1. **Draw the design flow of VHDL and explain each block.**

**Design Flow**

There are several steps in an HDL-based design process, often called the design flow. These steps are applicable to any HDL-based design process.

The so-called "front end" begins with figuring out the basic approach and building blocks at the **block-diagram** level. Large logic designs, like software programs, are hierarchical, and VHDL and Verilog give you a good framework for defining modules and their interfaces.

The next step is the actual writing of HDL **code** for modules, their interfaces, and their internal details. Although you can use any text editor for this step, the editor included in the HDL's tool suite can make the job a little easier.

Once you've written some code, you will want to **compile** it, of course. The HDL compiler analyzes your code for syntax errors and also checks it for compatibility with other modules on which it relies. It also creates the internal information that is needed for the simulator to process your design later.

The HDL **simulator** allows you to define and apply inputs to your design, and to observe its outputs, without ever having to build the physical circuit. In small projects, you would probably generate inputs and observe outputs manually. But for larger projects, HDL tool suites gives you the ability to create "**test benches**" that automatically apply inputs and compare them with expected outputs.

Actually, simulation is just one piece of a larger step called **verification**. Sure, it is satisfying to watch your simulated circuit produce simulated outputs.

Finding design bugs at this stage has a high value; if bugs are not found until later, all of the so-called "**back-end**" steps must typically be repeated.
In functional verification, we study the circuit's logical operation independent of timing considerations; gate delays and other timing parameters are considered to be zero. In timing verification, we study the circuit's operation including estimated delays, and we verify that the setup, hold, and other timing requirements for sequential devices like flip-flops are met.

After verification, we are ready to move into the back-end stage. The first step is synthesis, converting the HDL description into a set of primitives or components that can be assembled in the target technology. It may generate a list of gates and a netlist that specifies how they should be interconnected.

The designer may "help" the synthesis tool by specifying certain technology-specific constraints, such as the maximum number of logic levels or the strength of logic buffers to use.

In the fitting step, a fitter maps the synthesized primitives or components onto available device resources. For a PLD or CPLD, this may mean assigning equations to available AND-OR elements. It may mean selecting macrocells or laying down individual gates in a pattern and finding ways to connect them within the physical constraints; this is called the place-and-route process.

The "final" step is post-fitting timing verification of the fitted circuit. It is only at this stage that the actual circuit delays due to wire lengths, electrical loading, and other factors can be calculated with reasonable precision.

The most painful problems are the ones that you encounter in the back end of the design flow. For example, if the synthesized design doesn't fit into an available FPGA or doesn't meet timing requirements, you may have to go back as far as rethinking your whole design approach.

2. What are the various objects in VHDL and explain.

A data object holds a value or a range of values of a specified type. In VHDL there are four types of objects. They are

- Signals
- Variables
- Constants
- Files

**signal declaration**

A signal is a data object which holds a list of values. It gives the same information about a signal as in a port declaration, except that no mode is specified:

```
Signal signal-names : signal-type;
```
Zero or more signals can be defined within an architecture, and they roughly correspond to named \textit{wires} in a logic diagram.

\textbf{Variable declaration}

Variables are used in VHDL functions, procedures, and processes, each of which we'll discuss later. Within these program elements, the syntax of a variable declaration is just like that of a signal declaration, except that the variable keyword is used:

\begin{verbatim}
variable variable-names : variable-type;
\end{verbatim}

\textbf{Constants} contribute to the readability, maintainability, and portability of programs in any language. The syntax of a constant declaration in VHDL is

\begin{verbatim}
constant BUS_SIZE: integer := 32;  -- width of component constant MSB: integer := BUS_SIZE-1;  -- bit number of MSB
\end{verbatim}

\textbf{3.7.2.4 File}

This is a special class of data objects. It serves as the interface between VHDL programs and the host environment. There are some special operations that can be performed only on files. These are reading and writing files. The basic operations that we need for reading and writing files are as given below.

- Declaration of a file type
- Declaration of a file
- Opening and closing a file of a specified type
- Reading from a file and writing to a file

Now we will discuss each operation with an example.

3. Explain the use of packages. Give the syntax and structure of a package in VHDL.

\textbf{PACKAGE:}

Groups of procedures and functions that are related can be aggregated into a module that is called package. A package can be shared across many vhdl models. A VHDL package is a file containing definitions of objects that can be
used package in other programs. The kind of objects that can be put into a package include signal, type, constant, function, procedure, and component declarations.

