

## پیوست ۲

✓ نمونه فراخوانی و بکارگیری یک کتابخانه:

**LIBRARY** *library\_name*;

مثال: VHDL

```
library ieee;
library altera;
library work;
library std;
or
library ieee, altera, work, std;
```

MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.WWW.MORTEZA.9TIRG@YAHOO.COM

✓ نمونه فراخوانی و بکارگیری یک *Package* از داخل کتابخانه اش:

**USE** *library\_name.package\_name*.ALL;

مثال: VHDL

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_signed.all;
use altera.maxplus2.all;
or
use altera.maxplus2.max2lib , ieee.std_logic_1164.all ;
```

MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.MORTEZA.SHABANZADER.WWW.MORTEZA.9TIRG@YAHOO.COM

✓ فرم کلی *Entity*:

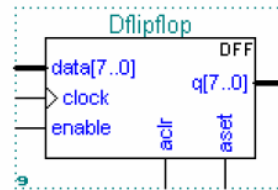
```
ENTITY entity_name IS
  GENERIC( parameter_name : string := default_value;
           parameter_name : integer := default_value);
  PORT(
    input_name , input_name           : IN          STD_LOGIC;
    input_vector_name               : IN   STD_LOGIC_VECTOR( high downto low);
    bidir_name , bidir_name         : INOUT       STD_LOGIC;
    output_name , output_name       : OUT          STD_LOGIC);
END entity_name;
```

دستور entity به منظور معرفی شکل ظاهری قطعه یا طرح از لحاظ نوع پایه های ورودی و خروجی (نه اتصالات داخلی) استفاده می شود.

عبارت مربوط به generic الزامی نیست بلکه بنا به نیاز مدار در حال طراحی و یا تشخیص برنامه نویس ، می تواند نوشته شود. مثالهای زیر را ببینید:

مثال: نمونه ای از entity یک فلیپ فلاپ نوع D. VHDL

```
Entity D_flipflop is
  Port( Data   : in std_logic_vector(7 downto 0);
        Clock  : in std_logic;
        Enable : in std_logic;
        Aclr   : in std_logic;
        Aset   : in std_logic;
        Q      : out std_logic_vector(7 downto 0));
End;
```



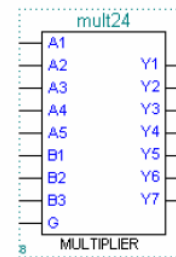
استفاده از نام entity و یا فور entity کلمه در جلوی end آن، اختیاری است.

مثال: VHDL

```
Entity generic_example is
  Generic ( delay : time := 10 ns );
  Port ( a, b : in std_logic;
        c : out std_logic );
End entity;
```

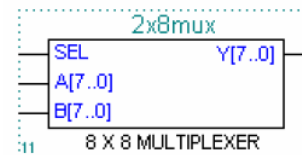
مثال: نمونه ای از entity یک Multiplier (ضرب کننده). VHDL

```
Entity multiplier is
  Port( A : in std_logic_vector ( 4 downto 0);
        B : in std_logic_vector ( 2 downto 0);
        Y : in std_logic_vector ( 6 downto 0));
End multiplier;
```



مثال: نمونه ای از entity یک Multiplexer. VHDL

```
Entity mux is
  Port(
    A : in integer range 0 to 255;
    B : in integer range 0 to 255;
    Y : in integer range 0 to 255
  );
End multiplier;
```



MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA\_STIRGEYAHOO.COM

✓ طریقه تفصیص یا انتساب سیگنالها بصورت همزمان:

signal <= expression;

سیگنال در VHDL یک سیم اتصال یا پایه قطعه را دارد .

مثال: VHDL

a <= b;

مقدار سیگنال b بطور آتی به سیگنال a داده می شود .

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA\_STIRGEYAHOO.COM

✓ فرم کلی Architecture :

```

ARCHITECTURE arch_name OF entity_name IS
    SIGNAL signal_name : STD_LOGIC;
    SIGNAL signal_name : STD_LOGIC;
BEGIN
    -- Process Statement

    -- Concurrent Procedure Call

    -- Concurrent Signal Assignment

    -- Conditional Signal Assignment

    -- Selected Signal Assignment

    -- Component Instantiation Statement

    -- Generate Statement
END arch_name;

```

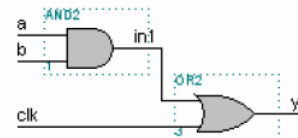
تمامی این عبارات در صورت وجود بطور همزمان اجرا می شوند

مثال: VHDL

```

Architecture arch_test of arch_example is
    Signal in1 : std_logic;      ممل تعریف اتصالات داخلی
Begin
    in1 <= a and b;              شرح دادن عملکرد مدار
    y <= in1 or clk;
end;

```



نوشتن نام architecture در جلوی end آن ، اختیاری است.

همین مدار را می توان به شکل زیر نوشت :

