**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **03-June-2020** | **Name:** | **Raziya Banu** |
| **Course:** | **HDL** | **USN:** | **4AL16EC058** |
| **Topic:** | **EDA Playground** | **Semester & Section:** | **8th sem & ‘B’ section** |
| **Github Repository:** |  |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report –**  In my first session today I have studied about the EDA Playground Tutorial, How to Download And Install Xilinx Vivado Design Suite.  In [computer engineering](https://en.wikipedia.org/wiki/Computer_engineering), a hardware description language (HDL) is a specialized [computer language](https://en.wikipedia.org/wiki/Computer_language) used to describe the structure and behavior of [electronic circuits](https://en.wikipedia.org/wiki/Electronic_circuit), and most commonly, [digital logic](https://en.wikipedia.org/wiki/Digital_logic) circuits.  A hardware description language enables a precise, [formal](https://en.wikipedia.org/wiki/Formal_language) description of an electronic circuit that allows for the automated analysis and [simulation](https://en.wikipedia.org/wiki/Electronic_circuit_simulation) of an electronic circuit. It also allows for the [synthesis](https://en.wikipedia.org/wiki/Logic_synthesis) of an HDL description into a [netlist](https://en.wikipedia.org/wiki/Netlist" \o "Netlist) (a specification of physical electronic components and how they are connected together), which can then be [placed and routed](https://en.wikipedia.org/wiki/Place_and_route) to produce the [set of masks](https://en.wikipedia.org/wiki/Mask_set) used to create an [integrated circuit](https://en.wikipedia.org/wiki/Integrated_circuit).  A hardware description language looks much like a [programming language](https://en.wikipedia.org/wiki/Programming_language) such as [C](https://en.wikipedia.org/wiki/C_(programming_language)) or [ALGOL](https://en.wikipedia.org/wiki/ALGOL); it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs explicitly include the notion of time.  HDLs form an integral part of [electronic design automation](https://en.wikipedia.org/wiki/Electronic_design_automation) (EDA) systems, especially for complex circuits, such as [application-specific integrated circuits](https://en.wikipedia.org/wiki/Application-specific_integrated_circuit), [microprocessors](https://en.wikipedia.org/wiki/Microprocessor), and [programmable logic devices](https://en.wikipedia.org/wiki/Programmable_logic_device).  **Motivation**  Due to the exploding complexity of digital electronic circuits since the 1970s (see [Moore's law](https://en.wikipedia.org/wiki/Moore%27s_law)), circuit designers needed [digital logic](https://en.wikipedia.org/wiki/Digital_logic) descriptions to be performed at a high level without being tied to a specific electronic technology, such as [ECL](https://en.wikipedia.org/wiki/Emitter-coupled_logic), [TTL](https://en.wikipedia.org/wiki/Transistor%E2%80%93transistor_logic) or [CMOS](https://en.wikipedia.org/wiki/CMOS). HDLs were created to implement [register-transfer level](https://en.wikipedia.org/wiki/Register-transfer_level) abstraction, a model of the data flow and timing of a circuit.[[1]](https://en.wikipedia.org/wiki/Hardware_description_language#cite_note-1)  There are two major hardware description languages: [VHDL](https://en.wikipedia.org/wiki/VHDL) and [Verilog](https://en.wikipedia.org/wiki/Verilog). There are different types of description in them: "dataflow, behavioral and structural". Example of dataflow of VHDL:  LIBRARY IEEE;  USE IEEE.STD\_LOGIC\_1164.ALL;  ENTITY not1 IS  PORT(  a : IN STD\_LOGIC;  b: OUT STD\_LOGIC;  );  END not1;  ARCHITECTURE behavioral OF not1 IS  BEGIN  b <= NOT a;  END behavioral; Structure of HDL HDLs are standard text-based expressions of the structure of electronic systems and their behaviour over time. Like [concurrent programming](https://en.wikipedia.org/wiki/Concurrent_programming) languages, HDL syntax and semantics include explicit notations for expressing [concurrency](https://en.wikipedia.org/wiki/Concurrency_(computer_science)). However, in contrast to most software [programming languages](https://en.wikipedia.org/wiki/Programming_language), HDLs also include an explicit notion of time, which is a primary attribute of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchy of blocks are properly classified as [netlist](https://en.wikipedia.org/wiki/Netlist" \o "Netlist) languages used in electric [computer-aided design](https://en.wikipedia.org/wiki/Computer-aided_design). HDL can be used to express designs in structural, behavioral or register-transfer-level architectures for the same circuit functionality; in the latter two cases the [synthesizer](https://en.wikipedia.org/wiki/Logic_synthesis) decides the architecture and logic gate layout.  