**DAY 10 ASSIGNMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **28-05-2020** | **Name:** | **Ashish Shanbhag** |
| **Course:** | **Logic Design** | **USN:** | **4AL16EC008** |
| **Topic:** | 1. **Boolean equations for digital circuits. Combinational circuits: Conversion of MUX and Decoders to logic gates.** 2. **Design of 7 segment decoder with common anode display** | **Semester & Section:** | **8th A** |
| **Github Repository:** | **Ashish Shanbhag** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS**        **Boolean equations for digital circuits**  Boolean algebra, a logic algebra, allows the rules used in the algebra of numbers to be applied to logic. It formalizes the rules of logic. Boolean algebra is used to simplify Boolean expressions which represent [combinational logic circuits](http://electronics-course.com/combinational-logic). It reduces the original expression to an equivalent expression that has fewer terms which means that less logic gates are needed to implement the combinational logic circuit. Laws of Boolean Algebra Boolean Algebra Laws are used to simplify boolean expressions.  **Basic Boolean Laws**   1. Idempotent Law    * A \* A = A    * A + A = A 2. Associative Law    * (A \* B) \* C = A \* (B \* C)    * (A + B) + C = A + (B + C) 3. Commutative Law    * A \* B = B \* A    * A + B = B + A 4. Distributive Law    * A \* (B + C) = A \* B + A \* C    * A + (B \* C) = (A + B) \* (A + C) 5. Identity Law    * A \* 0 = 0     A \* 1 = A    * A + 1 = 1     A + 0 = A 6. Complement Law    * A \* ~A = 0    * A + ~A = 1 7. Involution Law    * ~(~A) = A 8. DeMorgan's Law    * ~(A \* B) = ~A + ~B    * ~(A + B) = ~A \* ~B   Each law is described by two parts that are duals of each other. The Principle of duality is   * Interchanging the + (OR) and \* (AND) operations of the expression. * Interchanging the 0 and 1 elements of the expression. * Not changing the form of the variables.   **Conversion of MUX and Decoders to logic gates**  Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a Multiplexer.  The multiplexer, shortened to “MUX” or “MPX”, is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called “channels” one at a time to the output.  It is tough to remember all the Mux to Logic Gate conversion implicitly. So its important to know the method to do the conversion. Here are some steps that will help you with the process,  i) Start with the truth table of the logic gate to be converted  ii) Fix one of the input variables as the Select signal (S) and then decide on what the input signals to the Mux should be so that the Mux satisfies all the cases in the truth table  Example: For AND, Output = 0 for B=0, and Output = A for B = 1.  For XNOR, Output = A’ for B=0, and Output = A for B=1. |

**DAY 10 ASSIGNMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **28-05-2020** | **Name:** | **Ashish Shanbhag** |
| **Course:** | **PYTHON** | **USN:** | **4AL16EC008** |
| **Topic:** | **Object Oriented Programming** | **Semester & Section:** | **8th A** |
| **Github Repository:** | **Ashish Shanbhag** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS**      **Object Oriented Programming**  Like other general purpose languages, python is also an object-oriented language since its beginning. Python is an object-oriented programming language. It allows us to develop applications using an Object Oriented approach. In Python, we can easily create and use classes and objects.  Major principles of object-oriented programming system are given below.   * Object * Class * Method * Inheritance * Polymorphism * Data Abstraction * Encapsulation  Object The object is an entity that has state and behaviour. Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute \_\_doc\_\_, which returns the doc string defined in the function source code. Class The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class then it should contain an attribute and method, i.e. an email id, name, age, salary, etc. Syntax  1. **class** ClassName: 2. <statement-1> 3. . 4. . 5. <statement-N>  Method The method is a function that is associated with an object. In Python, a method is not unique to class instances. Any object type can have methods. Inheritance Inheritance is the most important aspect of object-oriented programming which simulates the real world concept of inheritance. It specifies that the child object acquires all the properties and behaviors of the parent object.  By using inheritance, we can create a class which uses all the properties and behavior of another class. The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class.  It provides re-usability of the code. Polymorphism Polymorphism contains two words "poly" and "morphs". Poly means many and Morphs means form, shape. By polymorphism, we understand that one task can be performed in different ways. For example You have a class animal, and all animals speak. But they speak differently. Here, the "speak" behavior is polymorphic in the sense and depends on the animal. So, the abstract "animal" concept does not actually "speak", but specific animals (like dogs and cats) have a concrete implementation of the action "speak". Encapsulation Encapsulation is also an important aspect of object-oriented programming. It is used to restrict access to methods and variables. In encapsulation, code and data are wrapped together within a single unit from being modified by accident. Data Abstraction Data abstraction and encapsulation both are often used as synonyms. Both are nearly synonym because data abstraction is achieved through encapsulation. Abstraction is used to hide internal details and show only functionalities. Abstracting something means to give names to things so that the name captures the core of what a function or a whole program does. |