

B.E.
Sixth Semester Examination, 2010
Digital System Design (EE-310-E)

Note : Attempt any five questions.

Q. 1. (a) Discuss all types of data used in VHDL.

Ans. Data Types used in VHDL : Every data object in VHDL can hold a value that belongs to a set of values. This set of values is specified by using a type declaration, a type is a name that has associated with it a set of values and a set of operations. Certain types and operations that can be performed on objects of these types.

The possible types that can exist in the language can be categorised into the following four major categories :

(i) **Scalar Types :** Values belonging to these types appear in a sequential order.

(ii) **Composite Types :** These are composed of elements of a single types (an array type) or elements of different types (record) type.

(iii) **Access Types :** These provides access to objects of a given type (via pointer).

(iv) **File Types :** These provides access to objects that contains a sequence of values of given type.

It is possible to derive subtypes from predefined or user defined types.

Q. 1. (b) Define and discuss with suitable examples all types of overloading.

Ans. Mainly there are two types of overloading occurs in VHDL :

(i) Subprogram overloading

(ii) Operator overloading

(i) **Subprogram Overloading :** Sometimes it is convenient to have two or more subprogram with the same name. In such a case, the subprogram name is said to be overloaded; also the subprogram are said to be overloaded.

For example :

Function COUNT (ORANGES : INTEGER) return INTEGER;

Function COUNT (APPLES : BIT) return INTEGER

Both functions are overloaded since they have the same name, COUNT.

(ii) **Operator Overloading :** Operator overloading is one of the most useful features in the language. When a standard operator symbol is made to behave differently based on the type of its operands. The operator is said to be overloaded.

For example :

The "and" operation is defined for the arguments of type BIT and BOOLEAN and for one-dimensional array of BIT and BOOLEAN only.

What if the arguments were of type 'MVL', it is possible argument the 'and' operations on a function that operates on argument of MVL.

Example : type MVL is ('U', 'O', 'T', 'Z');
 function "and" (L, R : MVL) return MVL;
 function "or" (L, R : MVL) return MVL;
 function "not" (L, R : MVL) return MVL;

Q. 2. Discuss the following with suitable examples :

- | | |
|---------------------------|---------------------|
| (i) Arrays and Loops | (ii) Case Statement |
| (iii) Assertion Statement | (iv) Functions |

Ans. (i) Arrays and Loops : An array type represents a collection of values all belonging to a single type.

Examples for array type declaration :

 type ADDRESS_WORD is array (0 to 63) of BIT;
 type DATA_WORD is array (7 down to 0) of STD_ULOGIC;

Elements of an array can be specifying the index values into array.

A loop statement is used to iterate through a set of sequential statements.

The syntax for loop statement is :

 [loop_label] iteration-scheme loop
 sequential statements
 end loop [loop_label];

Example : FACTORIAL = 1

 For NUMBER in 2 to N loop
 FACTORIAL = FACTORIAL*NUMBER;
 end loop.

(ii) Case Statement . The format of case statement is case expression is

 when choices \Rightarrow Sequential-statements.....branch #1
 when choices \Rightarrow Sequential-statements.....branch #2
 can have any number of branches
 [when others \Rightarrow sequential statement].....last branch
 end case

The case statement selects one of the branches for execution based on the values of the expression.

Example : Case DAY is

 when TUE \Rightarrow POCKET_MONEY: = 6;
 when MON/WED \Rightarrow POCKET_MONEY: = 2;
 when FRI to SUN \Rightarrow POCKET_MONEY: = 7;
 when others \Rightarrow POCKET_MONEY = 0;
 end case;

(iii) **Assertion Statements** : Assertion statements are useful in modeling constraints of an entity. For example, you may want to check if a signal value lies in within a specified range, or check the setup holds times for signal arrival at the input of an entity.

The syntax of assertion statement is :

```
assert boolean_expression
    [report string_expression]
    [severity expression];
```

If the value of the Boolean expression is false, the report message is printed along with severity level.

Example : assert NOW = 0 ns or
(now_last event on C_K) >= HOLD_TIME
report "Hold time is too short"
severity FAILURE;

(iv) **Functions** : Functions are used to define frequently used sequential algorithms, that returns single value.

This value is returned to the calling program using a return statement. Some of their common uses are as resolution function or type conversion function.

Example : function LARGEST (TOTAL_NO : INTEGER; SET : PATTERN)
return REAL is
variable RETURN_VALUE : REAL := 0.0;
begin
for K in SET RANGE loop
if SET(K) > RETURN_VALUE then
RETURN_VALUE := SET(K)
end if;
end loop;
return VALUE
end LARGEST;

Q. 3. (a) Compare the following :

(i) Sequential assignment statement and concurrent assignment statement.

(ii) Behavioural Modelling and structural modelling.

