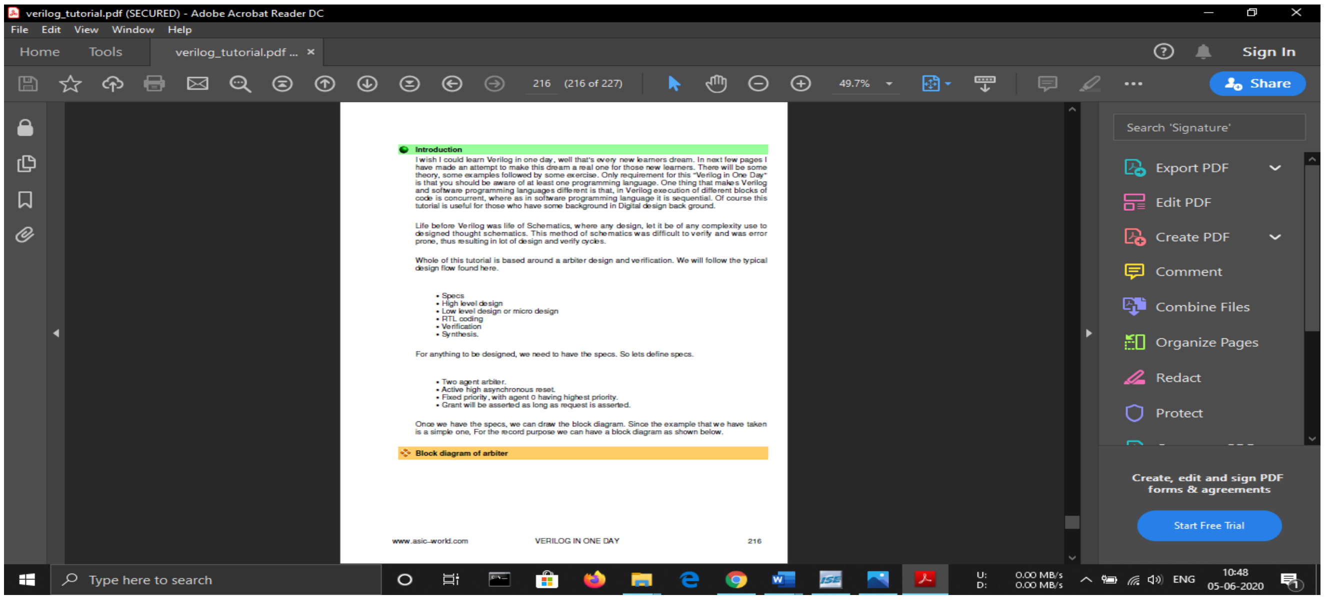
DAILY ASSESSMENT REPORT

Date:	05/06/2020	Name:	Abhishek M Shastry K
Subject:	Digital Design Using HDL	USN:	4AL17EC002
Topic:	1] Verilog Tutorials and practice programs 2] Building/ Demo projects using FPGA	Semester & Section:	6th 'A'
Github Repository:	AbhishekShastry-Courses		

FORENOON SESSION DETAILS

Image of session



Introduction

I wish I could learn Verilog in one day, well that's every new learners dream. In next few pages I have made an attempt to make this dream a real one for those new learners. There will be some theory, some examples followed by some exercise. Only requirement for this "Verilog in One Day" is that you should be aware of at least one programming language. One thing that makes Verilog and software programming languages different is that, in Verilog execution of different blocks of code is concurrent, where as in software programming language it is sequential. Of course this tutorial is useful for those who have some background in Digital design back ground.

Life before Verilog was life of Schematics, where any design, let it be of any complexity use to designed thought schematics. This method of schematics was difficult to verify and was error prone, thus resulting in lot of design and verify cycles.

Whole of this tutorial is based around a arbiter design and verification. We will follow the typical design flow found here.

- Specs
- High level design
- Low level design or micro design
- RTL coding
- Verification
- Synthesis.

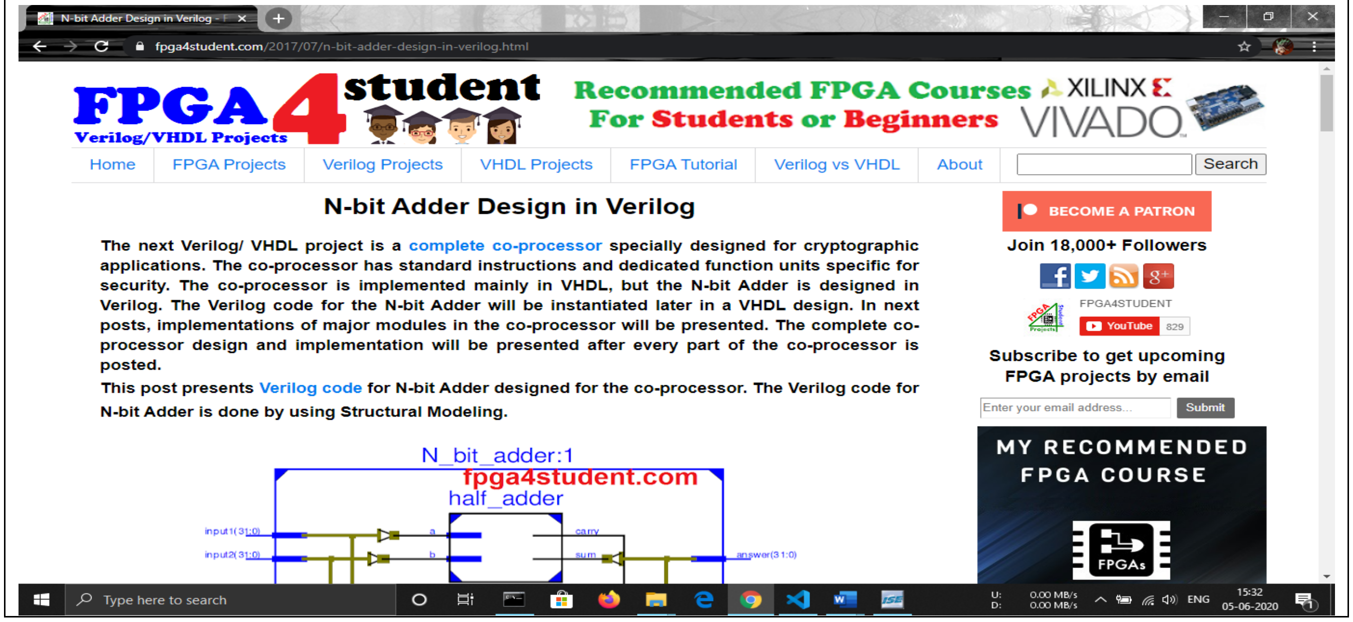
For anything to be designed, we need to have the specs. So lets define specs.

