Test Automation Framework for VLSI Design Verification Process using Python

H.Rahul[1], Tirumala Pooja[2], M.Sri Venkat Rami Reddy[3]
UG Students[1, 2] Assistant Professor[3]
Department of Electronics and Communication Engineering
TKR College of Engineering and Technology, Hyderabad, India.

Abstract: *Testing is a very important activity in design and development process. It is to check the quality of the product at various stages of the process. The effective testing methodologies produces high quality product. This Paper deals with a significant and vital issue of testing. This can be conducted manually as well as automated processes. These techniques have their own merits and demerits. In this paper we are proposing in building a Python automation framework. This automation framework is used to support verification and data collection scripts. The scripts control different sets of test equipment in addition to the Device Under Test (DUT) to characterize a specific performance with a specific configuration, to evaluate the correctness of the behavior of the DUT. The specific focus is on documenting our process in building an automation framework using Python.*

Keywords: *Testing, Test Automation, Device Under Test and Python*

I. INTRODUCTION

The ultimate goal of software development is to produce high quality software. Superior quality software has characteristics like low cost, reliable and user satisfactory. Testing is the process of executing a program with the intention of finding errors. This is a crucial & essential activity to discover all the errors early software development process. Emphatic & fruitful testing reduces the system cost. We basically focus on taping out the *Application Specific Integrated Circuit (ASIC)* with potential major defects due to untested features.

Test means program's response to ever possible input. A program should test for every valid & invalid input. Testing activity can be conducted in to two ways: Manual testing & automation testing. Any type of software testing type can be executed both manually as well using an automation tool. This basically limits what we can achieve in such a short period. Here Python comes to the rescue. This paper focuses on documenting the processes that we have on developing an automation framework with Python.

II. PURPOSE & GOALS OF TEST AUTOMATION PROCESS

With this automation process, we are heavily constrained by time and resources. We knew early on that we need something that other engineers can pick up very quickly, so that more people can contribute to the progress of the automation process by various means. We have at that time few hundreds test cases that need to be executed across a number of configurations. We argued that if anyone can use the framework and start automating the test cases, we would have better progress due to more contributions.

The ultimate goal of our automation process was quite simple, It reduces the verification time and potential defects by automating as many test cases as possible. To achieve this, we realized that we needed to develop an easy-to-use framework, so that many people can contribute in the development of the scripts.

The goals of test automation project:

i. Easy-to-reuse framework based on Python Scripting

ii. Optimization of test time by automating the process execution of test-cases

iii. Extensibility of the framework to support scheduled run, control of various equipment, and remote control and execution of the test suit

As the result, we can achieve optimal test time and verification coverage through this process.

III. Specialties of Python

To us, Python was an obvious choice due to the following reasons:

1. **No need of compilation:** No need of compilation means that we can use source code repository system such as CVS or SVN to store our test scripts, ready to run any time. We will not have problem identifying whether the script that we run is the latest and the greatest. No one likes to manage the compilation and the repository for hundreds of small executable. And having one big executable will hinder our development as there is a need for someone to glue the small pieces together, and to enforce safety and quality rules. We can set up development and test machines equally and easily. This doesn't sound like a significant advantage, but to in this project it has great importance. We can set a PC up just by executing several installers; and that machine is ready for rapid debug-fix cycles often necessary. And note also that test equipments are usually attached to specific PCs due to difficulties in moving them around.

2. **Python is easy to read:** We need to get as many people as possible to start writing test scripts. Easy to read means that it is easy to learn from examples and existing scripts. From this project, we learnt that once the momentum started, the scripts were adding up pretty fast.

3. **Python is very flexible:** Python is very flexible, not limiting anyone to specific programming paradigm, e.g. object oriented. Frankly speaking, object oriented paradigm is not easy to pick-up by non- programmer; and forcing object oriented paradigm on test scripting, where most of the operations consist of a) do this, b) and do this after, is just plain silly.

4. **Python is enriched with more built-in functions:** We need to have something better than previous. Python with so many built-in libraries for it really helps. For example, to interface with test equipment, we need to use *VXI plug & play I/O software language* (VISA) protocol [6]; conveniently, there is PyVISA [4] ready for our use. In addition to the above mentioned advantages, scripting in python many more.

IV. Automation Project

In our automation project, we built an automated environment to make it easy to script test and data collections. We need to work with various test equipment, for the purpose of various measurements and verification. We also need to perform various controls on our *Device Under Test* (DUT). Thus our automation project deals with three main components:

1. Equipment control using PyVISA.
2. Connectivity to the existing C++ software to control the DUT.
3. How to make scripting easy.

Our automation project produced an infrastructure that glues these three components. And Python is the key element to each of these three components, and also is the glue that brings the three components together.

