Verification Workflow

Author: **Michael Thompson** mike@openhwgroup.org

1 Introduction

Verification of the Core-V set of RISC-V processor cores will be executed by a distributed team of engineers working for multiple member organizations. This team will be distributed in every sense of the word: we will be working at multiple geographical locations, at multiple member organizations and each organization will have access to its own EDA tools and compute infrastructure. This document captures a workflow that will allow this highly distributed team to work together to achieve its goals in a timely fashion.

A workflow can be conceptualized as two related, yet distinct, aspects. One aspect is centred on how the team manages the code base. We will be using Git as the revision control tool and an OpenHW Group hosted GitHub repository for file revision control, project configuration management and task tracking. The other aspect defines the compute infrastructure and EDA tools used by individual developers on the team. Both are discussed in this document.

1.1 Definition of Terms

Member Company: any company or organization that contributes resources (capital, people, infrastructure, software tools etc.) to the OpenHW Group. Often abbreviated as “MemberCo” in this document.

Developer: an employee of a member company that has been assigned to work on an OpenHW Group project.

CI Regression: Continuous Integration Regression.

2 Code Management

A detailed code management workflow is still under discussion. We will be using something very similar to the git-branching model developed by Vincent Driessen. Thanks to Aimee Sutton of Metrics Technologies for proposing this flow.

3 Compute and Tools

Pre-silicon functional verification of digital designs is a compute intensive task and the OpenHW Group relies on contributions of its member companies to provide these resources. This contribution is currently envisioned to come from one or more of three sources: IBM Cloud virtual machines from IBM, Google Cloud resources via Metrics Technologies and the compute farms of individual member

companies who contribute Developers to the project. The OpenHW Group will manage the cloud-based compute resources provided by IBM. Management of the Google cloud compute resources is already part of the Metrics product offering. Other member companies have extensive in-house compute resources.

The situation is not as clear when considering EDA tool usage. The primary tool in question is an IEEE-1800-2017 compliant SystemVerilog compiler/simulator. There are at least five such tools, all of which are proprietary and only three, *Questa* from Mentor Graphics, *VCS* from Synopsys and *Xcelium* from Cadence, are in wide-spread use by Industry. The simulator from Metrics is called ***dsim***. It is known to be a fully compliant simulator and Metrics has made it available to the OpenHW Group. However, *dsim* is a new entrant and is not yet widely used in Industry, and it is therefore unlikely that any other member companies currently have access to *dsim*.

3.1 Use of Multiple Simulators

It is therefore all but certain that the CORE-V verification effort will be executed on multiple simulators. This creates substantial overhead for three reasons:

1. Each simulator uses a distinct set of command-line options to use the tool. The Makefiles and associated script-ware required to compile code, run tests and merge coverage results are all tool specific.

2. The IEEE-1800 standard does not define how the constraint solver should produce results. This means that testcases with constrained-random stimulus generation may produce different stimulus when run on different simulators, even with the same testbench, RTL and seed. This can make it difficult to replicate failures across simulators.

3. Each simulator supports its own ‘local dialect’ of the IEEE-1800 standard. It is therefore necessary that all Developers produce code that is compatible with all simulators used on the project.

3.2 Mitigation

There is no mitigation for issue #1 above. Each Developer (or team of Developers) will be required to maintain support for the simulator they use in the project’s Makefiles and script-ware. Fortunately, this is a one-time tax to the project which is estimated to be less than a person-week of effort per team.

Issue #2 can be mitigated by requiring the Developer who found a specific issue to validate the fix on their system. This can be written into our process and enforced by the Director of the Verification Task Group (the principal author of this document).

Issue #3 is a real problem and will present an on-going tax to the project. The next sub-section of this document articulates a compute/tool workflow that is designed to minimize the impact of this problem.

