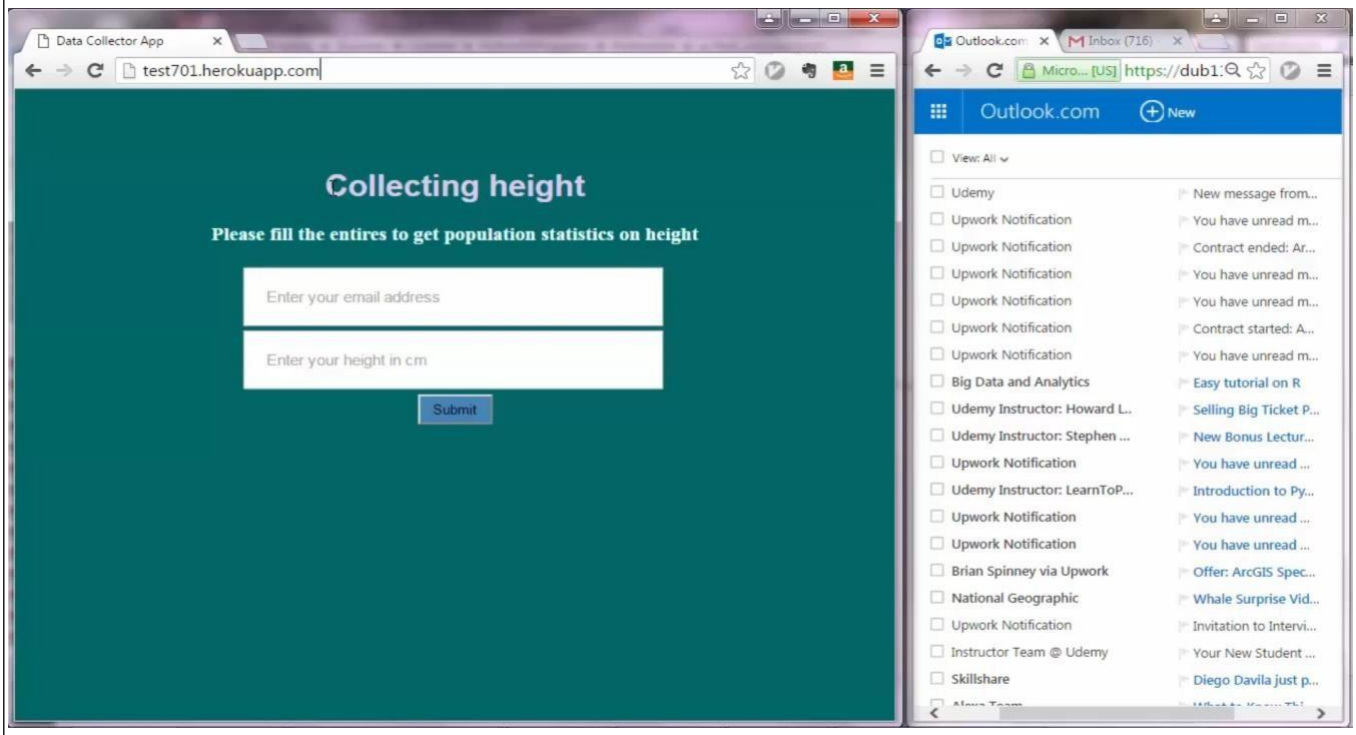
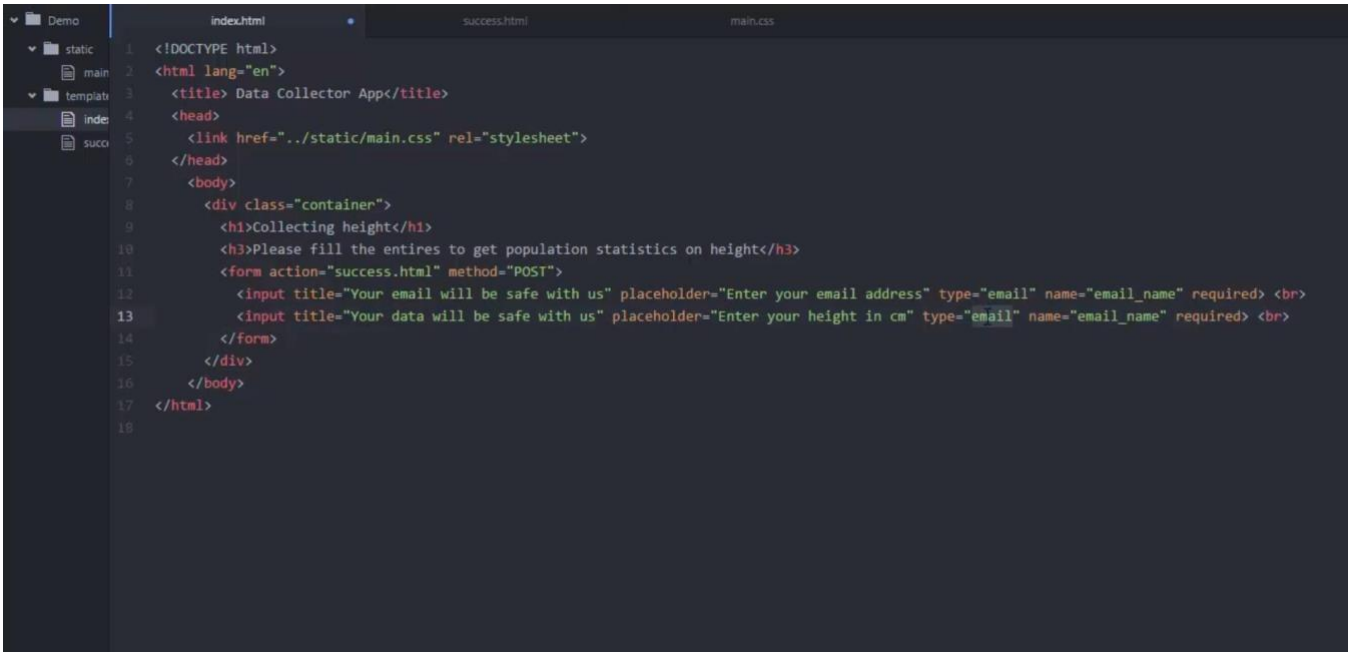
DAILY ASSESSMENT FORMAT

