به نام مهربان ترینِ مهربان ها

(با نام او بخوان ، با تعلیمش بدان ، با آیاتش بفهم)
بخوان به نام پروردگارت که بیافرید همه آفریدگان را، (۱)
آدمی را از خونی بسته آفرید. (۲)
بخوان ، و پروردگار تو بزرگوارترین [بخشندگان] است، (۳)
آن که [نوشتنِ] با قلم را [به آدمی] بیاموخت. (۴)
آموخت آدمی را آنچه نمی دانست. (۵)
علق(۵-۱)

مجموعه دستورات زبان توصیف سخت افزار
مدارهای مجتمع سرعت بالا
VHDL

(Very high speed integrated circuit Hardware Description Language)

و اوست که برای شما ستارگان را آفرید تا بدانها در تاریکیهای خشکی و دریا راه یابید،
ما نشانه ها[ی خود را] را برای گروهی که می دانند به تفصیل بیان کرده ایم.
انعام۹۷

تهیه و تنظیم

مرتضی شعبان زاده
Morteza Shabanzadeh
morteza_3tir61@yahoo.com

در سالهای قبل از ۱۹۸۶ زبانهای توصیف سخت افزار متنوعی مانند ABEL و PAL ASM و ... توسط شرکتهای مختلف برای برنامه ریزی PAL,PLA,PLD وجود داشت که کاربران به شکل سلیقه ای با آنها کار می کردند یعنی این زبانهای برنامه نویسی طرفداران مخصوص به خود را داشتند و یک قالب جامع و استانداردی برای آنها در نظر گرفته نشده بود امّا در سال ۱۹۸۰ وزارت دفاع امریکا با همکاری IEEE با هدف طراحی یک زبان جدید استاندارد و فراگیر برای توصیف مدارهای دیجیتال و توسعه در مدارات مجتمع پُرسرعت (FPGA , CPLD) و همچنین برای انتقال اطلاعات سیستمهای دیجیتالی از شرکتی به شرکت و یا به کشور دیگر را ، به سه شرکت قدرتمند IBM , Texas Instrument , Intermetrics سپرد تا شش سال بعد یعنی در سال ۱۹۸۶ اولین نسخه استاندارد و تایید شده آن به بازار عرضه شود (یعنی همان نسخه VHDL86) و نسخه بعدی آن یعنی VHDL93 در سال ۱۹۹۴ به بازار آمد که از آن زمان تا به امروز این نسخه مورد استفاده کاربران قرار گرفته است و این در حالیست که هرچند سال یکبار اصلاحات جزئی در آن صورت می گیرد.

✓ در اینجا یک نمای کلی از سلسله مراتب کدنویسی VHDL را ملاحظه می کنید:

Context Clauses

Library Clause

Use Clause

Library Units

Package Declaration (optional)

Package Body (optional)

Entity Declaration

Architecture Body

✓ جدول تقسیمات دستورات VHDL از لحاظ ترتیبی(Sequential) یا همزمانی (Concurrency) بودن آنها.

Concurrent VHDL	Sequential VHDL	Con. & Seq.
Process	If then else	Signal assignment
When else	Case	Type & Constant declaration
With select	Variable declaration	Function & Procedure Call
Signal declaration	Loop statement	Assert statement
Block statement	Variable assignment	Signal attribute
	Return	
	Null	
	Wait	

✓ Y_chart :

یکی از معروفترین نمایشهایی است که نگرشهای متفاوت و سلسله مراتبی مراحل طراحی یک سیستم دیجیتالی را نشان می دهد:
فرض کنید در یکی از بخشهای شرکت DSP مسئول طراحی نسل جدیدی از تراشه های پردازش سیگنال دیجیتالی هستیم. با توّجه به هزینه های ساخت و محدوده زمانی که تراشه بایستی وارد بازار شود تا بتواند قابل رقابت باشد ، می خواهیم قبل از اقدام به ساخت بررسی کنیم که آیا این تراشه می تواند کاربردهای موردنظر را تأمین کند یا خیر. بر اساس روند طراحی Y_chart می دانیم که رفتار تراشه درسطوح مختلف طراحی از جمله سطح عملیاتی ، RTL ، سطح منطقی و ... قابل توصیف به زبان VHDL است.

در مرحله اوّل توصیف رفتاری(behavioral) که شبیه سازی بر پایه آن می تواند صحّت عملکرد تراشه را نشان دهد، ضروری است.

