

Just TestIt! An SBST Approach To Automate System-Integration Testing

Original

Just TestIt! An SBST Approach To Automate System-Integration Testing / Terzano, T., Giuffrida, L., Sapriza, J., Schiavone, P.D., Maserà, G., Atienza, D., Lavagno, L., Martina, M.. - (2025), pp. 74-77. (22nd ACM International Conference on Computing Frontiers Cagliari (Ita) May 28–30, 2025) [10.1145/3706594.3726980].

Availability:

This version is available at: 11583/3001609 since: 2025-07-07T09:09:36Z

Publisher:

ACM

Published

DOI:10.1145/3706594.3726980

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

ACM postprint/Author's Accepted Manuscript

(Article begins on next page)

Just TestIt!

An SBST Approach To Automate System-Integration Testing

Tommaso Terzano
Politecnico di Torino
Torino, Italy
tommaso.terzano@polito.it

Luigi Giuffrida
Politecnico di Torino
Torino, Italy
luigi.giuffrida@polito.it

Juan Sapriza
École Polytechnique Fédérale de
Lausanne
Lausanne, Switzerland
juan.sapriza@epfl.ch

Pasquale Davide Schiavone
École Polytechnique Fédérale de
Lausanne
Lausanne, Switzerland
davide.schiavone@epfl.ch

Guido Masera
Politecnico di Torino
Torino, Italy
guido.masera@polito.it

David Atienza
École Polytechnique Fédérale de
Lausanne
Lausanne, Switzerland
david.atienza@epfl.ch

Luciano Lavagno
Politecnico di Torino
Torino, Italy
luciano.lavagno@polito.it

Maurizio Martina
Politecnico di Torino
Torino, Italy
maurizio.martina@polito.it

ABSTRACT

This paper introduces *TestIt*, an open-source Python package designed to automate full-system integration testing using a Software-Based Self-Test (SBST) approach. By dynamically generating test vectors and golden references, *TestIt* significantly reduces development time and complexity while supporting both simulation and FPGA environments. Its flexible design positions *TestIt* as a key enabler for the widespread adoption of CI/CD methodologies in open-source RTL development. A case study on the X-HEEP RISC-V microcontroller (MCU), which integrates a custom accelerator, showcases *TestIt*'s ability to detect hardware and software faults that traditional formal methods may overlook. Furthermore, the case study highlights how *TestIt* can be leveraged to characterize system performance with minimal effort. By automating testing on the PYNQ-Z2 FPGA development board, we achieved a 11× speed-up with respect to RTL simulations.

CCS CONCEPTS

• Hardware → Board- and system-level test; Functional verification.

ACM Reference Format:

Tommaso Terzano, Luigi Giuffrida, Juan Sapriza, Pasquale Davide Schiavone, Guido Masera, David Atienza, Luciano Lavagno, and Maurizio Martina. 2025. Just TestIt! An SBST Approach To Automate System-Integration Testing. In *22nd ACM International Conference on Computing Frontiers (CF Companion '25)*, May 28–30, 2025, Cagliari, Italy. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3706594.3726980>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

CF Companion '25, May 28–30, 2025, Cagliari, Italy

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1393-4/2025/05

<https://doi.org/10.1145/3706594.3726980>

1 INTRODUCTION

Verifying the functionality of a digital integrated circuit (IC) is a complex yet critical step in its development. While formal verification methods are highly effective for targeting individual components, their computational complexity increases exponentially, making them impractical for testing large-scale systems, like heterogeneous MCUs. Additionally, these methods are limited to hardware verification, leaving software layers untested.

Integration testing can be introduced as an additional step in the pre-production phase to alleviate formal verification limitations. This approach incrementally validates the entire system by assessing interactions between previously verified components. Moreover, integration testing can be executed on hardware platforms such as FPGAs at operational speed. This allows for the execution of long tests and real-scenario end-to-end applications, which can include interactions with external components (such as external memories, ADCs, DACs, etc.) as well as the whole data-flow digital-processing chain. Once developed, integration tests can also be used for post-production validation to ensure that the final product behaves correctly.

SBST techniques further enhance flexibility while eliminating the need for costly Automated Test Equipment (ATE) [6].

While several solutions have been proposed to optimize formal verification phases, as described in the next section, no open-source tool currently targets integration testing.

