DAILY ASSESSMENT FORMAT

Date:	29/05/2020	Name:	Shilpa N
Course:	Logic Design	USN:	4AL16EC071
Topic:	DAY 2	Semester & Section:	8 "B"
Github Repository:	Shilpan-test		

FORENOON SESSION DETAILS

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

Sequential circuits

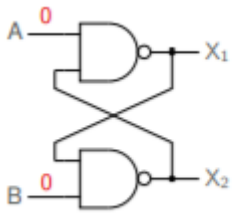
*The digital circuits we have seen so far (gates, multiplexer, demultiplexer, encoders, decoders) are combinatorial in nature, i.e., the output(s) depends only on the present values of the inputs and not on their past values.

* In sequential circuits, the "state" of the circuit is crucial in determining the output values. For a given input combination, a sequential circuit may produce different output values, depending on its previous state.

* In other words, a sequential circuit has a memory (of its past state) whereas a combinatorial circuit has no memory.

* Sequential circuits (together with combinatorial circuits) make it possible to build several useful applications, such as counters, registers, arithmetic/logic unit (ALU), all the way to microprocessors.

NAND latch (RS latch)



A	B	X_1	X_2
1	0	0	1
0	1	1	0
1	1	previous	
0	0		

* A, B: inputs, X1, X2: outputs

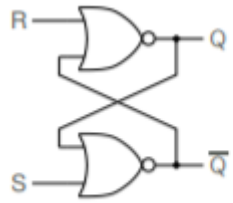
* Consider A = 1, B = 0. B = 0 ⇒ X2 = 1 ⇒ X1 = A X2 = 1 · 1 = 0. Overall, we have X1 = 0, X2 = 1.

* Consider A = 0, B = 1. → X1 = 1, X2 = 0.

* Consider A = B = 1. X1 = A X2 = X2, X2 = B X1 = X1 ⇒ X1 = X2 If X1 = 1, X2 = 0 previously, the circuit continues to “hold” that state. Similarly, if X1 = 0, X2 = 1 previously, the circuit continues to “hold” that state. The circuit has “latched in” the previous state.

* For A = B = 0, X1 and X2 are both 1. This combination of A and B is not allowed for reasons that will become clear later.

NOR latch (RS latch)



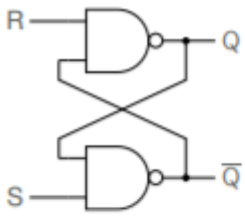
R	S	Q	$\overline{Q}$
1	0	0	1
0	1	1	0
0	0	previous	
1	1	invalid	

* The NOR latch is similar to the NAND latch: When R = 1, S = 0, the latch gets reset to Q = 0. When R = 0, S = 1, the latch gets set to Q = 1.

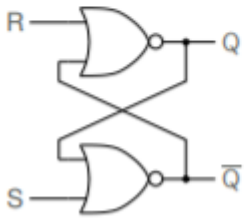
* For R = S = 0, the latch retains its previous state (i.e., the previous values of Q and Q).

* R = S = 1 is not allowed for reasons similar to those discussed in the context of the NAND latch.

Comparison of NAND and NOR latches

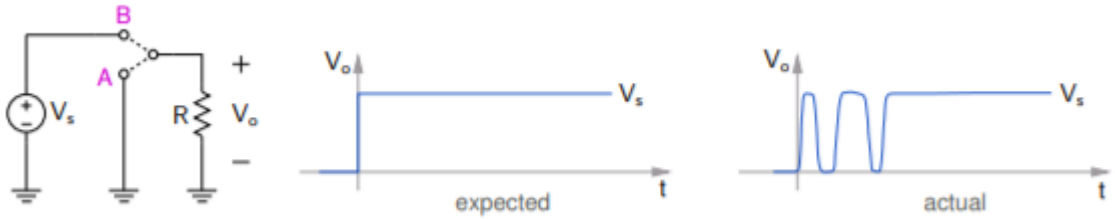


R	S	Q	$\overline{Q}$
1	0	0	1
0	1	1	0
1	1	previous	
0	0	invalid	



R	S	Q	$\overline{Q}$
1	0	0	1
0	1	1	0
0	0	previous	
1	1	invalid	

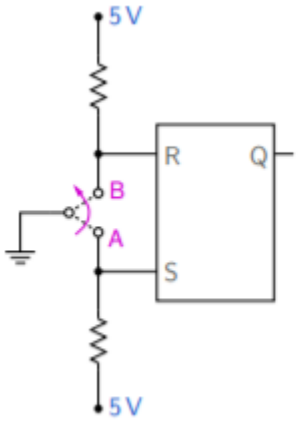
Chatter (bouncing) due to a mechanical switch



