Data Types

Prof. K. J. Hintz

Department of Electrical
and
Computer Engineering
George Mason University

Composite Date Types

- Arrays
  - Single and multi-dimensional
  - Single Type
- Records
  - Mixed types

Array

- Indexed Collection of Elements All of the Same Type
  - One-dimensional with one index
  - Multi-dimensional with several indices

Array

- Constrained
  - » the bounds for an index are established when the type is defined
- Unconstrained
  - » the bounds are established after the type is defined
- Each position in the array has a scalar index value associated with it

Array Definition Syntax

```
array ( discrete_range { , ... } )
  of    element_subtype_indication ;
```

- *discrete_range* is an index
  - name of previously declared type with optional range constraint

Array Declaration, *e.g.*,

```
type Large_Word is array ( 63 downto 0 )
  of bit ;

type Address_List is array ( 0 to 7 ) of
  Large_Word ;
```

Array Declaration, *e.g.*,

```
type 2D_FFT is array
        ( 1 to 128, 1 to 128 ) of real ;

type Scanner is array
        ( byte range 0 to 63 ) of integer ;

type Sensor_Status is array
        ( Stdby, On, Off ) of time ;
```

Unconstrained Declaration

```
type Detector_Array is array
    ( natural range <> ) of  natural ;
```

- The symbol ‘<>’ is called a box and can be thought of as a place-holder for the index range.
- Box is filled in later when the type is used.

```
variable X_Ray_Detector : Detector_Array
            ( 1 to 64 ) ;
```

Predefined Unconstrained Types

```
type string is array
    ( positive range <> ) of character ;

type bit_vector is array
    ( natural range <> ) of bit ;
```

Predefined Unconstrained Types

```
type std_ulogic_vector is array
     ( natural range <> ) of std_ulogic ;

type bit_vector is array
     ( natural range <> ) of bit ;
```

Unconstrained Array Ports

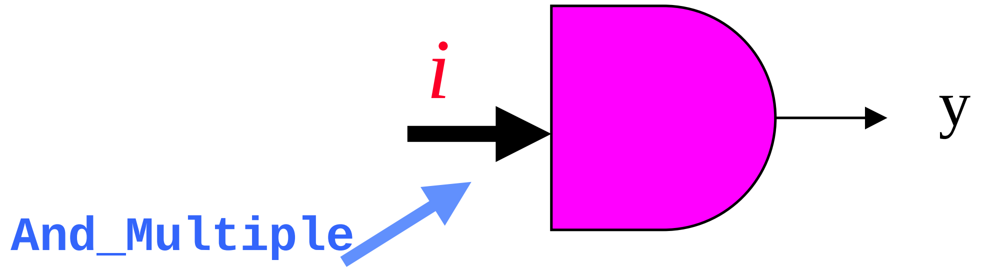
- 1. Specify Port As Unconstrained
- 2. Index Bounds of Signal Determine Size of Port
  - *e.g.*, AND Gates With Different Number of Inputs

1. Unconstrained Array Port, *e.g.*,

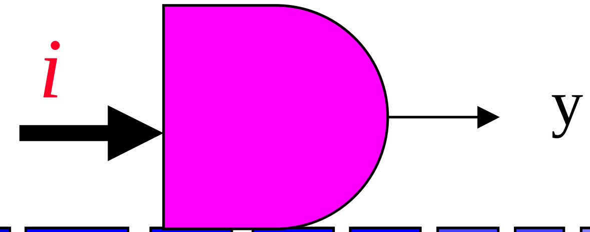
```
entity And_Multiple is

  port (  i    :    in   bit_vector ;
          y    :    out  bit ) ;

end entity And_Multiple ;
```