HDLs are used to write executable specifications for hardware. A program designed to implement the underlying semantics of the language statements and simulate the progress of time provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this executability that gives HDLs the illusion of being [programming languages](https://en.wikipedia.org/wiki/Programming_languages), when they are more precisely classified as [specification languages](https://en.wikipedia.org/wiki/Specification_language) or [modeling languages](https://en.wikipedia.org/wiki/Modeling_language). Simulators capable of supporting discrete-event (digital) and continuous-time (analog) modeling exist, and HDLs targeted for each are available. Comparison with control-flow languages It is certainly possible to represent hardware semantics using traditional programming languages such as [C++](https://en.wikipedia.org/wiki/C%2B%2B), which operate on [control flow](https://en.wikipedia.org/wiki/Control_flow) semantics as opposed to [data flow](https://en.wikipedia.org/wiki/Data_flow), although to function as such, programs must be augmented with extensive and unwieldy [class libraries](https://en.wikipedia.org/wiki/Class_library#Object_and_class_libraries). Generally, however, software programming languages do not include any capability for explicitly expressing time, and thus cannot function as hardware description languages. Before the introduction of [System Verilog](https://en.wikipedia.org/wiki/System_Verilog) in 2002, [C++](https://en.wikipedia.org/wiki/C%2B%2B) integration with a [logic simulator](https://en.wikipedia.org/wiki/Logic_simulation) was one of the few ways to use [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) in hardware verification. System Verilog is the first major HDL to offer object orientation and garbage collection.  Using the proper subset of hardware description language, a program called a synthesizer, or [logic synthesis tool](https://en.wikipedia.org/wiki/Logic_synthesis), can infer hardware logic operations from the language statements and produce an equivalent netlist of generic hardware primitives[[jargon](https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style#Technical_language)] to implement the specified behaviour.[[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] Synthesizers generally ignore the expression of any timing constructs in the text. Digital logic synthesizers, for example, generally use [clock edges](https://en.wikipedia.org/wiki/Clock_signal) as the way to time the circuit, ignoring any timing constructs. The ability to have a synthesizable subset of the language does not itself make a hardware description language. Design verification with HDLs [Functional verification](https://en.wikipedia.org/wiki/Functional_verification)  Design verification was a laborious, repetitive loop of writing and running simulation [test cases](https://en.wikipedia.org/wiki/Test_case) against the design under test. As chip designs have grown larger and more complex, the task of design verification has grown to the point where it now dominates the schedule of a design team. Looking for ways to improve design productivity, the [electronic design automation](https://en.wikipedia.org/wiki/Electronic_design_automation) industry developed the [Property Specification Language](https://en.wikipedia.org/wiki/Property_Specification_Language).  In [formal verification](https://en.wikipedia.org/wiki/Formal_verification) terms, a property is a factual statement about the expected or assumed behavior of another object. Ideally, for a given HDL description, a property or properties can be proven true or false using formal mathematical methods. In practical terms, many properties cannot be proven because they occupy an unbounded [solution space](https://en.wikipedia.org/wiki/Solution_space). However, if provided a set of operating assumptions or constraints, a property checker can prove (or disprove) certain properties by narrowing the solution space.  