**PACKAGE DECLARATION:**

It defines the interface to the package, the item declared can be accessed by other design unit by using the library.

**PACKAGE BODY**

It contains details of a package, that is the behavior of the sub program and the value of the constant declare in the package.

**SYNTAX**

```vhdl
package package-name is
    type declarations
    signal declarations
    constant declarations
    component declarations
    function declarations
    procedure declarations
end package-name;

package body package-name is
    type declarations
    constant declarations
    function definitions
    procedure definitions
end package-name;
```

Example:

```vhdl
PACKAGE MUX 4 to l_package IS
    COMPONENT MUX 4 to 1
        PORT (M0, M1, M2, M3 : IN STD_LOGIC;
               s : IN STD_LOGIC_VECTOR (1 DOWN TO 0)
               f : OUT STD_LOGIC ;
    END COMPONENT;
END MUX 4 to l_package;
```
4. Write syntax for function. Give one example and explain.

A VHDL function accepts a number of arguments and returns a result similar to a function in a high-level language. The type of arguments and result are to be specified in a VHDL function definition as shown below.

```vhdl
FUNCTION function_name
(
    signal_names1 : signal_type;
    signal_names2 : signal_type;
    ...
    signal_namesn : signal_type;
) RETURN return_type IS
    type declarations
    constant declarations
    variable declarations
    function definitions
    procedure definitions
BEGIN
    sequential statement
    ...
    sequential statement
END function_name;
signal_names1 : It is a list of one more user-selected identifiers to define external-interface signal.
signal_type : It is a built-in or user-defined signal type
```

As shown above, after giving the function-name, zero or more formal parameters are used within the function-body. When the function is called, the actual parameters in the function call are substituted for the formal parameters.

As shown above, a function may define its own local types, constants, variables, nested functions and procedures. A series of sequential statement can be enclosed within the keywords begin and end. These statements are executed when the function is called.

A VHDL procedure is similar to a function, except it does not return a result.

Example:

```vhdl
architecture Inhibit_archf of Inhibit is
function ButNot (A, B: bit) return bit is
    begin
        if B = '0' then return A;
        else return '0';
        end if;
    end ButNot;
    begin
        Z <= ButNot(X,Y);
    end Inhibit_archf;
```

5. Write syntax for procedure. Give one example and explain.
A VHDL procedure is similar to a function, except it does not return a result. While a function call can be used in the place of an expression, a procedure call can be used in the place of a statement. VHDL procedures allow their arguments to be specified with type out or inout, so it is actually possible for a procedure to "return" a result.

In addition with this write 4th question answer.

(6&7) Write a VHDL entity and architecture for 3-bit ripple counter using flip-flops.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Counter_3bit is
  Port ( CLK : in STD_LOGIC;
         Count : out STD_LOGIC_VECTOR (2 downto 0));
end Counter_3bit;

architecture Behavioral of Counter_3bit is
  signal cin : std_logic_vector(2 downto 0) :="000";
begin
  process(CLK)
  begin
    if(clk'event and clk='0') then
      cin <= "000";
    else
      cin <= cin + 1;
    end if;
    Count <= cin;
  end process;
end Behavioral;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity function is
  Port (a,b,c,d : in STD_LOGIC;
        F(X) : out STD_LOGIC);
end function;

architecture dataflow of function is
begin
  F(X)<= (a OR b) AND (c XOR d);
end dataflow;

9. Explain the syntax for structural design elements with example.
A VHDL architecture that uses components is often called a structural description or structural design.

The important features of structural type of architecture body are,

- Design hierarchy
- Components are used
- Each component is simulated separately

In the structural modeling, an entity is described as a set of components connected by signals, that is, as a netlist. The components used in an architecture may be from a library or may be ones that were previously defined as part of a design.

