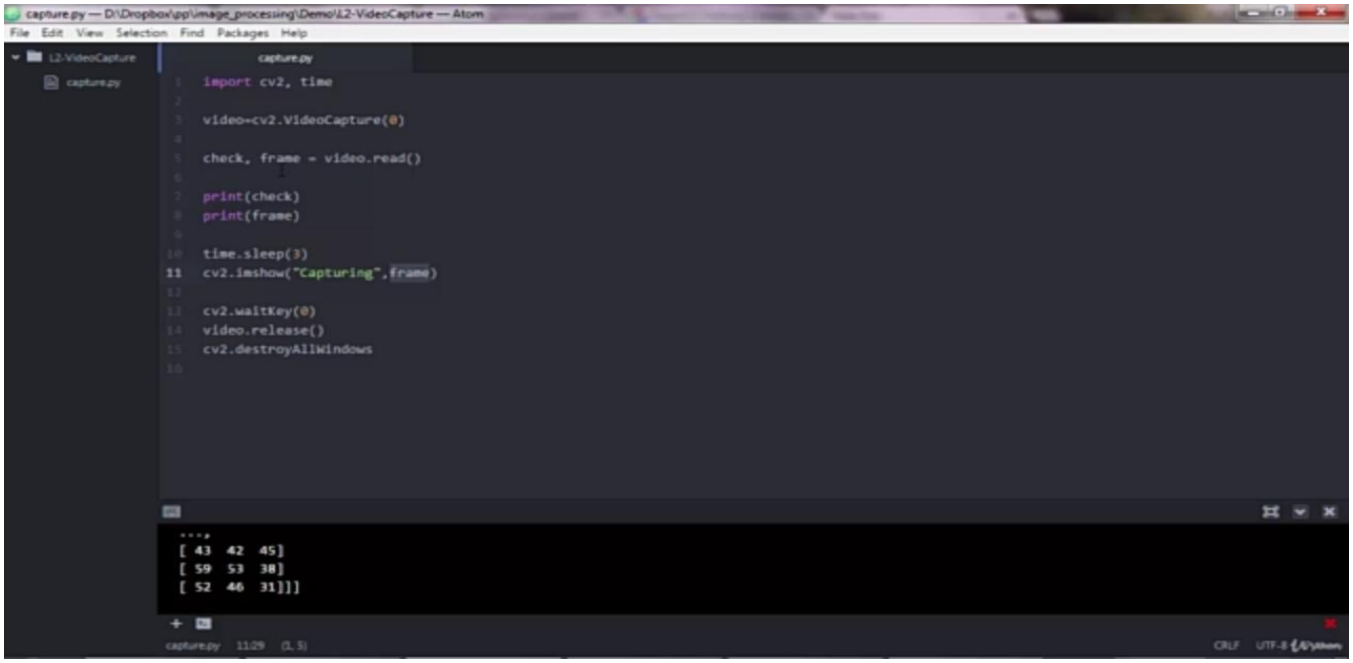
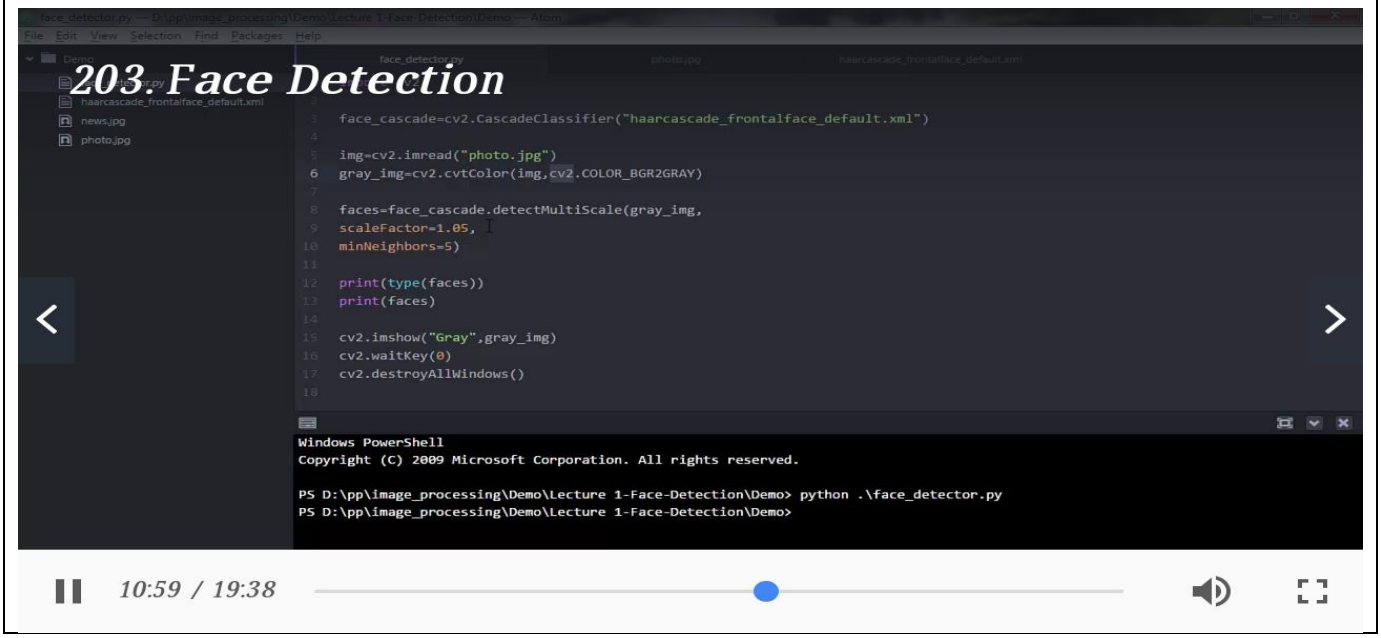
DAILY ASSESSMENT FORMAT

Date:	29th May 2020	**Name:**	Soundarya NA
Course:	UDEMY	**USN:**	4AL16EC077
Topic:	PYTHON: Python for Image and Video Processing with OpenCV	**Semester & Section:**	8th - B

FORENOON SESSION DETAILS

Image of session





Report:

Introduction:

Processing a video means, performing operations on the video frame by frame. Frames are nothing but just the particular instance of the video in a single point of time. We may have multiple frames even in a single second. Frames can be treated as similar to an image. So, whatever operations we can perform on images can be performed on frames as well.