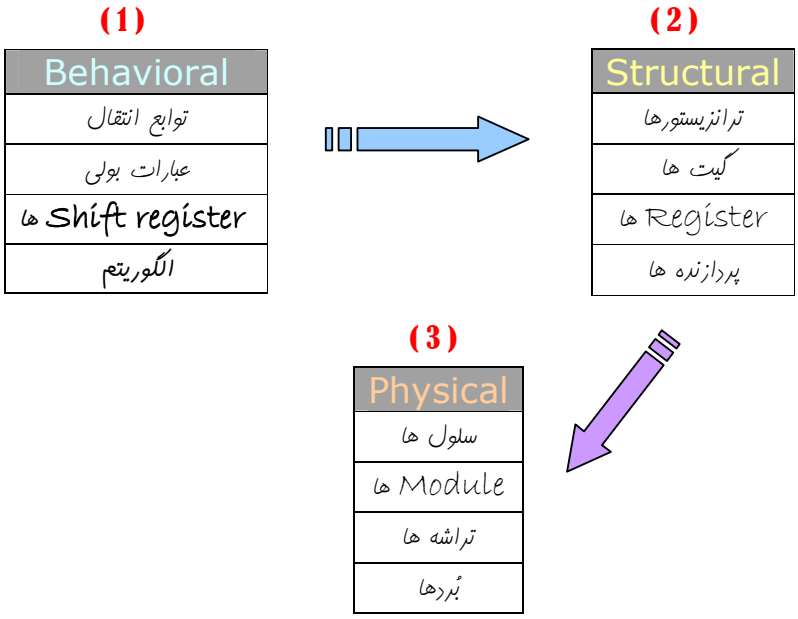
عملکرد تراشه را می توان در سایر سطوح طراحی نیز بررسی و شبیه سازی نمود، مزیت چنین رویکردی در این است که می توانیم ارزیابی را مستقل از روشهای پیاده سازی فیزیکی(physical) انجام دهیم.

پس از بررسی عملکرد ، می توانیم طرح را به یک توصیف ساختاری (structural) متشکل از واحدهای اصلی تراشه مانند memory، register وALU تبدیل نمائیم. باردیگر به کمک شبیه سازی می توان مطمئن شد که طرح ساختاری این DSP به وسیله واحدهای انتخاب شده ، عملیات دلخواه را به درستی انجام می دهد.

همانطور که در شکل می بینید توصیف ساختاری نیز می تواند با درجات متفاوتی از جزئیات ایجاد شود.

این توصیف را می توان آنقدر تکمیل کرد تا به یک توصیف فیزیکی (physical) دست پیدا کنیم که درنهایت مشخصات ساخت را از آن استخراج نمائیم .

تمامی این مراحل به کمک زبان توصیف سخت افزارVHDL و ابزارهای برنامه ریزی FPGA به سادگی امّا با حوصله و پشتکار زیاد ، امکان پذیر است.



Y_ chart

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ نحوه فراخوانی و بکارگیری یک کتابخانه:

LIBRARY library_name;

VHDL مثال:

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

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ نحوه فراخوانی و بکارگیری یک Package از داخل کتابخانه اش:

USE library_name.*package_name*.ALL;

VHDL مثال:

```
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.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ فرم کُلی 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;
```

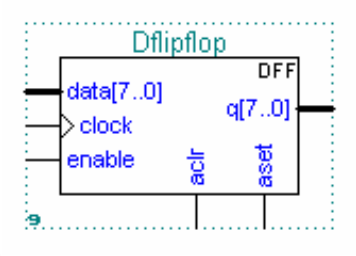
دستور entity به منظور معرفی شکل ظاهری قطعه یا طرح از لحاظ نوع پایه های ورودی و خروجی (نه اتصالات داخلی) استفاده می شود.

عبارت مربوط به generic الزامی نیست بلکه بنا به نیاز مدارِ در حال طراحی و یا تشخیص برنامه نویس ، می تواند نوشته شود.

مثالهای زیر را ببینید:

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

```
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;
```



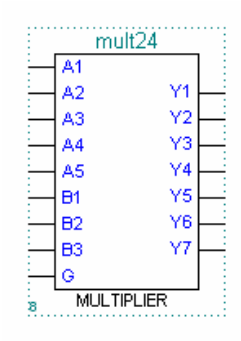
استفاده از نام entity و یا خود entity کلمه در جلوی end آن ، اختیاری است.

