**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **05/06/2020** | **Name:** | **Navya R** |
| **Course:** | **HDL design** | **USN:** | **4AL16EC041** |
| **Topic:** | **Verilog Tutorials and practice programs**  **Building/ Demo projects using FPGA** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **Navya-R** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **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  **Counter test bench:**  1`include "first\_counter.v"  2module first\_counter\_tb();  3// Declare inputs as regs and outputs as wires  4reg clock, reset, enable;  5wire [3:0] counter\_out;  6  7// Initialize all variables  8initial begin  9 $display ( "time\t clk reset enable counter" );  10 $monitor ( "%g\t %b %b %b %b" ,  11 $time, clock, reset, enable, counter\_out);  12 clock = 1; // initial value of  13 reset = 0; // initial value of reset  14 enable = 0; // initial value of enable  15 #5 reset = 1; // Assert the reset  16 #10 reset = 0; // De−assert the reset  17 #5 enable = 1; // Assert enable  18 #100 enable = 0; // De−assert enable  19 #10 $finish; // Terminate simulation  20end  21  22// Clock generator  23always begin  24 #5 clock = ~clock; // Toggle clock every 5 ticks  25end  26  27// Connect DUT to test bench  28first\_counter U\_counter (  29clock,  30reset,  31enable,  32counter\_out  33);  34  35endmodule |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date:** | **05/6/2020** | **Name:** | **Navya R** | |
| **Course:** | **Python** | **USN:** | **4al16ec041** | |
| **Topic:** | **Application 10: Build a Data Collector Web App with PostGreSQL and Flask** | **Semester & Section:** | **8 A** | |
| **AFTERNOON SESSION DETAILS** | | | |
| REPORT Step 1: Create an OpenShift Python Application $ rhc app create todo python-2.7 postgresql-9.2  The command shown above will create an application container for us, called a gear, and setup all of the required SELinux policies and cgroup configuration. Next, it will install all the required software on your gear. It will also install PotsgreSQL 9.2 on your application gear and will create a database with the same name as the application name. OpenShift will also setup a private git repository with some template code, and then clone the repository to your local system. Finally, OpenShift will propagate the DNS to the outside world.  You can view the application details using the command shown below.  $ rhc show-app --app todo  todo @ http://todo-xxxxx.rhcloud.com/569211897923166454  -------------------------------------------------------------------------------  Domain: xxxxx  Created: 11:14 AM  Gears: 1 (defaults to small)  SSH: 569211897923166454@todo-xxxx.rhcloud.com  python-2.7 (Python 2.7)  -----------------------  Gears: Located with postgresql-9.2  postgresql-9.2 (PostgreSQL Database 9.2)  ----------------------------------------  Gears: Located with python-2.7  Connection URL: postgresql://$OPENSHIFT\_POSTGRESQL\_DB\_HOST:$OPENSHIFT\_POSTGRESQL\_DB\_PORT  Database Name: todo  Password: AXtK\_CELQXJK  Username: adminiid3lsl Step 2: Look at Default Template Application The default structure of the template application created by OpenShift is shown below.  todo  wsgi/ Externally exposed wsgi code goes here  wsgi/static/ Public static content gets served here  libs/ Additional libraries  data/ For not-externally exposed wsgi code  setup.py Standard setup.py, specify deps here  app.py.disabled This file may be used instead of Apache mod\_wsgi to run your python web application in a different framework  .openshift/ Location for OpenShift specific files  action\_hooks/ Various scripts to hook into application lifecycle  markers/ Marker files for hot deployment , debugging etc  All the application code will be placed in the wsgi folder and application dependencies will be added to setup.py. Step 3: Adding Flask and Flask-SQLAlchemy Dependencies OpenShift uses [Setuptools](https://pythonhosted.org/setuptools/setuptools.html) which is a collection of enhancements to the Python distutils , that allow developers to more easily build and distribute Python packages, especially ones that have dependencies on other packages. We will add Flask and Flask-SQLAlchemy dependencies to setup.py as shown below.  