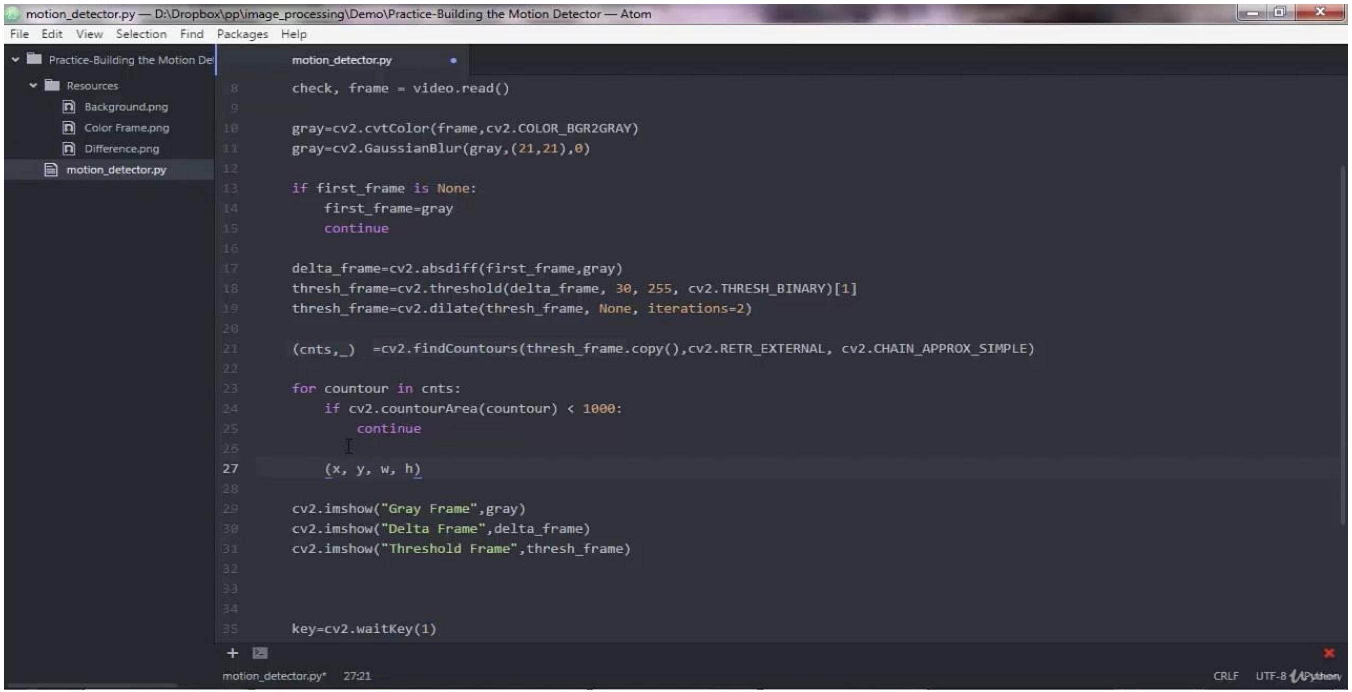
DAILY ASSESSMENT FORMAT

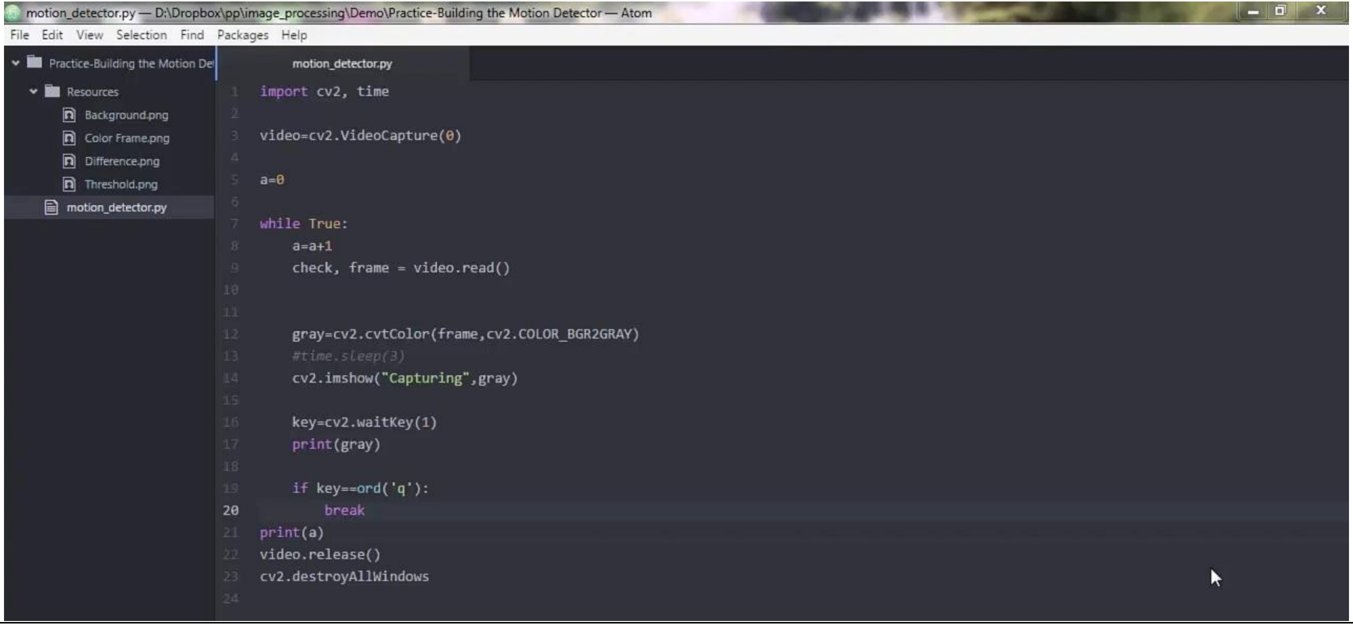
Date:	1st June 2020	**Name:**	Soundarya NA
Course:	UDEMY	**USN:**	4AL16EC077
Topic:	PYTHON: Interactive Data Visualization with Bokeh	**Semester & Section:**	8th - B

FORENOON SESSION DETAILS

Image of session


motion_detector.py — D:\Dropbox\pp\image_processing\Demo\Practice-Building the Motion Detector — Atom
File Edit View Selection Find Packages Help
Practice-Building the Motion De
Resources
Background.png
Color Frame.png
Difference.png
motion_detector.py
motion_detector.py
motion_detector.py* 27:21
CRLF UTF-8 Python

```
    check, frame = video.read()

    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.findCountours(thresh_frame.copy(),cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    for countour in cnts:
        if cv2.countourArea(countour) < 1000:
            continue

        (x, y, w, h)

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



    key=cv2.waitKey(1)
```


motion_detector.py — D:\Dropbox\pp\image_processing\Demo\Practice-Building the Motion Detector — Atom