Some of operations with examples:

Adaptive Threshold:

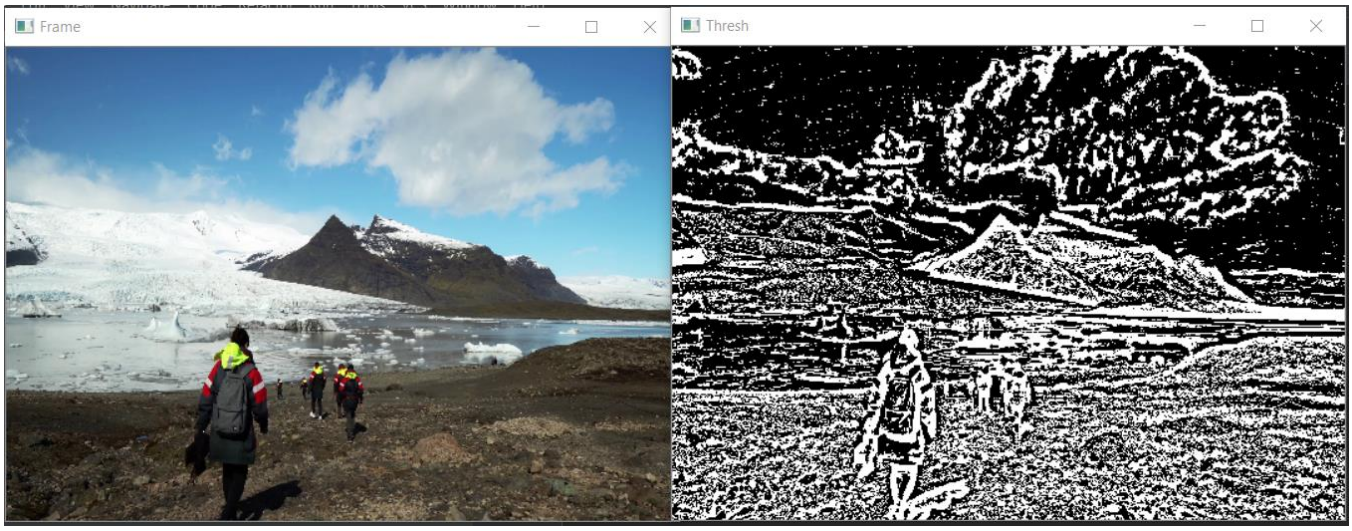
By using this technique, we can apply thresholding on small regions of the frame. So the collective value will be different for the whole frame.

Code:

```
# importing the necessary libraries
import cv2
import numpy as np
# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')
# Loop untill the end of the video
while (cap.isOpened()):
   # Capture frame-by-frame
   ret, frame = cap.read()
   frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                      interpolation = cv2.INTER_CUBIC)
   # Display the resulting frame
   cv2.imshow('Frame', frame)
   # conversion of BGR to grayscale is necessary to apply this operation
   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   # adaptive thresholding to use different threshold
   # values on different regions of the frame.
   Thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                  cv2.THRESH_BINARY_INV, 11, 2)
```

```
    cv2.imshow('Thresh', Thresh)
    # define q as the exit button
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:



Smoothening:

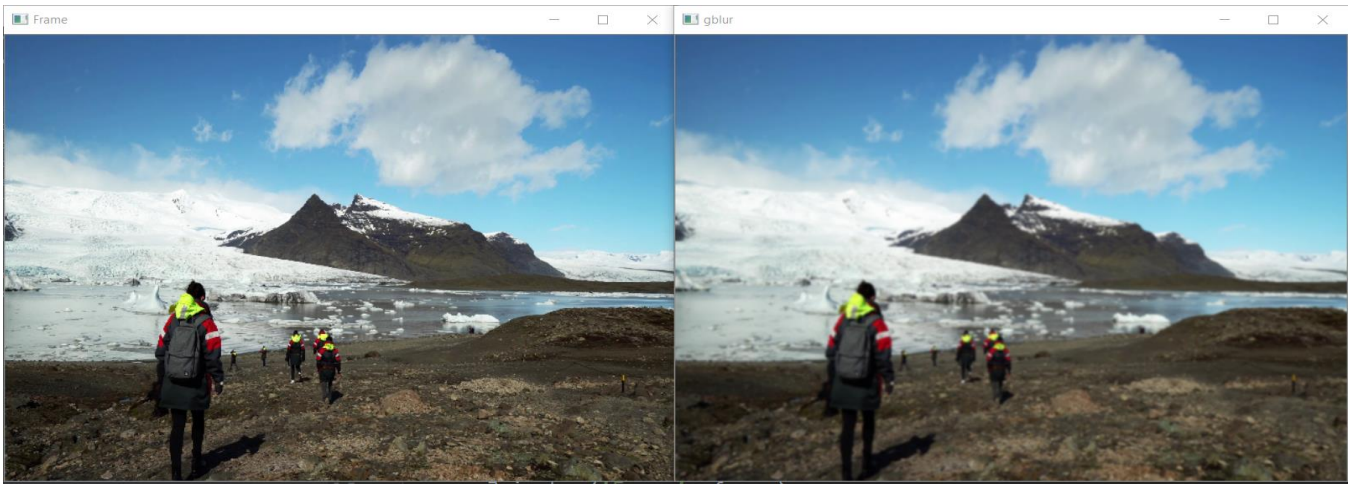
Smoothing a video means removing the sharpness of the video and providing a blurriness to the video. There are various methods for smoothing such as cv2.Gaussianblur(), cv2.medianBlur(), cv2.bilateralFilter(). For our purpose, we are going to use cv2.Gaussianblur().

Code:

```
# importing the necessary libraries
import cv2
import numpy as np
# Creating a VideoCapture object to read the video
```

```
cap = cv2.VideoCapture('sample.mp4')
# Loop untill the end of the video
while (cap.isOpened()):
   # Capture frame-by-frame
   ret, frame = cap.read()
   frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                      interpolation = cv2.INTER_CUBIC)
   # Display the resulting frame
   cv2.imshow('Frame', frame)
   # using cv2.Gaussianblur() method to blur the video
   # (5, 5) is the kernel size for blurring.
   gaussianblur = cv2.GaussianBlur(frame, (5, 5), 0)
   cv2.imshow('gblur', gaussianblur)
   # define q as the exit button
   if cv2.waitKey(25) & 0xFF == ord('q'):
      break
# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:



Edge Detection:

Edge detection is a useful technique to detect he edges of surfaces and objects in the video. Edge detection involves the following steps:

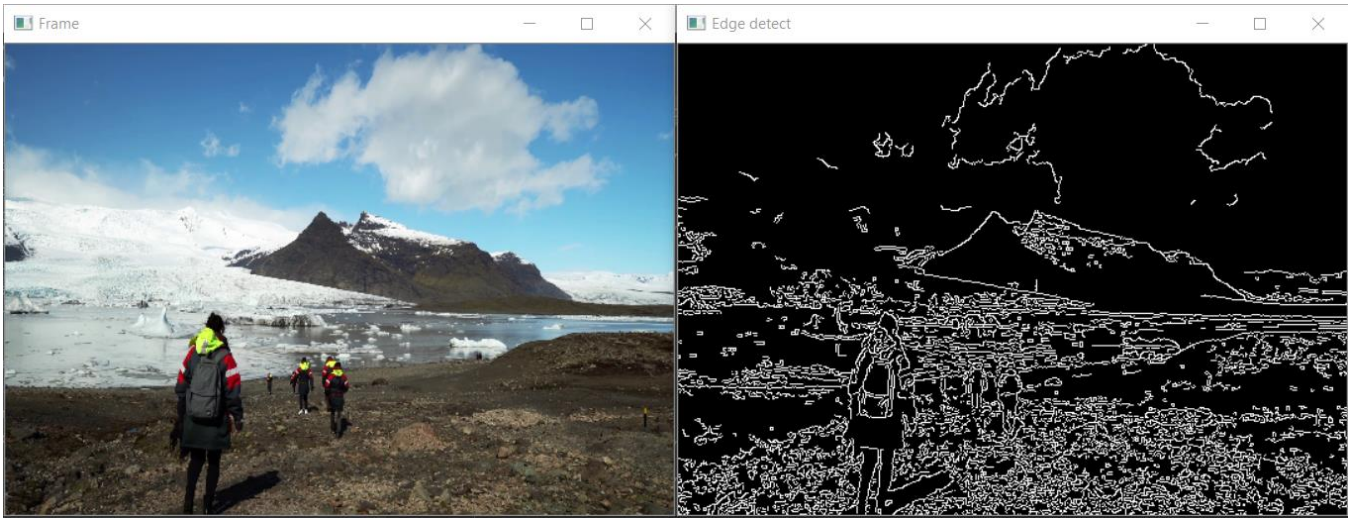
- Noise reduction
- Gradient calculation
- Non-maximum suppression
- Double threshold
- Edge tracking by hysteresis

Code:

```
# importing the necessary libraries
import cv2
import numpy as np
# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')
# Loop untill the end of the video
while (cap.isOpened()):
   # Capture frame-by-frame
   ret, frame = cap.read()
   frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                      interpolation = cv2.INTER_CUBIC)
   # Display the resulting frame
   cv2.imshow('Frame', frame)
   # using cv2.Canny() for edge detection.
   edge_detect = cv2.Canny(frame, 100, 200)
   cv2.imshow('Edge detect', edge_detect)
   # define q as the exit button
   if cv2.waitKey(25) & 0xFF == ord('q'):
      break
# release the video capture object
cap.release()
```

```
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:



Bitwise Operation:

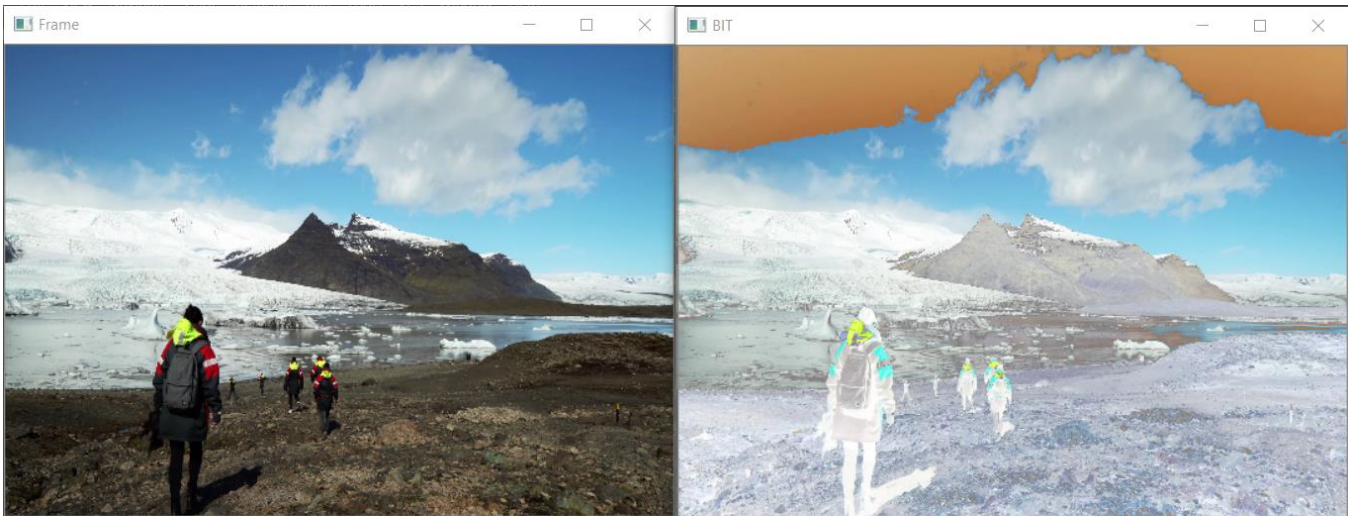
Bitwise operations are useful to mask different frames of a video together. Bitwise operations are just like we have studied in the classroom such as AND, OR, NOT, XOR.

Code:

```
# importing the necessary libraries
import cv2
import numpy as np
# Creating a VideoCapture object to read the video
cap = cv2.VideoCapture('sample.mp4')
# Loop untill the end of the video
while (cap.isOpened()):
        # Capture frame-by-frame
        ret, frame = cap.read()
        frame = cv2.resize(frame, (540, 380), fx = 0, fy = 0,
                                                interpolation = cv2.INTER_CUBIC)
        # Display the resulting frame
```

