DAILY ASSESSMENT

Date:	**5-6-2020**	**Name:**	**Kavyashree m**
Course:	**Digital design using HDL**	**USN:**	**4al15ec036**
Topic:	**Verilog Tutorials and practice programs, Building/ Demo projects using FPGA**	**Semester & Section:**	**8th A**
Github Repository:	**kavya**		

FORENOON SESSION DETAILS

Image of session

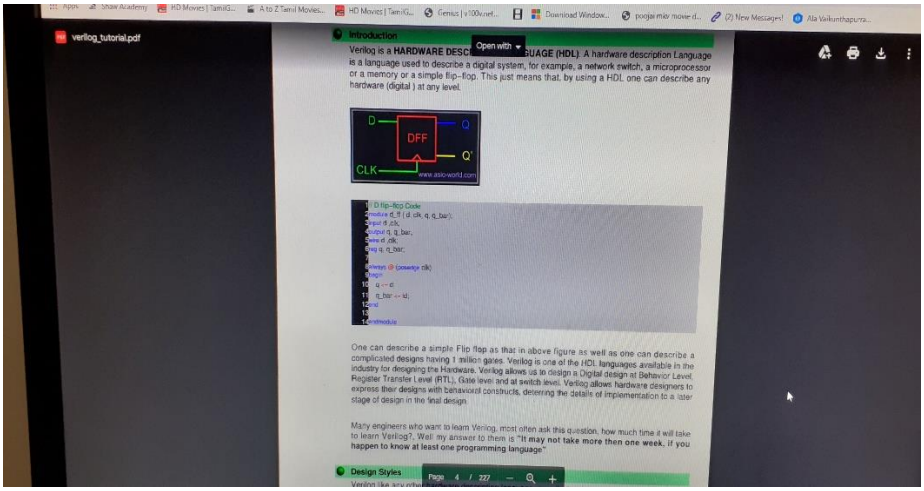


Fig 1: Verilog Tutorials and practice programs

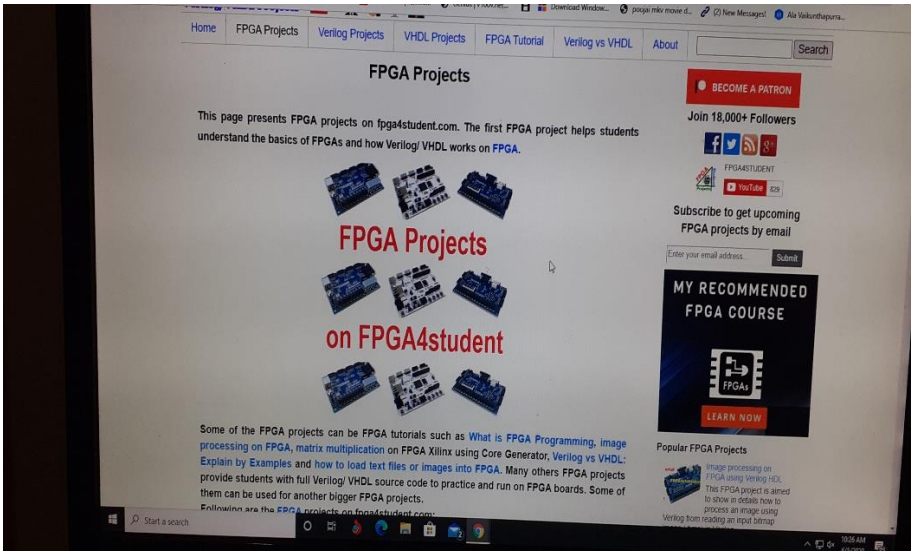


Fig 2 : Building/ Demo projects using FPGA

Verilog Tutorials and practice programs

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 at any level.

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.

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.

Various stages of ASIC/FPGA

- Specification : Word processor like Word, Kwriter, AbiWord, Open Office.
- High Level Design : Word processor like Word, Kwriter, AbiWord, for drawing waveform
  use tools like waveformer or testbencher or Word, Open Office.
- Micro Design/Low level design: Word processor like Word, Kwriter, AbiWord, for drawing
  waveform use tools like waveformer or testbencher or Word. For FSM StateCAD or some
  similar tool, Open Office.
- RTL Coding : Vim, Emacs, conTEXT, HDL TurboWriter.
- Simulation : Modelsim, VCS, Verilog−XL, Veriwell, Finsim, iVerilog, VeriDOS.