File Edit View Selection Find Packages Help
Practice-Building the Motion De
Resources
Background.png
Color Frame.png
Difference.png
Threshold.png
motion_detector.py
motion_detector.py

```
import cv2, time

video=cv2.VideoCapture(0)

a=0

while True:
    a=a+1
    check, frame = video.read()


    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    #time.sleep(3)
    cv2.imshow("Capturing",gray)

    key=cv2.waitKey(1)
    print(gray)

    if key==ord('q'):
        break
print(a)
video.release()
cv2.destroyAllWindows
```

Report:

Data Visualization using Bokeh:

Bokeh is a data visualization library in Python that provides high-performance interactive charts and plots. Bokeh output can be obtained in various mediums like notebook, html and server. It is possible to embed bokeh plots in Django and flask apps.

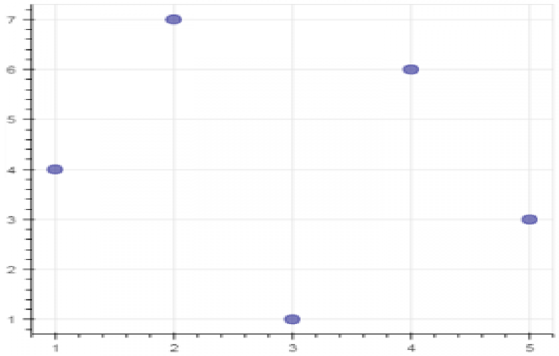
Bokeh provides two visualization interfaces to users:

bokeh.models : A low level interface that provides high flexibility to application developers.
bokeh.plotting : A high level interface for creating visual glyphs.