from setuptools import setup  setup(name='TodoApp',  version='1.0',  description='Todo Application',  author='Shekhar Gulati',  author\_email='',  url='http://www.python.org/sigs/distutils-sig/',  install\_requires=['Flask==0.7.2', 'MarkupSafe' , 'Flask-SQLAlchemy==0.16'],  )  The key attribute in the code shown above is **install\_requires=['Flask==0.7.2', 'MarkupSafe' , 'Flask-SQLAlchemy==0.16']**. The install\_requires attribute is used to specify a list of strings that represent python modules that your app needs. If you need other modules that are not listed you can just add new elements to setup.py. The reason we pegged to a certain version is 1) this prevents the build from checking versions with every git push and 2) it also prevents a build from putting in a version that breaks our code without our knowledge. Step 4: Make Flask Say Hello We will start developing our todo application by creating a new file called todoapp.py in wsgi folders. On windows you can just create a new file named todoapp.py, by right clicking in explorer and saying new text file, then change .txt extension with .py extension.  $ cd wsgi  $ touch todoapp.py  Open your favorite editor and add following lines to it.  from flask import Flask  app = Flask(\_\_name\_\_)  @app.route('/')  @app.route('/hello')  def index():  return "Hello from OpenShift"  if \_\_name\_\_ == '\_\_main\_\_':  app.run()  The code shown above does following :   1. Import the Flask class from the flask module and then create an instance of Flask class. This instance will be our WSGI application. 2. Next we define a route which tells Flask that on root('/') and home('/home') url, it should invoke index() function. The index() function just simply returns "Hello from OpenShift" string which will be rendered by the browser. 3. Finally, if the name of the application module is equal to "\_ \_main\_ \_" then run method is invoked to run the server.   The last change needed to make this "Hello World" application work on OpenShift is to update a file named application which OpenShift created under wsgi folder. Change the content of the file with the one shown below.  #!/usr/bin/python  import os  virtenv = os.environ['OPENSHIFT\_PYTHON\_DIR'] + '/virtenv/'  os.environ['PYTHON\_EGG\_CACHE'] = os.path.join(virtenv, 'lib/python2.7/site-packages')  virtualenv = os.path.join(virtenv, 'bin/activate\_this.py')  try:  execfile(virtualenv, dict(\_\_file\_\_=virtualenv))  except IOError:  pass  from todoapp import app as application  The application file is required by OpenShift and it basically calls the todoapp file that we created earlier.  After all the code changes are done, add the code to the git repository, commit it, and push it to OpenShift gear.  $ git add .  $ git commit -am "hello world from flask"  $ git push  The application will be accessible at http://todo-{domain-name}.rhcloud.com. Replace {domain-name} with your domain name. Step 5: Defining your Model In this blog, we are using [Flask-SQLAlchemy](http://pythonhosted.org/Flask-SQLAlchemy/) which is a Flask extension that adds SQLAlchemy support to our todoapp application. [SQLAlchemy](http://www.sqlalchemy.org/) is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.  Open the todoapp.py and add Todo model class to it as shown below.  from datetime import datetime  from flask import Flask  from flask\_sqlalchemy import SQLAlchemy  app = Flask(\_\_name\_\_)  app.config.from\_pyfile('todoapp.cfg')  db = SQLAlchemy(app)  class Todo(db.Model):  \_\_tablename\_\_ = 'todos'  id = db.Column('todo\_id', db.Integer, primary\_key=True)  title = db.Column(db.String(60))  text = db.Column(db.String)  done = db.Column(db.Boolean)  pub\_date = db.Column(db.DateTime)  def \_\_init\_\_(self, title, text):  self.title = title  self.text = text  self.done = False  self.pub\_date = datetime.utcnow()  @app.route('/')  @app.route('/hello')  def index():  return "Hello from OpenShift"  if \_\_name\_\_ == '\_\_main\_\_':  app.run()  In the code shown above we made the following additions.   1. First we imported SQLAlchemy class from flask\_sqlalchemy module. This is required to work with Flask-SQLAlchemy. 2. Then we created an instance of SQLAlchemy class by passing it application object. The application object was loaded with database configuration which we specified in todoapp.cfg file. We will be creating todoapp.cfg later in this post. 3. Next we defined our Todo model by extending db.Model class and declaring all the Todo model attributes.   Next, create a new file called todoapp.cfg in wsgi folder. It will house all our application configuration. On windows you can just create a new file named todoapp.cfg, by right clicking in explorer and saying new text file, then change .txt extension with .cfg extension.  $ cd wsgi  $ touch todoapp.cfg  Add following lines to todoapp.cfg  import os  SQLALCHEMY\_DATABASE\_URI = os.environ['OPENSHIFT\_POSTGRESQL\_DB\_URL']  SQLALCHEMY\_ECHO = False  SECRET\_KEY = 'secret key'  DEBUG = True  Once you go into production, you will probably want to turn off DEBUG until you run into problems. This will help with performance since you won't be writing as much to files.  Update the application file under wsgi folder so that it creates the database. Add the two lines at the end of file.  from todoapp import \*  db.create\_all() Step 6 : Persisting Todo Items to PostgreSQL Next we will add a route "/new" which will render a form when a user makes a get request to http://todo-{domain-name}.rhcloud.com/new. And when user submit the form using POST method it will write the todo item to database.  Add following lines to todoapp.py.  from flask import Flask, request, flash, url\_for, redirect, render\_template, abort  @app.route('/new', methods=['GET', 'POST'])  def new():  if request.method == 'POST':  todo = Todo(request.form['title'], request.form['text'])  db.session.add(todo)  db.session.commit()  return redirect(url\_for('index'))  return render\_template('new.html')  Flask uses [Jinja2](http://jinja.pocoo.org/docs/) as its templating language. For those of you new to the flask framework, templates basically facilitate the seperation of presentation and processing of your data. Templates are web pages that have mostly static elements and integrate programming logic which produces dynamic content. In the code snippet shown above we are rendering a template called 'new.html'. Create a new folder called templates under wsgi folder and create a new.html in it and add the following lines.  $ cd wsgi  $ mkdir templates  $ cd template  $ touch new.html  Next copy the content shown below in new.html.  {% extends "layout.html" %}  {% block body %}  <form action="" method=post class="form-horizontal">  <h2>Create New Todo</h2>  <div class="control-group">  <div class="controls">  <input type="text" id="title" name="title" class="input-xlarge"  placeholder="Please give title to todo item" value="{{ request.form.title }}"  required>  </div>  </div>    <div class="control-group">  <div class="controls">  <textarea name="text" rows=10 class="input-xlarge" placeholder="describe the todo item" required>{{ request.form.text }}</textarea>  </div>  </div>  <div class="control-group">  <div class="controls">  <button type="submit" class="btn btn-success">Create Todo</button>  <a href="{{ url\_for('index') }}">Back to list</a>  </div>  </div>  </form>  {% endblock %}  The new.html template extends another template called layout.html. The layout.html is where we will define the layout of our web application. Create a new file called layout.html in the templates folder under wsgi and following content. Basically, template inheritance makes it possible to keep certain elements on each page like header, footer, etc.  <!doctype html>  <title>TodoApp -- Store your Todo items</title>  <meta name="viewport" content="initial-scale=1.0, user-scalable=no">  <meta charset="utf-8">  <style>  body {  padding-top: 60px;  padding-bottom: 100px;  }  </style>  <link href="/static/bootstrap.css" rel="stylesheet">  <link href="/static/bootstrap-responsive.css" rel="stylesheet">  <script src="/static/jquery.js"></script>  <script src="/static/bootstrap.js"></script>    <div class="navbar navbar-inverse navbar-fixed-top">  <div class="navbar-inner">  <div class="container">  <button type="button" class="btn btn-navbar" data-toggle="collapse"  data-target=".nav-collapse">  <span class="icon-bar"></span> <span class="icon-bar"></span> <span  class="icon-bar"></span>  </button>  <a class="brand" href="/">TodoApp</a>  <div class="nav-collapse collapse">  <ul class="nav">  <li class="active"><a href="/">Home</a></li>  <li><a href="/new">New Todo</a></li>  </ul>  </div>  <!--/.nav-collapse -->  </div>  </div>  </div>    {%- for category, message in get\_flashed\_messages(with\_categories=true) %}  <p class="flask {{ category }}-flash">{{  "Error: " if category == 'error' }}{{ message }}</p>  {%- endfor %}  <div id="main" class="container">  {% block body %}{% endblock %}  <hr>  <footer id="footer">  <p>Todo App built using Flask, SQLAlchemy, PostgreSQL , and Twitter Bootstrap</p>  <p><a href="https://www.openshift.com/" target="\_blank"><img alt="Powered by OpenShift" src="https://www.openshift.com/sites/default/files/images/powered-transparent-black.png"></a></p>  </footer>  </div> | | | |