**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **29-May-2020** | **Name:** | **Russell D’souza** |
| **Course:** | **Logic Design** | **USN:** | **4AL15EC023** |
| **Topic:** | **1. Analysis of clocked sequential circuits**  **2. Digital clock design** | **Semester & Section:** | **8th sem & ‘A’ section** |
| **Github Repository:** | **Russell1005** |  |  |

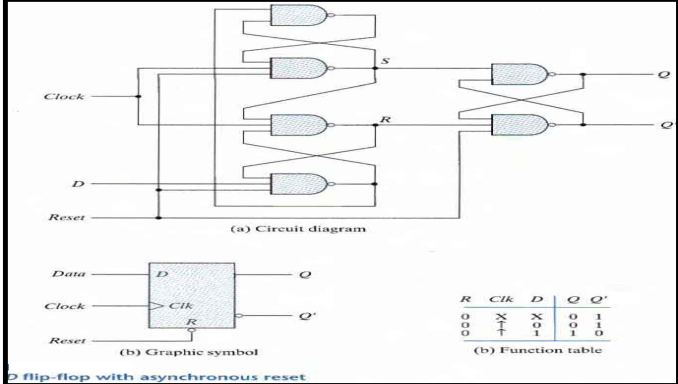
|  |
| --- |
| **MORNING SESSION DETAILS** |
| **Image of session** |

1) 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 statediagram**.** 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**

* A circuit diagram of a Positive edge triggered D Flip-flop is shown as below. It has an additional reset input connected to the three NAND gates.

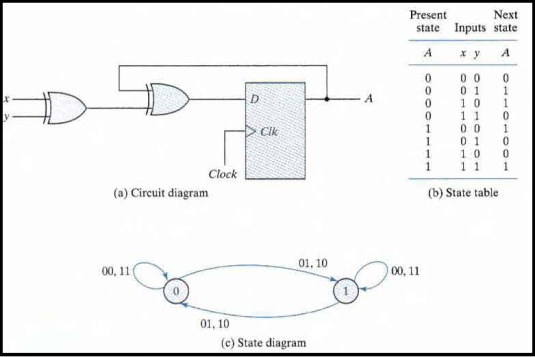


* When the reset input is 0 it forces output Q' to Stay at1 which clears output Q to 0 thus resetting the flip-flop.
* Two other connections from the reset input ensure that the S input of the third SR latch stays at logic 1 while the reset input is at 0 regardless of the values of D and Clk.
* Function 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.

### Analysis with D Flip-Flops

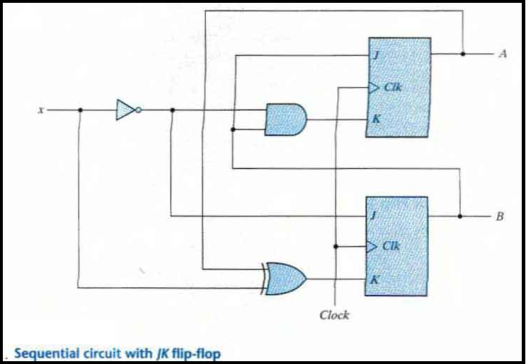
* The input equation of a D Flip-flop is given by **DA = A ⊕ x ⊕ y.** DA means a D Flip-flop with output A.
* The x and y variables are the inputs to the circuit. No output equations are given, which implies that the output comes from the output of the flip-flop.
* The state table has one column for the present state of flip-flop 'A' two columns for the two in­puts, and one column for the next state of A.
* The next-state values are obtained from the state equation **A(t + 1) = A ⊕ x ⊕ y.**
* The expression specifies an odd function and is equal to 1 when only one variable is 1 or when all three variables are 1.



