2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

HW-SW-Coverification as part of CI/CD

Doing continous HW-SW Verification between HW and SW

Alexander Hoffmann, Infineon Technologies AG, München, Germany
(*Alexander.Hoffmann3@infineon.com*)

Ganesh Nair, Infineon Technologies AG, München, Germany (*Ganesh.Nair@infineon.com*)

Nan Ni, Infineon Technologies AG, München, Germany (*Nan.Ni@infineon.com*)

Johannes Grischgl, Infineon Technologies AG, München, Germany
(*Johannes.Grinschgl@infineon.com*)

***Abstract*—Integration between HW and SW leads to disruptions if both flows run in parallel without continuous interaction. This raises the importance of HW-SW Co-design [3] and Co-verification [4] where the concurrent development and verification of hardware and software components of complex electronic systems happens simultaneously and well in collaboration than a discrete approach where only major milestones are the interaction points. Nowadays the software development [2] has become agile and systematic with the continuous integration as well as continuous deployment approaches that aims for early defect discovery, improved productivity, and ultimately contributes to faster release cycles. This paper demonstrates, how HW verification in a simulated environment can get integrated into SW Engineering Flow to have a robust CI/CD flow [1] in place.**

Keywords—continuous integration; hw-sw-coverification; shift-left; risk reduction; automation; co-debugging; code-coverage; data-warehouse; hardware simulation; test result handling;

I. INTRODUCTION

Reducing Time-To-Market requires a significant parallelization between hardware design and software engineering [2]. With decoupled workflows between hardware and software, the development organization faces a risk that the interfaces get out of sync and the deviations only get discovered at integration time usually short before deliveries/releases/tape-outs resulting in costly design changes or inefficient workarounds from the software side.

The software development teams working in a Continuous Integration (CI) / Continuous Delivery/Deployment (CD) [8] environment with a high-grade automation benefit from having tests defined and maintained from the hardware design integrated into their CI environment, so a set of hardware (HW) simulation tests can be run at the software team's discretion. This will give early feedback on necessary interface or behavioral changes or give fast feedback to a software developer on issues his particular change will introduce with the hardware.

This work has been developed in the project VE-VIDES (project label 16ME0243K) which is partly funded within the Research Programme ICT 2020 by the German Federal Ministry of Education and Research (BMBF)

