2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Increased Regression Efficiency with Jenkins Continuous Integration

"Time is really the only capital that any human being has, and the only thing he can't afford to lose" - Thomas Edison

Thomas Ellis, Mentor Graphics, DVT, Wilsonville, OR, USA (*thomas_ellis@mentor.com*)

Abstract—For all the incredible technological advances to date, no one has found a way to generate additional time. Consequently, there never seems to be enough of it. Since time cannot be created, it is utterly important to ensure that it is spent as wisely as possible. Applying automation to common tasks and identifying problems earlier are just two proven ways to best utilize time during the verification process. Continuous Integration a software practice, coupled with an intelligent regression system, can do precisely that, resulting in a more efficient use of time and resources.

Keywords—jenkins, continuous integration, regression, efficiency, management, functional verification, metrics

I. WHAT IS CONTINUOUS INTEGRATION?

The basic principle behind Continuous Integration (CI) is that the longer a branch of code is checked out, the more it begins to drift away from what is stored in the repository. The more the two diverge, the more complicated it becomes to eventually merge in changes easily. Ultimately leading to what is commonly referred to as "integration hell". To avoid this, and ultimately save engineers time, CI calls for integrating regularly and often (typically daily).

Regular check-ins are of course, only half the equation, you need to be able to verify their changes quickly as well, otherwise many small check ins over several days, is no different than one large check in at weeks end. Commonly, in a Continuous Integration environment, a CI server monitors the source control for check in's, which in turn triggers a CI process (time-based triggers are also common). This process will then build the necessary design files, and run the requisite integration tests. Once complete, the results of the tests are reported back to the user, and assuming everything passed, can now be safely committed to the repository.

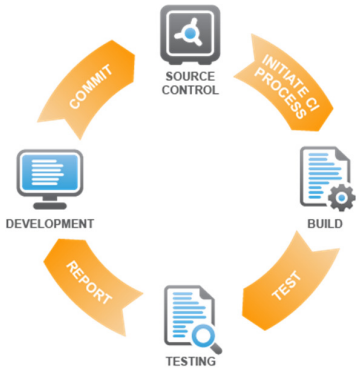

Figure 1. Continuous Integration Flow

By following this model, issues can be caught earlier in the development process, and can be resolved quicker as there is less variance between check ins.

This practice has been used successfully for many years in the software industry, so much so, that it is fairly common place today. However, the idea of Continuous Integration is still fairly new in the realm of hardware verification, so it is difficult to find any published metrics on its usage as it pertains to that space specifically. However, one of the benefits of adopting a more mature technology, is you can avoid encountering some of the pitfalls which plagued early adopters. Since Continuous Integration technology has been used by software teams for some time, you can glean a general idea of both how widespread its usage has become, as well as what technologies have risen to the top.

ZeroTurnaround is a development company, which amongst other things, conducts an annual global survey of Java developers, and produces a report of the tools and technologies being most commonly used by the industry [1]. In 2014, they received responses from nearly 2200 developers covering many topics, one of which was their usage of Continuous Integration Technologies. In that survey, they found that roughly 80% of (or 4 out of 5) developers, reported using Continuous Integration in their teams. A number which itself, showed fairly significant growth, up from 68% the prior year.

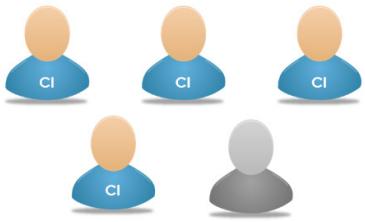

Figure 2. 4 in 5 developers use Continuous Integration

Another interesting aspect of the report, is the breakdown of which Continuous Integration servers were most commonly used. Far and away the most popular server was Jenkins, which was reportedly used by 70% of the developers who claimed to use CI. The second place tool, by comparison, was used by a mere 9% of users. So what is Jenkins, and why is it the favorite CI tool of so many users?

II. MEET JENKINS

Jenkins is a freely available, open-source continuous integration tool (released under the MIT license).