Ans. (i) Sequential Assignment Statement and Concurrent Assignment Statement : We know that signal assignment statements can also appear in the body of process statement : such statements are called sequential statement. While signal assignment statements that appear outside of a process are called concurrent assignment statement.

Concurrent signal assignment statement are event-triggered that is, they are executed. Whenever there is an event on a signal appears in its expression. While, **sequential assignment statements are not event triggered** and are executed in sequence in relation to others.

Consider following statements for their difference

architecture SEQ_SIG_ASG of FRAGMENT 1 is

.....A, B and Z are signals

begin

process(B)

{ begin followings are sequential assignment statements

A <= B

Z <= A

end process;

end

architecture CON_SIG_ASG of FRAGMENT2 is

begin

A <= B

Z <= A

end;

Aside from previous differences, the concurrent signal assignment statement is identical to the sequential signal assignment statement in terms of behaviour.

For every concurrent ASG statement, there is an equivalent process statement with the same semantic meaning.

(ii) Behavioural Modelling Vs. Structural Modelling :

In Behavioural Modelling :

(i) Entity declaration describes the external interface of entity. The syntax is :

entity entity_name is

{generic ;}

{port ;}

{entity_item declaration}

{begin entity statement}

end[entity]

(ii) Architecture Body Syntax :

architecture.....name of entity name is

begin

{..... statements }

end[architecture]

(iii) Process Statement :

[process label] process.....[is]

begin


```
statements  
end process
```

In Structural Modelling :

- (i) Component Declaration : (in this first declare components)
 component name [is]
 [port .]
 end component [none];
- (ii) Component Installation : Second is component installation
 component_label : component_name [port map];
 end structure;

Q. 3. (b) Write short notes on Packages and Libraries.

Ans. Packages : A package provides a convenient mechanism to store and share declarations that are common across many design units. A package is represented by :

- (i) A package declaration and, optionally,
- (ii) A package body.

A package declaration contains a set of declarations that may possibly be shared by many design units, it defines the interface to the package, that is, it defines items that can be made visible to others.

Syntax : Package name is

```
.....  
.....  
end[package]
```

A package body primarily contains the behaviour of the subprograms and the value of the deferred constants declared impulse.

Libraries : A compiled design unit is stored in a design library, a design library is an area of storage in the file system of the host environment.

The format of this storage is not defined by the language. An arbitrary number of design libraries may be specified. Each design library has a logical name which is referenced inside a VHDL description.

The libraries is classified as :

(i) Primary Unit :

- (a) Entity declaration
- (b) Package declaration
- (c) Configuration declaration

(ii) Secondary Units :

- (a) Architecture bodies
- (b) Package bodies

Q. 4. (a) Write VHDL code for design of 16 : 1 MUX using 4 : 1 MUX.

Ans. LIBRARY IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;

```
USE IEEE.STD_LOGIC_ARITH.ALL;
ENTITY MUX1615 is
PORT (A : IN STD_LOGIC_VECTOR(15 DOWN TO 0);
      S : IN STD_LOGIC_VECTOR (3 DOWN TO 0);
      Z : OUT STD_LOGIC);
END MUX 1615;
ARCHITECTURE struc OF MUX 1615 is
SIGNAL Z1, Z2, Z3, Z4 STD_LOGIC;
COMPONENT MUX 415b is
PORT (A, B, C, D, S0, S1 : IN STD_LOGIC);
      Q : OUT STD_LOGIC);
END COMPONENT;
BEGIN
M1 : MUX41b PORT MAP (A(0), A(1), A(2), A(3), S(0), S(1), Z1);
M2 : MUX41b PORT MAP (A(4), A(5), A(6), A(7), S(0), S(1), Z2);
M3 : MUX41b PORT MAP (A(8), A(9), A(10), A(11), S(0), S(1), Z3);
M4 : MUX41b PORT MAP (A(12), A(13), A(14), A(15), S(0), S(1), Z4);
M5 : MUX41b PORT MAP (Z1, Z2, Z3, Z4, S(2), S(3), Z);
END STRUC;
```

Q. 4. (b) What are generics? Explain the role of generics in VHDL with suitable example.

Ans. Generics : It is often useful to pass certain types of information into a design description from its environment. Examples of such information are rise and fall delays and the size of interface ports, this is accomplished by using generics. Generics of an entity are declared along with its ports in the entity declaration and example of a generic N-input and gate is shown.

entity AND_GATE is

```
generic (N : NATURAL);
Port (A : in BIT_VECTOR (1 to N); Z : out BIT);
end AND_GATE;
architecture GENERIC_EX of AND_GATE is
begin
process(A)
variable AND_OUT : BIT;
Z <= AND_OUT;
end process;
end GENERIC_EX;
```

A generic declares a constant object of mode in and can be used in the entity declaration and its corresponding architecture bodies.