- Two agent arbiter.
- Active high asynchronous reset.
- Fixed priority, with agent 0 having highest priority.
- Grant will be asserted as long as request is asserted.

Once we have the specs, we can draw the block diagram. Since the example that we have taken is a simple one, For the record purpose we can have a block diagram as shown below.

Block diagram of arbiter

www.asic-world.com VERILOG IN ONE DAY 216



N-bit Adder Design in Verilog

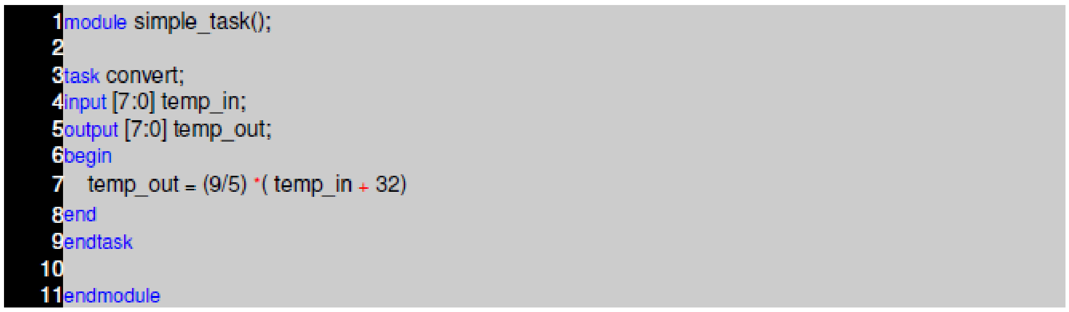
The next Verilog/ VHDL project is a complete co-processor specially designed for cryptographic applications. The co-processor has standard instructions and dedicated function units specific for security. The co-processor is implemented mainly in VHDL, but the N-bit Adder is designed in Verilog. The Verilog code for the N-bit Adder will be instantiated later in a VHDL design. In next posts, implementations of major modules in the co-processor will be presented. The complete co-processor design and implementation will be presented after every part of the co-processor is posted.

This post presents Verilog code for N-bit Adder designed for the co-processor. The Verilog code for N-bit Adder is done by using Structural Modeling.

Report

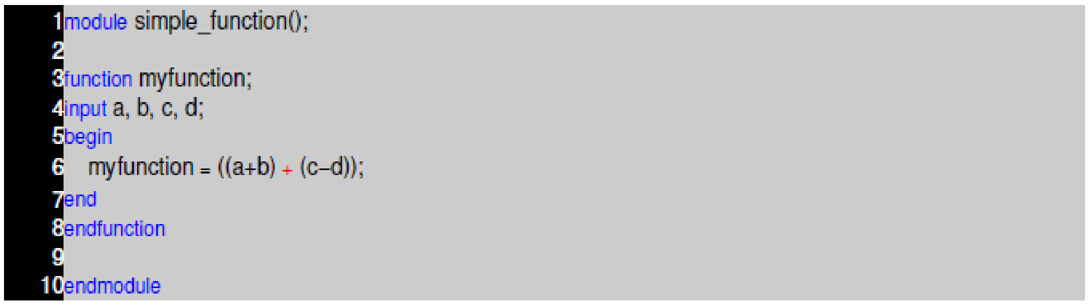
Tasks and Functions in Verilog

- **Tasks** are used in all programming languages, generally known as Procedures or sub routines.
- Many lines of code are enclosed in task.... end task brackets. Data is passed to the task, the processing done, and the result returned to a specified value. They have to be specifically called, with data ins and outs, rather than just wired in to the general netlist. Included in the main body of code they can be called many times, reducing code repetition.
    - ✓ Task is defined in the module in which they are used. it is possible to define task in separate file and use compile directive 'include to include the task in the file which instantiates the task.
    - ✓ Task can include timing delays, like posedge, negedge, # delay and wait.
    - ✓ Task can have any number of inputs and outputs.
    - ✓ The variables declared within the task are local to that task. The order of declaration within the task defines how the variables passed to the task by the caller are used.
    - ✓ Task can take, drive and source global variables, when no local variables are used. When local variables are used, it basically assigned output only at the end of task execution. Task can call another task or function.
    - ✓ Task can be used for modelling both combinational and sequential logic.
    - ✓ A task must be specifically called with a statement, it cannot be used within an expression as a function can.
- **Syntax**
    - ✓ Task begins with keyword task and end's with keyword **endtask.**
    - ✓ Input and output are declared after the keyword task.
    - ✓ Local variables are declared after input and output declaration.
- Example – **Simple Task**



```
module simple_task();

task convert;
input [7:0] temp_in;
output [7:0] temp_out;
begin
   temp_out = (9/5) *( temp_in + 32)
end
endtask

endmodule
```