Figure 3. Jenkins CI

A quick background, Jenkins was initially developed by Kohsuke Kawaguchi while he was working at Sun Microsystems in 2004. However, at the time, the project was named Hudson. After its initial release in 2005, it quickly became a favorite open-source build server. In 2010, issues began to arise between the open source community working on Hudson, and Oracle (who had since acquired Sun). Eventually requiring a vote to be called, as to whether to continue development, or to break ties with Oracle and fork the project. Based on an overwhelmingly supportive community vote, Jenkins' was born, created as a fork of Hudson. The majority of those working on, or using Hudson at the time, eventually migrated to Jenkins. Currently there are at least 127,000 active installations of Jenkins (based on the anonymous usage statistics collected by the tool). As for Hudson, remember the ZeroTurnaround study? They found only 8% of users to still be using Hudson.

Apart from being open-source, Jenkins is easy to install and highly configurable via its web interface. While Jenkins offers a lot itself, it is also highly extensible via plug-ins to the tool. At present, it boasts 1350+ plugins from 580+ contributors, to perform a myriad of different tasks, allowing for many third-party tools to leverage the power of Jenkins.

III. INSTALLING JENKINS

Getting Jenkins up and running is a very straight forward, and simple process. The easiest way to run Jenkins, and the method which this paper will demonstrate, is to run Jenkins via it's built in Jetty servlet container. Note, Jenkins does require Java (1.7 or later) be installed on the system. Details on additional installation methods, such as using the built-in packages provided by certain OS's (such as Red Hat and Windows), or running Jenkins through Tomcat, can all found on the Jenkins Wiki [2].

First, you need to download the war file from the Jenkins website [3]. Once downloaded, to start the Jenkins server, you simply type the following command (additionally, we will send the log output to a file):

```
java –jar jenkins.war > Jenkins.log
```

Jenkins is now up and running. To access Jenkins, simply open your web browser and navigate to http://<servername>:8080 (where the <servername> is the name of the machine you are running on). You should see the following page displayed:

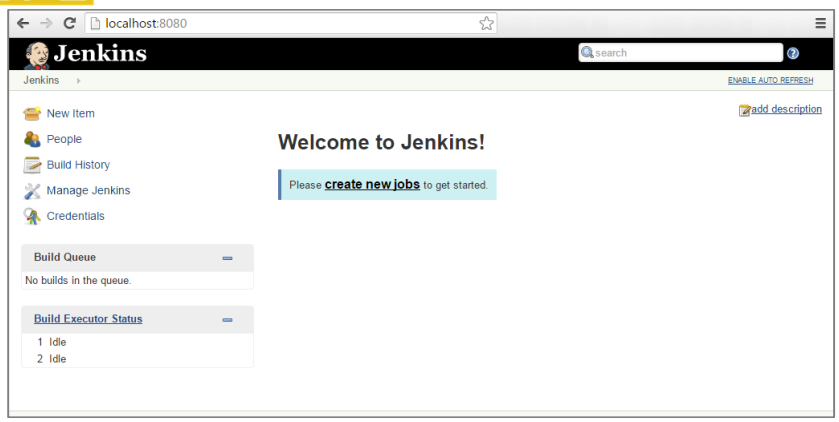

Figure 4. Jenkins Web Interface

As the page says, Welcome to Jenkins! That is all there is to getting Jenkins running. Once running, there are many ways to customize, and administer your Jenkins environment, to your liking. That customization however, is beyond the scope of this paper. For now we will simply focus on creating a simple Jenkins project to launch a regression of verification tests.

IV. RUNNING A REGRESSION IN JENKINS

Let's take a quick look at setting up a project to run a regression in Jenkins. Below is the project configuration page in Jenkins when you create a new freestyle project.

