Circuit Design with VHDL

Operators and Attributes

Instructor: Ali Jahanian

Outline

- I-CIRCUIT DESIGN
  - 4 Operators and Attributes
    - 4.1 Operators
    - 4.2 Attributes
    - 4.3 User-Defined Attributes
    - 4.4 Operator Overloading
    - 4.5 GENERIC
    - 4.6 Examples
    - 4.7 Summary

Purpose of this chapter

- In this chapter:
  - The purpose of this chapter, along with the preceding chapters, is to lay the basic foundations of VHDL.
  - It is indeed impossible to write any code efficiently without undertaking first the sacrifice of understanding:
    - data types,
    - *operators*,
    - *attributes*.

Operators

- VHDL provides several kinds of pre-defined operators:
  - Assignment operators
  - Logical operators
  - Arithmetic operators
  - Relational operators
  - Shift operators
  - Concatenation operators

Assignment operators

<= Used to assign a value to a SIGNAL.

:= Used to assign a value to a VARIABLE, CONSTANT, or GENERIC. Used also for establishing initial values.

=> Used to assign values to individual vector elements or with OTHERS.

Example (Assignment operators)

```
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'
```

Logical Operators

- Used to perform logical operations. The data must be of type:
  - BIT (BIT_VECTOR)
  - STD_LOGIC (STD_LOGIC_VECTOR)
  - STD_ULOGIC (STD_ULOGIC_VECTOR)

Logical Operators …

- The logical operators are:
  - NOT
  - AND
  - OR
  - NAND
  - NOR
  - XOR
  - XNOR

Example (Logical Operators)

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

Arithmetic Operators

- Used to perform arithmetic operations. The data can be of type
  - INTEGER
  - SIGNED
  - UNSIGNED
  - REAL
    - (Not synthesizable).
  - STD_LOGIC_VECTOR
    - (*ieee*.std_logic_signed and *ieee*.std_logic_unsigned)

Arithmetic Operators …

+	Addition	
-	Subtraction	
*	Multiplication	
/	Division	
**	Exponentiation	
MOD	Modulus	(Not synthesizable)
REM	Remainder	(Not synthesizable)
ABS	Absolute	(Not synthesizable)

Comparison Operators

=	Equal to
/=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Shift Operators

⟨left operand⟩ ⟨shift operation⟩ ⟨right operand⟩

BIT_VECTOR INTEGER

- sll Shift left logic – positions on the right are filled with '0's
- srl Shift right logic – positions on the left are filled with '0's

Data Attributes

- The pre-defined, synthesizable data attributes are the following:
  - d'LOW: Returns lower array index
  - d'HIGH: Returns upper array index
  - d'LEFT: Returns leftmost array index
  - d'RIGHT: Returns rightmost array index
  - d'LENGTH: Returns vector size
  - d'RANGE: Returns vector range
  - d'REVERSE_RANGE: Returns vector range in reverse order

Example (Data Attributes)

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

d'LOW=0
d'HIGH=7
d'LEFT=7
d'RIGHT=0
d'LENGTH=8
d'RANGE=(7 downto 0)
d'REVERSE_RANGE=(0 to 7)

Example (Data Attributes) …

```
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 ...
```

Data Attributes (*enumerated* type)

- If the signal is of enumerated type, then:
  - d'VAL(pos): Returns value in the position specified
  - d'POS(value): Returns position of the value specified
  - d'LEFTOF(value): Returns value in the position to the left of the value specified
  - d'VAL(row, column): Returns value in the position specified; etc.

There is little or no synthesis support for enumerated data type attributes.

Signal Attributes

- Let us consider a signal s. Then:
  - s'EVENT: Returns true when an event occurs on s
  - s'STABLE: Returns true if no event has occurred on s
  - s'ACTIVE: Returns true if s = '1'
  - s'QUIET <time>: Returns true if no event has occurred during the time specified
  - s'LAST_EVENT: Returns the time elapsed since last event
  - s'LAST_ACTIVE: Returns the time elapsed since last s ='1'
  - s'LAST_VALUE: Returns the value of s before the last event; etc.

Example (Signal Attributes)

```
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
```

User-Defined Attributes

- To employ a user-defined attribute, it must be *declared* and *specified*.

Attribute declaration:

BIT, INTEGER, STD_LOGIC_VECTOR, etc.

```
ATTRIBUTE attribute_name: attribute_type;
```

TYPE, SIGNAL, FUNCTION, etc.