- A **Verilog HDL function** is same as task, with very little difference, like function cannot drive more than one output, cannot contain delays.
    - ✓ Function is defined in the module in which they are used. it is possible to define function in separate file and use compile directive 'include to include the function in the file which instantiates the task.
    - ✓ Function cannot include timing delays, like posedge, negedge, # delay. Which means that function should be executed in "zero" time delay.
    - ✓ Function can have any number of inputs and but only one output.
    - ✓ The variables declared within the function are local to that function. The order of declaration within the function defines how the variables passed to the function by the caller is used.
    - ✓ Function can take drive and source global variables, when no local variables are used. When local variables are used, it basically assigned output only at the end of function execution.
    - ✓ Function can be used for modelling combinational logic.
    - ✓ Function can call other functions, but cannot call task.
- **Syntax**
    - ✓ Function begins with keyword function and end's with keyword **endfunction**.
    - ✓ Input is declared after the keyword function.
- Example – **Simple Function**



```
module simple_function();

function myfunction;
input a, b, c, d;
begin
   myfunction = ((a+b) + (c-d));
end
endfunction

endmodule
```

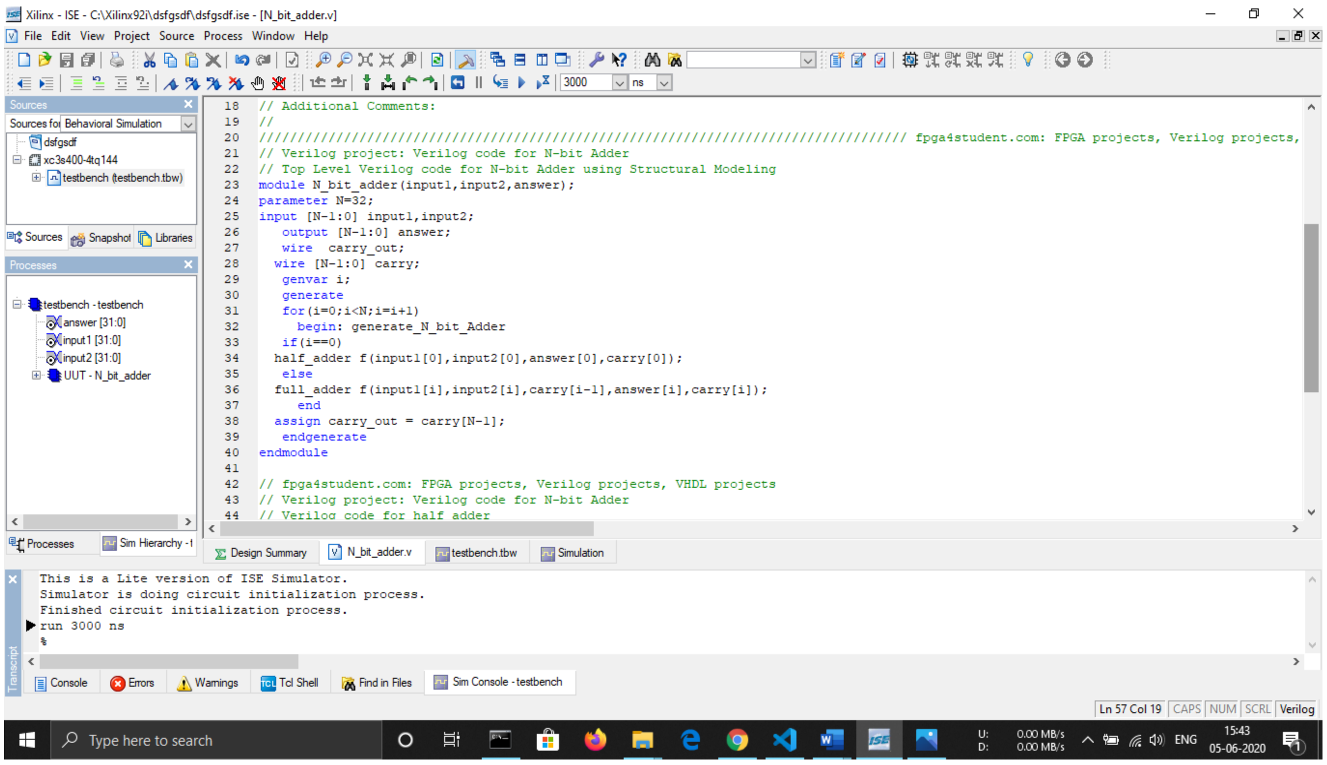
N-bit Adder Design in Verilog

The N-bit Adder is simply implemented by connecting 1 Half Adder and N-1 Full Adder in series. The Verilog code for N-bit Adder is designed so that the N value can be initialized independently for each instantiation. To do it, the Verilog code for N-bit Adder uses Generate Statement in Verilog to create a chain of full adders for implementing the N-bit Adder.