```

Architecture arch_test of arch_example is
    Signal in1 : std_logic;      ممل تعریف اتصالات داخلی
Begin
    y <= in1 or clk;             شرح دادن عملکرد مدار
    in1 <= a and b;
end;

```

اختلاف ظاهری این دو برنامه در جابجا نوشتن خطوط برنامه در برنه Architecture است اما نتیجه کارهیچ فرقی نمی کند و علت آن هم این است که برنه Architecture ، به شکل Concurrency یا همزمانی عمل می کند یعنی هر قلاف دیگر برنامه ها مانند C ، Pascal ، Assembly و ... که خط به خط برنامه را پیش می برند ، VHDL دارای این ویژگی منحصربه فرد است که در داخل برنه Architecture خود می تواند تمامی عبارات و دستورات را بطور همزمان و یکجا مقداردی و تحلیل کند( البته در واقعیت با یک تأخیر بسیار کوچک  $\delta$  ، زیرا اگیتهای منطقی نیز در عمل دارای کمی تأخیر(delay) می باشند ) .

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA.STIRGIVAHOO.COM

✓ فرم کلی دستور CASE:

CASE *expression* IS

```
WHEN constant_value =>
    statement;
    statement;
WHEN constant_value =>
    statement;
    statement;
    .
    .
    .
WHEN OTHERS =>
    statement;
    statement;
```

تمامی این عبارات بصورت ترتیبی (فقط به فقط) اجرا می شوند

END CASE;

دستور CASE چون یک دستور sequential است یعنی عبارات داخلی آن به ترتیب و فقط به فقط اجرا می شوند لذا حتماً می بایست داخل بدنه process نوشته شود.

مثال: VHDL

Case select is

```
When '0' => c<=a;
When '1' => c<=b;
```

End case;

اگر when others نوشته نشود در اینصورت می بایست تمامی حالت‌های ممکن عبارت (در اینجا select) ذکر شود.

مثال: VHDL

Case a is

```
When 0    => b<=3;
When 1|2  => b<=2;
When others => b<=0;
```

عملگر (|) به معنی (یا) می باشد.

End case;

مثال: VHDL

Case sel is

```
When 0          => q<=5;
When 1 to 17    => q<=7;
When 23 downto 18 => q<=2;
When others     => q<=0;
```

End case;

محدوده حالت‌های ممکن expression نباید باهم همپوشانی داشته باشند.

```

case int is
  When "000000" => a<="000";
                  b<="111";
                  c<="001";
  when "001110"  => a<="011";
  when others    => qout<="001";
end case;

```

توجه داشته باشید که تمامی این case ها داخل بدنه process نوشته می شوند.

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_3TIRG@YAHOO.COM

✓ تفصیص (انتساب) سیگنالها بصورت شرطی:

label:

```

signal <= expression WHEN boolean_expression ELSE
          expression WHEN boolean_expression ELSE
          expression;

```

```

dbus <= d when en1='1' else 'Z';
q <= a when sel='0' else b;
z <= A when sel="00" else
    B when sel="01" else
    C when sel="10" else
    D;

```

هرگز این دستور را نباید داخل دستور process بنویسید.

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_3TIRG@YAHOO.COM

✓ فرم کلی تعریف و مقداردهی اولیه Constant :

CONSTANT constant\_name : type\_name := constant\_value;

```

Constant delay : time := 5 ns;
Constant a : a_type := "1001";

```

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_3TIRG@YAHOO.COM

✓ فرم کلی تعریف سیگنال داخلی:

SIGNAL signal\_name : type\_name;

```

signal in1 : std_logic;

```

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_3TIRG@YAHOO.COM

✓ فرم کلی تعریف متغیر (variable) :

VARIABLE variable\_name : type\_name;

