VHDL Data Types
What is a “Data Type”?

● This is a classification objects/items/data that defines the possible set of values which the objects/items/data belonging to that type may assume.

● E.g. (VHDL) integer, bit, std_logic, std_logic_vector

● Other languages (float, double, int, char etc)
VHDL Data Types

• Predefined Data Types
  • Specified through the IEEE 1076 and IEEE 1164 standards
  • The IEEE Standard 1076 defines the VHSIC Hardware Description Language or VHDL
    – Developed by Intermetrics, IBM and Texas Instruments for United States Air Force.
    – 1076-1987 was the first version
VHDL Data Types

- Data Types
  - Scalar
    - Integer
    - Real
    - Boolean
    - Bit
    - Bit_vector
    - Physical
    - Natural
    - Positive
    - Std_logic & Std_ulogic
  - Array
    - Bit_vector
    - String
    - Std_logic_vector & Std_ulogic_vector
  - Enumerated
    - User define
    - Type
  - Composite
    - Array
    - Array of array
    - Record
VHDL Data Types

• Package standard of library std (Included by default):
  
  • bit type (0, 1)
  • bit vectors (group of multi-bit signal → bus)

• Example
  
  - SIGNAL x: BIT;
  - SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
  - SIGNAL w: BIT_VECTOR (0 TO 7);
  -

• Signal assignment operator <=
  
  - x <= '1';
  - y <= "0111";
  - w <= "01110001";
VHDL Data Types

- Package standard of library std (Included by default):
  - BOOLEAN (TRUE, FALSE)
    - Example
      - variable VAR1: boolean := FALSE;
  - INTEGER (32 bit, -2,147,483,647 to +2,147,483,647)
    - Example
      - SIGNAL SUM: integer range 0 to 256 :=16;
  - REAL (from -1.0E38 to +1.0E38)
    - Example
      - constant Pi : real := 3.14159;
The IEEE Standard 1164

- Introduce Multivalue Logic (std_logic_1164) Packages
- The primary data type std_ulogic (standard unresolved logic) consists of nine character literals in the following order:

1. 'U' – uninitialized (default value)
2. 'X' - strong drive, unknown logic value
3. '0' - strong drive, logic zero
4. '1' - strong drive, logic one
5. 'Z' - high impedance (for tri-state logic)
6. 'W' - weak drive, unknown logic value
7. 'L' - weak drive, logic zero
8. 'H' - weak drive, logic one
9. '-' - don't care

- std_ulogic and its subtype (std_logic, std_logic_vector, std_ulogic_vector) values can be categorized in terms of their state and strength (forcing, weak and high impedance.)
- Weak strength is used for multi-driver inputs catering for pullup/pulldown
## VHDL Data Types

- **std_ulogic data type** possible values and corresponding strength

<table>
<thead>
<tr>
<th>Data Value</th>
<th>State</th>
<th>Strength</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Uninitialised</td>
<td>None</td>
<td>Default value before simulation.</td>
</tr>
<tr>
<td>X</td>
<td>Unknown</td>
<td>Forcing</td>
<td>Represents driven signals whose value cannot be determined as 1 or 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Forcing</td>
<td>Represents signals from active output drivers</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Forcing</td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>None</td>
<td>High Impedance</td>
<td>Represents output of tri-state buffer when not enabled.</td>
</tr>
<tr>
<td>W</td>
<td>Unknown</td>
<td>Weak</td>
<td>Represents signals from resistive drivers e.g. pull-up and pull-down resistors</td>
</tr>
<tr>
<td>L</td>
<td>0</td>
<td>Weak</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>1</td>
<td>Weak</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Don't care</td>
<td>None</td>
<td>Allows synthesiser to decide whether to assign a 0 or a 1 for minimum synthesised logic circuit.</td>
</tr>
</tbody>
</table>
VHDL Data Types

- **std_ulogic**
  - Is an unresolved data type
  - Declared in package STD_LOGIC_1164 of library IEEE.
  - All data signals are of unresolved type by default.
  - Unresolved data type signals cannot be driven by more than one driver/sources. (adding multiples sources will result in compiler error).
  - Helps checking that designer has not accidentally assigned two sources to a signal.
• Resolved Data Types
  • Always declared with a resolution function (within its library).

