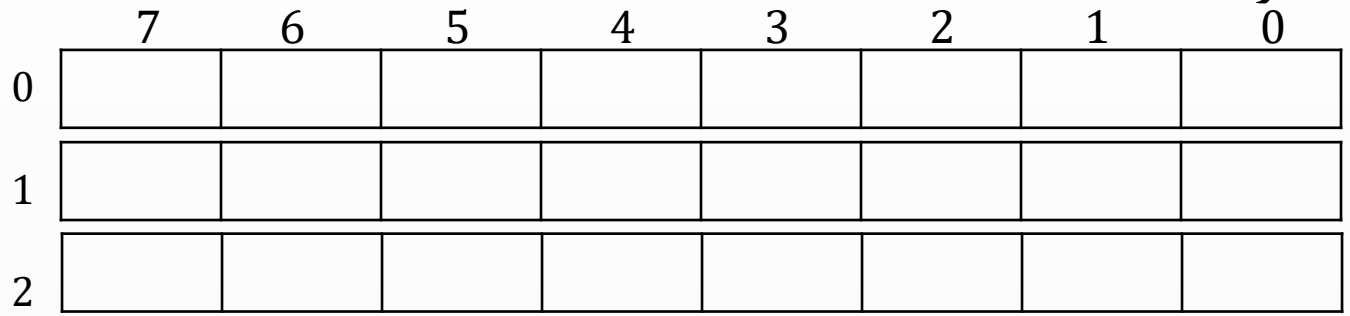
```
TYPE type_name IS ARRAY (specification) OF data_type;
```

- نحوه استفاده از نوع داده آرایه:

```
SIGNAL signal_name: type_name [:= initial_value];
```

آرایه‌ها (ARRAYS) (۳)

- مثال: یک آرایه 1D*1D




```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC; -- 1D array
TYPE matrix IS ARRAY (0 TO 3) OF row;         -- 1Dx1D array
SIGNAL x: matrix;                             -- 1Dx1D signal

TYPE matrix IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
```

آرایه‌ها (ARRAYS) (۴)

- مثال: یک آرایه 2D

	7	6	5	4	3	2	1	0
0								
1								
2								

- واحدهای سازنده این نوع آرایه، اسکالر ها هستند(به طور مثال BIT ها) در حالی که واحدهای سازنده آرایه های یک بعد در یک بعد، وکتور ها هستند.

```
TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;

-- 2D array
```

آرایه‌ها (ARRAYS) (۵)

- مثال: مقداردهی اولیه آرایه ها

```
... :="0001";                                  -- for 1D array
... :=('0','0','0','1')                       -- for 1D array
... :=(('0','1','1','1'), ('1','1','1','0'));
-- for 1Dx1D or 2D array
```

آرایه‌ها (ARRAYS) (۶)

- مثال: عملیات تخصیص(assignment) مجاز و غیرمجاز

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;                -- 1D Array
TYPE array1 IS ARRAY (0 TO 3) OF row;                       -- 1Dx1D Array
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;     -- 2D Array


SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;
```

آرایه‌ها (ARRAYS) (۷)

```
--------- Legal scalar assignments: ---------------
-- The scalar (single bit) assignments below are all legal,
-- because the "base" (scalar) type is STD_LOGIC for all signals
-- (x,y,v,w).
x(0) <= y(1)(2);          -- notice two pairs of parenthesis. (y is 1Dx1D)
x(1) <= v(2)(3);
x(2) <= w(2,1);           -- a single pair of parenthesis (w is 2D)
y(1)(1) <= x(6);
y(2)(0) <= v(0)(0);
y(0)(0) <= w(3,3);
w(1,1) <= x(7);
w(3,0) <= v(0)(3);
```

آرایه‌ها (ARRAYS) (۸)

```
--------- Vector assignments: ---------------------

x <= y(0);                    -- legal (same data types: ROW)

x <= v(1);                    -- illegal (type mismatch: ROW x STD_LOGIC_VECTOR)

x <= w(2);                    -- illegal (w must have 2D index)

x <= w(2, 2 DOWNTO 0);        -- illegal (type mismatch: ROW x STD_LOGIC)

v(0) <= w(2, 2 DOWNTO 0);     -- illegal (mismatch: STD_LOGIC_VECTOR x STD_LOGIC)

v(0) <= w(2);                 -- illegal (w must have 2D index)

y(1) <= v(3);                 -- illegal (type mismatch: ROW x STD_LOGIC_VECTOR)

y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0);   -- legal (same type, same size)

v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0); -- legal (same type, same size)

w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0)  -- illegal (type mismatch)
```

آرایه‌ها (ARRAYS) (۹)

- مثال:

```
TYPE bit_vector IS ARRAY (NATURAL RANGE <>) OF BIT;
-- This is indeed the pre-defined type BIT_VECTOR.
-- RANGE <> is used to indicate that the range is unconstrained.
-- NATURAL RANGE <>, on the other hand, indicates that the only
-- restriction is that the range must fall within the NATURAL
-- range.
```

PORT ARRAY

- چگونه پورت‌های تعریف شده در entity یک آرایه باشند؟
  - ← پس از یادگیری مفهوم package، تدریس خواهد شد!

(۱) RECORDS

- مشابه آرایه ها با این تفاوت که می‌توانند شامل اشیائی از انواع‌داده های مختلف باشند.

- مثال:

```
TYPE birthday IS RECORD
        day: INTEGER RANGE 1 TO 31;
        month: month_name;
END RECORD;
```

(۲) RECORDS

```vhdl
library ieee;
use ieee.std_logic_1164.all;

package example_record_pkg is

  -- Outputs from the FIFO.
  type t_FROM_FIFO is record
    wr_full  : std_logic;                    -- FIFO Full Flag
    rd_empty : std_logic;                    -- FIFO Empty Flag
    rd_dv    : std_logic;
    rd_data  : std_logic_vector(7 downto 0);
  end record t_FROM_FIFO;

  -- Inputs to the FIFO.
  type t_TO_FIFO is record
    wr_en    : std_logic;
    wr_data  : std_logic_vector(7 downto 0);
    rd_en    : std_logic;
  end record t_TO_FIFO;

  constant c_FROM_FIFO_INIT : t_FROM_FIFO := (wr_full => '0',
                                              rd_empty => '1',
                                              rd_dv => '0',
                                              rd_data => (others => '0'));

  constant c_TO_FIFO_INIT : t_TO_FIFO := (wr_en => '0',
                                          wr_data => (others => '0'),
                                          rd_en => '0');


end package example_record_pkg;
```

(۳) RECORDS

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.example_record_pkg.all; -- USING PACKAGE HERE!

entity example_record is
  port (
    i_clk  : in  std_logic;
    i_fifo : in  t_FROM_FIFO;
    o_fifo : out t_TO_FIFO := c_TO_FIFO_INIT  -- intialize output record
    );
end example_record;

architecture behave of example_record is

  signal r_WR_DATA : unsigned(7 downto 0) := (others => '0');

begin

  -- Handles writes to the FIFO
  p_FIFO_WR : process (i_clk) is
  begin
    if rising_edge(i_clk) then
      if i_fifo.wr_full = '0' then
        o_fifo.wr_en   <= '1';
        o_fifo.wr_data <= std_logic_vector(r_WR_DATA + 1);
      end if;
    end if;
  end process p_FIFO_WR;

  -- Handles reads from the FIFO
  p_FIFO_RD : process (i_clk) is
  begin
    if rising_edge(i_clk) then
      if i_fifo.rd_empty = '0' then
        o_fifo.rd_en <= '1';
      end if;
    end if;
  end process p_FIFO_RD;

end behave;
```

نوع‌داده های SIGNED و UNSIGNED (۱)

- در پکیج *std_logic_arith* تعریف شده اند.
- ساختار آن‌ها:

```
SIGNAL x: SIGNED (7 DOWNTO 0);

SIGNAL y: UNSIGNED (0 TO 3);
```

- UNSIGNED، یا بدون علامت، عددی است که هیچ‌وقت کوچکتر از ۰ نیست. برای مثال "0101" عدد ۵ ده‌دهی را نشان می‌دهد و "1101" عدد ۱۳ ده‌دهی را.
- SIGNED، یا علامت‌دار، عددی است که می‌تواند منفی یا مثبت باشد. برای مثال "0101" عدد ۵ ده‌دهی را نشان می‌دهد و "1101" عدد ۳- ده‌دهی را.

نوع‌داده های SIGNED و UNSIGNED (۲)

- عملیات مجاز بر روی نوع‌داده های SIGNED و UNSIGNED و STD_LOGIC_VECTRO با توجه به این که از چه پکیجی استفاده شده است:

package Name	STD_LOGIC_VECTOR	SIGNED & UNSIGNED
std_logic_1164	منطقی	---
std_logic_1164 *std_logic_unsigned* یا *std_logic_1164* *std_logic_signed*	منطقی و حسابی	---
std_logic_1164 *std_logic_arith*	---	حسابی
std_logic_1164 *numeric_std*	---	حسابی و منطقی

تبدیل انواع‌داده به یکدیگر (۱)

- VHDL اجازه عملیات مستقیم بین نوع‌داده های مختلف را نمی‌دهد.
- تبدیل انواع‌داده ای که به طور نزدیک با هم در ارتباط هستند. (*closely related*)
  - به این معنا که هر دو عملوند نوع *پایه*(*base* type) یکسانی دارند.
  - در این حالت پکیج *std_logic_1164*/*ieee* دارای توابع مستقیم برای تبدیل انواع‌داده به یکدیگر است.
- مثال:

```
TYPE long IS RANGE -100 TO 100;
TYPE short IS RANGE -10 TO 10;
SIGNAL x : short;
SIGNAL y : long;
...
y <= 2*x + 5;              -- error, type mismatch
y <= long(2*x + 5);        -- OK, result converted into type long
```