```
        cv2.imshow('Frame', frame)
        # conversion of BGR to grayscale is necessary to apply this operation
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        _, mask = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
        # apply NOT operation on image and mask generated by thresholding
        BIT = cv2.bitwise_not(frame, frame, mask = mask)
        cv2.imshow('BIT', BIT)
        # define q as the exit button
        if cv2.waitKey(25) & 0xFF == ord('q'):
                break
# release the video capture object
cap.release()
# Closes all the windows currently opened.
cv2.destroyAllWindows()
```

Output:



Loading and Displaying an Image:

```
# import the necessary packages
import cv2
# load the image and show it
```

```
image = cv2.imread("jurassic-park-tour-jeep.jpg")
cv2.imshow("original", image)
cv2.waitKey(0)
```

Getting the Dimensions of the Image:

```
# print the dimensions of the image
print image.shape
```

Resizing the Image using Python and OpenCV:

```
# we need to keep in mind aspect ratio so the image does
# not look skewed or distorted -- therefore, we calculate
# the ratio of the new image to the old image
r = 100.0 / image.shape[1]
dim = (100, int(image.shape[0] * r))
# perform the actual resizing of the image and show it
resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
cv2.imshow("resized", resized)
cv2.waitKey(0)
```

Rotating an Image using Python and OpenCV:

```
# grab the dimensions of the image and calculate the center
# of the image
(h, w) = image.shape[:2]
center = (w / 2, h / 2)
# rotate the image by 180 degrees
M = cv2.getRotationMatrix2D(center, 180, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
cv2.imshow("rotated", rotated)
cv2.waitKey(0)
```

Cropping an Image using Python and OpenCV:

```
# crop the image using array slices -- it's a NumPy array
# after all!
cropped = image[70:170, 440:540]
cv2.imshow("cropped", cropped)
cv2.waitKey(0)
```

Saving an Image using Python and OpenCV:

```
# write the cropped image to disk in PNG format
cv2.imwrite("thumbnail.png", cropped)
```

Batch Image Resizing:

```
import os
import argparse
from PIL import Image
DEFAULT_SIZE = (320, 180)
def resize_image(input_dir, infile, output_dir="resized", size=DEFAULT_SIZE):
    outfile = os.path.splitext(infile)[0] + "_resized"
    extension = os.path.splitext(infile)[1]
    try:
        img = Image.open(input_dir + '/' + infile)
        img = img.resize((size[0], size[1]), Image.LANCZOS)
        new_file = output_dir + "/" + outfile + extension
        img.save(new_file)
    except IOError:
        print("unable to resize image {}".format(infile))
if __name__ == "__main__":
    dir = os.getcwd()
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--input_dir', help='Full Input Path')
```

```
    parser.add_argument('-o', '--output_dir', help='Full Output Path')
    parser.add_argument('-w', '--width', help='Resized Width')
    parser.add_argument('-t', '--height', help='Resized Height')
    args = parser.parse_args()
    if args.input_dir:
        input_dir = args.input_dir
    else:
        input_dir = dir + '/images'
    if args.output_dir:
        output_dir = args.output_dir
    else:
        output_dir = dir + '/resized'
    if args.width and args.height:
        size = (int(args.width), int(args.height))
    else:
        size = DEFAULT_SIZE
    if not os.path.exists(os.path.join(dir, output_dir)):
        os.mkdir(output_dir)
    try:
        for file in os.listdir(input_dir):
            resize_image(input_dir, file, output_dir, size=size)
    except OSError:
        print('file not found')
```

Face Recognition:

```
import cv2
# Get user supplied values
imagePath = sys.argv[1]
cascPath = sys.argv[2]
# Create the haar cascade
```

```
faceCascade = cv2.CascadeClassifier(cascPath)
# Read the image
image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Detect faces in the image
faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags = cv2.cv.CV_HAAR_SCALE_IMAGE
)
print "Found {0} faces!".format(len(faces))