Verilog code:

```
// Verilog project: Verilog code for N-bit Adder
// Top Level Verilog code for N-bit Adder using Structural Modeling
module N_bit_adder(input1,input2,answer);
parameter N=32;
input [N-1:0] input1,input2;
   output [N-1:0] answer;
   wire  carry_out;
  wire [N-1:0] carry;
   genvar i;
   generate
   for(i=0;i<N;i=i+1)
     begin: generate_N_bit_Adder
   if(i==0)
  half_adder f(input1[0],input2[0],answer[0],carry[0]);
   else
  full_adder f(input1[i],input2[i],carry[i-1],answer[i],carry[i]);
     end
  assign carry_out = carry[N-1];
   endgenerate
endmodule

// Verilog project: Verilog code for N-bit Adder
// Verilog code for half adder
module half_adder(x,y,s,c);
   input x,y;
   output s,c;
   assign s=x^y;
   assign c=x&y;
endmodule // half adder

// Verilog project: Verilog code for N-bit Adder
// Verilog code for full adder
module full_adder(x,y,c_in,s,c_out);
   input x,y,c_in;
   output s,c_out;
   assign s = (x^y) ^ c_in;
   assign c_out = (y&c_in)| (x&y) | (x&c_in);
endmodule // full_adder
```

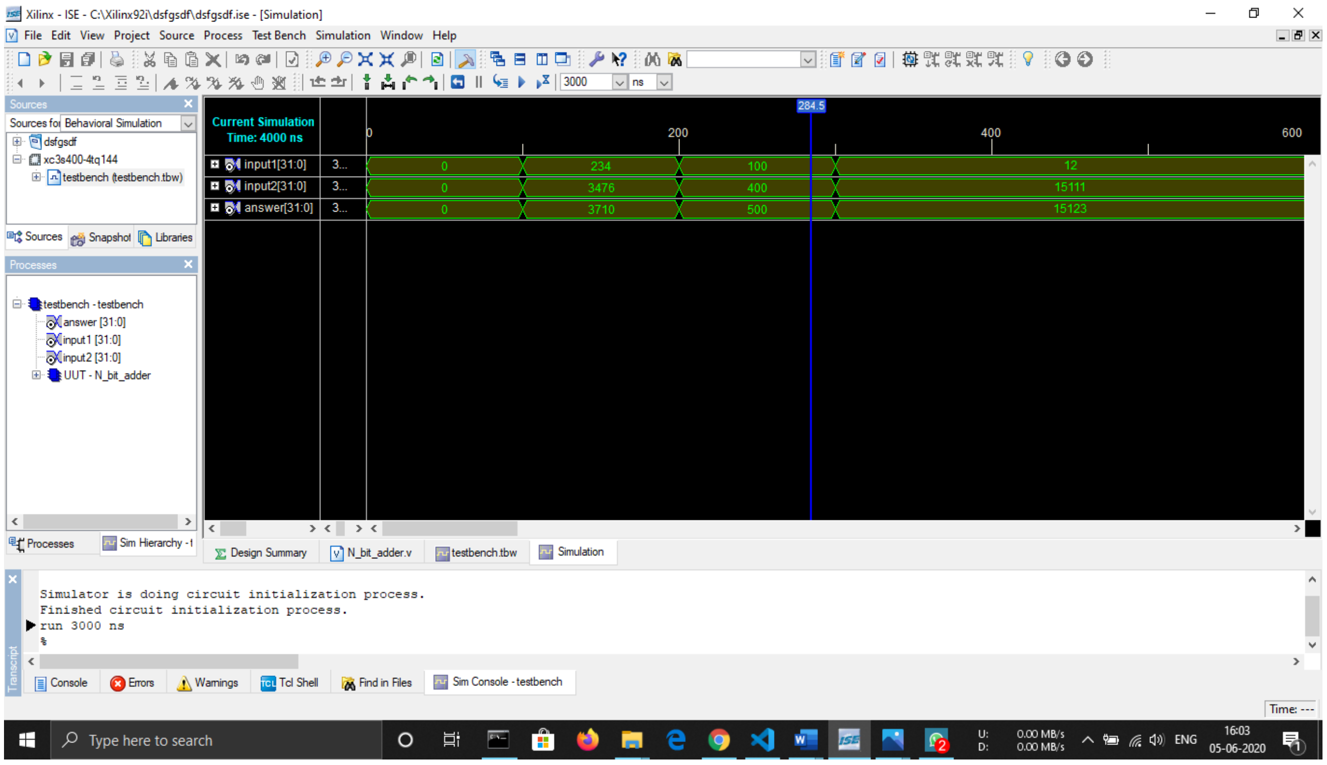
Compiler Output:



Xilinx - ISE - C:\Xilinx92i\dsfgsdf\dsfgsdf.ise - [N_bit_adder.v]
File Edit View Project Source Process Window Help
Sources for Behavioral Simulation
dsfgsdf
xc3s400-4tq144
testbench (testbench.tbw)
Sources Snapshot Libraries
Processes
testbench - testbench
answer [31:0]
input1 [31:0]
input2 [31:0]
UUT - N_bit_adder
Processes Sim Hierarchy
Design Summary N_bit_adder.v testbench.tbw Simulation
Console Errors Warnings Tcl Shell Find in Files Sim Console - testbench
Ln 57 Col 19 CAPS NUM SCRL Verilog
Type here to search
15:43 05-06-2020

```verilog
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////// fpga4student.com: FPGA projects, Verilog projects,
// Verilog project: Verilog code for N-bit Adder
// Top Level Verilog code for N-bit Adder using Structural Modeling
module N_bit_adder(input1,input2,answer);
parameter N=32;
input [N-1:0] input1,input2;
   output [N-1:0] answer;
   wire  carry_out;
  wire [N-1:0] carry;
   genvar i;
   generate
   for(i=0;i<N;i=i+1)
     begin: generate_N_bit_Adder
   if(i==0)
  half_adder f(input1[0],input2[0],answer[0],carry[0]);
   else
  full_adder f(input1[i],input2[i],carry[i-1],answer[i],carry[i]);
     end
  assign carry_out = carry[N-1];
   endgenerate
endmodule

// fpga4student.com: FPGA projects, Verilog projects, VHDL projects
// Verilog project: Verilog code for N-bit Adder
// Verilog code for half adder
```

```
This is a Lite version of ISE Simulator.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
run 3000 ns
%
```



