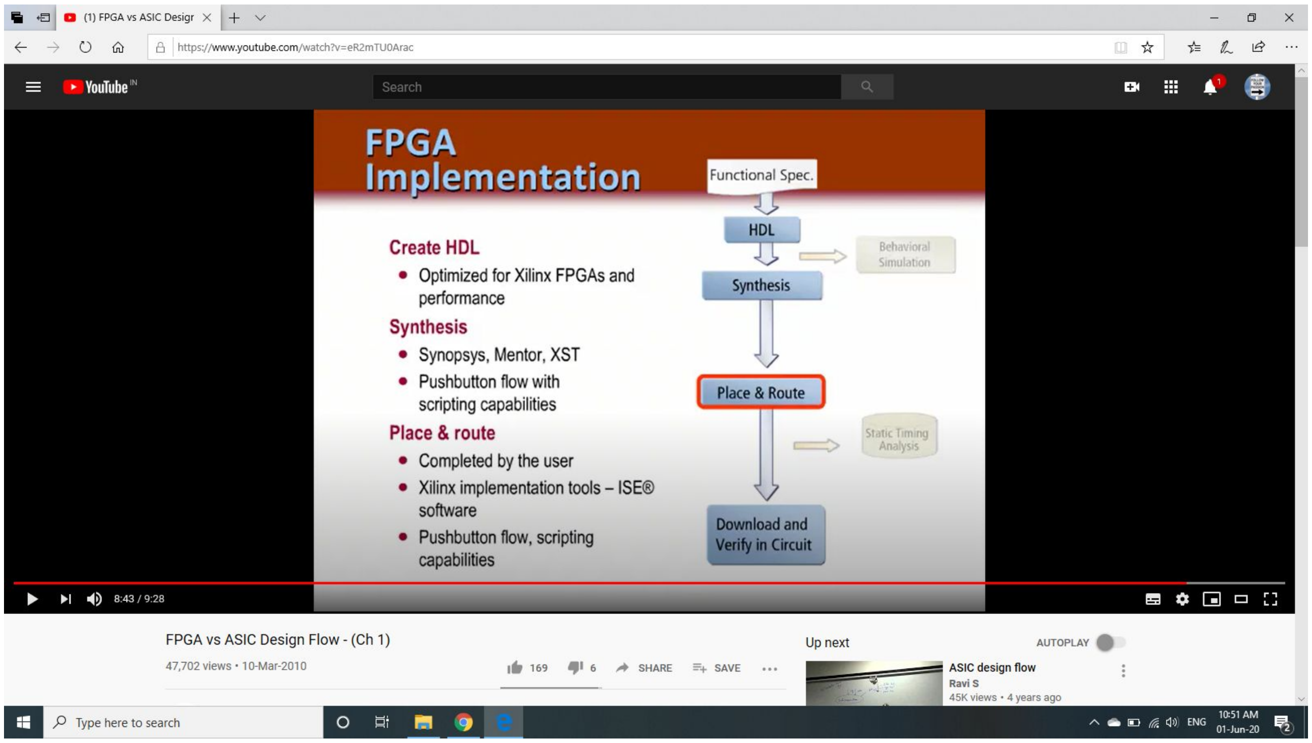
REPORT JUNE 01

Date:	**01 JUNE 2020**	**Name:**	**Rakshith B**
Course:	**Digital Design Using HDL**	**USN:**	**4AL16EC409**
Topic:	**FPGA Basic Fundamentals**	**Semester & Section:**	**6th SEM B**
Github Repository:	**Rakshith-B**		

FORENOON SESSION DETAILS

Image of session



FPGA Implementation

Create HDL
- Optimized for Xilinx FPGAs and performance

Synthesis
- Synopsys, Mentor, XST
- Pushbutton flow with scripting capabilities

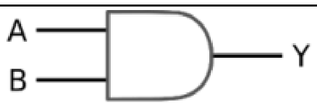
Place & route
- Completed by the user
- Xilinx implementation tools – ISE® software
- Pushbutton flow, scripting capabilities

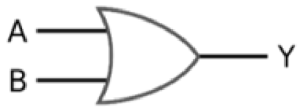
Report –

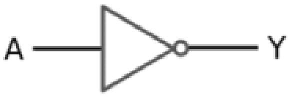
What is an FPGA?

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. So what do I mean by digitally configurable ASIC?