Attribute specification:

'0', 27, "00 11 10 01", etc

```
ATTRIBUTE attribute_name OF target_name: class IS value;
```

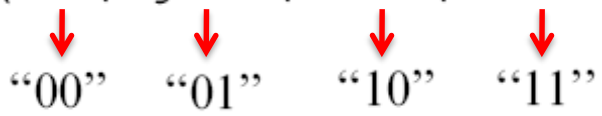
Example (User-Defined Attributes)

- Example 1:

```
ATTRIBUTE number_of_inputs: INTEGER;                -- declaration
ATTRIBUTE number_of_inputs OF nand3: SIGNAL IS 3; -- specification
   ...
inputs <= nand3'number_of_pins;     -- attribute call, returns 3
```

- Example 2: Enumerated encoding

```
TYPE color IS (red, green, blue, white);
```




```
ATTRIBUTE enum_encoding OF color: TYPE IS "11 00 10 01";
```

A user-defined attribute can be declared anywhere, except in a PACKAGE BODY.

Operator Overloading

- We have just seen that attributes can be user-defined. The same is true for operators.
- As an example, the pre-defined "+" operator does not allow addition between data of type BIT.
- We can define our own operators, using the same name as the pre-defined ones. For example, we could use "+" to indicate a new kind of addition, this time between values of type BIT_VECTOR. This technique is called operator *overloading*.

Example (Operator Overloading)

```
---------------------------------------
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;
   (...)
-------------------------------
```

GENERIC

- A GENERIC statement, when employed, must be declared in the ENTITY. The specified parameter will then be truly global

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

Example (GENERIC)

```
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");
```

Generic Decoder (Example 4.1)

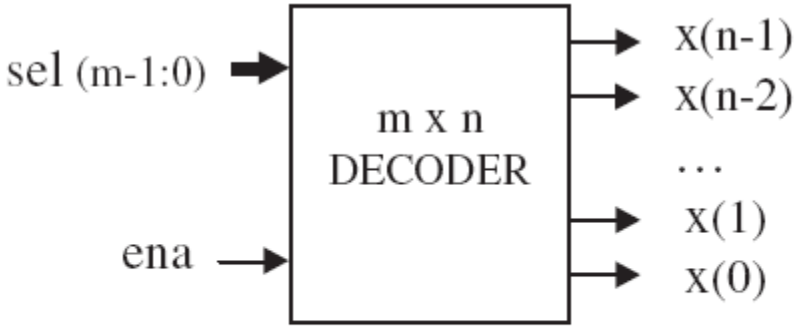


Figure 4.1
Decoder of example 4.1.

ena	sel	x
0	00	1111
1	00	1110
	01	1101
	10	1011
	11	0111

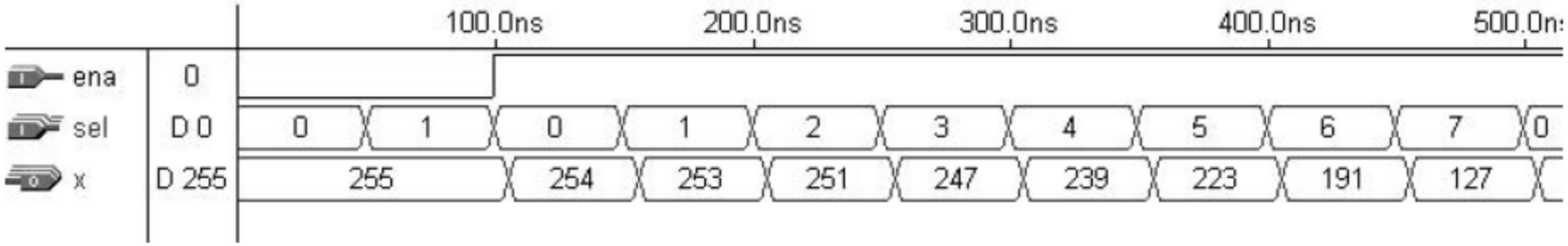


Figure 4.2
Simulation results of example 4.1.

Generic Decoder (Example 4.1) …

```
1  ----------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------
5  ENTITY decoder IS
6     PORT ( ena : IN STD_LOGIC;
7            sel : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8            x : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
9  END decoder;
10 ----------------------------------------------
```

Generic Decoder (Example 4.1) …

```
11 ARCHITECTURE generic_decoder OF decoder IS
12 BEGIN
13    PROCESS (ena, sel)
14       VARIABLE temp1 : STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
15       VARIABLE temp2 : INTEGER RANGE 0 TO x'HIGH;
16    BEGIN
17       temp1 := (OTHERS => '1');
18       temp2 := 0;
19       IF (ena='1') THEN
20          FOR i IN sel'RANGE LOOP  -- sel range is 2 downto 0
21             IF (sel(i)='1') THEN  -- Bin-to-Integer conversion
22                temp2:=2*temp2+1;
23             ELSE
24                temp2 := 2*temp2;
25             END IF;
26          END LOOP;
27          temp1(temp2):='0';
28       END IF;
29       x <= temp1;
30    END PROCESS;
31 END generic_decoder;
32 ------------------------------------------
```

Generic Parity Detector

Figure 4.3
Generic parity detector of example 4.2.