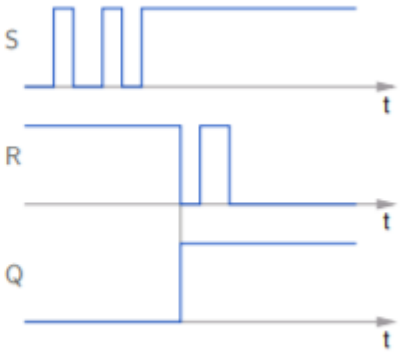
* When the switch is thrown from A to B, Vo is expected to go from 0 V to Vs (say, 5 V).

* However, mechanical switches suffer from “chatter” or “bouncing,” i.e., the transition from A to B is not a single, clean one. As a result, Vo oscillates between 0 V and 5 V before settling to its final value (5 V).

* In some applications, this chatter can cause malfunction → need a way to remove the chatter.



R	S	Q	$\overline{Q}$
1	0	0	1
0	1	1	0
1	1	previous	
0	0	invalid	



* Because of the chatter, the S and R inputs may have multiple transitions when the switch is thrown from A to B.

* However, for S = R = 1, the previous value of Q is retained, causing a single transition in Q, as desired.

CLOCK

* Complex digital circuits are generally designed for synchronous operation, i.e., transitions in the various signals are synchronized with the clock.

* Synchronous circuits are easier to design and troubleshoot because the voltages at the nodes (both output nodes and internal nodes) can change only at specific times.

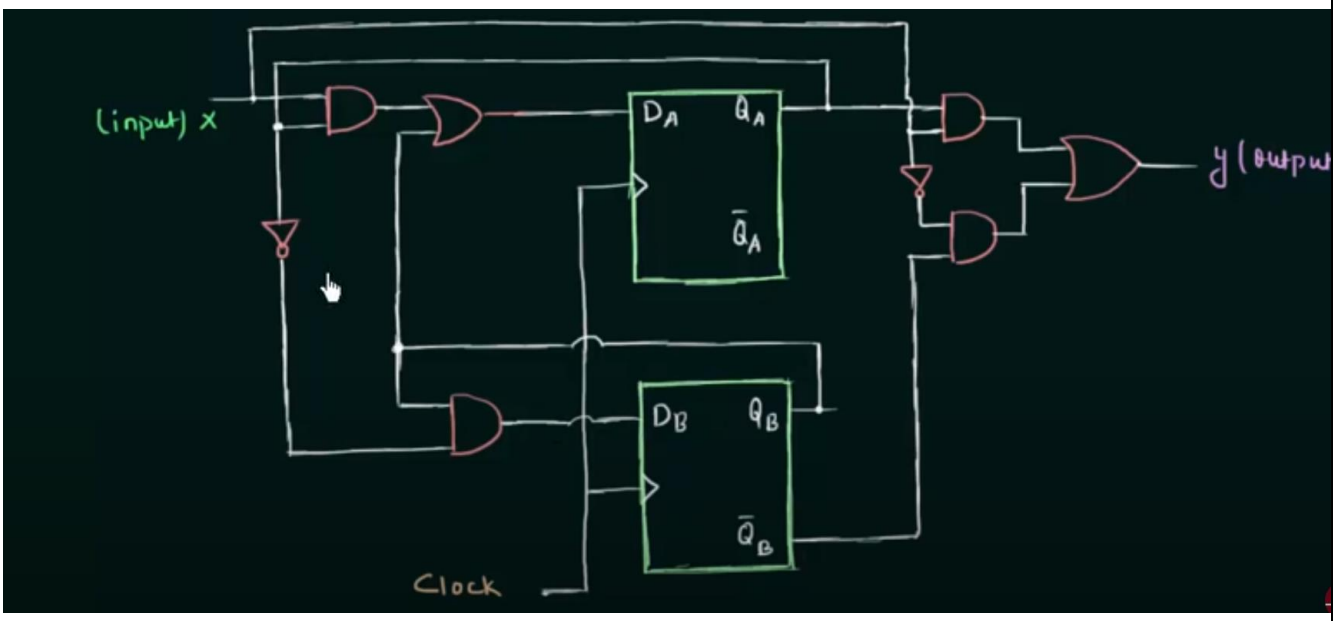
* A clock is a periodic signal, with a positive-going transition and a negative-going transition.

