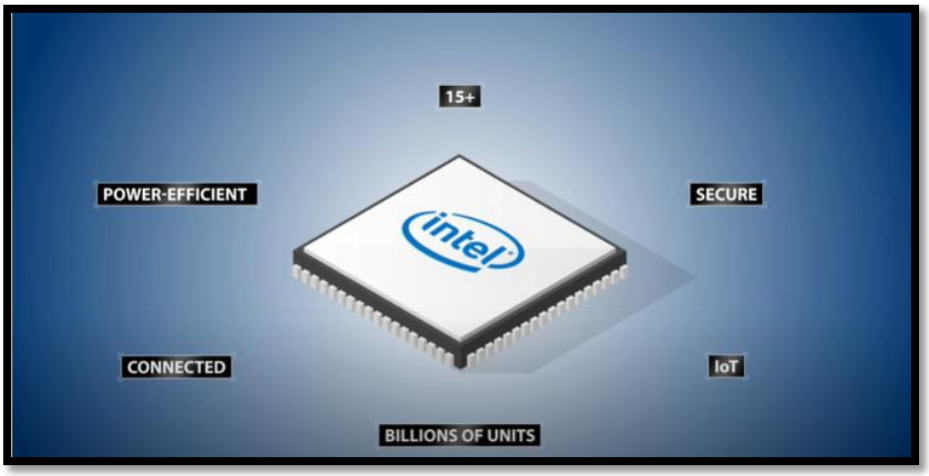
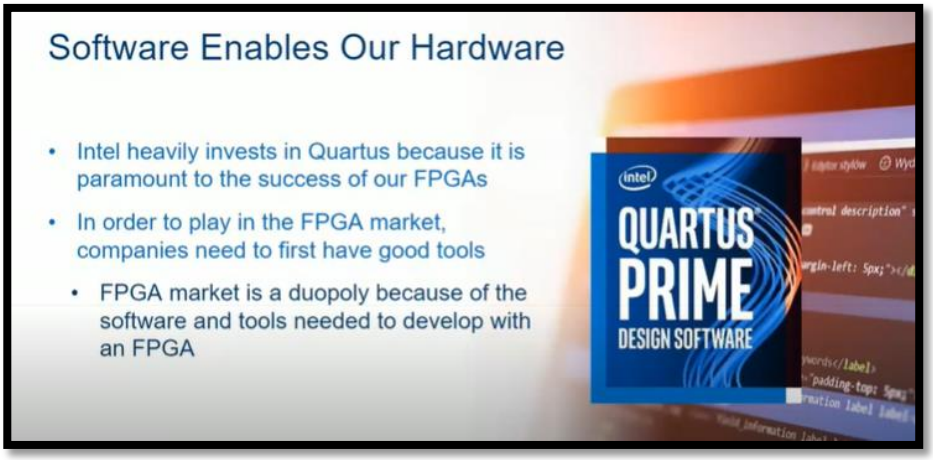
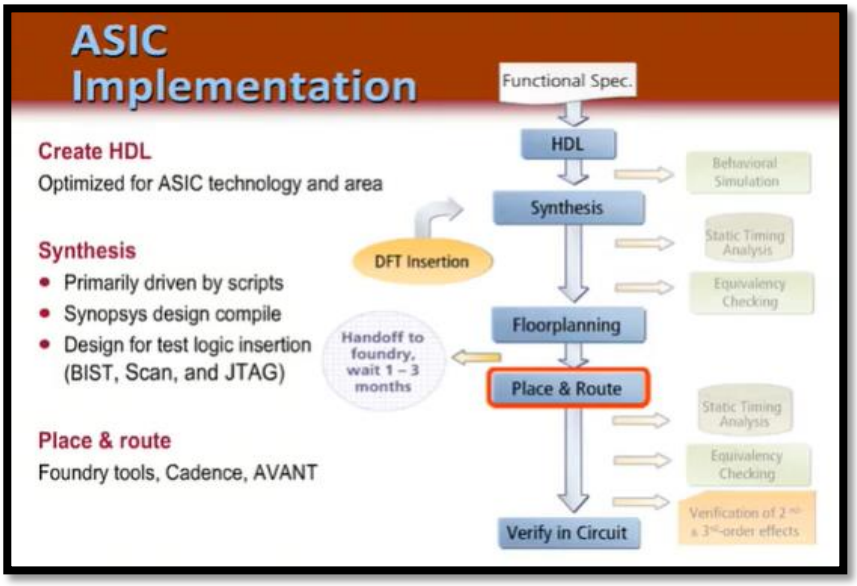
DAILY ASSESSMENT FORMAT

