

ادامی درین

نکته: real type معادل همان float است که ۲۲ بیتی است.

SIGNAL t: TIME;

نکته: یک type هم برای زمان داریم:

t <= 1ns;

تعریف یک نوع type جدید از ۴ روش زیر انجام می شود:

enumerated -

Range -

Arrays -

Records -

روش enumerated، برای نوعهایی است که دارای مقادیر محدود هستند:

TYPE bit IS ('0', '1');

در این روش، مقدار پیش فرض، مقدار اول است.

TYPE std_logic IS ('U', '0', '1', 'X', 'Z', 'W', 'L', 'H', '-');

TYPE boolean IS (TRUE, FALSE)

نکته: در زبان VHDL، ما فقط نوع std_logic را تعریف کرده ایم و std_logic resolved نمی آید. روش دیگر تعریف، استفاده از Range است:

TYPE integer IS RANGE -2147234987 TO +2147234987

TYPE my_int IS RANGE 0 TO 31

روش دیگر تعریف، استفاده از روش آرایه برای تعریف است:

TYPE my_array IS ARRAY(0 TO 31) OF bit; (معادل bit-vector(0 TO 31))

TYPE my_little_endian_array IS ARRAY(31 TO 0) OF bit;

ENTITY example IS

مثال:

END example;

ARCHITECTURE test OF example IS

TYPE word IS ARRAY (31 DOWNTO 0) OF std_logic;

TYPE byte IS ARRAY (15 DOWNTO 0) OF std_logic;

SIGNAL x, z: word;

SIGNAL y: byte;

SIGNAL w: std_logic_vector (31 DOWNTO 0);

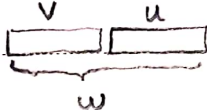
SIGNAL v: std_logic_vector (15 DOWNTO 0);

BEGIN

x(0) <= y(0); ✓

w <= x; X → VHDL is highly typed

x <= z; ✓

w <= v & u; →  : u و v با یکدیگر concat می شوند

w <= (0 => 'Z', 1 => '0', OTHERS => 'U');

w <= (OTHERS => '0');

w <= ((31 DOWNTO 24) => "11111111", OTHERS => '0');

v <= w(31 DOWNTO 16) → در طرف یک عمل assign باید دارای طول یکسان باشند

v <= '0' & v(15 DOWNTO 1); -- logical right shift

v <= v(14 DOWNTO 0) & '0'; -- logical left shift

v <= v(15) & v(15 DOWNTO 1); -- arithmetic right shift (با درون بودن علامت درستی ۱۵)

v <= v(0) & v(15 DOWNTO 1); -- rotate right

END

نکته: برای تعریف چوری از روش زیر استفاده می شود:

```
TYPE mem1024x8 IS ARRAY (0 TO 1023, 7 DOWNTO 0) OF bit;
```

در روش بالا از یک آرایه دو بعدی استفاده شده است (multidimensional array).

یک روش دیگر تعریف به صورت زیر است (Array of Array).

```
TYPE mem IS ARRAY (0 TO 1023) OF bit_vector(7 DOWNTO 0);
```

در دنیای واقعی، نوع چوری از نوع دوم است چون عموماً با `byte addressable` می باشد.

مثال:

```
TYPE mem1 IS ARRAY (0 TO 1023, 7 DOWNTO 0) OF std_logic;
```

```
TYPE mem2 IS ARRAY (0 TO 1023) OF std_logic_vector(7 DOWNTO 0);
```

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

```
SIGNAL m1 : mem1;
```

```
SIGNAL m2 : mem2;
```

```
SIGNAL s : std_logic
```

```
x <= m2(0); ✓
```

```
x <= m1(0); ✗
```

```
x <= m1(0,0); ✗
```

```
x <= m1(0, 7 DOWNTO 0); ✓
```

```
s <= m1(1,3) ✓
```

```
s <= m2(1)(3) ✓
```

```
m1 <= (OTHERS => (OTHERS => '0'));
```

```
m2 <= (OTHERS => "1111111"); 1 m2 <= (OTHERS => (OTHERS => '1'));
```

```
m2 <= (0 => "11001100", OTHERS => "00000000");
```

