Arithmetic Packages- Introduction

- It would be very painful if when building a counter we had to think about all the internals
  - We need an adder etc.
    - » How do we build the adder ?
  - Would be much better if we could simply say
    - » Add 1 to the current counter value
    - » count <= count + 1
  - And an (optimal) circuit was automatically generated for us
  - Furthermore, if we work in this way the design is still architecturally independent- A good thing !!

Arithmetic Package Overview- I

- VHDL offers a number of *packages* which provide common arithmetical functions
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Comparison (=, >, <, <=, >=, /=)
- By simply adding in a ‘use’ clause at the top of your code these become available
  - Synthesis automatically generates the logic required !!

Arithmetic Package Overview- II

- There are a number of different packages that exist for historical reasons
  - STD_LOGIC_ARITH, std_logic_signed, std_logic_unsigned
  - NUMERIC_STD (IEEE)
- We will only consider 'NUMERIC_STD' as it is the only standard package which is defined on all commercial synthesis and simulation tools
  - Tools must provide a common set of arithmetical functions
  - Synthesis result (gates and how they are connected) will change with synthesis tool, but functionality will not

Declaring Arithmetic Signals & Variables

- NUMERIC_STD offers 2 data types
  - SIGNED, UNSIGNED
  - These are declared in a similar method to ‘std_logic_vector’
  - Can be used to ‘declare’ signals, variables, even ports in an entity
- UNSIGNED
  - Assumes that only positive values are going to be used
  - Example declaration

```
signal count: unsigned (3 downto 0)
```

  - This creates a signal used for storing values 0 -> 15

Declaring Arithmetic Signals & Variables

- SIGNED
  - 2's complement form, with MSB used as a sign bit
  - Example declaration

```
signal count: signed (3 downto 0)
```

  - Creates a signal used for storing the values -8 -> +7

Integer	Signed
-8	1000
-1	1111
0	0000
+7	0111

Representation of Signed/ Unsigned

- Signed/ Unsigned values are represented using a subset of std_logic_vector
  - I.e. ‘0’, ‘1’ in each bit
- However, cannot perform comparisons, assignments etc. directly with std_logic
  - We need to use conversion functions (see later)

Arithmetic Package Functions- I

- For a detailed list of functions (and their operations) see the program listing from ‘NUMERIC_STD.VHD’ this is the official IEEE package
- How do I read the package header ?
- Consider the function Id: A.6

```
function "+" (L: UNSIGNED; R: NATURAL) return UNSIGNED
```

UNSIGNED + NATURAL = UNSIGNED

Arithmetic Package Functions- II

- Signed Arithmetic Functions:

Function	Argument 1	Argument 2	Returns
+	signed	signed	signed
	signed	integer	signed
-	signed	signed	signed
	signed	integer	signed

- i.e. functions to add/ subtract signed numbers
  - Note integer as a +ve/ -ve argument for signed op's

Arithmetic Package Functions- III

- Unsigned Arithmetic Functions:

Function	Argument 1	Argument 2	Returns
+	unsigned	unsigned	unsigned
	unsigned	natural	unsigned
-	unsigned	unsigned	unsigned
	unsigned	natural	unsigned

- i.e. functions to add/ subtract unsigned numbers
  - Note natural as a +ve only argument since no notion of sign

Arithmetic Package Functions- IV

- Comparison functions:

Function	Name	Argument 1	Argument 2	Returns
=	Equal	unsigned	unsigned	boolean
/=	Not equal			
>	Greater than	signed	signed	
<	Less than	natural	unsigned	
>=	Greater than/equal	integer	signed	
<=	Less than/equal			

Arithmetic Package Functions- V

- Resize functions
  - Used for resizing a signed/ unsigned value
  - Useful if we want to extract carry bit etc.

