CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Improvement Test – Nov. 2017

Sub:	Verilog HDL					Sub Code:	15EC53	Branch:	ECE & TCE
Date:	21 / 11 / 2017	Duration:	90 min's	Max Marks:	50	Sem / Sec:	5 / All sec		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	Explain the advantages and shortcomings of using VHDL?	[10]	CO4	L1
2	Explain the process of digital system synthesis using VHDL in detail.	[10]	CO4	L2
3	Describe the different VHDL description styles with examples.	[10]	CO4	L3
4 a)	What is the output of the synthesizer (optimized logic expression) for the following sequence of code:	[4]	CO4	L4
4 b)	Draw the simulation waveform for the following sequence of code:	[6]	CO4	L4
5	Write the entity declaration for a 4-bit magnitude comparator. Name the output port 'altb' for a less than b. Then write architecture bodies' one using if-then-else statement, one using when-else statement and one using component instantiation statements and components similar to xnor2 and and4 components. (Hint: the < symbol is used as relational operator for "less than"). Which architecture body is needed to be modified if it is an 8-magnitude comparator? And what are the modifications.	[10]	CO4	L4
6	Explain the components of VHDL program and discuss the different ports supported by VHDL.	[10]	CO4	L2
7	Explain the data objects, data types of VHDL. What are attributes? Explain with examples.	[10]	CO4	L2

```
architecture behavioral of eqcorop4 is
begin
syntp: process (c, d)
       begin
       x <= '0'; y <= '1';
       if (c = '0' and d = '0') then
             x<='1';
       elsif (c = '0' and d = '1') then
             x<='1';
       elsif (c='1' and d = '1') then
             y<='0';
       end if;
       end process syntp;
end behavioral;
```

```
architecture a_model of two_gates
begin
    x <= a AND b after 5 ns;
    y <= not b;
end a_model;
```

1 Explain the advantages and shortcomings of using VHDL? [10]

Ans: **Advantages of using VHDL:**

i) Power and Flexibility
VHDL not only gives you the power to describe circuits quickly using powerful language constructs but also permits other levels of design description including Boolean equations and structural netlists.

ii) Device-Independent Design
VHDL permits the creation of design without having to first choose a device for implementation. With one design description, one target many device architectures. No need to be familiar with a device's architecture in order to optimize your design for resource utilization or performance.

iii) Portability
VHDL's portability permits design used for synthesis can be used for simulation as well. Since VHDL is a standard language, design description can be taken from one simulator to another, one synthesis tool to another, and one platform to another.

iv) Benchmarking Capabilities
Device independent design and portability gives us the ability to benchmark the design using different architectures and different synthesis tools. Without targeting the device one can complete design and synthesize it, creating logic for an architecture of choice. Then evaluate the results and choose the device that best fits your design requirements.

v) ASIC Migration
The same design description used to synthesize logic for a programmable logic device can be used for an ASIC when production volumes ramp. VHDL permits your product to hit the market quickly in programmable logic by synthesizing your design description for an FPGA. When production volumes reach appropriate levels, the same VHDL code can be used in the development of an ASIC.

vi) Fast Time-to-Market and Low Cost
VHDL and programmable logic pair well together to facilitate a speedy design process. VHDL permits designs to be described quickly. Programmable logic eliminates nonrecurring expenses (NREs) and facilitates quick design iterations.

Shortcomings

7 marks

There are three common concerns expressed by design engineers about VHDL:

(1) You give up control of defining the gate-level implementation of circuits that are described with high-level, abstract constructs:
The intent of using VHDL as a language for synthesis is to free the engineer from having to specify gate-level circuit implementation. But one should understand how compiler synthesizes the logic. There is a need to dictate implementation policy.

(2) The logic implementations created by synthesis tools are inefficient:
VHDL compilers will not always produce optimal implementations. Compilers use algorithms to decide upon logic implementations, following standard design methodologies. An algorithm cannot look at a design problem in a unique way.

(3) The quality of synthesis varies from tool to tool:
Different synthesizers look different designs in their own way. Results in best or worst logic implementations.

