Improving Traceability and Collaboration in HDL Design with GitFlow and CI Pipelines

Renan Carlos Gomes de Farias, João Pedro Buzatti Mendes, Lisandra Manzoni Fontoura, Mateus Beck Rutzig

[1] Federal University of Santa Maria, Brazil
Postgraduate Program in Computer Science (PPGCC)

Abstract. *The complexity of digital hardware design has been increasing with the scale of CMOS technology. Thus, front-end design and verification teams face demands for modularity and traceability to shrink tape-out time and avoid failures on functional verification process. However, the employment of non-standard practices of project management in HDL-based workflows have long been spread in the microelectronics community. This work proposes a version control and continuous integration pipeline methodologies based on Gitflow aiming to improve collaboration between front-end and verification teams focusing on ASIC designs. By employing such an approach in a blueMacaw Microcontroller project, which resulted in a 22 nm tape-out, we achieved a highly effective workflow that ensured early error detection, traceability and strong team integration.*
Keywords. *GitFlow, HDL design, ASIC front-end, version control, continuous integration.*

1. Introduction

As digital hardware designs have been growing in complexity and scale, front-end design and verification teams face increasing demands for modularity and traceability. Project management is mandatory to shrink tape-out time and avoid failures on functional verification process. Such an argument is not a novelty: in the 1980s, researchers advocated for structured design methodologies to manage increasing circuit complexity [Bryant 1986, Clarke et al. 1986]. However, the employment of non-standard practices of project management in HDL-based workflows have long been spread in the microelectronics community.

Several works in HDL-based project management have been contributing to improve the project management on HDL-based designs. Mead-Conway methodology [Conway 2012] and Gajski's Y-chart abstraction [Gajski and Kuhn 1983] established foundational concepts like design hierarchy, modularity, and separation of concerns that remain essential to modern digital design flows. Despite these advances, practical design environments often lack version control and repeatable validation strategies. This gap is particularly evident between front-end and verification teams, where Project Specification Document (PSD), RTL code and testbenches evolve independently and asynchronously. As a result, integration errors, undocumented changes, and limited design traceability are common.

In recent years, researchers have increasingly adapted software engineering principles such as distributed version control, peer-reviewed development, and CI/CD pipelines

to hardware design. For example, Biesuz et al. [Biesuz et al. 2021] introduced Hog, a Git-based framework for HDL repositories with traceable bitstream generation. Stirling et al. [Stirling et al. 2022] demonstrated how DevOps inspired workflows can benefit open hardware projects, and Glein et al. [Glein et al. 2019] showed how CI pipelines can automate verification for FPGA firmware development. Other works have emphasized the role of automation and platform-based design in managing system complexity [Sangiovanni-Vincentelli and Martin 2001].

This work goes a step further by proposing a version control and continuous integration pipeline methodologies based on Gitflow, an already established software development pattern, aiming to improve collaboration between front-end and verification teams focusing on ASIC designs. The workflow introduces structured branching strategies aligned with design and verification activities, supported by a centralized GitLab CI pipeline. It enables traceability across features, simulation results, and releases, while enforcing peer review and reproducibility at each integration step without disrupting existing toolchains and design languages.

2. Motivation for Version Control and CI employment on Front-End and Verification Teams

2.1. RTL Coding

Digital circuits play a central role in ASIC projects, such as System-on-Chip (SoC) development, where digital IC designers typically work in parallel on different Intellectual Properties (IPs), each organized within its own subsystem. The adoption of version control and CI improves maintainability and consistency across the entire design by tracking and testing hardware modifications before integrating them into the SoC, errors can be identified and corrected earlier, while new faults are prevented from propagating to higher levels of the system.

2.2. Functional Verification

Functional verification ensures that the design intent is faithfully preserved during implementation [Piziali 2007]. In ASIC projects that rely heavily on newly developed IPs, this process becomes even more critical, as nearly half of the project time is typically spent on debugging [Foster 2024]. The absence of automated CI can result in design errors and inconsistencies reaching the verification team without prior filtering by systematic pipeline simulations. In addition, the lack of proper version control can lead to mismatches between the latest RTL developed by front-end designers and the version under verification. Together, the absence of version control and CI tends to increase debugging time and delay the overall development flow.