The value of this constant can be specified as a globally static expression in on of the followings :

- (i) Entity declaration
- (ii) Component declaration
- (iii) Component instantiation
- (iv) Configuration specification
- (v) Configuration declaration

The value of generic must be determinable at elaboration time.

Q. 5. Write VHDL code for following :

(i) 3 Bit Binary to Gray Code Converter

(ii) Boolean Function $F = AB + CD$.

Ans. (i) 3 Bit Binary to Gray Code Converter :

```
library ieee;
use ieee_std_logic-1164 all;
entity neon_b2g is
port (b : in std_logic_vector (2down to 0);
      g : out std_logic_vector (2 down to 0);
end neon_b2g;
architecture a of neon_b2g is
begin
    g(2) <= b(2)
    'g(1) <= b(2) X or (b(1)
    g(0) <= (b(1) X or b(0);
end a;
```

(ii) Boolean Function $F = AB + CD$:

VHDL code for $F = AB + CD$

architecture A_STRUCTURE of FUNCTION_F is component OR2

port (X, Y : in BIT; Z : out BIT);

end component;

component AND2

port (A, B : in BIT; X : out BIT);

end component;

component AND2

port (C, D : in BIT; Y : out BIT);

begin

X : AND2 port map (A, B, X);

Y : AND2 port map (C, D, Y);

Z : OR2 port map (X, Y, Z);

end function_F structure;

Q. 6. Write VHDL code for following :

(i) 4 bit UP/DOWN Counter

(ii) 4 bit serial in and parallel out shift register.

Ans. (i) 4 bit UP/DOWN Counter :

```
Library ieee;
use ieee-std_logic_1164.all;
entity bit 4ud counter is
port (S, clk in std_logic; g : inout std_logic_vector (3 down to 0))
end bit 4 ud counter;
architecture struc of bit 4 ud counter is
component MUX21
port (a, b, s : in std_logic; Y : out std_logic);
end component;
component fft
port (t, clk, reset : in std_logic; q, q_inv : inout std_logic);
end component;
signal m: std_logic_vector (2 down to 0);
signal q-inv.std_logic_vector (3 down to 0);
begin
t0 : fft port map ('1', clk, '0', q(0), q-inv(0));
M0 : MUX21 portmap (q(0), q-inv(0), s, m(0));
t1 : fft port map ('1', m(0), '0', q(1), q-inv(1));
M1 : MUX 21 portmap (q(1), q-inv(1), s, m(1));
t2 : fft port map ('1', m(1), '0', q(2), q-inv(2));
M2 : MUX21 portmap (q(2), q-inv(2), s, (m(2));
t3 : fft portmap ('1', m(2), '0', q(3), q-inv (3));
end structure;
```

(ii) 4 Bit Serial In and Parallel Out Shift Register :

```
library ieee;
use ieee.std_logic_1164.all;
entity shift is
port(C, ST, in std_logic;
Po : out std_logic_vector (3 down to 0));
end shift;
architecture archi shift is
signal temp : std_logic_vector (3 down to 0));
```



```
begin
process(c)
begin
if ('c' event and c = '1') then
temp <= temp (2 down to 0) & SI;
end if;
end process;
po <= temp;
end arch;
```

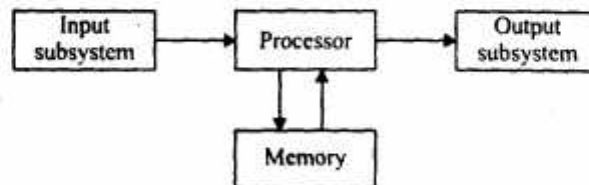
Q. 7. Explain how a simple microcomputer system works. Explain its implementation using VHDL.

Ans. Specification & Implementation of a Microcomputer :

- (i) Basic components of a computer system.
- (ii) Informal and μ vhdl-based description
- (iii) Architecture
- (iv) Implementation
- (v) Operation of simple microcomputer system

XMC : eXample Micro Computer :

- (i) Its cycle time
- (ii) Processor
- (iii) Memory subsystem
- (iv) Input/output (I/O) subsystem



μ vhdl Structural Description :

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
PACKAGE comp_pkg IS
SUBTYPE WordT
IS
SUBTYPE MAddrT IS
SUBTYPE IOAddrT IS
SUBTYPE Byte T
```