- Synthesis : Design Compiler, FPGA Compiler, Synplify, Leonardo Spectrum. You can
  download this from FPGA vendors like Altera and Xilinx for free.
- Place & Route : For FPGA use FPGA' vendors P&R tool. ASIC tools require expensive P&R tools like Apollo. Students can use LASI, Magic.
- Post Si Validation : For ASIC and FPGA, the chip needs to be tested in real environment.
- Board design, device drivers needs to be in place.

Hello World Program

```
1//------------------------------------------------------------
2// This is my first Verilog Program
3// Design Name : hello_world
4// File Name : hello_world.v
5// Function : This program will print 'hello world'
6// Coder : Deepak
7//------------------------------------------------------------
8module hello_world ;
9
10initial begin
11 $display ( "Hello World by Deepak" );
12 #10 $finish;
13end
14
15endmodule // End of Module hello_world
```

Counter Design Specs

• 4−bit synchronous up counter.

• active high, synchronous reset.

• Active high enable.

Counter Design

```
1//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
2// This is my second Verilog Design
3// Design Name : first_counter
4// File Name : first_counter.v
5// Function : This is a 4 bit up−counter with
6// Synchronous active high reset and
7// with active high enable signal
8//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
9module first_counter (
10clock , // Clock input ot the design
11reset , // active high, synchronous Reset input
12enable , // Active high enabel signal for counter
13counter_out // 4 bit vector output of the counter
14); // End of port list
15//−−−−−−−−−−−−−−Input Ports−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
16input clock ;
17input reset ;
18input enable ;
19//−−−−−−−−−−−−−−Output Ports−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
20output [3:0] counter_out ;
21//−−−−−−−−−−−−−−Input ports Data Type−−−−−−−−−−−−−−−−−−−−
22// By rule all the input ports should be wires
```

```
23wire clock ;
24wire reset ;
25wire enable ;
26//−−−−−−−−−−−−−−Output Ports Data Type−−−−−−−−−−−−−−−−−−
27// Output port can be a storage element (reg) or a wire
28reg [3:0] counter_out ;
29
30//−−−−−−−−−−−−−−Code Starts Here−−−−−−−−−−−−−−−−−−−−−−−−−
31// Since this counter is a positive edge trigged one,
32// We trigger the below block with respect to positive
33// edge of the clock.
34always @ (posedge clock)
35begin : COUNTER // Block Name
36 // At every rising edge of clock we check if reset is active
www.asic−world.com MY FIRST PROGRAM IN VERILOG 22

37 // If active, we load the counter output with 4'b0000
38 if (reset == 1'b1) begin
39 counter_out <= #1 4'b0000;
40 end
41 // If enable is active, then we increment the counter
42 else if (enable == 1'b1) begin
43 counter_out <= #1 counter_out + 1;
44 end
45end // End of Block COUNTER
46
47endmodule // End of Module counter
```

Identifiers

Identifiers are names used to give an object, such as a register or a function or a module, a name.so that it can be referenced from other places in a description.

- Identifiers must begin with an alphabetic character or the underscore character (a−z A−Z _)
- Identifiers may contain alphabetic characters, numeric characters, the underscore, and the
  dollar sign (a−z A−Z 0−9 _ $)
- Identifiers can be up to 1024 characters long.

Hierarchical Identifiers

Hierarchical path names are based on the top module identifier followed by module instant identifiers, separated by periods.

This is basically useful, while we want to see the signal inside a lower module or want to force a value on to internal module. Below example shows hows to monitor the value of internal module signal.