I mean that at the core of it, you're designing a digital logic circuit, as in AND, OR, NOT, flip-flops, etc. Of course that's not entirely accurate and there's much more to it than that, but that is the gist at its core.







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, image processing, or control algorithms.**
5. **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.**
6. **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.**
7. **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).**
8. **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.**
9. **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.**

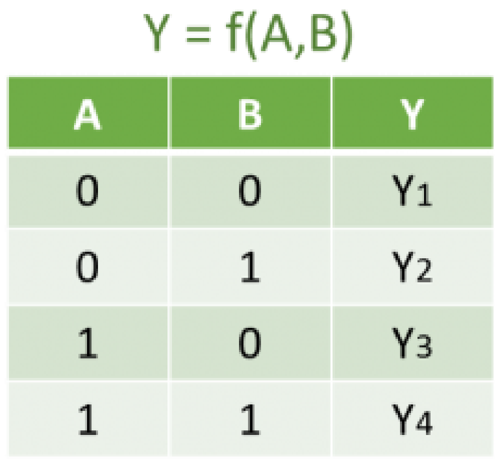
How Does an FPGA work?

You're designing a digital circuit more than anything else, basically at one layer of abstraction above the logic gate (AND, OR, NOT) level. At the most basic level, you need to think about how you're specifying the layout and equations at the level of LUTs (Look-Up Tables) and FFs (Flip-Flops).

What's Inside – Advanced components

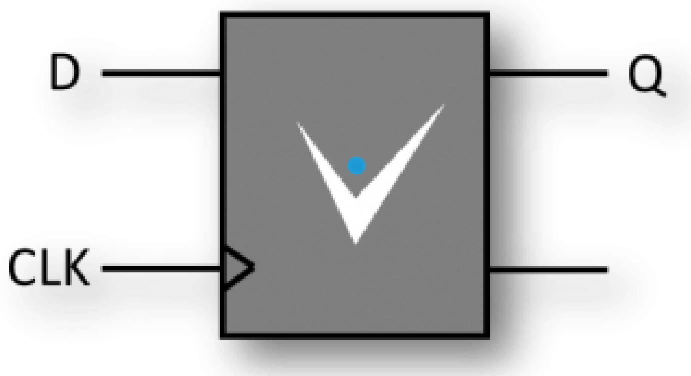
Hard cores – These are functional blocks that (at least for the most part) have their own dedicated logical resources. In other words, they are already embedded into your FPGA silicon. You configure them with various parameters and tell the tools to enable them for you. This could include functions such 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.



Y = f(A,B)

A	B	Y
0	0	Y_1
0	1	Y_2
1	0	Y_3
1	1	Y_4



Otherwise you're circuit can get very large and slow very quickly. You've got a very detailed level of control at your fingertips, which is very powerful, but can be overwhelming, so start slow. You'll be determining the # of bits, and exact math / structure of each function.An FPGA is a synchronous device, meaning that logical operations are performed on a clock cycle-by-cycle basis. Flip-flops are the core element to enabling this structure.

In general, you're going to put digital data into an FPGA and get digital data out of it through various low-voltage digital I/O lines, sometimes many bits in parallel (maybe through one or more

A/D converter outputs or an external DRAM chip), sometimes through high-speed serial I/O (maybe connecting to an Ethernet PHY or USB chip).

Write a verilog code to implement NAND gate in all different styles.

Data Flow:

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

Behavioral Modeling:

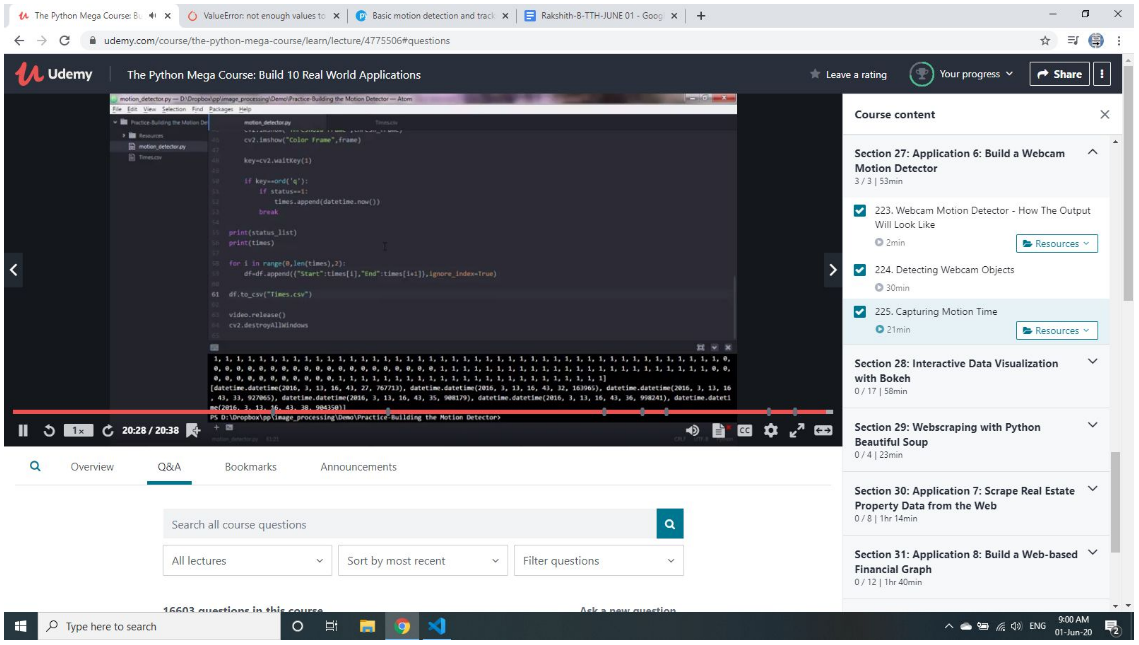
```
module NAND_2_behavioral (output reg Y, input A, B);
always @ (A or B) begin
    if (A == 1'b1 & B == 1'b1) begin
        Y = 1'b0;
    end
    else
        Y = 1'b1;
end
endmodule
```

Date: 01 JUNE 2020
Course: Python On Udemy
Topic: Built a Webcam Motion Detector, Data Visualization with Bokeh, Webscraping with Python

