

COMPLEX DIGITAL SYSTEM VERIFICATION

EEEE-722 Python Verification Project Assignment

Assigned: Monday, January 24, 2022

EEEE-722 Python Verification Project Code Review:	Wednesday, February 02, 2022, In Lab
Python Verification Project Submission Due:	Monday, February 07, 2022, 11:30 PM

Revision History:

V9 R1	Updates for EEEE-722 by: Mark A. Indovina
V8 R3	Updates for EEEE-722 by: Mark A. Indovina
V8 R2	Updates for EEEE-722 by: Mark A. Indovina
V8 R1	Updates for EEEE-722 by: Mark A. Indovina
V7 R1	Updates for EEEE-722 by: Mark A. Indovina
V6 R2	Updates for EEEE-722 by: Mark A. Indovina
V6 R1	Updates for EEEE-722 by: Mark A. Indovina
V5 R1	Updates for EEEE-722 by: Mark A. Indovina
V4 R1	Updates for EEEE-722 by: Mark A. Indovina
V3 R1	Updates for EEEE-722 by: Mark A. Indovina
V2 R1	Updates for EEEE-722 / EEEE-799 by: Mark A. Indovina
V1 R2	Updates for EEEE-722 by: Mark A. Indovina
V1 R1	Original created for EEEE-722 by: Mark A. Indovina

TABLE OF CONTENTS

1. INTRODUCTION	4
2. SOURCE CONTROL	9
3. PROJECT SUBMISSION	11
4. PROJECT GRADING POLICY	12
APPENDIX	14

1. INTRODUCTION

The EEEE-722 Python Verification Project requires development of a program written in the language Python which generates synthesizable Verilog® HDL, and a companion behavioral testbench, for a multistage pipeline register (similar in fashion to a parallel word shift register) based on the following requirements:

- a. Using the Python Getopt package (using Getopt is mandatory), accept multiple command line options as follows:
 - i. --help
 1. (optional) issue help message and program exits
 2. Overrides all other options
 - ii. --param *file name*
 1. (optional) input parameter file
 - iii. --width *nn*
 1. (optional) operand width
 - iv. --stages *nn*
 1. (optional) number of pipeline stages
 - v. --reset *nn*
 1. (optional hex or decimal) register reset value
 - vi. --outfile *file name*
 1. (optional) output file name for generated module
 2. The *file name* naming convention is as follows:
 - a. The *file name* must be of the format *<module_name>.v*
 - b. *<module_name>* will be extracted and used as the module name of the generated module
 - c. The generated testbench filename will be *<module_name>_test.v*
 - d. the testbench module name will be *test*
 - e. the testbench device under test will be *<module_name>*
 - f. the testbench device under test instance will be *top*

Note that the user must pass either "--param" or "--width, --stages, --reset, and --outfile" and any other mandatory options. Passing both "--param" and any other options, or skipping mandatory options is an error for which a message will be issued along with a program usage statement before the program exits. The operand width (1-64 bits wide) and number of pipeline stages (2-128 stages) will be checked for limits, errors will generate a message along with a program usage statement before the program exits.

CONTINUED NEXT PAGE

- b. The optional input parameter file will be formatted as follows (in any order):

```
width = nn ;  
stages = nn ;  
reset = nn ; or reset = 0xnn ;  
outfile = file name ;
```

Missing or unknown options will generate an error along with a program usage statement before the program exits. The operand width (1-64 bits wide) and number of pipeline stages (2-128 stages), and reset value (to make sure the value passed does not exceed the register width) will be checked for limits, errors will generate a message along with a program usage statement before the program exits.

- c. Due to constraints on the Linux system in lab, you will author your program using **Python 2.7.5** or the current version you find by typing **python --version** on the command line.
- d. Use *pydoc* to embed the user manual in the Python source code. Note that Python programs must have a “main” function for *pydoc* to correctly find the embedded user manual.
- e. The generated RTL Verilog® code must compile with *xmverilog* and synthesize with Synopsys Design Compiler.
- i. The code will include full-scan DFT ports as shown in the template.
 - ii. The code will include active high, asynchronous reset.
- f. The generated RTL Verilog® code must be straightforward and precise. This means the RTL code is “simple” in nature and **CANNOT CONTAIN**:
- i. Macros, parameter's or localparam to “parameterize” the RTL code
 - ii. The RTL code cannot include any loop constructs such as “for” or “while”.

