Signals & Data-Types

Introductory VHDL Methodology

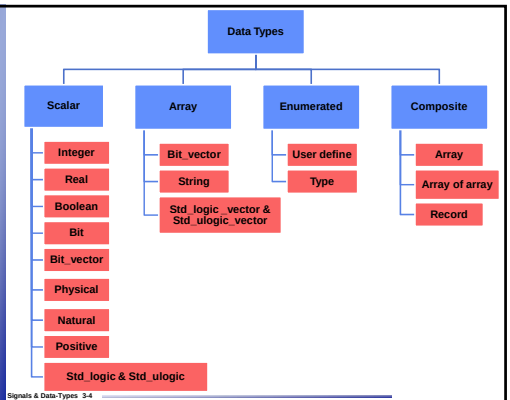
Objectives

After completing this module, you will be able to…

- Declare ports and signals using appropriate data-types
- Define all possible values for each data-type
- Declare ‘array’ for composite data-types
- Assign to ‘array’ or ‘scalar’ object
- Create and use ‘enumerated’ data-types

Data-Types

- **Data-types are very important in VHDL. A given data-type allows only values within its range to be applied. Each object (signal, variable, constant) or port must have its type defined when declared**
- **VHDL is considered to be a strongly typed language. Connected signals must be of the same type**
- **The wide range of data-types available provides both flexibility in hardware modeling and built-in error checking to ensure signal compatibility in large and complex models. This checking exists for behavioral simulation, not RTL/gate level implementation**



Standard Data Types in package STANDARD

Type	Range of Values
Integer	-2,147,483,647 to +2,147,483,647
Real	-1.0E+38 to +1.0E+38
Boolean	(TRUE, FALSE)
Character	Defined in package STANDARD
Bit	‘0’ , ‘1’
Bit_vector	Array with each element of type bit
Time	Fs, ps, ns, us, ms, sec, min, hr
String	Array with each element of type character
Natural	0 to the maximum integer value in the implementation
Positive	1 to the maximum integer value in the implementation

Signals & Ports

```
signal   abus , bbus : bit_vector (3 downto 0);
signal   int_a, int_b : integer;
signal   real_a, real_b : real;
...
begin
        abus <= bbus;        ✓
        abus <= int_a;       ✗
        real_a <= real_b;    ✓
        int_b <= real_b;     ✗
        …
```

- **Data-types must match on signal assignments**

Scalar Data-Types

- **Scalar data-types are single values**
  - *In VHDL, this class includes:*
    - Bit
    - Boolean
    - Integer
    - Real
    - Physical
    - Character
    - Std_logic (Std_ulogic)
    - Enumerated

Bit & Boolean

type **bit** is (**'0', '1'**) ;

```
architecture BEHAVE of MUX is
signal A,B,Sel, Z : bit ;
begin
    if Sel = '1' then
        Z <= A ;
    else
        Z <= B ;
    end if . . .
```

Type bit is helpful and concise for modeling hardware, but does not provide for high-impedance, unknown, don't care, etc.

type **boolean** is (**false, true**) ;

```
if Sel ='1',    if F >= G..
both yield boolean result
```

Type boolean is useful for modeling at the more abstract level. All relational operations return a value of type boolean

Integer & Real

type **integer** is range . . .

```
signal A : integer   range 0 to 7;
signal B : integer   range 15 downto 0 ;
```

Type integer allows for flexible, readily intuitive quantities and values in our models. It is essential to specify the range of any object of type integer, otherwise the language requires that synthesis tools generate a minimum 32 bit implementation

type **real** is range . . .

```
type CAPACITY is range -25.0 to 25.0 ;
signal Sig_1 : CAPACITY := 3.0 ;
```

Type real allows us to utilize floating point values and operations in our models. Since the range of real numbers is unlimited, we declare our type with the intended range of real values (Real Values are not synthesizable)

Physical

- **Physical types are used to quantify real world physical concepts and amounts, such as mass, length, time, etc**
- **A physical type must be defined in terms of its primary unit, any secondary units must be multiples of the primary**

type **time** is range
units
fs;
ps = 1000 fs;
ns = 1000 ps;
us = 1000 ns;
ms = 1000 us; **. . .**

***Time* is a pre-defined physical type in VHDL, it is important in our models for cell delays and other time based parameters**

```
constant Tpd : time := 3ns ;
. . .
Z <= A after Tpd ;
```