To install bokeh package, run the following command in the terminal:

```
# import modules
from bokeh.plotting import figure, output_notebook, show
# output to notebook
output_notebook()
# create figure
p = figure(plot_width = 400, plot_height = 400)
# add a circle renderer with
# size, color and alpha
p.circle([1, 2, 3, 4, 5], [4, 7, 1, 6, 3],
                    size = 10, color = "navy", alpha = 0.5)
# show the results
show(p)
```

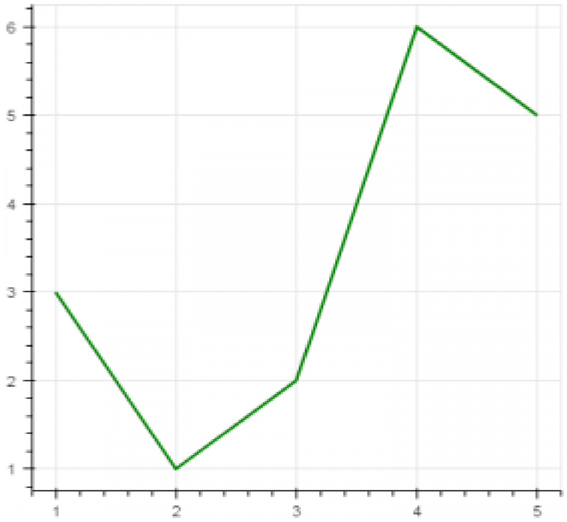
Output:



To create a single line, line() method is used.

```
# import modules

from bokeh.plotting import figure, output_notebook, show

# output to notebook

output_notebook()

# create figure

p = figure(plot_width = 400, plot_height = 400)

# add a line renderer

p.line([1, 2, 3, 4, 5], [3, 1, 2, 6, 5],

                    line_width = 2, color = "green")

# show the results

show(p)
```

Output:

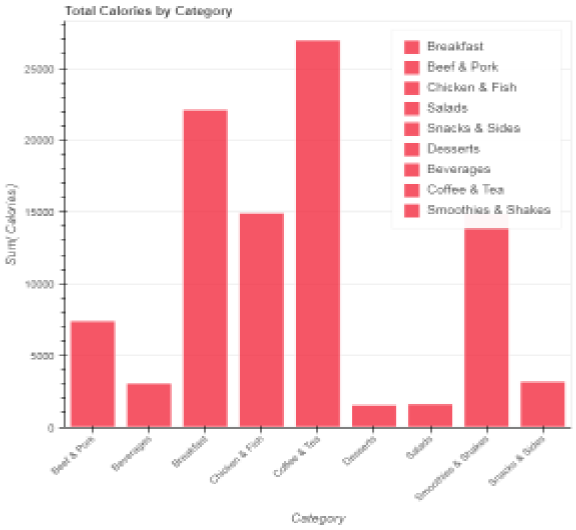


Bar chart presents categorical data with rectangular bars. The length of the bar is proportional to the values that are represented.

```
# import necessary modules

import pandas as pd

from bokeh.charts import Bar, output_notebook, show

# output to notebook

output_notebook()
```

```
# read data in dataframe

df = pd.read_csv(r"D:/kaggle/mcdonald/menu.csv")

# create bar

p = Bar(df, "Category", values = "Calories",

                   title = "Total Calories by Category",

                                                       legend = "top_right")

# show the results

show(p)
```

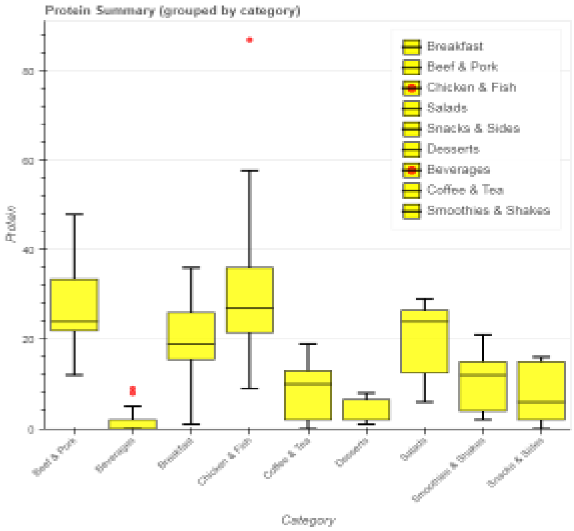
Output:



Box plot is used to represent statistical data on a plot. It helps to summarize statistical properties of various data groups present in the data.

```
# import necessary modules

from bokeh.charts import BoxPlot, output_notebook, show

import pandas as pd

# output to notebook

output_notebook()

# read data in dataframe

df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")

# create bar

p = BoxPlot(df, values = "Protein", label = "Category",
                    color = "yellow", title = "Protein Summary (grouped by category)",
                    legend = "top_right")
# show the results
show(p)
```

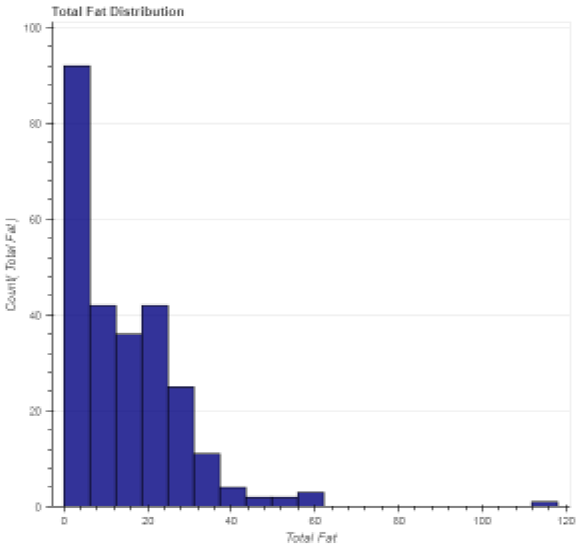
Output:



Histogram is used to represent distribution of numerical data. The height of a rectangle in a histogram is proportional to the frequency of values in a class interval.

```
# import necessary modules
from bokeh.charts import Histogram, output_notebook, show
import pandas as pd
# output to notebook
output_notebook()
# read data in dataframe
df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")
# create histogram
p = Histogram(df, values = "Total Fat",
                    title = "Total Fat Distribution",
                    color = "navy")
# show the results
show(p)
```

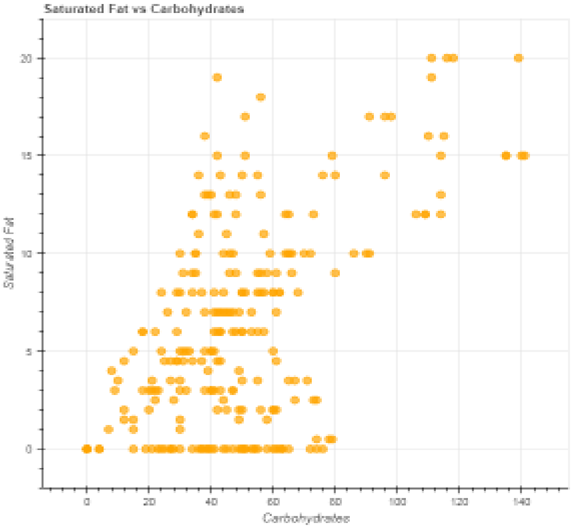
Output:



Scatter plot is used to plot values of two variables in a dataset. It helps to find correlation among the two variables that are selected.