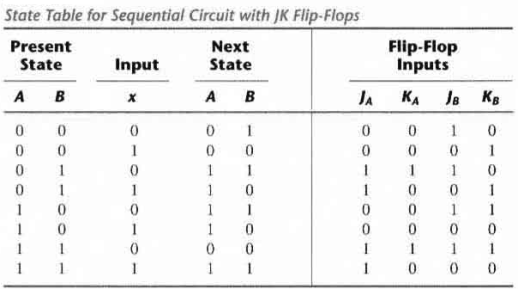
### Analysis with JK Flip-Flops

* The circuit can be specified by the flip-flop input equations:
  + **JA = B; KA = Bx'**

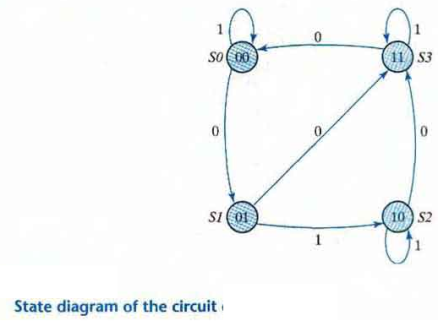
* + **JB = x'; KB = A'x + Ax' = A ⊕ x**
* The next state of each flip-flop is evaluated from the correspon­ding J and K inputs and the characteristic table of the JK flip-flop listed as:
  + **When J = 1 and K = 0 the next state is 1**
  + **When J = 0 and K = 1 the next state is 0**
  + **When J = 0 and K = 0 there is no change of state and the next-state value is the same as that of the present state.**
  + **When J = K = 1, the next-state bit is the com­plement of the present-state bit.**



* The characteristic equations for the flip-flops are
  + **A(t + 1) = JA' + K'A**
  + **B(t + 1) = JB' + K'B**
* This gives us the state equation of A by substituting the values of JA, KA

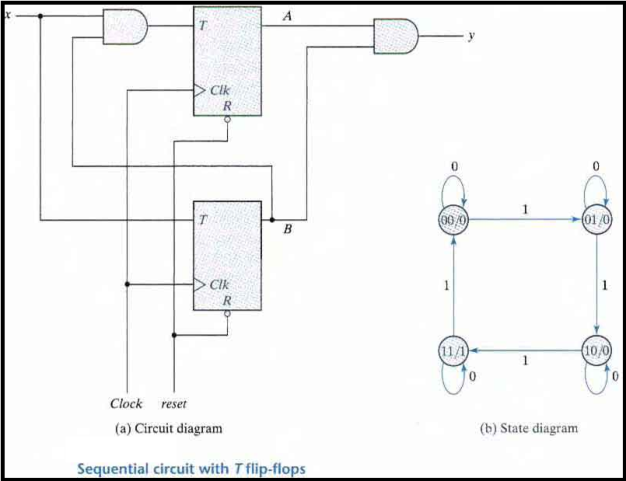
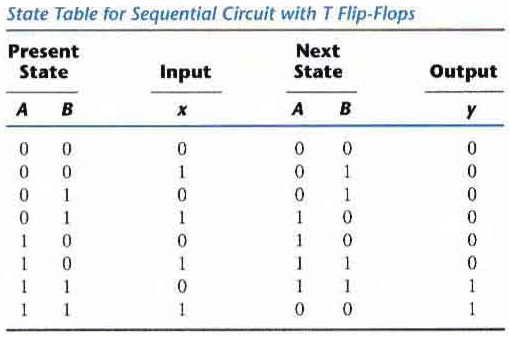


* + **A(t + 1) = BA' + (Bx')'A = A'B + AB' + Ax**
* The state equation provides the bit values for the column headed "Next State" for A in the state table. Similarly, the state equation for flip-flop B can be derived from the characteristic equa­tion by substituting the values of JB and KB.:  
  + **B(t + 1) = x'B' + (A ⊕ x)'B = B'x' + ABx + A'Bx'**



### Analysis with T Flip-Flop

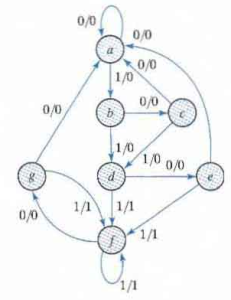
* The circuit can be specified by the characteristic equations
  + **Q(t+1) = T ⊕ Q = T'Q + TQ'**
* The sequential circuit has two flip-flops A and B, one input x, and one output y and can be described algebraically by two input equations and an output equation
  + **TA = Bx**
  + **TB = x**
  + **y = 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'B**
  + **B(t + 1) = x ⊕ B**

