Verification Work Flow

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

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 in at geographical locations, working from multiple member organizations and each organization will have access to its own EDA tools and compute infrastructure. This document discusses two workflows 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 configuration management and task tracking. The other aspect defines the compute infrastructure and EDA tools used by individual developers on the team. Both of these will be detailed in this document.

Definition of Terms

Member Company: any company or organization that contributes resources (capital, people, infrastructure, software tools etc.) to the OpenHW Group.

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

Code Management

A detailed code management work-flow is still under discussion. It is a safe bet that 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.

Compute and Tools

Pre-silicon functional verification of digital designs is a compute intensive task and the OpenHW Group leans 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 or 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. So management of compute resource may already be a solved problem.

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. I know of five such tools, all of which are proprietary and only three, *Questa* from Mentor Graphics, *VCS* from Synopsys and *Incisive* 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, so it is therefore unlikley that any other member companies currently have access to dsim.

Use of Multiple Simulators

This opens the possibility 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 generation will 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.

Mitigation

There is no mitigation for issue #1 above. Fortunately, this is a one-time tax to the project which is estimated to be less than a person-week of effort.

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 (me).

Issue #3 is a real problem and will present an on-going tax to the project. The next two sub-sections of this document propose two different compute/tool workflows that attempt to minimize the impact of the this problem.

Compute/Tool Proposed Workflow #1

This workflow is shown in Illustration 1, below. 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 is made available to the project. Note that only Developers from a given MemberCo can access that MemberCo's compute/ tools. These are not expected to be shared outside of MemberCo. 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.

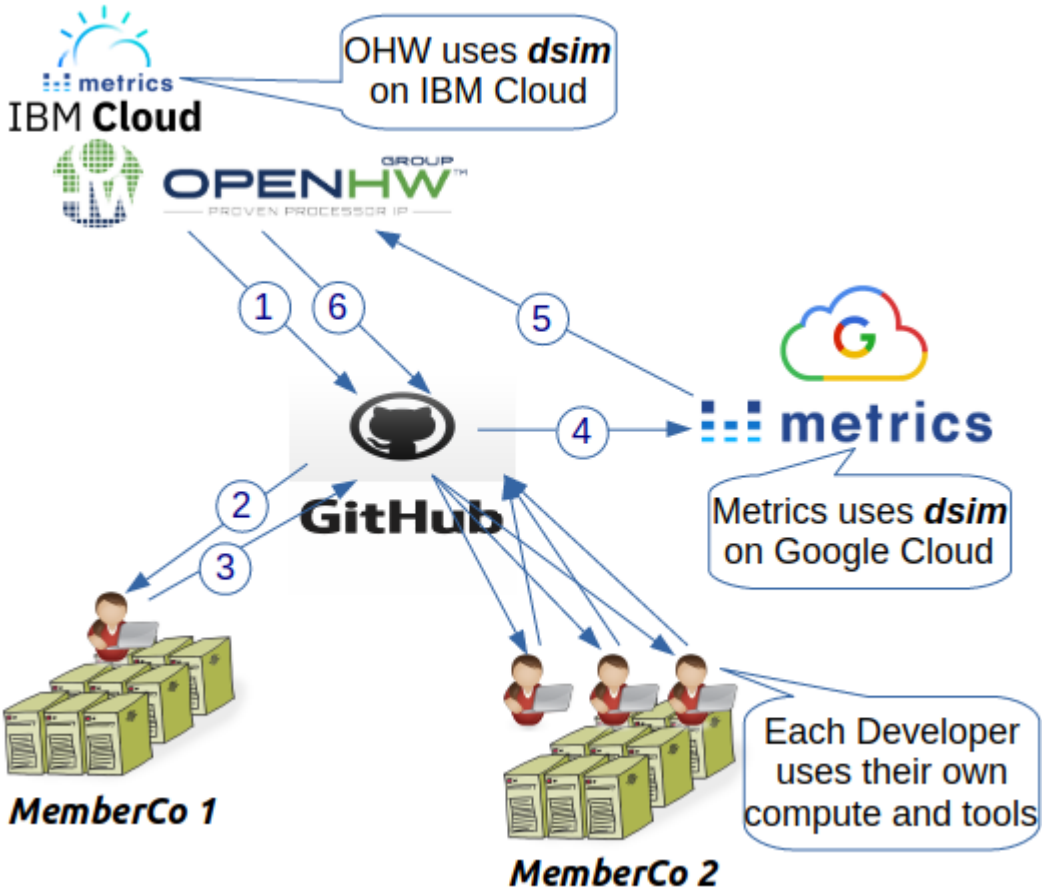

Illustration 1: Proposed Workflow #1

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.
3. Developers perform the work associated with their issue using their company's compute/tool infrastructure. When they believe the work to be done they will submit the updated code to the appropriate git-branch, update their issue and generate a pull request.
4. Any commits to a Developer branch automatically invoke a CI regression 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 regressions will be available to all Developers. Each Developer receives a Metrics account which will allow them to use Metrics' web application to view results, debug, and trigger another regression.
6. 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***.

Compute/Tool Proposed Workflow #2

The second proposed workflow is shown in Illustration 2: Proposed Workflow #2, on page 5 In this flow, all verification coding/testing/debugging is done using dsim on the IBM Cloud and regressions are executed on the Google Cloud by Metrics.

The enumerated workflow is similar to the first:

1. Tasks for a given work-interval are added to the GitHub repo as a set of one or more issues.
2. Developers from MemberCos claim issues to work on.
3. Developers perform the work associated with their issue using a Virtual Machine running on the IBM Cloud. This machine has dsim installed. It is otherwise isolated from all other VMs to limit the possibility of proprietary information from each MemberCo propagating between

Developers.
As in workflow #1, when a Developer believes the work to be done they will submit the updated code to the appropriate git-branch, update their Issue and generate a pull request.