CONTINUED NEXT PAGE

- g. The generated Verilog testbench must be self checking
 - i. As noted earlier, your testbench module name must be test and the module under test instance name must be top.
 - ii. Your testbench must include a procedure that checks that your pipeline works properly cycle by cycle based on the number of stages. This includes testing both the reset sequence, and active sequence.
 - iii. Your testbench can (and should) include any loop constructs such as “for” or “while” as necessary for testing. Your testbench must not include any \$stop statements.
 - iv. After a fixed amount of testing, your testbench MUST FINISH with a \$finish statement.
 - 1. The number of test passes must be at least 100 times the number of stages.
 - v. Your generated Verilog testbench must properly back annotate timing. Note this SDF annotate statement in the template testbench:
\$sdf_annotate("sdf/chrpl_tsmc18_scan.sdf", test.top);
Your program should generate this statement with the user configured module name as:
\$sdf_annotate("sdf/<module_name>_tsmc18_scan.sdf", test.top);
- h. Check your generated Verilog RTL and testbench using the chrpl database.

CONTINUED NEXT PAGE

Requirements for your program:

All work will be source controlled with *git* – paths shown are based on the source control requirements defined in section 2.

You will create your program in the directory:

`eeee722/<RIT_UID>_<Semester>/py`

Your program name must be `<RIT_UID>.py`

Given the above requirements, your program will can be found as:

`eeee722/<RIT_UID>_<Semester>/pl/<RIT_UID>.py`

Your program will generate all files in the local directory, `eeee722/<RIT_UID>_<Semester>/py`, ONLY. THERE WILL BE NO HARD CODED PATHS EMBEDDED IN YOUR PROGRAM.

DO NOT SUBMIT SOURCE CODE EDITED ON A WINDOWS PC! ALL CODE MUST BE CREATED ON THE LINUX SYSTEM.

The following cannot be used in variable names or comments:

1. `reg`
2. `register`
3. Anything containing “`reg`”

All registers must be individually declared.

Acceptable coding style:

```
reg [7:0] s1;  
reg [7:0] s2;  
reg [7:0] s2;
```

Not acceptable coding style:

```
reg [7:0] s1,  
s2,  
s3;
```

Note: `RIT_UID` is your RIT computer network logon.

CONTINUED NEXT PAGE

It is strongly recommended that you use a context sensitive editor to author your code. The Linux systems have vim, gvim, and sublime. While vim is a fantastic editor, it will take some practice getting used to the control sequences. Therefore it is recommended that you use sublime.

sublime should already be accessible in your account – just type `sublime` on the command line and the editor will open.

If not accessible, carefully add the following line the very end of your `.bashrc`

```
alias sublime='/classes/eeee720/bin/sublime_text'
```


2. SOURCE CONTROL

To start, you will need to create a work area for the project. Logon to one of the CEDA lab computers or compute servers and create the work area for **EEEE-722 Python Verification** Project as follows:

```
cd  
mkdir eeee722  
cd eeee722
```

Now clone the initial repository.

```
git clone git@kgcoe-git.rit.edu:maiee/<RIT_UID>_<Semester>.git
```

Where: (i) **RIT UID** is your computer log on ID; and (ii) **Semester** is short hand for the current semester (i.e. su2016 is for Summer 2016, f2016 is for Fall 2016).

All work will be source controlled with *git*:

```
git status
```

If the *git status* command does not report “nothing to commit (create/copy files and use “git add” to track)”, something went wrong – ask for assistance.

Throughout the design and development of the **EEEE-722 Python Verification** Project, you must maintain your source code with *git*, in a nutshell you will be using the following commands daily:

- *git status* – to check the status of the local repository
- *git add* – to add new files to the local repository
- *git commit* – to commit your changes to the local repository
- *git show* – show you repository objects
- *git log* – show you commit logs
- *git push* – to push your latest updates back to the master repository
- *git pull* – to pull the latest updates from the master repository

If you get stuck, *git help* will display of list of *git* commands, and *git help <command>* will display a command manual page.

A couple of critical comments on *git*:

1. If you don't know *git*, or feel you are a novice, extensive *git* documentation is available in the *git* website: <http://www.git-scm.com/> Also take time to run through a *git* tutorial: <http://try.github.com/>
2. Many of the *git* commands include a **FORCE (-f)** option. **NEVER, EVER** use the **FORCE (-f)** option during the course of this class. If you get stuck, please consult with the instructor.
3. The recommended procedure for working with the repository database is as follows:
 - a. Start a work session
 - i. Modify your local repository
 - ii. Add files your repository changes by executing *git add*
 - iii. Check in your repository changes by executing *git commit*
 - iv. Modify your local repository
 - v. Check in your repository changes by executing *git commit*
 - b. Work session complete
4. *git commit* forces you to enter comments to document your changes. The comment window is a traditional *vi* window. If you are not comfortable with *vi*, install *gVim* on your personal computer (PC or Mac) and work through the *gVim* tutorial.