• Resolution function defines all possible combinations of one or more source values and the correspond resolved value (result).
VHDL Data Types

- std_logic (this is a resolved data type)
  - A subtype of std_ulogic
  - Declared in package STD_LOGIC_1164 of library IEEE as
    `subtype std_logic is resolved std_ulogic;`
  - Specified a resolution function called “resolved”
VHDL Data Types

- `std_logic` resolution table

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
<th>W</th>
<th>L</th>
<th>H</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>Z</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>Z</td>
<td>W</td>
<td>L</td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>L</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td>W</td>
<td>L</td>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>H</td>
<td>W</td>
<td>W</td>
<td>H</td>
<td>X</td>
</tr>
<tr>
<td>-</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
VHDL Data Types

• std_logic declaration examples
  
  • SIGNAL x: STD_LOGIC;
  
  • SIGNAL y: STD_LOGICVECTOR (3 DOWNTO 0) := "0001";
Arrays are collections of objects of the same type. Can be 1D (1 dimensional) or 2D (2 dimensional) arrays. Higher dimensional arrays are not synthesizable.

- Scalar
- 1D Array
- 1Dx1D array (Array of vectors)
- 2D array

There are no pre-defined 2D or 1Dx1D arrays; have to be defined by designer.
VHDL Data Types: Arrays

- **Defining VHDL Arrays**
  - First define a new data type
  - Second declare a signal, variable or constant of the defined data type.

- **General Format of Array definition**

  ```vhdl
  TYPE type_name IS ARRAY (specification) OF data_type;
  SIGNAL signal_name: type_name [:= initial_value];
  ```
VHDL Data Types: Arrays

• Example:

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
```
- Defines a row (1D array) (data type) with of seven STD_LOGIC values with MSB on left.

```
TYPE matrix IS ARRAY (0 TO 3) OF row;
```
- Defines an 1Dx1D ARRAY (matrix) data type containing 4 row defined in previous line.

```
SIGNAL x: matrix;
```
- Defines 1Dx1D signal of type matrix as defined in previous line.
VHDL Data Types: Arrays

- Example: 1Dx1D Array (of vectors) --- Alternative method
  
  ```vhd
  TYPE matrix IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
  ```

- Example: 2D Array Data type
  
  ```vhd
  TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
  ```
  
  - Array construction is not based on vectors, but rather entirely on scalars.
  - It is a 2 dimensional array of scalars
VHDL Data Types: Array Assignments

- **Type Definition:**
  
  ```vhdl
  TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;      -- 1D array
  TYPE array1 IS ARRAY (0 TO 3) OF row;                -- 1Dx1D array
  ```

- **Signal Declaration:**
  
  ```vhdl
  SIGNAL x: row;
  SIGNAL y: array1;
  ```

- **Scalar Signal (array) assignment:**
  
  ```vhdl
  x(0) <= y(1)(2);
  ```
  
  - Note the two pairs of parentheses since y is a 1Dx1D array.
VHDL Data Types: Array Assignments

- **Type Definition:**
  
  ```vhdl
  TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
  -- 1Dx1D
  
  TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
  -- 2D array
  ```

- **Signal Declarations:**
  
  ```vhdl
  SIGNAL v: array2;
  SIGNAL w: array3;
  ```

- **Scalar Signal Assignments:**
  
  ```vhdl
  x(1) <= v(2)(3);
  x(2) <= w(2,1);
  ```

  - Single pair of parentheses since w is 2D array
**VHDL Data Types: Array Assignments**

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
TYPE array1 IS ARRAY (0 TO 3) OF row;
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
```

- **Signal Declaration;**

  ```
  SIGNAL x: row;
  SIGNAL y: array1;
  SIGNAL v: array2;
  SIGNAL w: array3;
  ```

- **Scalar Signal Assignments:**

  ```
  y(1)(1) <= x(6);
  y(2)(0) <= v(0)(0);
  y(0)(0) <= w(3,3);
  w(1,1) <= x(7);
  w(3,0) <= v(0)(3);
  ```
VHDL Data Types: Array Assignments