Date:	**1/06/2020**	**Name:**	**Davis S. Patel**
Course:	**Digital Design Using Hdl**	**USN:**	**4AL16EC045**
Topic:	**Industry Applications of FPGA FPGA Business Fundamentals FPGA vs ASIC Design Flow FPGA Basics – A Look Under the Hood**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

FORENOON SESSION DETAILS
Image of session    



Report –

An FPGA is a (mostly) digital, (re-)configurable ASIC. I say mostly because there are analog and mixed-signal aspects to modern FPGAs. For example, some have A/D converters and PLLs. I put *re-* in parenthesis because there are actually one-time-programmable FPGAs, where once you configure them, that's it, never again. However, most FPGAs you'll come across are going to be re-configurable.

There are four processing/algorithm attributes –

1. **Parallel processes** – if you need to process several input channels of information (e.g. many simultaneous A/D channels) or control several channels at once (e.g. several PID loops).

2. **High data-to-clock-rate-ratio** – if you've got lots of calculations that need to be executed over and over and over again, essentially continuously. The advantage is that you're not tying up a centralized processor. Each function can operate on its own.

3. **Large quantities of deterministic I/O** – the amount of determinism that you can achieve with an FPGA will usually far surpass that of a typical sequential processor. If there are too many operations within your required loop rate on a sequential

processor, you may not even have enough time to close the loop to update all of the I/O within the allotted time.

4. **Signal processing** – includes algorithms such as digital filtering, demodulation, detection algorithms, frequency domain processing, and image processing, or control algorithms.

With any significant benefit, there's often times a corresponding cost. In the case of FPGAs, the following are generally the main disadvantages of FPGA-based solutions.

Complex calculations infrequently – If the majority of your algorithms only need to make a computation less than 1% of the time, you've generally still allocated those logic resources for a particular function (there are exceptions to this), so they're still sitting there on your FPGA, not doing anything useful for a significant amount of time.

Sorting/searching – this really falls into the category of a sequential process. There are algorithms that attempt to reduce the number of computations involved, but in general, this is a sequential process that doesn't easily lend itself to efficient use of parallel logical resources. Check out the sorting section here and check out this article here for some more info.

Floating point arithmetic – historically, the basic arithmetic elements within an FPGA have been fixed-point binary elements at their core. In some cases, floating point math can be achieved (see Xilinx FP Operator and Altera FP White Paper), but it will chew up a lot of logical resources. Be mindful of single-precision vs double-precision, as well as deviations from standards. However, this FPGA weakness appears to be starting to fade, as hardened floating-point DSP blocks are starting to be embedded within some FPGAs (see Altera Arria 10 Hard Floating Point DSP Block).

Very low power – Some FPGAs have low power modes (hibernate and/or suspend) to help reduce current consumption, and some may require external mode control ICs to get the most out of this. Check out an example low power mode FPGA here. There are both static and dynamic aspects to power consumption. Check out these power estimation spreadsheets to start to get a sense of power utilization under various

conditions. However, if low power is critical, you can generally do better power-wise with low-power architected microprocessors or microcontrollers.

Very low cost – while FPGA costs have come down drastically over the last decade or so, they are still generally more expensive than sequential processors.

Hard cores – These are functional blocks that (at least for the most part) have their own dedicated logical resources. In other words, they are already embedded into your FPGA silicon. You configure them with various parameters and tell the tools to enable them for you. This could include functions such as high-speed communications (e.g. high-speed serial, Ethernet), low-speed A/D converters for things like measuring slowly varying voltages, and microprocessor cores to handle some of the functions that FPGA logic is not as well suited for.