```

variable count, temp : integer;

```

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_3TIRG@YAHOO.COM

✓ فرم کلی تعریف *PROCESS* :

معمولاً با توجه به ساختار مدار ، می توان از یکی از دو نمونه زیر استفاده نمود:

I. *process* با لیست حساسیت (*Sensitivity List*):

*process\_label*:

*PROCESS* ( *signal\_name* , *signal\_name* , *signal\_name* )

*VARIABLE* *variable\_name* : *STD\_LOGIC*;

*VARIABLE* *variable\_name* : *STD\_LOGIC*;

*BEGIN*

-- *Signal Assignment Statement*

-- *Variable Assignment Statement*

-- *Procedure Call Statement*

-- *If Statement*

-- *Case Statement*

-- *Loop Statement*

فرد دستور *process* , *concurrent* است  
( یعنی داخل بدنه *architecture* نوشته می شود )  
اما داخل بدنه اش برعکس بدنه *architecture* ,  
بصورت *sequential* کار می کند.  
( البته حضور تأخیر  $\delta$  یا دتان ورود )

*END PROCESS* *process\_label*;

II. *process* بدون لیست حساسیت ( به کمک عبارات *wait*, *wait until*, *wait for*, *wait on* ):

*process\_label*:

*PROCESS*

*VARIABLE* *variable\_name* : *STD\_LOGIC*;

*VARIABLE* *variable\_name* : *STD\_LOGIC*;

*BEGIN*

*WAIT UNTIL* *clk\_signal* = '1';

غیر قابل سنتز

-- *Signal Assignment Statement*

-- *Variable Assignment Statement*

-- *Procedure Call Statement*

-- *If Statement*

-- *Case Statement*

-- *Loop Statement*

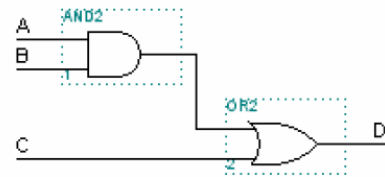
فرد دستور *process* , *concurrent* است  
( یعنی داخل بدنه *architecture* نوشته می شود )  
اما داخل بدنه اش برعکس بدنه *architecture* ,  
بصورت *sequential* کار می کند.  
( البته حضور تأخیر  $\delta$  یا دتان ورود )

*END PROCESS* *process\_label*;

می توان به فرات گفت که ۹۰ درصد کدنویسی VHDL برای مدارات سفت افزاری دیجیتال ، توسط دستور *process* نوشته می شود و این یعنی اینکه *process* یکی از پرکاربردترین و مهمترین دستورات VHDL می باشد.

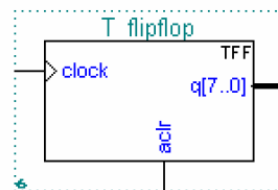
**VHDL** مثال: دستور process را می توان برای مدارهای ترکیبی (Combinational) اجرا نمود به این صورت که ورودیهای آن را در داخل لیست حساسیت قرار داد تا با کوچکترین تغییر وضعیت آنها، process این تغییر را حس کرده و خروجی موردنظر را تولید کند.

```
Process (A, B, C)
begin
    D <= (A and B) or C;
end process;
```



**VHDL** مثال: دستور process را می توان برای مدارهای ترتیبی (Sequential) نوشت به این صورت که ورودی clock و در برخی مواقع، ورودیهای preset و reset و enable را در داخل لیست حساسیت process قرار داد تا به محض تغییر وضعیت یکی از آنها، یکبار دستور process از بالا تا پایین عبارات ترتیبی داخلش را اجرا کند و خروجی موردنظر را تولید نماید.

```
process (clock, aclr)
begin
    if aclr='1' then
        q <= (others=>'0');
    elsif (clock'event and clock='1') then
        q <= q+1;
    end if;
end process;
```



عبارت  $others=>'0'$  به این معنی است که تمامی بیتهای  $q$  را صفر کن و برای سادگی کار از این فرم نوشتن استفاده شده و می توان بجای آن از فرم نسبتاً طولانی تر  $q<="00000000"$  نیز استفاده نمود.

عبارت  $clock'event$  به معنی بالا رونده بودن لبه پالس clock می باشد. در واقع جمله  $clock'event$  و جمله  $clock='1'$  به معنی بالا رونده بودن لبه پالس ساعت می باشد.

فرم کلی دستور  $if$  را می توانید در ادامه این مطلب ملاحظه نمایید.

MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH WWW.MORTEZA\_STIRGI@YAHOO.COM

✓ فرم کلی دستور  $if... then ... else$  :

```
IF expression THEN
    statement;
    statement;
ELSIF expression THEN
    statement;
    statement;
ELSE
    statement;
    statement;
END IF;
```

عبارات  $if$  بصورت ترتیبی (فقط به خط) اجرا می شوند

اگر  $if$  ساده یعنی بدون  $else$  باشد فقط یک عبارت را ارزیابی و در صورت True بودن اجرا می کند و از  $if$  خارج می شود و برنامه ادامه می یابد ولی اگر  $if...else$  داشته باشیم بین دو عبارت یکی را انتخاب می کند.



```
if sel='1' then
    q <= a;
else
    q <= b;
end if;
```

دستور if با دستور case، با وجود شباهت عملکرد اما در VHDL سه فرق اساسی دارند که بهتر است آنها را مدنظر داشته باشیم:

۱. عبارت case مشابه if می باشد و وسیله ای برای تصمیم گیری و انشعاب شرطی است ولی نتیجه شرط به شکل Boolean می باشد ( یعنی true و false ) در حالیکه در مورد case نتیجه شرط می تواند type های مختلفی از جمله integer، std\_logic\_vector، bit\_vector، enumerated و ... باشد.

۲. موقعی که case بکار برده می شود، تمامی انشعابها دارای اولویت مساوی هستند در حالیکه در if اینطور نیست.

۳. در case، شرط تست می گردد و مستقیماً یکی از انشعابها انتخاب می شود ولی در دستور if چند انشعاب است که به ترتیب اولویت، پشت سرهم تست می شوند.

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_STIRG@YAHOO.COM

✓ فرم کلی انتساب در سیگنالها:

signal\_name <= expression;

```
qout <= a;
count <= cnt + 1;
b <= "00011010";
sum <= A xor B xor C;
int <= 123;
```

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_STIRG@YAHOO.COM

✓ فرم کلی انتساب در متغیرها:

variable\_name := expression ;

```
var1 := var2;
var2 := sig2;
a := 16;
cnt := cnt + 1;
```

MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.MORTEZA SHABANZADEH.WWW.MORTEZA\_STIRG@YAHOO.COM

✓ دستور wait و انواع آن:

همانطور که در قسمت process بیان شد نوع دیگر استفاده از این دستور، نوع بدون لیست حساسیت است که بهای آن

می بایست از یکی از انواع دستور wait بهره بگیریم.

البته این دستور غیر قابل سنتز است!