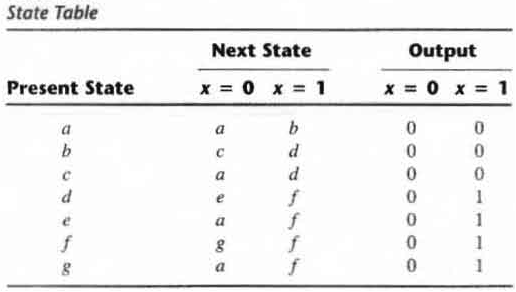
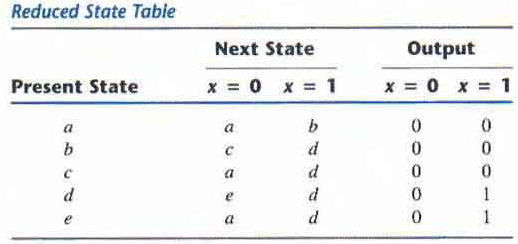
  
  


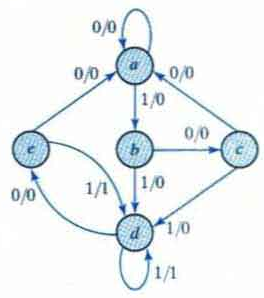
### STATE REDUCTION AND ASSIGNMENT

* Two sequen­tial circuits may exhibit the same input-output behavior but have a different number of inter­nal states in their state diagram.
* Certain properties of sequential circuits may simplify a design by reducing the number of gates and flip-flops it uses. Reducing the number of flip-flops reduces the cost of a circuit.
* The reduction in the number of flip-flops in a sequential circuit is referred to as the state­ reduction problem. State-reduction algorithms are concerned with procedures for reducing the number of states in a state table while keeping the external input-output requirements un­change

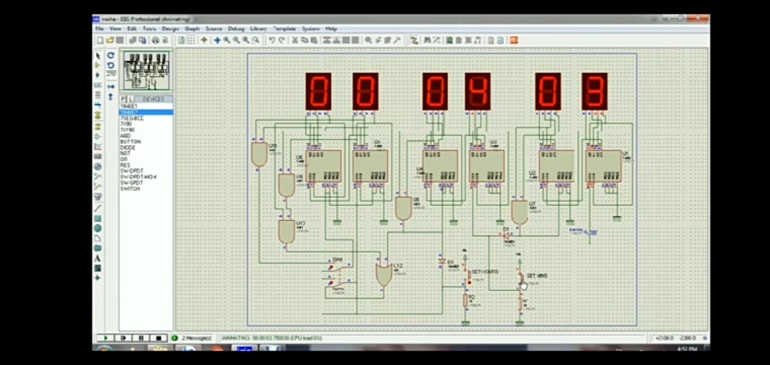
**Example of State Reduction**



* First we need the state table: it is more convenient to apply procedures for state reduction with the use of a table rather than a diagram.  
    
  
* Then we apply the reduction algorithms "Two states are said to be equivalent if for each member of the set of in­puts they give exactly the same output and send the circuit either to the same state or to an equivalent state."
* When two states are equivalent one of them can be removed without alter­ing the input-output relationships.
* Going through the state table, we look for two pres­ent states that go to the same next state and have the same output for both input combinations. States **g and e** are two such states.
* The procedure of removing a state and replacing it by its equivalent is "The row with present state g is removed and state g is replaced by state e each time it occurs in the columns headed "Next State,"
* Similarly, states f and d are equivalent, and state f can be removed and replaced by d.  
    
  
* In general reducing the number of states in a state table may result in a circuit with less equipents. But it does not guarantee a saving in the number of flip-flops or the number of gates.



**2. Digital clock signal**



A digital clock is a one kind of clock used to display the time in the form of digital includes symbols or numerals. These clocks are frequently connected with electronic drives, but the term digital refers only to the LCD display, not to the drive mechanism. The digital clock circuit uses the 50-60hz oscillation of AC power.Most digital alarm clocks display the hour of the day in the form of 12 hours or 24 hours with an indication of AM or PM. Most digital alarm clocks use LCD display, seven segment display or VFD.

Digital clocks run with mains electricity and must be reset the time when the power is off. Most of the clocks don’t have a battery back up, so this will cause to fail to generate an alarm sound at the fixed time. To overcome this problem, many digital alarm clocks are available to operate with a battery during the power outage. Commercial digital clocks are generally more consistent than consumer clocks. Because, these clocks give backup to maintain the time using multi decade battery during power off.