2.3. Documentation and Design Traceability

HDL descriptions require efficient documentation to properly capture design intent, features, and possible constraints. Consequently, CI pipelines and version control strategies support RTL design and verification by automatically generating specification documents and tracking the exact version of the most updated documentation through unique commit hashes. These mechanisms link the verification team with the most up-to-date documentation by associating files with unique commit hashes, fostering inter-team collaboration guaranteeing traceability throughout the development flow.

2.4. Tape-out Management

Versioning hardware descriptions not only supports integration between the front-end and verification teams but also benefits back-end professionals responsible for combining digital and analog modules during the tape-out process. By tagging specific versions, it is possible to determine whether a description has already been verified and is ready to be transformed into silicon. Moreover, version control also plays a critical role in the bring-up phase, as it enables the engineers responsible for post-silicon validation to know exactly which version is being tested and what behavior to expect based on that release.

3. Background and Related Work

3.1. GitFlow: a version control strategy for improving productivity in software development

Vincent Driessen [Driessen 2010] proposed GitFlow, a structured and robust branching strategy that enables developers to collaborate simultaneously on the same project by defining five types of branches, each associated with a specific task:

- **main:** contains the stable version of the application.
- **develop:** contains the pre-stable version with the new features that are being tested.
- **feature:** used for the development of new functionalities.
- **release:** prepares a new stable version.
- **hotfix:** applies quick corrections to issues in *main* branch.

3.2. Version Control in Hardware Development

While version control is a long-established practice in software development, its application in hardware projects has historically been limited. Hardware Description Languages, such as Verilog and VHDL, are often managed in isolated, tool-dependent environments, making it difficult to maintain clean revision histories or collaborate efficiently across teams.

Biesuz et al. [Biesuz et al. 2021] introduced *Hog* (HDL on Git), a framework designed to bring modern version control workflows to hardware development. Hog enforces a structured repository layout for HDL on FPGA projects and integrates Git-based tracking into the build and simulation process. A key contribution of this work is the automatic embedding of the Git commit hash into the generated bitstream, ensuring that each FPGA image is uniquely linked to the corresponding version of the source code.

Stirling et al. [Stirling et al. 2022] proposed *HardOps*, a methodology for adapting DevOps workflows to open hardware projects. Although the implementation domain is neither FPGA nor ASIC, their study emphasized the limitations of traditional Product Data Management (PDM) systems and advocated for the distributed version control using Git on hardware projects, such as those involving mechanical and electronic components. Through a case study involving a modular open-source microscope, they demonstrated how decentralized Git repositories, coupled with automated workflows, can empower geographically distributed design teams and accelerate hardware development cycles.

3.3. CI/CD Automation in Digital Design

The adaptation of CI/CD to digital design flows aims to replicate the benefits seen in software development: early error detection, reproducible builds, and streamlined collaboration. However, the implementation of CI/CD in hardware presents distinct challenges, such as long synthesis times, complex toolchains, and licensing constraints.

Glein et al. [Glein et al. 2019] described the implementation of a CI pipeline for FPGA firmware development in the CMS experiment. Their pipeline, built around GitLab CI, automated tasks such as synthesis, elaboration, and simulation, enabling rapid feedback after each commit. The authors reported improved stability and fewer integration failures, attributing these gains to the early detection of issues and the systematic enforcement of testing procedures.

In a follow-up effort, Biesuz et al. [Biesuz et al. 2023] extended their Hog framework to include a complete CI/CD setup, named *Hog-CI*. This system provides preconfigured pipelines that perform HDL compilation, simulation, and implementation steps using GitLab or GitHub CI runners. Hog-CI also automates release tagging and artifact generation, ensuring that each firmware build is traceable, reproducible, and verifiable.

