# دوره مقدماتی آشنایی با FPGA و VHDL

## عملگرها و صفت‌ها

محمدرضا عزیزی

امیرعلی ابراهیمی

بهار ۱۳۹۷

## مقدمه

- در این فصل عملگرها و صفت‌های موجود در زبان VHDL معرفی می‌شوند.
- این فصل هم‌چنان یکی از فصل‌های «پایه» است و از فصل بعد(کد هموند) مدارات کاربردی و پیچیده‌تر بررسی می‌شوند.
- با این حال، در میان و پایان این فصل نیز مثال‌هایی آموزشی و کاربردی آورده شده است.

# عملگرها

- زبان VHDL تعداد زیادی عملگر از پیش تعریف شده، تهیه کرده است که به صورت زیر دسته بندی می‌شوند:
  - عملگرهای انتساب(Assignment operators)
  - عملگرهای منطقی(Logical operators)
  - عملگرهای حسابی(Arithmetic operators)
  - عملگرهای مقایسه ای(Comparison operators)
  - عملگرهای شیفت(Shift operators)
  - عملگر الحاق(Concatenation operator)

# عملگرهای انتساب(ASSIGNMENT OPERATORS) (۱)

- برای انتساب مقدار به سیگنال‌ها، متغیرها و ثابت‌ها استفاده می‌شود.
  - **<=**
    - مقداردهی به سیگنال(SIGNAL)
  - **:=**
    - مقداردهی به متغیر(VARIABLE)
    - مقداردهی به ثابت(CONSTANT)
    - مقداردهی به GENERIC
    - **مقداردهی اولیه**
  - **=>**
    - مقداردهی به عناصر خاصی از وکتورها با کلیدواژه OTHERS

# عملگرهای انتساب(ASSIGNMENT OPERATORS) (۲)

- مثال:

```
SIGNAL x : STD_LOGIC;
VARIABLE y : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Leftmost bit is MSB
SIGNAL w: STD_LOGIC_VECTOR(0 TO 7);        -- Rightmost bit is MSB

x <= '1';                 -- '1' is assigned to SIGNAL x using "<="
y := "0000";              -- "0000" is assigned to VARIABLE y using ":="
w <= "10000000";          -- LSB is '1', the others are '0'
w <= (0 =>'1', OTHERS =>'0'); -- LSB is '1', the others are '0'
```

# عملگرهای منطقی(LOGICAL OPERATORS) (۱)

- برای انجام عملیات منطقی
- بر روی انواع‌داده زیر قابل اجرا هستند:
  - BIT و BIT_VECTOR
  - STD_LOGIC و STD_LOGIC_VECTOR
  - STD_ULOGIC و STD_ULOGIC_VECTOR
- عملگرهای منطقی تعریف شده در VHDL:
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - XOR
  - XNOR

## عملگرهای منطقی(LOGICAL OPERATORS) (۲)

- نکته ۱: اولویت عملگر NOT از بقیه عملگرهای منطقی بالاتر است.
- مثال:

```
y <= NOT a AND b;          -- (a'.b)
y <= NOT (a AND b);        -- (a.b)'
y <= a NAND b;             -- (a.b)'
```

# عملگرهای حسابی(ARITHMETIC OPERATORS) (۱)

- جهت انجام عملیات حسابی

- قابل اجرا بر روی نوع داده‌های:
  - INTEGER
  - SIGNED
  - UNSIGNED
  - REAL – غیرقابل سنتز
  - STD_LOGIC_VECTOR – در صورت استفاده از پکیج‌های *std_logic_signed* یا *std_logic_unsigned* از کتابخانه *ieee*

# عملگرهای حسابی(ARITHMETIC OPERATORS) (۲)

- لیست دستورات حسابی
  - + : جمع
  - – : تفریق
  - * : ضرب
  - / : تقسیم
  - **: توان
  - **MOD**: باقیمانده
  - **REM**: باقیمانده
  - **ABS**: قدرمطلق

# عملگرهای حسابی(ARITHMETIC OPERATORS) (۳)

- سنتزپذیری:
  - جمع: سنتزپذیر
  - تفریق: سنتزپذیر
  - ضرب: سنتزپذیر
  - تقسیم: فقط تقسیم بر توان های ۲ قابل سنتز است (عملیات شیفت)
  - توان: فقط پایه‌ها و توان‌های استاتیک قابل سنتز است
  - عملگرهای باقیمانده: عدم سنتزپذیری یا سنتزپذیری پایین
  - قدر مطلق: عدم سنتزپذیری یا سنتزپذیری پایین

# عملگرهای حسابی(ARITHMETIC OPERATORS) (۴)

- تفاوت MOD و REM:
  - X REM y: باقیمانده x تقسیم بر y را با علامت x بر می‌گرداند.
    - x REM y = x – (x/y)*y
  - X MOD y: باقیمانده x تقسیم بر y را با علامت y بر می‌گرداند.
    - x MOD y = x REM y +a * y
    - a =1 when the signs of x and y are different or a = 0 otherwise.