# Draw a rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
2.imshow("Faces found", image)
cv2.waitKey(0)
```

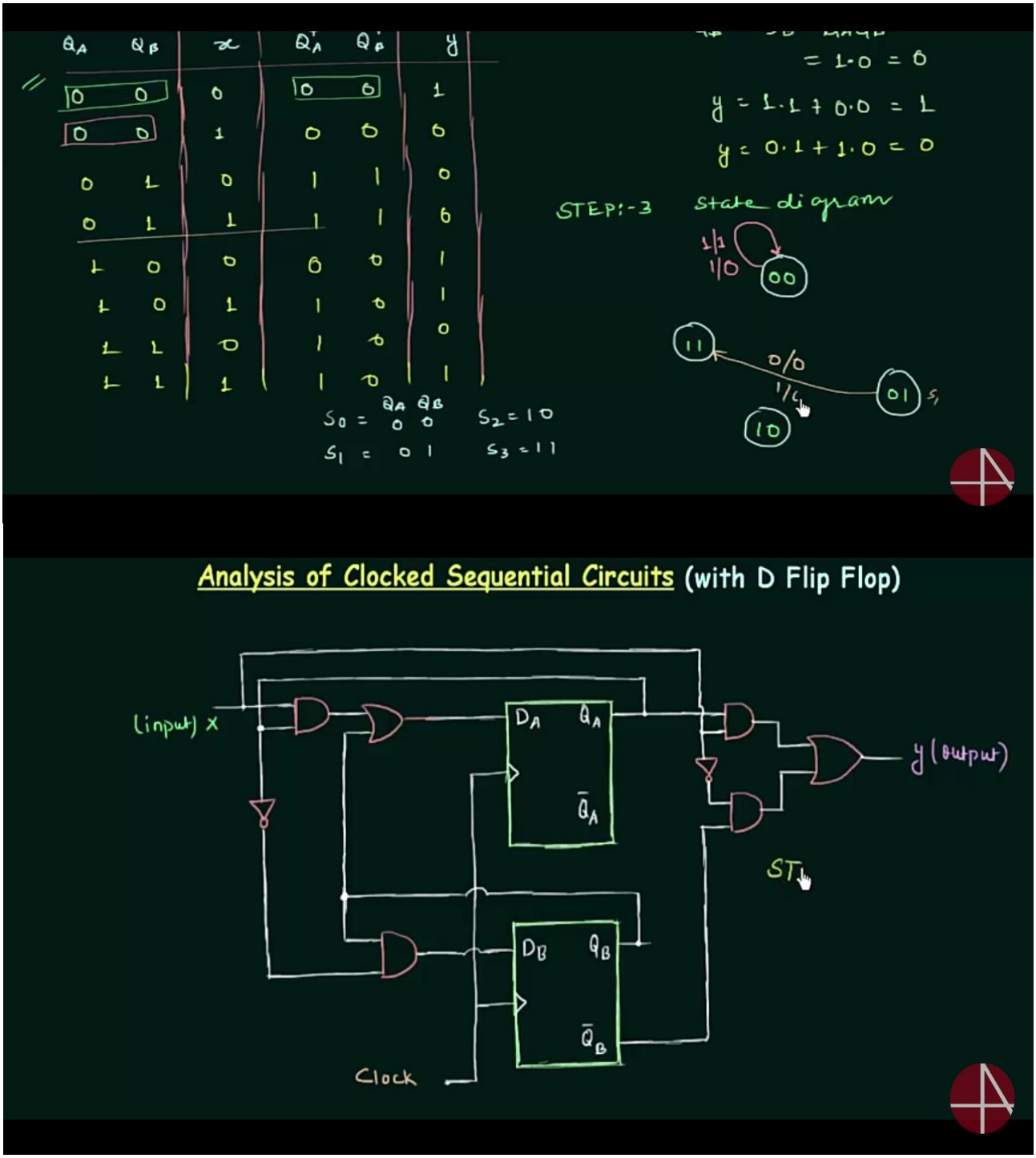
Capturing video:

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
while(True):
    # Capture frame-by-frame
    ret, frame = cap.read()
    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Display the resulting frame
```

```
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Date:	28th May 2020	Name:	Soundarya NA
Course:	Logic Design	USN:	4AL16EC077
Topic:	Logic Design Day 2	Semester & Section:	8th - B

Image:



Report:

Analysis of Clocked Sequential Circuits:

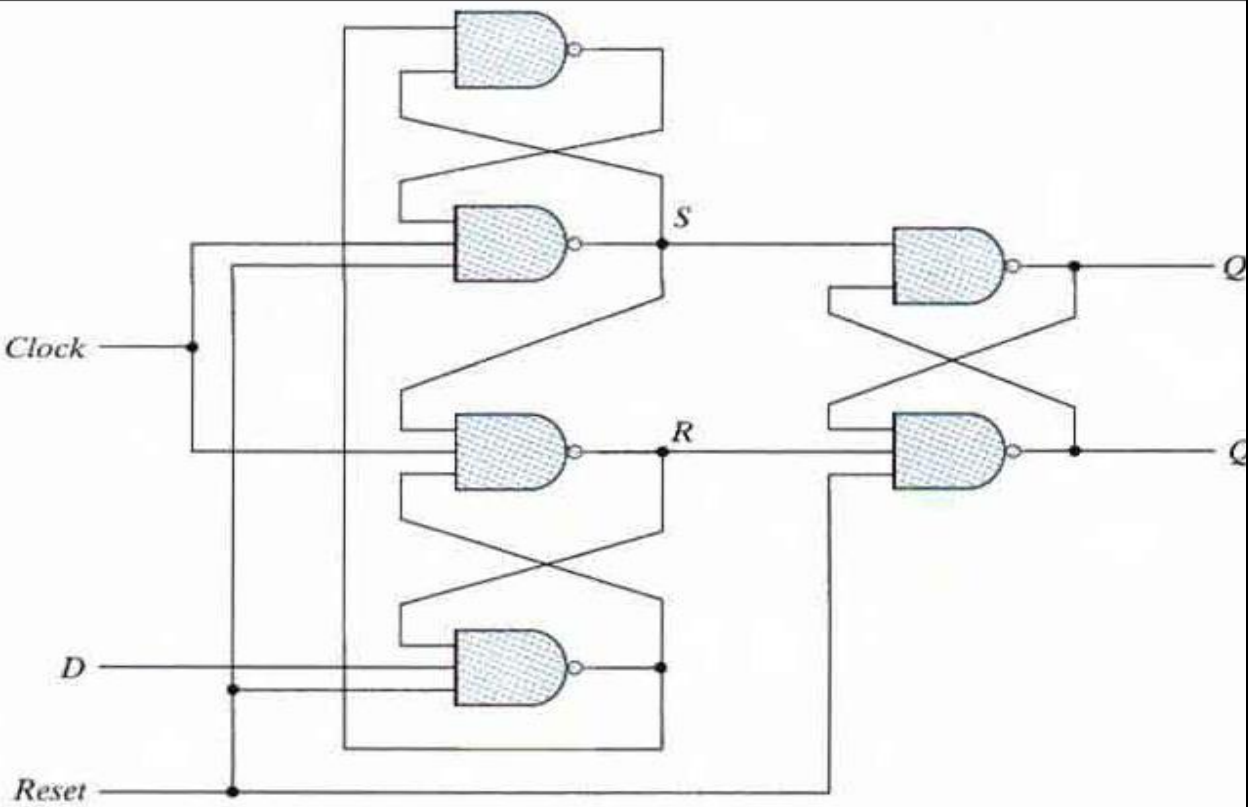
The analysis of a clocked sequential circuit consists of obtaining a table of a diagram of the time sequences of inputs, outputs and states.

- 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

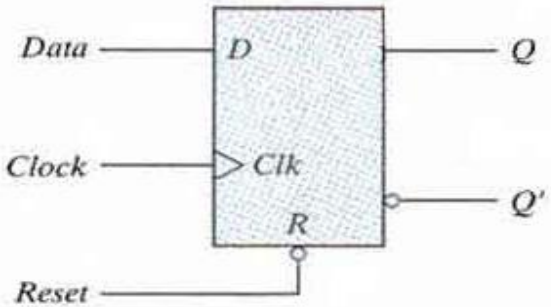
Positive Edge Triggered D Flipflop:

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



(a) Circuit diagram