Example

```
1//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
2// This is simple adder Program
3// Design Name : adder_hier
4// File Name : adder_hier.v
5// Function : This program shows verilog hier path works
6// Coder : Deepak
7//−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
8`include "addbit.v"
9module adder_hier (
```

```
10result , // Output of the adder
11carry , // Carry output of adder
12r1 , // first input
13r2 , // second input
14ci // carry input
15);
16
17// Input Port Declarations
18input [3:0] r1 ;
19input [3:0] r2 ;
20input ci ;
21
22// Output Port Declarations
23output [3:0] result ;
24output carry ;
25
26// Port Wires
27wire [3:0] r1 ;
28wire [3:0] r2 ;
29wire ci ;
30wire [3:0] result ;
31wire carry ;
32
33// Internal variables
34wire c1 ;
35wire c2 ;
36wire c3 ;
```

```
37
38// Code Starts Here
39addbit u0 (r1[0],r2[0],ci,result[0],c1);
40addbit u1 (r1[1],r2[1],c1,result[1],c2);
41addbit u2 (r1[2],r2[2],c2,result[2],c3);
42addbit u3 (r1[3],r2[3],c3,result[3],carry);
43
44endmodule // End Of Module adder
45
46module tb();
47
48reg [3:0] r1,r2;
49reg ci;
50wire [3:0] result;
51wire carry;
52
53// Drive the inputs
54initial begin
55 r1 = 0;
56 r2 = 0;
57 ci = 0;
58 #10 r1 = 10;
59 #10 r2 = 2;
60 #10 ci = 1;
61#10$display(
"+————————————————————————————————————————————————————————————+" );
62 $finish;
```

```
end

// Connect the lower module
adder_hier U (result,carry,r1,r2,ci);

// Hier demo here
initial begin
$display("+——————————————————————————————————————————————————————————
————+" );
 $display( "| r1 | r2 | ci | u0.sum | u1.sum | u2.sum | u3.sum |" );
$display("+——————————————————————————————————————————————————————————
————+" );
 $monitor( "| %h | %h | %h | %h | %h | %h | %h |" ,
 r1,r2,ci, tb.U.u0.sum, tb.U.u1.sum, tb.U.u2.sum, tb.U.u3.sum);
end

endmodule
+——————————————————————————————————————————————————————————+
| r1 | r2 | ci | u0.sum | u1.sum | u2.sum | u3.sum |
+——————————————————————————————————————————————————————————+
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a | 0 | 0 | 0 | 1 | 0 | 1 |
| a | 2 | 0 | 0 | 0 | 1 | 1 |
| a | 2 | 1 | 1 | 0 | 1 | 1 |
+——————————————————————————————————————————————————————————+
```

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.

Strings

A string is a sequence of characters enclosed by double quotes and all contained on a single line. Strings used as operands in expressions and assignments are treated as a sequence of eight−bit ASCII values, with one eight−bit ASCII value representing one character.

Special Characters in Strings

Character Description

\n-New line character

\t -Tab character

\\ -Backslash (\) character

\" -Double quote (") character

\ddd- A character specified in 1−3 octal digits (0 <= d <= 7)

%% -Percent (%) character

Gate Primitives

The gates have one scalar output and multiple scalar inputs. The 1st terminal in the list of gate terminals is an output and the other terminals are inputs**.**

Gate Description

and N−input AND gate

nand N−input NAND gate

or N−input OR gate

nor N−input NOR gate

xor N−input XOR gate

xnor N−input XNOR gate

Examples

```
1module gates();
2
3wire out0;
4wire out1;
5wire out2;
6reg in1,in2,in3,in4;
7
8not U1(out0,in1);
9and U2(out1,in1,in2,in3,in4);
10xor U3(out2,in1,in2,in3);
11
12initial begin
13 $monitor( "in1 = %b in2 = %b in3 = %b in4 = %b out0 = %b out1 = %b out2 = %b"
,in1,in2,in3,in4,out0,out1,out2);
14 in1 = 0;
15 in2 = 0;
```

```
16 in3 = 0;
17 in4 = 0;
18 #1 in1 = 1;
19 #1 in2 = 1;
20 #1 in3 = 1;
21 #1 in4 = 1;
22 #1 $finish;
23end
24
25endmodule
```

in1 = 0 in2 = 0 in3 = 0 in4 = 0 out0 = 1 out1 = 0 out2 = 0

in1 = 1 in2 = 0 in3 = 0 in4 = 0 out0 = 0 out1 = 0 out2 = 1

in1 = 1 in2 = 1 in3 = 0 in4 = 0 out0 = 0 out1 = 0 out2 = 0

in1 = 1 in2 = 1 in3 = 1 in4 = 0 out0 = 0 out1 = 0 out2 = 1

in1 = 1 in2 = 1 in3 = 1 in4 = 1 out0 = 0 out1 = 1 out2 = 1

Gate and Switch delays

In real circuits , logic gates haves delays associated with them. Verilog provides the mechanism to associate delays with gates.