* The clock frequency determines the overall speed of the circuit. For example, a processor that operates with a 1 GHz clock is 10 times faster than one that operates with a 100 MHz clock. Intel 80286 (IBM PC-AT): 6 MHz Modern CPU chips: 2 to 3 GHz.

Analysis of clocked sequential circuits

Optimum goal of the presentation is o find state diagram of the circuits.

1)D-FLIP FLOP



Step1 – find out input/output equation

$D_A = XQ_A+Q_B$

$D_B=Q_A'Q_B$

$Y=X' Q_B'+ XQ_A$

Steo2 – state table

The value of $D=Q_{n+1}$(next state)

Q_A+= D_A= XQ_A+Q_B

Q_B+= D_B=Q_A'Q_B

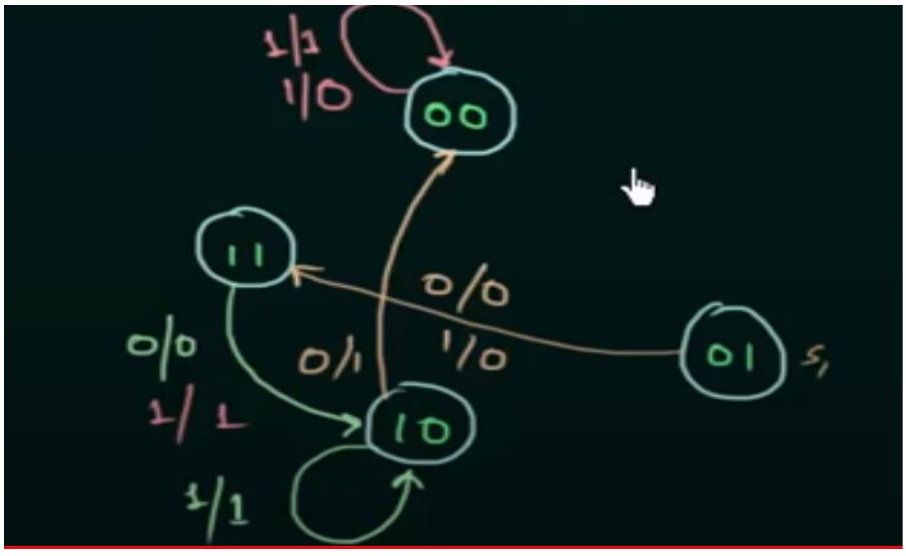
using these values, we find out next state

PRESENT STATE			NEXT STATE		
Q_A	Q_B	X	Q_A+	Q_B+	Y
0	0	0	0	0	1
0	0	1	0	0	0
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	1	0	0
1	1	1	1	0	1

Step3 – State Diagram

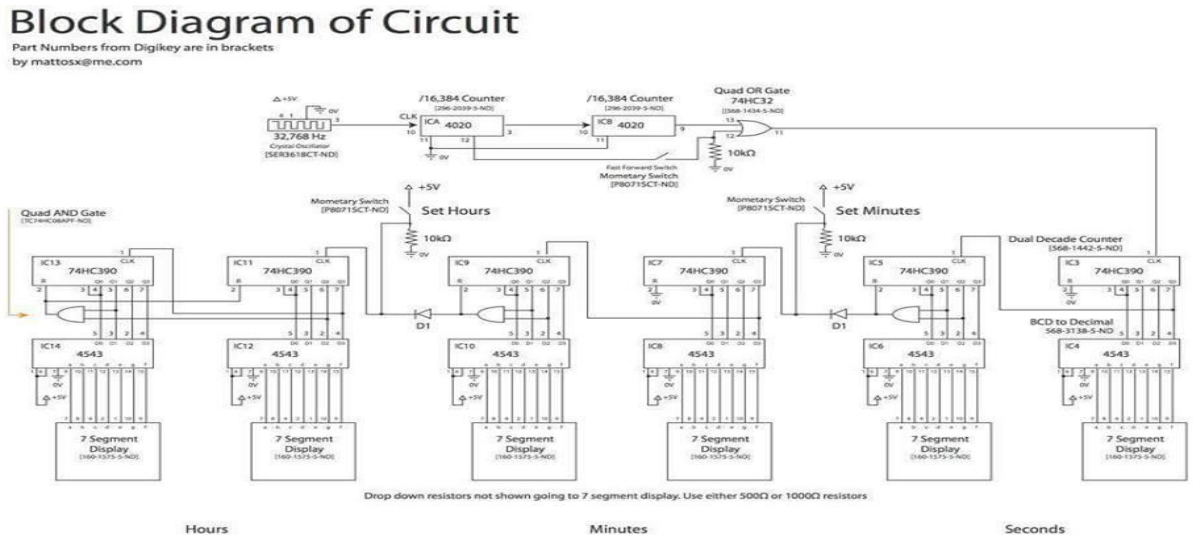
there are 2 flip flops so there are 4 possible states