CERTIFICATIONSESSION DETAILS				
Date:	**29/6/2020**	**Name:**	**SAMRUDDI**	
Course:	**Python**	**USN:**	**4AL16IS047**	
Topic:	**DAY 9**	**Semester & Section:**	**8th sem**	
Github Repository:				

In this section that we're going to dive into the world of web applications with Python.The website that you're looking at is a web application built with Python so this is what you're going to build in this section. Now this was built with Python and Flask and also we're using a PostgreSQL database in here to store data and then we use Heroku to deploy the application online, and it can be accessed through this URL.It's simple but you can modify it and do whatever you want with that. In my case what I did is you know I'm asking the users or visitors to to enter their email address and their height and what the application will do is one it won't want to get this pair of data from the user, it sends the data to the database to the server and then it calculates the average height of all the users that have been submitting data so far and then it sends the average to the user that just submitted their address, so it sends the average height by email to the user so all this is automatic. So you may want to pass your email address there and your height. You know this also has some restrictions you can not pass a height of more than let's say 3 meters and you press submit. Once that is successful the user will be sent to a success page so they get the confirmation that their submission was successful and then the user will get an email to their email address, email account so I send my height using my host mail address and here is the email that I got. And I'm using my gmail address to send emails to the users and I used my of hotmail address as a visitor here you get these emails automatically, so the height that I entered there was 1.75 and this is a average height. Some sort of confirmation email.It also gives you the number of people that have been registered this far.The application that you're going to build store data in a database from your web application and send data or via email to users and so on.