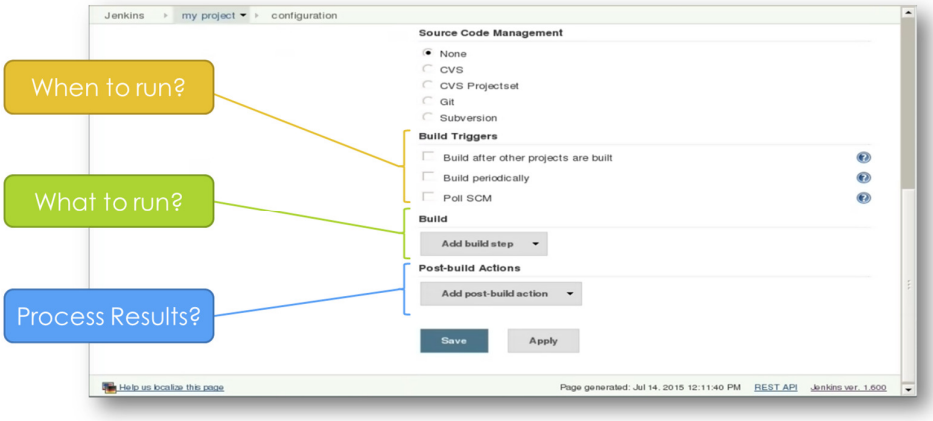

Figure 5. Configuring a Project in Jenkins

Here you can see the basic steps for configuring a project in Jenkins. Tasks in Jenkins are represented by builds. A build could be a complete regression, it could be the running of unit tests, or any other task you may wish Jenkins to automate.

First you specify when to run your tests via a build trigger. A build trigger can be a period of time, a specific time, or you can even have Jenkins monitor your repository for changes, and automatically start a build for you.

Next you tell Jenkins what to do when the trigger occurs. Jenkins is capable of running just about anything you can think to throw at it, which for this example, will be to launch a set of regression tests. Being able to setup builds and triggers is really Jenkins' bread and butter. Additionally, you can manually execute a build anytime you would like by simply clicking a button.

The final configuration step allows you to tell Jenkins to do something additional with the results of the build. Out of the box, Jenkins will give you basic pass/fail information, meaning, if you launch a script to run your regression, it will tell you whether or not your script passed or failed. It will also keep track of the history of your runs,

including information such as the last stable build and changes made between builds (if you are using some form of source code management. However, its lack of the metrics verification engineers are most commonly interested in, makes it feel a bit empty.

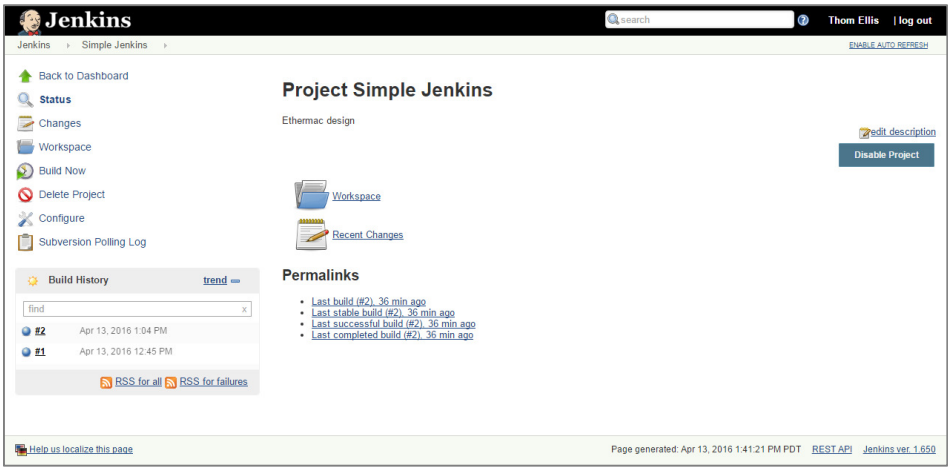

Figure 6. Jenkins Project Page

At a minimum, you would like to see pass/fail results on a per test basis. Additional information on the tests, additional metrics including coverage, etc., as well as trend of this information would also be very valuable. Commonly this is where plug-ins come into play, and as mentioned earlier there are a myriad of different ones available. Most plug-ins rely on the user running simulations using a specific tool, and in turn, it can take those results, and process them to report more relevant data through Jenkins project pages. The first step then, is to find a regression management tool, which also has a Jenkins plug-in which can be used to better report on your regression results. The best option for that, is Questa's Verification Run Manager (VRM).

V. JENKINS AND VRM

On the surface, one might think that Jenkins and VRM are competitive technologies; after all, both tools can build, run and report on regressions. However, in actuality, they are truthfully complementary technologies. Furthermore, by marrying the two technologies together, you can benefit from the strengths of both, and create an extremely powerful solution for building and testing hardware designs.

