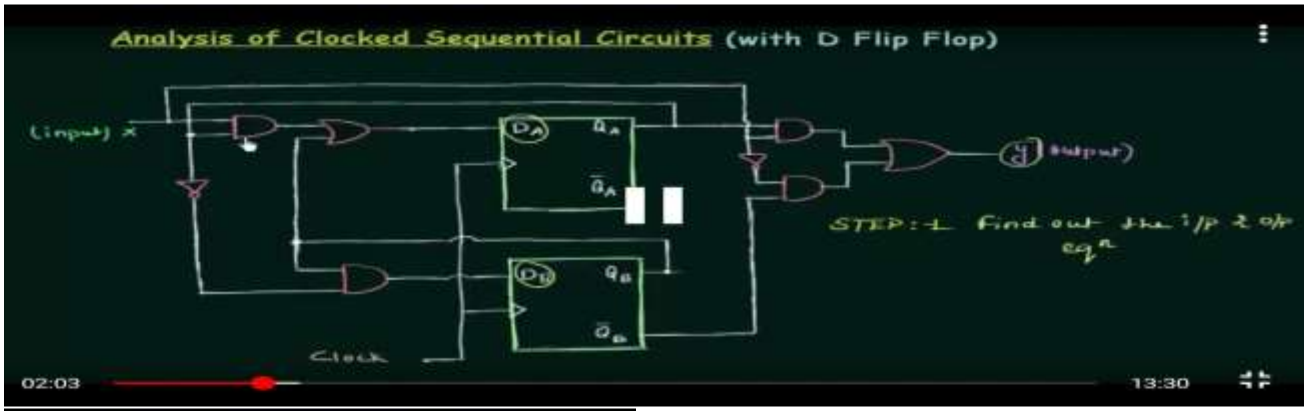
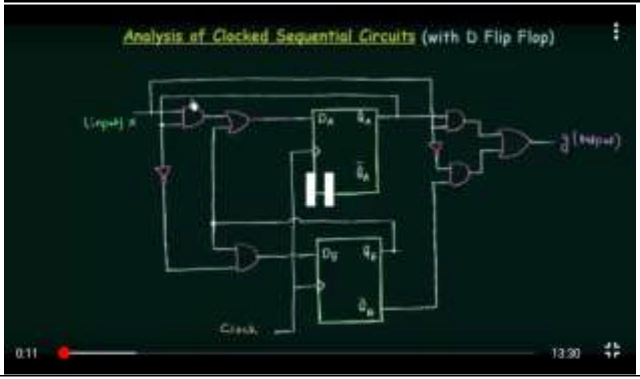
DAILY ASSESSMENT FORMAT

Date:	**29-05-2020**	**Name:**	**PRINCIA MELITA DSOUZA**
Course:	**LOGIC DESIGN**	**USN:**	**4AL17EC075**
Topic:	**Analysis of clocked sequential circuits & digital clock design**	**Semester & Section:**	**6TH B**
Github Repository:	**MELITA-1999**		

FORENOON SESSION DETAILS
Image of session


Analysis of Clocked Sequential Circuits (with D Flip Flop)
(input) x
D_A
Q_A
$\bar{Q}_A$
D_B
Q_B
$\bar{Q}_B$
Clock
y (output)
STEP:1 Find out the i/p & o/p eq^n
02:03
13:30



Report – Report can be typed or hand written for up to two pages.

CLOCKED SEQUENTIAL CIRCUITS

The behavior of a clocked sequential circuit is determined from its inputs, outputs and state of the flip-flops (i.e., the output of the flip-flops). The analysis of a clocked sequential circuit consists of obtaining a table of a diagram of the time sequences of inputs, outputs and states.

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

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 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.

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.

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 unchanged

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.