	Q_A	Q_B
S0	0	0
S1	0	1
S2	1	0
S3	1	1



DIGITAL CLOCK DESIGN

24h /12h Digital clock using ic555 ,7490-decade counters and 7447 bcd-7seg decoders



The main parts of the circuit are as follows:

1- Timer 555: Responsible for generating the clock pulses for the counters, the frequency of the output should be 1 hz which means 1 second for each pulse.

2- Counters: Responsible for generating the time in BCD (Binary Coded decimal).

3- Decoders: Takes the BCD of the counter as input and produces 7 segment output.

4- 7 segments: Displays the time, of course!

* Seconds have 2 displays, 2 decoders and 2 counters. The same for minutes and hours.

we should know before how the circuit is going to work is that 7490-decade counters respond only to the negative going edge of 555 pulses, which means it will change its state only when the clock goes from 1 to 0.

The circuit works as follows:

555 timer produces 1 second pulses to the clock input of the first counter which is responsible for the first column of seconds, so its output will change every second.
The counter produces numbers from 0 to 9 in BCD form and automatically resets to 0 after that.
so, the output of the first counter will count from 0 to 9 every second and that's exactly what we want from it, so we are done here. let's move to the next one.

What do we want in the circuit is,
First, we want the 2nd counter to start counting when the 1st one moves for 9 to 0 (that makes 10 seconds!)

This can be done as shown,
let's check the output of the fist counter in BCD:

MSB---LSB
0: 0000
1: 0001
2: 0010
3: 0011
4: 0100
5: 0101
6: 0110
7: 0111
8: 1000
9: 1001
0: 0000

Remember that 7490-decade counters respond only to the pulses that go from 1 to 0 and notice that this case only happens in the BCD code above when the output changes from 9 to 0 (the Most significant bit changes from 1 to 0). So, we'll just connect the clock input of the 2nd counter to the most significant bit of the output of the first counter.

Second, Since we have 60 seconds in the minute we want the 2nd counter to count only to 5, that makes 59 seconds maximum, when it take another pulse it doesn't count to 60, instead it resets itself to 0 and send a pulse to the first counter in minutes to tell it to count 1 minute

This can be done, From the BCD code above (6: 0110) when the output is 6 the two middle bits are 1 (Q1,Q2),
So By ANDing these two bits the output will be 1, This output will be connected to the reset pin of the same counter (2nd one) and the clock input of the next one(3rd).
When the output is 6 the AND gate output (1) will reset the same counter and its outputs goes 0000 so the output of the and gate again goes to 0 (1---->0), that will clock the next counter. Beautiful!

*Notice that the output of the counters are named: Q0 , Q1 , Q2 , Q3

The 4th counter will be the same as the second one so we are clocking it using the Most Significant Bit of the output of the previous one.

Again, the 5th counter is the same as the 3rd one and takes its clock from the AND gate.

The 5th and the 6th counters are responsible for hours so they are limited to 23, and resets themselves to 00 when the 5th counter is 4 and the last one is 2 (24).
This is done using and gate with Q2 (3rd bit) of the 5th counter as one input and Q1 (second bit) of the last counter as the other input, and the output of this AND gate will be connected to both resets of the last 2 counters.

When the last counter is 0(0000) or 1(0001), Q1 which is one of the inputs to the AND gate will be 0 so the output of the AND gate will be zero. when it counts to 2 this bit will be 1 so the output of the and gate will depend on the the other input which is Q2 of the previous counter, and this bit will be zero until it reaches 4 (0100),So, the output of the and gate will be 1 (0--->1) resetting both counters to 00,

The output of these counters is converted to 7 segment output using 7447 decoders, then to the 7 segments, we won't get into the details of their datasheets.