Aiming to accelerate the functional verification flow and increase productivity, Ayari and Mikhael [Ayari and Mikhael 2024] proposed the application of continuous integration methodologies to hardware verification, adapting concepts traditionally used in software development. Their work describes a CI pipeline integrated with version control and data visualization tools, which decomposes the HDL design into units, subsystems, and system-level blocks, and enhances error detection by applying tests across different layers.

3.4. Our Contributions

The aforementioned works show a growing adoption of traditional software practices in hardware projects. However, as it can be seen in Table 1, state-of-the-art studies do not address the ASIC flow, CI pipelines and collaboration between front-end design and verification as a unique hardware development management proposal. This work goes step further by proposing a version control and continuous integration pipeline methodologies based on Gitflow aiming to improve collaboration between front-end and verification teams focusing on ASIC designs.

Table 1. Comparison of Related Works

Work	ASIC	CI Pipeline	Inter-team collaboration	
			Front-end	Verification
[Biesuz et al. 2021]	✗	✓	✓	✗
[Stirling et al. 2022]	✗	✓	✗	✗
[Glein et al. 2019]	✗	✓	✓	✗
[Biesuz et al. 2023]	✗	✓	✗	✗
[Ayari and Mikhael 2024]	✓	✓	✗	✓
This work	✓	✓	✓	✓

4. The Proposed GitFlow for ASIC design

Version control and CI pipelines introduce structure, automation, and accountability across the entire HDL front-end flow. These practices reduce integration errors, accelerate verification feedback, and ensure that design artifacts remain reproducible, documented, and aligned with project goals. As the complexity of ASIC designs increases, adopting such methodologies becomes not only beneficial but essential to maintain scalability and maintainability.

GitFlow is a structured branching model and development workflow built on top of the Git version control system. Figure 1 introduces a modified GitFlow strategy tailored to ASIC-based design to meet the specific requirements of front-end and verification teams. The workflow defines specialized branches to separate responsibilities and reduce integration risks:

- **Main**: Contains the latest stable and verified version of the design, suitable for downstream stages such as synthesis or tape-out. Only fully validated code reaches this branch.
- **Develop**: Serves as the integration branch for validated feature blocks. Once a module has passed RTL and gate-level simulation, it is merged into this branch pending full system-level integration.
- **Verification**: Dedicated to the development of testbenches, assertions, coverage analysis, and simulation scripts. It typically branches off from `feature` or `develop` branches and is maintained by the verification team.
- **Feature**: Used for the implementation of new functional blocks or enhancements. Each feature branch is tied to a GitLab issue that defines the scope and requirements of the feature.
- **Bugfix**: Used to isolate and correct functional or structural bugs identified during simulation or review. Bugfix branches also follow issue tracking to ensure traceability.

This branching model encourages early testing, enables traceable collaboration across teams, and supports automated validation through continuous integration pipelines. The separation between design and verification branches ensures that test development can progress independently while maintaining alignment with design iterations.

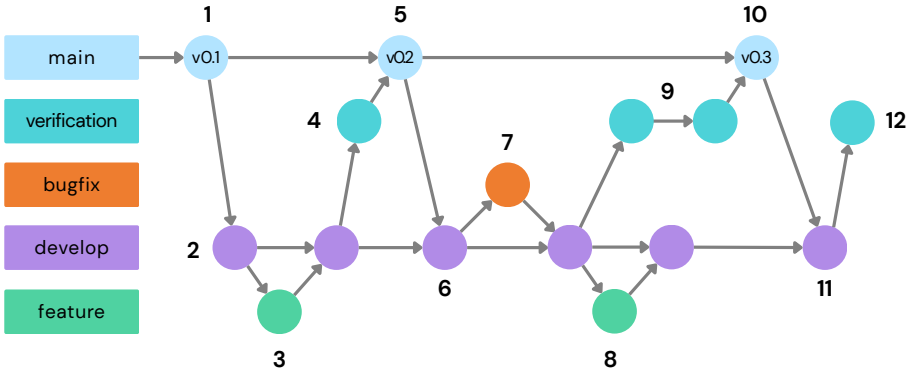

Figure 1. Adapted GitFlow model for HDL-based ASIC front-end design. Each branch is associated with a specific role in the development and verification process.