DIGITAL CLOCK DESIGN

Digital Clock Circuit Design Using 7493.The 4 blocks of a digital clock are 1 Hz clock generator to generate 1 PPS (pulse per second) signal to the seconds block.SECONDS block - contains a divide by 10 circuit followed by a divide by 6 circuit. Will generate a 1 PPM (pulse per minute) signal to the minutes block. The BCD outputs connect to the BCD to Seven Segment circuit to display the seconds values. MINUTES block - identical to the seconds block it contains 2 dividers; a divide by 10 followed by a divide by 6. Will generate a 1 PPH (pulse per hour) signal to the HOURS block. The BCD outputs connects to the BCD to Seven Segment circuit to display the minutes values.HOURS block - depending on whether it is a 12 or 24H clock, will have a divide 24 or divide by 12. For 24H, it will count from 00 to 23. For 12H, it will count from 00 to 11. The BCD outputs connects to the BCD to Seven Segment circuit to display the hours values.SECONDS blockThe 74LS93 is used to implement the divide by 10 and divide by 6 circuits. The 74LS93 is a high-speed 4-bit ripple type counters partitioned into two sections. The counter has a divide-by-two section and divide-by-eight section which are triggered by a HIGH-to-LOW transition on the clock inputs. Divide by 10 Counter In order to use all 4 bits of the counter, Q0 must be connected to CP1. Q0 is LSB and Q3 is MSB. The input clock is connected to CP0.To implement a divide by 10 or MOD10 counter, Q1 is connected to MR1 and Q3 is connected to MR2. With this connection, when the count reaches 10 (1010 binary), it resets to 0.The output frequency at Q3 is the input clock frequency divided by 10. To display the values, Q3..Q0 are connected to the respective D..A inputs of the BCD to 7 segment display.Divide by 6 Counter Since only 3 bits are required Q0 is not used. Q1 is LSB and Q3 is MSB.The input clock is connected to CP1.To implement a divide by 6 or MOD6 counter, Q2 is connected to MR1 and Q3 is connected to MR2. With this connection, when the count reaches 6 (110 binary), it resets to 0.The output frequency at Q3 is the input clock frequency divided by 6.
To display the values, Q3..Q1 are connected to the respective C..A inputs of the BCD to 7 segmentdisplay. D of the BCD to 7 segment display input is connected to GND.HOURS block
The clock can be designed as a 24H or 12H clock. We will explain the steps to arrive at the combinational logic to obtain a 12H clock and we will leave it to the reader to design the 24H clock as an exercise. Click hints if you need help to design the 24H clock.

12H Clock

In order to use all 4 bits of the IC1 (ones) counter, Q0 must be connected to CP1. Q0 is LSB and Q3 is MSB. The input clock is connected to CP0.
Since less than 3 bits are required for IC2 (tens), Q0 is not used. Q1 is LSB and Q3 is MSB. The input clock is connected to CP1.
The truth table of the counter is abbreviated - omitting those rows where the MR inputs to the counters are 0. Recall that for the 7493, a 1 to the MR will reset the counters to 0.
To simplify the table, K is Q0 of IC1 (ones), G is Q0 of IC2 (tens) and so on.
For the 12H clock, when the count in BCD reaches
0A, IC1 must be cleared (Y=1)

12, IC1 must be cleared (Y=1) and IC2 must be cleared (X=1)
Using SOP (sum of products), we obtain
Y = HJ + GJ where Y is the IC1 MR1, MR2 inputs connected together
X = GJ where X is the IC2 MR1, MR2 inputs connected together
Simulate and Breadboard the 24H Clock circuit.

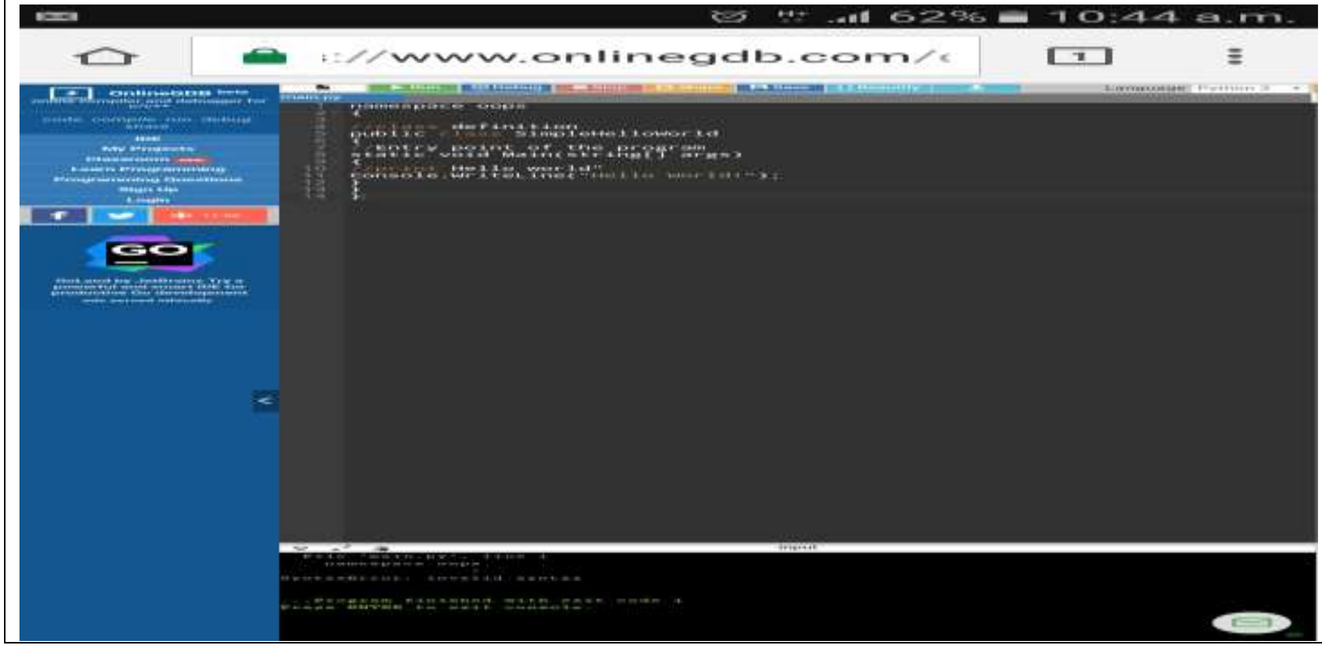
1 Hz Clock
The 1 Hz clock can be implemented using the schmitt trigger oscillator.