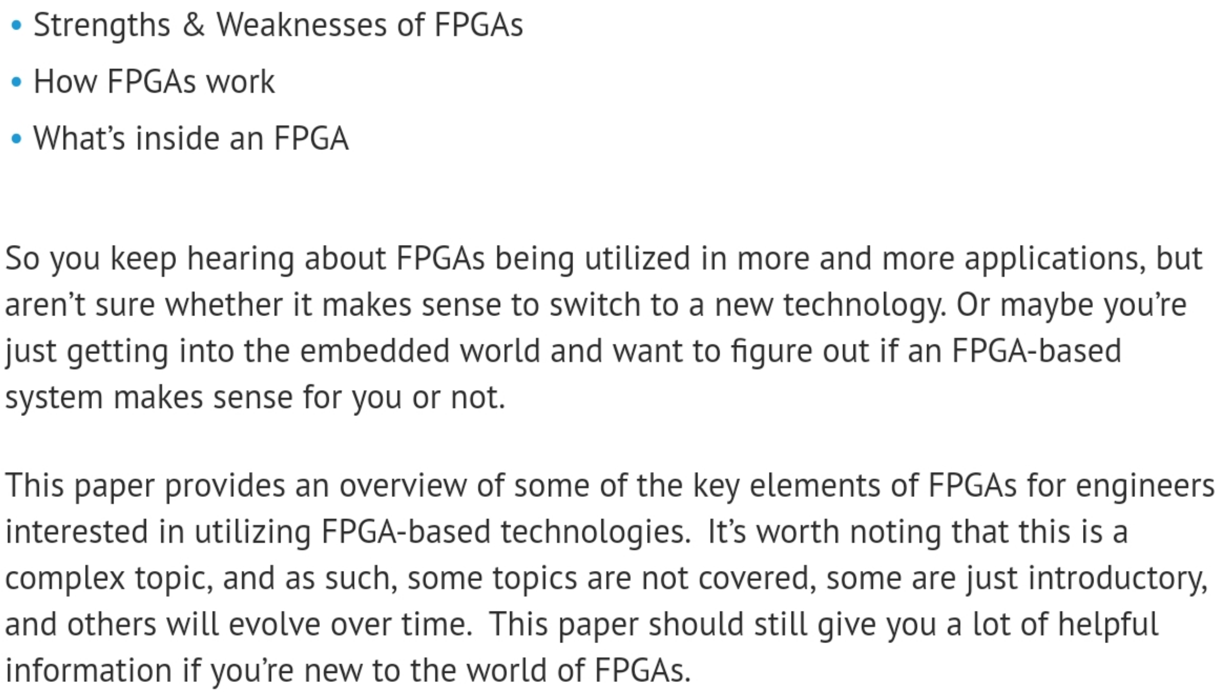
```
# import necessary modules
from bokeh.charts import Scatter, output_notebook, show
import pandas as pd
# output to notebook
output_notebook()
# read data in dataframe
df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")
# create scatter plot
p = Scatter(df, x = "Carbohydrates", y = "Saturated Fat",
                        title = "Saturated Fat vs Carbohydrates",
                        xlabel = "Carbohydrates", ylabel = "Saturated Fat",
                        color = "orange")
# show the results
show(p)
```

Output:


Saturated Fat vs Carbohydrates
Saturated Fat
Carbohydrates
0
5
10
15
20
0
20
40
60
80
100
120
140

Date:	1st June 2020	**Name:**	Soundarya NA
Course:	Digital Design using HDL	**USN:**	4AL16EC077
Topic:	FPGA and its industry applications	**Semester & Section:**	8th - B

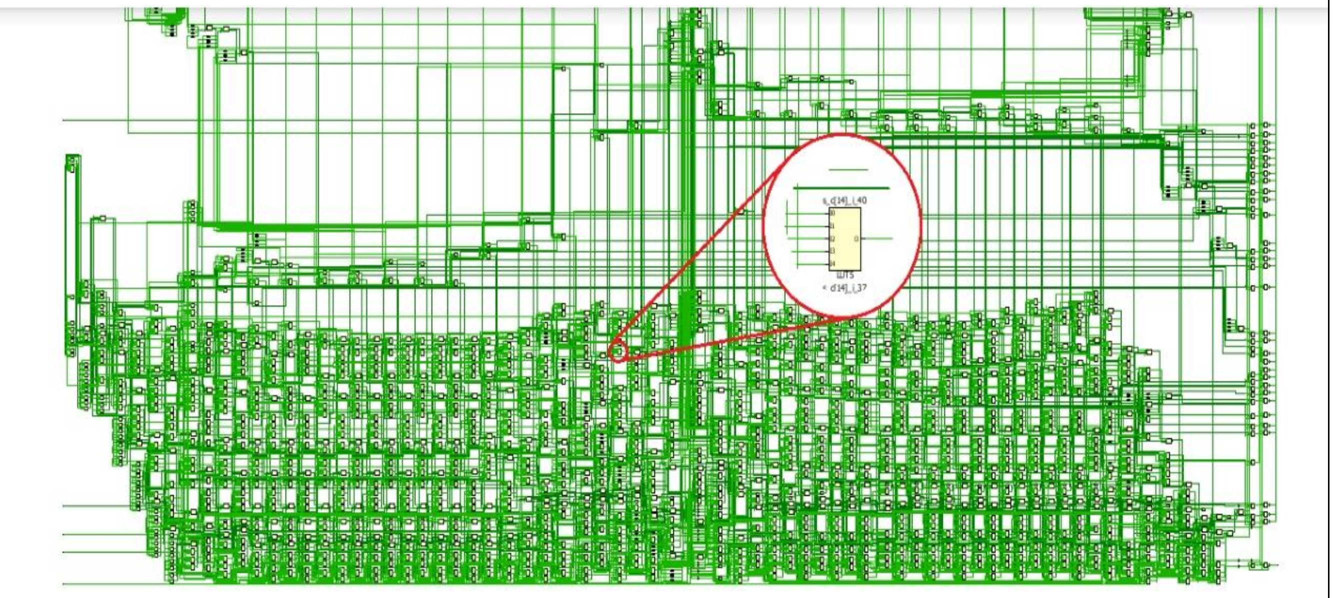
Image:



- Strengths & Weaknesses of FPGAs
- How FPGAs work
- What's inside an FPGA

So you keep hearing about FPGAs being utilized in more and more applications, but aren't sure whether it makes sense to switch to a new technology. Or maybe you're just getting into the embedded world and want to figure out if an FPGA-based system makes sense for you or not.

This paper provides an overview of some of the key elements of FPGAs for engineers interested in utilizing FPGA-based technologies. It's worth noting that this is a complex topic, and as such, some topics are not covered, some are just introductory, and others will evolve over time. This paper should still give you a lot of helpful information if you're new to the world of FPGAs.

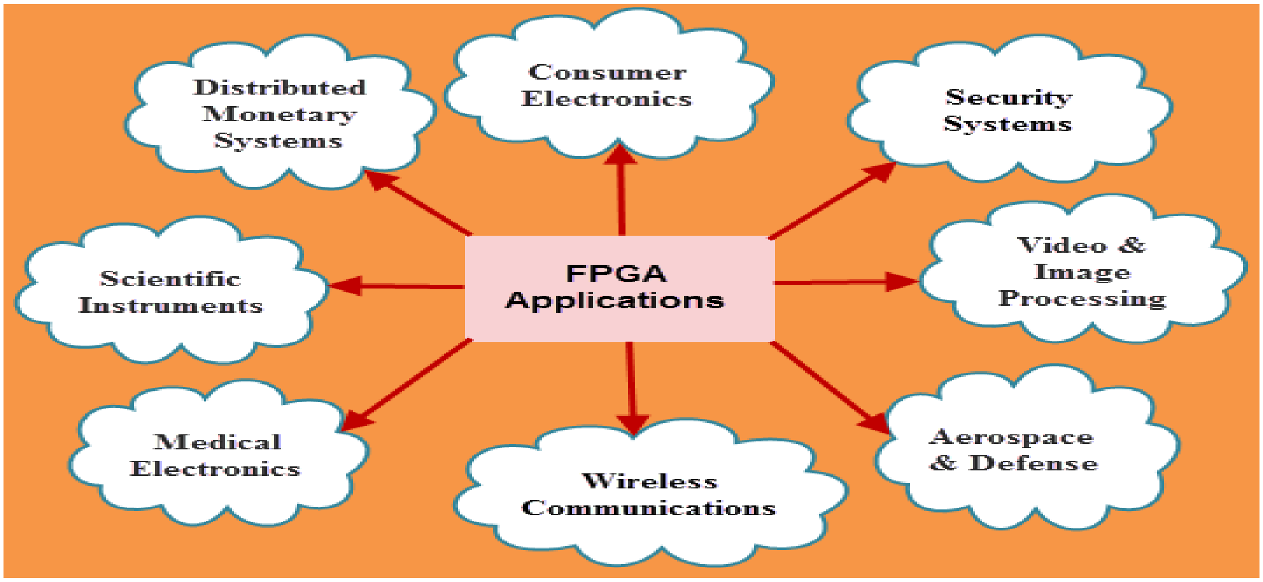


Report:

Industry applications of FPGA:

FPGAs have gained rapid growth over the past decade because they are useful for a wide range of applications. Specific application of an FPGA includes digital signal processing, bioinformatics, device controllers, software-defined radio, random logic, ASIC prototyping, medical imaging, computer hardware emulation, integrating multiple SPLDs, voice recognition, cryptography, filtering and communication encoding and many more.

Usually, FPGAs are kept for particular vertical applications where the production volume is small. For these low-volume applications, the top companies pay in hardware costs per unit. Today, the new performance dynamics and cost have extended the range of viable applications.