- Vector Signal Assignments

  \[
  \text{TYPE row IS ARRAY (7 DOWNTO 0) OF STD\_LOGIC;} \\
  \text{TYPE array1 IS ARRAY (0 TO 3) OF row;} \\
  \text{TYPE array2 IS ARRAY (0 TO 3) OF STD\_LOGIC\_VECTOR(7 DOWNTO 0);} \\
  \text{TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD\_LOGIC;} \\
  \]

- Signal Declaration;

  \[
  \text{SIGNAL x: row;} \\
  \text{SIGNAL y: array1;} \\
  \text{SIGNAL v: array2;} \\
  \text{SIGNAL w: array3;} \\
  \]

- Legal Assignments

  \[
  x \leftarrow y(0); \\
  y(1)(7 \text{ DOWNTO 3}) \leftarrow x(4 \text{ DOWNTO 0}); \\
  v(1)(7 \text{ DOWNTO 3}) \leftarrow v(2)(4 \text{ DOWNTO 0}); \\
  \]
VHDL Data Types: Array Assignments

- **Vector Signal Assignments**

  ```vhdl
  TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
  TYPE array1 IS ARRAY (0 TO 3) OF row;
  TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
  TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
  ```

- **Signal Declaration**;

  ```vhdl
  SIGNAL x: row;
  SIGNAL y: array1;
  SIGNAL v: array2;
  SIGNAL w: array3;
  ```

- **Why are the following assignments illegal?**

  ```vhdl
  x <= v(1);
  x <= w(2);
  x <= w(2, 2 DOWNTO 0);
  v(0) <= w(2, 2 DOWNTO 0);
  v(0) <= w(2);
  y(1) <= v(3);
  w(1, 5 DOWNTO 1) <= v(2)(4 DOWNTO 0);
  ```
### VHDL OPERATORS

- **Logical operators**

<table>
<thead>
<tr>
<th>Logical operation</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>AND</td>
<td>Z &lt;= (A AND B);</td>
</tr>
<tr>
<td>NAND</td>
<td>NAND</td>
<td>Z &lt;= (A NAND B);</td>
</tr>
<tr>
<td>NOR</td>
<td>NOR</td>
<td>Z &lt;= (A NOR B);</td>
</tr>
<tr>
<td>NOT</td>
<td>NOT</td>
<td>Z &lt;= NOT (A);</td>
</tr>
<tr>
<td>OR</td>
<td>OR</td>
<td>Z &lt;= (A OR B);</td>
</tr>
<tr>
<td>XNOR</td>
<td>XNOR</td>
<td>Z &lt;= (A XNOR B);</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR</td>
<td>Z &lt;= (A XOR B);</td>
</tr>
</tbody>
</table>
VHDL OPERATORS

- Arithmetic operators

<table>
<thead>
<tr>
<th>Arithmetic operation</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>Z &lt;= A + B;</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>Z &lt;= A - B;</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>Z &lt;= A * B;</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>Z &lt;= A / B;</td>
</tr>
<tr>
<td>Exponentiating</td>
<td>**</td>
<td>Z &lt;= 4 ** 2;</td>
</tr>
<tr>
<td>Modulus</td>
<td>MOD</td>
<td>Z &lt;= A MOD B;</td>
</tr>
<tr>
<td>Remainder</td>
<td>REM</td>
<td>Z &lt;= A REM B;</td>
</tr>
<tr>
<td>Absolute value</td>
<td>ABS</td>
<td>Z &lt;= ABS A;</td>
</tr>
</tbody>
</table>
VHDL OPERATORS

- Relational operators

<table>
<thead>
<tr>
<th>Relational operation</th>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal to</td>
<td>=</td>
<td>If (A = B) Then</td>
</tr>
<tr>
<td>Not equal to</td>
<td>/=</td>
<td>If (A /= B) Then</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>If (A &lt; B) Then</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>If (A &lt;= B) Then</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>If (A &gt; B) Then</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>If (A &gt;= B) Then</td>
</tr>
</tbody>
</table>
**VHDL Reserved Words**

- Reserved words cannot be used by designers for identifiers such as variables, signal names, etc.