تبدیل انواع‌داده به یکدیگر (۲)

- تبدیل انواع‌داده به یکدیگر با استفاده از توابع موجود در در پکیج *std_logic_arith* از کتابخانه *ieee*:

  - **conv_integer(p) :**
    - **P:** INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC
    - **to:** INTEGER
    - Notice that STD_LOGIC_VECTOR is not included.

  - **conv_unsigned(p, b):**
    - **P:** INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC
    - **to:** UNSIGNED
    - **b:** size

تبدیل انواع‌داده به یکدیگر (۳)

- **conv_signed(p, b):**
  - **P:** INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC
  - **to:** SIGNED
  - **b:** size

- **conv_std_logic_vector(p, b):**
  - **P:** INTEGER, UNSIGNED, SIGNED, or STD_LOGIC
  - **to:** STD_LOGIC_VECTOR
  - **b:** size

تبدیل انواع‌داده به یکدیگر (۴)

- مثال: تبدیل انواع‌داده به یکدیگر:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
...
SIGNAL a: IN UNSIGNED (7 DOWNTO 0);
SIGNAL b: IN UNSIGNED (7 DOWNTO 0);
SIGNAL y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
-- Legal operation: a+b is converted from UNSIGNED to an
-- 8-bit STD_LOGIC_VECTOR value, then assigned to y.
```

تبدیل انواع‌داده به یکدیگر (۵)

- دو تبدیل نوع‌داده مهم:
  - تبدیل INTEGER به STD_LOGIC_VECTOR:

```
use ieee.numeric_std.all;
...
my_slv <= std_logic_vector(to_unsigned(my_int, my_slv'length));
```

  - تبدیل STD_LOGIC_VECTOR به INTEGER:

```
use ieee.numeric_std.all;
my_int <= to_integer(signed(my_slv));          -- for signed integer
-- or
my_int <= to_integer(unsigned(my_slv));        -- for unsigned integer
```

مثال ۱

- Single Bit Versus Bit Vector

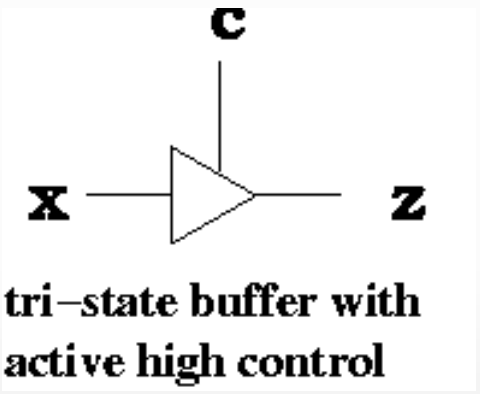
```
-----------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT;
         x: OUT BIT);
END and2;
-----------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------
```

```
-----------------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT_VECTOR (0 TO 3);
         x: OUT BIT_VECTOR (0 TO 3));
END and2;
-----------------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------------
```

مثال ۲

Tri-state Buffer ➢

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY tri_state IS
        PORT (input, ena: IN STD_LOGIC;
        output: OUT STD_LOGIC);
END ENTITY;
-----------------------------------------
ARCHITECTURE tri_state OF tri_state IS
BEGIN
        output <= input WHEN ena='1' ELSE 'Z';
END ARCHITECTURE;
```



مثال ۳

ROM ➢

```
entity ROM_design IS
        PORT (address: IN INTEGER RANGE 0 TO 7;
              output: OUT BIT_VECTOR (3 DOWNTO 0));
END ROM_design;
-------------------------------------------------------
ARCHITECTURE rom_design OF ROM_design IS
   TYPE rom IS ARRAY (0 TO 7) OF BIT_VECTOR (3 DOWNTO 0);
   CONSTANT my_rom: rom := ("1111",
                            "1110",
                            "1101",
                            "1100",
                            "1000",
                            "0111",
                            "0110",
                            "0101");
BEGIN
output <= my_rom (address);
END rom_design;
```

مطالعه بیشتر

- بخش ۳.۱۰ کتاب *Circuit Design With VHDL*:
  - مثال ۳.۱ (صفحه ۳۸)
  - مثال ۳.۳ (صفحه ۴۱)

تمرین

- بخش ۳.۱۱ کتاب *Circuit Design With VHDL*:
  - تمرین ۳.۲ (صفحه ۴۴)
  - تمرین ۳.۵ (صفحه ۴۴)

پایان بخش انواع داده