II. Problem Statement

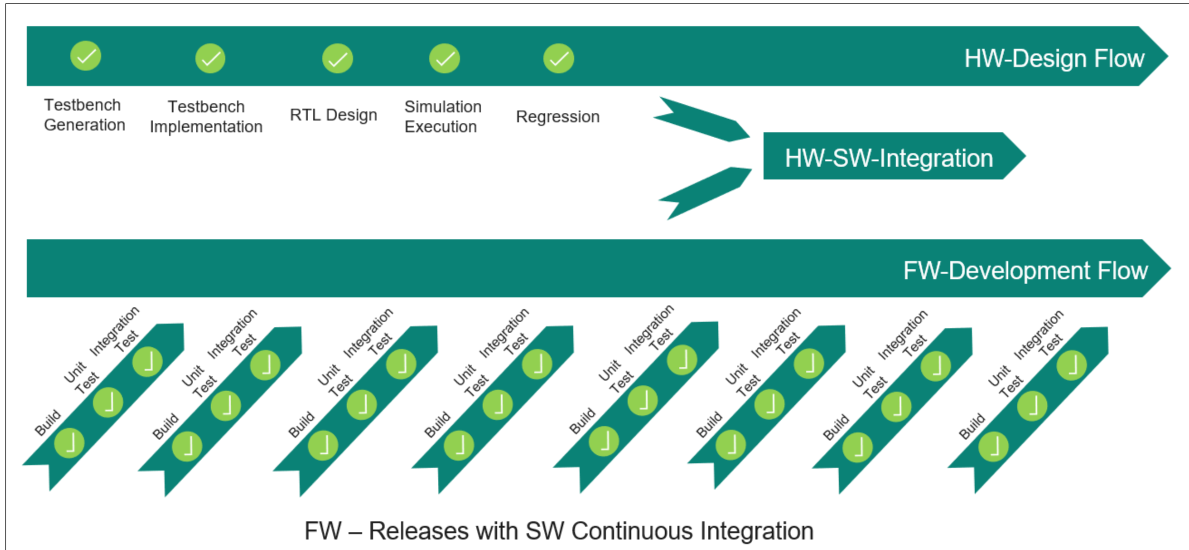

Figure 1 : Typical HW and SW development setups with the integration timelines

A. Problems of a Typical HW and SW Development Flow

Figure 1, a typical setup splits the work on an embedded product between the hardware design and the software development. Apart from integration milestones, these two teams/organizations work independent and use the HW-SW Interface definition as the agreed functionality and behavior to implement for. These integration milestones usually are organized as “bring-ups” or “power-up” campaigns, where hardware and software engineers are co-located and work together to make the hardware work together with the software components. During this time, issues are identified and solved as hotfixes. Thes issues can be bugs, non-matching interfaces, or even incomplete features as both sides worked with a potential incomplete definition, requirements and specifications had changed while hardware or software was developed without a proper communication of these changes.

Without a continuous integration and validation between hardware and software, time spans between a month and half a year are planned, where extreme examples can span multiple years. This results in unplanned delays in the product development planning, allocates costly resources, which are already planned for other engineering work, which propagates the delay into the next generation products. On top, these delays can make bad news especially on products in the consumer market, where brand names can severely be damaged by having a delayed product release or by a product not having all features or being released with many issues.

B. Other Challenges

To implement a continuous integration [10] between the hardware and the software world, many organizations lack the tooling to trigger an immediate test to validate a firmware on an up-to-date hardware or hardware model (in case a real hardware does not exist yet). Usually, the design, development, and verification flows evolved separately from each other, resulting in different base tools for configuration management, change control and test automation. Different tools for implementing tests, checking sources, or design definitions are natural and are no concern due to different tests to be implemented using different PDLs or programming languages.

III. Solution

A. Approach

This paper sketches a potential solution from the software workflow perspective, where agile and continuous integration setups are common and in place. Figure 2, with continuous integration [8], the integrity of the software mainline is assured through automatic testing configured in a so-called pipeline gating [9] merges into the mainline.

With testing a developer's individual change, a failing test result will indicate issues, which would break the mainline.