Name:RAKSHITH B
USN:4AL16EC409
Semester & Section:6 B

AFTERNOON SESSION DETAILS

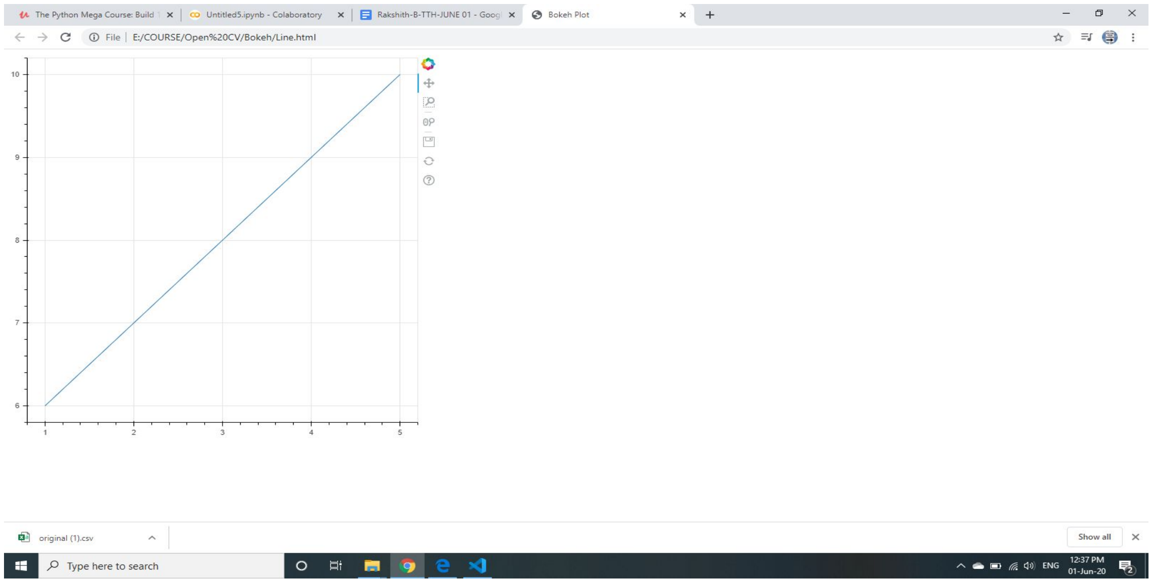
Image of session:Output


The Python Mega Course: Build 10 Real World Applications
Leave a rating
Your progress
Share
Course content
Section 27: Application 6: Build a Webcam Motion Detector
3 / 3 | 53min
223. Webcam Motion Detector - How The Output Will Look Like
2min
224. Detecting Webcam Objects
30min
225. Capturing Motion Time
21min
Section 28: Interactive Data Visualization with Bokeh
0 / 17 | 58min
Section 29: Webscraping with Python Beautiful Soup
0 / 4 | 23min
Section 30: Application 7: Scrape Real Estate Property Data from the Web
0 / 8 | 1hr 14min
Section 31: Application 8: Build a Web-based Financial Graph
0 / 12 | 1hr 40min
Overview
Q&A
Bookmarks
Announcements
Search all course questions
All lectures
Sort by most recent
Filter questions
20:28 / 20:38

```
    cv2.imshow("Color Frame",frame)

    key=cv2.waitKey(1)

    if key==ord('q'):
        if status==1:
            times.append(datetime.now())
        break

print(status_list)
print(times)

for i in range(0,len(times),2):
    df=df.append({"Start":times[i],"End":times[i+1]},ignore_index=True)

df.to_csv("Times.csv")

video.release()
cv2.destroyAllWindows
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[datetime.datetime(2016, 3, 13, 16, 43, 27, 767713), datetime.datetime(2016, 3, 13, 16, 43, 32, 163965), datetime.datetime(2016, 3, 13, 16
, 43, 33, 927065), datetime.datetime(2016, 3, 13, 16, 43, 35, 908179), datetime.datetime(2016, 3, 13, 16, 43, 36, 998241), datetime.dateti
me(2016, 3, 13, 16, 43, 38, 904350)]
PS D:\Dropbox\pp\image_processing\Demo\Practice-Building the Motion Detector>
```

Data Visualisation:



```
import cv2
import time
import pandas
from datetime import datetime

first_frame=None
status_list=[None,None]
times=[]
df=pandas.DataFrame(columns=["Start","End"])

video=cv2.VideoCapture(0)

while True:
    check, frame = video.read()
    status=0
    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    gray=cv2.GaussianBlur(gray,(21,21),0)

    if first_frame is None:
        first_frame=gray
        continue

    delta_frame=cv2.absdiff(first_frame,gray)
    thresh_frame=cv2.threshold(delta_frame, 30, 255, cv2.THRESH_BINARY)[1]
    thresh_frame=cv2.dilate(thresh_frame, None, iterations=2)

    (_,cnts,_) =cv2.findContours(thresh_frame.copy(),cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    for contour in cnts:
        if cv2.contourArea(contour) < 10000:
            continue
        status=1

        (x, y, w, h)=cv2.boundingRect(contour)
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 3)
    status_list.append(status)

    status_list=status_list[-2:]


    if status_list[-1]==1 and status_list[-2]==0:
        times.append(datetime.now())
    if status_list[-1]==0 and status_list[-2]==1:
        times.append(datetime.now())


    cv2.imshow("Gray Frame",gray)
    cv2.imshow("Delta Frame",delta_frame)
    cv2.imshow("Threshold Frame",thresh_frame)
    cv2.imshow("Color Frame",frame)

    key=cv2.waitKey(1)

    if key==ord('q'):
        if status==1:
            times.append(datetime.now())
        break

print(status_list)
print(times)

for i in range(0,len(times),2):
    df=df.append({"Start":times[i],"End":times[i+1]},ignore_index=True)

df.to_csv("Times.csv")

video.release()
cv2.destroyAllWindows
```

Data Visualization with Bokeh:
#pip install bokeh

```
from bokeh.plotting import figure
from bokeh.io import output_file,show
import pandas
```

```
df=pandas.read_csv("data.csv")
x=df["x"]
y=df["y"]
output_file("Line.html")
f=figure()
f.line(x,y)
show(f)
```

Web Scraping:

```
pip install requests
import requests
from bs4 import BeautifulSoup
r = requests.get("http://www.pythonhow.com/example.html")
c=r.content
c
soup=BeautifulSoup(c,"html.parser")
print(soup.prettify())
all=soup.find_all("div",{"class","cities"})
all
all[0].find_all("h2")[0].text
for item in all:
  print(item.find_all("p")[0].text)
```