8051 Microcontroller based Digital Alarm Clock with LCD Display

The required components of this 8051 microcontroller based digital clock circuit with LCD display mainly include LCD display, AT89C51 Microcontroller, Preset, piezo buzzer and speaker. The function of each and every component of this project is discussed below.

LCD Display

A 16×2 LCD display is an electronic display and it is used in a wide range of applications.These kind of displays is used in a multi segment LEDs an 7-segment displays. In this LCD display, each character is shown in 5×7 pixel matrix. This LCD display consists of two registers, they are data register and command register. A command register is an order for LCD display to do a task like clearing of its screen, initializing, controlling of display and cursor position setting. The data (ASCII value of the character) register is used to display the stored data on LCD display.

**DAILY ASSESSMENT FORMAT**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Date:** | **28-5-2020** | **Name:** | **Russell D’souza** | | **Course:** | **Python programming** | **USN:** | **4AL15EC023** | | **Topic:** | **1. Objects orientation** | **Semester & Section:** | **8th A** | | **Github Repository:** | **Russell1005** |  |  | |
|  |
|  |
| **AFTERNOON SESSION DETAILS** | |
| **Image of session** | |

Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy. This chapter helps you become an expert in using Python's object-oriented programming support.

If you do not have any previous experience with object-oriented (OO) programming, you may want to consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic concepts.

However, here is small introduction of Object-Oriented Programming (OOP) to bring you at speed −

## Overview of OOP Terminology

* **Class** − A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
* **Class variable** − A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
* **Data member** − A class variable or instance variable that holds data associated with a class and its objects.
* **Function overloading** − The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved.
* **Instance variable** − A variable that is defined inside a method and belongs only to the current instance of a class.
* **Inheritance** − The transfer of the characteristics of a class to other classes that are derived from it.
* **Instance** − An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
* **Instantiation** − The creation of an instance of a class.
* **Method** − A special kind of function that is defined in a class definition.
* **Object** − A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
* **Operator overloading** − The assignment of more than one function to a particular operator.

## Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows −

class ClassName:

'Optional class documentation string'

class\_suite

* The class has a documentation string, which can be accessed via *ClassName.\_\_doc\_\_*.
* The *class\_suite* consists of all the component statements defining class members, data attributes and functions.

### Example

class Employee:

'Common base class for all employees'

empCount = 0

def \_\_init\_\_(self, name, salary):

self.name = name

self.salary = salary

Employee.empCount += 1

def displayCount(self):

print "Total Employee %d" % Employee.empCount

def displayEmployee(self):

print "Name : ", self.name, ", Salary: ", self.salary

* The variable *empCount* is a class variable whose value is shared among all instances of a this class. This can be accessed as *Employee.empCount* from inside the class or outside the class.
* The first method *\_\_init\_\_()* is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.
* You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the *self* argument to the list for you; you do not need to include it when you call the methods.

## Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its *\_\_init\_\_* method accepts.

"This would create first object of Employee class"

emp1 = Employee("Zara", 2000)

"This would create second object of Employee class"

emp2 = Employee("Manni", 5000)

## Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows −

emp1.displayEmployee()

emp2.displayEmployee()

print "Total Employee %d" % Employee.empCount

**Built-In Class Attributes**

Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute −

* **\_\_dict\_\_** − Dictionary containing the class's namespace.
* **\_\_doc\_\_** − Class documentation string or none, if undefined.
* **\_\_name\_\_** − Class name.
* **\_\_module\_\_** − Module name in which the class is defined. This attribute is "\_\_main\_\_" in interactive mode.
* **\_\_bases\_\_** − A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

## Class Inheritance

Instead of starting from scratch, you can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name.

The child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class. A child class can also override data members and methods from the parent.

### Syntax

Derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name −

class SubClassName (ParentClass1[, ParentClass2, ...]):

'Optional class documentation string'

class\_suite

### Example

#!/usr/bin/python

class Parent: # define parent class

parentAttr = 100

def \_\_init\_\_(self):

print "Calling parent constructor"

def parentMethod(self):

print 'Calling parent method'

def setAttr(self, attr):

Parent.parentAttr = attr

def getAttr(self):

print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class

def \_\_init\_\_(self):

print "Calling child constructor"

def childMethod(self):

print 'Calling child method'

c = Child() # instance of child

c.childMethod() # child calls its method

c.parentMethod() # calls parent's method

c.setAttr(200) # again call parent's method

c.getAttr() # again call parent's method

**When the above code is executed, it produces the following result −**

Calling child constructor

Calling child method

Calling parent method

Parent attribute : 200

**Data Hiding**

An object's attributes may or may not be visible outside the class definition. You need to name attributes with a double underscore prefix, and those attributes then are not be directly visible to outsiders.