• Rise, Fall and Turn−off delays.

• Minimal, Typical, and Maximum delays.

Arithmetic Operators

• Binary: +, −, *, /, % (the modulus operator)

• Unary: +, − (This is used to specify the sign)

• Integer division truncates any fractional part

• The result of a modulus operation takes the sign of the first operand

• If any operand bit value is the unknown value x, then the entire result value is x

Register data types are used as unsigned values.

Bit−wise Operators

Bitwise operators perform a bit wise operation on two operands. They take each bit in one operand and perform the operation with the corresponding bit in the other operand**.**

Verilog HDL Abstraction Levels

• Behavioral Models : Higher level of modeling where behavior of logic is modeled.

• RTL Models : Logic is modeled at register level

• Structural Models : Logic is modeled at both register level and gate level.

Procedural blocks and timing controls.

• Delays controls.

• Edge−Sensitive Event controls

• Level−Sensitive Event controls−Wait statements

• Named Events

Task

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 in 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 are 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 modeling 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**.**

Function

A Verilog HDL function is same as task, with very little difference, like function cannot drive more then one output, can not contain delays.

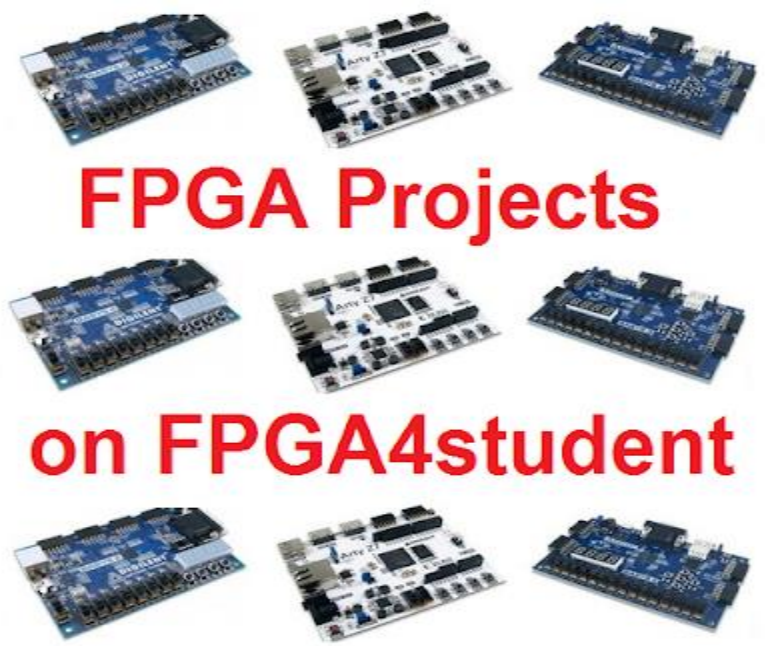
- function are defined in the module in which they are used. it is possible to define function in separate file and use compile directive 'include to include the function in the file which instantiates the task ,can have any number of inputs and but only one output.
- The variables declared within the function are local to that function. The order of declaration within the function defines how the variables passed to the function by the caller are used.

- function can take drive and source function can not include timing delays, like posedge, negedge, # delay. Which means that function should be executed in "zero" time delay.
- function 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 modeling combinational logic.
- function can call other functions, but can not call task.

- SYSTEM TASK AND FUNCTION
- ART OF WRITING TESTBENCHES
- MODELING MEMORIES AND FSM
- PARAMETERIZED MODULES
- VERILOG SYNTHESIS TUTORIAL
- VERILOG PLI TUTORIAL

FPGA Projects

This page presents FPGA projects on fpga4student.com. The first FPGA project helps students understand the basics of FPGAs and how Verilog/ VHDL works on FPGA.