Xilinx - ISE - C:\Xilinx92i\dsfgsdf\dsfgsdf.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Current Simulation Time: 4000 ns
284.5
0
200
400
600
input1[31:0] 3... 0 234 100 12
input2[31:0] 3... 0 3476 400 15111
answer[31:0] 3... 0 3710 500 15123
Design Summary N_bit_adder.v testbench.tbw Simulation
Console Errors Warnings Tcl Shell Find in Files Sim Console - testbench
Time: ---
Type here to search
16:03 05-06-2020

```
Simulator is doing circuit initialization process.
Finished circuit initialization process.
run 3000 ns
%
```

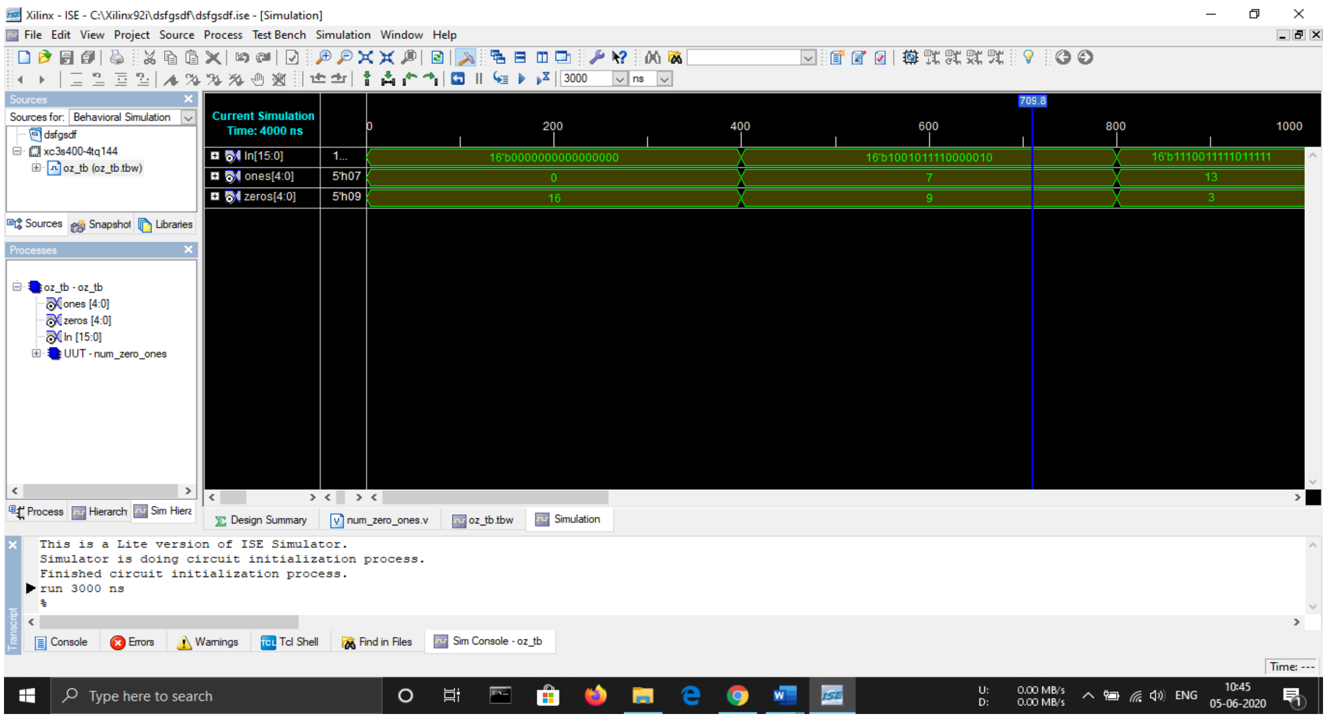
Task (DAY - 5)

Implement a Verilog module to count number of 1's and 0's in a 16-bit number in compiler.

Verilog Code:

```
module num_zero_ones (
    input [15:0] In,
    output reg [4:0] ones,
    output reg [4:0] zeros);

integer i, o, z;

always @ (In)
begin
    o = 0;  //initialize count variable.
    z = 0;  //initialize count variable.
    for(i=0;i<16;i=i+1)   //check for all the bits.
        if(In[i] == 1'b1)    //check if the bit is '1'
            o = o + 1;    //if its one, increment the count.
    z = 16-o; //number of zeros.
    ones = o;
    zeros = z;
end

endmodule
```

Compiler Output:


Xilinx - ISE - C:\Xilinx92i\dsfgsdf\dsfgsdf.ise - [Simulation]
File Edit View Project Source Process Test Bench Simulation Window Help
Sources
Sources for: Behavioral Simulation
dsfgsdf
xc3s400-4tq144
oz_tb (oz_tb.tbw)
Sources Snapshot Libraries
Processes
oz_tb - oz_tb
ones [4:0]
zeros [4:0]
In [15:0]
UUT - num_zero_ones
Process Hierarch Sim Hiera
Current Simulation Time: 4000 ns
709.8
0 200 400 600 800 1000
Design Summary num_zero_ones.v oz_tb.tbw Simulation
Console Errors Warnings Tcl Shell Find in Files Sim Console - oz_tb
Transcript
Time: ---
Type here to search
10:45 05-06-2020

