انواع داده

Data Types

انواع داده

- انواع داده در پکیج standard:
  - bit و bit_vector:
  - integer:
    - برای کارهای محاسباتی صحیح

```
type integer is range -2147483647 to 2147483647;
```

  - boolean:
    - برای کارهای منطقی و شرطها(مانند گزاره‌های منطقی، مقایسه)

```
type boolean is (false, true);
```

انواع داده

- character:

```
type CHARACTER is (
NUL, ..., CR, ..., ESC, ...
' ', '!', '"', '#', '$', '%', '&', ...,
'0', '1', '2', ...,
'A', 'B', 'C', );
```

- string:

```
type STRING is array (POSITIVE range <>) of CHARACTER;
```

  - در میان دو گیومه: “myFile.txt”

- real:
  - برای کارهای محاسباتی با اعداد حقیقی

انواع داده

- time:
  - برای توصیف تأخیر اجزای مدار، تولید شکل موج
  - دقت شبیه‌سازی: fs
  - تقسیم دو time
  - جمع و تفریق دو time
  - ضرب دو time

```
type TIME is range
implementation_defined
units
fs; -- femtosecond
ps = 1000 fs; -- picosecond
ns = 1000 ps; -- nanosecond
us = 1000 ns; -- microsecond
ms = 1000 us; -- millisecond
sec = 1000 ms; -- second
min = 60 sec; -- minute
hr = 60 min; -- hour
end units;
```

```
variable T1, T2: time := 10 ns;
...
wait for T1;
...
wait for T1 * 1.2;
...
if T2/T1 > 2 then ...
```

انواع داده

- بررسی نوع (type checking):
  - زبان VHDL: بررسی دقیق هماهنگی نوع داده‌ها
    - ← پیغام خطا
  - تبدیل نوع (casting) برای انواع با رابطهٔ نزدیک

```
signal S1: integer;
variable v1: real;

v1 := (real) s1;
```

  - real و integer
  - دو نوع آرایه که محتوای آن‌ها از یک نوعند و تعداد عناصر و محدودهٔ اندیس آن‌ها یکسان است.

انواع داده

- انواع تعریف شده توسط کاربر:

```
type OPCODE is (STA, LDA, ADD, SUB, AND, NOP, JMP, JSR);
type MODE is range 0 to 3;
type ADDRESS is bit_vector (10 downto 0);
```

- تعریف زیرنوع (subtype):

```
subtype MY_BYTE is bit_vector (31 downto 0);
subtype positive is integer range 0 to integer'high;
```

– سازگار با نوع پایه‌شان

نوع دادهٔ std_ulogic

- bit:
  - ۲ مقدار
- std_ulogic:
  - ۹ مقدار
  - در پکیج std_logic_1164

```
type   std_ulogic  is (
                     'U',      -- uninitialized
                     'X',      -- Forcing  Unknown
                     '0',      -- Forcing  0
                     '1',      -- Forcing  1
                     'Z',      -- High Impedance
                     'W',     -- Weak Unknown
                     'L',      -- Weak 0
                     'H',      -- Weak 1
                     '-',       -- don't care
);
```

نوع دادهٔ std_logic

- دو درایور همزمان برای یک سیگنال:
  - غیرمجاز در حالت معمول
    - در یک بدنهٔ همروند

```
architecture ARCH1 …
   signal S1 : std_logic;
    ⋮
begin
   S1 <= '0';
    ⋮
   S1 <= '1'; --Wrong! More than one driver for S1!
    ⋮
end architecture ARCH1;
```

نوع دادهٔ std_logic

- دو درایور همزمان برای یک سیگنال:
  - غیرمجاز در حالت معمول
    - در دو بدنهٔ همروند

```
architecture ARCH2 …
   signal S1 : std_ulogic;
    ⋮
begin
    ⋮
    AND1: ANDGATE port map(A1,B1,S1);
    ⋮
    AND2: ANDGATE port map(A2,B2,S1);
    ⋮
end architecture ARCH2;
```

نوع دادهٔ std_logic

- دو درایور همزمان برای یک سیگنال:
  - غیرمجاز در حالت معمول
    - در دو بدنهٔ همروند

```
architecture ARCH2 …
   signal S1 : std_ulogic;
    ⋮
begin
    ⋮
    process (...)
    ⋮
      S1 <= A;
    ⋮
    end process;
 process (...)
    ⋮
      S1 <= B;
    ⋮
    end process;
end architecture ARCH2;
```