Building the Front End HTML Part

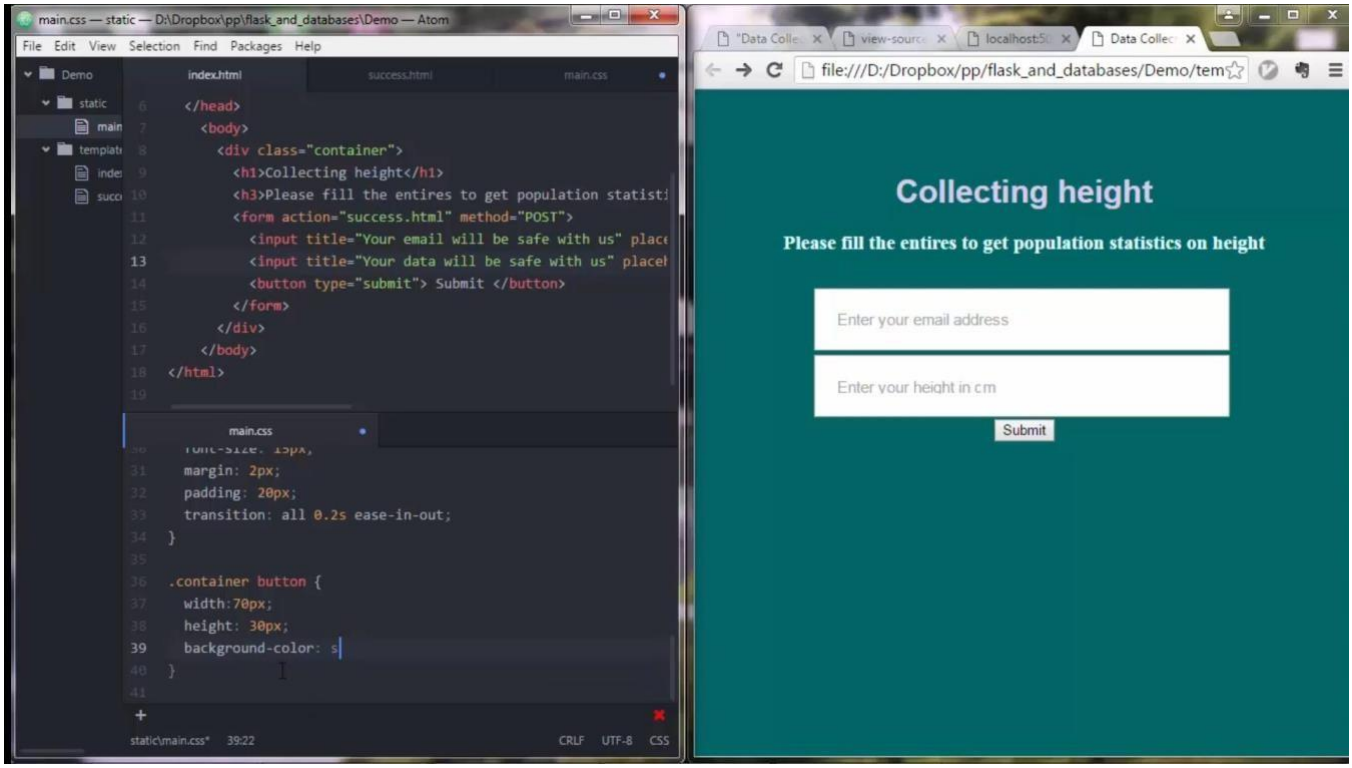


<!DOCTYPE html>

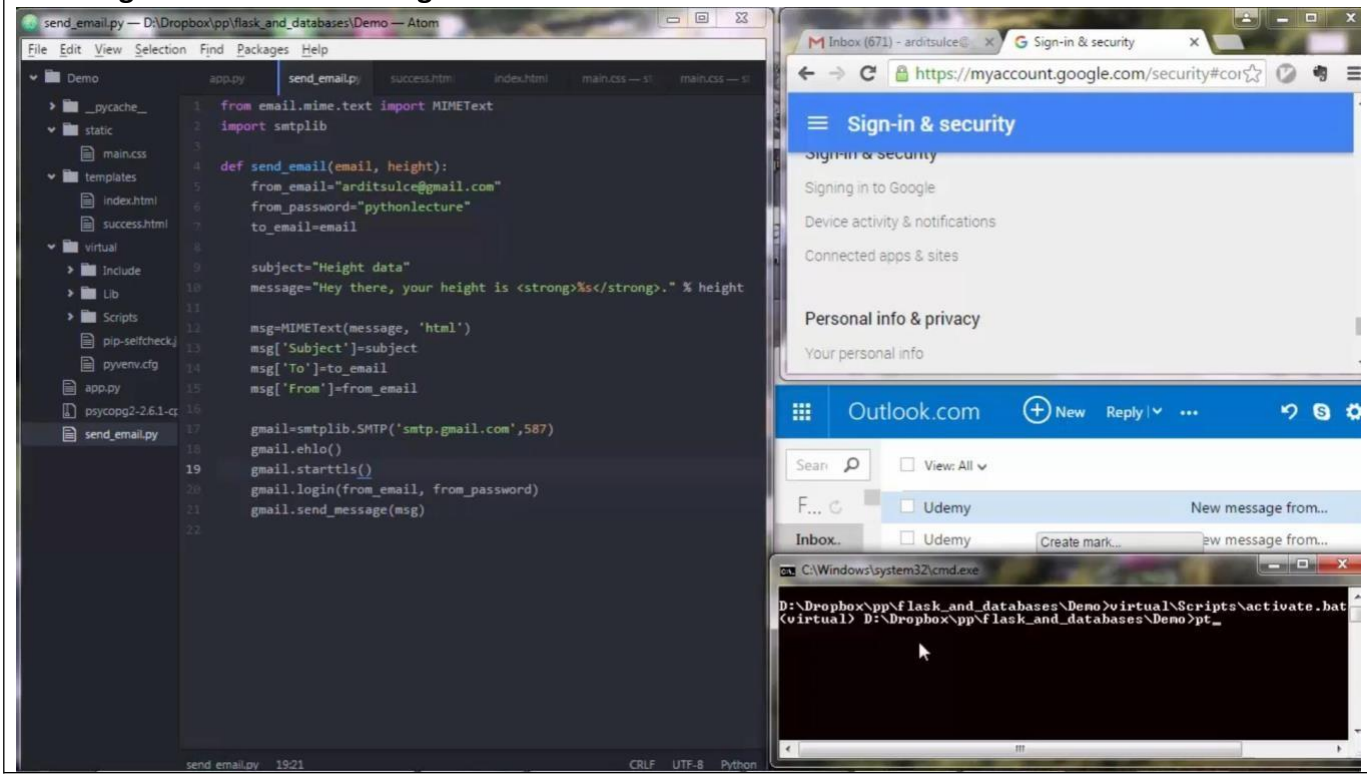
<html lang="en">

```
 <title> Data Collector App</title>

 <head>

  <link href="../static/main.css" rel="stylesheet">

 </head>

  <body>

    <div class="container">

      <h1>Collecting height</h1>

      <h3>Please fill the entires to get population statistics on height</h3>

      <div class="email"> {{text | safe}} </div>

      <form action="{{url_for('success')}}" method="POST">

       <input title="Your email will be safe with us" placeholder="Enter your email address" type="email"
name="email_name" required> <br>

       <input title="Your data will be safe with us" placeholder="Enter your height in cm" type="number"
min="50", max="300" name="height_name" required> <br>

       <button type="submit"> Submit </button>

      </form>

    </div>

  </body>

</html>
```