This paper presents *TestIt*¹, a Python package that provides a highly flexible SBST-based solution for developing full-system integration tests. *TestIt* supports both simulation and FPGA targets, automating the entire workflow: from generating random datasets to interfacing with the FPGA development board.

2 RELATED WORKS

Automating testing and verification is an increasingly important aspect of the design of modern complex Systems-on-Chip (SoCs). Indeed, numerous frameworks have been developed to automate the generation of test-benches and formal verification environments.

¹GitHub repository URL: <https://github.com/vlsi-lab/TestIt>

However, they primarily focus on unit testing or platform development, leaving system-level integration testing largely unaddressed. Furthermore, existing solutions are unable to comprehensively test interactions between the hardware platform and the software stack, which is essential for validating real-world functionality.

Table 1: Comparison of the analyzed frameworks

Feature	AutoSVA	Cocotb	Renode	TestIt
High-Level Models	×	×	✓	✓
RTL-Level Models	✓	✓	×	✓
System-Level Integration	×	×	✓	✓
Automated Randomized Tests	✓	✓	×	✓
Simulated Testing	✓	✓	✓	✓
FPGA Testing	×	×	×	✓
SW Stack Testing	×	×	×	✓

Renode [5] allows users to assemble virtual SoCs using modular building blocks, including ARM and RISC-V CPUs, as well as various communication buses and interfaces. However, *Renode* operates at a quite high level of abstraction; it lacks support for automated test execution, and it works exclusively in a simulated environment.

AutoSVA [4] automates formal testbench generation for unit-level RTL verification. While reducing development effort compared to UVM, it still requires additional work in RTL design. Its primary focus is formal verification, which, despite optimizations, remains highly time-consuming for large-scale systems integration. Additionally, *AutoSVA* relies on an extra software layer to perform the verification process, currently limited to *SymbiYosys* [7].

Finally, *cocotb* [1] is a testbench environment for verifying RTL designs similar to the UVM approach, but using Python. While *cocotb* can reduce the overhead of test creation, it does not automate the test flow, it doesn't enable software-layer testing, and it's limited to simulated environments.

Table 1 presents a visual comparison of the analyzed frameworks, highlighting how *TestIt* addresses the gaps identified in state-of-the-art alternatives.

3 OVERVIEW

TestIt is a Python package that implements a command-line application for executing comprehensive integration test campaigns using an SBST approach. It leverages software-driven testing to verify both the integration of hardware components and the correct operation of the software stack, from HAL (Hardware Abstraction Layer) drivers to applications.

To ensure sufficient randomness, *TestIt* dynamically generates a unique input dataset for each test iteration, along with a corresponding golden reference dataset. These datasets are written by the tool into a C-code source and header files pair. These files are linked during compilation, allowing the application to validate the

correctness of the System-Under-Test (SUT), without relying on external dependencies.

3.1 Requirements

To function properly, *TestIt* requires a few key components:

- A **Target RTL Project** with a complete and functional development flow, including software compilation, model synthesis, simulation, and debugger support.
- A set of predefined **Make targets** in the project's Makefile, enabling *TestIt* to interact with the existing workflow.
- A **configuration file** that defines the project's structure and test parameters.
- A **Python module**, developed by the test engineers, containing the functions required by *TestIt* to generate the golden datasets.

Makefile Targets. *TestIt* relies on eight predefined Makefile targets that must be present in the RTL project. They provide maximum flexibility, allowing developers to implement each one according to their specific project characteristics.

The required Makefile targets are:

- sw-sim [app]** : Compiles software applications for simulation environments. Accepts an app argument.
- sw-fpga [app]** : Compiles software applications for FPGA development boards. Accepts the same "app" argument.
- sim-build [tool]** : Builds the simulation model, with a "tool" argument.
- sim-run [tool]** : Sets up and runs the simulation, also with a "tool" argument.
- fpga-build [target]** : Builds an FPGA model, which can be implemented as a bitstream. Accepts a "target" argument.
- fpga-load [target]** : Loads the FPGA model onto the FPGA development board. Accepts the same "target" argument.
- gdb-setup** : Sets up the GDB debugger of choice.
- deb-setup** : Configures the preferred debugging tool, such as OpenOCD.