<table>
<thead>
<tr>
<th>abs</th>
<th>file</th>
<th>of</th>
<th>then</th>
</tr>
</thead>
<tbody>
<tr>
<td>after</td>
<td>for</td>
<td>open</td>
<td>to</td>
</tr>
<tr>
<td>all</td>
<td>generic</td>
<td>or</td>
<td>transport</td>
</tr>
<tr>
<td>and</td>
<td>if</td>
<td>others</td>
<td>type</td>
</tr>
<tr>
<td>architecture</td>
<td>in</td>
<td>out</td>
<td>until</td>
</tr>
<tr>
<td>array</td>
<td>inertial</td>
<td>package</td>
<td>use</td>
</tr>
<tr>
<td>begin</td>
<td>inout</td>
<td>port</td>
<td>variable</td>
</tr>
<tr>
<td>case</td>
<td>is</td>
<td>process</td>
<td>wait</td>
</tr>
<tr>
<td>component</td>
<td>library</td>
<td>rem</td>
<td>when</td>
</tr>
<tr>
<td>configuration</td>
<td>linkage</td>
<td>report</td>
<td>while</td>
</tr>
<tr>
<td>constant</td>
<td>loop</td>
<td>rol</td>
<td>with</td>
</tr>
<tr>
<td>downto</td>
<td>mod</td>
<td>ror</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td>nand</td>
<td>select</td>
<td>xnor</td>
</tr>
<tr>
<td>elsif</td>
<td>next</td>
<td>signal</td>
<td>xor</td>
</tr>
<tr>
<td>end</td>
<td>nor</td>
<td>sla</td>
<td></td>
</tr>
<tr>
<td>entity</td>
<td>not</td>
<td>sll</td>
<td></td>
</tr>
</tbody>
</table>

- Note: `sra` and `srl` are not standard reserved words in VHDL, but they might be present in specific contexts or implementations.
Data Types:
Advanced Topics
VHDL Data Types

- Package std_logic_arith of library IEEE:
  - Defines SIGNED and UNSIGNED data types, plus several data conversion functions, like:
    - `conv_integer(p),`
    - `conv_unsigned(p, b),`
    - `conv_signed(p, b),` and
    - `conv_std_logic_vector(p, b).`
  - Allow arithmetic operations
  - Data conversion to be discussed in later slides
VHDL Data Types

• Packages std_logic_signed and std_logic_unsigned of library IEEE:

  • Contain functions that allow operations with STD_LOGIC_VECTOR data to be performed as if the data were of type SIGNED or UNSIGNED, respectively.
User Defined VHDL Data Types

• User Defined Integer Data Types
  • Subtype of Integer
  • Examples
    – TYPE integer IS RANGE -2147483647 TO +2147483647;
    – TYPE my_integer IS RANGE -32 TO 32;
    – -- A user-defined subset of integers.
    – TYPE student_grade IS RANGE 0 TO 100;
    – -- A user-defined subset of integers or naturals.
    – TYPE natural IS RANGE 0 TO +2147483647;
User Defined VHDL Data Types

- User Defined ENUMERATED Data Types
  - Data type consisting of a set of named values.
  - Examples
    - **TYPE** bit **IS** ('0', '1');
    - **TYPE** my_logic **IS** ('0', '1', 'Z');
      - This is the pre-defined type BIT
    - **TYPE** bit_vector **IS** ARRAY (NATURAL RANGE <>) **OF** BIT;
      -- This is the pre-defined type BIT_VECTOR.
      NATURAL RANGE <>, on the other hand, indicates that the only restriction is that the range must fall within the NATURAL range.
User Defined VHDL Data Types

- User Defined ENUMERATED Data Types
  - More Examples
    - TYPE state IS (idle, forward, backward, stop);
      -- An enumerated data type, typical of finite state machines.
      - Two bits will be used to encode this data type values.
      - Idle will be the default value
    - TYPE color IS (red, green, blue, white, black);
      -- Another enumerated data type.
      - Three bits will be used for encoding this data type.
      - Red will be the default value
Like Arrays, arrays records are collections of objects.
Unlike arrays, records can contain objects of different data types.
Example

```
TYPE birthday IS RECORD
  day: INTEGER RANGE 1 TO 31;
  month: month_name; -- month_name datatype should be pre-defined
END RECORD;
```
VHDL Data Types: Signed and Unsigned Types