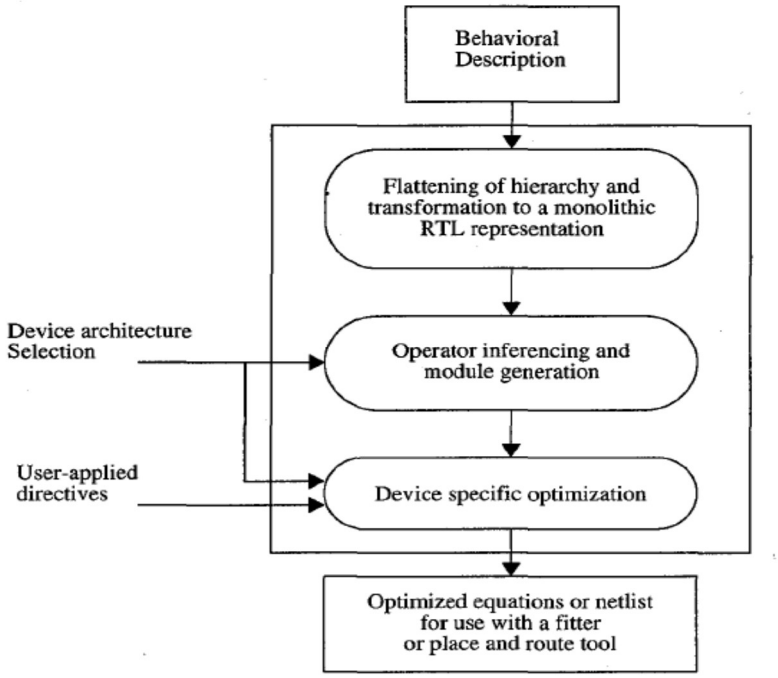
3 marks

2 Explain the process of digital system synthesis using VHDL in detail. [10]

The design process can be broken into the six steps enumerated below:

1. **Define the Design Requirements:**
   Decide upon design objectives and requirements like function of the design, required setup and clock-to-out times, maximum frequency of operation, and critical paths. Having a clear idea of the requirements help to choose a design methodology and helps to choose a device architecture to which design will be synthesized.

2. **Describe the Design in VHDL (Formulate and Code the Design):**
   Formulate the Design: Decided upon a design methodology. Such as top-down, bottom-up, or flat. The first two methods involve creating design hierarchies, the latter involves describing the circuit as one monolithic design. The top-down approach requires that you divide your design into functional blocks, each block having specific inputs and outputs and performing a particular function. A netlist is then created to tie the functional blocks together. The bottom-up approach involves just the opposite: defining and designing the individual blocks of a design, then bringing the different pieces together to form the overall design. A flat design is one in which the details of functional blocks are defined at the same level as the interconnection of those functional blocks.
   Code the Design: The key to writing good VHDL code is to think in terms of hardware and being careful of
   syntax and semantics
3. **Simulate the Source Code:**
   For large designs, simulating the design source code with a VHDL simulator will prove time efficient. The process of concurrent engineering brings circuit simulation to the early stages of design. If the design does not simulate as expected, then you can check your code and make the appropriate corrections before proceeding.
4. **Synthesize, Optimize, and Fit (Place and Route) the Design:**
   Synthesis is the realization of design descriptions into circuits. In other words, synthesis is the process by which logic circuits are created from design descriptions. VHDL synthesis software tools convert VHDL descriptions to technology-specific netlists or sets of equations. Synthesis tools allow designers to design logic circuits by creating design descriptions without having to perform all of the Boolean algebra or create technology-specific, optimized netlists. Synthesis should be technology specific. Synthesis should be technology specific. Figure shown below illustrates the synthesis and optimization processes. The synthesis process then converts the design to internal data structures, allowing the "behavior" of a design to be translated to a register transfer level (RTL) description. RTL descriptions specify registers, signal inputs, signal outputs, and the combinational logic between them. At this point, the combinational logic is still represented by internal data structures. Some synthesis tools will search the data structures for identifiable operators and their operands, replacing these portions of logic with technology-specific, optimized components. Other portions of logic that are not identified are then converted to Boolean expressions that are not yet optimized.