For example:

```vhdl
ARCHITECTURE gate OF or_gate IS

COMPONENT or_gate

PORT (X, Y : IN STD_LOGIC;
     Z : OUT STD_LOGIC);

END COMPONENT;

BEGIN

U1 : or_gate PORT MAP (a, b, c);

END gate;
```

The name of the architecture body is gate. The entity declaration for or_gate specifies the interface ports for this architecture body. The architecture body is composed of two parts: the declarative part (before the keyword BEGIN) and the statement parts (after the keyword BEGIN). The components may be either be predefined components in a library or they may later be bound to other components in a library. The declared components are instantiated in the statement part of the architecture body using component instantiation statement. U1 is a component label for this component instantiation. x is connected to signal a and y is connected to signal b, z is connected to c in or_gate (entity) portmap. Note that in this case, the signals in the port map of a component instantiation and the port signals in the component declaration are associated by position. A component instantiation statement is a concurrent statement. The details about the component

10. Explain with an example the syntax and the function of the following VHDL statements.
   (a) Process statement
   (b) If, else and else-if statements
   (c) Case statement
   (d) Loop statement
(a).

Process is main concurrent statement in VHDL code which describe the sequential behavior of design. All statements within process execute sequentially in zero time. Only one driver is placed on a signal. The signal is updated with the last value assigned to it within the process.

**Syntax:**

```vhdl
PROCESS (sensitivity list)
BEGIN
    Sequential statements
END PROCESS;
```

**Example:**

```vhdl
PROCESS (en, a, b, c, d)
BEGIN
    if en = '1' then
        z <= a AND b;
        z <= c AND d;
    END IF;
END PROCESS;
```

A process statement is concurrent statement that can appear within architecture body. A process statement also has a declarative part (before keyword BEGIN) and a statement part (between the keywords BEGIN and END PROCESS). The statements appearing within the statement part are sequential statements and are executed sequentially.

**Sensitivity List**

List of signals on which process should execute after changing its state. Every process must have either sensitivity list or wait statement. Only static signal names are allowed in sensitivity list. Wait statement is not synthesizable. In the above example en, a, b, c, d are in the sensitivity list.

**Process Types**

**Combinational Process**

- Generates a combinational logic.
- All input must be present in sensitivity list.

**Clocked Process**

- Generate a register logic.
- Input under clock transition may not be included in sensitivity list.
(b) An IF statement selects a sequence of statements for execution based on the value of a condition. The condition can be any expression that evaluates to a Boolean value.

Syntax:

```plaintext
IF condition 1 THEN
  sequential statement
ELSIF condition 2 THEN
  sequential statement
ELSE
  sequential statement
END IF;
```

Example:

```plaintext
PROCESS (a, b, en)
BEGIN
  if en = '00' then
    c <= a;
  elsif en = '01' then
    c <= b
  else
    c <= '0'
  END if;
END PROCESS;
```

(c).

The case statement selects one of the branches for execution based on the value of the expression. The expression value must be of a discrete type of a one dimensional array type. Choice may be expressed as single values, as a range of values, by using a | vertical bar : represents an "or" choice, or by using the others clause (when others). You can not overlap the range.

Syntax:

```plaintext
CASE expression IS
  WHEN choice1 =>
    (statements)
  WHEN choice 2 =>
    (statements)
  WHEN OTHERS
    (statements)
END CASE;
```
(d). loop statements.

Another important class of sequential statements are the loop statements. The simplest of these has the syntax shown in table 5-48 and creates an infinite loop. A more familiar loop, one that we've seen before, is the for loop, with the syntax shown in Table 5-49. Note that the loop variable, identifier, is declared implicitly by its appearance in the for loop and has the same type as range. This variable may be used within the loop's sequential statements, and it steps through all of the values in range, from left to right, one per iteration.

Table 5-48
Syntax of a basic VHDL loop statement.

<table>
<thead>
<tr>
<th>loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential-statement</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>sequential-statement</td>
</tr>
<tr>
<td>end loop;</td>
</tr>
</tbody>
</table>

Table 5-49
Syntax of a VHDL for loop.

<table>
<thead>
<tr>
<th>for identifier in range loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential-statement</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>sequential-statement</td>
</tr>
<tr>
<td>end loop;</td>
</tr>
</tbody>
</table>

Two more useful sequential statements that can be executed within a loop are "exit" and "next". When executed, exit transfers control to the statement immediately following the loop end. On the other hand, next causes any remaining statements in the loop to be bypassed and begins the next iteration of the loop.

The last kind of loop statement is the while loop, with the syntax shown in table 5-51. In this form, boolean-expression is tested before each iteration of the loop.

Table 5-51
Syntax of a VHDL while loop.

<table>
<thead>
<tr>
<th>while boolean-expression loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential-statement</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>sequential-statement</td>
</tr>
<tr>
<td>end loop;</td>
</tr>
</tbody>
</table>
and the loop is executed only if the value of the expression is true.