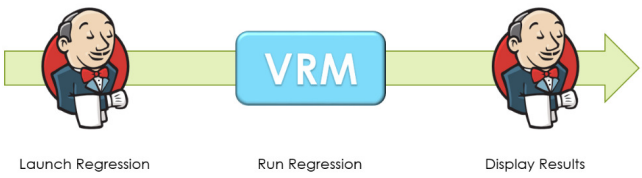

Figure 7. Jenkins and VRM Work Together

While Jenkins is extremely flexible, and can run just about anything, with lots of neat bells and whistles to boot, nothing within the Jenkins core is knowledgeable about hardware verification. In the same way that VRM does not natively monitor code repositories for developer check-ins, concepts like merging System Verilog functional coverage, or recognizing why a UVM testbench failed are not native to Jenkins, in the way that they are at the core of VRM. What you want to do is leverage Jenkins' strengths as a build system to monitor your source repository and allow it to launch builds. Ultimately what it will launch in the build step though, is VRM, which will handle managing the individual verification tasks by integrating with your grid software, automatically collecting and merging the coverage and results, etc. Once the regression is completed, Jenkins can then ask VRM to supply metrics for what was accomplished during the run, and display those results in its web dashboard.

VI. VRM JENKINS PLUG-IN

Finally, to leverage one of key benefits of Jenkins, its high extensibility through plug-ins. To get Jenkins to become more useful with respect to running regressions with VRM, you can leverage the Questa VRM Jenkins plug-in. To do this, you simply install the plug-in through Jenkins plug-in manager by a few simple clicks, and voila! Jenkins now has the ability to understand code and functional coverage, determine where log files reside, monitor host utilization, and many other verification centric tasks.

All that is required is to modify the project to leverage the Jenkins VRM plug-in. To display the regression results, and enable these features, you merely need to add what is called a post-build action (in Jenkins terms), which has Jenkins call the plug-in to process and bring together of the regression results.

Figure 8. VRM Jenkins Plug-In

The setup is very straightforward, all that is required is for you to Jenkins where the regression ran. Additionally, you can optionally select to enable a few other features such as creating HTML reports and publishing a coverage graph to the project page. That's it! Jenkins and VRM will do the rest.

VII. VRM REGRESSION RESULTS IN JENKINS

Another great feature of Jenkins is its web dashboard. Now that it is using the VRM Jenkins Plug-in, you get access to a lot of great information at a glance. There is far too much to show in this paper, but here are a few of the highlights.