The optimization process depends on three things: the form of the Boolean expressions, the type of resources available, and automatic or user-applied directives (sometimes called constraints). Some forms of expressions may be mapped to logic resources more efficiently than others. Other user or automatic constraints may be applied to optimize expressions for the available resources. These constraints may be to limit the number of appearances of a literal in an expression (to reduce signal loading), limit the number of literals in an expression (to reduce fan-in), or limit the number of terms in an expression (to limit the number of product terms).

"Fitting" is the process of taking the logic produced by the synthesis and optimization processes, and placing it into a logic device, massaging the logic (if necessary) to obtain the best fit. Placing and routing is the process of taking logic produced by synthesis and optimization, packing the logic (massaging it if necessary) into the FPGA logic structures (logic cells), placing the logic cells in optimal locations, and routing signals from logic cell to logic cell or 110. For brevity, we will use the terms "fit" and "place and route" interchangeably, leaving you to discern when we mean to use one or both terms. A "good" placement and route will place critical portionsof a circuit close together to eliminate routing delays.

5. **Simulate the Post-fit (layout) Design Implementation**
   A post-layout simulation enables us to verify not only the functionality of the design but also the timing, such as setup, clock-to-output, and register-to register times. If design is unable to meet your design objectives, then one has to resynthesize and fit design to a new logic device, massage any combination of the synthesis or fitting processes, or choose a different speed grade device.
6. **Program the Device**
   After completing the design description, and synthesizing, optimizing, fitting, and successfully simulating the design, program the device and continue work on the rest of system design. The synthesis, optimization, and fitting software will produce a file for use in programming the device.

10 marks

3 Describe the different VHDL description styles with examples. [10]

The different styles of VHDL descriptions are

I. Behavioral Descriptions: The following is an example of behavioral description:

```
library ieee;
use ieee.std_logic_1164.all;
entity eqcomp4 is port(a,b:in std_logic_vector(3 downto 0);
equals: out std_logic);
end eqcomp4;
architecture behavioral of eqcomp4 is
begin
comp: process (a, b)
          begin
                 if a = b then
           equals<= '1';
        else
               equals<= '0';
        end if;
end process comp;
end behavioral;
```

The architecture of the design is put in an **algorithmic way**. Behavioral descriptions are sometimes referred to as "high-" descriptions because of the resemblance to high-level programming languages. Rather than specifying the structure or netlist of a circuit, signal assignments, or circuit "behavior" is specified. The advantage of high-level descriptions is that no need to focus on the gate-level implementation of a design; instead, one can focus their efforts on describing how the circuit is to "behave."

II. Dataflow Assignment: The following is an example of dataflow description: 3 marks

```
library ieee;
use ieee.std_logic_1164.all;
entity eqcomp4 is port(a,b:in std_logic_vector(3 downto 0);
equals: out std_logic);
end eqcomp4;
architecture dataflow of eqcomp4 is
begin
    equals <= '1' WHEN (a = b) ELSE '0';
end dataflow;
```

It is called dataflow description because it specifies how data will be transferred from signal to signal without the use of sequential statements. Dataflow descriptions are used in those cases where it's more succinct to write simple equations or CASE-WHEN or WITH-SELECT-WHEN statements rather than a complete algorithm.