4. Commits automatically invoke a CI regression by the Metrics tool-flow.
5. Pull request are reviewed by OpenHW Group. The decision to accept the request is made as before.

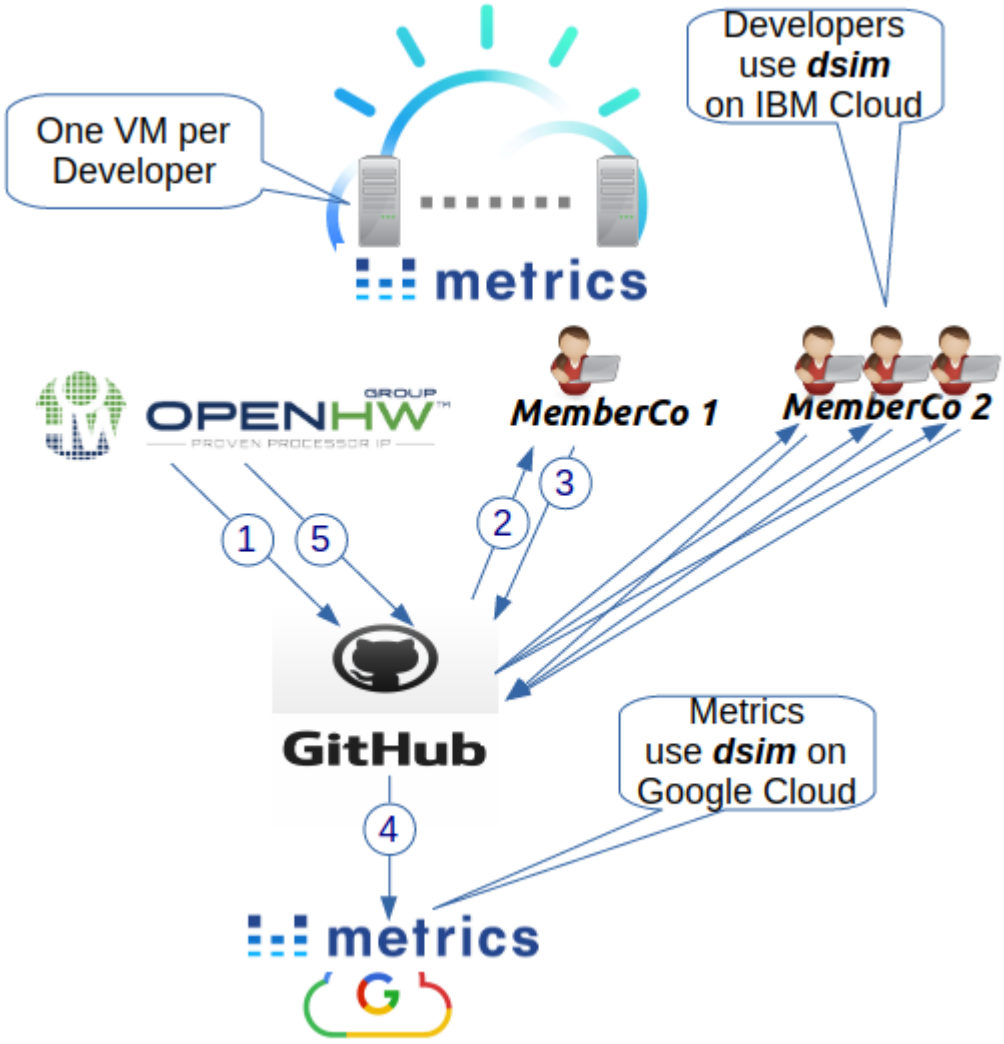

Illustration 2: Proposed Workflow #2

Again, the project works towards its goals by clearing off issues. Note that in this flow, all updates are known to run under ***dsim*** by definition.

Compute/Tool Proposed Workflow #2.5

The following was suggested by Steve Richmond of SiLabs.

1. Tasks for a given work-interval are added to the GitHub repo as a set of one or more issues.
2. Developer clones a new development branch (dev-branch) from the feature branch into their corporate environment.
3. Developer codes and integrates the feature using their "native-simulator" switches/scripts/etc. In their corporate environment (this could be Questa, VCS or Xcelium, etc. but OpenHW would not specify any specify simulator be used at this point.) Debug and regress in corporate environment as needed to achieve appropriate quality.
4. In IBM Cloud VM, pull updates from devel-branch into feature branch.
5. Regress using dsim and commit when "dialect" issues resolved.
6. Issue pull request to feature branch back to OpenHW.

Adding these steps might make the proposal a bit more palatable to Developers from individual companies while also trying to emphasize using the best tool at each step (Metrics is strong at regression, VCS/Questa/Xcelium are "strong" at interactive debug).

Discussion

ToDo: update to properly consider proposal #2.5.

Workflow option #2 eliminates the local dialect problem completely. This is very attractive as this problem will present an on-going tax that must be paid throughout the project. For this reason, this is my preferred workflow. As an added bonus it also solves the script-ware and constraint-solver problems.

Some will point out that option #2 forces users of Core-V produces verified by the OpenHW Group to do the work of writing script-ware and dealing with the dialect problem themselves. This is consistent with the goals of the OpenHW Group, so I do not see this as a barrier. In any case, individual member companies of the OpenHW Group will probably use the same Developer resources to port the script-ware and re-factor the SystemVerilog code to run using their tools. I would expect OpenHW to support this effort.