Soft cores – These are functional blocks that don't have their own dedicated logical resources. In other words, they are laid out with your core logic resources. You configure them with various parameters and tell the tools to build them for you. This could include everything from DDR memory interfaces to FFT cores to FIR filters to microprocessors to CORDICs. The library of available soft cores can be impressive. On the plus side, you don't have to take as much time to develop these cores. On the negative side, since you won't know the intricacies of the design, when you plop it down and it doesn't work, it will generally take you longer to figure out why.

Task 1 -Write a Verilog code to implement NAND gate in all different styles.

Gate Level modeling

```
module NAND_2(output Y, input A, B);

wire Yd;

and(Yd, A, B);
```

```
not(Y, Yd);
endmodule;
module NAND_2_gate_level(output Y, input A, B);
wire Yd;
and(Yd, A, B);
not(Y, Yd);
endmodule
```

Data flow modeling

```
module NAND_2_data_flow (output Y, input A, B);
assign Y = ~(A & B);
module NAND_2_data_flow (output Y, input A, B);
assign Y = ~(A & B);
endmodule
```

Behavioral Modeling

```
module NAND_2_behavioral (output reg Y, input A, B);
always @ (A or B) begin
  if (A == 1'b1 & B == 1'b1) begin
     Y = 1'b0
  end
  else
```

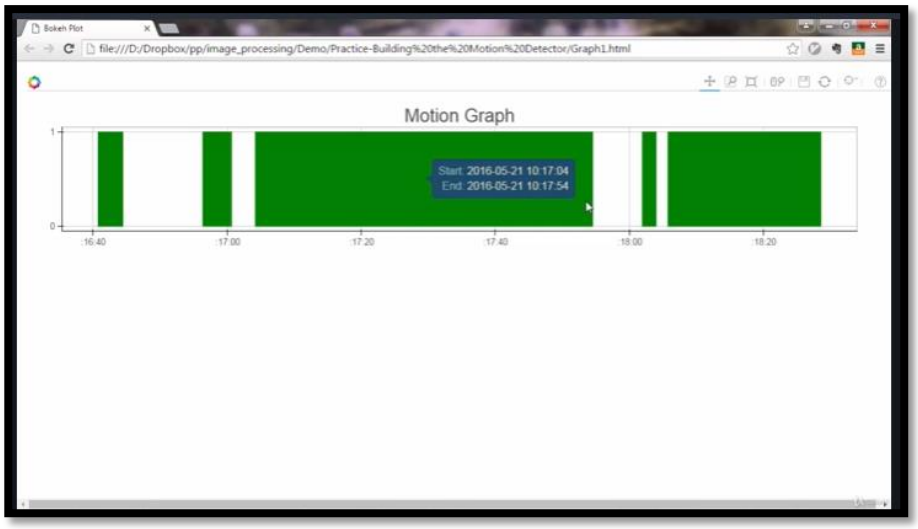
```
      Y = 1'b1;
end
module NAND_2_behavioral (output reg Y, input A, B);
always @ (A or B) begin
   if (A == 1'b1 & B == 1'b1) begin
      Y = 1'b0;
   end
   else
      Y = 1'b1;
end
endmodule
```

