Operators & Expressions

Introductory VHDL Methodology

Objectives

After completing this module, you will be able to…

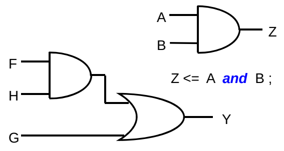
- Write standard VHDL expressions
- Infer logic and functionality using VHDL operators
- Apply appropriate operators to each data-type
- Reference appropriate packages for arithmetic functions
- Use VHDL 'slice' to reference sub-bus structures
- Use VHDL 'concatenation' operator

Operators

Logical Operator
 and, or, nand, nor, xor, xnor
Relational Operator
 =, /=, <, <=, >, >=
Shifting Operator
 sll, srl, sra, rol, ror
Adding Operator
 +, -, &
Sign Operator
 +, -
Multiplying Operator
 *, /, mod, rem
Miscellaneous Operator
 * *, abs, not

Logical Operators

Logical operators are pre-defined for data-types bit, boolean and 1 dimensional arrays of bits



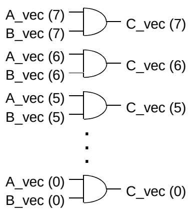
Z <= A ***and*** B ;

Y <= G ***or*** (F ***and*** H) ;

and
or
nand
nor
xor
not
xnor (VHDL-93)

Logical Operations on Arrays

signal A_vec, B_vec, C_vec **:** ***bit_vector*** (7 *downto* 0) ;



C_vec **<=** A_vec ***and*** B_vec ;

Rules for use on Arrays
1. Arrays must be the same type
2. Arrays must have the same length
3. Operation applied to positional elements within array, left to right

Relational Operators

Relational operators are pre-defined for most data-types

All Relational operations return type Boolean

=	Equality
/=	Inequality
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal

signal FLAG_BIT **:** ***boolean ;***
signal A, B **:** ***integer ;***

FLAG_BIT **<= (** A **>** B **) ;**

If A is greater than B, FLAG_BIT will be assigned ***true,*** otherwise ***false***

Relational Operations on Arrays

```
signal  A_vec : bit_vector  ( 7 downto 0 ) := “11000110” ;
signal  B_vec : bit_vector  ( 5 downto 0 ) := “111001” ;
```

Rules for use on Arrays
1. Arrays must be same type
2. Arrays may be different lengths
3. Arrays of different lengths are aligned left and then lexically compared.   (used primarily for comparing character strings, not recommended for arrays that indirectly represent binary data)

```
if ( A_vec  > B_vec )  then
State <=  Normal
else
State <=  Code_Red
end if  ...
```

In the above example, the operation will return ***false*** since the two arrays are first left aligned, and then compared. There is no pre-defined binary or numerical inference

As a Rule, ALWAYS insure that the arrays are the same size

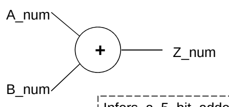
Arithmetic

- **Arithmetic operators are pre-defined for data-types integer, real and physical**
- **They are not pre-defined for Arrays**

```
signal   A_num, B_num : integer  range  0 to 15 ;
signal   Z_num  : integer  range  0 to 31 ;
```

+	Addition
-	Subtraction
*	Multiplication
/	Division
abs	Absolute Value
**	Exponentiation

```
Z_num <=  ( A_num  +  B_num ) ;
```



Infers a 5 bit adder..

Arithmetic of Arrays

- **To accomplish arithmetic operations on arrays- in effect treating them as binary or numerical representations, requires functions (sub-programs) supplied by the IEEE or the tool vendor**

```
package  NUMERIC_STD is
  function “+” (A,B: bit_vector) return  bit_vector ;
  function “+” (A: bit_vector, B: integer ) return  bit_vector ;
  function “-”  (A,B: bit_vector) return  bit_vector ;
. . . .
```

```
library  IEEE ;
use IEEE.std_logic_1164.all ;
use  IEEE.numeric_std.all ;
```

- **The package numeric_std is an IEEE standard library and will provide maximum code portability**
- **The operator “+” is overloaded in that it refers to a different function call, based on the left and right operand, and the return parameter in the function declaration. Such functions will normally be included in so-called “arithmetic packages”. The package may require compilation into the work library. Some tools pre-compile these packages into their own internal library**