III. Structural Descriptions: The following is an example of structural description: 3 marks

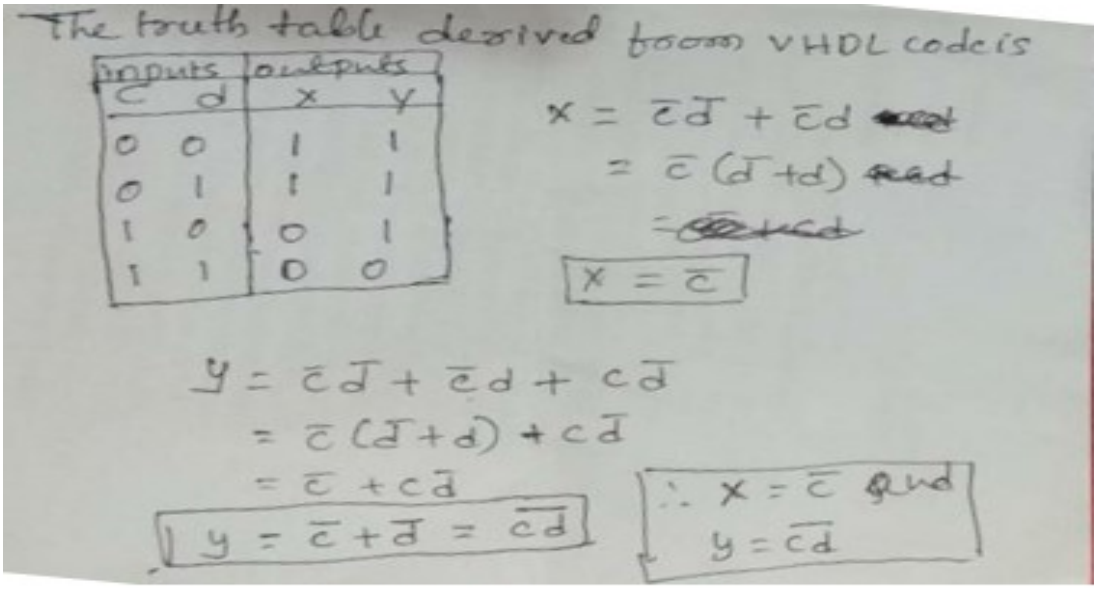
```
library ieee;
use ieee.std_logic_1164.all;
entity eqcomp4 is port(a,b:in std_logic_vector(3 downto 0);
equals: out std_logic);
end eqcomp4;
USE work.gatespkg.all;
architecture struct of eqcomp4 is
SIGNAL x : STD_LOGIC_VECTOR(0 to 3);
begin
u0: xnor2 port map (a(0) ,b(0) ,x(0));
u1: xnor2 port map (a(1) ,b(1) ,x(1));
u2: xnor2 port map (a(2) ,b(2) ,x(2));
u3: xnor2 port map (a(3) ,b(3) ,x(3));
u4: and4 port map (x(0) ,x(1) ,x(2) ,x(3) ,equals);
END struct;
```

Structural descriptions consist of VHDL netlists. These netlists are very much like schematic netlists: Components are instantiated and connected together with signals. Structural designs are hierarchical. In this example, separate entity and architecture pairs are created for the and4, xnor2, and eqcomp4 designs. This design requires that and4 and xnor2 components be defined in a package. One can access these components by including a USE clause, which allows us to instantiate components from the "**gatespkg**" package found in the work library. The eqcomp4 design contains instances of the xnor2 and and4 components. The xnor2 and and4 components must each have associated entity and architecture pairs. The entity and architecture descriptions for the xnor2 and and4 are not contained in the design file for our eqcomp4 component, rather they are accessed by way of the USE clause. 4 marks

4 a) What is the output of the synthesizer (optimized logic expression) for the following sequence of code:

```
architecture behavioral of eqcorop4 is
begin
syntp: process (c, d)
       begin
       x <= '0'; y <= '1';
       if (c = '0' and d = '0') then
             x<='1';
       elsif (c = '0' and d = '1') then
             x<='1';
       elsif (c='1' and d = '1') then
             y<='0';
       end if;
       end process synth;
end behavioral;
```

[4]



The optimized logic expressions obtained after logic synthesis are

$$x = \bar{c},$$
$$y = \overline{c \,.\, d}$$

4 marks

4 b) Draw the simulation waveform for the following sequence of code:

```
architecture a_model of two_gates
begin
    x <= a AND b after 5 ns;
    y <= not b;
end a_model;
```

[6]