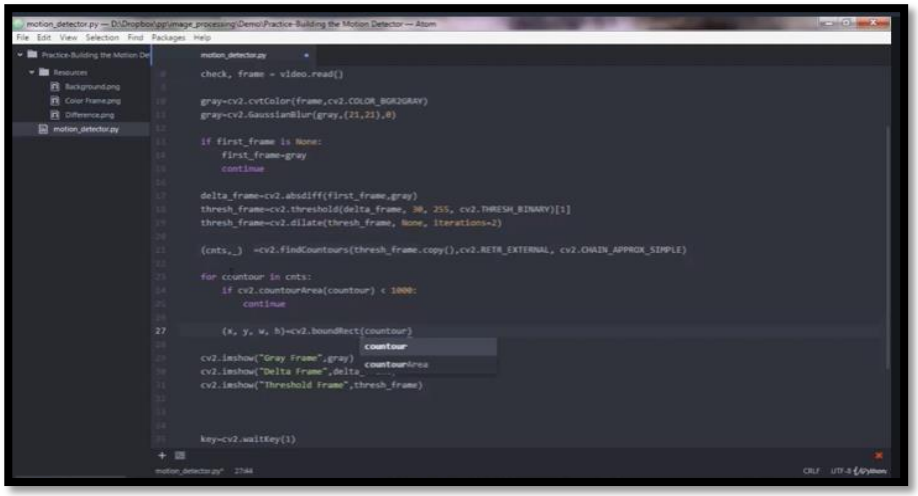
DAILY ASSESSMENT FORMAT

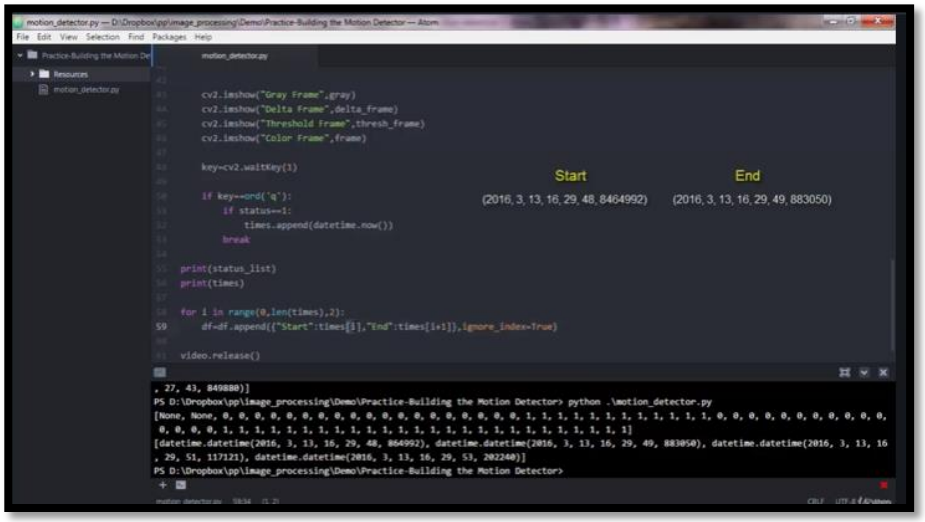
Date:	**1/06/2020**	**Name:**	**Davis S. Patel**
Course:	**Python Course**	**USN:**	**4AL16EC045**
Topic:	**Application 6: Build a Webcam Motion Detector**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Davis**		

AFTERNOON SESSION DETAILS

Image of Session







Report –

Webcam Motion Detector in Python

Python program will allow us to detect motion and also store the time interval of the motion.

- **Install Requirements**: Install Python3, install Pandas and OpenCV libraries.
- **Main Logic:** Videos can be treated as stack of pictures called frames. Here I am comparing different frames (pictures) to the first frame which should be static(No movements initially). We compare two images by comparing the intensity value of each pixels.