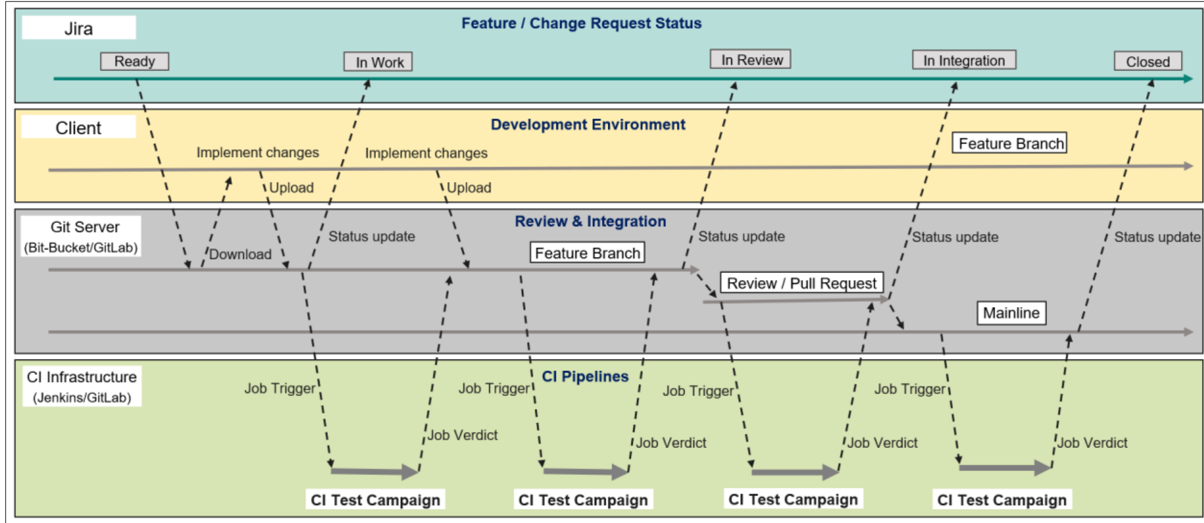

Figure 2 : CI/CD Flow

Software engineers are used to have their individual changes checked by multiple builds, static code checks and test runs through an automation and workflow enforcement, where classic tools are Jira, GitHub [12], GitLab [13] or Jenkins [9]. This means, adding another step, where a particular change with a hardware or hardware model is widely accepted.

B. *Prerequisites*

The expectations from the software development flow are:

- SW developers work in a continuous integration [8] mode forcing them through a build and test campaign as sketched (as described above)
- The CI pipeline contains a build step, which produces a binary for the hardware test to run. This depends on what the hardware test setup requires and might result in an additional build to be set up on the SW continuous integration pipeline.
- There is a common storage where the build step uploads content, from where the hardware test setup can download from

The hardware side needs to provide a test which can be automated, which means:

- Tests need to run without human interaction after being triggered.
- Tests have all configurations, tests scripts and other resources change managed so they can be retrieved before executing the tests.
- Test frameworks offer a way to import content into a test environment, so an executable resulting from the software continuous integration pipeline can get transferred into the hardware testing environment.
- Various frameworks offer a standardized API to set up, run tests, retrieve results and teardown a test setup afterwards.

- Tests produce an explicit result of pass or fail, so this can be used to generate a corresponding verdict in the software test automation.
- Preferably the hardware test framework produces test result information in a standard format (e.g., jUnit)
- Preferably, tests run without a graphical user interface, as automation works more reliable and more efficient in a headless mode.

C. Adaptations for Integration of HW and SW Flows

Achieving the above-mentioned prerequisites might require some adaption effort like,

- Extending test scripts to replace manual inputs by information through variables, parameters or maybe even hardcoding a behavior.
- Storing configuration or scripting in a controlled environment like Git or some other versioning system. In case this is not possible, storing this information in the system used by the software development flow could be considered.
- Putting an analysis scripting in place, so a test verdict can get computed after the test run from tool internal information, a proprietary format or analyzing console log information.
- Putting a conversion from a proprietary format into a standard format in place (e.g., converting tool internal database information into jUnit [6])
- Putting a test framework abstraction layer in place, which offers a glue layer between proprietary test framework APIs, so there is a standard signature to set up and run a test suite, retrieve result and tear down a test setup afterwards.
- Reaching out to a tool vendor or the team providing the test framework to provide an automatable headless way of calling test scripts. Very often this is available as the origins of test framework usually are in command-line triggered testing.

D. Specifics of Hardware Design Flow

The setup used for this paper is using Infineon's established hardware design flow and the established software development flow. The hardware design flow is predominantly Linux-based, offers a change-controlled environment, and integrates several hardware specific test frameworks, where the starting point was an RTL simulation using regression management tool. This tool offers a command line triggered test execution, allows exporting test results from its proprietary database into comma separated values, which can get converted into a standard test result reporting format.

E. Specifics of Software Development Flow

Infineon's software test pipeline is agnostic to the operating systems used for building, testing and other quality checks, but Artifactory [14] from jFrog is in use as a storage infrastructure of binary content, so this was put in place for bridging the gap between the software development environment, and the hardware design flow. This transfer is needed in two directions:

1. Transferring the compiled executable into the hardware environment, so it can be used for running the test suite.
2. Transferring test result information back into the software development environment so it can be used to create a verdict in the software test automation, so a software engineer gets a verdict for his automated test campaign.