Limitations
The clock cannot be set to the correct time. Hint - use additional logic to allow the 1 PPS clock to drive the MINUTES and HOURS block depending on a button press. Below is the block diagram of one solution using a 2 to 1 multiplexer. Depending on SET, either the 1 PPS (Pulse Per Second) or the 1 PPH (Pulse Per Hour) clock drives the Hour circuit.The 12H clock counts from 00 to 11 rather than 01 to 12. Hint - use regular JK flip flops (74LS73) instead of the 74LS93 so on terminal count, the counter output is preset to 01.

Date:	**20-05-2020**	**Name:**	**PRINCIA MELITA DSOUZA**
Course:	**PYTHON**	**USN:**	**4AL17EC075**
Topic:	**OBJECT ORIENTED PROGRAMMING**	**Semester & Section:**	**6TH B**

AFTERNOON SESSION DETAILS

Image of session



10:44 a.m.

://www.onlinegdb.com/

```
namespace oops
{
//class definition
public class SimpleHelloWorld
{
//Entry point of the program
static void Main(string[] args)
{
//print Hello world"
Console.WriteLine("Hello World!");
}
}
}
```

```
  File "main.py", line 1
    namespace oops
SyntaxError: invalid syntax

...Program finished with exit code 1
Press ENTER to exit console.
```

Report – Report can be typed or hand written for up to two pages.

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.
Many of the most widely used programming languages (such as C++, Java, Python, etc.) are multi-paradigm and they support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming. Significant object-oriented languages include Java, C++, C#, Python, R, PHP, JavaScript, Ruby, Perl, Object Pascal, Objective-C, Dart, Swift, Scala, Kotlin, Common Lisp, MATLAB, and Smalltalk.

The Basic OOP Concepts
If you are new to object-oriented programming languages, you will need to know a few basics before you can get started with code. The following Webopedia definitions will help you better understand object-oriented programming:

Abstraction: The process of picking out (abstracting) common features of objects and procedures.
Class: A category of objects. The class defines all the common properties of the different objects that belong to it.
Encapsulation: The process of combining elements to create a new entity. A procedure is a type of encapsulation because it combines a series of computer instructions.
Information hiding: The process of hiding details of an object or function. Information hiding is a powerful programming technique because it reduces complexity.
Inheritance: a feature that represents the "is a" relationship between different classes.
Interface: the languages and codes that the applications use to communicate with each other and with the hardware.
Messaging: Message passing is a form of communication used in parallel programming and object-oriented programming.
Object: a self-contained entity that consists of both data and procedures to manipulate the data.
Polymorphism: A programming language's ability to process objects differently depending on their data type or class.
Procedure: a section of a program that performs a specific task.
Advantages of Object Oriented Programming
One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

OOPL - Object Oriented Programming Languages
An object-oriented programming language (OOPL) is a high-level programming language based on the object-oriented model. To perform object-oriented programming, one needs an object-oriented programming language. Many modern programming languages are object-oriented, however some older programming languages, such as Pascal, do offer object-oriented versions. Examples of object-oriented programming languages include Java, C++ and Smalltalk.

The First OOPL
Simula, developed in the 1960s at the Norwegian Computing Center in Oslo, is considered to be the first object-oriented programming language. Despite being first, Smaslltalk is considered to be the only true object-oriented programming environment and the one against which all others must be compared. It was first developed for educational use at Xerox Corporation's Palo Alto Research Center in the late 1960s and released in 1972.

Recommended Reading: Webopedia Study Guides - Java Basics: Variables, Syntax and Conventions and Java Basics Part 2: Operators, Modifiers and Structures.
Stackify in Developer Tips, Tricks & Resources
OOP Concepts in C#: Code Examples and How to Create a Class

Object oriented programming (OOP) is a programming structure where programs are organized around objects as opposed to action and logic. This is essentially a design philosophy that uses a different set of programming languages such as C#. Understanding OOP concepts can help make decisions about how you should design an application and what language to use.