Test Configuration Files. To run *TestIt*, two configuration files are required, which can be generated in the working directory using the command "testit setup", as described in the next paragraph. These files are:

- config.test** : An HJSON file containing all necessary information about the target project, including the test descriptions.
- testit_golden.py** : A Python module that contains the functions used to generate the golden result dataset, which the application utilizes to compare its outputs and verify correct behavior.

The config.test file consists of three main fields: **target**, **report**, and **test**. Each field provides essential configuration details for *TestIt*.

Target Field [1]. This specifies details about the test environment, including the test platform's name and type, serial connection settings (for FPGA boards), the number of random iterations, and the output file directory when using a simulation tool.

Report Field [1]. This field specifies the directory where *TestIt* stores the test report and HJSON test data.

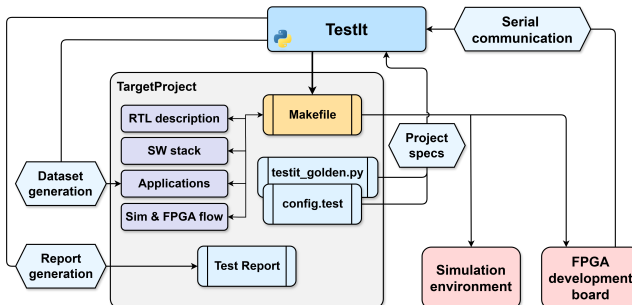


Figure 1: Structure of a TestIt environment

Listing 1: Example config.test - Target and Report Field

```
target: {
  name: "pynq-z2"
  type: "fpga"
  usbPort: 2
  baudrate: 9600
  iterations: 10
  outputFile: "path/to/sim/dump"
}
report: {
  dir: "path/to/report/folder"
}
```

Test Field [2]. This field defines all tests executed within a single iteration. Each entry includes:

- The application name and its directory.
- The .c and .h files storing input and golden datasets.
- The regular expression used by the SUT to communicate test results to the host, either via serial communication for FPGA boards or file dumping in a simulation environment.
- A list of test parameters with fixed values or ranges.
- Input and output datasets, specifying data type, value range, and dimensions, which can be parameter dependent.
- The golden function used to generate reference values.

Listing 2: Example config.test - Test Field

```
test: [
  {
    appName: "application_name"
    dir: "path/to/app"
    genFileName: "test_data"
    outputFormat: "(\\d+):(\\d+):(\\d+)"
    outputTags: ["TestID", "Cycles", "Outcome"]
    parameters: [
      {
        name: "SIZE"
        value: [4, 10]
        step: 2
      }
    ]
    inputDataset: [
      {
        name: "input_matrix"
        dataType: "uint8_t"
        valueRange: [0, 255]
        dimensions: ["SIZE", "SIZE"]
      }
    ]
    outputDataset: [
      {
        name: "output_matrix"
        dataType: "uint8_t"
      }
    ]
    goldenResultFunction: {
      name: "softmax"
    }
  }
]
```

Each test can define parameters with a name, value, and optionally a range with a step size. If a parameter is defined as a range, *TestIt* selects a value within it for each iteration. These parameters are included in the .h file and passed as arguments to the golden function in `testit_golden.py`, which can use them for the computation of reference values, if needed.

For each dataset, it is possible to specify the name of the C array storing the data, its data type, the range of values for random generation, and its dimensions. If the dimensions depend on a parameter, *TestIt* parses the corresponding parameter value for every matching dimension.

3.2 TestIt Commands

Following the design criterion of simplicity, *TestIt* needs just three commands to run the test campaign.

testit setup. This command checks for the presence of the required files, `config.test` and `testit_golden.py`, in the working directory. If these files are not found, it automatically generates fully commented templates that can be easily modified and tailored to the specific needs of the target project.

testit run. This is the core command of *TestIt*, enabling the execution of an integration test campaign as defined in `config.test`.

The process begins with initial checks on the target project's Makefile and `config.test`. Afterward, the simulation or FPGA model is built. If needed, this step can be skipped by using the `--nobuild` flag.

In the case of FPGA-based testing, the model is loaded onto the development board, the serial connection is initialized, and the debugger is set up.