11. **Write a VHDL program in behavioural style to generate a clock with OFF time and ON time equal to 5 ns.**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity program is
  Port ( CLK : inout STD_LOGIC);
end program;

architecture Behavioral of program is
begin
  process(CLK)
  begin
    if(clk'event and clk='0') then
      clk<= "0";  // clock is OFF
      wait for 5 ns;
      clk<='1';  //clock is ON
    end if;
  end process;
end Behavioral;
```

12. **What is the importance in time dimension in VHDL and explain its function.**
3.10 The Time Dimension and Simulation

3.10.1 The Time Dimension

The VHDL provides one predefined physical type: time. The definition of the type 'time' as given in the package 'STANDARD' is as below.

```
TYPE TIME IS RANGE < implementation dependent >

UNITS

fs;                        -- femtoseconds
ps = 1000 fs;              -- picoseconds
ns = 1000 ps;              -- nanoseconds
us = 1000 ns;              -- microseconds
ms = 1000 us;              -- milliseconds
s = 1000 ms;               -- seconds
min = 60 s;                -- minutes
hr = 60 min;               -- hours

END UNITS;
```

The first unit, femtoseconds (fs) is the smallest unit of time. It is referred to as the 'base unit'. All of the other units can be defined in terms of any of the units defined earlier.

There are two ways to specify a time delay in a VHDL-code,

1. Using the keyword after,
2. Using wait statement.

**Using the keyword after**

The keyword 'after' can be used to specify a time delay in any signal-assignment statement such as selected assignment, sequential, concurrent and conditional. For example, consider the VHDL-code for a positive edge triggered D-flip-flop as given below.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY flip_flop IS
  PORT ( D, Clock : IN STD_LOGIC;
          Q : OUT STD_LOGIC);
END flip_flop;
ARCHITECTURE Behavior OF flip_flop IS
```
BEGIN

PROCESS (Clock)
BEGIN

IF Clock'EVENT AND Clock = '1' THEN
  Q <= D AFTER 3 ns;
END IF;

END PROCESS;

END Behavior;

After applying a positive edge trigger at the clock input, the output of the flip-flop Q takes the value of input D after 3 ns. Thus the flip-flop has a delay of 3 ns. It is customary to include such delay parameters while writing VHDL codes for any components. The VHDL simulator uses these delays and predicts approximate timing behavior of a larger circuit that uses these components.

Using wait statement

In VHDL, a more general way of specifying when a process is executed or suspended pending the occurrence of an event or events, is by the wait statement. The different forms of the wait statement are given below.

1) WAIT FOR time expression,

This wait statement causes suspension of the process for the time period obtained by evaluating the time expression. This expression should evaluate a value that is of type time.

For example,

WAIT FOR 10 ns;

2) WAIT ON signal;

This form of the wait statement causes a process to suspend execution until an event occurs on one or more signals.

For example,

WAIT ON clk, reset;

Here, an event on any of the signals causes the process to resume execution with the first statement following the wait statement.

3) WAIT UNTIL <condition>;

In this case, the condition evaluates a Boolean value, True or False.

For example,

WAIT UNTIL (Clk'EVENT AND Clk = '1')

-- for true rising edge detection
4) WAIT

This statement suspends the process indefinitely.

By using wait statements within processes, it is possible to model components that are not necessarily data driven but are driven only by certain types of events such as rising edge of clock signal. Most of the times, we need to construct models in which process is to be suspended at multiple points within a process. It becomes possible with the use of wait statement. A process can have a sensitivity list or wait statement but not both.

The following example explains this concept. In the previous part of this section, we have seen the VHDL-code for a positive edge triggered D-flip-flop. The same flip-flop can be described using WAIT UNTIL statement as given below. Here, the sensitivity list is omitted. The WAIT UNTIL construct implies that the sensitivity list includes only the clock signal.