4.1. Repository Structure

The repository is organized into well-defined directories to separate responsibilities and facilitate reuse:

- `rtl/`: synthesizable RTL modules.
- `tb/`: testbenches, simulation scripts, and test vectors.
- `docs/`: design specifications, interface descriptions, and changelogs.
- `scripts/`: auxiliary scripts for building, simulation, and CI support.

Each directory is associated with a code ownership and review policy, depending on the branch type.

4.2. Issue-Driven Development

All changes to the repository—including new features, bug fixes, and documentation updates—must be linked to a GitLab issue. Each issue defines a clear objective, scope, and acceptance criteria. Contributors create branches from the `develop` base using a consistent naming convention, such as:

- `feature/<issue-id>-<short-description>`
- `bugfix/<issue-id>-<short-description>`
- `doc/<issue-id>-<short-description>`

This policy ensures full traceability from problem definition to implementation and verification.

4.3. Continuous Integration Pipeline

The Continuous Integration (CI) pipeline is composed of three steps: compilation, elaboration, and simulation. Once a merge is requested, the RTL design is first compiled. This step involves performing a syntax check on the HDL files and generating an intermediate netlist. Successful compilation leads to the elaboration phase, which links all modules and produces a complete design hierarchy ready for simulation. Finally, the simulation is executed, and its correct completion represents the last step before approving the merge request. The simulation testbench is prepared by the design team at the beginning of the project and typically consists of a basic test—such as sending a single message or executing a “hello world” program. If any errors occur in the pipeline, the flow is halted and the merge request is blocked. The `main` and `develop` branches contain the tape-out description and the integration of new features, respectively. Thus, a merge request can only occur in these branches if the CI pipeline completes successfully.

4.4. Branching Workflow

Figure 1 shows the proposed branching flow, that works as follows:

- To start a project, an initial tag is created (`1`) on `main`, containing the initial HDL description. The first version of `develop` is then created from this tag (`2`).
- During the front-end development flow, new features (`3`) may be conceived or bugs (`7`) may be fixed. The modified description is merged into `develop` if the CI pipeline completes successfully.

- After performing modifications on the RTL description, front-end designers notify the verification team and create a new `verification` branch (`4`, `12`) from the most recent version of `develop`. The completion of verification results in a merge request to `main` and the creation of a new tag (`5`,`10`), which are only approved if the CI pipeline executes without errors.
- The verification team may report inconsistencies in the design, which can lead to modifications implemented by the front-end designer as new commits in the `verification` branch (`9`).
- The flexibility of the proposed methodology allows new features to be implemented while the design is still under verification (`8`). After the verified RTL description is merged into `main`, a merge is requested to `develop` (`6`,`11`). Any conflicts between these two branches are resolved by the front-end team, and the modifications are only merged if the CI pipeline completes successfully.

Each branch is developed and tested in isolation before being proposed for integration via a Merge Request (MR). To support parallel workflows, the `verification` branch was introduced as a persistent branch dedicated to simulation artifacts and verification code. Verification engineers use this branch to integrate testbenches with the RTL under development, enabling early validation of new features. Simulation environments are updated continuously to reflect RTL changes, and coverage metrics are tracked throughout the development process.

5. Case Study

For evaluating the practical effectiveness of the proposed methodology, the approach has been applied in the development flow of the Brazilian 32-bit RISC-V microcontroller project, named as blueMacaw [blueMacaw 2025] . This project aims at conceiving a national collaboration to build the first Brazilian RISC-V microcontroller. The project involved around 50 participants from five brazilian universities: UFSM, UFMG, UFPR, UFCG and UnB. The blueMacaw is an MCU designed with a 22nm node for low-power IoT applications, providing a complete software suite ensures high productivity. Its set of digital (SPI, I2C, I2S, ADC, SDIO, CAN, UART, JTAG, Watchdog, PWM, Timers) and analog (SigmaDelta and SAR ADCs) peripherals offers compatibility with state-of-the-art sensors. The power management unit includes the most efficient methods, such as transparent clock gating and memory retention mode. Its fully integrated power supply, oscillators, and BLE antenna reduce product packaging.