(b) Graphic symbol

R	Clk	D	Q	Q'
0	X	X	0	1
0	↑	0	0	1
0	↑	1	1	0

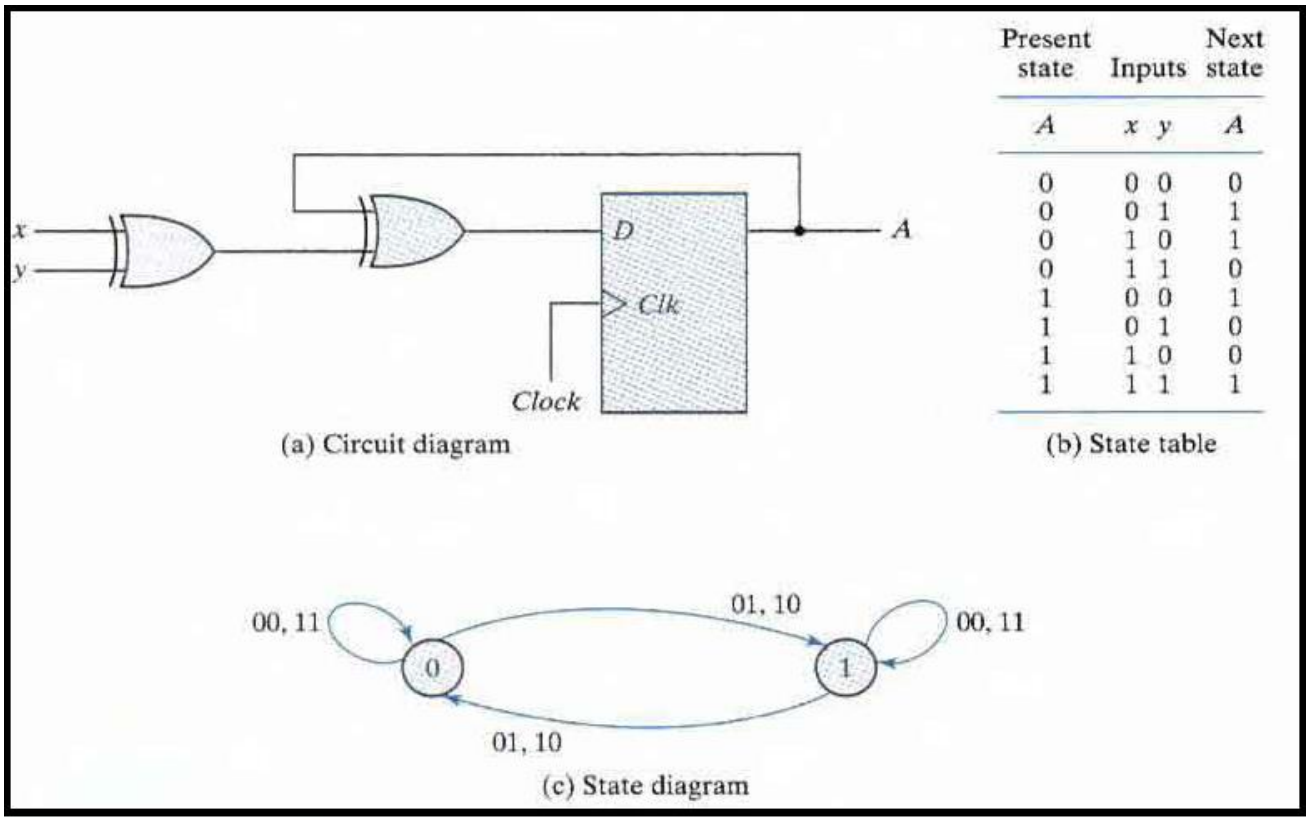
(b) Function table

D flip-flop with asynchronous reset

- 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 Flipflop:

- 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.



Present state	Inputs		Next state
A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) Circuit diagram

(b) State table

(c) State diagram

Analysis of JK Flipflop:

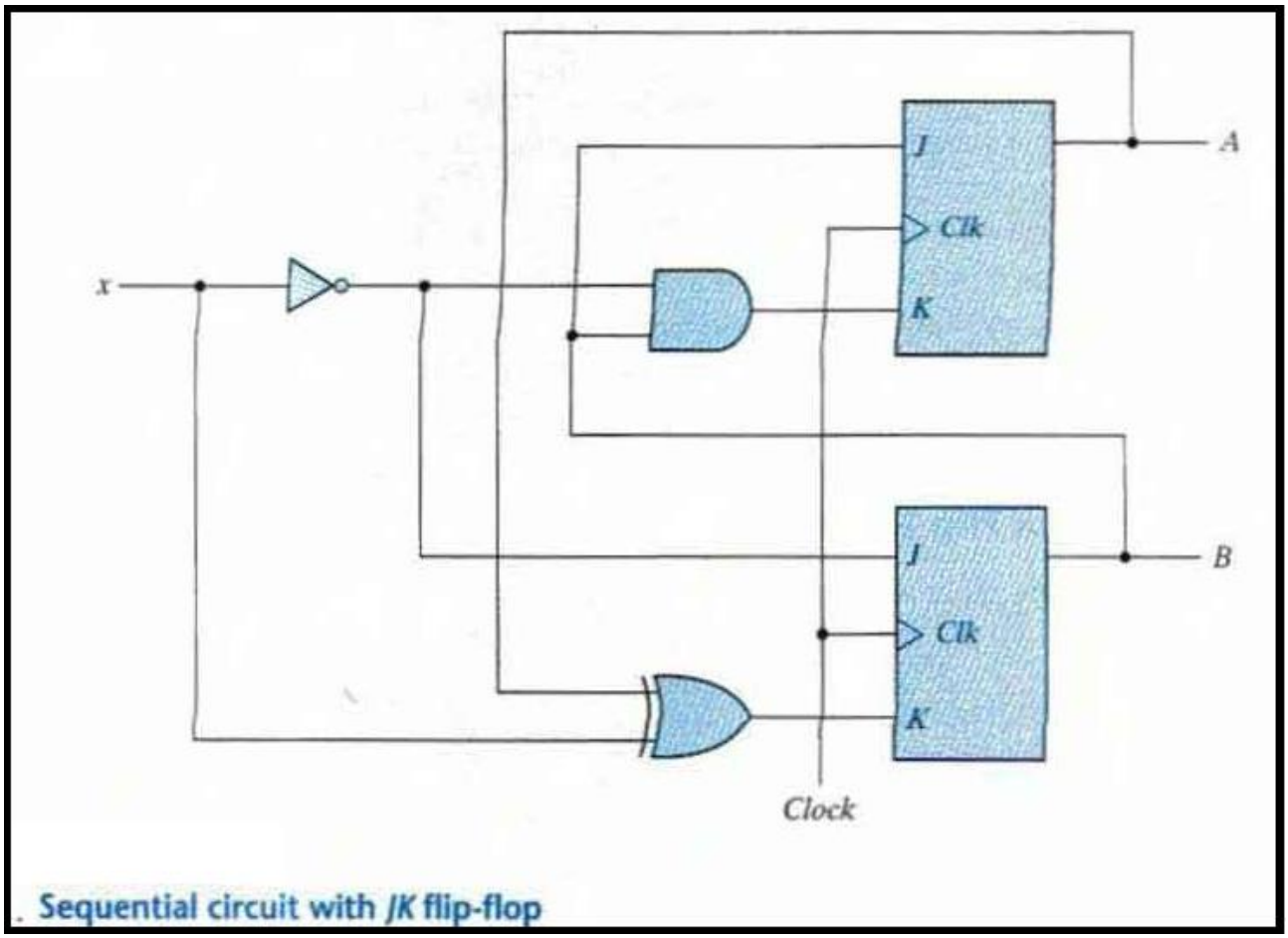
The circuit can be specified by the flip-flop input equations:

JA = B; KA = Bx'

JB = x'; KB = A'x + Ax' = A $\oplus$ 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.



Sequential circuit with *JK* flip-flop

The characteristic equations for the flip-flops are

A(t + 1) = JA' + K'A

B(t + 1) = JB' + K'B

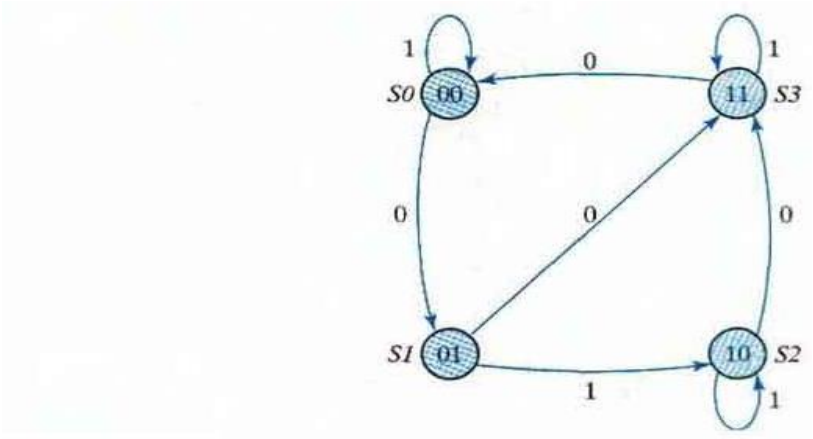
State Table for Sequential Circuit with JK Flip-Flops

Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

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'