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY reg IS
    PORT (nrst: IN std_logic; → بهتوانست از نوع با active باشد
          clk: IN std_logic;
          din: IN std_logic_vector(15 DOWNTO 0);
          dout: OUT std_logic_vector(15 DOWNTO 0));
BEGIN
END reg;

ARCHITECTURE test OF reg IS
BEGIN
    PROCESS (clk)
    BEGIN
        IF (clk = '1') THEN
            IF (nrst = '0') THEN
                dout <= (OTHERS => '0');
            ELSE
                dout <= din;
            END IF;
        END IF;
    END PROCESS;
END

```

نکات مربوط به simulation:

- می توانیم تمامی دستورات مورد نیاز برای اجرا simulation و در قالب یک فایل با پسوند `do` ذخیره نماییم و سپس از طریق دستور `do file.do` آن را اجرا کنیم.
- می توانیم فایل مربوط به waveform را در قالب یک `do` خروجی بگیریم بدون آن که نیازی به نوشتن کد آن باشد.

ادامه‌ی درس

نکته: می‌توانیم از روش روبرو برای محدود کردن تعداد بایت‌های integer استفاده نمود و در این حال از قابلیت‌های جمع و

SUBTYPE myint IS integer RANGE 0 TO 31; (arithmetic) استفاده کنیم.

دقت بشود که اگر از روش روبرو استفاده کنیم ما یک range تعریف کرده‌ایم اما بایستی خاصیت‌های آن را هم تعریف کنیم و این را خودمان

TYPE my-range IS RANGE 0 TO 31; پیاده‌سازی کنیم.

اجزای type های روبرو به صورت مقابل تعریف شده‌اند:

TYPE integer IS RANGE ... TO ... امکان assign یک subtype به خود type وجود دارد.

SUBTYPE positive IS integer RANGE 1 TO INTEGER'HIGH

SUBTYPE natural IS integer RANGE 0 TO INTEGER'HIGH

نکته: اگر بخواهیم در حین تعریف یک type بازه‌ی آرایه‌ای آن را مشخص کنیم، از روش روبرو استفاده می‌شود:

TYPE std_logic_vector IS ARRAY (natural RANGE <>) OF std_logic

لیست بازه را محدود می‌کنند.

تعریف یوساده‌ی record

روش تعریف یک Record:

RECORD record_name IS

name: string;

id : integer;

END RECORD;

روش استفاده از تعریف بالا در یک یوساده در دسترس نیست:

SIGNAL x: record_name;

x.name <= "ALI"

نکته: در ادامه با type های string و FILE آشنا خواهیم شد.

نکته: زبان VHDL دارای استانداردهای مختلفی است که ما در این درس از استانداردهای ۲۰۰۸ استفاده می‌کنیم.

نکته: زبان VHDL یک زبان سطح بالا است و باید برای تغییر type از توابع convert استفاده کرد.
به عنوان مثال:

```
SIGNAL x : std_logic_vector(31 DOWNTO 0);
```

```
SIGNAL y : integer
```

```
y <= x; -- compiler error
```

```
y <= conv_integer(x);
```

جدول کتاب در صفحه ۱۷۹ لیست کامل این تبدیل ها را دارد.

این تبدیل تابع که در بالا انجام شده است، درست است اما معادل decoder است. (؟)

در تبدیل type که در بالا صورت گرفته است، ما 32 بیت داریم که باید به صحیح تبدیل می شود که می تواند به ایندی

یک عدد مثبت یا منفی باشد. این که تبدیل type آن را به کدام صورت در نظر می گیریم بستگی به این دارد که کدام از یک package های

std_logic_signed یا std_logic_unsigned استفاده شود. تابع بالا در درون این دو مورد تعریف شده است.

مثال طراحی یک 1024x8 static ram