```
IS
TYPE Status IS
STD_LOGIC_VECTOR(31 DOWN TO 0);
STD_LOGIC_VECTOR(23 DOWN TO 0);
STD_LOGIC_VECTOR(10 DOWN TO 0);
STD_LOGIC_VECTOR(7 DOWN TO 0);
(undef, p_reset, fetch, execute, memop, ioop);
FUNCTION get_carry (RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
RETURN STD_LOGIC;
FUNCTION get_ovf (RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
RETURN STD_LOGIC;
FUNCTION get_cc (RA_Data, RB_Data, Opcode: STD_LOGIC_VECTOR)
RETURN STD_LOGIC_VECTOR;
END comp_pkg;
PACKAGE BODY comp_pkg IS
FUNCTION get_carry (RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
RETURN STD_LOGIC
IS VARIABLE cy: STD_LOGIC := '0';
BEGIN
--description of carry generation included here
RETURN(cy);
END get_carry;
FUNCTION get_ovf (RA_Data, RB_Data, Imm, Opcode: STD_LOGIC_VECTOR)
RETURN STD_LOGIC
IS VARIABLE ovf: STD_LOGIC := '0';
BEGIN
--description of overflow generation included here
RETURN(ovf);
END get_ovf;
FUNCTION get_cc (RA_Data, RB_Data, Opcode : STD_LOGIC_VECTOR)
RETURN STD_LOGIC_VECTOR
IS VARIABLE cc : STD_LOGIC_VECTOR(3 DOWN TO 0) := "0000";
BEGIN
--description of cc generation included here
RETURN(cc);
END get_cc;
```

```
END comp_pkg;
BEGIN
U1 : ENTITY Memory
PORT MAP (MemAddr, MemLength, MemRd, MemWr, MemEnable,
MemRdy, MemData);
U2 : ENTITY IO
PORT MAP (IOAddr, IOLength, IORd, IOWr, IOEnable, IORdy, IOData);
U3 : ENTITY Processor
PORT MAP (MemAddr, MemData, MemLength, MemRd, MemWr,
MemEnable, MemRdy, IOAddr, IOData, IOLength, IORd, IOWr,
IOEnable, IORdy,
Status, Reset, Clk);
END structural;
```

Q. 8. Write short notes on :

(a) PEEL

(b) PLAs

(c) Operators

Ans. (a) PEEL : The PEEL is a Programmable Electricity Erasable Logic (PEEL device providing an attractive alternative to ordinary PLDs. The PEEL offers the performance, flexibility, ease of design and production practicality needed by logic designers today. The PEEL is available in 20-pin DIP, PLCC, SOIC and TSSOP packages with speeds ranging from 5ns to 25ns with power consumption as low as 37mA. EE-Reprogrammability provide the convenience of instant reprogramming for development and reusable production inventory minimizing the impact of programming changes or errors. EE-Reprogrammability also improves factory testability, thus, assuring the highest quality possible.

The programmable AND array of the PEEL is formed by input lines intersecting product terms. The input lines and product terms are used as follows :

36 Input Lines :

- (i) 20 input lines carry the true and complement of the signals applied to the 10 input pins.
- (ii) 16 additional lines carry the true and complement values of feedback or input signals from the 8 I/Os.

74 Product Terms :

- (i) 64 product terms (arranged in groups of 8) are used to form sum of product functions.
- (ii) 8 output enable terms (one for each I/O).
- (iii) 1 global synchronous present term.
- (iv) 1 global asynchronous clear term.

(b) PLA's : A programmable logic array (PLA) is a programmable device used to implement combinational logic circuits. The PLA has a set of programmable AND gate planes, which link to a set of

programmable OR gate planes, which can then be conditionally complemented to produce an output. This layout allows for a large number of logic functions to be synthesized in the sum of products (and sometimes product of sums) canonical forms.

One application of a PLA is to implement the control over a datapath. It defines various states in an instruction set, and produces the next state (by conditional branching). [e.g., if the machine is in state 2, and will go to state 4 if the instruction contains an immediate field; then the PLA should define the actions of the control in state 2, will set the next state to be 4 if the instruction contains an immediate field, and will define the actions of the control in state 4]. Programmable logic arrays should correspond to a state diagram for the system.

Other commonly used programmable logic devices are PAL, CPLD and FPGA.

Note that the use of the word "programmable" does not indicate that all PLAs are field-programmable, in fact many are mask-programmed during manufacture in the same manner as a mask ROM. This is particularly true of PLAs that are embedded in more complex and numerous integrated circuits such as microprocessors. PLAs that can be programmed after manufacture are called FPLA (Field Programmable PLA).

(c) Operators : There are seven groups of predefined VHDL operators :

(i) Binary logical operators : and or nand nor xor xnor

(ii) Relational operators : = / = < <= > >=

(iii) Shifts operators : sll srl sla sra rol ror

(iv) Adding operators : + - & (concatenation)

(v) Unary sign operators : + -

(vi) Multiplying operators : * / mod rem

(vii) Miscellaneous operators : not abs**

The above classes are arranged in increasing priority when parentheses are not used.