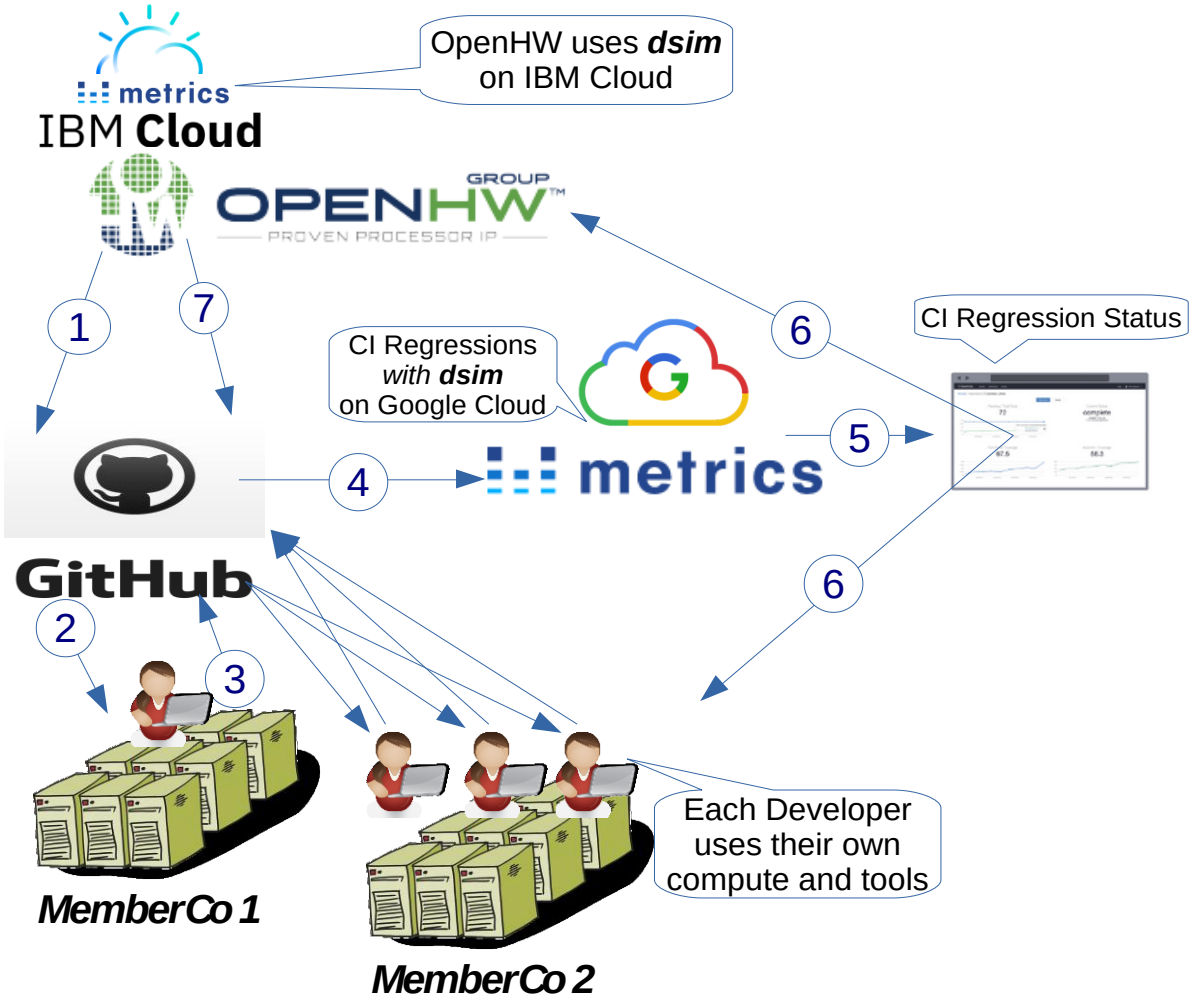

Illustration1:Proposed Workflow #1

3.3 Compute/Tool Workflow

The workflow is shown in Illustration 1. In this example, there are two member companies (MemberCo 1 and MemberCo 2) that are providing one and three Developers to the project respectively. Each MemberCo has its own compute infrastructure and tools that are made available to the project. Note that only Developers from a given MemberCo can access that MemberCo's compute/tools. That is, MemberCo 2's Developers cannot access MemberCo 1's compute/tools. Similarly, employees and contractors of the OpenHW Group cannot access a specific MemberCo's compute/tools unless specifically granted such access.

The enumerated arrows on Illustration 1 show the time-order of specific steps in the workflow:

1. Tasks for a given work-interval are added to the GitHub repo as a set of one or more GitHub issues.
2. A Developer from a MemberCo claims an issue to work on. Other issues are claimed by other Developers from the various MemberCos participating. When a Developer claims an issue, they will also create a branch (dev-branch) from the head of the master branch unless directed otherwise (in the Issue).
3. Developers will code and integrate their feature using their native-simulator switches/scripts/etc. in their corporate environment. This could be Questa, VCS, Xcelium, etc. and OpenHW would not specify any specify simulator be used at this point. Test and debug using the corporate environment will be performed as needed to achieve appropriate quality. When an individual Developer completes the task, they will update their issue, push the updates to their branch and generate a pull request.
4. Any pull request from a Developer branch automatically invokes a CI regression on that branch by the Metrics tool-flow. This regression is run using ***dsim*** and runs on the Google Cloud.
5. Results of the Metrics CI regression are automatically published for review. Metrics CI regression status will be available to all Developers. TODO: document the process by which Metrics CI Regressions are published to MemberCos.
6. Results of CI Regression are reviewed by both OpenHW and Developer(s) at the MemberCo who initiated the regression when they created the pull-request.
7. Based on the results of the CI regression, and perhaps in conjunction with some local testing using ***dsim*** on the IBM Cloud, management at OpenHW will either:

    a) Re-open the issue and re-assign to the original Developer.
    b) Accept the merge request and close the issue.

In this way, the project works towards its goals by clearing off GitHub issues. An issue is never considered closed unless and until it is shown to be working on the appropriate developer branch by ***dsim***.

This workflow is designed to be both palatable to Developers from individual MemberCos while also striving to minimize the impact of the “dialect problem” via continuous integration. It also emphasizes strengths of each tool - Metrics is strong at regression, VCS/Questa/Xcelium have extensive interactive debug capabilities.