نوع دادهٔ std_logic

- دو درایور همزمان برای یک سیگنال:
  - گاهی لازم است:
    - گذرگاه مشترک
    - دو پودمان که در هر لحظه یک حتماً Hi-Z است
    - مدارهای open-drain
    - wired-AND یا wired-OR
  - Resolved data types
- std_logic:
  - Subtypeی از std_ulogic
  - همان ۹ مقدار

نوع دادهٔ std_logic

- نحوهٔ resolveکردن:
  - متقارن ← خاصیت جابجایی

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

```
library IEEE;
use IEEE.std_logic_1164.all;
```

نوع دادهٔ std_logic

- std_logic_vector و std_ulogic_vector
  - همهٔ عملگرهای bit_vector در پکیج تعریف شده
  - توابع تبدیل bit به std_(u)logic و بالعکس
  - توابع تبدیل بردارهای آنها به هم

نوع دادهٔ std_logic

- عملگر AND

AND	U	X	0	1	Z	W	L	H	-
U	'U'	'U'	'0'	'U'	'U'	'U'	'0'	'U'	'U'
X	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'
0	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
1	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'X'
Z	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'
W	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'
L	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
H	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'X'
-	'U'	'X'	'0'	'X'	'X'	'X'	'0'	'X'	'X'

نوع دادهٔ std_logic

- std_ulogic و std_logic
  - ترجیح std_logic مگر آنکه:
    - سرعت شبیه‌سازی مهم باشد
    - بخواهیم عمداً چنددرایوری را گزارش دهد.

نوع دادهٔ std_logic

- زیرنوع‌های std_logic
  - X01
  - X01Z
  - UX01
  - UX01Z

signed و unsigned

- کارهای محاسباتی (جمع، مقایسه)
  - با bit_vector و std_logic_vector غیرمجاز
- کارهای رشته‌بیتی (شیفت، AND)
  - با integer غیرمجاز
- signed و unsigned
  - پکیج numeric_bit
  - پکیج numeric_std
- توابع تبدیل:
  - to_integer
  - to_signed و to_unsigned

unsigned و signed

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
  ⋮
   signal S1, S2, S3: signed;
        ⋮
           S3 <= S1 + S2;
           if (S1 < S2) then ··· ;
                ⋮
           S3 <= S1 rol  5;
                ⋮
```

std_logic_(un)signed

- دو پکیج دیگر:
  - std_logic_signed
  - std_logic_unsigned
    - عملگرهای ریاضی روی std_logic تعریف شده

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
  ⋮
   signal S1, S2, S3: std_logic_vector(31 downto 0);
       ⋮
          S3 <= S1 * S2;
           ⋮
```

آرایه‌ها

- تعریف نوع آرایه:

```
type ARR_T is array (31 downto 0) of integer;
 ⋮
 signal SA : ARR_T;
```

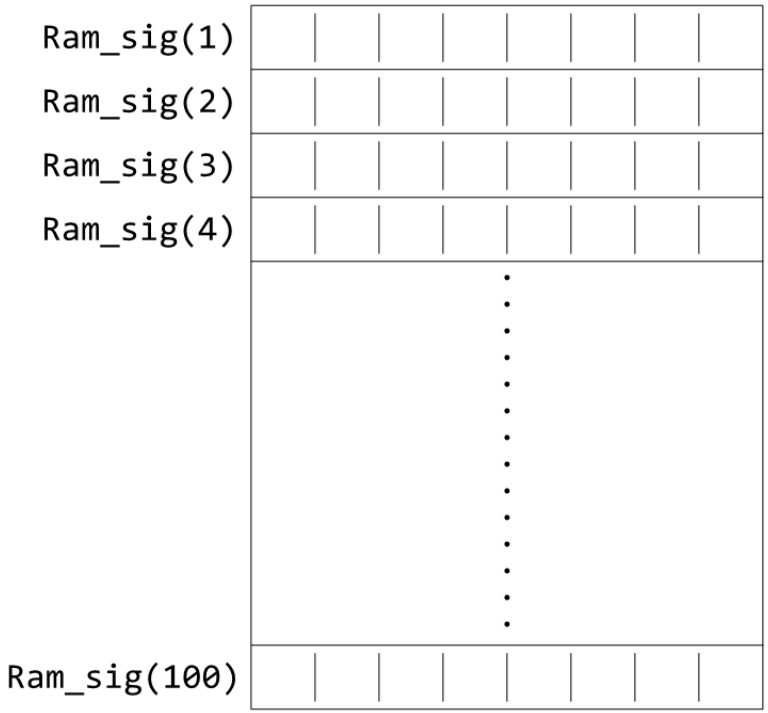
- اندیس غیر صحیح (شمارشی)

```
type BCD_COUNT is (thousand, hundred, ten, one);
type BCD_ARR is array(BCD_COUNT) of integer range 0 to
4;
   ⋮
signal S_BCD_A : BCD_ARR := (7,5,4,2);
   ⋮
begin
   ⋮
   S_BCD_A(thousand) <= 9;
   S_BCD_A(hundred to one) <= (2,3,8);
   ⋮
end … ;
```