Std_logic (Std_ulogic)

- **Type Std_logic was developed from the MVL (Multi-Value Logic) system and provides for more detailed hardware modeling**
- **It supports different signal strengths, *don't-care* conditions and bussed structures with tri-state drivers**
- **(Defined in package std_logic_1164)**

```
type std_ulogic is ( 'U' ; -- Uninitialized
                     'X' ; -- Forcing Unknown
                     '0' ; -- Forcing Zero
                     '1' ; -- Forcing One
                     'Z' ; -- High Impedance
                     'W' ; -- Weak Unknown
                     'L' ; -- Weak Zero
                     'H' ; -- Weak One
                     '-' ; -- Don't Care )
```

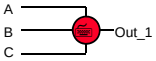
Note: type **bit** is limited to ('0', '1').

Std_logic Vs. Std_ulogic

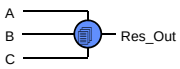
- **Both Std_logic and Std_ulogic contain the same set of same possible values. The difference is in implementation, the *u* in ulogic means *unresolved***
- **If we are using std_logic, and we wish to drive two or more signals to a common output, we may write a resolution function to indicate which driver is actually applied to the output**
- **Std_ulogic offers no such capability, but does provide a built-in means of error checking for inadvertent wire-o-ring of outputs**

```
signal A,B,C,Res_Out : std_logic ;
signal Out_1 : std_ulogic ;
```

```
Out_1 <= A ;
Out_1 <= B ;
Out_1 <= C ;
```




```
Res_Out <= A;
Res_Out <= B;
Res_Out <= C;
```



Std_logic Vs. Std_ulogic

```
signal   a, b, z : std_ulogic;
signal   abus : std_ulogic_vector , (3 downto 0);
signal   res_z: std_logic;
signal   res_zbus : std_logic_vector , (3 downto 0);
...
begin
        z <= a;                 ✓
        res_z <= a;             ✓
        a<= res_z;              ✗
        res_zbus <= abus;       ✓
        abus <= res_zbus;       ✗
        …
```

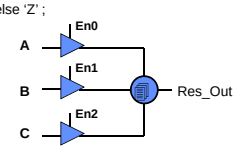
```
res_z  <= a;   ✓
res_z  <= b;   ✓

z <= a;        ✗
z <= b;        ✗
```


Signal Resolution

- **A single output may not have multiple drivers, to model a bussed (tri-state) output, use a conditional signal assignment and data_type std_logic**

```
signal  A,B,C,Res_Out : std_logic ;
```

```
Res_Out <= A when En0 = '1' else 'Z' ;
Res_Out <= B when En1 = '1' else 'Z' ;
Res_Out <= C when En2 = '1' else 'Z' ;
```



Enumerated

- **Enumerated types offer perhaps the most flexibility in abstract hardware modeling. User defined enumerated types allow values that are immediately recognizable and intuitively relevant to the operation of the model**
- **This capability makes our code more readable when describing state machines and complex systems**

```
type  My_State is  ( RST, LOAD, FETCH, WAIT, SHIFT ) ;
. . .
signal  STATE, NEXT_STATE : My_State ;
. . .
```

```
case  (STATE)  is
when  LOAD => . . .
if  COND_A  and  COND_B  then
NEXT_STATE  <=  FETCH ;
else  NEXT_STATE  <= WAIT ;
. . .
```


Composite Data-Types

- **Composite Data_types are groups of elements in the form of an array or record**
  - *(Bit_vector and Std_logic_vector are pre-defined composite types)*

```
signal  A_word : bit_vector  (3 downto 0) :=  "0011" ;
```

- **This represents four *bit* elements grouped together into an array. However, there is no pre-defined LSB or MSB interpretation, therefore this would not be read as 3, '3', or "3"**

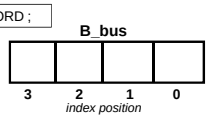
Arrays

- **Arrays are groups of elements, all of the same type!**

```
type  WORD is array (3 downto 0) of  std_logic ;
```

```
signal  B_bus :  WORD ;
```

What are the possible values for each element of the array?




```
signal  B_bus :  DATA ;
```

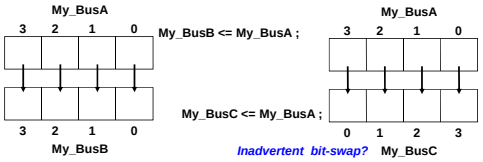
```
type  DATA is array (3 downto 0) of  integer range  0 to 9  ;
```