Applications of FPGA

Some More Common FPGA Applications are: Aerospace and Defense, Medical Electronics, ASIC Prototyping, Audio, Automotive, Broadcast, Consumer Electronics, Distributed Monetary Systems, Data Center, High Performance Computing, Industrial, Medical, Scientific Instruments, Security systems, Video & Image Processing, Wired Communications, Wireless Communications.

Types of FPGA based on application:

Low End FPGAs:

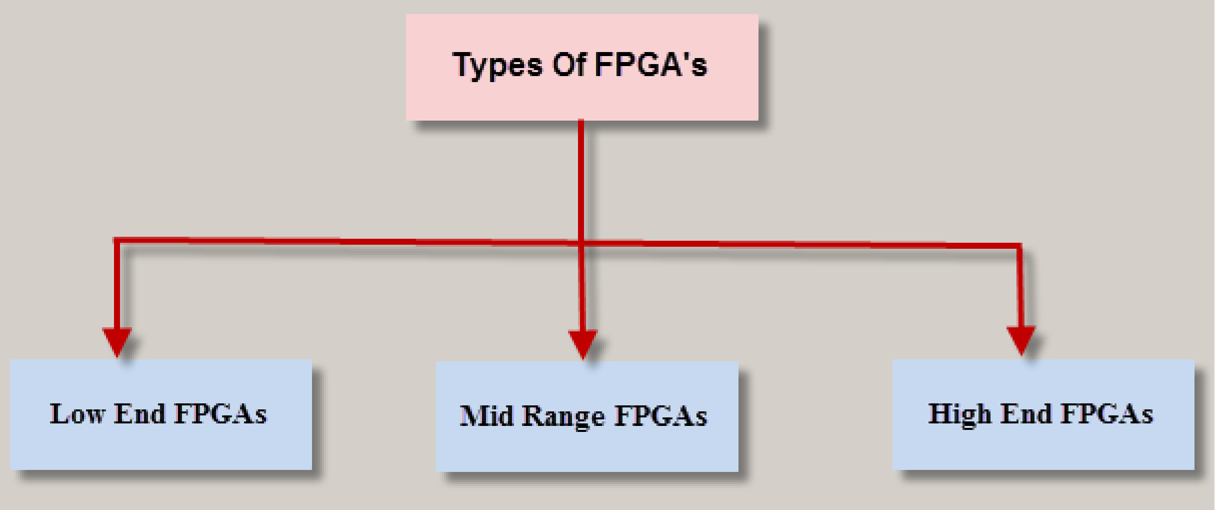
These types of FPGAs are designed for low power consumption, low logic density and low complexity per chip. Examples of low-end FPGAs are Cyclone family from Altera, Spartan family from Xilinx, fusion family from Microsemi and the Mach XO/ICE40 from Lattice semiconductor.

Mid-Range FPGAs:

These types of FPGAs are the optimum solution between the low-end and high- end FPGAs and these are developed as a balance between the performance and the cost. Examples of Mid-range FPGAs are Arria from Altera, Artix-7/Kintex-7 series from Xlinix, IGL002 from Microsemi and ECP3 and ECP5 series from Lattice semiconductor.

High End FPGAs:

These types of FPGAs are developed for logic density and high performance. Examples of High-end FPGAs are a Stratix family from Altera, Virtex family from Xilinx, Speedster 22i family from Achronix, and ProASIC3 family from Microsemi.



Types of FPGAs

FPGA vs ASIC Design Flow:

FPGA stands for Field Programmable Gate Array. It is an integrated circuit which can be "field" programmed to work as per the intended design. It means it can work as a microprocessor, or as an encryption unit, or graphics card, or even all these three at once. As implied by the name itself, the FPGA is field programmable. So, an FPGA working as a microprocessor can be reprogrammed to function as the graphics card in the field, as opposed to in the semiconductor foundries. The designs running on FPGAs are generally created using hardware description languages such as VHDL and Verilog.

FPGA is made up of thousands of Configurable Logic Blocks (CLBs) embedded in an ocean of programmable interconnects. The CLBs are primarily made of Look-Up Tables (LUTs), Multiplexers and Flip-Flops. They can implement complex logic functions. Apart from CLBs, and routing interconnects, many FPGAs also contain dedicated hard-silicon blocks for various functions such as Block RAM, DSP Blocks, External Memory Controllers, PLLs, Multi-Gigabit Transceivers etc. A recent trend is providing a hard-silicon processor core (such as ARM Cortex A9 in case of Xilinx Zynq) inside the same FPGA die itself so that the processor can take care of mundane, non-critical tasks whereas FPGA can take care of high-speed acceleration which cannot be done using processors. These dedicated hardware blocks are critical in competing with ASICs.