3. PROJECT SUBMISSION

The **Python Verification Project** Submission shall be a copy of the Python program along with a brief user manual authored in LyX (submit an exported **PDF** copy) that mimics a Unix man page submitted to mycourses – this can be a copy of your embedded user manual.

Note: upload a copy of your Python program to mycourses with a “.txt” extension added.

4. PROJECT GRADING POLICY

1. Completion of the **EEEE-722 Python Verification** Project assignment by indicated date and time - No Exceptions!
2. On the on-on-one code presentation and review of your **EEEE-722 Python Verification** Project codebase with the instructor: Wednesday, February 02, 2022, In Lab.
3. On your **Python Verification Project** Submission copy of the Python program along with a brief user manual (PDF) submitted to mycourses.
4. On your Python Verification Project Submission database copy, uploaded to the *mycourses* Python Verification Project Submission git repository, due, due: Monday, February 07, 2022, 11:30 PM.

The EEEE-722 Python Verification Project is graded per the syllabus as a separate item with individual components graded as follows:

Table 1 Project Grading

Graded Item	% of Grade	Notes
Python Verification Project Submission	5%	Code Review
Python Verification Project Submission	75%	Python Program: (i) Independent Review By Instructor; (ii) Final Database Submission (GIT)
Python Verification Project Submission	20%	Users Guide
Total:	100%	

APPENDIX

How to get rid of the black background for printing and documentation purposes:

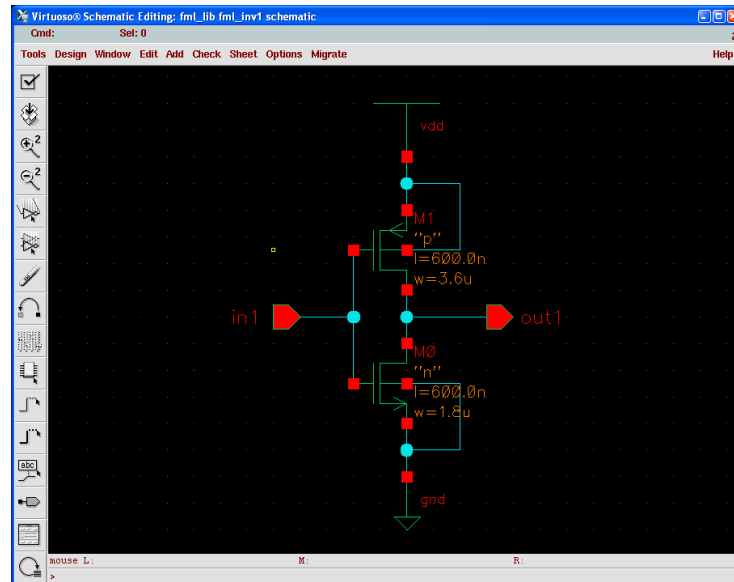


Figure 1 Sample Schematic – Original Black Background

After you copy/paste the window containing the schematic in a word file, using ALT-PRT_SCR, right click on the image and select SHOW PICTURE TOOLBAR.

Once the toolbar opens, towards the right hand side there is a command SELECT TRANSPARENT COLOR. Click on it and then click anywhere on the black background in your window. See the result below.

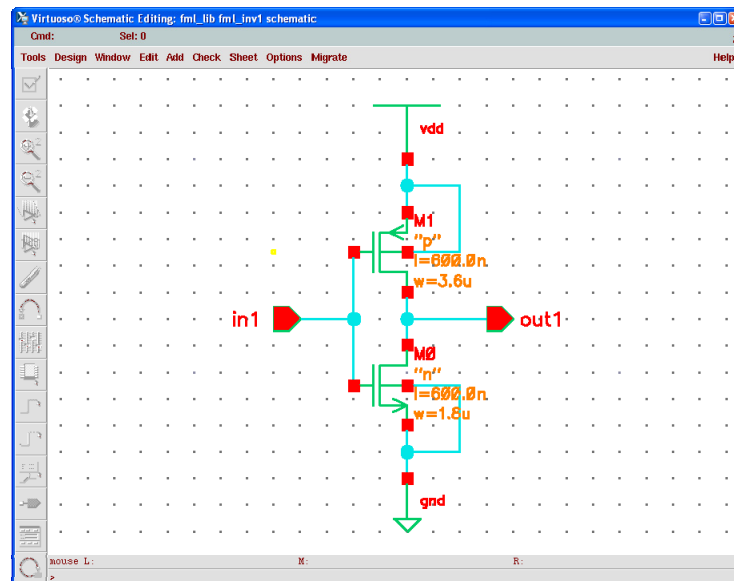


Figure 2 Sample Schematic – Modified White Background