- **Remember to specify the integer range, to limit width of synthesized module**

Array Assignments

- **When assigning arrays, the following rules apply**
  1. *The arrays must be the same type*
  2. *The arrays must be the same length*
  3. *The assignment is positional, from left to right!*

```
signal  My_BusA, My_BusB:  bit_vector   (3 downto 0 ) ;
signal  My_BusC :  bit_vector   ( 0 to 3 ) ;
```

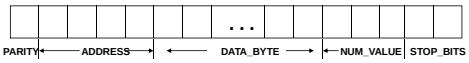


Inadvertent bit-swap?

Records

```
type  OPCODE is  record
PARITY  : bit;
ADDRESS : std_logic_vector ( 0 to 3 );
DATA_BYTE : std_logic_vector ( 7 downto 0 );
NUM_VALUE : integer range 0 to 6;
STOP_BITS : bit_vector (1 downto 0);
end record ;
. . .
signal  TX_PACKET, RX_PACKET  : OPCODE;
```

Records are groups of elements, that may be of different types



TX_PACKET

Aggregates

- **Aggregates are a convenient means of grouping both scalar and composite data-types for assignment**

```
signal  H_BYTE, L_BYTE: std_logic_vector (7 downto 0);
signal  DATA : std_logic_vector (15 downto 0);
signal  A, B, C, D  : std_logic;
signal  WORD : std_logic_vector (3 downto 0);
signal  TX_PACKET, RX_PACKET : OPCODE;
```

```
( H_BYTE, L_BYTE )  <=  DATA ;
    TX_PACKET  <= ( '1', "0011", "11101010",5, " 10" ) ;
        WORD  <= ( 2 => '1', 3 => D, Others => '0' ) ;
            DATA  <=  ( Others => '1') ;
                WORD <=  ( A, B, C, D ) ;
                    TX_PACKET. ADDRESS  <= ( "0011" ) ;
```

The total number of elements on both sides of any assignment must match, "Others" can be used as a default assignment, regardless of the array size

Review Questions

Given:

```
signal  A_Bus, B_Bus: std_logic_vector (7 downto 0) ;
signal  Data_Word : std_logic_vector (15 downto 0) ;
signal  A, B, C, D,  : std_logic ;
signal  Nibble : std_logic_vector (3 downto 0) ;
type    My_State is ( S1, S2, S3, S4) ;
signal  State, Next_State : My_State ;
signal  D_Bus, E_Bus : bit_vector ( 0 to 7) ;
```

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

```
A_Bus <=  B_Bus ;
Data_Word <= A_Bus  ;
Nibble <=  ( A, B, C, 1 ) ;
State <=  "0011" ;
B_Bus <= (7=>A, 5=>B, 3=>C,  Others => '1' ) ;
Next_State <= (S1, S2) ;
Data_Word <= (Nibble, Nibble, A_Bus) ;
A_Bus <= E_Bus ;
```


Answers

Given:

```
signal  A_Bus, B_Bus: std_logic_vector (7 downto 0) ;
signal  Data_Word : std_logic_vector (15 downto 0) ;
signal  A, B, C, D,  : std_logic ;
signal  Nibble : std_logic_vector (3 downto 0) ;
type    My_State is ( S1, S2, S3, S4) ;
signal  State, Next_State : My_State ;
signal  D_Bus, E_Bus : bit_vector ( 0 to 7) ;
```

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

```
A_Bus <=  B_Bus ;          OK, size and type are the same
Data_Word <= A_Bus  ;      BAD, size mismatch, although type matches
Nibble <=  ( A, B, C, 1 ) ;  BAD, the 1 without single quotes is read as integer
State <=  "0011" ;         BAD, only the literals of the type 'My_State' are valid
B_Bus <= (7=>A, 5=>B, 3=>C,  Others => '1' ) ;  OK, size and type match
Next_State <= (S1, S2) ;   BAD, single target can only have one value
Data_Word <= (Nibble, Nibble, A_Bus) ;          OK, size and type match
A_Bus <= E_Bus ;           BAD, type mismatch, although size matches
```


Summary

- **Each object and port must have its type defined**
- **VHDL provides scalar and composite data-types**
- **Enumerated types can be used to enhance code readability**
- **Aggregates assignments can be made for arrays and records**
- **Types on connecting signals must match**