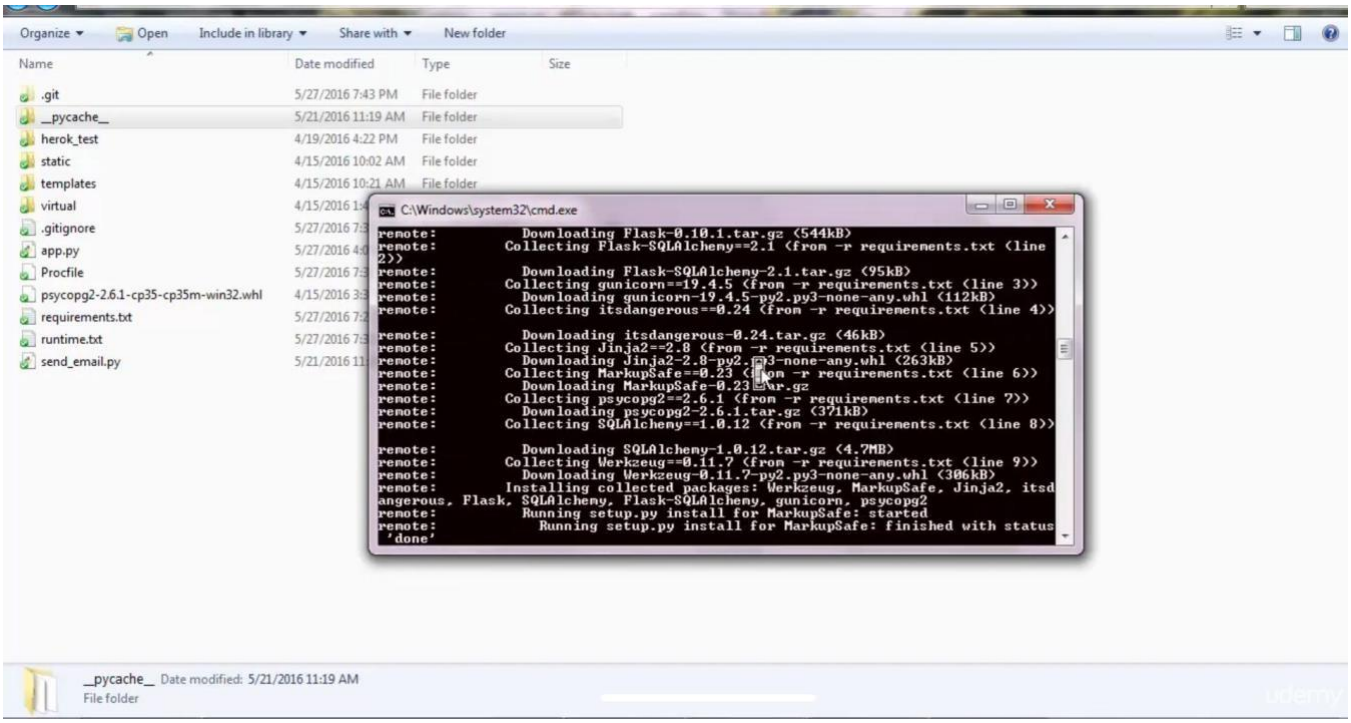
Building the Front End CSS Part



Building the Back End Emailing Database Values Back to the User



Deploying the Web Application to a Live Server



Program:

```
from flask import Flask, render_template, request

from flask.ext.sqlalchemy import SQLAlchemy

from send_email import send_email

from sqlalchemy.sql import func


app=Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI']='postgresql://postgres:postgres123@localhost/height_collector'

db=SQLAlchemy(app)


class Data(db.Model):
```

```
    __tablename__="data"
    id=db.Column(db.Integer, primary_key=True)
    email_=db.Column(db.String(120), unique=True)
    height_=db.Column(db.Integer)


    def __init__(self, email_, height_):
        self.email_=email_
        self.height_=height_


@app.route("/")
def index():
    return render_template("index.html")


@app.route("/success", methods=['POST'])
def success():
    if request.method=='POST':
        email=request.form["email_name"]
        height=request.form["height_name"]
        print(email, height)
        if db.session.query(Data).filter(Data.email_ == email).count()== 0:
            data=Data(email,height)
            db.session.add(data)
            db.session.commit()
```

```
        average_height=db.session.query(func.avg(Data.height_)).scalar()

        average_height=round(average_height, 1)

        count = db.session.query(Data.height_).count()

        send_email(email, height, average_height, count)

        print(average_height)

        return render_template("success.html")

    return render_template('index.html', text="Seems like we got something from that email once!")


if __name__== '_main_':

    app.debug=True

    app.run(port=5005)
```

Send_email program:

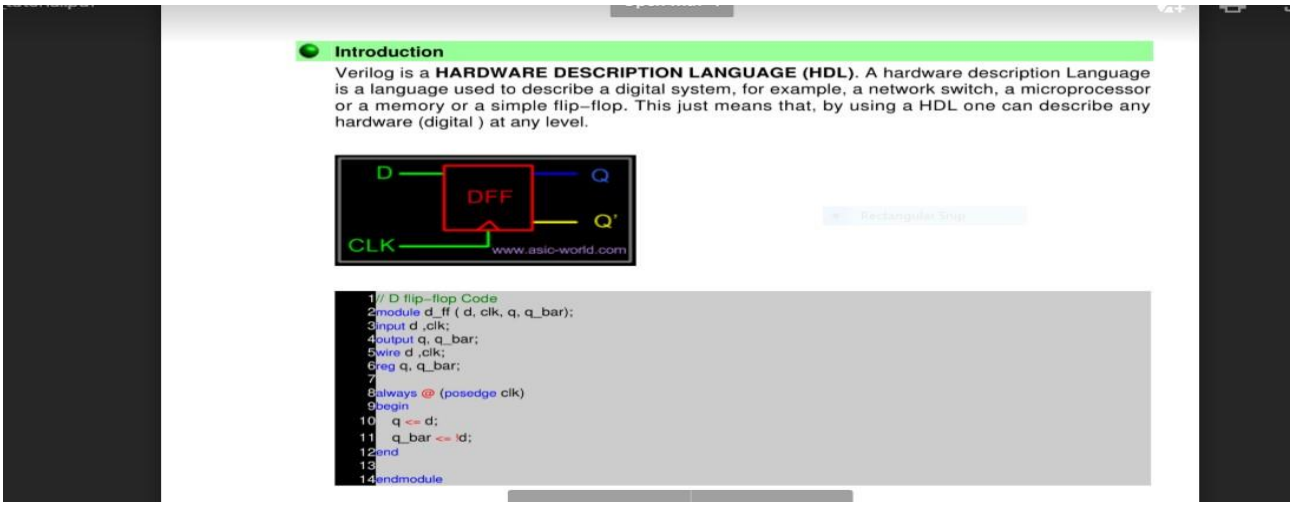
```
from email.mime.text import MIMEText

import smtplib


def send_email(email, height, average_height, count):

    from_email="myemail@gmail.com"

    from_password="mypassword"

    to_email=email


    subject="Height data"

    message="Hey there, your height is <strong>%s</strong>. <br> Average height of all is
<strong>%s</strong> and that is calculated out of <strong>%s</strong> people. <br> Thanks!" % (height,
average_height, count)
```

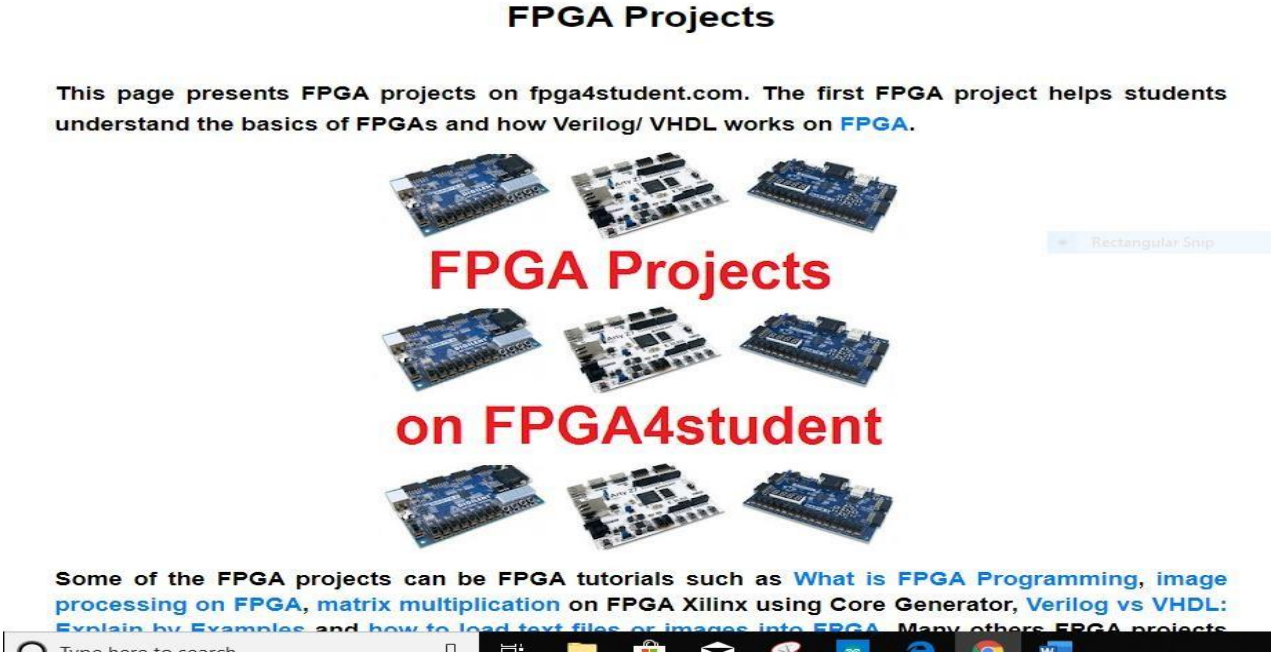
```
msg=MIMEText(message, 'html')

msg['Subject']=subject

msg['To']=to_email

msg['From']=from_email


gmail=smtplib.SMTP('smtp.gmail.com',587)

gmail.ehlo()

gmail.starttls()

gmail.login(from_email, from_password)

gmail.send_message(msg)
```

Name:Ramanath Naik

USN:4AL16EC054

Course: HDL **Topic: Day 5**





Introduction

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). A hardware description Language is a language used to describe a digital system, for example, a network switch, a microprocessor or a memory or a simple flip−flop. This just means that, by using a HDL one can describe any hardware (digital) at any level.

```
// D flip−flop Code
module d_ff ( d, clk, q, q_bar);
input d ,clk;
output q, q_bar;
wire d ,clk;
reg q, q_bar;
always @ (posedge clk)
begin
 q <= d;
q_bar <= !d;
end
endmodule
```

One can describe a simple Flip flop as that in above figure as well as one can describe a complicated designs having 1 million gates. Verilog is one of the HDL languages available in the industry for designing the Hardware. Verilog allows us to design a Digital design at Behavior Level,Register Transfer Level (RTL), Gate level and at switch level. Verilog allows hardware designers to express their designs with behavioral constructs, deterring the details of implementation to a later stage of design in the final design.