Branching Workflow Results The GitFlow-based branching strategy, combined with issue tracking and structured merge requests in GitLab, enabled effective coordination across distributed teams. Over two years and five months of development, 501 issues were created in the main digital project repository, of which 89.8% were successfully resolved and closed, while 10.2% remain open and are being handled by the front-end team (Figure 3). Additionally, Figure 4 shows that nearly 90% of the 491 merge requests submitted were successfully merged, whereas 11.6% could not be integrated into the target branch. These metrics demonstrate the effectiveness of the proposed work in ensuring that issues and merge requests are not random hardware design choices, but rather components of a well-structured methodology that guarantees high responsiveness and effective task tracking, since most proposed modifications are completed and integrated into the main modules.

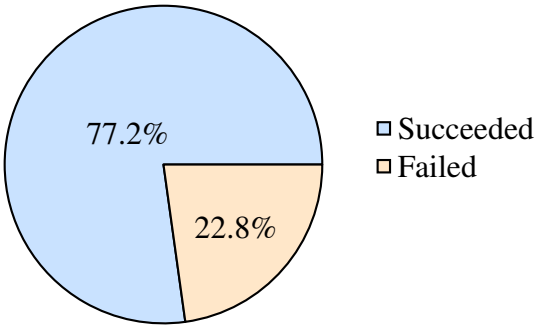

Figure 2. CI pipeline execution

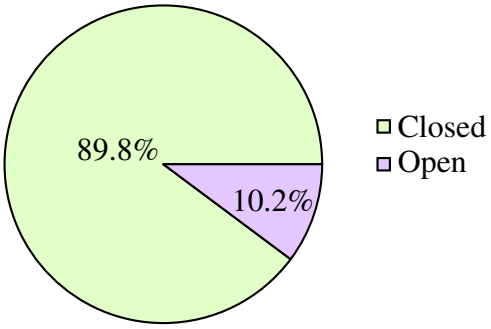

Figure 3. Issues

CI Pipeline Results The Continuous Integration (CI) pipeline was configured using the GitLab CI to orchestrate simulations and verification steps. The simulation executes exhaustive program tests in the microcontroller. If the program completes successfully, the CI job is marked as passed, validating the integrity of the integrated modules. Throughout the development cycle, a total of 434 CI pipelines were executed at the top level, of which 335 (77.2%) completed successfully, as summarized in Figure 2. This metric highlights the central role of CI in bug detection, indicating the ability of the Continuous Integration flow to catch errors before they propagate.

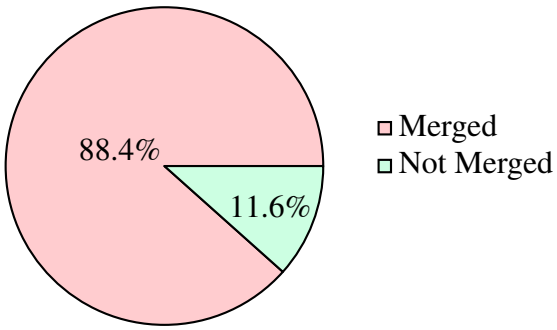

Figure 4. Merge Requests

Inter-Team Collaboration Figure 5 shows that more than half of the verification review requests resulted in a documentation update. In addition to highlighting the communication between the two teams, this also reinforces the critical role of a well-specified design. The chart also reinforces the effectiveness of proposed methodology on avoiding error propagation, as the verification requests resolved with RTL corrections (21.7%)

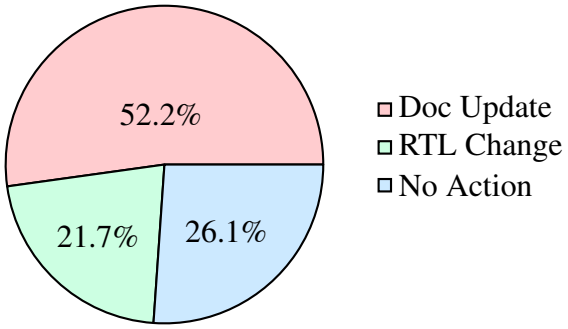

Figure 5. Verification outcomes

are fewer than those addressed without changes (26.1%), which were settled in meetings between both teams to clarify specific design points.