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.

Following are the FPGA projects on fpga4student.com:

1. What is an FPGA? How does FPGA work?
2. Basys 3 FPGA OV7670 Camera
3. How to load text file or image into FPGA
4. Image processing on FPGA using Verilog
5. License Plate Recognition on FPGA
6. Alarm Clock on FPGA using Verilog
7. Digital Clock on FPGA using VHDL
8. Simple Verilog code for debouncing buttons on FPGA
9. Traffic Light Controller on FPGA
10. Car Parking System on FPGA in Verilog

11. VHDL code for comparator on FPGA
12. Verilog code for Multiplier on FPGA
13. N-bit Ring Counter in VHDL on FPGA
14. Verilog implementation of Microcontroller on FPGA
15. Verilog Carry Look Ahead Multiplier on FPGA
16. VHDL Matrix Multiplication on FPGA Xilinx
17. Fixed Point Matrix Multiplication on FPGA using Verilog
18. Verilog Divider on FPGA
19. VHDL code for Microcontroller on FPGA
20. VHDL code for FIR Filter on FPGA
21. Verilog code for Digital logic components on FPGA
22. Delay Timer Implementation on FPGA using Verilog
23. Single-Cycle MIPS processor on FPGA using Verilog
24. FIFO Verilog Implementation on FPGA
25. FIFO VHDL Implementation on FPGA
26. Verilog D Flip Flop on FPGA
27. Comparator Design on FPGA using Verilog
28. D Flip Flop on FPGA using VHDL
29. Full Adder Design on FPGA using Verilog
30. Full Adder Design on FPGA using VHDL
31. Counters on FPGA with Verilog Testbench
32. RISC Processor Design on FPGA using Verilog
33. Verilog test bench for inout ports on FPGA
34. PWM Generator on FPGA using VHDL
35. Tic Tac Toe Game on FPGA using Verilog
36. VHDL code for ALU on FPGA
37. Verilog code for ALU on FPGA

38. Counter design on FPGA with VHDL test bench
39. Pipelined MIPS Processor on FPGA in Verilog (Part-1)
40. Pipelined MIPS Processor on FPGA in Verilog (Part-2)
41. Pipelined MIPS Processor on FPGA in Verilog (Part-3)
42. Verilog Decoder on FPGA
43. Verilog Multiplexers on FPGA
44. N-bit Adder Design on FPGA in Verilog
45. VHDL ALU on FPGA using N-bit Verilog Adder
46. VHDL Shifter on FPGA
47. Lookup Table VHDL example code on FPGA
48. Coprocessor VHDL Implementation on FPGA
49. Affordable Xilinx FPGA boards for beginners
50. Affordable Altera FPGA boards for beginners
51. What is FPGA Programming?
52. Verilog vs VHDL: Explain by Examples
53. VHDL code for clock divider on FPGA
54. Verilog code for clock divider on FPGA
55. How to generate a clock enable signal instead of creating another clock domain
56. VHDL code for debouncing buttons on FPGA
57. VHDL code for Traffic light controller on FPGA
58. Verilog code for PWM Generator on FPGA
59. VHDL code for a simple 2-bit comparator on FPGA
60. VHDL code for a single-port RAM
61. FPGA car Parking System in VHDL using Finite State Machine (FSM)
62. VHDL code for MIPS Processor
63. Verilog code for Sequence Detector using Moore FSM
64. Full VHDL code for Sequence Detector using Moore FSM

65. Seven-Segment LED Display Controller on Basys 3 FPGA
66. VHDL code for Seven-Segment Display on Basys 3 FPGA
67. How to interface a mouse with Basys 3 FPGA
68. Verilog Code for Ripple Carry Adder
69. How to Read Image into FPGA using VHDL

AFTERNOON SESSION DETAILS

Date:	**5-6-2020**	**Name:**	**Kavyashreem**
Course:	**Python programming**	**USN:**	**4al15ec036**
Topic:	**Build a data collector Web App with PostGreSQL and flask**	**Semester & Section:**	**8th A**
Github Repository:	**Kavya**		