Many engineers who want to learn Verilog, most often ask this question, how much time it will take to learn Verilog?, Well my answer to them is "It may not take more then one week, if you happen to know at least one programming language".

Design Styles Verilog like any other hardware description language, permits the designers to design a design in either Bottom−up or Top−down methodology.

Bottom−Up Design

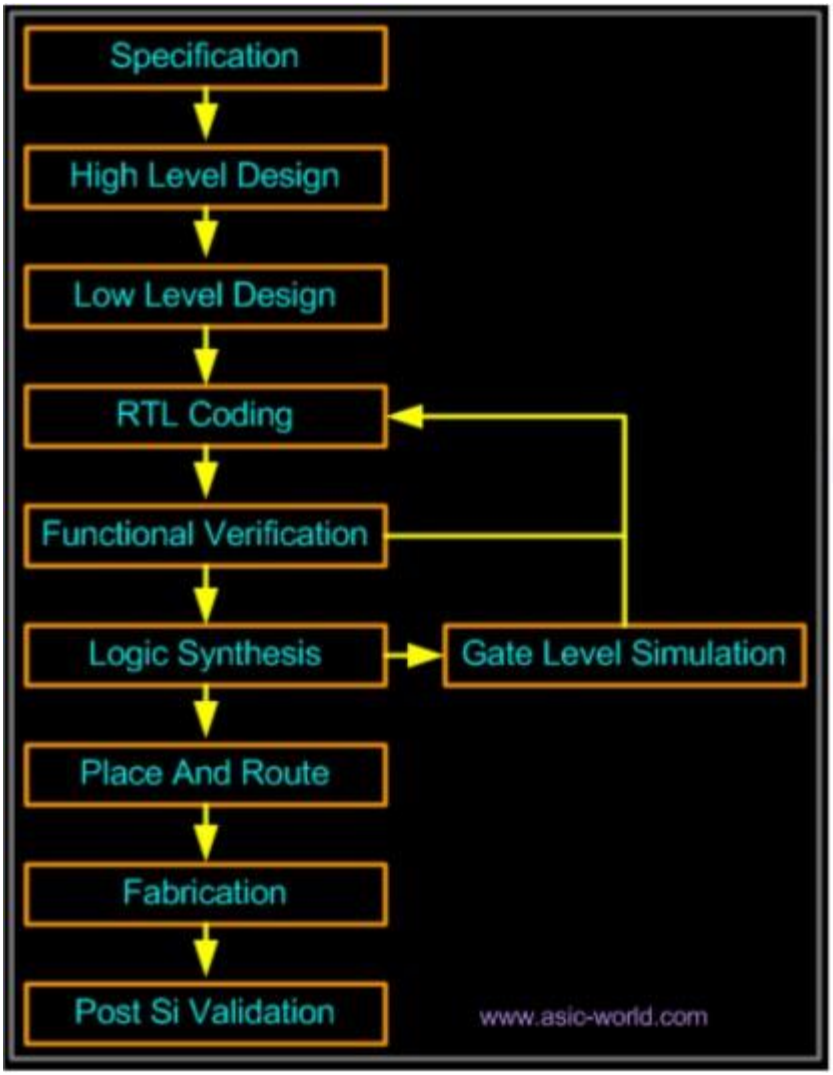
The traditional method of electronic design is bottom−up. Each design is performed at the gate−level using the standard gates (Refer to the Digital Section for more details) With increasing complexity of new designs this approach is nearly impossible to maintain. New systems consist of ASIC or microprocessors with a complexity

of thousands of transistors. These traditional bottom−up designs have to give way to new structural, hierarchical design methods. Without these new design practices it would be impossible to handle the new complexity.

Top−Down Design

The desired design−style of all designers is the top−down design. A real top−down design allows early testing, easy change of different technologies, a structured system design and offers many other advantages. But it is very difficult to follow a pure top−down design. Due to this fact most designs are mix of both the methods, implementing some key elements of both design styles.

Figure shows a Top−Down design approach.



Abstraction Levels of Verilog

Verilog supports a design at many different levels of abstraction. Three of them are very important:

• Behavioral level

• Register−Transfer Level
• Gate Level

Behavioral level

This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is sequential, that means it consists of a set of instructions that are executed one after the other. Functions, Tasks and Always blocks are the main elements. There is no regard to the structural realization of the design.
Register−Transfer Level Designs using the Register−Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers. An explicit clock is used. RTL design contains exact timing possibility, operations are scheduled to occur at certain times. Modern definition of a RTL
code is "Any code that is synthesizable is called RTL code".

Gate Level

Within the logic level the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X',`Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates). Using gate level modeling might not be a good idea for any level of logic design. Gate level code is generated by tools like synthesis tools and this netlist is used for gate level simulation and for backend.