F. Bridging the Gaps and Integrating the Flow

As the hardware verification is used to set up long-running test campaigns, a shorter test suite was required, which is an acceptable runtime for software developers. Figure 3, the setup considers the hardware verification team in charge to define and implement the tests, so this team is in control, which tests are strategically important to be run with each change a software developer wants to merge into the software mainline.

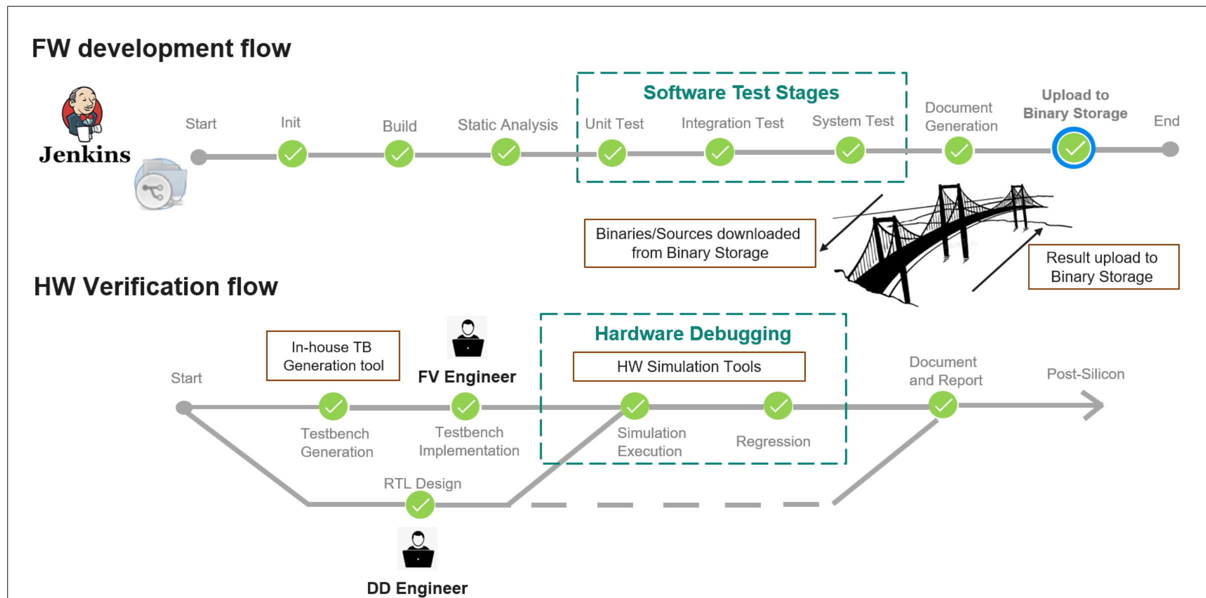

Figure 3 : The CI/CD pipeline corresponding to both HW and SW development together their integration phase.

By textbook, a complete continuous integration [8] run should be finished within 10-20 minutes, which can be a serious challenge to achieve given there'll be a build step before the hardware test can get triggered at all.

There are a couple of thoughts to consider,

- The tests being included into the pre-merge testing should be tests which get broken frequently by software changes, so there is a benefit by having it checked before a software developer can merge his change.
- The same setup can be used to test a different or additional test suite after a change was merged and this test can take longer as an individual developer won't be blocked by the runtime of the test. Putting even more tests into a nightly build can be considered as well. Post-merge tests are the second-best solution as they still get a relative early indication of an issue within the mainline and in case they're run frequently, the change causing the issue can be identified relatively easy (compared to running a test suit only once in a week or less). The downside is, the change is already merged, results in a broken software mainline and resolving the issue is more costly.
- The split between pre-merge and post-merge tests can be dynamic. While the hardware and software team might decide to run less pre-merge tests for while and accept a larger effort due to broken mainlines, while before integration milestones, more tests are run pre-merge resulting in longer tests for each individual change but keeping the mainline working.