The actual testing process then begins. Each iteration starts with dataset generation, which can be performed in two ways:

- By default, *TestIt* uses the "iterations" parameter in the `config.test` file. For each iteration, it selects random parameter values if required.
- Alternatively, if the command "testit run" is executed with the `--sweep` flag, *TestIt* systematically tests all possible parameter value combinations, incrementing values according to the "step" parameter.

Following the SBST approach, the application itself processes the input dataset and compares it with the golden result to verify the functionality of the SUT.

TestIt parses the output of the test using the regular expressions specified in "outputFormat" and "outputTags" parameters from `config.test`. This extracted information is then processed and stored in a JSON database within the report directory.

This approach was chosen over alternatives such as memory reads via *gdb* due to its greater flexibility. It allows test engineers to specify, for each individual test application, which data should be acquired. In some cases, this may be limited to the test ID and result, but the regular expression can be extended to include additional details such as execution cycles, the number of detected errors, and other relevant metrics.

testit report. Finally, *TestIt* can generate a summary report containing the data acquired during the test campaign. The report can be displayed directly in the terminal sorted by any metric extracted from the regular expression using the `--sort_key [KEY]` flag, with the option to sort in descending order by specifying `--descending`.

4 USE CASE: CUSTOM ACCELERATOR INTEGRATION IN X-HEEP MCU

As previously discussed, *TestIt* has been developed primarily for system-level integration testing. To validate this approach and demonstrate its potential, we present an integration test campaign conducted on X-HEEP [3], an open-source 32-bit RISC-V MCU.

In this scenario, X-HEEP has been extended with a Smart Peripheral Controller (SPC) accelerator that optimizes the *im2col* reshaping transformation, enabling convolutions to be performed as single matrix multiplications [8]. This accelerator is tightly integrated with the X-HEEP's DMA to perform 2D transactions, which are called sequentially by the *im2col* accelerator to perform the final transformation. Due to its tight integration within the X-HEEP platform, it serves as a prime example of the need for system-level integration testing.

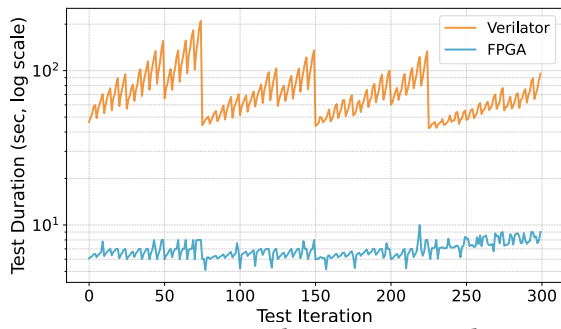


Figure 2: Test duration in seconds

4.1 Performance Characterization

Performance characterization is crucial for IC design because it enables designers to find optimal trade-offs. While simulations can be time-consuming, *TestIt*'s FPGA support and automation features allow efficient characterization of RTL designs.

To demonstrate the speedup that FPGA-based testing can achieve, we carried out a 300 iteration test campaign both on the PYNQ-Z2 FPGA development board and performing Verilator simulations, targeting the im2col SPC+X-HEEP system. The test application was modified to take advantage of the on-board timer to record timestamps before and after each test. Their difference was then transmitted to the host device using the previously mentioned regular expression mechanism, thus enabling performance characterization. The FPGA-based campaign took around 0:31 hours to complete, while the Verilator-based one took 6:07 hours, a 11× increase in test time. Figure 2 graphically compares the two approaches by showing the duration of each test iteration in seconds on a logarithmic scale.

4.2 Hardware fault

To evaluate the effectiveness of our approach, we injected a hardware fault into the SPC, one that could only be detected through system-level testing. Specifically, we modified the controller responsible for managing the communication between the accelerator and X-HEEP's peripheral interface. The used protocol implements a simple request-response handshake, where the controller must wait for the response valid signal before de-asserting the request valid signal. Our fault removed this synchronization mechanism, potentially disrupting the entire functionality of the SPC.

In a standard *UVM* environment, idealized external interfaces cause the response valid signal to consistently appear one cycle later, rendering synchronization issues undetectable even in full system-level simulations, whereas FPGA-based testing introduces real-world effects that can reveal such flaws.