Data Types

Verilog Language has two primary data types
• Nets − represents structural connections between components.
• Registers − represent variables used to store data.
Every signal has a data type associated with it:
• Explicitly declared with a declaration in your Verilog code.
Implicitly declared with no declaration but used to connect structural building blocks in your code.
•Implicit declaration is always a net type "wire" and is one bit wide.

Types of Nets

Each net type has functionality that is used to model different types of hardware (such as PMOS,NMOS, CMOS, etc)

Net Data Type	Functionality
wire, tri	Interconnecting wire – no special resolution function
wor, trior	Wired outputs OR together (models ECL)
wand,triand	Wired outputs AND together (models open–collector)
tri0,tri1	Net pulls–down or pulls–up when not driven
supply0,suppy1	Net has a constant logic 0 or logic 1 (supply strength)
trireg	

Register Data Types

Registers store the last value assigned to them until another assignment statement changes their value.

• Registers represent data storage constructs.

• You can create arrays of the regs called memories.

• register data types are used as variables in procedural blocks.

• A register data type is required if a signal is assigned a value within a procedural block

• Procedural blocks begin with keyword initial and always.

Data Types	Functionality
reg	Unsigned variable
integer	Signed variable – 32 bits
time	Unsigned integer – 64 bits
real	Double precision floating point variable

Some of the FPGA projects can be FPGA tutorials such as What is FPGA Programming, image processing on FPGA, matrix multiplication on FPGA Xilinx using Core Generator, Verilog vs VHDL: Explain by Examples and how to load text files or images into FPGA. Many others FPGA projects provide students with full Verilog/ VHDL source code to practice and run on FPGA boards. Some of them can be used for another bigger FPGA projects.

