Lecture #8

In this lecture we will cover the following material:

The standard package,
The std_logic_1164

- Objects & data Types (Signals, Variables, Constants, Literals, Character)
- Types and Subtypes (Scalar & Composite types).

Library Clauses

- A library clause declares a name that denotes a library. The name can be any legal identifier.
- A library clause can declare more than one library name by declaring all the library names separated by comma.

- Examples for user defined libraries:

library **Designs;** ***-- Declaration for the Designs library***
use **Designs.*all*;** ***-- Use statement grants visibility of declarations inside the library***

--Examples for predefined libraries

- Every design assumes the following 2 statement:

library **STD;**
use **STD.STANDARD.*all*;** ***-- Package STANDARD inside library STD***

-Every design assumes " Library WORK" clause

Other Packages

- In VHDL there are no arithmetic operators for types that require bit operation.

- Most VHDL Simulators provide arithmetic packages to perform arithmetic operations using STD_LOGIC_1164 types.

- Some companies provide many math packages such as floating point.
- Some synthesis tool provide special software that maps some functions such as adders, subtractors, multipliers, registers, counters etc. to ASIC library cells.
- Most synthesis companies nowadays provide component packages for a variety of cells such as power and ground pads, I/O buffers, clock driver, 3-state pads etc.
  This is usually done by 2 stages first technology-independent codes are generated and then the components are mapped to primitives for technology dependent libraries after synthesis.

STANDARD PACKAGE

```
-- the types declared in  the standard package are used similar to the way
that the reserved words are used
package subset_STANDARD is
type BOOLEAN is (FALSE, TRUE);
type BIT is ( '0' , ' 1' );
type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
subtype NATURAL is INTEGER range 0 to INTEGER' HIGH;
subtype Positive is INTEGER range 1 to INTGER' HIGH;
type BIT_VECTOR is array (Natural range <> ) of BIT;
type STRING is array (Positive range <>) of CHARACTER;
subtype DELAY_LENGTH is TIME range 0 fs to TIME' HIGH;
-- A STRING array must have a positive index.
-- The type TIME is declared in the STANDARD  package as:
type TIME is range implementation_defined
Units fs;
     ps = 1000 fs;
     ns = 1000 ps;
     us = 1000 ns;
     ms =1000 us;
     sec = 1000 ms;
end units;
end subset_STANDARD;
```

STD_LOGIC_1164 .....

- VHDL does not have a in build logic value system. The STANDARD package pre-defines the type **BIT** with two logic values of **'0'** and **'1'**.
- Normally additional values of **'X'** (unknown) and **'Z'** (high impendence) are also needed.
- CMOS circuits require more levels and strengths.
- The STD_LOGIC_1164 package includes functions to perform logical, shift, resolution and conversion functions.

  To access this package, the following statements have to be included:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

- The STD_LOGIC_1164 package contains definitions for a nine-value logic system.

- The STD_ULOGIC is defined in STD_LOGIC_1164.

STD_LOGIC_1164

This type is used in most structural designs

```
package subset_STD_1164 is
-- defines the 9-vlaue logic system
type STD_ULOGIC  is
        ( ‘ U ’,                    -- Un-initialized
          ‘ X ’,                    -- Forcing unknown
          ‘ 0 ’,                    -- Forcing zero
          ‘ 1 ’,                    -- Forcing one
          ‘ Z ’,                    -- High Impedance
          ‘ W ’,                    -- Weak Impedance
          ‘ L ’,                    -- Weak zero
          ‘ H ’,                    -- Weak 1
          ‘ _ ’, );                  -- Don’t Care

type STD_ULOGIC_VECTOR is array (Natural range <> ) of STD_ULOGIC;
function resolved ( S: STD_ULOGIC_VECTOR) return STD_ULOGIC;
subtype STD_LOGIC is resolved STD_ULOGIC;
type STD_LOGIC_VECTOR is array(NATURAL range <>) of STD_LOGIC;
function rising_edge (signal S: STD_ULOGIC) return BOOLEAN;
             falling_edge (signal S: STD_ULOGIC) return BOOLEAN;
end subset_STD_1164;
```