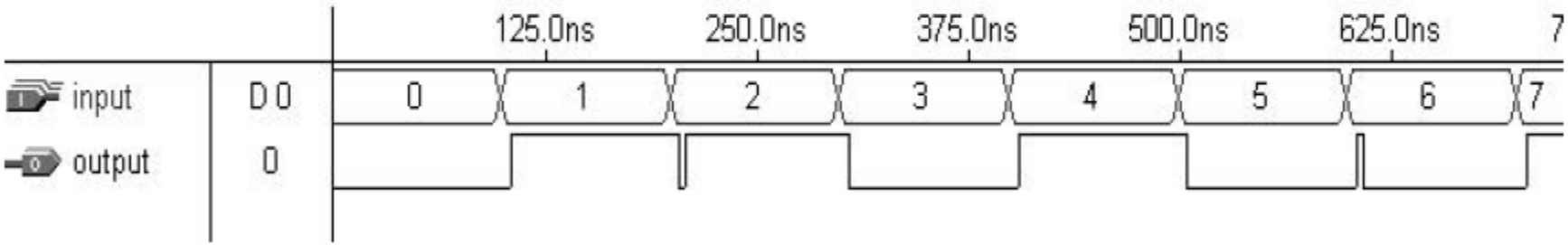


Figure 4.4
Simulation results of example 4.2.

Generic Parity Detector …

```
1  ----------------------------------------
2  ENTITY parity_det IS
3     GENERIC (n : INTEGER := 7);
4     PORT ( input: IN BIT_VECTOR (n DOWNTO 0);
5            output: OUT BIT);
6  END parity_det;
7  ----------------------------------------
8  ARCHITECTURE parity OF parity_det IS
9  BEGIN
10    PROCESS (input)
11       VARIABLE temp: BIT;
12    BEGIN
13       temp := '0';
14       FOR i IN input'RANGE LOOP
15          temp := temp XOR input(i);
16       END LOOP;
17       output <= temp;
18    END PROCESS;
19 END parity;
20 ----------------------------------------
```

Generic Parity Generator

Figure 4.5
Generic parity generator of example 4.3.

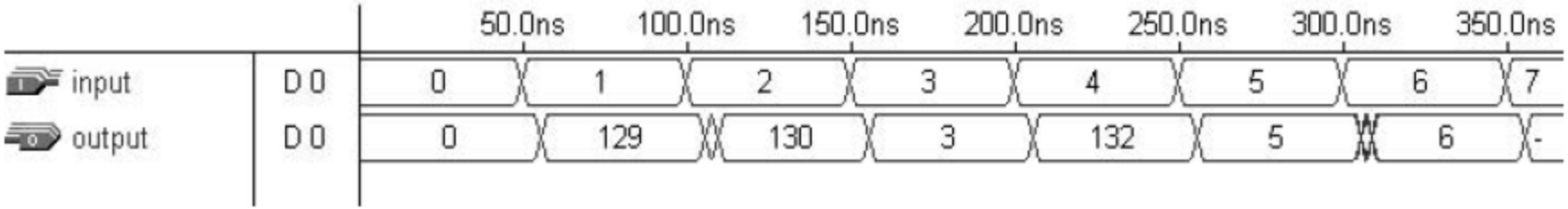


Figure 4.6
Simulation results of example 4.3.

Generic Parity Generator …

```
1  ------------------------------------------------
2  ENTITY parity_gen IS
3     GENERIC (n : INTEGER := 7);
4     PORT ( input: IN BIT_VECTOR (n-1 DOWNTO 0);
5            output: OUT BIT_VECTOR (n DOWNTO 0));
6  END parity_gen;
7  ------------------------------------------------
```

Generic Parity Generator …

```
8  ARCHITECTURE parity OF parity_gen IS
9  BEGIN
10    PROCESS (input)
11       VARIABLE temp1: BIT;
12       VARIABLE temp2: BIT_VECTOR (output'RANGE);
13    BEGIN
14       temp1 := '0';
15       FOR i IN input'RANGE LOOP
16          temp1 := temp1 XOR input(i);
17          temp2(i) := input(i);
18       END LOOP;
19       temp2(output'HIGH) := temp1;
20       output <= temp2;
21    END PROCESS;
22 END parity;
23 ---------------------------------------------
```

Summary

- A summary of VHDL 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

Summary …

- A summary of VHDL 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

Thanks for your attention

- Don’t forget

Problems!!!

Next session

5 Concurrent Code

- 5.1 Concurrent versus Sequential
- 5.2 Using Operators
- 5.3 WHEN (Simple and Selected)
- 5.4 GENERATE
- 5.5 BLOCK