G. Test Framework Abstraction Layer for CI Integration

For seamless integration of various test frameworks to the CI infrastructure, a test framework abstraction layer is introduced. This is to standardize on how,

- tests workspaces with corresponding dependencies are set up.

- predefined testcases are executed in the form of test suites in the regression [9].
- results are collected and finally fed back into the dashboarding of the software environment.

Therefore, a simulation framework specific glue layer inheriting the abstract procedures of test framework abstraction layer was inevitable.

H. Setup Workspace

The setup procedure takes care of preparing the workspace with,

- The relevant binary packages are downloaded from the corresponding binary storage location.
- Each binary file is extracted to its specific target location in the workspace where the hardware simulation is referencing them in runtime.

The hardware flow specifics are shadowed away from the software development flow, so the software pipeline automation does not require knowledge, on environment variables, commands or other specifics needed to setup and prepare the execution of a test suite. The usage of Artifactory [14] is part of the setup step to make sure, the software pipeline has a standardized way to build a firmware executable and to upload it into a standard location while the test framework abstraction layer knows about the hardware test specifics and can do a proper setup.

I. Test Execution

The execution is running the test suite. In the best case, it is the same series of commands as a manual hardware verification engineer would run. In the setup in place, the hardware setup uses a compute cluster for parallelization, so the hardware abstraction layer takes proper care, the execution step waits for all parallel jobs to get finished.

J. Results Handling

Retrieving a result after the test execution has been finished is calling Regression Management Tool to generate a comma separated value list of all executed test cases and converting it to jUnit [6]. Additionally, some selected log files are packaged as well to be transferred back into the software development environment so the information can be used to analyze an error in the execution.

K. Teardown Workspace

Teardown is cleaning up the workspace in which the test suite was executed as it is a singular folder structure reused from one test run to the next. So intermediate executable files are reset or removed, and intermediate log outputs are removed to have a clean workspace for the next run.

IV. RESULTS

A. Ease of Use

By using this approach in regular operations,

1. A software developer runs a defined set of HW simulation tests before concluding the changes. This doesn't need the knowledge on how to trigger the tests as it is running on the test automation.
2. In case simulation produces an error, there is the need for a short analysis whether,
    a. The code change broke the tests, which would result in an issue once integration starts.
    b. The test unveiled an incompatibility, where the HW-SW interface needs to be adapted. This change can be done at this stage and will not come up as a surprise during HW-SW integration.
3. The code on the software development mainline is proven to work with the test set defined as well from the software team as well as from the software side, reducing the risk of delaying the integration later.

B. Reduced Interdependency

Already discussing and concluding on certain setups resulted in a better understanding of formerly detached teams. Having a setup like this can lead to a shift of responsibility as well, as the software team might commit to deliver a mainline running a certain set of hardware tests. The strategy and operations on how to achieve this can be left to the software team, leaving the hardware validation team more bandwidth for hardware related issues.

The software team can act independently and decide by itself, how test executions are distributed between pre-merge, post-merge and nightly builds, depending on the feature growth, timeline and resources in the team. To build trust in the team, regular test reports from the nightly builds can be made available to the hardware verification team, so this team has a continuous overview on how the software team is performing with respect to hardware integration and what issues can be foreseen in case an integration with the hardware is planned.

C. Cost and Effort Savings

Wherever HW-SW Co-design is used in productive environments, the issues found at integration time can be reduced dramatically. There are extreme examples where bring up time got reduced from several months to 2 weeks for a baseband product for a Tier 1 smartphone brand after a HW test was introduced into the software automation pipeline with a seven-digit cost reduction.

V. Outlook

There are a couple of aspects that the setup of co-verification between hardware and software can go for in the future.

A. HW-SW Co-debugging

With a setup in place for a batched execution, this can get extended into an interactive debugging on hardware or a hardware model on a debugging environment in a cloud-based setup. The starting point is to take an automated test content with its executable, execution information and offer an easy way to a software developer to single-step through his code running the test setup to identify errors leading to a failed test. This would allow further off-loading of the hardware verification team after software developers gain basic knowledge of the debugging environment.