STD_LOGIC Example

New type

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity control_buffer is
            port (A,OE : in std_logic;
                  Y: out std_logic);
end control_buffer;
architecture arch1  of control_buffer is
signal n: std_logic;          - - Internal Signal Declaration
begin
            n <= not A ;
            Y <= n when OE =  ‘0’  else  ‘Z’;

end arch1;
```

See new logic value

Use Clause

```
-- Place the library & use statements in front every entity
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity package_1
….
end package_1;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity NO_1
….
end NO_1;
library IEEE;
use IEEE.STD_LOGIC_1164.all;
Architecture structure of No_1 is
….
end structure
use IEEE.STD_LOGIC_1164.all;
entity NO_2
….
end NO_2;
```

Makes the entity visible to logic type used

Not essential

Signal Declarations

signal identifier (Label) *: subtype* (bus/register) [<= expression]

- A signal has a history of values. It has a past, present and future value

-The expression specifies the initial value of the signal at simulation time. The expression is evaluated at each elaboration of the signal.

- The default initial value for a signal of a scalar type T is T' left.

- Signal can be only declared in concurrent descriptions.

```
signal Sig_1: Bit;
signal Address_Bus: BIT_VECTOR (31 downto 0);
 signal Bus: tristate;
 signal A, B, C : BIT_VECTOR (0 to 2);
 signal A: BIT<= '0' ;
```

Initialized

Global Signals

Concordia University

```
-- to make a signal global,  declare it in a package and make the
--package visible to the entities that requires it

package GLOBAL_1 is

   signal  M: std_logic;
        ¦
        ¦

end GLOBAL_1;


use WORK.GLOBAL_1. M ;  -- Accessing a global signal
architecture DATA of MULT is
begin
        ¦
        ¦

end DATA;
```

M can be used here without declaration

Example of Signal assignment

A Concurrent signal assignment assigns a new value to the target signal whenever any of the signals on the right hand side change:
Example:

```
architecture SIG_ASSIGN of Half_Adder is
begin
        SUM   <= A xor B;
        CARRY <= A and B;
end SIG_ASSIGN;
```

Example of Signal assignment with delay

A signal assignment may have a delay specified:

```
architecture DIFFERENT of SIG is
        constant A_Delay : time := 20ns;
begin
        SUM   <= A xor B after A-Delay -1 ns;
        CARRY <= A and B after 9 ns;
end DIFFERENT;
```

NOTE: **Synthesis tools usually treat a signal assigned with a concurrent statement as combinational logic. Delays are ignored.**

Variable Declarations

variable identifier (Label): subtype (bus/register) [:= expression]

- A variable has only one value (current).

- The expression specifies the initial value. The expression is evaluated at each elaboration of the variable.

- The default initial value for a scalar type variable T is T' left.

- It is a sequential statement and used inside processes, procedures and functions.

-The value of the right hand side is copied *immediately*

variable A : BIT := ' 0' ;

variable R20 : Integer;

variable *CHA : character := 'Z ';*

variable *BV: Bit Vector (0 to 3) := "0101 ";*

variable *volts : real := 2.67;*

Variables are synthesizeable if their type is acceptable by the synthesis tools.

variable/Signal conversion

Delay can not be assigned to variables

Assignments may be made from signals to variables and vice-versa, but types have to match:

```
process (A, B, C, DIFF)
  variable Z : integer range 0 to 7;
begin
  if DIFF = '1' then
    Z := B;
  else
    Z := C;
  end if;
    Y <= A * B + Z;  -- Y has been declared as signal
end process;
```

Constant Declarations

constant identifier (Label) : subtype (bus/register) [:= expression]

-- Example

constant my_weight: weight := 65 Kg ;

constant period : TIME := 20ns ;

constant Pai : real := 3.14 ;

constant All_Ones: vector_4 := “1111” ;

-- All subtypes are pre-defined.

Constants can be synthesized if their type is acceptable by the synthesis tool.

Literals Examples

--Literals are the values given to VHDL objects

--Examples of decimal literals

I1:= 150000; or I2:= 150_000; 0r I3:= 15 e4; .. All are the same

R1:=1500.0; or R2:= 1_500.0; or R3:-1.5e3;

--examples of based literals

--The base can be either hexadecimal, octal or binary

For example Integer literal of value 255 can be expressed as:

In binary as	2#1111_1111 or
In Octal as	O#773# or
In Hexadecimal as	16 # FF#

--Bit string-literals

X“FFE” = B”1111-1111-1110”

O“765” = B”111_110_101”

(Where X is hexadecimal, O is Octal)

Character, String literals

Character literals

Between two ‘ ’ (comas) characters:

‘B’ ‘b’ ‘?’

String literals

- Text inside two “ ” (double comas) characters:

- A string must be fitting on a single line.

- Concatenation operation is used to combine strings longer than one line.

“When First line is full then we can continue be placing an “ &”

“ we have continued on the second line ”

Types, Subtypes

-Scalar Types

- Single numerical value
- Enumerated

} → Integer, real, physical

- Composite Types

- Arrays
- Records

} → Collection of values

Arrays ← Of the same type

Records ← Of different types

- Access Types

- Pointers

- File Types

- Objects that contain a sequence of values for reference

Types, Subtypes Declarations

- A type is defined by a set of values and a set of actions/operations.

Example: *type* Nibble_value *is range* 0 *to* 15 ;

- A further constraint might be applied to values of a given type.
Then a subtype is a type along with the added constraint.
Example: *subtype* COUNT *is* INTEGER *range* 1 *to* 255 ;
***subtype* SEVEN_BIT *is* COUNT *range* 1 *to* 127 ;**

- **Constraint is checked during each simulation.**

Predefined Enumeration types

- ***CHRACTER***
  128 ASCII characters
- ***BIT***
  ('0' , '1')
- ***BOOLEAN***
  (FALSE, TRUE)
- ***SEVERITY_LEVEL***
  (NOTE, WARNING, ERROR, FAILURE)

--Examples of Enumerations types are:

```
type Transition is (H,L,U,Z) ;
type STATE is (S1,S2,S3) ;
type logic_strength is (W,S,D,) ;
-- Enumeration literals must be characters and in ascending range
```

Integer Types

-- The only pre-defined integer is **INTEGER**.
-- Example

```
type  WEIGHT     is INTEGER range 300 downto 30 ;
subtype HEAVY  is WEIGHT   range 100 to 250 ;
subtype LIGHT   is  WEIGHT range 60 downto 40 ;
```

-- Pre-defined

-- Integer Range: $\pm\ 2^{31}$

-- Natural Range: 0 to 2^{31}

-- Positive Range: 1 to 2^{31}

Floating Point Types

-- ~ (Approx.) Real Numbers
-- Contains a range constraint

-- Example
Type WEIGHT ***is range*** 0.5 ***to*** 300.0 ;
Subtype VERY-LIGHT ***is*** WEIGHT ***range*** 1.0 ***to*** 10.0 ;

-- Pre-defined
-- Real numbers are in Package STANDARD
-- Guaranteed to include the range
-- Natural Range: **- 1E ^{31} to + 1E ^{31}**
-- Real data types supports: **=, /=, <=, >, >=, +, -, abs, *, /**

Physical Types

-It is predefined in Package STANDARD

- **Includes the range - 2 ^{31} + 1 to + 2 ^{31} - 1**
- **All delays declared in VHDL must be of the type TIME**

```
type TIME is range - 1E 18  to + 1E 18;
units fs;
      ps = 1000 fs;
      ns = 1000 ps;
      us = 1000 ns;
      ms =1000 u;
      sec = 1000 ms;
      min = 60 sec;
      hr = 3600 sec;
end units;
```

-- fs femto second (10^{-15} seconds) is the base unit

-- Values can be real or integer

User-defined Physical Types