ASIC stands for Application Specific Integrated Circuit. As the name implies, ASICs are application specific. They are designed for one sole purpose and they function the same their whole operating life. For example, the CPU inside your phone is an ASIC. It is meant to function as a CPU for its whole life. Its logic function cannot be changed to anything else because its digital circuitry is made up of permanently connected gates and flip-flops in silicon. The logic function of ASIC is specified in a similar way as in the case of FPGAs, using hardware description languages such as Verilog or VHDL. The difference in case of ASIC is that the resultant circuit is permanently drawn into silicon whereas in FPGAs the circuit is made by connecting a number of configurable blocks. For a comparison, think of creating a castle using Lego blocks versus creating a castle using concrete. The former is analogous to FPGAs, whereas the latter is analogous to ASICs. You can reuse Lego blocks to create a different design, but the concrete castle is permanent.

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

Gate level modelling:

```
module NAND_2(output Y, input A, B);
wire Yd;
and(Yd, A, B);
not(Y, Yd);
```

```
endmodule;
```

Data flow modelling:

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

Behavioral modelling:

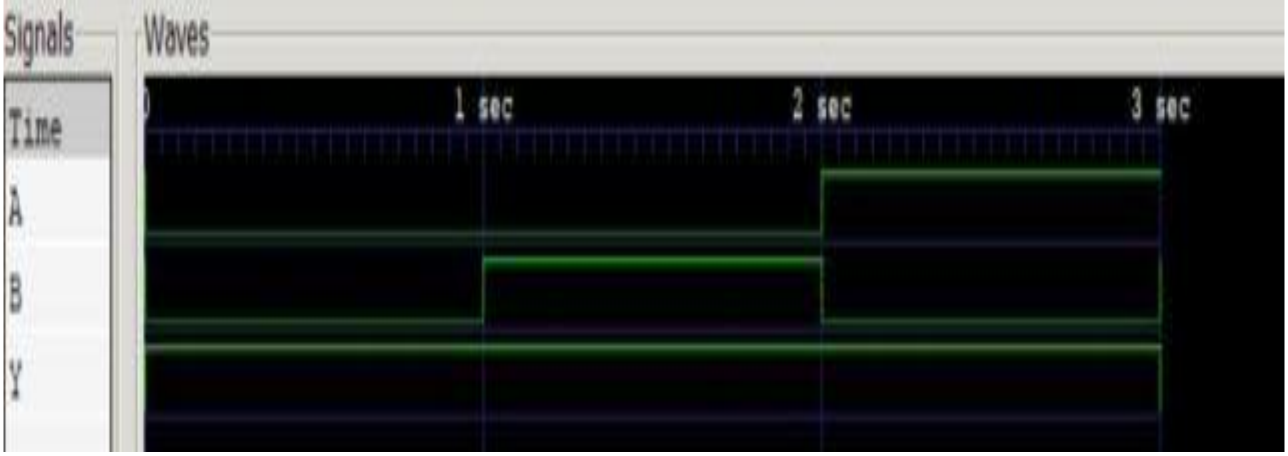
```
module NAND_2_behavioral (output reg Y, input A, B);
always @ (A or B) begin
   if (A == 1'b1 & B == 1'b1) begin
      Y = 1'b0;
   end
   else
      Y = 1'b1;
end
endmodule
```

Test bench of the NAND gate using Verilog:

```
module NAND_2_behavioral_tb;
reg A, B;
wire Y;
NAND_2_behavioral Indtance0 (Y, A, B);
initial begin
   A = 0; B = 0;
 #1 A = 0; B = 1;
 #1 A = 1; B = 0;
 #1 A = 1; B = 1;
end
initial begin
```

```
   $monitor ("%t | A = %d| B = %d| Y = %d", $time, A, B, Y);
   $dumpfile("dump.vcd");
   $dumpvars();
end
endmodule
```

Simulation waveform:


Signals
Waves
Time
A
B
Y
0
1 sec
2 sec
3 sec