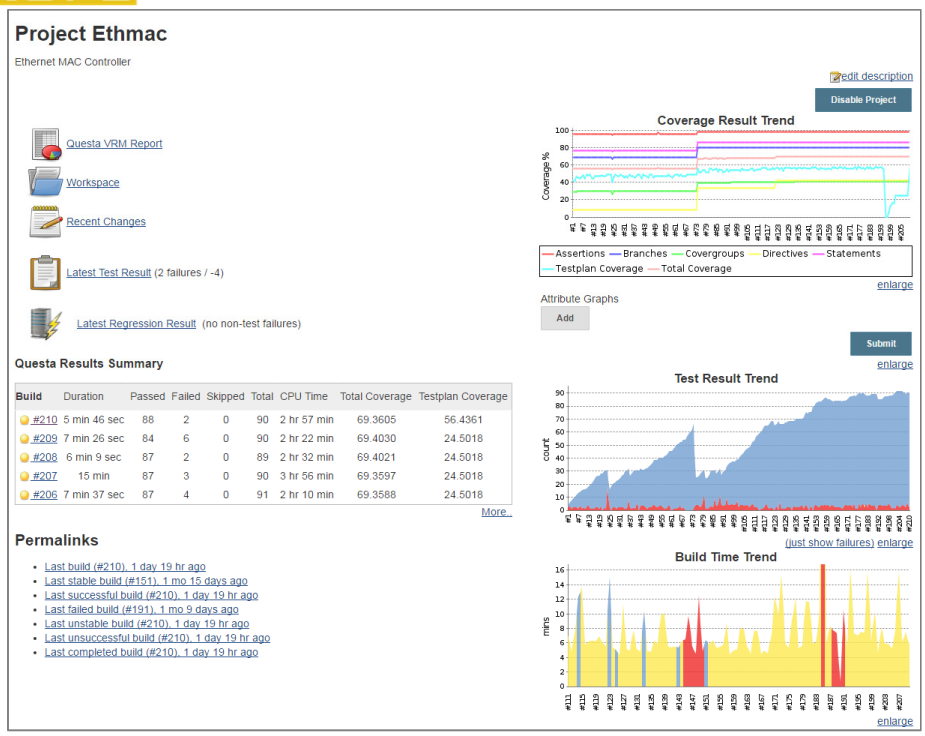

Figure 9. Jenkins Project Dashboard

You will notice big differences on the project page now that the VRM plug-in is in use (compare to figure 5). The main project page has two graphs which shows you a trend of the test results, as well as the coverage results from all your past regression runs. Making it easy to see at a glance how you are progressing towards coverage closure, as well as spot key data points in your regression history.

There is also a summary table which lists the last several regressions (you can expand it by clicking the 'More...' link), including information on their duration, pass/fail statistics and coverage. The 'VRM HTML Report' provides a quick link to the VRM HTML report, as well as the coverage HTML report.

Additionally, you will notice a link to the 'Latest Test Result', which provides you the ability to dive into the individual test results, something you weren't able to do before. In addition, it also gives more detailed data on that particular regression.

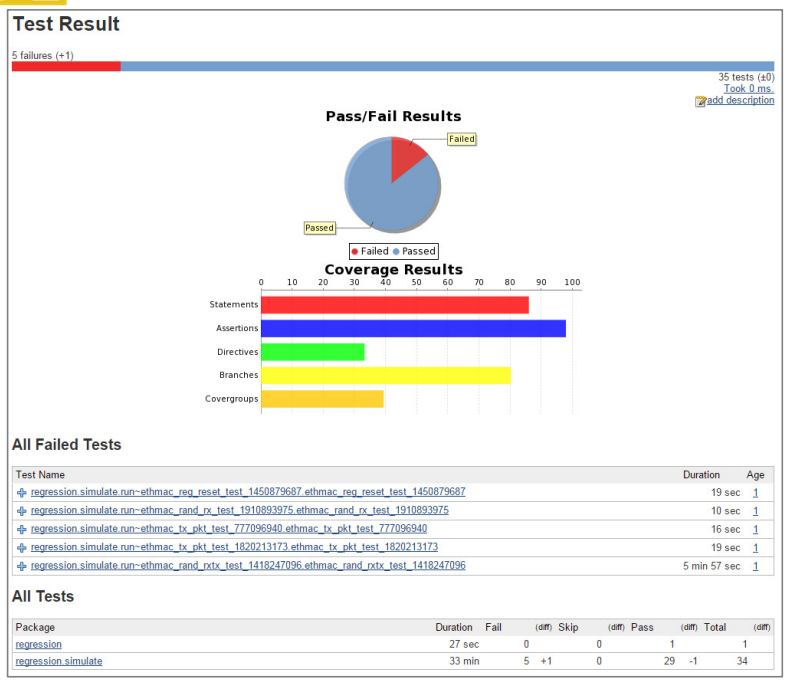

Figure 10. Regression Results

Here, in addition to pass/fail and coverage results, you can also see a list of the specific tests which failed, providing a means for easy high-level inspection. Expanding a given test, will give us both the reason for the failure, as well as the standard output for the test in question.