```
WAIT UNTIL clk_name = '1';
WAIT ON clk_name = '1';
WAIT FOR time_value ns;
WAIT;
```



VHDL مثال: این دستور در برنه sequential نوشته می شود.

`wait until clk='1';`  $\equiv$  `wait until (clk'event and clk='1');`  $\equiv$  `wait on clk='1';`

این سه دستور معادل هم هستند و به این معنی می باشند که آنقدر منتظر می مانند تا یک لبه بالارونده برای `clk` بیاید.

Process

Begin

`wait until clk='1';`

`c <= a nor b;`

End process;

دستور `wait` به تنهایی معمولاً در انتهای عبارات `process` استفاده می شود و به مفهوم توقف کامل اجرای برنامه است در واقع `wait` به تنهایی یعنی انتظار بی پایان.

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA.SHABANZADEH.YAHOO.COM

✓ فرم کلی دستور *With ... select* :

label:

WITH *expression* SELECT

`signal <= expression WHEN constant_value,`  
`expression WHEN constant_value,`  
`expression WHEN constant_value,`  
`expression WHEN constant_value;`

VHDL مثال:

with sel select

`z <= a when "00" ,`

`b when "01" ,`

`c when "10" ,`

`d when "11";`

تمامی حالت‌های ممکن `sel` می بایست پوشش داده شود حتی اگر لازم باشد می توان از `others` نیز استفاده کرد.

VHDL مثال:

with inp select

`target <= value1 when "000" ,`

`value2 when "001" | "110" | "011" ,`

`value3 when others;`

عملگر (|) به معنی (یا) می باشد.

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA.SHABANZADEH.YAHOO.COM

✓ فرم کلی دستور *For ... loop* :

loop\_label:

FOR *index\_variable* IN *discrete\_range* LOOP

`statement;`

`statement;` }

END LOOP *loop\_label*;

عبارات داخل `for ... loop` به صورت خط به خط

اجرا می شوند برعکس عبارات `for ... generate`

که بصورت همزمان اجرا می شوند.

VHDL مثال:

for *i* in 0 to 4 loop

`if a(i)='1' then`

`q(i) <= b(i);`

`end if;`

end loop;

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA.SHABANZADEH.YAHOO.COM

✓ فرم کلی دستور *While ... loop* :

**loop\_label:**

```
WHILE boolean_expression LOOP
    statement;
    statement;
END LOOP loop_label;
```

عبارات داخل دستور *while ... loop* بصورت sequential (فقط به خط اجرا می شوند).

مثال: VHDL

lp\_test :

```
while j < 31 loop
    j <= j+1;
    c(j+1) <= a(j)+b(j);
end loop;
```

✓ فرم کلی دستور *For ... generate* :

**generate\_label:**

```
FOR index_variable IN discrete_range GENERATE
    statement;
    statement;
END GENERATE;
```

عبارات *for ... generate* بصورت همزمان اجرا می شوند برعکس عبارات داخل *for ... loop* که به صورت فقط به خط اجرا می شوند.

مثال: VHDL

```
Gen_test1 : for i in 0 to 7 generate
    Sum(i) <= a(i) xor b(i) xor c(i);
end generate;
```

✓ فرم کلی دستور *If ... generate* :

**generate\_label:**

```
IF expression GENERATE
    statement;
    statement;
END GENERATE;
```

عبارات داخل دستور *if ... generate* بصورت Concurrent (همزمان) اجرا می شوند.

مثال: VHDL

```
Gen_test2 : if i < 8 generate
    Sum(i) <= a(i) xor b(i) xor c(i) ;
End generate;
```

✓ فرم کلی تعریف آرایه یک بُعدی یا چند بُعدی:

```
TYPE type_name IS ARRAY <index_value> OF element_type;
STD_LOGIC_VECTOR( high DOWNTO low );
BIT_VECTOR( high DOWNTO low );
INTEGER RANGE low TO high ;
```

I. انواع آرایه های یک بُعدی که می توان سیگنالها و متغیرها و حتی ثابتها را از این نوع انتساب نمود.

مثال: VHDL

```
Signal a : std_logic_vector(7 downto 0);
a(7), a(6), a(5), a(4), a(3), a(2), a(1), a(0)
```



معاود است با:

مثال: VHDL