Task

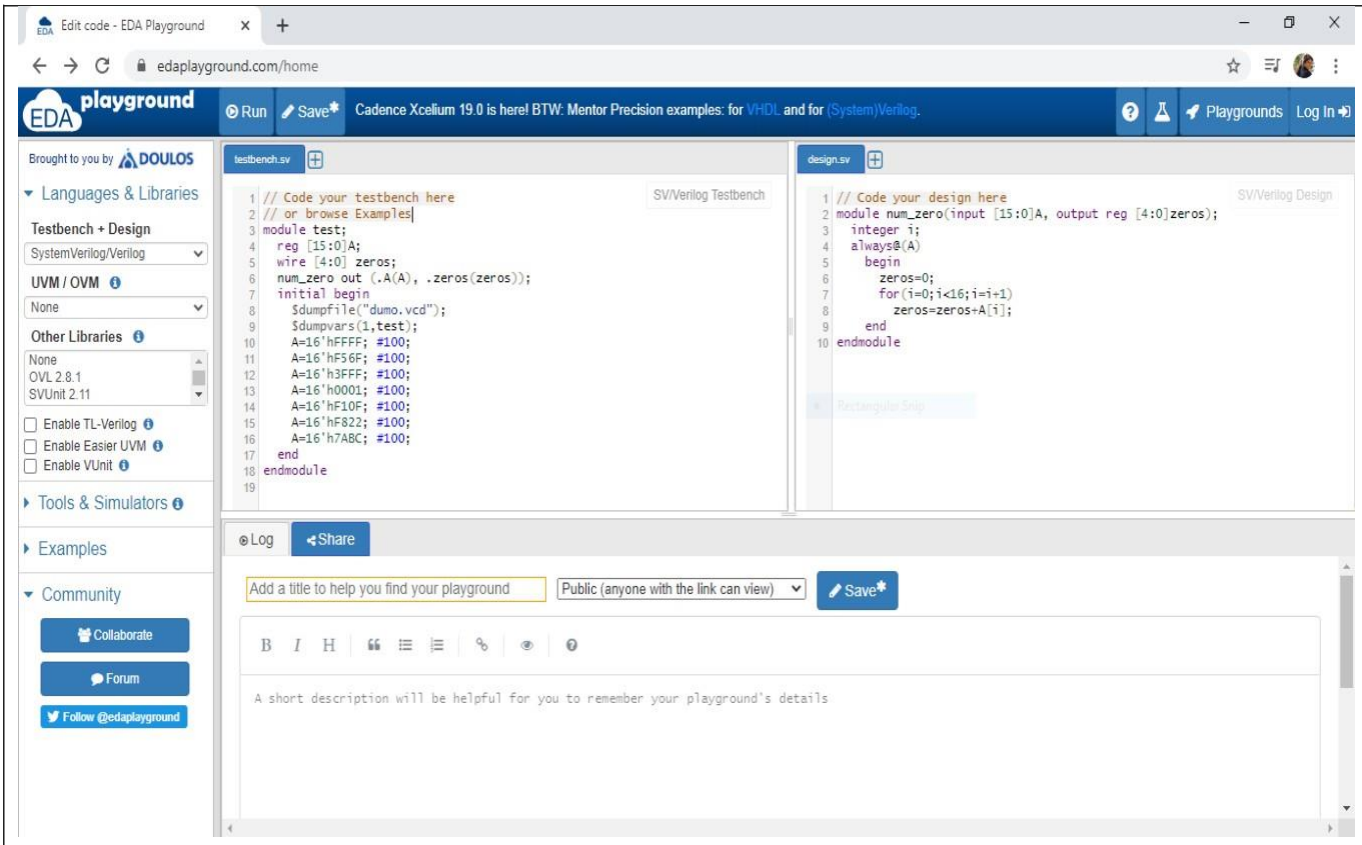
code

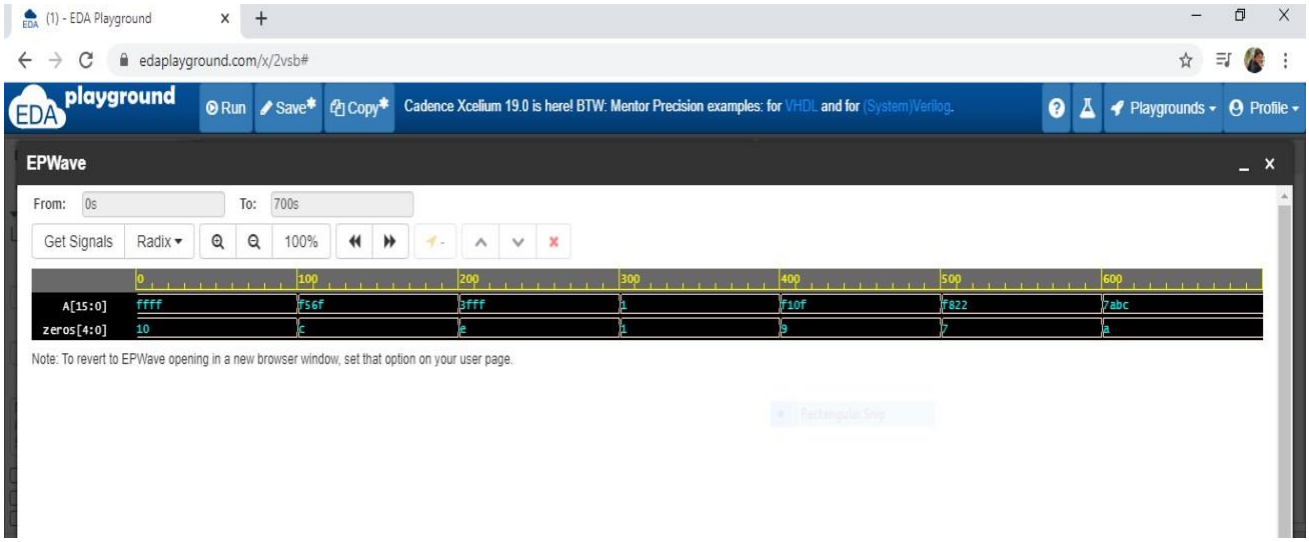
```
module num_zero(input [15:0]A, output reg [4:0]zeros);
```

```
  integer i;
  always@(A)
  begin
     zeros=0;
    for(i=0;i<16;i=i+1)
    zeros=zeros+A[i];
    end
endmodule
```

test bench code

```
module test;
  reg [15:0]A;
  wire [4:0] zeros;
  num_zero out (.A(A), .zeros(zeros));
  initial begin
    $dumpfile("dumo.vcd");
    $dumpvars(1,test);
    A=16'hFFFF; #100;
    A=16'hF56F; #100;
    A=16'h3FFF; #100;
    A=16'h0001; #100;
    A=16'hF10F; #100;
    A=16'hF822; #100;
    A=16'h7ABC; #100;
  end
endmodule
```


Edit code - EDA Playground
edaplayground.com/home
playground
Run
Save*
Cadence Xcelium 19.0 is here! BTW: Mentor Precision examples: for VHDL and for (System)Verilog.
Playgrounds
Log In
Brought to you by DOULOS
Languages & Libraries
Testbench + Design
SystemVerilog/Verilog
UVM / OVM
None
Other Libraries
None
OVL 2.8.1
SVUnit 2.11
Enable TL-Verilog
Enable Easier UVM
Enable VUnit
Tools & Simulators
Examples
Community
Collaborate
Forum
Follow @edaplayground
testbench.sv
SV/Verilog Testbench
// Code your testbench here
// or browse Examples
module test;
reg [15:0]A;
wire [4:0] zeros;
num_zero out (.A(A), .zeros(zeros));
initial begin
$dumpfile("dumo.vcd");
$dumpvars(1,test);
A=16'hFFFF; #100;
A=16'hF56F; #100;
A=16'h3FFF; #100;
A=16'h0001; #100;
A=16'hF10F; #100;
A=16'hF822; #100;
A=16'h7ABC; #100;
end
endmodule
design.sv
SV/Verilog Design
// Code your design here
module num_zero(input [15:0]A, output reg [4:0]zeros);
integer i;
always@(A)
begin
zeros=0;
for(i=0;i<16;i=i+1)
zeros=zeros+A[i];
end
endmodule
Log
Share
Add a title to help you find your playground
Public (anyone with the link can view)
Save*
A short description will be helpful for you to remember your playground's details


(1) - EDA Playground
edaplayground.com/x/2vsb#
playground
Run
Save*
Copy*
Cadence Xcelium 19.0 is here! BTW: Mentor Precision examples: for VHDL and for (System)Verilog.
Playgrounds
Profile
EPWave
From:
0s
To:
700s
Get Signals
Radix
100%
A[15:0]
zeros[4:0]
ffff
f56f
3fff
1
f10f
f822
7abc
10
c
e
1
9
7
a
Note: To revert to EPWave opening in a new browser window, set that option on your user page.