آرایه‌ها

- آرایه چندبعدی:
  - راه اول

```
type ROW_BYTE is array (7 downto 0) of bit;
type RAM100 is array (1 to 100) of ROW_BYTE;
signal RAM_SIG : RAM100;
```



آرایه‌ها

- آرایه چندبعدی:
  - راه دوم

```
type RAM100_2D is array (1 to 100)(7 downto 0) of bit;
signal RAM_SIG2D : RAM100_2D;
```

Ram_sig2d	7	6	5	4	3	2	1	0
1								
2								
3								
4								
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
100								

آرایه‌ها

- **آرایه چندبعدی:**
  - دسترسی به یک سطر و عنصر در حالت اول

```
RAM_SIG(2) <= ('1', '1', '0', '1', '0', '0', '1', '1');
RAM_SIG(2)(5) <= '0';
```

  - دسترسی به یک سطر و عنصر در حالت دوم

```
RAM_SIG(2) <= ('1', '1', '0', '1', '0', '0', '1', '1');
RAM_SIG2D(2,5) <= '0';
```

- **سنتزکننده:**
  - معمولاً تا دو بعدی را می‌پذیرند

آرایه‌ها

- Array aggregation:

```
signal S1, S2, S3, S4 : bit;
signal S_ARR1 : bit_vector(3 downto 0);
begin
   ⋮
   (S1, S2, S3, S4) <= S_ARR1;
   S_ARR1 <= (S1, S2, S3, S4);
   (S1, S2, S3, S4) <= "1001";
   S_ARR1 <= (3 => '1', 2 downto 1 => '0', others => '1');
   ⋮
end ⋯ ;
```

```
S_ARR1 <= (others => '0');
```

رکوردها

- تعریف و استفاده از نوع رکورد:

```
type OP_CODE is (LOAD, STORE, ADD, CONVERT, SUBTRACT, JUMP);
type REGISTER is range 0 to 7;
type ADDRESS is bit_vector(6 downto 0);
type INSTRUCTION is record
   OC : OP_CODE;
   REG1 : REGISTER;
   REG2 : REGISTER;
   ADDR : ADDRESS;
end record;
   ⋮
type INST_ARR is array (0 to 1023) of INSTRUCTION;
   ⋮
   signal INST_MEMORY : INST_ARR;
   ⋮
   INST_MEMORY(1).OC <= ADD;
   INST_MEMORY(1).REG1 <= 0 ;
   INST_MEMORY(1).REG2 <= 1 ;
   INST_MEMORY(1).ADDR <= "1100000";
   ⋮
```

رکوردها

- تعریف نوع رکورد:

Instr_memory(0)	oc	reg1	reg2	addr
Instr_memory(1)				
	⋮	⋮	⋮	⋮
Instr_memory(1023)				

رکوردها

- Record aggregation:
  - Positional association
  - Named association

```
INST_MEMORY(1) <= (ADD, 0, 1, "1100000");
INST_MEMORY(1) <= (REG1 => 0, REG2 => 1, ADDR => ((6 downto 5)
=> '1', others => '0'), OC => ADD);
```