6 marks

5 Write the entity declaration for a 4-bit magnitude comparator. Name the output port 'altb' for a less than b. Then write architecture bodies' one using if-then-else statement, one using when-else statement and one using component instantiation statements and components similar to xnor2 and and4 components. (Hint: the < symbol is used as relational operator for "less than"). Which architecture body is needed to be modified if it is an 8-magnitude comparator? And what are the modifications. [10]

The entity declaration with port name 'altb' for a less than b is

```
entity ltcomp is
    Port ( a, b : in  STD_LOGIC_VECTOR (3 downto 0);
           altb : out  STD_LOGIC);
end ltcomp;
```

The architecture `behavioral` defined using `if-then-else` statement is as shown below:

```
architecture behavioral of ltcomp is
begin
p1: process(a,b)
     if(a < b) then
          altb <= '1';
     else
          altb <= '0';
     end if;
end process p1;
end behavioral;
```

2 marks

The architecture `dataflow` defined using `when-else` statement is as shown below:

```
architecture dataflow of ltcomp is
begin
     altb <= '1' when (a < b) else '0';
end dataflow;
```

2 marks

The architecture `structural` defined using `component` instantiation similar to `xnor2` and `and4` components is as shown below:

```
architecture structural of ltcomp is
SIGNAL x : STD_LOGIC_VECTOR(0 to 13);
begin
```

```
u00: inv port map (a(0), x(0));
u01: inv port map (a(1), x(1));
u02: inv port map (a(2), x(2));
u03: inv port map (a(3), x(3));

u04: and2 port map (x(0), b(0), x(4));
u05: and2 port map (x(1), b(1), x(5));
u06: and2 port map (x(2), b(2), x(6));
u07: and2 port map (x(3), b(3), x(7));

u08: xnor2 port map (a(1), b(1), x(8));
u09: xnor2 port map (a(2), b(2), x(9));
u10: xnor2 port map (a(3), b(3), x(10));

u11: and2 port map (x(4), x(8), x(11));
u12: and2 port map (x(5), x(9), x(12));
u13: and2 port map (x(6), x(10), x(13));

u14: or4 port map (x(7), x(11), x(12), x(13), altb);
end structural;
```

2 marks

The architecture with name `structural` is needed to be modified in order to design implement 8-bit magnitude comparator.
The modifications are

I. Signal declaration should be changed to
   `SIGNAL x : STD_LOGIC_VECTOR(0 to 29);`
II. The instantiation of four more `inv` components.
III. The instantiation of eight more `and2` components.
IV. The instantiation of four more `xnor2` components.
V. 8-input `OR(or8)` gate is needed instead of 4-input `OR(or4)` gate.

```
architecture structural of ltcomp is
SIGNAL x : STD_LOGIC_VECTOR(0 to 29);
begin
u00: inv port map (a(0), x(0));
u01: inv port map (a(1), x(1));
u02: inv port map (a(2), x(2));
u03: inv port map (a(3), x(3));
u04: inv port map (a(4), x(4));
u05: inv port map (a(5), x(5));
u06: inv port map (a(6), x(6));
u07: inv port map (a(7), x(7));

u08: and2 port map (x(0), b(0), x(8));
u09: and2 port map (x(1), b(1), x(9));
u10: and2 port map (x(2), b(2), x(10));
u11: and2 port map (x(3), b(3), x(11));
u12: and2 port map (x(4), b(4), x(12));
u13: and2 port map (x(5), b(5), x(13));
u14: and2 port map (x(6), b(6), x(14));
u15: and2 port map (x(7), b(7), x(15));

u16: xnor2 port map (a(1), b(1), x(16));
u17: xnor2 port map (a(2), b(2), x(17));
u18: xnor2 port map (a(3), b(3), x(18));
```

```
u16: xnor2 port map (a(4), b(4), x(19));
u17: xnor2 port map (a(5), b(5), x(20));
u18: xnor2 port map (a(6), b(6), x(21));
u16: xnor2 port map (a(7), b(7), x(22));

u11: and2 port map (x(16), x(8), x(23));
u12: and2 port map (x(17), x(9), x(24));
u13: and2 port map (x(18), x(10), x(25));
u11: and2 port map (x(19), x(11), x(26));
u12: and2 port map (x(20), x(12), x(27));
u13: and2 port map (x(21), x(13), x(28));
u13: and2 port map (x(22), x(14), x(29));
u14: or8 port map (x(15), x(23), x(24), x(25), x(26), x(27), x(28),
x(29), altb);
end structural;
```