VHDL مثال:

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

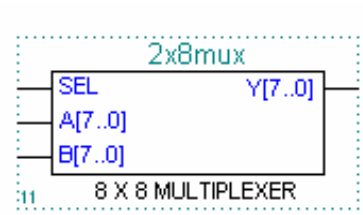
VHDL مثال: نمونه ای از entity یک Multiplier (ضرب کننده).

```
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;
```



VHDL مثال: نمونه ای از entity یک Multiplexer .

```
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.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ طریقه تخصیص یا انتساب سیگنالها بصورت همزمان:

signal <= *expression*;

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

VHDL مثال:

```
a <= b;
```

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

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

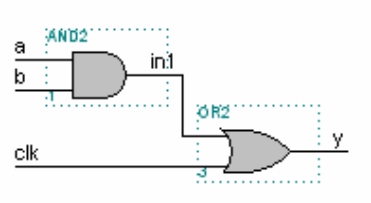
✓ فرم کلی *Architecture* :

```
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

VHDL مثال:

```
Architecture arch_test of arch_example is
 Signal in1 : std_logic;          محل 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;          محل 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 بدنه Architecture است امّا نتیجه کارهیچ فرقی نمی کند و علّت آن هم این است که بدنه Architecture ، به شکل Concurrency یا همزمانی عمل می کند یعنی بر خلاف دیگر برنامه ها مانند C ، Pascal ، Assembly و ... که خط به خط برنامه را پیش می برند ، VHDL دارای این ویژگی منحصر به فرد است که در داخل بدنه Architecture خود می تواند تمامی عبارات و دستورات را بطور همزمان و یکجا مقداردهی و تحلیل کند(البته در واقعیت با یک تأخیر بسیار کوچک δ ، زیرا گیتهای منطقی نیز در عمل دارای کمی تأخیر(delay) می باشند) .

✓ فرم کُلی دستور case:

```
CASE expression IS

    WHEN   constant_value =>
        statement;
        statement;
    WHEN   constant_value =>
        statement;
        statement;
          .
          .
          .

    WHEN OTHERS =>
        statement;
        statement;

END CASE;
```

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

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

VHDL مثال:

```
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.

VHDL مثال:

```
Case a is
    When 0      => b<=3;
    When 1|2    => b<=2;
    When others => b<=0;
End case;
```

عملگر(|) به معنی (یا) می باشد.

VHDL مثال:

```
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.

مثال: VHDL

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

توجّه 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_3TIRGI@YAHOO.COM

✓ تخصیص (انتساب) سیگنالها بصورت شرطی:

label:

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

مثال: VHDL

```
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_3TIRGI@YAHOO.COM

✓ فرم کلی تعریف و مقداردهی اوّلیه *Constant* :

CONSTANT constant_name **:** **type_name := constant_value;**

مثال: VHDL

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

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ فرم کلی تعریف سیگنال داخلی:

SIGNAL signal_name **:** type_name**;**

مثال: VHDL

```
signal in1 : std_logic;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ فرم کلی تعریف متغیر (*variable*) :

VARIABLE variable_name **:** type_name**;**

مثال: VHDL

```
variable count , temp : integer;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@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 حساسیّت(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
END PROCESS   process_label;
```

خود دستور process ، concurrent است
(یعنی داخل بدنه architecture نوشته می شود)
امّا داخل بدنه اش برعکس بدنه architecture ،
بصورت sequential کار می کند.
(البته حضور تأخیر δ یادتان نرود)

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

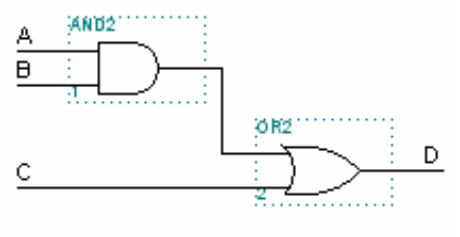
```
process_label:
 PROCESS
        VARIABLE   variable_name : STD_LOGIC;
        VARIABLE   variable_name : STD_LOGIC;
    BEGIN
        WAIT UNTIL   clk_signal = '1';          غیرقابل سنتز
        -- Signal Assignment Statement
        -- Variable Assignment Statement
        -- Procedure Call Statement
        -- If  Statement
        -- Case Statement
        -- Loop Statement
END PROCESS  process_label;
```

خود دستور process ، concurrent است
(یعنی داخل بدنه architecture نوشته می شود)
امّا داخل بدنه اش برعکس بدنه architecture ،
بصورت sequential کار می کند.
(البته حضور تأخیر δ یادتان نرود)

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

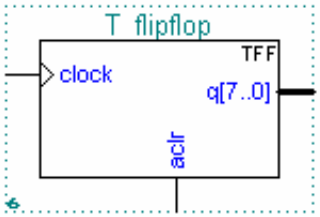
VHDL مثال: دستور process را می توان برای مدارهای ترکیبی (Combinational) اجرا نمود به این صورت که ورودیهای آن را در داخل لیست حساسیت قرار داد تا با کوچکترین تغییر وضعیت آنها ، process این تغییر را حس کرده و خروجی موردنظر را تولید کند.