Tape-out The strategy proposed in this work resulted in a 22 nm TSMC manufactured microcontroller which, at the time of writing this paper, is in the packaging process. This achievement represents a milestone in ASIC project management and a significant step forward for the Brazilian semiconductor industry.

6. Conclusion

This work presented a GitFlow model aiming to improve the collaboration of digital front-end and verification teams in ASIC projects. By incorporating disciplined branching strategies, issue-driven tracking, peer review, and centralized continuous integration using GitLab, the proposed methodology shows enhancing traceability, reproducibility, and collaboration in case study where teams are distributed in different Brazilian universities. As future work, we intend to evaluate the scalability of the proposed flow in larger multi-team projects and explore its integration with additional DevOps practices, such as automated release tagging and artifact archiving.

7. Acknowledgment

This paper is supported by the Brazilian Ministry of Science, Technology and Innovations, with funding provided under Law No. 8,248 of October 23, 1991, within the scope of the SOFTEX program, coordinated by Softex and published under Microelectronics 1A, Official Gazette (DOU) No. 01245.000250/2022-53

References

Ayari, A. and Mikhael, K. (2024). Functional verification from chaos to order: Using continuous integration for hardware. In *Proceedings of the Design and Verification Conference (DVCon)*.

Biesuz, N. V., Caballero, R., Cieri, D., Giangiacomi, N., Gonnella, F., Loustau De Linares, G., and Peck, A. (2023). Hog 2023.1: A collaborative management tool to handle git-based hdl repository. In *Workshop on Open-Source Design Automation (OSDA)*.

Biesuz, N. V., Camplani, A., Cieri, D., Giangiacomi, N., Gonnella, F., and Peck, A. (2021). Hog (hdl on git): A collaborative management tool to handle git-based hdl repository. *Journal of Instrumentation*, 16(04):T04006–T04006.

blueMacaw (2025). bluemacaw webpage:. `https://sites.google.com/inf.ufsm.br/bluemacaw/home`. Acessado em: 12 maio 2026.

Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

Clarke, E. M., Emerson, E. A., and Sistla, A. P. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263.

Conway, L. (2012). Reminiscences of the vlsi revolution: How a series of failures triggered a paradigm shift in digital design. *IEEE Solid-State Circuits Magazine*, 4(4):8–31.

Driessen, V. (2010). A successful git branching model. `https://nvie.com/posts/a-successful-git-branching-model/`. Accessed: 2025-09-12.

Foster, H. D. (2024). 2024 wilson research group ic/asic functional verification trend report. White Paper 86424-D3, Siemens Digital Industries Software.

Gajski, D. D. and Kuhn, R. H. (1983). New vlsi tools – guest editors' introduction. In *Computer*, volume 16, pages 11–14.

Glein, R. et al. (2019). Continuous integration of fpga designs for cms. In *Topical Workshop on Electronics for Particle Physics (TWEPP)*. Available via CERN Document Server.

Piziali, A. (2007). *Functional Verification Coverage Measurement and Analysis*. Springer Publishing Company, Incorporated, 1st edition.

Sangiovanni-Vincentelli, A. and Martin, G. (2001). Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of Computers*, 18(6):23–33.

Stirling, J., Bumke, K., Collins, J., Dhokia, V., and Bowman, R. (2022). Hardops: Utilising the software development toolchain for hardware design. *International Journal of Computer Integrated Manufacturing*, 35(12):1297–1309.