- مثال:

9 rem 5 = 4
9 mod 5 = 4
9 rem (-5) = 4
9 mod (-5) = -1
(-9) rem 5 = -4
(-9) mod 5 = 1
(-9) rem (-5) = -4
(-9) mod (-5) = -4

# عملگرهای مقایسه‌ای (COMPARISON OPERATORS) (۱)

- جهت مقایسه کردن استفاده می‌شود.
- بر روی هر یک از انواع‌داده می‌تواند اجرا شود.
- لیست عملگرهای مقایسه‌ای:
  - = : برابر با
  - =/ : نامساوی با
  - < : کوچکتر از
  - > : بزرگتر از
  - =< : کوچتر یا مساوی با
  - => : بزرگتر یا مساوی با

# عملگرهای مقایسه‌ای (COMPARISON OPERATORS) (۲)

- مثال:

```
123 = 123          --result TRUE
'A' = 'A'          --result TRUE
123 = 456          --result FALSE
'A' = 'Z'          --result FALSE
123 < 456          --result TRUE
'1' > '0'          --result TRUE
96 >= 102          --result FALSE
'X' < 'X'          --result FALSE
```

# عملگرهای شیفت (SHIFT OPERATORS) (۱)

- جهت شیفت دادن اطلاعات.
- `<left operand> <shift operation> <right operand>`
  - `<left operand>` باید `BIT_VECTOR` باشد.
    - در صورت استفاده از *std_logic_unsigned* یا *std_logic_signed* می‌تواند STD_LOGIC_VECTOR نیز باشد.
  - `<right operand>` باید `INTEGER` باشد.
    - + و – می‌تواند در جلوی این عدد قرار بگیرد.
- لیست دستورات شیفت:
  - **SLL**: **Shift Left Logic** – شیفت به چپ منطقی – سمت راست با ۰ پر می‌شود.
  - **SRL**: **Shift Right Logic** – شیفت به راست منطقی – سمت راست با ۰ پر می‌شود.
  - **SLA**: **Shift Left Arithmetic** – شیفت به چپ حسابی – بیت سمت راست تکرار می‌شود.
  - **SRA**: **Shift Right Arithmetic** – شیفت به راست حسابی – بیت سمت چپ تکرار می‌شود.
  - **ROL**: **Rotate Left** – شیفت چرخشی به چپ
  - **ROR**: **Rotate Right** – شیفت چرخشی به راست

# عملگرهای شیفت (SHIFT OPERATORS) (۲)

- مثال: فرض کنید x یک BIT_VECTOR با مقدار "01001" باشد.

```
-------------------------------------------------------------------
y <= x SLL 2;   --y<="00100" (y <= x(2 DOWNTO 0) & "00";)
y <= x SLA 2;   --y<="00111" (y <= x(2 DOWNTO 0) & x(0) & x(0);)
y <= x SRL 3;   --y<="00001" (y <= "000" & x(4 DOWNTO 3);)
y <= x SRA 3;   --y<="00001" (y <= x(4) & x(4) & x(4) & x(4 DOWNTO 3);)
y <= x ROL 2;   --y<="00101" (y <= x(2 DOWNTO 0) & x(4 DOWNTO 3);)
y <= x SRL -2;  --same as "x SLL 2"
-------------------------------------------------------------------
```

# عملگر الحاق(CONCATENATION OPERATOR) (۱)

- برای دسته‌بندی داده ها استفاده می‌شود.
- همچنین برای شیفت‌دادن، طبق مثال قبل، می‌تواند استفاده شود.
- نماد: **&**
- قابل استفاده بر روی نوع‌داده‌های زیر:
  - BIT_VECTOR
  - STD_(U)LOGIC_VECTOR
  - (UN)SIGNED