```
-- Example: User Defined Capacitance
type Capacitance is range 0 to 2** 31 –1 ;
Units aF;
      fF = 1000 aF;
      pF = 1000 fF;
      nF = 1000 pF;
      uF = 1000 nF;
      mF = 1000 uF;
end units;
```

```
-- Example: User Defined Voltage
type Voltage is range 0 to 2** 31 –1 ;
Units pV;
      nV = 1000 pV;
      uV = 1000 nV;
      mV = 1000 uV;
      V =1000 mV;
end units;
```

Array Types

- **Array** is collection of objects/values of similar type.

<u>Example</u>

<u>type</u> array_1 ***<u>is array</u>*** (15 ***<u>downto</u>*** 0) ***<u>of</u>*** std_ulogic;

This array declaration defines array_1 that contains 16 elements. The indexes are decremented from15 to 0.

An Array can be put under a decrement or increment mode with

(y ***<u>downto</u>*** x) or (x ***<u>to</u>*** y) .

- **Multi-dimensional Array**

Arrays can have multiple indices:

<u>type</u> Dimension_2 ***<u>is array</u>*** (0 to 7, 10 ***<u>downto</u>*** 0) ***<u>of</u>*** byte;

Array type....

Array Type can be declared as **Constrained** or **Unconstrained**

Constrained Array:

array (discrete range) ***of*** element subtype indication

Unconstrained Array:

array (subtype name range) ***of*** element subtype indication

```
-- Example: Array Types

type word is array (7 downto 0) of BIT ;
type memory is array (natural range <> ) of word ;
variable ROM: memory (0 to 2 ** 10) ;
variable Address: BIT_VECTOR (0 to N) ;
signal mem1: word;
```

Constrained and unconstrained Array type…

```
entity example_array
--------
--------
end example_array;
architecture Behave of example_array is
type word is  array (0 to 15) of BIT  ;
type byte is array (NATURAL range 7 downto 0) of BIT  ;
type unconstrained is array (NATURAL range < >) of BIT  ;
begin
--------
```

-- < > is for the range that is not defined at the time.

-- Range can be declared later when using the array, for example:

```
        subtype  array_a  is  unconstrained (4 downto 0) ;
```

Bit Vector

- Is a single dimensional vector, each element is of type BIT.
- Bit Vector Array is pre-defined in the STANDARD Package.
- Bit Vector is really a template for the elements of an array. Example

COEN: *out* BIT_VECTOR (7 *downto* 0) ;

Subtype Indication

Indices (Length) of the array

COEN <= B"10101010" ;

COEN →

1	0	1	0	1	0	1	0

signal destiny : BIT_VECTOR (*7 **downto*** 0)
destiny <= B"1110-0011" ;

destiny →

1	1	1	0	0	0	1	1

Pre-defined Array Type

```
-- Example

subtype NATURAL is INTEGER range 0 to 2** 31 ;

type BIT_VECTOR array (NATURAL range < >) of BIT;

