**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **29/05/2020** | **Name:** | **Navya R** |
| **Course:** | **Logic design** | **USN:** | **4AL16EC041** |
| **Topic:** | **Analysis of clocked sequential circuits**  **Digital clock design** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **Navya-R** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report** ***ANALYSIS OF CLOCKED SEQUENTIAL CIRCUITS***    * Some flip-flops have asynchronous inputs that are used to force the flip-flop to a particular state independently of the clock      * The input that sets the flip-flop to 1 is called preset or direct set. The input that clears the flip-flop to 0 is called clear or direct reset.      * When power is turned on in a digital system, the state of the flip-flops is unknown. The direct inputs are useful for bringing all flip-flops in the system to a known starting state prior to the clocked operation.      * The knowledge of the type of flip-flops and a list of the Boolean expressions of the combinational circuit provide the information needed to draw the logic diagram of the se­quential circuit. * The part of the combinational circuit that gene rates external outputs is de­scribed algebraically by a set of Boolean functions called output equations. * The part of the circuit that generates the inputs to flip-flops is described algebraically by a set of Boolean func­tions called flip-flop input equations (or excitation equations). * The information available in a state table can be represented graphically in the form of a state diagram. In this type of diagram a state is represented by a circle and the (clock-triggered) transitions between states are indicated by directed lines connecting the circles. * The time sequence of inputs, outputs, and flip-flop states can be enumerated in a state table (transition table). The table has four parts present state, next state, inputs and outputs. * In general a sequential circuit with 'm' flip-flops and 'n' inputs needs 2m+n rows in the state table.   ***Positive Edge Triggered D Flip-flop***   * When the reset input is 0 it forces output Q' to Stay at 1 which clears output Q to 0 thus resetting the flip-flop. * Two other connections from the reset input ensure that the S input of the third SR latch stays at logic 1 while the reset input is at 0 regardless of the values of D and Clk. * Function table suggests that: * When R = 0, the output is set to 0 (independent of D and Clk). * The clock at Clk is shown with an upward arrow to indi­cate that the flip-flop triggers on the positive edge of the clock. * The value in D is transferred to Q with every positive-edge clock signal provided that R = 1.   Positive Edge Triggered D Flip-flop ***Analysis with T Flip-Flops***  * The circuit can be specified by the characteristic equations:      * + **Q(t+1) = T ⊕ Q = T'Q + TQ'** * The sequential circuit has two flip-flops A and B, one input x, and one output y and can be described algebraically by two input equations and an output equation:      * + **TA = Bx**      * + **TB = x**      * + **y = AB** * The state table for the circuit is listed below. The values for y are obtained from the out­put equation. The values for the next state can be derived from the state equations by substi­tuting TA and TB in the characteristic equations yielding:      * + **A(t + 1) = (Bx)' A + (Bx)A' = AB' + Ax' + A'Bx**      * + **B(t + 1) = x ⊕ B** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **29/5/2020** | **Name:** | **Navya R** | |
| **Course:** | **Python** | **USN:** | **4al16ec041** | |
| **Topic:** | **Object Oriented Programming** | **Semester & Section:** | **8 A** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **REPORT** *Object-Oriented Programming (OOP)*  * Object-oriented Programming, or OOP for short, is a [programming paradigm](http://en.wikipedia.org/wiki/Programming_paradigm) which provides a means of structuring programs so that properties and behaviors are bundled into individual objects*.* * For instance, an object could represent a person with a name property, age, address, etc., with behaviors like walking, talking, breathing, and running. Or an email with properties like recipient list, subject, body, etc., and behaviors like adding attachments and sending. * Put another way, object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc. OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.  *How To Define a Class in Python* Defining a class is simple in Python:  class Dog:  pass  You start with the class keyword to indicate that you are creating a class, then you add the name of the class ***Class Attributes*** While instance attributes are specific to each object, class attributes are the same for all instances—which in this case is all dogs.  class Dog:  # Class Attribute  species = 'mammal'  # Initializer / Instance Attributes  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age *Instantiating Objects* Instantiating is a fancy term for creating a new, unique instance of a class.  For example:  >>>  >>> class Dog:  ... pass  ...  >>> Dog()  <\_\_main\_\_.Dog object at 0x1004ccc50>  >>> Dog()  <\_\_main\_\_.Dog object at 0x1004ccc90>  >>> a = Dog()  >>> b = Dog()  >>> a == b  False  **Example**  class Dog:  # Class Attribute  species = 'mammal'  # Initializer / Instance Attributes  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  # Instantiate the Dog object  philo = Dog("Philo", 5)  mikey = Dog("Mikey", 6)  # Access the instance attributes  print("{} is {} and {} is {}.".format(  philo.name, philo.age, mikey.name, mikey.age))  # Is Philo a mammal?  if philo.species == "mammal":  print("{0} is a {1}!".format(philo.name, philo.species))  **Output**  Philo is 5 and Mikey is 6.  Philo is a mammal! *Instance Methods* Instance methods are defined inside a class and are used to get the contents of an instance. They can also be used to perform operations with the attributes of our objects. Like the \_\_init\_\_ method, the first argument is always self:  class Dog:  # Class Attribute  species = 'mammal'  # Initializer / Instance Attributes  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  # instance method  def description(self):  return "{} is {} years old".format(self.name, self.age)  # instance method  def speak(self, sound):  return "{} says {}".format(self.name, sound)  # Instantiate the Dog object  mikey = Dog("Mikey", 6)  # call our instance methods  print(mikey.description())  print(mikey.speak("Gruff Gruff")) | | | |