Array Arithmetic

- **If the necessary functions are available and made visible within the module , (via the “use” clause) the compiler will automatically pass the arguments to the sub-program, and return the result**

```
signal   A_vec : std_logic_vector  ( 7 downto 0 ) := “11001001”  ;
signal   B_vec : std_logic_vector  ( 7 downto 0 ) := “11100100 ” ;
signal   Z_vec : std_logic_vector  ( 8 downto 0 )  ;
signal   D_int  :  integer  range ( 0 to 9 )  ;
```

Otherwise the compiler would inform you that the expressions below are undefined

```
Z_vec  <=  A_vec +  B_vec  ;

Z_vec  <=  A_vec +  D_int  ;
```

Concatenation

- **The Concatenation operator (&) allows flexible grouping of scalars and arrays into larger arrays.**

```
signal   A_vec, B_vec : std_logic_vector  ( 7 downto 0 ) ;
signal   Z_vec : std_logic_vector  ( 15 downto 0 ) ;
signal   A_bit, B_bit, C_bit, D_bit : std_logic ;
signal   X_vec : std_logic_vector  ( 2 downto 0 ) ;
signal   Y_vec : std_logic_vector  ( 8 downto 0 ) ;
```

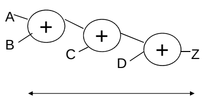
```
Z_vec  <=  A_vec & B_vec ;

X_vec  <=  A_bit & B_bit & C_bit ;

Y_vec  <=  B_vec & D_bit ;
```

This type of assignment uses positional association

Grouping Operators

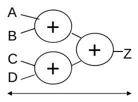
- **Grouping operators in a given expression can help to guide some aspects of logic synthesis while enhancing the readability of the code**

Z <= A + B + C + D ;



3 logic levels

Z <= (A + B) + (C + D) ;



2 logic levels

This is especially important when the target technology is LUT (Look-Up Table) based. Each added level of logic incurs additional block and routing delays

Array Slices

- **Any group of contiguous elements within an array can be referenced as a slice. The remaining elements are unaffected by the assignment**

```
signal   A_vec, B_vec : std_logic_vector  ( 7 downto 0 ) ;
signal   Z_vec : std_logic_vector  ( 15 downto 0 ) ;
signal   A_bit, B_bit, C_bit, D_bit : std_logic ;
```

```
Z_vec (15 downto 8)  <=  A_vec ;
B_vec  <=  Z_vec (12 downto 5) ;
A_vec (1 downto 0)  <=  C_bit & D_bit ;
      . . .
Z_vec (5 downto 1)  <=  B_vec (1 to 5 ) ;
```

The direction (ascending or descending) of the slice must be consistent with the direction of the array as originally declared

Review Questions

- **What are the rules for logical operations on arrays?**
- **What are the rules for relational operations on arrays?**

Given:

```
signal   A_Bus, B_Bus: std_logic_vector (7 downto 0) ;
signal   Data_Word : std_logic_vector (15 downto 0) ;
signal   A_Bit, B_Bit, C_Bit : std_logic ;
```

Which of the following are permissible in VHDL, and why ?

```
A_Bus < =  B_Bus & C_bit ;
Data_Word <= A_Bus & B_Bus;
Data_Word ( 8 downto 0 ) <= A_Bus & B_Bit ;
Data_Word ( 4 downto 0 ) <= A_Bus (0 to 3) & A_Bit ;
A_Bit & B_Bit <= Data_Word(2) & Data_Word(7) ;
```

Answers

- **What are the rules for logical operations on arrays?**
  — *Size & type must match, operation applied to matching elements (left to right)*
- **What are the rules for relational operations on arrays?**
  — *Type must match, if size is different, arrays are left aligned then lexically compared*

Which of the following are permissible in VHDL, and why ?

```
A_Bus < =  B_Bus & C_bit ;     BAD, size mismatch, type OK
Data_Word <= A_Bus & B_Bus;             OK, size and type match
Data_Word ( 8 downto 0 ) <= A_Bus & B_Bit ;  OK, size and type match
Data_Word ( 4 downto 0 ) <= A_Bus (0 to 3) & A_Bit ; BAD, null slice on  'A_Bus'

A_Bit & B_Bit <= Data_Word(2) & Data_Word(7) ; OK, size and type match
```

Summary

- **Not all VHDL operators are defined for each data type**
- **Arithmetic operations on arrays require sub-programs**
- **All relational operations return type boolean**
- **Logical operations on arrays are performed on matching elements within the arrays**
- **Grouping operators helps guide logic synthesis**
- **Contiguous groups of elements within an array can be treated as a slice**