type POSITIVE is INTEGER range 1 to 2** 31 ;
```

BEHAVIORAL_Concurrent verses Algorithm

Concurrent entity normally infer logic design

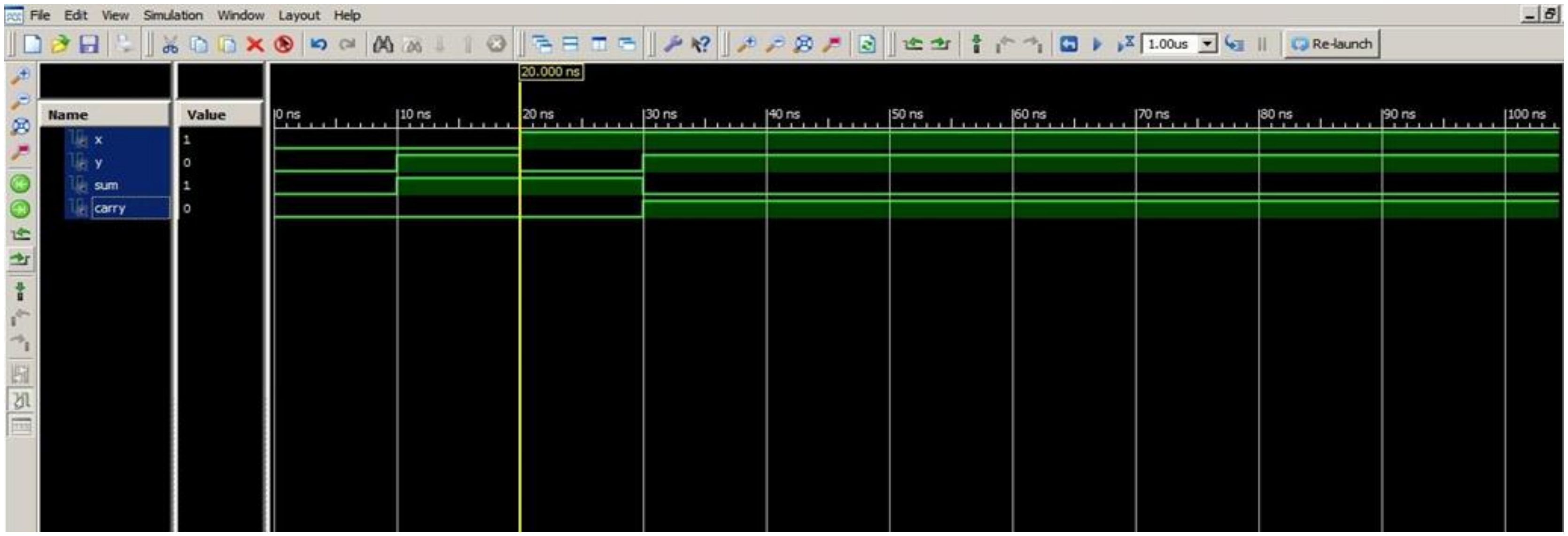
```
architecture CONCURRENT of Half_Adder is
begin
SUM<= X XOR Y;
CARRY<= X AND Y;
End CONCURRENT;
```

```
architecture  BEHAVIORAL of Half_Adder is
  -- signal X,Y : integer; --X and Yare read as integers
  --signal SUM,CARRY: BIT; in the interface
begin
  process (X, Y)
    variable Z : integer;
  begin
   SUM<='0'; CARRY<='0';
    Z := X+ Y;
    if Z =1 then SUM <= '1';
    elsif Z=2 then CARRY<='1';
end if;
  end process;
end BEHAVIORAL;
```

```
architecture ALGORITHMIC of
Half_Adder is
begin
 process (X,Y)
  begin
SUM<= X XOR Y;
CARRY<= X AND Y;
end process;
End ALGORITHMIC;
```

```
================ Test Bench  =========================
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Half_Adder_Con_Tb IS
END Half_Adder_Con_Tb;

ARCHITECTURE CONCURRENT OF Half_Adder_Con_Tb IS

    COMPONENT Half_Adder_Con
    PORT
        (X,Y : IN  std_logic;
         SUM,CARRY : OUT  std_logic);
    END COMPONENT;

   signal X : std_logic := '0';
   signal Y : std_logic := '0';

   signal SUM : std_logic;
   signal CARRY : std_logic;

BEGIN
 -- Instantiate the Unit Under Test (UUT)
   uut: Half_Adder_Con PORT MAP (
          X => X,
          Y => Y,
          SUM => SUM,
                    CARRY=>CARRY);

                -- Stimulus process

   stim_proc: process
   begin
                    X<='0';
                    Y<='0';
                    wait for 10 ns;

                    X<='0';
                    Y<='1';
                    wait for 10 ns;

                    X<='1';
                    Y<='0';
                    wait for 10 ns;

                    X<='1';
                    Y<='1';
```

Simulation of Half Adder



Name	Value
x	1
y	0
sum	1
carry	0

A good site for VHDL Syntax + Examples

http://www.ics.uci.edu/~jmoorkan/vhdlref/