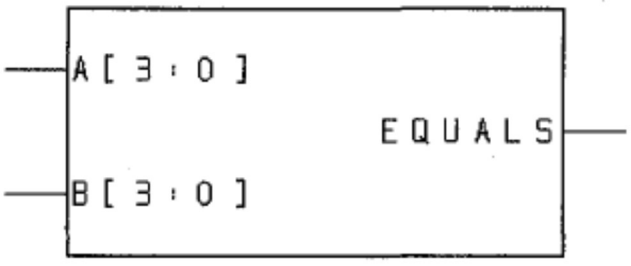
4 marks

6 Explain the components of VHDL program and discuss the different ports supported by VHDL. [10]

An example VHDL program is as given below:

```
-- eqcomp4 is a four bit equality comparator
      ENTITY eqcomp4 IS
      PORT(a,b:in bit_vector(3 DOWNTO 0);equals: OUT BIT);
      end eqcomp4;
      ARCHITECTURE dataflow OF eqcomp4 IS
      BEGIN
        equals <= '1' WHEN (a = b) ELSE '0'; -- equals is active high
      END dataflow;
```

In the above VHDL program line 1 is a comment line. Comments help to document your design; they are for the reader and are ignored by the compiler. Lines 3 describe the I/O of a 4-bit equality comparator called eqcomp4. Lines 2 and 4 begin and end the declaration for the eqcomp4 entity. A VHDL entity describes the inputs and outputs of a design. This could be the I/O of a component in a larger, hierarchical design. Line 2 begins a port, or pin, declaration and the characters ");" at the end of line 2 end the port declaration. Ports are points of communication of the entity with anything outside of the entity. At the beginning of the line 3 of this example, we declare two ports called a and b. These ports are inputs to the design that are 4-bit buses. Each member of the bus, a(0) for instance, is a BIT, which means it may have the value of '0' of '1'. Finally, equals is declared as an output bit at the end of line 3. The entity has a schematic symbol equivalent, as shown in following figure



Lines 5 through 8 describe what our entity, eqcomp4, does. This is called the architecture of the entity; it begins on line 5 with the keyword ARCHITECTURE and ends with "END dataflow" at line 8. In line 5, we give the architecture a name, dataflow, and identify the entity that it describes: "OF eqcomp4." Line 6, obviously enough, begins the architecture description with the keyword BEGIN, and line 7 is where the digital logic is described. This simple architecture includes one equality comparator. Line 7 states that when the value of bus 'a' is equal to the value of bus 'b', then equals gets '1', otherwise equals gets '0'. The '<=' symbol is an operator that can be read "gets" or "is assigned to. The most significant bits (MSB) for a

and b are the leftmost bits a(3) and b(3). The typical order of the bits are from "x downto 0" in order that the most significant bit is the one with the highest index 6 marks

The different types of ports supported by VHDL are IN, OUT, INOUT, or BUFFER.
IN: A port that is declared as mode IN describes a port in which data flows only into the entity. The driver for a port of mode IN is external to the entity.
OUT: A port that is declared as mode OUT describes a port in which data flows only from its source to the output port of the entity. The driver for a port of mode OUT is from within the entity. Mode OUT does not allow for feedback within the associated architecture.
BUFFER: For internal feedback i.e., to use this port as a driver within the architecture, port will be declared as mode BUFFER or mode INOUT. A port that is declared as mode BUFFER is similar to a port that is declared as mode OUT, except that it does allow for internal feedback. Mode BUFFER does not allow for bidirectional ports, however, because it does not permit the signal to be driven from outside of the entity.
INOUT: Mode INOUT can be used anywhere that mode BUFFER is used; that is, everywhere that mode BUFFER is used in a design could be replaced with mode INOUT. 4 marks