Function	Description	Argument 1	Argument 2	Returns
resize	Resize argument	unsigned	natural (new size)	unsigned (new size)
		signed	natural (new size)	signed (new size)

- Example

```
newvalue = resize(oldvalue, 5)
```

Arithmetic Package Functions- VI

- Simple conversion functions
  - Used to convert integer/ natural numbers to and from signed/ unsigned

Function	Description	Argument 1	Argument 2	Returns
to_integer	Convert to integer	unsigned		natural
		signed		integer
to_unsigned	Convert to unsigned	natural	natural (size)	unsigned (size)
to_signed	Convert to signed	integer	natural (size)	signed (size)

Arithmetic Package Functions- VII

- Conversion to/ from std_logic_vector
  - Used to convert signed and unsigned values to and from std_logic_vector
    - » Really just copies each bit

Function	Description	Argument	Returns
unsigned	Convert to unsigned	std_logic_vector	unsigned
signed	Convert to signed	std_logic_vector	signed
std_logic_vector	Convert to std_logic_vector	unsigned	std_logic_vector
std_logic_vector	Convert to std_logic_vector	signed	std_logic_vector

Making the Package Visible

- At the top of the VHDL source file, the line

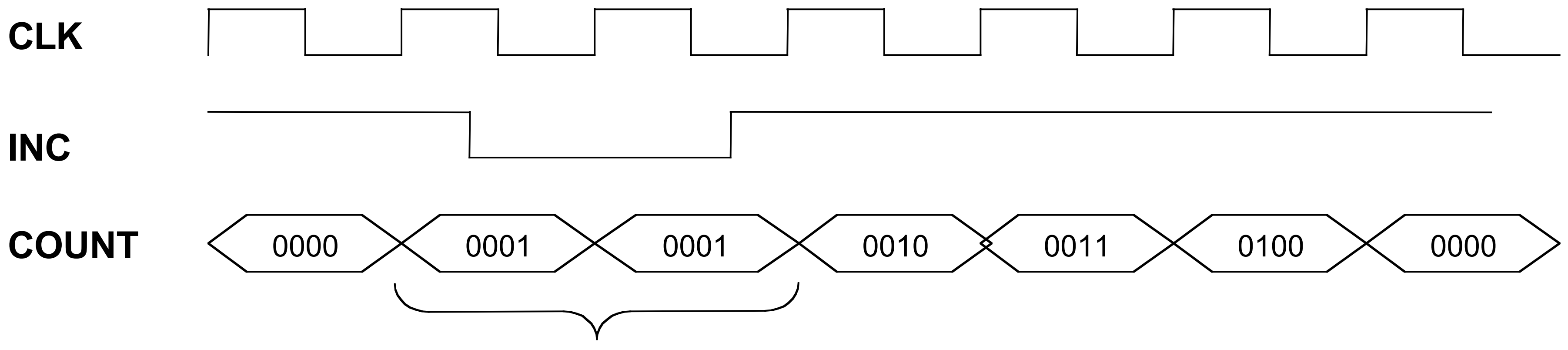
```
use ieee.numeric_std.all
```

Library Name **Package Name**

- is added
- This makes the package (functions, data types, etc.) 'visible'

Arithmetic Package Example- I

- Some concrete examples of the use of the arithmetic package !
- Simple counter
  - Counts up to certain value then resets back
  - Only increments if input ‘inc’ is set to ‘1’



No change when inc = ‘0’

Arithmetic Package Example- II

- VHDL process

```
process (CLK,RESET)
   begin
   if RESET='1'  then   -- reset (asynchronous)
      internal_count <= "0000";
   elsif CLK'EVENT and CLK='1'  then   -- clock
      if inc = '1'  then
         if internal_count = "0100"  then
            internal_count <= "0000";
         else
            internal_count <= internal_count + 1;
         end if;
      else null;
   end if;
 end process;
```

Arithmetic Package Example- III

- Architecture definition

```
use ieee.numeric_std.all;

architecture BEHAVIOR of UPCOUNTER is
  signal internal_count : unsigned (3 downto 0);
begin
-- see previous slide for process

count <= std_logic_vector(internal_count);
end BEHAVIOR;
```

Arithmetic Packages- Conclusion

- Have given an overview of the numeric package and the functions which it contains
- These are widely use for counters etc.
- Can be used to generate many different arithmetic circuits (e.g. multipliers etc.) though it may be better to design these for the given application as the target after synthesis may be inefficient