**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **05/06/2020** | **Name:** | **RASHMI KB** |
| **Course:** | **HDL design** | **USN:** | **4AL16EC056** |
| **Topic:** | **Verilog Tutorials and practice programs**  **Building/ Demo projects using FPGA** | **Semester & Section:** | **8th B** |
| **Github Repository:** | **rashmikb** |  |  |

|  |
| --- |
| **FORENOON SESSION DETAILS** |
| **Image of session** |
| **Report**  **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.    **Design and tool flow**  **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.   **VERILOG HDL SYNTAX AND SEMANTICS**  **Lexical Conventions**  The basic lexical conventions used by Verilog HDL are similar to those in the C programming  language. Verilog HDL is a case−sensitive language. All keywords are in lowercase.  **White Space**  White space can contain the characters for blanks, tabs, newlines, and form feeds. These characters are ignored except when they serve to separate other tokens. However, blanks and tabs are significant in strings.  White space characters are :  • Blank spaces  • Tabs  • Carriage returns  • New−line  • Form−feeds |

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **05/06/2020** | **Name:** | **Rashmi KB** |
| **Course:** | **Udemy** | **USN:** | **4AL16EC056** |
| **Topic:** | **Python** | **Semester & Section:** | **8th B** |
| **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/ (uuid: 522425cd500446b3ec000294)  -------------------------------------------------------------------------------  Domain: xxxxx  Created: 11:14 AM  Gears: 1 (defaults to small)  Git URL: ssh://522425cd500446b3ec000294@todo-xxxxx.rhcloud.com/~/git/todo.git/  SSH: 522425cd500446b3ec000294@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. | | | |