7 Explain the data objects, data types of VHDL. What are attributes? Explain with examples. [10]

Data objects are assigned types and hold values of the specified types. Data objects belong to one of three classes: constants, signals, or variables.
Constants: A constant holds a specific value of a type that cannot be changed within the design description, and therefore is usually assigned upon declaration.
Ex: constant width: integer := 8;
Signals: Signals can represent wires, and they can therefore interconnect components.
Ex: signal count: bit_vector(3 downto 0);
Variables: Variables are used only in processes and subprograms (functions and procedures) and must therefore be declared in the declarative region of a process or subprogram.
Ex: variable result: integer := 0;
Aliases: An alias is an alternate identifier for an existing object; it is not a new object.
Ex: signal stored_ad: std_logic_vector(31 downto 0);
alias top_ad: std_logic_vector(3 downto 0) is stored_ad(31 downto 28);
alias bank: std_logic_vector(3 downto 0) is stored_ad(27 downto 24);
alias row_ad: std_logic_vector(ll downto 0) is stored_ad(23 downto 12); 3 marks

Data Types:
A type has a set of values and a set of operations.
Scalar types:
Scalar types have an order that allows relational operators to be used with them. Scalar types comprise four classes: enumeration, integer, floating, and physical types.
Enumeration Types:
An enumeration type is a list of values that an object of that type may hold.
Ex: type states is (idle, preamble, data, jam, nosfd, error);
signal current_state: states;

Integer Types:
An integer type can be defined, as well as a data object declared, with or without specifying a range.
Ex: variable a: integer range 0 to 255;

Floating Types:

The only predefined floating type is **REAL**, which includes the range -1.0E38 to + 1.0E38, inclusive, at a minimum.

Physical Types
Physical type values are used as measurement units. The only predefined physical type is TIME.
EX: TYPE time IS range -2147483647 to 2147483647
units
fs;
ps =1000 fs;
ns =1000 ps;
us =1000 ns;
ms =1000 us;
sec = 1000 ms;
min = 60 sec;
hr = 60 min;
end units;
Physical types do not carry meaning for synthesis

Composite Types:
Composite types define collections of values for which data objects of these types can hold multiple values at a time.
Array Types: An object of an array type is an object consisting of multiple elements of the same type.
Ex1: type word is array(15 downto 0) of bit;
signal b: word;
Ex2: type table8x4 is array(O to 7, 0 to 3) of bit;
constant exclusive_or: table8x4 := ("000_0","001_1","010_1","011_0","100_1"
,"101_0","110_0","111_1") ;

Record Types:
An object of a record type is an object that can consist of multiple elements of different types.
Ex:
type iocell is record
buffer_inp: bit_vector(7 downto 0);
enable: bit;
buffer_out: bit_vector(7 downto 0);
end record;
signal busa, busb, busc: iocell;
signal vec: bit_vector(7 downto 0);
busa.buffer_inp <= vec;
busb.buffer_inp <= busa.buffer_inp;
busb.enable <= '1';
busc <= busb;

Types and Subtypes:
Declaration of different type data and its subtypes:
Ex1: type byte_size is range 0 to 255;
signal my_int: byte_size;
Ex2: subtype byte is bit_vector(7 downto 0);
signal by tel , byte2: byte;
signal datal, data2: byte;
signal addrl, addr2: byte;

Attributes

An attribute provides information about items such as entities, architectures, types, and signals. 5 marks

There are several predefined value, signal, and range attributes that are useful in synthesis.

Scalar types have value attributes. The value attributes are 'left, 'right, 'high, 'low, and 'length.

Ex:

type count is integer range 0 to 127;
type states is (idle, decision, read, write);
type word is array(15 downto 0) of std_logic;
count'left = 0
states'left = idle
word'left = 15
count'right = 127
states 'right write
word'right 0
count 'high 127
states'high = write
word'high 15
count'low 0
states'low = idle

A useful range attribute is the 'range attribute which yields the range of a constrained object.

For example: word' range = 15 downto 0 2 marks