# عملگر الحاق(CONCATENATION OPERATOR) (۲)

- مثال:

```
-------------------------------------------------------------
CONSTANT v: BIT :='1';
CONSTANT x: STD_LOGIC :='Z';
SIGNAL y: BIT_VECTOR(1 TO 4);
SIGNAL z: STD_LOGIC_VECTOR(7 DOWNTO 0);
y <= (v & "000");                    --result: "1000"
y <= v & "000";      --same as above (parentheses are optional)
z <= (x & x & "11111" & x);          --result: "ZZ11111Z"
z <= ('0' & "011111" & x);           --result: "0011111Z"
-------------------------------------------------------------
```

# OTHERS

➢ مثال:

```
SIGNAL y: BIT_VECTOR(1 TO 4);
-------------------------------------------------------------------------
y <= (OTHERS=>'0');                 --result: "0000"
y <= (4=>'1', OTHERS=>'0');         --result: "0001" (nominal mapping)
y <= ('1', OTHERS=>'0');            --result: "1000" (positional mapping)
y <= (4=>'1', 2=>v, OTHERS=>'0');  --result: "0101" (nominal mapping)
z <= (OTHERS=>'Z');                 --result: "ZZZZZZZZ"
z <= (4=>'1', OTHERS=>'0');         --result: "00010000" (nominal mapping)
z <= (4=>x, OTHERS=>'0');           --result: "000Z0000" (nominal mapping)
z <= ('1', OTHERS=>'0');            --result: "10000000" (posit. mapping)
```

# خلاصه عملگرها

**Table 4.1**
Operators.

| Operator type | Operators | Data types |
|---|---|---|
| Assignment | <=, :=, => | Any |
| Logical | NOT, AND, NAND, OR, NOR, XOR, XNOR | BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR, STD_ULOGIC, STD_ULOGIC_VECTOR |
| Arithmetic | +, −, *, /, ** (mod, rem, abs)♦ | INTEGER, SIGNED, UNSIGNED |
| Comparison | =, /=, <, >, <=, >= | All above |
| Shift | sll, srl, sla, sra, rol, ror | BIT_VECTOR |
| Concatenation | &, (, , ,) | Same as for logical operators, plus SIGNED and UNSIGNED |

# صفت‌ها(ATTRIBUTES) (۱)

- **صفت‌های دادهٔ** سنتزپذیر:
  - **d’LOW** : مقدار **پایین**
  - **d’HIGH** : مقدار **بالا**
  - **d’LEFT** : مقدار **چپ**
  - **d’RIGHT** : مقدار **راست**
  - **d’LENGTH** : اندازه وکتور
  - **d’RANGE** : بازه وکتور
  - **d’REVERSE_RANGE** : بازه وکتور به صورت معکوس
  - **d’ASCENDING** : آیا بازه، صعودی است؟

# صفت‌ها(ATTRIBUTES) (۲)

- مثال:

```
SIGNAL d : STD_LOGIC_VECTOR (0 TO 7);

-----------------------------------------------------------

-- d'LOW=0

-- d'HIGH=7

-- d'LEFT=0

-- d'RIGHT=7

-- d'LENGTH=8

-- d'RANGE=(0 TO 7)

-- d'REVERSE_RANGE=(7 DOWNTO 0)
```

# صفت‌ها(ATTRIBUTES) (۳)

- مثال:

```
TYPE my_integer IS RANGE 0 TO 255;
------------------------------------------------------------------
x1 <= my_integer'LEFT;   --result=0 (type of x1 must be my_integer)
x2 <= my_integer'RIGHT;  --result=255 (type of x2 must be my_integer)
x3 <= my_integer'LOW;    --result=0 (type of x3 must be my_integer)
x4 <= my_integer'HIGH;   --result=255 (type of x4 must be my_integer)
y <= my_integer'ASCENDING; --result=TRUE (type of y must be BOOLEAN)
```

## صفت‌ها(ATTRIBUTES) (۴)

- مثال: یک حلقه را می‌توان به ۴ صورت زیر نوشت:

```
SIGNAL x: STD_LOGIC_VECTOR (0 TO 7);
-----------------------------------------------------
FOR i IN RANGE (0 TO 7) LOOP ...
FOR i IN x'RANGE LOOP ...
FOR i IN RANGE (x'LOW TO x'HIGH) LOOP ...
FOR i IN RANGE (0 TO x'LENGTH-1) LOOP ...
```

# صفت‌ها(ATTRIBUTES) (۵)

- **صفت‌های دادهٔ برای نوع‌داده‌های شمارشی(enumerated):**
  - **d’VAL(pos)** : مقدارِ مکان مشخص شده
  - **d’POS(value)** : مکانِ مقدار مشخص شده
  - **d’LEFTOF(value)** : مقداری که در مکان سمت چپ value قرار دارد.
  - **d’RIGHTOF(value)** : مقداری که در مکان سمت راست value قرار دارد.
  - **d’VAL(row, column)** : مقدارِ مکان مورد نظر را بر می‌گرداند.
- این صفت‌ها **سنتزپذیری کم** دارند یا **سنتزپذیر نیستند**.