BONUS TUTORIAL-SIMPLIFYING THE BRAIN

The session was by V. Srinivasa Chakravarthy, department of biotechnology, IIT madras,The numbers of neurons in the brain are more than number of stars in the milky way galaxy. The brain is the most complex object in the universe. Reasons for neuroscience include Large projects are there in neuroscience, which needs large number of data. Descriptive tradition of biology is also reason for neuroscience. We do structural analysis then functional analysis to fix a radio by observing its PCB. Discussed about a architecture of a radio in the engineer point of view. Data as the fourth pillar of science. The firs 3 are theory, experiment, computation and last is data pillar. Neurons come in different shapes. The updated technologies which uses big data is the reason the neuroscience make brain look complexed. Briefed about neuromorph. The connectome project aims at working on the graph structure of the brain. Human brain has 100 billion neurons. Human connectome project includes 1200individuals, IMRI+dMRI+MRI+MEG, Washington U+ U. Minnesota. Discussed about history of astronomy like Newtonian astronomy and Ptolemaic astronomy. The problem with computational modelling includes includes more biology-style modelling, less physics style modelling, a thin rapper over data, need deep integrative theories that relate diverse phenomena. Discussed about contemporary neuroscience and cortex-subcortex. Modeling the basal ganglia which has multiple structure and has 7 nuclei. Cortex has many areas like visual cortex, motor cortex, sensory cortex. Explained about Functional anatomy of basal ganglia and Basal ganglia functions.

Date:	29/05/2020	Name:	Shilpa N
Course:	PYTHON	USN:	4AL16EC071
Topic:	DAY 11	Semester & Section:	8 “B”
AFTERNOON SESSION DETAILS			

Python Object Oriented Programming

We learn about the Object-Oriented Programming (OOP) in Python and their fundamental concept with examples.

Introduction to OOPs in Python

Python is a multi-paradigm programming language. Meaning, it supports different programming approach.

One of the popular approach to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).

An object has two characteristics:

attributes

behavior

Let's take an example:

Parrot is an object,

name, age, color are attributes

singing, dancing are behavior

The concept of OOP in Python focuses on creating reusable code. This concept is also known as DRY (Don't Repeat Yourself).

In Python, the concept of OOP follows some basic principles:

Class

A class is a blueprint for the object.

We can think of class as an sketch of a parrot with labels. It contains all the details about the name, colors, size etc. Based on these descriptions, we can study about the parrot. Here, parrot is an object.

The example for class of parrot can be :

```
class Parrot:
    pass
```

Here, we use `class` keyword to define an empty class `Parrot`. From class, we construct instances. An instance is a specific object created from a particular class.

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, `obj` is object of class `Parrot`.

Suppose we have details of parrot. Now, we are going to show how to build the class and objects of parrot.

Creating Class and Object in Python

```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))

# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

When we run the program, the output will be:

```
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
```

In the above program, we create a class with name `Parrot`. Then, we define attributes. The attributes are a characteristic of an object.

Then, we create instances of the `Parrot` class. Here, `blu` and `woo` are references (value) to our new objects.

Then, we access the class attribute using `__class __.species`. Class attributes are same for all instances of a class. Similarly, we access the instance attributes using `blu.name` and `blu.age`. However, instance attributes are different for every instance of a class.

To learn more about classes and objects, go to Python Classes and Objects

Methods

Methods are functions defined inside the body of a class. They are used to define the behaviors of an object.

Creating Methods in Python

```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
```

```
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)

# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())
```

When we run program, the output will be:

```
Blu sings 'Happy'
Blu is now dancing
```

In the above program, we define two methods i.e `sing()` and `dance()`. These are called instance method because they are called on an instance object i.e `blu`

Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

Use of Inheritance in Python

```
# parent class
class Bird:

    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")
```

```
    def swim(self):
        print("Swim faster")

# child class
class Penguin(Bird):

    def __init__(self):
        # call super() function
        super().__init__()
        print("Penguin is ready")

    def whoisThis(self):
        print("Penguin")

    def run(self):
        print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

When we run this program, the output will be:

```
Bird is ready
Penguin is ready
Penguin
Swim faster
Run faster
```

Encapsulation

Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ _ “ or double “ __“.

Data Encapsulation in Python

```
class Computer:
```

```
    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()
```

When we run this program, the output will be:

```
Selling Price: 900
Selling Price: 900
Selling Price: 1000
```

Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

Using Polymorphism in Python

```
class Parrot:

    def fly(self):
        print("Parrot can fly")

    def swim(self):
```

```
        print("Parrot can't swim")

class Penguin:

    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

#instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

When we run above program, the output will be:

```
Parrot can fly
Penguin can't fly
```

Key Points to Remember:

The programming gets easy and efficient.

The class is sharable, so codes can be reused.

The productivity of programmers increases

Data is safe and secure with data abstraction.