```
Type A_type is array (3 downto 0) of std_logic;
Type my_int is integer range 0 to 15;
    → Signal d: A_type;
    variable q: my_int;
```

مثال: VHDL

```
Type var is array (0 to 7) of integer;
    → Constant addr : var := (5,10,2,4,6,12,7,14);
```

II. تعریف آرایه دو بُعدی

```
TYPE array_type_name IS ARRAY ( high DOWNTO low ) OF type_name;
TYPE array_type_name IS ARRAY ( integer RANGE <> ) OF type_name;
```

مثال: VHDL

```
Type my_memory is array( 4 downto 0 ) of std_logic_vector( 2 downto 0 );
```

→ Signal RAM : my\_memory;

	2	1	0
4			
3			
2			
1			
0			

مثال: بعضی مواقع آسانتر است که در موقع تعریف نوع آرایه، ابعاد آرایه را مشخص نکنیم برای اینکه از الگوی دوم استفاده می کنیم.

VHDL

```
Type matrix is array ( integer range <> ) of integer;
    → variable mat : matrix ( 2 downto -8 ) := ( 3,5,1,4,7,9,12,14,20,18 );
```

MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH WWW.MORTEZA\_STIRING@YAHOO.COM

✓ فرم کلی نوع شمارشی Enumerated :

```
TYPE enumerated_type_name IS ( name , name , name );
```

نوع شمارشی شامل لیستی از نامها و کاراکترها است که با آن می توان مدارهای دیجیتال را متناسب با عملیاتی که انجام می دهند و یا انواع مقادیر فیزیکی که تمایل داریم بگیرند، با توجه به نیاز برنامه نویس، مدل سازی نمود.

این type بیشتر در طراحی ماشین حالت ( state machine ) استفاده می شود.

مثال: VHDL

```
Type morteza_type is ( '0', '1', 'Z' );
Type state_type is ( s0 , s1 , s2 , s3 );
Type Mano_PC is ( add , sub , shiftr , shiftr , mult , div , inc , load , store );
Type my_state is ( start , idle , waiting , run );
    → Signal msh : morteza;
    Signal state : state_type := s1 ;
    variable ALU_inputs : Mano_PC ;
```

MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH MORTEZA SHABANZADEH WWW.MORTEZA\_STIRING@YAHOO.COM

✓ فرم کلی تعریف Subtype :

SUBTYPE *subtype\_name* IS *type\_name* RANGE *low\_value* TO *high\_value*;  
SUBTYPE *array\_subtype\_name* IS *array\_type\_name*( *high\_index* DOWNTO *low\_index* );

subtype در واقع زیرمجموعه ای از type است یعنی نوع محدود شده و دارای تعداد عناصر کمتر type می باشد.

مثال: VHDL

```
Subtype my_int is integer range 0 to 511 ;
Subtype Byte is bit_vector( 7 downto 0 );
```

```
→ Signal a : my_int;
   Signal b : Byte;
```

توجه داشته باشید که type و subtype در قسمت اعلانات architecture یعنی قبل از begin تعریف می شوند.

✓ فرم کلی تعریف و بکارگیری Component :

COMPONENT *component\_name*

GENERIC( *parameter\_name* : *string* := *default\_value*;

*parameter\_name* : *integer* := *default\_value* );

} بستگی به انتخاب برنامه نویس دارد  
(اختیاری)

PORT(

*input\_name* , *input\_name* : IN STD\_LOGIC;

*bidir\_name* , *bidir\_name* : INOUT STD\_LOGIC;

*output\_name* , *output\_name* : OUT STD\_LOGIC);

END COMPONENT;

نامهای ورودی و خروجی component دقیقاً باید مشابه نامهای ورودی و خروجی entity قطعه مورد نظر باشد.  
در ضمن تعریف قطعه یا همان component باید در قسمت اعلانات architecture صورت بگیرد.

مثال: VHDL