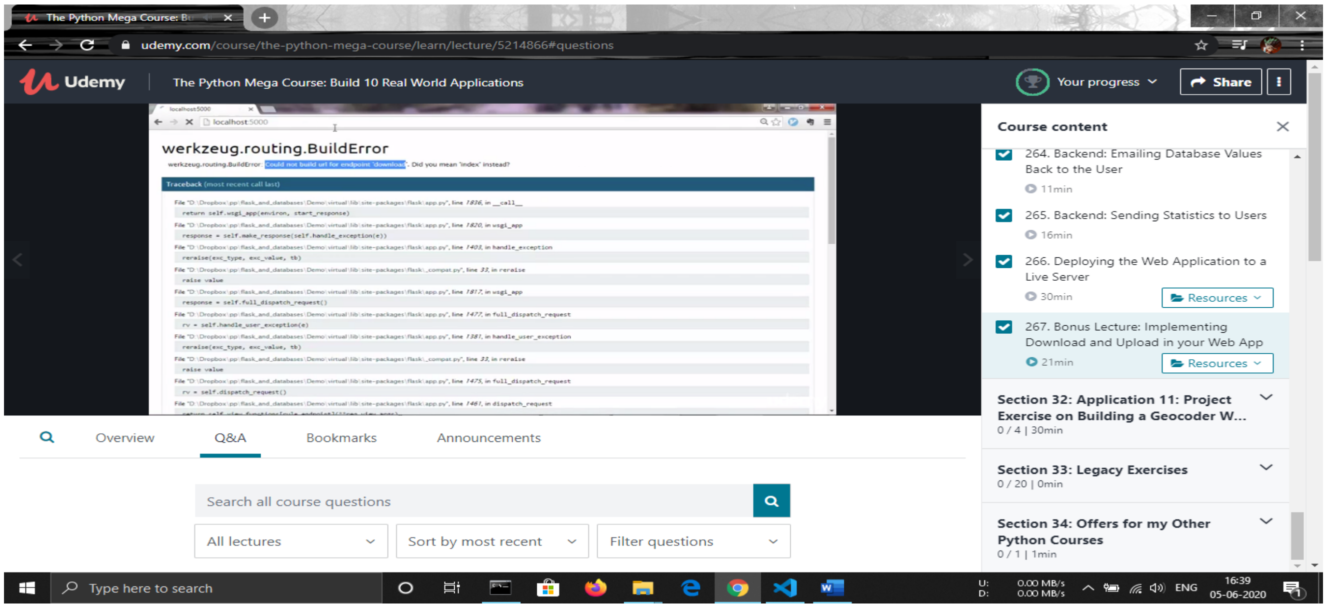
```
In[15:0]     1...    16'b0000000000000000 | 16'b1001011110000010 | 16'b1110011111011111
ones[4:0]    5'h07   0                    | 7                    | 13
zeros[4:0]   5'h09   16                   | 9                    | 3
```

```
This is a Lite version of ISE Simulator.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
run 3000 ns
%
```

Date:	**05/06/2020**	**Name:**	**Abhishek M Shastry K**
Course:	**The Python Mega Course: Build 10 Real World Applications**	**USN:**	**4AL17EC002**
Topic:	**1] Application 9: Build a Data Collector Web App with PostGreSQL and Flask**	**Semester & Section:**	**6th ‘A’**
Github Repository:	**AbhishekShastry-Courses**		

AFTERNOON SESSION DETAILS

Image of session



werkzeug.routing.BuildError

werkzeug.routing.BuildError: Could not build url for endpoint 'download'. Did you mean 'index' instead?