2. AND, *e.g.,*




```
architecture And_Multiple_B of
 And_Multiple is
begin
 And_Reducer : process ( i ) is
  variable Result : bit ;
  begin
   Result :=  '1' ;
   for Index in i'Range loop
    Result :=  Result and i ( Index ) ;
   end loop ;
```

variable

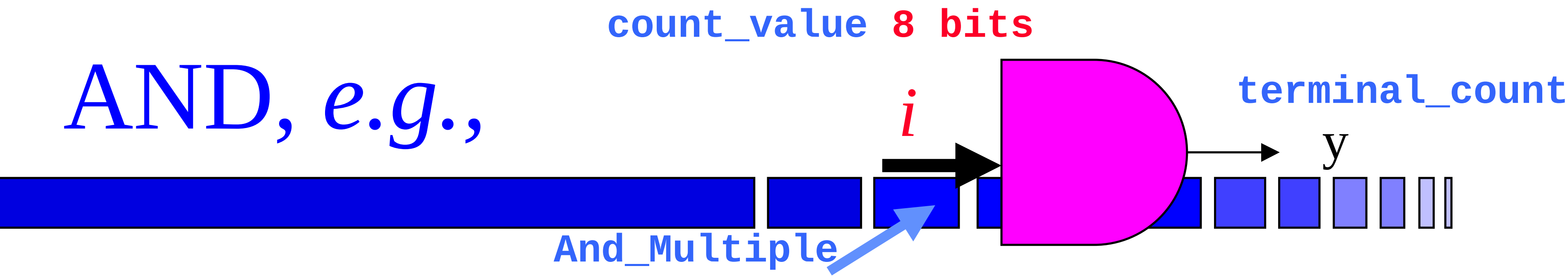
Signal created outside the loop

AND, *e.g.*,

```
    y  <=  Result ;
  end process And_Reducer ;
end architecture And_Multiple_B ;
```

signal

AND, *e.g.*,



architecture And_Multiple_B

```
signal count_value :
    bit_vector ( 7 downto 0 ) ;
signal terminal_count : bit ;
 tc_gate :  entity work.And_Multiple
      ( And_Multiple_B )
  port map (    i =>  count_value ,
                y => terminal_count ) ;
```

AND, *e.g.*,

- The Input Port Is <u>Constrained</u> by the Index Range of the Input Signal, *i.e.*, An 8-Input AND Gate.

Array References

- Arrays Can Be Equated, Rather Than Having to Transfer Element by Element
- Refer to Individual Elements By
  - Single Index Value, *e.g.*, `A ( 5 )`
  - Range: a contiguous sequence of a one-dimensional array can be referred to by using it as an index. *e.g.*, `A( 5 to 15 )`
  - Previously defined subtype
  - Index types do not have to be the same

Array Aggregate Syntax

- A List of Element Values Enclosed in Parentheses
- Used to Initialize Elements of an Array to Literal Values

aggregate **<=** **(** **[** *choices* **=>** **]**
expression **{ . . . }** **)**

Array Aggregate

- Two Ways of Referring to Elements
  - Positional: explicitly list values in order
  - Named Association: Explicitly list values by their index using “choices”
    - » Order NOT important

- Positional and Named Association Cannot Be Mixed Within an Aggregate.

Array Aggregate, *e.g.*,

```
type Sensor_Status is
  array ( Stdby , On , Off )  of time ;

variable FLIR_Status :
 Sensor_Status := ( 0 sec , 0 sec , 0 sec );

variable FLIR_Status :
 Sensor_Status := ( On => 5 sec ) ;
```

Array Aggregate, *e.g.*,

- **`others`** Can Be Used in Place of an Index in a Named Association,
  - Indicating a Value to Be Used for All Elements Not Explicitly Mentioned

```
variable FLIR_Status : Sensor_Status :=
 ( Off  => 10 min, others => 0 sec ) ;
```

Array Aggregate, *e.g.*,

- A Set of Values *Can Be Set to a Single Value* by Forming a List of Elements Separated by Vertical Bars, | .