The assertions do not model circuit activity, but capture and document the designer's intent in the HDL code. In a simulation environment, the simulator evaluates all specified assertions, reporting the location and severity of any violations. In a synthesis environment, the synthesis tool usually operates with the policy of halting synthesis upon any violation. Assertion based verification is still in its infancy, but is expected to become an integral part of the HDL design toolset.  Implement 4 to 1 MUX using two 2 to 1 MUX using structural modelling style and test the module in online/offline compiler.  module m41(out, a, b, c, d, s0, s1);  output out;  input a, b, c, d, s0, s1;  wire sobar, s1bar, T1, T2, T3, T4;  not (s0bar, s0), (s1bar, s1);  and (T1, a, s0bar, s1bar), (T2, b, s0bar, s1),(T3, c, s0, s1bar), (T4, d, s0, s1);  or(out, T1, T2, T3, T4);  endmodule |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **03-June-2020** | **Name:** | **Raziya Banu** | |
| **Course:** | **Udemy** | **USN:** | **4AL16EC058** | |
| **Topic:** | **Return Functions in Python** | **Semester & Section:** | **8th sem & ‘B’ section** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **Image of session** | | | |
| **Python return statement:** A return statement is used to end the execution of the function call and “returns” the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.  **Note:** Return statement cannot be used outside the function.  **Syntax:**  def fun():  statements  .  .  return [expression]  **Example:**  filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | # Python program to  # demonstrate return statement    def add(a, b):      # returning sum of a and b      return a + b  def is\_true(a):      # returning boolean of a      return bool(a)  # calling function  res = add(2, 3)  print("Result of add function is {}".format(res))  res = is\_true(2<5)  print("\nResult of is\_true function is {}".format(res)) |   **Output:**  Result of add function is 5  Result of is\_true function is True  Returning Multiple Values  In Python, we can return multiple values from a function. Following are different ways.   * **Using Object:** This is similar to C/C++ and Java, we can create a class (in C, struct) to hold multiple values and return an object of the class.   filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | # A Python program to return multiple  # values from a method using class  class Test:      def \_\_init\_\_(self):          self.str = "geeksforgeeks"          self.x = 20  # This function returns an object of Test  def fun():      return Test()    # Driver code to test above method  t = fun()  print(t.str)  print(t.x) |   **Output:**  Geeksforgeeks  20   * **Using Tuple:** A Tuple is a comma separated sequence of items. It is created with or without (). Tuples are immutable. See [this](https://www.geeksforgeeks.org/python-tuples/)for details of [tuple](https://www.geeksforgeeks.org/python-tuples/).   filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | # A Python program to return multiple  # values from a method using tuple  # This function returns a tuple  def fun():      str = "geeksforgeeks"      x = 20      return str, x;  # Return tuple, we could also                      # write (str, x)    # Driver code to test above method  str, x = fun() # Assign returned tuple  print(str)  print(x) |   **Output:**  Geeksforgeeks  20   * **Using a list:** A list is like an array of items created using square brackets. They are different from arrays as they can contain items of different types. Lists are different from tuples as they are mutable. See [this](https://www.geeksforgeeks.org/python-list/)for details of [list](https://www.geeksforgeeks.org/python-list/).   filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | # A Python program to return multiple  # values from a method using list  # This function returns a list  def fun():      str = "geeksforgeeks"      x = 20      return [str, x];  # Driver code to test above method  list = fun()  print(list) |   **Output:**  ['geeksforgeeks', 20]   * **Using a Dictionary:** A Dictionary is similar to hash or map in other languages. See [this](https://www.geeksforgeeks.org/python-dictionary/) for details of [dictionary](https://www.geeksforgeeks.org/python-dictionary/).   filter\_none  edit  play\_arrow  brightness\_4   |  | | --- | | # A Python program to return multiple  # values from a method using dictionary  # This function returns a dictionary  def fun():      d = dict();      d['str'] = "GeeksforGeeks"      d['x']   = 20      return d  # Driver code to test above method  d = fun()  print(d) |   **Output:**  {'x': 20, 'str': 'GeeksforGeeks'} | | | |