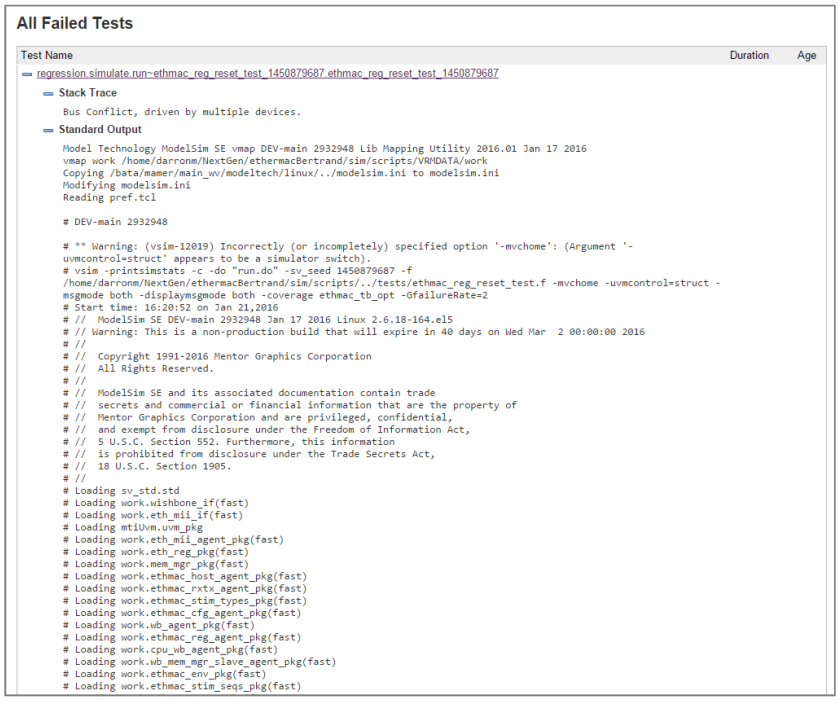

Figure 11. Test Output

You can also dive deeper into the data, and analyze the statistics of a given test, and its individual coverage numbers, additional metrics, etc.

The plug-in leverages the vast amount of data VRM collects from the regression runs, allowing for all sorts of data to be analyzed that would otherwise need to be collected and reported manually. Otherwise difficult questions become easy to answer. Has this test, with this seed ever failed before? What is the host utilization like during a nightly regression? When did coverage drop off?

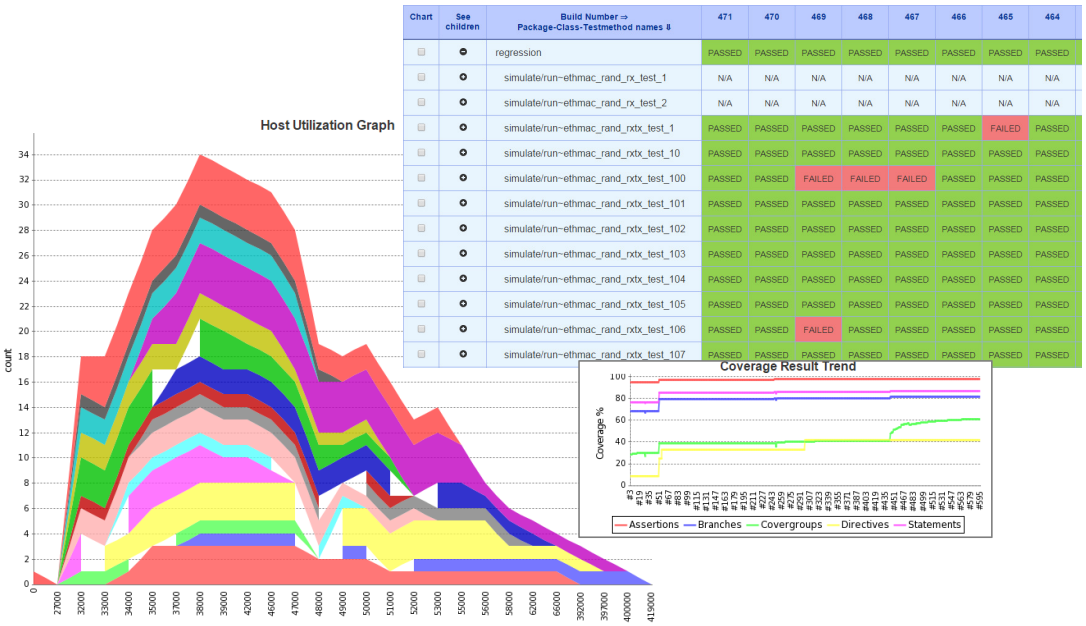

Figure 12. Regression Data Examples

Thanks to the plug-in, you can now get access to the results and history of every regression and every test ran through Jenkins, including a mass of other metrics from memory usage and simulation time, to code and functional coverage, now all available through the Jenkins dashboard.

VIII. SUMMARY

Continuous Integration with Jenkins CI, coupled with Questa Verification Run Manager, provides a powerful automated solution for build and regression management. In automating the regression process and helping to identify problem areas earlier, they allow verification engineers to make more efficient use of the time given even in the tightest of schedules.

REFERENCES

[1] RebelLabs, “Java Tools and Technologies Landscape for 2014”, 2014

[2] Jenkins Wiki, https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

[3] Jenkins, https://jenkins.io