- Defined in the **STD_LOGIC_ARITH** package of the IEEE library
- For arithmetic operations.
- Signal Declaration Examples
  ```vhdl
  SIGNAL x: SIGNED (7 DOWNTO 0);
  SIGNAL y: UNSIGNED (0 TO 3);
  ```
- Syntax is similar to that of **STD_LOGIC_VECTOR** not like integers
- An UNSIGNED value is a number never lower than zero.
  
  - Unsigned “0101” = the decimal 5
  - Unsigned “1101” signifies 13.
  - Signed “0101” = the decimal 5
  - Signed “1101” signifies -3 (Two's complement)
VHDL Data Types:
Signed and Unsigned Types

- Operations Example

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

SIGNAL a: IN SIGNED (7 DOWNTO 0);
SIGNAL b: IN SIGNED (7 DOWNTO 0);
SIGNAL x: OUT SIGNED (7 DOWNTO 0);
SIGNAL u: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL v: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);

x <= a + b; -- legal
x = a AND b; -- illegal
y = a + b;  -- illegal
y = a AND b;  -- legal
```
VHDL Data Types:
Signed and Unsigned Types

- std_logic_signed and std_logic_unsigned packages allows both logical and arithmetic operations

Example:

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
...
SIGNAL a: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL x: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
...
v <= a+ b; -- legal
w <= a AND b; -- legal
```
Direct operation between different data types is illegal in VHDL
Solution!!!! = Data conversion
Examples:

```vhdl
TYPE long IS INTEGER RANGE -100 TO 100;
TYPE short IS INTEGER RANGE -10 TO 10;
SIGNAL x : short;
SIGNAL y : long;
...
y <= 2*x + 5;       -- error, type mismatch
y <= long(2*x + 5); -- OK, result converted into type long
```
Type Conversion

- Data conversion defined in STD_LOGIC_ARITH
- `conv_integer(p)`:
  - Converts a parameter `p` of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an INTEGER value.
  - Notice that STD_LOGIC_VECTOR is not included.

- `conv_unsigned(p, b)`: Converts a parameter `p` of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to an UNSIGNED value with size `b` bits.

- `conv_signed(p, b)`: Converts a parameter `p` of type INTEGER, UNSIGNED, SIGNED, or STD_ULOGIC to a SIGNED value with size `b` bits.
Type Conversion

- **conv_std_logic_vector(p, b):**
  - Converts a parameter p of type INTEGER, UN-SIGNED, SIGNED, or STD_LOGIC to a STD_LOGIC_VECTOR value with size b bits.

- Example:

  ```
  LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.std_logic_arith.all;
  ...
  SIGNAL a: IN UNSIGNED (7 DOWNTO 0);
  SIGNAL b: IN UNSIGNED (7 DOWNTO 0);
  SIGNAL y: OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
  ...
  y <= CONV_STD_LOGIC_VECTOR ((a+b), 8);
  ```

  - a+b is converted from UNSIGNED to an 8-bit STD_LOGIC_VECTOR value, then assigned to y.
VHDL Data Type: Example

- Four Bit Adder       ------ Solution 2: in/out=SIGNED ---------

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
------------------------------------------
ENTITY adder1 IS
    PORT ( a, b : IN SIGNED (3 DOWNTO 0);
            sum : OUT SIGNED (4 DOWNTO 0));
END adder1;
------------------------------------------
ARCHITECTURE adder1 OF adder1 IS
    BEGIN
        sum <= a + b;
    END adder1;
```
Four Bit Adder------ Solution 2: out=INTEGER ---------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
---------------------------------------------------------------------
ENTITY adder2 IS
  PORT ( a, b : IN SIGNED (3 DOWNTO 0);
         sum : OUT INTEGER RANGE -16 TO 15);
END adder2;
---------------------------------------------------------------------
ARCHITECTURE adder2 OF adder2 IS
  BEGIN
    sum <= CONV_INTEGER(a + b);
END adder2;