4.3 Software Fault

To assess *TestIt*'s ability to validate the software stack, we conducted a test campaign on the im2col transformation using three implementations: a C-based algorithm, a DMA-optimized version, and the im2col SPC. To evaluate the fault detection capabilities of the tool, we introduced an error in the DMA HAL function `get_increment_b_1D()`. Specifically, a `uint8_t` variable was incorrectly used instead of a `uint32_t` for computing byte increments, leading to an overflow error in sufficiently large transactions. The fault appears only in the DMA-based test, while the im2col SPC functions correctly. This distinction enables test engineers to quickly identify the issue as software-related rather than hardware-related. Figure 3 presents the results of this testing campaign, illustrating how the size of the modified variable impacts test outcomes. Notably, issues arise when increments exceed 256 words, triggering the overflow error.

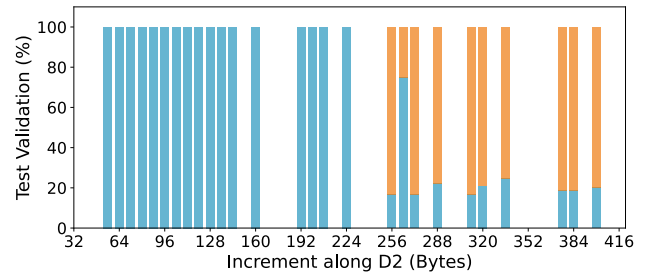


Figure 3: Integration test results with the DMA HAL fault

5 FUTURE WORKS

Continuous Integration and Continuous Deployment (CI/CD) is a widely adopted methodology that aims at simplifying and accelerating development cycles. It consists of automatically integrating code changes into a shared repository, running extensive automated tests, and deploying updates with minimal manual intervention. While CI/CD is a popular practice in software engineering, its adoption in hardware design remains limited.

As a future development, we propose leveraging *TestIt*'s capabilities to implement an FPGA-based CI/CD system for RTL projects. This would involve integrating a *TestIt*-based environment into the development workflow using a self-hosted GitHub Action. Upon each git push to the repository, *TestIt* would automate FPGA synthesis, model loading, and test execution, streamlining the integration and validation of new features. Therefore, *TestIt* could serve as a critical enabler in integrating CI/CD methodologies within open-source RTL development, especially when combined with unit-level formal verification. In conclusion, this work represents a significant step forward in the broader adoption of agile practices [2] in hardware design.

ACKNOWLEDGMENTS

This work is part of the project NODES which has received funding from the MUR - M4C2 1.5 of PNRR funded by the European Union - NextGenerationEU (Grant agreement no. ECS00000036). Also, this work was supported in part by the the Swiss NSF Edge-Companions project (GA No. 10002812); in part by the EC H2020 FVLLMONTI Project under Grant 101016776; in part by the ACCESS—AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong, SAR; and in part by the Swiss State Secretariat for Education, Research, and Innovation (SERI) through the SwissChips Research Project.

REFERENCES

- [1] Cocotb 2017. *Cocotb GitHub repository*. Retrieved Feb. 22, 2025 from <https://github.com/cocotb/cocotb>
- [2] John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* 62, 2 (Jan. 2019), 48–60. doi:10.1145/3282307
- [3] Simone Machetti, Pasquale Davide Schiavone, Thomas Christoph Müller, Miguel Peón-Quirós, and David Atienza. 2024. X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators. arXiv:2401.05548 [cs.AR]
- [4] Marcelo Orenes-Vera, Aninda Manocha, David Wentzlaff, and Margaret Martonosi. 2021. AutoSVA: Democratizing Formal Verification of RTL Module Interactions. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 535–540.
- [5] Renode 2017. *Renode GitHub repository*. Retrieved Feb. 20, 2025 from <https://github.com/renode/renode>
- [6] A. Ruospo, D. Piumatti, A. Florida, and E. Sanchez. 2021. A Suitability Analysis of Software Based Testing Strategies for the On-line Testing of Artificial Neural Networks Applications in Embedded Devices. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 1–6. doi:10.1109/IOLTS52814.2021.9486704
- [7] SymbiYosys 2017. *SymbiYosys GitHub repository*. Retrieved Feb. 23, 2025 from <https://github.com/YosysHQ/sby>
- [8] Tommaso Terzano. 2024. *Development of an Advanced Configurable DMA System for Edge AI Accelerators in a 16nm Low Power RISC-V Microcontroller*. Master's Thesis. Politecnico di Torino. <https://webthesis.biblio.polito.it/33222/>