Image of session

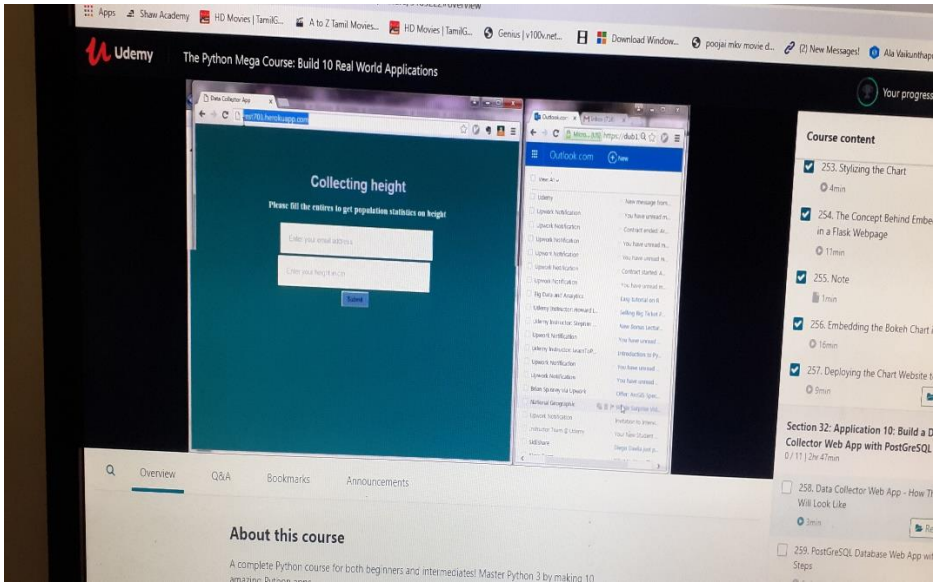


Fig 1: Build a data collector Web App with PostGreSQL and flask

Build a data collector Web App with PostGreSQL and flask

Creating an API or Web application using python has been made easy with Flask. It is a micro web framework written in Python.Here you will create a python server using Flask, create database with PostgreSQL

Install PostgreSQL to local machine

Follow this step if you already haven't installed PostgreSQL on your machine.

Install PostgreSQL in Linux using the command,

sudo apt-get install postgresql postgresql-contrib

Now create a superuser for PostgreSQL

sudo -u postgres createuser --superuser ***name_of_user***

And create a database using created user account

sudo -u ***name_of_user*** createdb ***name_of_database***

You can access created database with created user by,

psql -U ***name_of_user*** -d ***name_of_database***

Create a sample code with Flask to check

For using Flask, first you need to install Flask. (Make sure that you have activated the virtual environment)

pip install Flask

Now create a file named ***app.py*** in ***books_server*** directory and put below code to test Flask before we move into real application development to execute above code run from flask import Flask, request

```
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

@app.route("/name/<name>")
def get_book_name(name):
    return "name : {}".format(name)
```

```
@app.route("/details")
def get_book_details():
    author=request.args.get('author')
    published=request.args.get('published')
    return "Author : {}, Published: {}".format(author,published)

if __name__ == '__main__':
    app.run()
```

```
python app.py
```

Task 5

Implement a verilog module to count number of 0's in a 16 bit number in compiler.

Verilogcode:

```
module num_ones_for(
  input [15:0] A,
  output reg [4:0] ones
```

```
  );
integer i;
always@(A)
begin
ones = 0;  //initializecountvariable.
for(i=0;i<16;i=i+1)   //for all the bits.ones = ones + A[i]; //Add the bit to the count.
end
endmodule
```

Testbench

```
module tb;
   // Inputs
   reg [15:0] A;
   // Outputs
   wire [4:0] ones;
   // Instantiate the Unit Under Test (UUT)
   num_ones_for uut (
      .A(A),
      .ones(ones)
   );
initial begin
      A = 16'hFFFF;   #100;
      A = 16'hF56F;   #100;
      A = 16'h3FFF;   #100;
      A = 16'h0001;   #100;
      A = 16'hF10F;   #100;
      A = 16'h7822;   #100;
```

```
        A = 16'h7ABC;   #100;
    end
endmodule
```