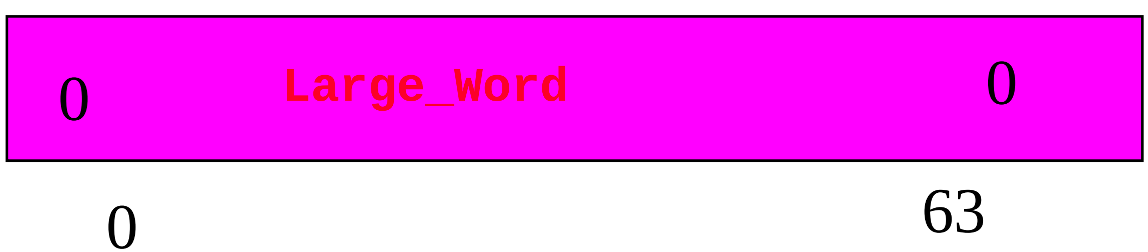
```
type 2D_FFT is array
  ( 1 to 128, 1 to 128 ) of real ;
variable X_Ray_FFT : 2D_FFT :=
  ( ( 60, 68 ) | ( 62, 67 ) | ( 67, 73 )
  | ( 60, 60 )  => 1.0 , others 0.0 ) ;
```

Array Operations

- One-Dimensional Arrays of Bit or Boolean
  - Element by element AND, OR, NAND, NOR, XOR, XNOR *can be done on array*

```
type Large_Word is array
            ( 63 downto 0 ) of bit ;

variable Samp_1 , Samp_2 : Large_Word
            ( 0 to 63 => ‘0’ ) ;
```



Array Operations, *e.g.*,

```
constant Bit_Mask : Large_Word
              ( 8 to 15 => '1' ) ;

Samp_2 := Samp_1 and Bit_Mask ;
```

Bits from 8 to 15 are
AND-ed with Bit_Mask

Array Operations

– Complement of elements of a single array, **NOT**

```
Samp_2 := not Samp_1 ;
```

Array Operations

- One-Dimensional Arrays Can Be Shifted and Rotated
  - Shift
    - Logical: Shifts and fills with zeros
    - Arithmetic: Shifts and fills with copies from the end being vacated
  - Rotate
    - Shifts bits out and back in at other end

Array Operations, *e.g.*,

Shift left logic

```
B" 1010_1100 " sll 4 == B" 1100_0000 "

B" 1010_1100 " sla 4 == B" 1100_0000 "

B" 1010_1100 " sra 4 == B" 1111_1010 "

B" 1010_1100 " rol 4 == B" 1100_1010 "
```

Shift right arithmetic

Rotate left

Array Operations

- One-Dimensional Arrays Can Be Operated on by Relational Operators,

  `=` , `/=` , `<` , `<=` , `>` , `>=`

  – Arrays need not be of the same length
  – Arrays must be of same type

Array Operations

- Concatenation Operator, **&**
  - Can combine array and scalar

```
B" 1010_1100 " & B" 1100_0000 " ==
           B" 1010_1100_1100_0000 "

B" 1010_1100 " & '1' == B" 1010_1100_1 "
```

Array Type Conversions

- One Array Type Can Be Converted to Another If:
  - Same element type
  - Same number of dimensions
  - Same index types

Array Type Conversions, *e.g.*,

Example

```
subtype name is string ( 1 to 20 ) ;
type display_string is array ( integer
  range 0 to 19 ) of character ;
variable item_name : name ;
variable display : display_string ;
display  :=  display_string ( item_name ) ;
```

Array Aggregate, *e.g.*,

- Assignments Can Be Made From a Vector to an Aggregate of Scalars or Vice-Versa.

```
type Sensor_Status is array
     ( Stdby, On, Off ) of time ;

variable Stdby_Time, On_Time, Off_Time :
  time ;
```

Array Aggregate, *e.g.*,

```
Variable FLIR_Status :
 Sensor_Status :=  ( 0 sec ,
                     0 sec ,
                     0 sec ) ;

( Stdby_Time,
  On_Time,
  Off_Time )    := Flir_Status ;
```

Records

- Collections of Named Elements of Possibly Different Types.
- To Refer to a Field of a Record Object, Use a Selected Name.

Records

- Aggregates Can Be Used to Write Literal Values for Records.

- Positional and Named Association Can Be Used
  - Record field names being used in place of array index names.

Record *e.g.*,*

```
type instruction is
 record
  op_code          : processor_op ;
  address_mode     : mode ;
  operand1, operand2 :
                   integer range 0 to 15 ;
  end record ;
```

*Ashenden, VHDL cookbook

End of Lecture