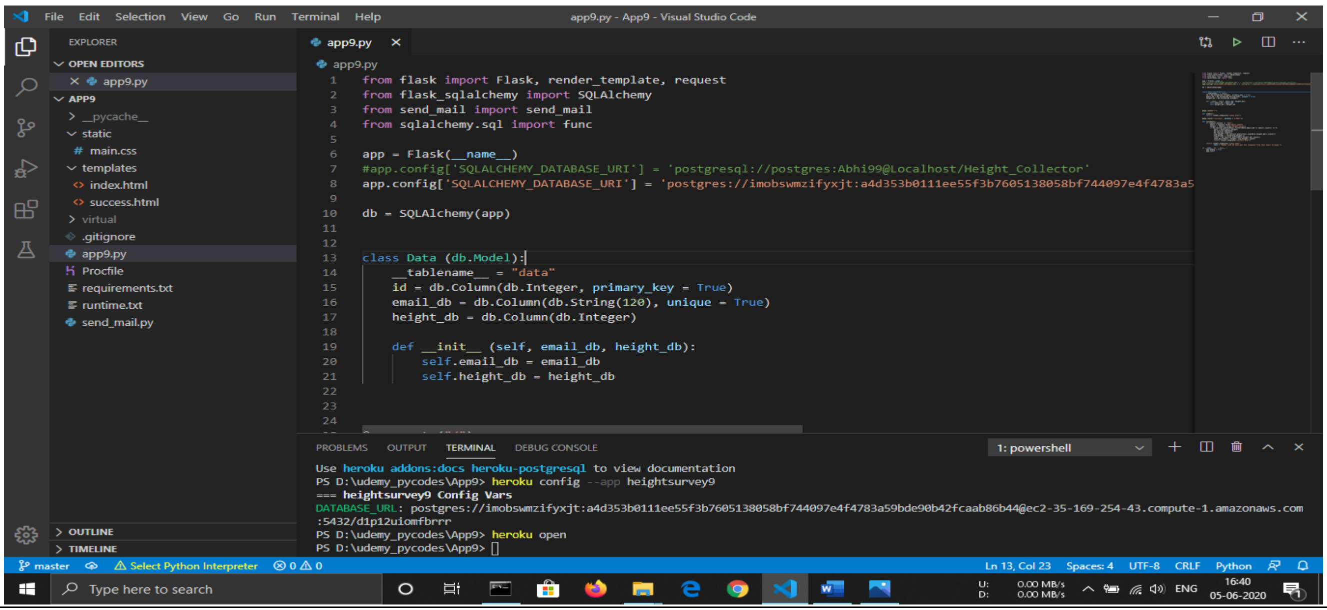
Course content

264. Backend: Emailing Database Values Back to the User — 11min
265. Backend: Sending Statistics to Users — 16min
266. Deploying the Web Application to a Live Server — 30min
267. Bonus Lecture: Implementing Download and Upload in your Web App — 21min

Section 32: Application 11: Project Exercise on Building a Geocoder W... 0 / 4 | 30min

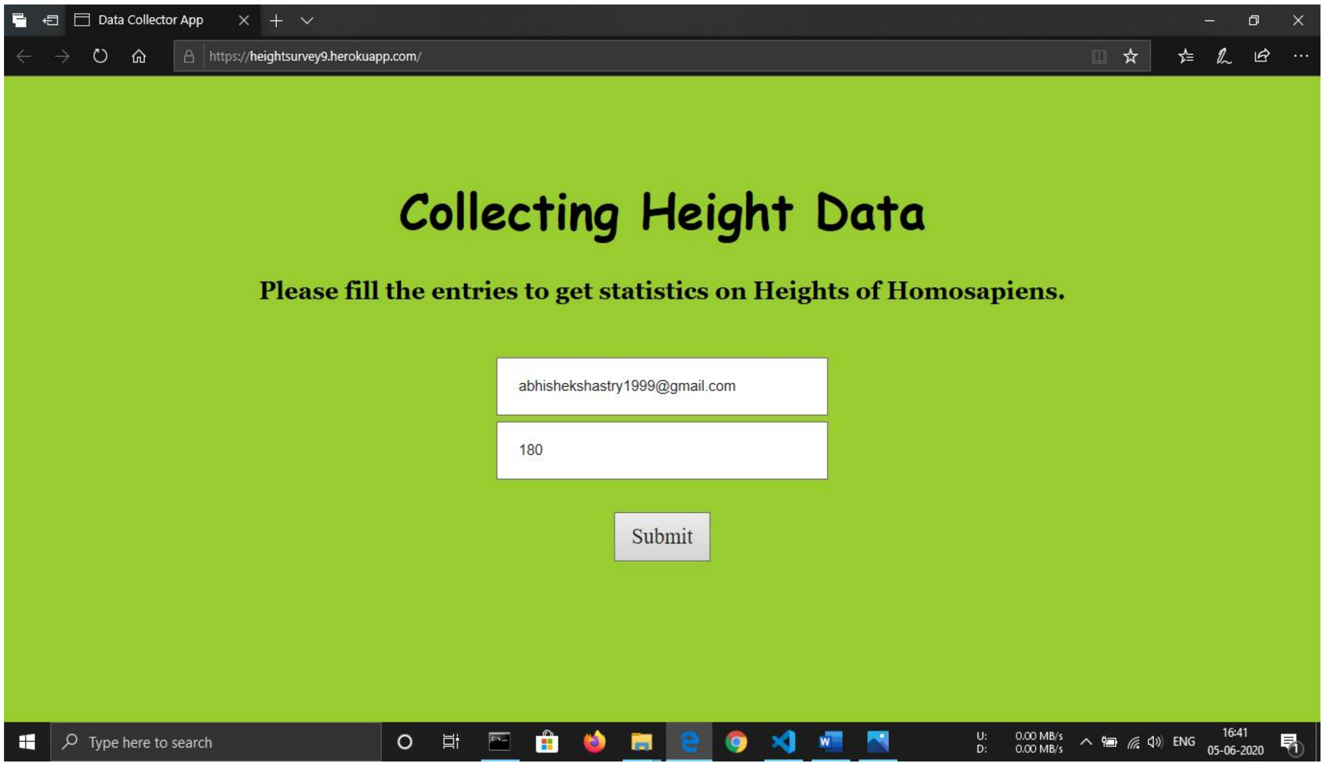
Section 33: Legacy Exercises 0 / 20 | 0min

Section 34: Offers for my Other Python Courses 0 / 1 | 1min



```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
from send_mail import send_mail
from sqlalchemy.sql import func

app = Flask(__name__)
#app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:Abhi99@Localhost/Height_Collector'
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgres://imobswmzifyxjt:a4d353b0111ee55f3b7605138058bf744097e4f4783a5

db = SQLAlchemy(app)


class Data (db.Model):
    __tablename__ = "data"
    id = db.Column(db.Integer, primary_key = True)
    email_db = db.Column(db.String(120), unique = True)
    height_db = db.Column(db.Integer)

    def __init__ (self, email_db, height_db):
        self.email_db = email_db
        self.height_db = height_db
```

```
Use heroku addons:docs heroku-postgresql to view documentation
PS D:\udemy_pycodes\App9> heroku config --app heightsurvey9
=== heightsurvey9 Config Vars
DATABASE_URL: postgres://imobswmzifyxjt:a4d353b0111ee55f3b7605138058bf744097e4f4783a59bde90b42fcaab86b44@ec2-35-169-254-43.compute-1.amazonaws.com
:5432/d1p12uiomfbrrr
PS D:\udemy_pycodes\App9> heroku open
PS D:\udemy_pycodes\App9> 
```