```
component full_adder
```

```
Port ( a , b , c : in std_logic ;
```

```
Sum , carry : out std_logic );
```

```
end component;
```

مثال: VHDL

```
Component OR2
```

```
Port ( in1 , in2 : in std_logic ;
```

```
Out1 : out std_logic ) ;
```

```
End component;
```

✓ فرم کلی نمونه گیری از یک قطعه (Component Instance) :

instance\_name: component\_name

PORT MAP ( component\_port => connect\_port ,  
component\_port => connect\_port ); نوع ۱

instance\_name: component\_name

PORT MAP ( connect\_port , connect\_port ); نوع ۲

گاهی مواقع لازم است از یک قطعه یا چندین قطعه متفاوت دیگر در طرح خود استفاده کنیم لذا در این مواقع دستور component به کمک ما خواهد آمد تا بتوانیم این قطعات را به مدار اصلی خود معرفی کنیم و در نهایت به کمک دستور component instance از این قطعات استفاده کنیم.

نوع ۱ تعریف کامل تری است زیرا به کمک آن می توانیم از برخی پایه های قطعه نمونه گرفته شده استفاده نکنیم در حالی که در نوع ۲ با وجود اینکه روش اتصال سیمها (سیگنالها) به پایه های قطعه نمونه گرفته شده آسانتر است اما در این روش باید از همه پایه ها استفاده شود همچنین امکان جابجا نوشتن پایه ها از لحاظ ترتیب تعریف آنها وجود ندارد.

در ضمن component instance در بدنه architecture نوشته می شود.

مثال: VHDL

u1 : full\_adder

Port map ( A(0)=>a , B(0)=>b ,  
Cin=>ci ,  
SUM(0)=>sum,  
CARRY(0)=>cout );

u2 : full\_adder

Port map ( A(1)=>a , B(1)=>b ,  
CARRY(0)=>ci ,  
SUM(1)=>sum,  
CARRY(1)=>cout );

u3 : full\_adder

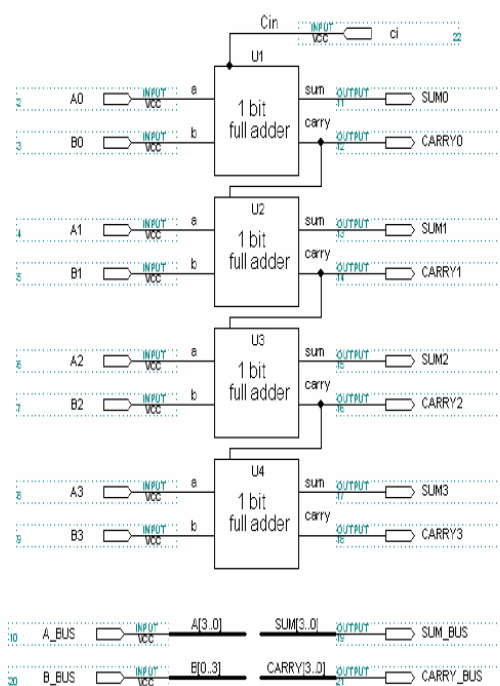
Port map ( A(2)=>a , B(2)=>b ,  
CARRY(1)=>ci ,  
SUM(2)=>sum,  
CARRY(2)=>cout );

u4 : full\_adder

Port map ( A(3)=>a , B(3)=>b ,  
CARRY(2)=>ci ,  
SUM(3)=>sum,  
CARRY(3)=>cout );

اگر بعنوان مثال بخواهیم برای نمونه برداری از u3 از نوع ۲ دستور port map استفاده کنیم ، بدین شکل باید عمل کرد:

u3 : full\_adder port map ( A(2) , B(2) , CARRY(1) , SUM(2) , CARRY(2) );



✓ فرم کلی تعریف و بکارگیری *Package* :

**PACKAGE** *pack\_name* IS

• *Pack\_declarations* → می تواند شامل تعریف *type* ها ، *subtype* ها ، *component* ها ،  
• *signal* ها و *constant* ها می باشد

**END** *pack\_name* ;

**PACKAGE BODY** *pack\_name* IS

• *Pack\_declarations* اگر در *package* فقط *type* و *constant* داشتیم  
• دیگر نیازی به نوشتن *package body* نیست.

**END PACKAGE BODY** *pack\_name* ;

مثال: VHDL

```
package my_int is
    type small_int is integer range 0 to 15;
end package;
```

ما می خواهیم از این *type* که در داخل *package* ای به اسم *my\_int* است و این *package* بطور اتوماتیک در داخل کتابخانه *work* (کتابخانه *work* یک کتابخانه پیش فرض برای قرارگیری اتوماتیک تمامی برنامه های کاربر می باشد) قرار گرفته در برنامه دیگری به اسم *smaller\_adder* استفاده کنیم پس اینطور می نویسیم :

```
use work.my_int.all;
entity smaller_adder is
port ( a , b : in small_int ;
      s : out small_int );
end;
```

مثال: در اینجا قصد داریم دو تابع منطقی **XOR** و **NAND** دو ورودی را در داخل یک **package** با نام **logical\_pack** قرار دهیم: VHDL

```
Library ieee;
use ieee.std_logic_1164.all;
Package logical_pack is
    component NAND2
        port ( in1 , in2 : in std_logic;
              out1 : out std_logic );
    end component;
    component XOR2
        port ( in1 , in2 : in std_logic;
              out1 : out std_logic );
    end component;
end package;
```

```
Library ieee;
use ieee.std_logic_1164.all;
Package body logical_pack is
    Entity NAND2
        port ( in1 , in2 : in std_logic;
```



```

        out1 : out std_logic );
end NAND2;
Architecture nand2arch of NAND2 is
begin
    out1 <= in1 nand in2;
end nand2arch;
Entity XOR2
port ( in1 , in2 : in std_logic;
      out1 : out std_logic );
end XOR2;
Architecture xor2arch of XOR2 is
begin
    out1 <= in1 xor in2;
end xor2arch;
end package body;

```

این package ای که تعریف نموده ایم را باید compile کرده و در کتابخانه مورد نظر خودمان با هر نامی که دوست داریم مثلاً `morteza_lib` قرار دهیم تا هر موقع و در هر برنامه دیگری که به آن نیاز پیدا کردیم به کمک دستورات فراخوانی `library` و `use` از آنها استفاده کنیم.

مانند:

```

Library ieee,morteza_lib;
use ieee.std_logic_1164.all;
use morteza_lib.logical_pack.all;

```

MORTEZA.SHAHANZADEH.MORTEZA.SHAHANZADEH.MORTEZA.SHAHANZADEH.MORTEZA.SHAHANZADEH.MORTEZA.SHAHANZADEH.WWW.MORTEZA.9TIRG@YAHOO.COM

✓ فرم کلی تعریف و فراخوانی تابع ( *Function* ):

```

FUNCTION func_name ( func_inputs : in inputs_type ) RETURN output_var_type IS
    VARIABLE var_name : var_type ;
    .
    .
    .
BEGIN
    .
    .
    .
    Output_var := statement ;
    RETURN output ;
END func_name ;

```

چون، تابع هیپگاه فروبی سیگنال نمی دهد بلکه باید  
هتماً از یک متغیر تعریف شده داخل خود بعنوان  
فروبی برگشتی استفاده کند.

تابع در architecture و یا package اعلان می شود.

هر تابع برعکس procedure، فقط و فقط یک فروبی دارد.

مثال: VHDL

```

function analysis ( value , max , min : integer ) return integer is
begin
    if value > max then return max;
    elsif value < min then return min;
    else return value;
    end if;
end function;

```

فراخوانی تابع می تواند در جملات تفصیص ( انتساب ) متغیر و سیگنال نیز صورت بگیرد مانند :

```

var1 := analysis ( current_temperature+increment , 10 , 100 ) ;
cout <= carry ( a , b , c ) ;

```



```
function carry ( bit1 , bit2 , bit3 : in std_logic ) return std_logic is
    variable result : std_logic ;
begin
    result := ( bit1 and bit2 ) or ( bit1 and bit3 ) or ( bit2 and bit3 ) ;
    return result ;
end carry ;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA\_STIRGI@YAHOO.COM

✓ فرم کلی تعریف و فراخوانی Procedure :

```
PROCEDURE procedure_name ( procedure_inputs : in inputs_type ;
                           procedure_outputs : out outputs_type ) IS
    VARIABLE var_name : var_type ;
    VARIABLE var_name : var_type ;
    .
    .
    .
BEGIN
    .
    .
    .
    Output_var := statement ;
    Output_var := statement ;
END PROCEDURE procedure_name ;
```

Procedure همانند function یک برنامه فرعی برای انجام محاسبات و routin های تکراری به کار می رود.  
 فرق Procedure با function در این است که Procedure می تواند هر تعداد خروجی که بخواهیم داشته باشد در حالیکه تابع فقط مجاز به برگرداندن یک خروجی می باشد همچنین Procedure داده هایی از نوع inout نیز داشته باشد یعنی اطلاعات وارد procedure شود سپس عملیاتی روی آن انجام گردد و در نهایت به خروجی فرستاده شود.

```
procedure full_adder4 ( a , b : in std_logic_vector(3 downto 0);
                      result : out std_logic_vector(3 downto 0);
                      overflow : out boolean ) is
    variable sum : std_logic_vector(3 downto 0);
    variable carry : bit := '0';
begin
    for i in 0 to 3 loop
        sum(i) := a(i) xor b(i) xor carry ;
        carry := ( a(i) and b(i) ) or ( carry and ( a(i) or b(i) ) ;
    end loop;
    result := sum ;
    overflow := carry = '1' ;
end procedure ;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA\_STIRGI@YAHOO.COM

✓ در اینجا فرم کلی کد نویسی یک شمارنده (Counter) را ملاحظه می کنید :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY entity_name IS
```

```
PORT
```

```
(
```

```
    data_input_name      : IN  INTEGER RANGE 0 TO count_value;
    clk_input_name       : IN  STD_LOGIC;
    clrn_input_name      : IN  STD_LOGIC;
    ena_input_name       : IN  STD_LOGIC;
    ld_input_name        : IN  STD_LOGIC;
    count_output_name    : OUT INTEGER RANGE 0 TO count_value
```

```
);
```

```
END entity_name;
```

```
ARCHITECTURE arch_name OF entity_name IS
```

```
    SIGNAL count_signal_name : INTEGER RANGE 0 TO count_value;
```

```
BEGIN
```

```
    PROCESS ( clk_input_name , clrn_input_name )
```

```
    BEGIN
```

```
        IF clrn_input_name = '0' THEN
```

```
            count_signal_name <= 0;
```

```
        ELSIF ( clk_input_name'EVENT AND clk_input_name = '1' ) THEN
```

```
            IF ld_input_name = '1' THEN
```

```
                count_signal_name <= data_input_name;
```

```
            ELSE
```

```
                IF ena_input_name = '1' THEN
```

```
                    count_signal_name <= count_signal_name + 1;
```

```
                ELSE
```

```
                    count_signal_name <= count_signal_name;
```

```
                END IF;
```

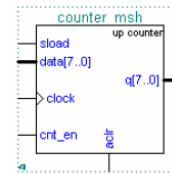
```
            END IF;
```

```
        END IF;
```

```
    END PROCESS;
```

```
    count_output_name <= count_signal_name;
```

```
END arch_name;
```



معروف به Asynchronous reset  
چون قبل از دستور بررسی کردن لبه *clk* نوشته شده  
است یعنی مستقل از آمدن *clk* مدار را *reset* می کند.