State diagram of the circuit

Analysis with T Flipflops:

The circuit can be specified by the characteristic equations:

$Q(t+1) = T \oplus Q = T'Q + TQ'$

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:

$TA = Bx$

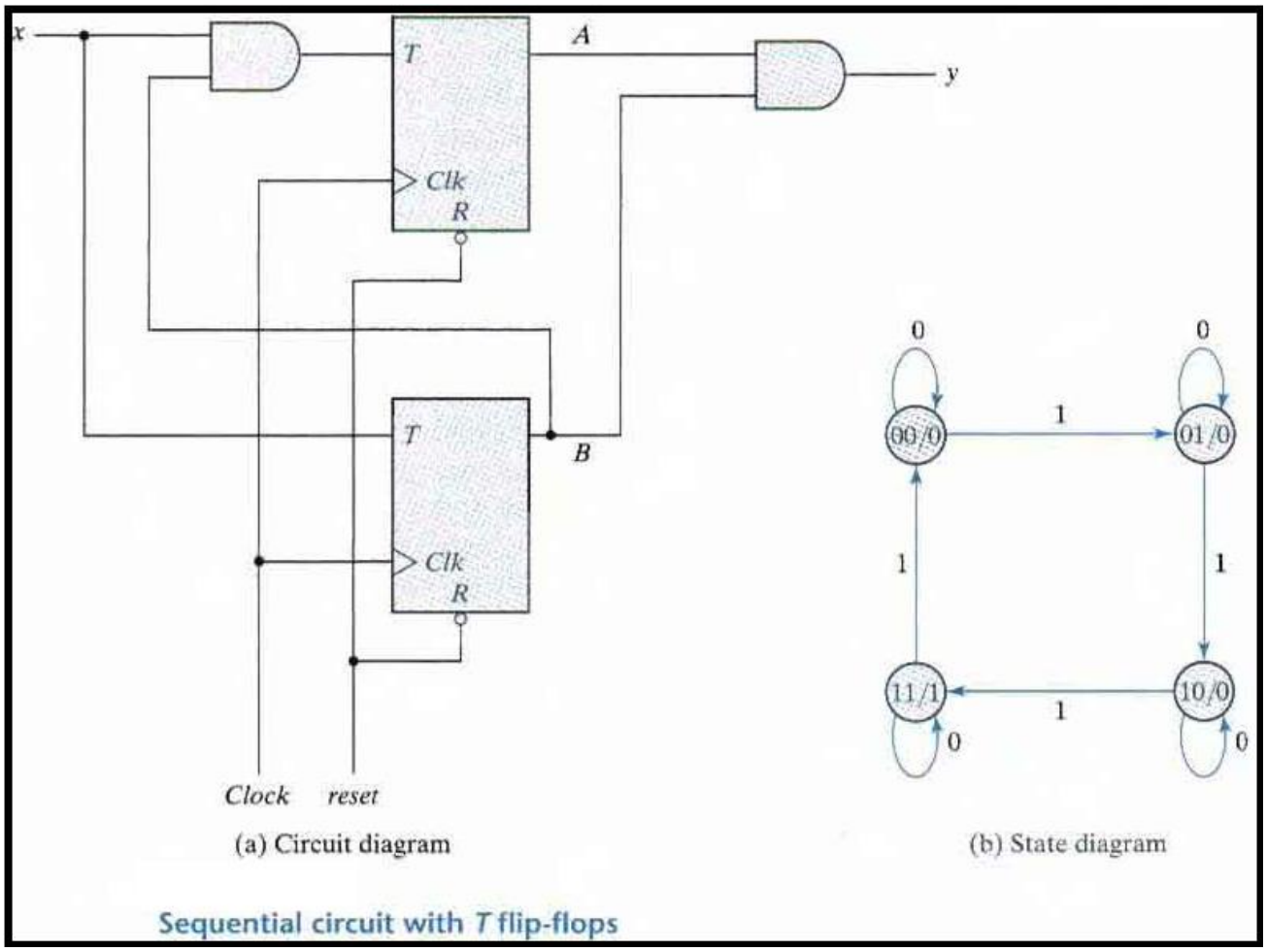
$TB = x$

$y = AB$

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:

$A(t + 1) = (Bx)' A + (Bx)A' = AB' + Ax' + A'Bx$

$$B(t + 1) = x \oplus B$$



Sequential circuit with *T* flip-flops

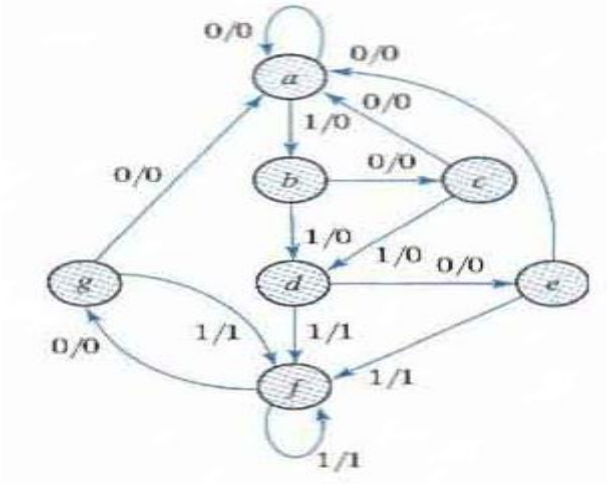
State Table for Sequential Circuit with T Flip-Flops

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

State Reduction:

- 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:



State Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

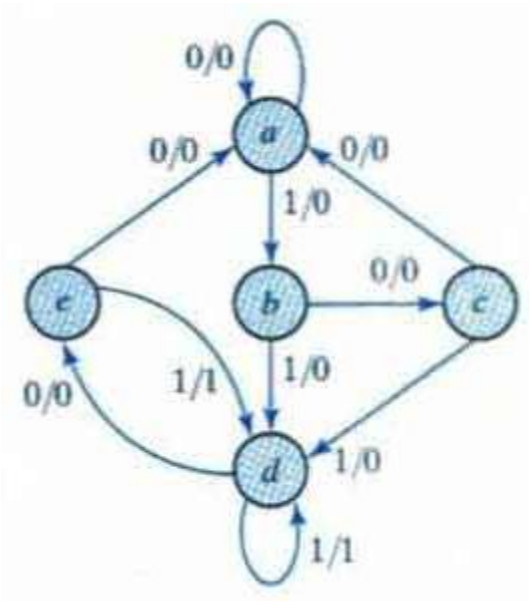
- Then we apply the reduction algorithms "Two states are said to be equivalent if for each member of the set of inputs they give exactly the same output and send the circuit either to the same state or to an equivalent state."