4.1. Equipment control using PyVISA

VISA [6] is the interfacing protocol adopted by most of the test equipment vendors. Using VISA protocol, communicating and controlling various test equipment become much simpler as we need not deal with the lower layer protocol. With VISA, we can focus on the control and the results processing.

PyVISA [4], written by TorstenBronger, provides the necessary interfacing of VISA driver with Python. Thanks to TorstenBronger, when we want to write a data collection script in Python, we just need to install a VISA driver[3] and PyVISA. After we connect the equipment (through one of the many supported VISA interfaces: GPIB, USB, LAN, etc.), we can immediately take control of the equipment using Python.

In our project, we need to develop a higher layer abstraction for each test equipment that we use so that it is easy to use even for non-software programmers.

After that, we studied the MSO's display image in two phases. In the first phase with default setting (with persistence OFF), and in the second phase with persistence ON. This is a very simple example of what we can do with PyVISA controlling the test equipment. We can do quite complex computation here. We can basically read out all the measurement data from the scope, and process them point-by-point in Python, or we can use the MSO's functions, e.g., to measure delay/distance between two edges. An example of the saved image is in figure-1.

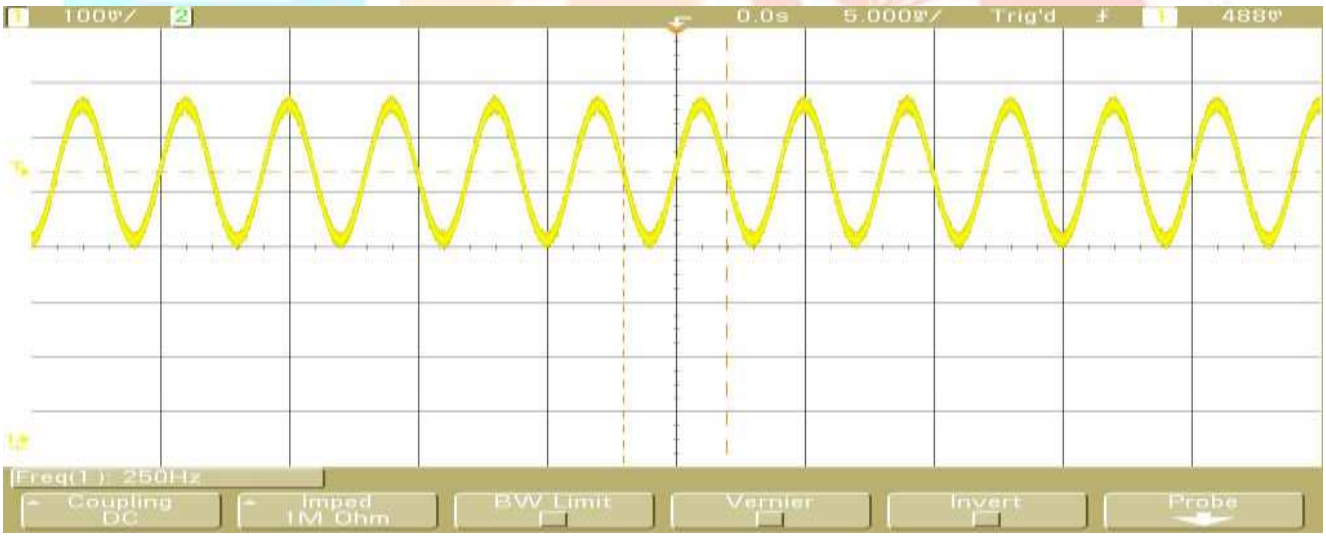

Figure-1: Image of Sine wave Audio captured.

It looks easy; in real-life it couldn't be easier. We have our test engineers, without real programming background, starting to write data collection and test scripts with very little explanations. What they need is only a decent easy-to-use text editor. At this point of time, we have built abstractions for various types of test equipment: Mixed Signal Oscilloscope (MSO), spectrum analyzer, signal generator, digital multi-meter, and DC power supply. I think that you can pretty much imagine what we can do with all these equipment controls. Adds ease of scripting, and we have a very usable platform, capable to fulfill all our test expectations.

4.2. Connectivity to the existing software

Other than controlling the test equipment, we also need to control the Device Under Test. Our existing software is implemented in C++. We need a simple mean to control the C++ software from Python. We decided to use socket for the communication between the C++ software and the Python automation framework. This socket will be used to pass commands and data, supporting the required capabilities of controlling multiple DUTs from one script. In the Python implementation itself, We decided to use the basic socket library rather than the very useful Twisted framework [5]. We have two reasons for this decision:

1. One less installation to care about.
2. Event driven programming is harder to pick up by developers used to procedural programming.