Everything in OOP is placed together as self-sustainable "objects." An object is a combination of variables, functions, and data that performs a set of related activities. When the object performs those activities, it defines the object's behavior. In addition, an object is an instance of a class. Furthermore, C# offers full support for OOP including inheritance, encapsulation, abstraction, and polymorphism:

Encapsulation is when a group of related methods, properties, and other members are treated as a single object.
Inheritance is the ability to receive ("inherit") methods and properties from an existing class.
Polymorphism is when each class implements the same methods in varying ways, but you can still have several classes that can be utilized interchangeably.
Abstraction is the process by which a developer hides everything other than the relevant data about an object in order to simplify and increase efficiency.
We'll discuss each of these concepts in more detail in this post.

What is an Object?
Objects are instances of classes. In other words, an instance of a class is an object defined by that particular class. Creating a new instance, or an object, is called instantiation. This is how you define a class:

C#

```
class SampleClass

{

}
```
A light version of classes in C# is called structures. These are beneficial when you want to create a large array of objects but don't want to overwhelm your available memory. A class is made up of three things:

A name
Operations
Attributes
Objects are created from a class blueprint, which defines the data and behavior of all instances of that type.

```
public class Student

{

}
```

Here’s another example:

```
Class Example

{

/* fields,

Variables,

Methods,

Properties,

*/

}
```

Here’s an example of an object:

```
Example exmplObject = new Example();
```

In memory, you can create an object using the “new” keyword. In C#, value types refer to other data type variables while objects are reference types. Moreover, other value types are stored in the stack while objects are stored in the heap.

Keep in mind that everything in C# is a class. An object is a section of memory that has been configured according to the class blueprint. Every instance, or object, of a particular class has access to several methods and properties of that class.

Data Abstraction
This provides essential features without describing any background details. Abstraction is important because it can hide unnecessary details from reference objects to names. It is also necessary for the construction of programs. Instead of showing how an object is represented or how it works, it focuses on what an object does. Therefore, data abstraction is often used for managing large and complex programs.

Encapsulation
This binds the member function and data member into a single class. This also allows for abstraction.

Within OOP, encapsulation can be achieved through creating classes. Those classes then display public methods and properties. The name encapsulation comes from the fact that this class encapsulates the set of methods, properties, and attributes of its functionalities to other classes.

Polymorphism
This is the ability of an object to perform in a wide variety of ways. There are two types:

1. Dynamic polymorphism (runtime time). You can obtain this type through executing function overriding.

2. Static polymorphism (compile time). You can achieve static polymorphism through function overloading and operator overloading.

Within OOP, polymorphism can be achieved using many techniques including:

Method overloading (defining several methods at the same time)
Method overriding (this allows a subclass to override a specific implementation of a method already issued by one of its super-classes)
Operator overloading (some or all of the operators are handled has polymorphic functions with different behaviors depending on the types of its arguments)
Inheritance
Through inheritance, a class can "inherit" the characteristics of another general class. To illustrate, dogs are believed to be the descendants of wolves. All dogs have four paws and can hunt their prey. This function can be coded into the Wolf class. All of its descendants can use it. Inheritance is also an is-kind-of relationship. For instance, a golden retriever is a kind of animal. Here's an example:

```
class BaseClass

{

}

class DerivedClass : BaseClass

{

}
```
How to Create a Class
All you have to do to create a class is to add a class file to your project. The next step is to right-click on your project within the solution explorer and click Add, then choose New Item. You'll see a new window. On the left side of the window, click Class in the Code template. Select a name for your class and click Add. It looks like this:

```
namespace OOPLearning

{

class Cat
{

}

}
```
A method is an action an object can execute. In most instances, you can declare a method within a class definition. Yet, C# supports extension methods that let you add methods to an existing class outside the definition of a class.

Here's an example of a simple C# program called "Hello World" from C# Corner that illustrates many of OOP fundamentals:

```
using System;

namespace oops
{

//class definition
public class SimpleHelloWorld
{
//Entry point of the program
static void Main(string[] args)
{
//print Hello world"
Console.WriteLine("Hello World!");
}
}
}
```
In order for a programming language to be object-oriented, it must have the ability to work with classes and objects. Moreover, it must use the fundamental object-oriented principles of abstraction, inheritance, polymorphism, and encapsulation.

Additional Resources and Tutorials
For more information on OOP concepts in C# and how you can best utilize OOP, visit the following resources and tutorials:

C# Object-Oriented Programming Tutorial
Object-oriented programming
Introduction to C# classes
Object oriented programming with C#
Summary
Going into 2018, we expect C# and .NET Core to be huge, so it's worth learning the fundamentals if you're not already familiar. Check out some of our other posts for more basics and advanced concepts in C#, such as throwing C# exceptions, how to handle C# exceptions, catching them and finding application errors, and other tips