**Overloading Operators**

Suppose you have created a Vector class to represent two-dimensional vectors, what happens when you use the plus operator to add them? Most likely Python will yell at you.

You could, however, define the *\_\_add\_\_* method in your class to perform vector addition and then the plus operator would behave as per expectation −

**Program:**

from tkinter import \*

from backend import Database

database=Database("books.db")

class Window(object):

def \_\_init\_\_(self,window):

self.window = window

self.window.wm\_title("BookStore")

l1=Label(window,text="Title")

l1.grid(row=0,column=0)

l2=Label(window,text="Author")

l2.grid(row=0,column=2)

l3=Label(window,text="Year")

l3.grid(row=1,column=0)

l4=Label(window,text="ISBN")

l4.grid(row=1,column=2)

self.title\_text=StringVar()

self.e1=Entry(window,textvariable=self.title\_text)

self.e1.grid(row=0,column=1)

self.author\_text=StringVar()

self.e2=Entry(window,textvariable=self.author\_text)

self.e2.grid(row=0,column=3)

self.year\_text=StringVar()

self.e3=Entry(window,textvariable=self.year\_text)

self.e3.grid(row=1,column=1)

self.isbn\_text=StringVar()

self.e4=Entry(window,textvariable=self.isbn\_text)

self.e4.grid(row=1,column=3)

self.list1=Listbox(window, height=6,width=35)

self.list1.grid(row=2,column=0,rowspan=6,columnspan=2)

sb1=Scrollbar(window)

sb1.grid(row=2,column=2,rowspan=6)

self.list1.configure(yscrollcommand=sb1.set)

sb1.configure(command=self.list1.yview)

self.list1.bind('<<ListboxSelect>>',self.get\_selected\_row)

b1=Button(window,text="View all", width=12,command=self.view\_command)

b1.grid(row=2,column=3)

b2=Button(window,text="Search entry", width=12,command=self.search\_command)

b2.grid(row=3,column=3)

b3=Button(window,text="Add entry", width=12,command=self.add\_command)

b3.grid(row=4,column=3)

b4=Button(window,text="Update selected", width=12,command=self.update\_command)

b4.grid(row=5,column=3)

b5=Button(window,text="Delete selected", width=12,command=self.delete\_command)

b5.grid(row=6,column=3)

b6=Button(window,text="Close", width=12,command=window.destroy)

b6.grid(row=7,column=3)

def get\_selected\_row(self,event):

index=self.list1.curselection()[0]

self.selected\_tuple=self.list1.get(index)

self.e1.delete(0,END)

self.e1.insert(END,self.selected\_tuple[1])

self.e2.delete(0,END)

self.e2.insert(END,self.selected\_tuple[2])

self.e3.delete(0,END)

self.e3.insert(END,self.selected\_tuple[3])

self.e4.delete(0,END)

self.e4.insert(END,self.selected\_tuple[4])

def view\_command(self):

self.list1.delete(0,END)

for row in database.view():

self.list1.insert(END,row)

def search\_command(self):

self.list1.delete(0,END)

for row in database.search(self.title\_text.get(),self.author\_text.get(),self.year\_text.get(),self.isbn\_text.get()):

self.list1.insert(END,row)

def add\_command(self):

database.insert(self.title\_text.get(),self.author\_text.get(),self.year\_text.get(),self.isbn\_text.get())

self.list1.delete(0,END)

self.list1.insert(END,(self.title\_text.get(),self.author\_text.get(),self.year\_text.get(),self.isbn\_text.get()))

def delete\_command(self):

database.delete(self.selected\_tuple[0])

def update\_command(self):

database.update(self.selected\_tuple[0],self.title\_text.get(),self.author\_text.get(),self.year\_text.get(),self.isbn\_text.get())

window=Tk()

Window(window)

window.mainloop()