Analysis of all windows

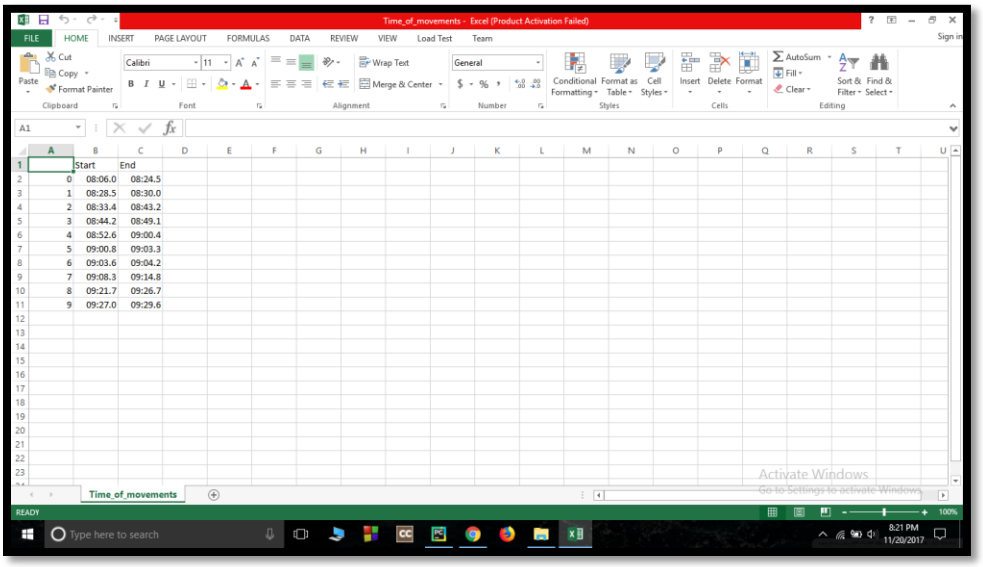
Gray Frame : In Gray frame the image is a bit blur and in grayscale we did so because, In gray pictures there is only one intensity value whereas in RGB(Red, Green and Blue) image thre are three intensity values. So it would be easy to calculate the intensity difference in grayscale.

Difference Frame: Difference frame shows the difference of intensities of first frame to the current frame.

Threshold Frame: If the intensity difference for a particular pixel is more than 30(in my case) then that pixel will be white and if the difference is less than 30 that pixel will be black

Color Frame: In this frame you can see the color images in color frame along with green contour around the moving objects.

The Time_of_movements file will be stored in the folder where your code file is stored. This file will be in csv extension. In this file the start time of motion and the end time of motion will be recorded.



Example Code

```
#Import required libraries

import cv2

import pandas as pd

import time

from datetime import datetime
```

```
#Initialise variables
stillImage = None
motionImage = [ None, None ]
time = []

# Initializing the DataFrame with start and end time
df = pd.DataFrame(columns = ["start", "end"])

# Capturing video
video = cv2.VideoCapture(0)

while True:
  # Start reading image from video
  check, frame = video.read()
  motion = 0

  # Convert color image to gray_scale image
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  gray = cv2.GaussianBlur(gray, (21, 21), 0)
  if stillImage is None:
    stillImage = gray
    continue
```

```
# Still Image and current image.
diff_frame = cv2.absdiff(stillImage, gray)

# change the image to white if static background and current frame is greater than 25.
thresh_frame = cv2.threshold(diff_frame, 25, 255, cv2.THRESH_BINARY)[1]
thresh_frame = cv2.dilate(thresh_frame, None, iterations = 2)
# Finding contour and hierarchy from a moving object.
contours,hierachy = cv2.findContours(thresh_frame.copy(),
  cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
for contour in contours:
  if cv2.contourArea(contour) < 10000:
    continue
  motion = 1
  (x, y, w, h) = cv2.boundingRect(contour)
  cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 3)
# Append current status of motion
motionImage.append(motion)
motionImage = motionImage[-2:]
# Append Start time of motion
if motionImage[-1] == 1 and motionImage[-2] == 0:
  time.append(datetime.now())
```

```
  # Append End time of motion
  if motionImage[-1] == 0 and motionImage[-2] == 1:
    time.append(datetime.now())
  # Displaying image in gray_scale
  cv2.imshow("Gray_Frame", gray)

  # Display black and white frame & if the intensity difference is > 25, it turns white
  cv2.imshow("Threshold Frame", thresh_frame)
  # Display colored frame
  cv2.imshow("Colored_Frame", frame)

  key = cv2.waitKey(1)
  # Press q to stop the process
  if key == ord('q'):
    if motion == 1:
      time.append(datetime.now())
    break

# Append time of motion
for i in range(0, len(time), 2):
  df = df.append({"Start":time[i], "End":time[i + 1]}, ignore_index = True)
```

```
# Creating a csv file in which time of movements will be saved

df.to_csv("FrameInMotion_time.csv")


video.release()

# close window

cv2.destroyAllWindows()
```