# صفت‌ها(ATTRIBUTES) (۶)

- مثال:

```
TYPE state IS (a, b, c);

-----------------------------------------------------------------

x1 <= state'LEFT;        --result=a (="00") (type of x1 must be state)

x2 <= state'RIGHT;       --result=c (="10") (type of x2 must be state)

x3 <= state'LOW;         --result=a (="00") (type of x3 must be state)

x4 <= state'HIGH;        --result=c (="10") (type of x4 must be state)

y <= state'POS(b);       --result=1 (="01") (type of y is INTEGER)

z <= state'VAL(1);       --result=b (="01") (type of z must be state)
```

# صفت‌ها(ATTRIBUTES) (۷)

- **صفت‌های سیگنال(signal attributes):**
  - **s’EVENT** : وقتی بر روی سیگنال s یک رویداد(event) رخ دهد، TRUE برمی‌گرداند.
  - **s’STABLE** : وقتی هیچ رویدادی بر روی سیگنال s رخ نداده باشد، TRUE برمی‌گرداند.
  - **s’ACTIVE** : وقتی s = ‘1’، TRUE برمی‌گرداند.
  - **s’QUIET<time>** : وقتی بر روی سیگنال s برای time واحد زمانی رویداد رخ نداده باشد، TRUE برمی‌گرداند.
  - **s’LAST_EVENT** : مدت زمانی که از رویداد قبل گذشته است را برمی‌گرداند.
  - **s’LAST_ACTIVE** : مدت زمانی که از آخرین باری که سیگنال ‘1’ بوده است، گذشته است را برمی‌گرداند.
  - **s’LAST_VALUE** : مقدار سیگنال s را قبل از آخرین رویداد باز می‌گرداند.
- اکثر صفت‌های سیگنال جهت شبیه‌سازی استفاده می‌شود و از این لیست **فقط دو مورد اول سنتزپذیر است** و s’EVENT پر کاربرد ترین آن‌هاست.

## صفت‌ها(ATTRIBUTES) (۸)

- **مثال:** هر ۴ دستور زیر سنتزپذیر و مشابه یک‌دیگرند. هر ۴ دستور در صورتی که لبه بالارونده clk رخ دهد، مقدار TRUE برمی‌گردانند.

```
IF (clk'EVENT AND clk='1')...          -- EVENT attribute used
                                       -- with IF
IF (NOT clk'STABLE AND clk='1')...     -- STABLE attribute used
                                       -- with IF
WAIT UNTIL (clk'EVENT AND clk='1');    -- EVENT attribute used
                                       -- with WAIT
IF RISING_EDGE(clk)...                 -- call to a function
```

# خلاصه صفت‌ها

**Table 4.2**
Attributes.

| Application | Attributes | Return value |
|---|---|---|
| For regular DATA | d'LOW | Lower array index |
| | d'HIGH | Upper array index |
| | d'LEFT | Leftmost array index |
| | d'RIGHT | Rightmost array index |
| | d'LENGTH | Vector size |
| | d'RANGE | Vector range |
| | d'REVERSE_RANGE | Reverse vector range |
| For enumerated DATA | d'VAL(pos)◆ | Value in the position specified |
| | d'POS(value)◆ | Position of the value specified |
| | d'LEFTOF(value)◆ | Value in the position to the left of the value specified |
| | d'VAL(row, column)◆ | Value in the position specified |
| For a SIGNAL | s'EVENT | True when an event occurs on s |
| | s'STABLE | True if no event has occurred on s |
| | s'ACTIVE◆ | True if s is high |

# سربارگذاری عملگرها(OPERATOR OVERLOADING) (۱)

- در بخش عملگرهای حسابی، عملگرهایی مانند ’+‘، ’-‘، ’*‘ و ... معرفی شدند که این عملگرها بر روی نوع داده‌های مشخصی قابل اجرا هستند. برای مثال عملگر ’+‘ امکان جمع دو BIT_VECTOR را به ما نمی‌دهد.
- در این بخش و در قالب یک مثال، عملگری با همان نام( ’+‘ ) برای جمع دو BIT_VECTOR تعریف می‌کنیم.