✓ در اینجا فرم کلی که نویسی یک *Flip Flop* را ملاحظه می کنید :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY entity_name IS
```

```
PORT
```

```
(
    d_input_name      : IN  STD_LOGIC;
    clk_input_name    : IN  STD_LOGIC;
    clrn_input_name   : IN  STD_LOGIC;
    ena_input_name    : IN  STD_LOGIC;
    q_output_name     : OUT STD_LOGIC
);
```

```
END entity_name;
```

```
ARCHITECTURE arch_name OF entity_name IS
```

```
SIGNAL    q_signal_name : STD_LOGIC;
```

```
BEGIN
```

```
PROCESS ( clk_input_name , clrn_input_name )
BEGIN
```

```
IF clrn_input_name = '0' THEN
```

```
    q_signal_name <= '0';
```

```
ELSIF ( clk_input_name'EVENT AND clk_input_name = '1' ) THEN
```

```
    IF ena_input_name = '1' THEN
```

```
        q_signal_name <= d_input_name;
```

```
    ELSE
```

```
        q_signal_name <= q_signal_name;
```

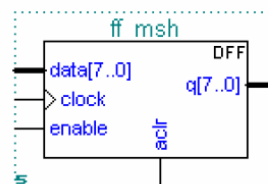
```
    END IF;
```

```
END IF;
```

```
END PROCESS;
```

```
q_output_name <= q_signal_name;
```

```
END arch_name;
```



معروف به Asynchronous reset  
چون قبل از دستور بررسی کردن لبه *clk* نوشته شده  
است یعنی مستقل از آمدن *clk* مدار را *reset* می کند.

✓ فرم کلی کدنویسی ماشین حالت با *Asynchronous Reset* :

```
ENTITY machine_name IS
    PORT(
        clk                : IN    STD_LOGIC;
        reset               : IN    STD_LOGIC;
        input_name , input_name : IN  STD_LOGIC;
        output_name , output_name : OUT STD_LOGIC);
END machine_name;
```

```
ARCHITECTURE arch_name OF machine_name IS
    TYPE STATE_TYPE IS ( state_name , state_name , state_name );
    SIGNAL state : STATE_TYPE;
```

BEGIN

PROCESS (clk)

BEGIN

IF reset = '1' THEN

state <= state\_name;

ELSIF clk'EVENT AND clk = '1' THEN

CASE state IS

WHEN state\_name =>

IF condition THEN

state <= state\_name

END IF;

WHEN state\_name =>

IF condition THEN

state <= state\_name;

END IF;

WHEN state\_name =>

IF condition THEN

state <= state\_name;

END IF;

END CASE;

END IF;

END PROCESS;

WITH state SELECT

output\_name <= output\_value WHEN state\_name,

output\_value WHEN state\_name,

output\_value WHEN state\_name;

END arch\_name;

معروف به *Asynchronous reset*

چون قبل از دستور بررسی کردن لبه clk، نوشته شده

است یعنی مستقل از آمدن clk مدار را reset می کند.

اگر این دستور را بعد از دستور آمدن لبه clk بنویسید در

این حالت گفته می شود *Synchronous reset*

زیرا باید '1' شدن reset با آمدن لبه clk همزمان

صورت بگیرد.

✓ در اینجا فرم کلی که نویسی یک *Tri State Buffer* را، ملاحظه می کنید :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
ENTITY entity_name IS
```

```
    PORT
```

```
    (
        oe_input_name      : IN  STD_LOGIC;
        data_input_name    : IN  STD_LOGIC;
        tri_output_name    : OUT STD_LOGIC
    );
```

```
END entity_name;
```

```
ARCHITECTURE arch_name OF entity_name IS
BEGIN
```

```
    PROCESS ( oe_input_name , data_input_name )
    BEGIN
```

```
        IF oe_input_name = '0' THEN
```

```
            tri_output_name <= 'Z';
```

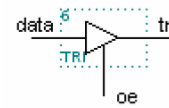
```
        ELSE
```

```
            tri_output_name <= data_input_name;
```

```
        END IF;
```

```
    END PROCESS;
```

```
END arch_name;
```



برای مدارهای ترکیبی (Combinational) نیز می توان از دستور *process* استفاده نمود زیرا با این کار مدار را نسبت به کوچکترین تغییرات ورودیها حساس می کنیم.