```
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 enable و reset و preset 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 and clock='1' به معنی بالا رونده بودن لبه پالس clock می باشد. در واقع جمله clock'event به مفهوم تغییر در لبه پالس clock می باشد و جمله clock='1' به مفهوم بالارونده بودن لبه پالس ساعت می باشد.

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

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(خط به خط) اجرا می شوند

اگر if ساده یعنی بدون else باشد فقط یک عبارت را ارزیابی و درصورت 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.

VHDL مثال:

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

دستور if با دستورcase ، با وجود شباهت عملکرد امّا در VHDL سه فرق اساسی دارند که بهتر است آنها رامدنظر داشته باشیم:

۱. عبارت case مشابه if می باشد و وسیله ای برای تصمیم گیری وانشعاب 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 ، enumerated ، bit_vector ، std_logic_vector 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_3TIRGI@YAHOO.COM

✓ فرم کلی انتساب در سیگنالها:

signal_name <= ***expression***;

VHDL مثال:

```
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_3TIRGI@YAHOO.COM

✓ فرم کلی انتساب در متغیرها:

variable_name := ***expression*** ;

VHDL مثال:

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

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@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 بجای 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';  ≡  wait until  ( clk'event and clk='1' );  ≡  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 توّقف کامل اجرای برنامه است در واقع waitبه تنهایی یعنی انتظار بی پایان.

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@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 حتّی 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_3TIRGI@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_3TIRGI@YAHOO.COM

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@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 (خط 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;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ 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 شوندبرعکس عبارات داخل for ... loop که به صورت خط 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;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ 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;
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ 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 انتخاب 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)
```

معادل 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: VHDL بعضی مواقع آسانتر است که در موقع تعریف نوع آرایه ، ابعاد آرایه را مشخص نکنیم برای اینکار از الگوی دوّم استفاده می کنیم.

```
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_3TIRGI@YAHOO.COM

✓ فرم کلی نوع شمارشی *Enumerated* :

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

نوع شمارشی شامل لیستی از نامها وکاراکترها است که با آن می توان مدارهای دیجیتال را متناسب با عملیاتی که انجام می دهند و یا انواع مقادیر خروجی که تمایل داریم بگیرند ، با توجّه به نیاز برنامه نویس ، مدلسازی 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 , shiftl , 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_3TIRGI@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 .

VHDL مثال:

```
Subtype my_int is integer range 0 to 511 ;
Subtype Byte is bit_vector( 7 downto 0 );
    ⟶      Signal a : my_int;
            Signal b : Byte;
```

توّجه 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 componentدقیقاً باید مشابه نامهای ورودی و خروجی entity قطعه موردنظر باشد.
در ضمن تعریف قطعه یا همان component باید در قسمت اعلانات architecture صورت بگیرد.

VHDL مثال:

```
component full_adder
                Port ( a , b , c : in std_logic ;
                       Sum , carry : out std_logic );
end component;
```

VHDL مثال:

```
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

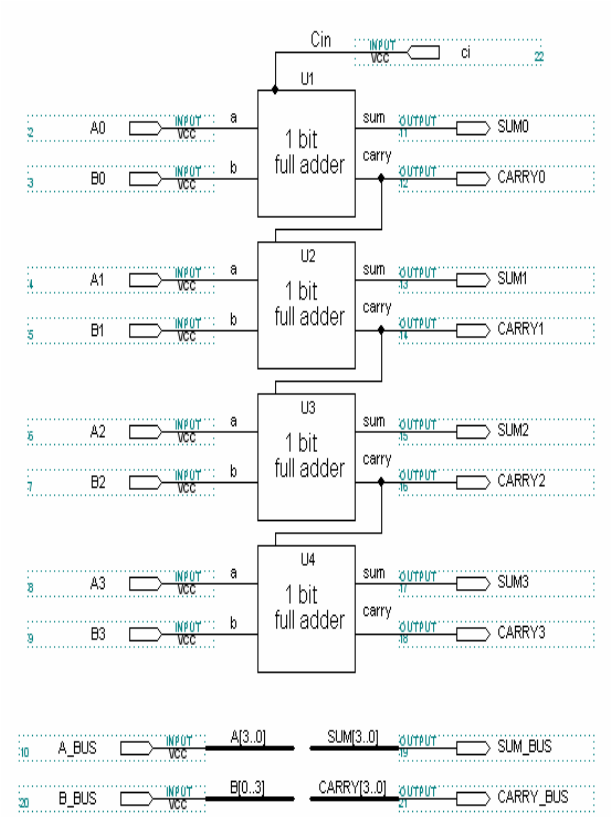
گاهی مواقع لازم است از یک قطعه یا چندین قطعه متفاوت دیگر در طرح خود استفاده کنیم لذا در این مواقع دستور component به کمک ما خواهد آمد تا بتوانیم این قطعات را به مدار اصلی خود معرفی کنیم و در نهایت به کمک دستور component instance از این قطعات استفاده کنیم .

نوع ۱ تعریف کامل تری است زیرا به کمک آن می توانیم از برخی پایه های قطعه نمونه گرفته شده استفاده نکنیم در حالیکه در نوع۲ با وجود اینکه روش اتصال سیمها (سیگنالها) به پایه های قطعه نمونه گرفته شده آسانتر است امّا 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.

VHDL مثال:

```
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 نوع۲ 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) );
```

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

✓ 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

.

END PACKAGE BODY *pack_name* **;**

اگر در package فقط type و constant داشتیم دیگر نیازی به نوشتن package body نیست.

VHDL مثال:

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

حال 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;
```

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

```
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 NAN2
      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.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIR61@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 هیچگاه خروجی سیگنال نمی دهد بلکه باید حتماً از یک متغیر تعریف شده داخل خود بعنوان خروجی برگشتی استفاده کند.

تابع در architecture و یا package اعلان می شود.

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

VHDL مثال:

```
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 تخصیص (انتساب) متغیر و سیگنال نیز صورت بگیرد مانند :

```
Var1 := analysis ( current_temperature+increment , 10 , 100 ) ;
Cout <= carry ( a , b , c ) ;
```

مثال: VHDL

```
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_3TIRGI@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.

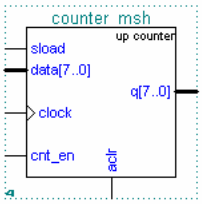
فرق Procedure با function در این است که Procedure می تواند هر تعداد خروجی که بخواهیم داشته باشد در حالیکه تابع فقط مجاز به برگرداندن یک خروجی می باشد همچنین Procedure داده هایی از نوع inout نیز داشته باشد یعنی اطّلاعات وارد procedure شود سپس عملیاتی روی آن انجام گردد و در نهایت به خروجی فرستاده شود.

مثال: VHDL

```
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_3TIRGI@YAHOO.COM

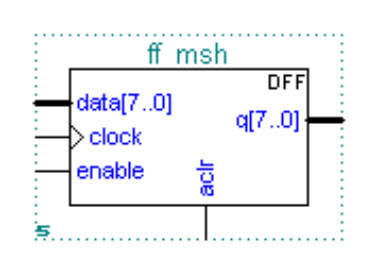
✓ در اینجا فرم کُلی کد نویسی یک شمارنده (Counter) را ملاحظه می کنید :




```
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 کُلی کد 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 :

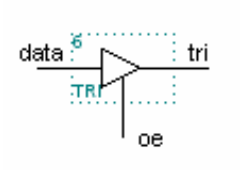
```
ENTITY  machine_name  IS
        PORT(
                clk                                 : IN     STD_LOGIC;
                reset                               : IN     STD_LOGIC;
                input_name ,  input_name            : IN     STD_LOGIC;
                output_name , output_name           : 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_nam
                                        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 کُلی کد 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 حسّاس می کنیم.

MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.MORTEZA.SHABANZADEH.WWW.MORTEZA_3TIRGI@YAHOO.COM

و از هر چه خواستید به شما داد، و اگر [بخواهید] نعمت خدا را بشمارید آن را شمار نتوانید کرد ;
به راستی آدمی ستمگر و ناسپاس است.
ابراهیم ۳۴