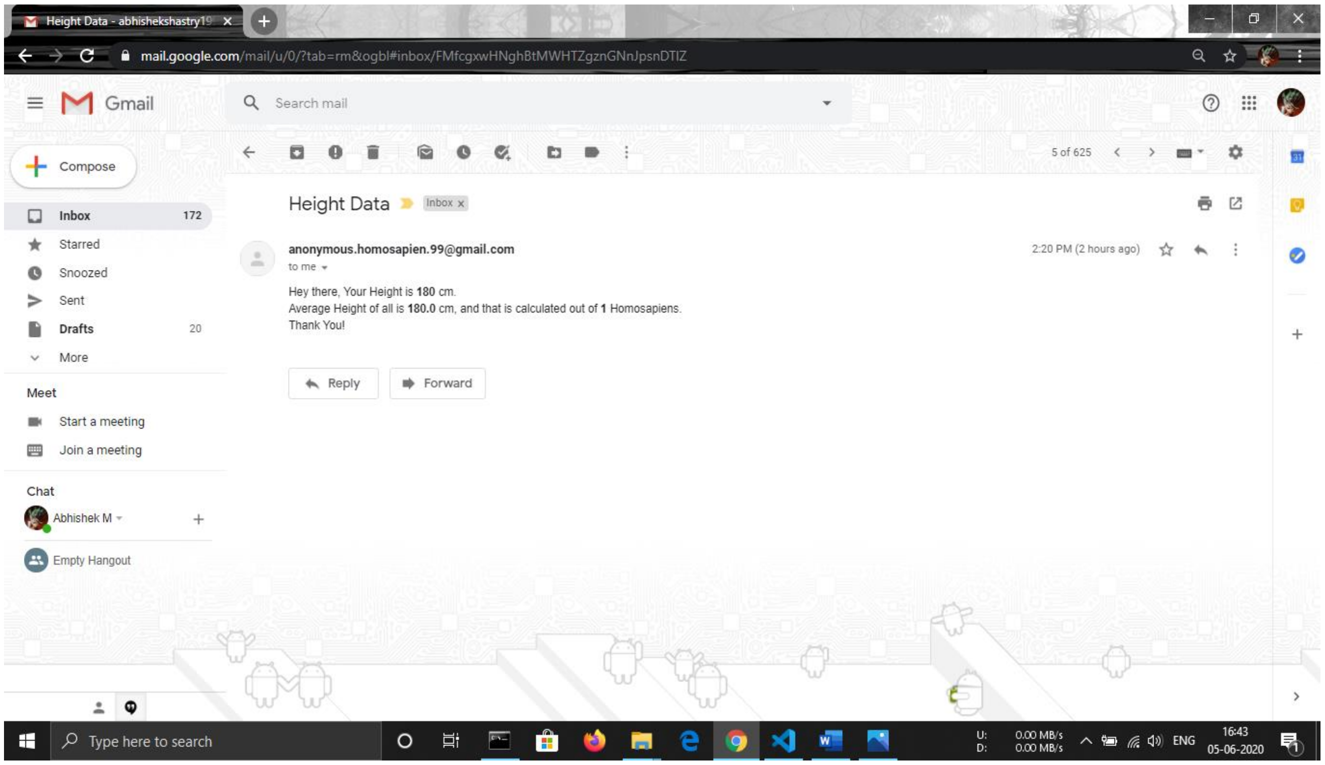
Collecting Height Data

Please fill the entries to get statistics on Heights of Homosapiens.

abhishekshastry1999@gmail.com

180

Submit



Height Data Inbox

anonymous.homosapien.99@gmail.com
to me
2:20 PM (2 hours ago)

Hey there, Your Height is **180** cm.
Average Height of all is **180.0** cm, and that is calculated out of **1** Homosapiens.
Thank You!

Report

Application 9: Build a Data Collector Web App with PostGreSQL and Flask

- Python code to build a data collector web application, which collects height data from the user and sends the survey result via e-mail.
- Some of the functions/modules used in this application:
  - ✓ **SQLAlchemy** is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.
  - ✓ SQL functions which are known to SQLAlchemy with regards to database-specific rendering, return types and argument behavior. Generic functions are invoked like all SQL functions, using the **func** attribute.
  - ✓ **email.mime** module can create a new object structure by creating Message instances, adding attachments and all the appropriate headers manually. For MIME messages the email package provides some convenient subclasses to make things easier.
  - ✓ The **smtplib** module defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon. For details of SMTP and ESMTP operation, consult RFC 821 (Simple Mail Transfer Protocol) and RFC 1869 (SMTP Service Extensions).
  - ✓ **Extended HELO (EHLO)** is an Extended Simple Mail Transfer Protocol (ESMTP) command sent by an email server to identify itself when connecting to another email server to start the process of sending an email. The EHLO command tells the receiving server it supports extensions compatible with ESMTP.
  - ✓ The **starttls ()** command extends the Transport Layer Security (TLS) protocol in order to encrypt the information transmitted using the TLS protocol. **Starttls ()** is mainly used as a protocol extension for communication by e-mail, based on the protocol's SMTP, IMAP and POP.
- To deploy the application to a live web server, Heroku platform is used and to save the data from the users, database is created in Heroku using credentials.