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
ENTITY flip_flop IS
  PORT ( D, Clock : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
END flip_flop;
ARCHITECTURE Behavior OF flip_flop IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL Clock'EVENT AND Clock = '1';
    Q <= D;
  END PROCESS;
END Behavior;
```

13. Explain briefly about the following statements:
(a) Signal assignment statement  (b) Conditional signal assignment statement
(c) Select signal assignment statement  (d) Variable assignment statement.

What is the value for the following function?
Variable signal assignment statement

Variables are used in VHDL functions, procedures, and processes, each of which we'll discuss later. Within these program elements, the syntax of a variable declaration is just like that of a signal declaration, except that the variable keyword is used:

```
variable variable-names : variable-type;
```

14 to 20-refer my class note.

Example problem:
Example 4.69: Design the logic circuit and write a data-flow style VHDL program for the following function. \( f(p) = \sum_{ABCD(1, 5, 6, 7, 9, 13) + d(4, 15)} \).

**Solution: K-Map Simplification**

\[
\begin{array}{c|ccc}
CD & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 1 & 1 & X \\
01 & X & 1 & 1 & 1 \\
11 & 1 & X & X & X \\
10 & 1 & 1 & X & X \\
\end{array}
\]

Fig. 4.158

\[ f(p) = \overline{CD} + \overline{AB} \]

**Logic Circuit**

```
Library IEEE;
Use IEEE std_logic_1164.all;
Entity LC is
  Port (A, B, C, D : in std_logic;
       P : out std_logic);
end LC;

Architecture behaviour of LC is
Begin
  P <= (Not C and D) or (Not A and B);
end behaviour;
```

21. Design a logic circuit to detect prime number detector of a 4-bit input.
Solution: In 4-bit (0 - 15) possible prime numbers are 1, 2, 3, 5, 7, 11 and 13.

\[ P = \overline{A}D + \overline{A} \overline{B}C + B \overline{C}D + B \overline{C}D \]

Logic Circuit

VHDL Code

Entity prime is
Port (A : in STD_LOGIC vector (3 downto 0);
    P : out STD_LOGIC);
end prime;

Architecture primeP_arch of prime is
Signal P_0, P_1, P_2, P_3, P

Component AND2 port (I_0, I_1 : in STD_LOGIC;
                     O : out STD_LOGIC);
end component;

Component AND3 port (J_0, J_1, J_2 : in STD_LOGIC;
                     B : out STD_LOGIC);
end component;
Component OR4 Port (R₀, R₁, R₂, R₃ : in STD_LOGIC;
            R : out STD_LOGIC);
end component;

Begin
  A₁ : AND2 port map (Not A, D, R₀);
  A₂ : AND3 port map (Not A, Not B, C, P₁);
  A₃ : AND3 port map (Not B, C, D, P₂);
  A₄ : AND3 port map (B, Not C, D, P₃);
  OR₁ : OR4 port map (P₂, P₃, P₁, P₀);
end primeP_arch;

Example 4.71: Design a logic circuit to detect prime number of a 5-bit input. Write the structural VHDL program for the above design.

Solution: In 5-bit (0 - 31) possible prime numbers are: 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31.

![Fig. 4.160](image)

:. \( Z = \overline{A} \overline{B} \overline{E} + \overline{B} \overline{C} \overline{E} + \overline{A} \overline{C} \overline{D} \overline{E} + \overline{A} \overline{C} \overline{D} \overline{E} + \overline{A} \overline{B} \overline{C} \overline{E} + \overline{A} \overline{B} \overline{C} \overline{D} \overline{E} + \overline{A} \overline{B} \overline{C} \overline{D} \)
Entity prime is
port (A : in STD_LOGIC vector (4 downto 0);
P : out STD_LOGIC);
end prime;

Architecture primeP_arch of prime is
signal P0, P1, P2, P3, P4, P5, P
Component    AND3 port (I0, I1, I2 : in STD_LOGIC;
                O : out STD_LOGIC);
                end component;
Component    AND4 port (A0, A1, A2, A3 : in STD_LOGIC;
                B : out STD_LOGIC);
                end component;
Component    OR7 port (R0, R1, R2, R3, R4, R5, R6 : in STD_LOGIC;
                R : out STD_LOGIC);
                end component;
begin
    A1 :  AND3 port map (Not A, Not B, E, P0);
    A2 :  AND3 port map (Not B, Not C, E, P1);
    A3 :  AND4 port map (Not A, Not C, D, E, P2);
    A4 :  AND4 port map (Not A, C, Not D, E, P3);
    A5 :  AND4 port map (A, B, C, E, P4);
    A6 :  AND4 port map (A, C, D, E, P5);
    A7 :  AND4 port map (Not A, Not B, Not C, D, P6);
    OR1 :  OR7 port map (P0, P1, P2, P3, P4, P5, P6, P);
end primeP_arch;