In the hindsight, these reasons do not really matters in our project. One less installer does not matter much. And the implementation of our communication protocol's lower layer (i.e., socket) was so short (less than 200 lines of code) written in Python that it wouldn't matter whether we use socket or Twisted in our project.

4.3. Scripting in Python is easy:

In the early stage, the basic necessity was to control DUT, test equipment and to save results to a file. We developed a number of higher layer abstractions to make this task easy to script.

There are several imports, logging and debugging facilities provided in the framework. In many use cases, the script developer copies from one of the existing templates, which include the main, imports, and output file functionalities, and modify the content accordingly.

From my experience, these facilities are good enough for most people to start working with. Many of the easier scripts can be done by anyone. And the more difficult scripts are developed with assistance from the software developers. Some of we might wonder why not just the software developers write the scripts have. The burden on the developments of test scripts is not only on the scripting itself, but with all the setups, testing, equipment controls, and the understanding of the actual system behavior. And the domains that need to be covered are much wider than just actual software behavior. Thus, it's either that the respective engineer writes a specification for the software developer, or writes the scripts directly themselves with some assistance from the software developer. It is observed that the second approach requires much less time in some environments, as it cuts down misunderstanding on the specification or the expected behavior and the requirements of a more rigid procedure for test case development.

V. Automation Project: Results and Experiences

At the time this paper is written, I am happy to say that the automation framework is extensively used for various uses in the whole of our engineering department. The time saved is just too significant to ignore. I admit that at the beginning many if not most of the engineers are skeptical of how the automation framework can help their work. However, after witnessing how efficient or how much faster that many things can be done using the framework, many people just started to use. And they started to expect more and more measurements or evaluations to be automated.

Just to give a rough idea on the time spent to develop the automation framework that we have. The basic framework supporting DUT controls and some basic libraries to make it

Easy to develop scripts took less than two weeks of one engineer. (Actually, it was quite usable after the first week). The development time for the equipment supports varies depending on the complexities and the number of operations required. However, most of the equipments require less than three days developing for, including example scripts and testing with the equipments themselves.

In my opinion, the two weeks required to build the basic framework is a time well spent. Having the basic framework setup, the results build up pretty fast with more people working on them. The time saved from having this automation framework is substantial. Just to give an example, the current measurements using digital multi-meter take one engineer one week just to collect all the data with the numerous possible configurations. With the automation framework, the same measurements took 24 hours only.

The automation framework also opens up a number of new possibilities. Some measurements that in the past were thought of to be too time-consuming and too tedious to do are now easily done with some (arguably more complex) scripts. However, the benefits of this cannot be measured easily; one can argue that by having these new possibilities, we actually open up more opportunities for better design and engineering.

All of the capabilities of the automation framework described in this paper can be implemented with other tools: C, C++, Visual Basic, .NET, Selenium, etc. However, Python offers vital advantages that are crucial for this automation framework: ease of use, lightweight, doesn't require compilation, and free to use. The readability of a Python script makes it easy for any engineer to learn by examples. The Python will play a key role in the field of Test Automation.

VI. References:

[1] Agilent VISA. Download from Agilent website (Agilent IO LibrariesSuite).

[2] NI VISA. Download fromhttp://www.ni.com/visa/

[3] Notepad++. Download fromhttp://notepad-plus.sourceforge.net/uk/site.htm

[4] PyVISA. Download fromhttp://pyvisa.sourceforge.net/

[5] Twisted networking framework.http://twistedmatrix.com/

[6] VISA Specifications. Download from
http://www.ivifoundation.org/Downloads/Specification .htm

VII. About Authors:

H.Rahul pursuing Bachelor of Technology in Electronics & Communications Engineering from TKR College of Engineering & Technology, Hyderabad, India.

Tirumala Pooja pursuing Bachelor of Technology in Electronics & Communications Engineering in TKR College of Engineering & Technology, Hyderabad, India.

M. Sri Venkat Rami Reddy working as Assistant Professor of ECE in TKR College of Engineering & Technology, Hyderabad, India. He received the B.Tech. degree in Electronics & Communications Engineering from the J.N.T. University, Hyderabad, India, in the year 2004, the M.Tech. degree in Instrumentation & Control Systems from the J.N.T. University, Kakinada, India, in the year 2011, and the M.Tech. degree in VLSI System Design from the J.N.T. University, Hyderabad, India, in the year 2012.

His current research interests include VLSI based Systems Design, Wireless Sensor Networks (WSN) Internet of Things (IoT), Cyber Physical Systems (CPS), Network-On-Chip (NoC), System on Chip (SoC), Low Power VLSI, Electronic Instrumentation. He is a Life Member of the Indian Society for Technical Education (ISTE), New Delhi, India.