- When two states are equivalent one of them can be removed without altering 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

Reduced State Table

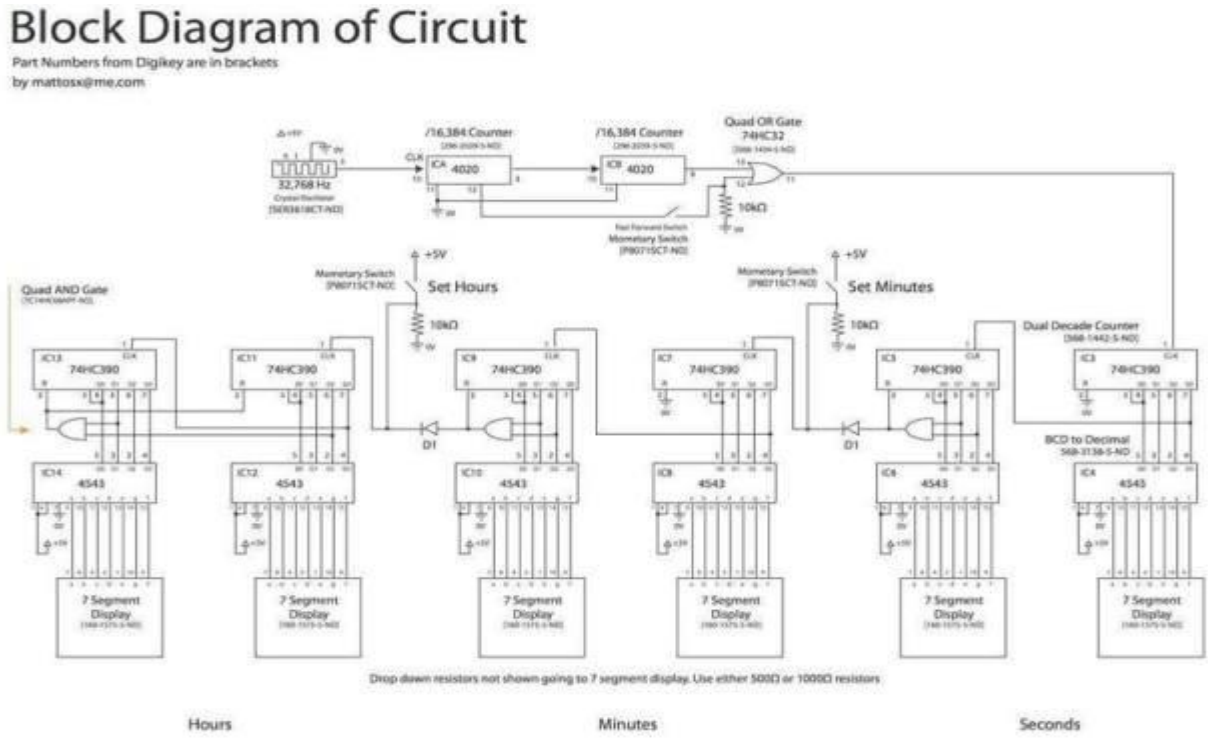
Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

In general, reducing the number of states in a state table may result in a circuit with less equipment's.

But it does not guarantee a saving in the number of flip-flops or the number of gates.



Digital clock using Logic gates:



First of all we're going to divide this tuorial into 3 parts:

– How it works in the 24h mode.

– How it works in the 12h mode.

– How to transfer between them using simple switch.

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

The circuit works as follows:

555 timer produces 1 second pulses to the clock input of the first counter which is responsible 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.

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

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

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.