```
LIBRARY ieee;
```

```
USE ieee.std_logic_unsigned.ALL;
```

```
ENTITY sram1024x8 IS
```

```
PORT (clk : IN std_logic;
```

```
      din : IN std_logic_vector(7 DOWNTO 0);
```

```
      addr : IN std_logic_vector(9 DOWNTO 0);
```

```
      dout : OUT std_logic_vector(7 DOWNTO 0));
```

```
END sram1024x8;
```

```
ARCHITECTURE behavioral OF sram1024x8 IS
```

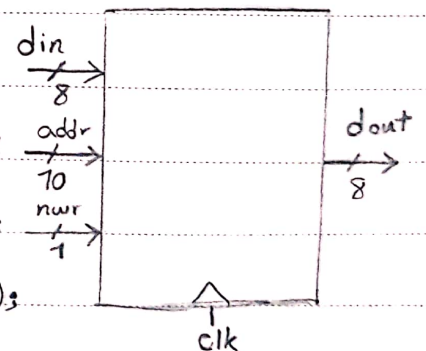
```
    TYPE buf IS ARRAY (0 TO 1023) OF std_logic_vector(7 DOWNTO 0);
```

```
    SIGNAL ram_buf : buf;
```

```
BEGIN
```

... → صفحه بعد

```
END behavioral;
```



ARCHITECTURE behavioral OF ...

ادامی کد:

BEGIN

PROCESS (clk) BEGIN

IF (clk = '1') THEN

IF (nwr = '1') THEN -- read mode

dout <= ram_buf(conv_integer(addr));

ELSE -- write mode

ram_buf(conv_integer(addr)) <= din;

END IF;

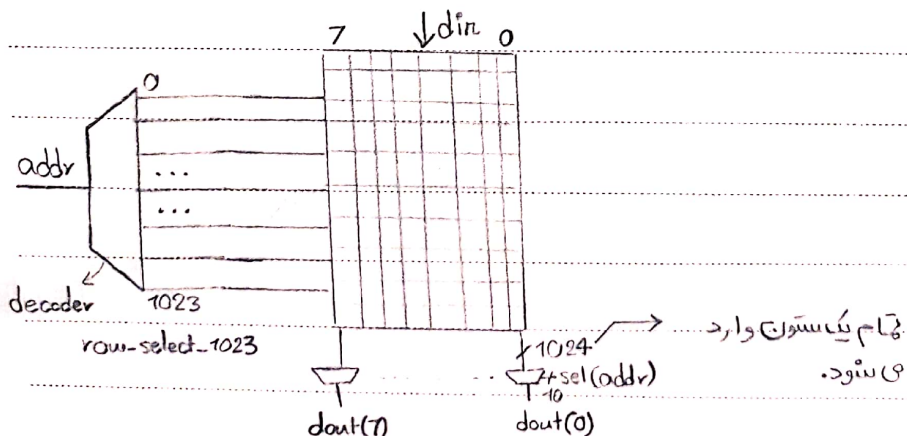
END IF;

END PROCESS;

END behavioral;

نکته: معمولاً یک برنامه‌نویس به وسیله نوع‌های `fixed`، `integer` و `std_logic_vector` می‌تواند انواع گاهای خود را توسط VHDL پیاده‌سازی کند و نیازی به دانستن تمام نوع‌ها نیست.

نکته: کد بالا به صورت زیر پیاده‌سازی خواهد شد:



عملگرها (Operators):

- assignment:

برای assign مقدار به یک سیگنال از '=' استفاده می شود.

در صورتی که بخواهیم یک مقدار به یک type بهیم از '=' استفاده می کنیم:

SIGNAL x: integer := 5;

در حالت استفاده از OTHERS، علامت '=' را هم داریم.

- logical: شامل عملگرهای NOT، AND، OR، NAND، NOR، XOR و XNOR است.

- arithmetic: شامل +، -، *، /، ** (توان)، MOD، REM و ABS است.

عملگر REM علامت @/b را خواهد گرفت و عملگر MOD علامت a/b را خواهد گرفت.

البته خروجی آن دوی توان متفاوت باشد.

- Comparison: شامل عملگرهای =، <، >، <=، >=، <> (نامساوی)، = است.

- shift: شامل SLL، SRL، SLA، SRA، ROL و ROR است.

- concatenation: عملگر & است.

- Matching: شامل =؟، <؟، >؟، <=؟ و >=؟ می باشد.

تفاوت این موارد با عملگرهای مقایسه این است که سیگنال 'L' را '0' و سیگنال 'H' را '1' در نظر می گیرد.