B. Code Coverage

In the software testing area, code coverage analysis is a standard way to ensure that all code is covered by tests. Providing coverage information is a standard feature, either by test frameworks themselves or through the compiler suite. Making this feature available for testing in a hardware environment gives more information about test cases available in the hardware validation. Information on uncovered firmware code by a test suite can inspire enhancements in test cases, identify dead firmware code or help predicting potential undiscovered bugs, which only surface after running a firmware on the hardware or a hardware model.

C. Data Warehousing and usage of AI/ML Algorithms

With test data being collected in an automated and systematic manner under suitable Data Warehousing solutions [5] and later analyzed using appropriate data mining techniques [7], way better statistics can be captured. This can help to

- identify patterns of frequently or never-failing test cases.
- extending or reducing a standard test suite with respect to relevance of testcases.

Additionally, machine learning algorithms [7] can be applied as a larger data set is available due to automatic runs by software developers to look to behavioral patterns in the test executions. The setup allows for easy storing additional information like code coverage or console logs with the test runs, so machine learning can identify dependencies and patterns not seen otherwise.

VI. CONCLUSION

Continuous Integration in the context of hardware-software co-verification offers a transformative approach to modern system development. By merging hardware and software testing into a unified process, CI allows for early and consistent bug detection, leading to more efficient issue resolution and a significant reduction in later-stage rework. This not only expedites development timelines, but also ensures a consistent quality level, with automated tests providing immediate feedback to developers. Such a collaborative framework encourages a seamless synergy between hardware and software teams, optimizing the use of resources.

Additionally, with CI's traceability and documentation capabilities, there's a clear history of project evolution, enhancing accountability. In essence, integrating CI into hardware-software co-verification can lead to higher product quality, faster time to market, cost savings, and reduced overall project risks.

REFERENCES

[1] Jez Humble, David Farley, “Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation”, Addison Wesley, 2010, ISBN: 978-0321601919

[2] David Farley, “Modern Software Engineering: Doing What Works to Build Better Software Faster”, Addison-Wesley Professional, 2022, ISBN: 978-0137314911

[3] Jeab-Michel Berge, OzLevia and Jacques, “Hardware/Software Co-Design and Co-Verification”, Springer, 2010, ISBN: 978-1441951595

[4] Hason R.Andrews, “Co-verification of Hardware and Software fro ARM SoC Design”, Newnes, 2004, ISBN: 978-0750677301

[5] W.H. Inmon, Derek Strauss, Genia Neushloss, “DW 2.0: The Architecture for the Next Generation of Data Warehousing”, Morgan Kaufmann, 2008, ISBN: 9780080558332

[6] Kent Beck, “JUnit Pocket Guide”, O'Reilly & Associates, 2004, ISBN: 978-0596007430

[7] Ian H. Witten, Eibe Frank, Mark A. Hall, Christopher J. Pal, “Data Mining: Practical Machine Learning Tools and Techniques”, Morgan Kaufmann, 2016, ISBN: 978-0128042915

[8] Mojtaba Shahin , Muhammad Ali Babar , Liming Zhu, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices”, IEEE Access, 2017, Digital Object Identifier 10.1109/ACCESS.2017.2685629

[9] “Martin Fowler: Continuous Integration”, 2006, https://martinfowler.com/articles/continuousIntegration.html

[10] Thomas Ellis , “Increased Regression Efficiency with Jenkins Continuous Integration”, DVCON, Europe, 2014

[11] André Winkelmann, Jason Sprott, and Gordon McGregor, “A Guide To Using Continuous Integration Within The Verification Environment”, DVCON, Europe, 2014

[12] Github, https://docs.github.com/en/get-started/quickstart/github-flow

[13] GitLab, https://docs.gitlab.com/

[14] Jfrog Artifactory, https://jfrog.com/help/r/jfrog-artifactory-documentation