# سربارگذاری عملگرها(OPERATOR OVERLOADING) (۲)

- **مثال:** فرض کنید می‌خواهیم یک عدد INTEGER را با یک عدد دودویی یک بیتی جمع کنیم. برای این کار، تابع زیر را می‌نویسیم(نحوه نوشتن تابع و استفاده از آن، در آینده توضیح داده خواهد شد):

```
FUNCTION "+" (a: INTEGER, b: BIT) RETURN INTEGER IS
BEGIN
    IF (b='1') THEN RETURN a+1;
    ELSE RETURN a;
    END IF;
END "+";
-------------------------------------------------------------
SIGNAL inp1, outp: INTEGER RANGE 0 TO 15;
SIGNAL inp2: BIT;
(...)
outp <= 3 + inp1 + inp2;
(...)
```

## (۱) GENERIC

- GENERIC یک پارامتر عمومی است که هدف آن افزایش انعطاف و reusibility کد است.
- GENERIC در ENTITY تعریف می‌شود و سراسری است، یعنی در تمام طراحی، و خود ENTITY، قابل استفاده است.
- ساختار:

```
GENERIC (parameter_name : parameter_type := parameter_value);
```

## (۲) GENERIC

➢ مثال:

```
ENTITY my_entity IS
    GENERIC (n : INTEGER := 8);
    PORT (...);
END my_entity;
ARCHITECTURE my_architecture OF my_entity IS
    ...
END my_architecture:
-------------------------------------------------------------
GENERIC (n: INTEGER := 8; vector: BIT_VECTOR := "00001111");
```

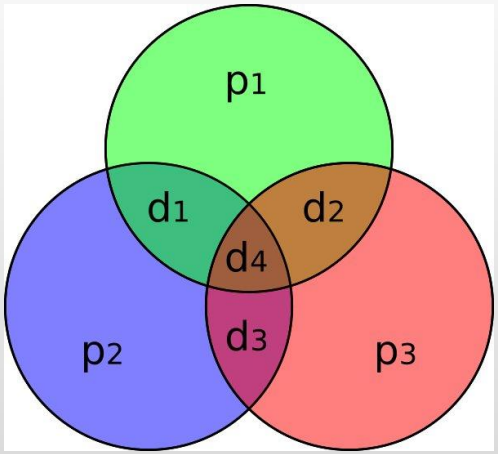
# مثال

- تولید بیت توازن (Parity bit):
  - بیت توازن بیتی است که برای نشان دادن زوج یا فرد بودن تعداد بیت‌هایی که ۱ می‌باشند به بیت‌ها اضافه می‌شود.
  - هنگامی که از **توازن زوج** استفاده می‌شود، اگر **تعداد یک‌های ورودی زوج** باشد **بیت توازن صفر** می‌شود و بالعکس.
    - برای محاسبه بیت توازن زوج، باید حاصلXOR بیت‌های ورودی را محاسبه کرد.
  - هنگامی که از **توازن فرد** استفاده می‌شود اگر **تعداد یک‌های ورودی فرد** باشد **بیت توازن صفر** می‌شود و بالعکس.
    - برای محاسبه بیت توازن فرد، باید حاصل XNOR بیت‌های ورودی را محاسبه کرد.

# تمرین (۱)

## کد همینگ(Hamming code):

- در مخابرات، کد همینگ، کد تصحیح خطایی است که به افتخار ریچارد همینگ، مخترع آن گذاشته شده‌است.
- کدهای همینگ می‌توانند همزمان ۲ بیت خطا را شناسایی کنند و ۱ بیت خطا را تصحیح کنند.
- فرض کنید یک سیستم سخت‌افزاری طراحی کرده اید و می‌خواهید اطلاعاتی ۴ بیتی را با استفاده از الگوریتم کدگذاری همینگ، کدگذاری کرده و آن را برای سیستم دیجیتال دیگری ارسال کنید، تا در صورت بروز خطا، قابل تشخیص باشد. **(پیاده سازی بخش فرستنده)**
  - کد VHDL این سیستم کدگذاری همینگ را نوشته و آن را شبیه‌سازی کنید.



# تمرین (۲)

## کد همینگ(Hamming code):

- فرض کنید یک سیستم سخت‌افزاری اطلاعات خود را به صورت همینگ کدگذاری کرده و آن را برای سیستم دیجیتال شما ارسال کرده است. این اطلاعات را بررسی کنید و محل خطا را تشخیص دهید. **(پیاده سازی بخش گیرنده)**
  - کد VHDL این سیستم تشخیص خطای همینگ را نوشته و آن را شبیه‌سازی کنید.

# پایان بخش عملگرها و صفت‌ها