VFUK 2025

HDLRegression – A reliable and efficient tool for FPGA regression testing

Inventas

- Norway’s largest independent design center and product development company
- Established in 1997
- 170 designers and engineers
- Developing UVVM & HDLRegression
- Established the UVVM Steering Committee
- Provider of UVVM methodology and IP

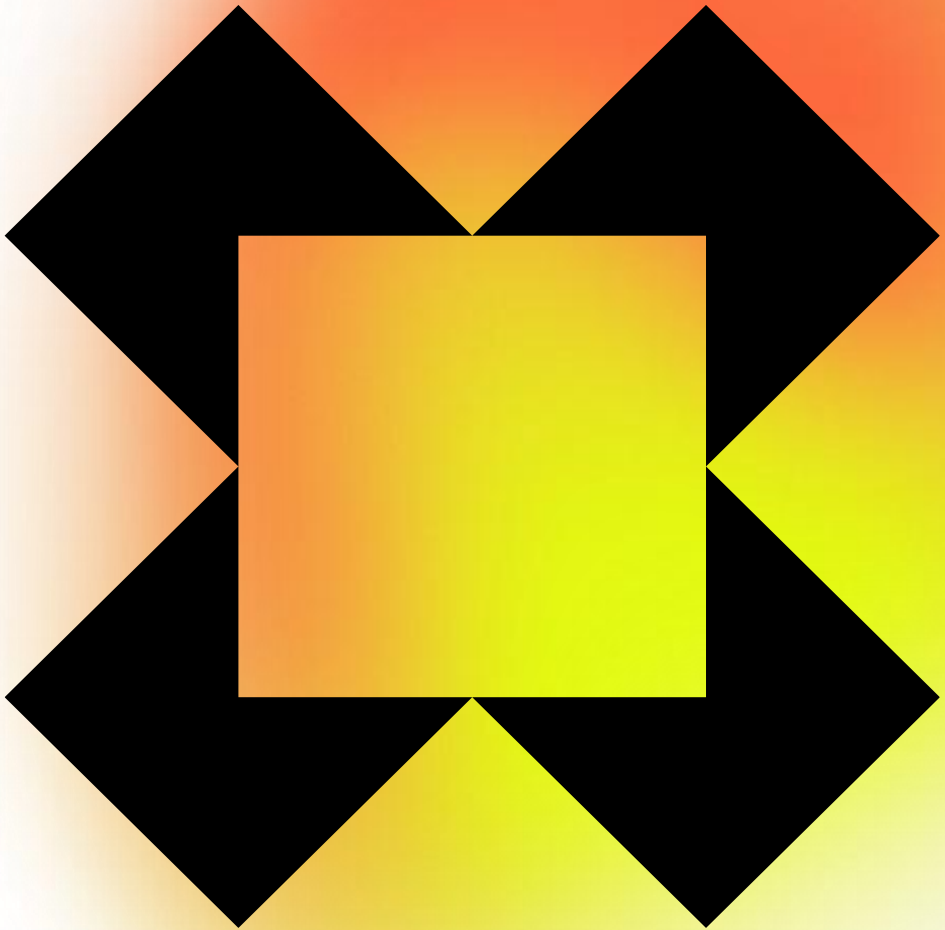
Topics

Verification challenges

Why regression - and what it is

HDLRegression - key features & workflow

Key take-aways

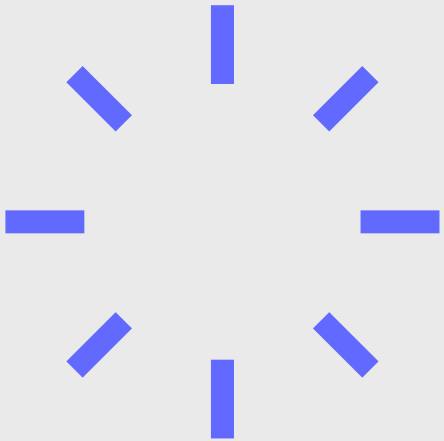


Verification challenges

- 87 % of projects had at least one non-trivial bug escape into production.
- ≈ 67 % missed their original schedule.
- Verification can consume up to 50% of total project effort.
- Debugging is the single largest time sink for FPGA verification engineers.

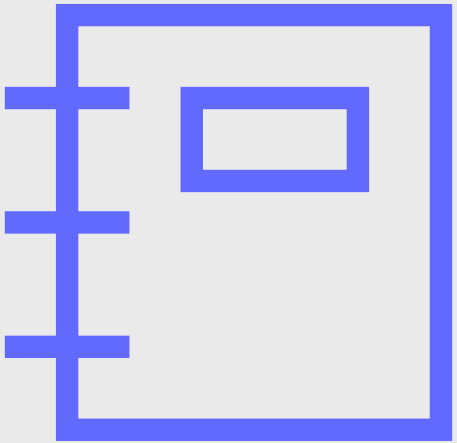
*2024 Wilson Research Group FPGA functional verification trends

Why do these problems happen?



- Late discovery → many teams still only do a quick test before going to the lab; bugs appear late and cost more.
- Design complexity → SoC FPGAs, several clock domains, and embedded CPUs.
- Ad-hoc scripts → manually maintained compile orders and test lists get outdated fast.
- Unpredictable debug → one bad failing test can take days to fix.
- Change ripple → a fix in one block breaks logic that previously worked.

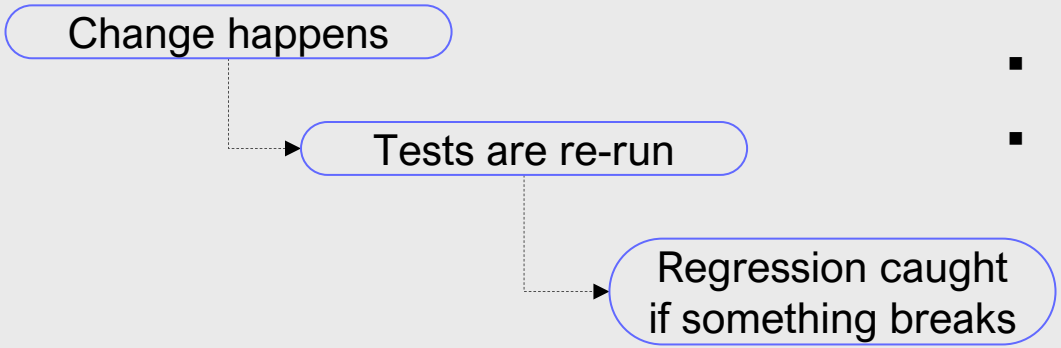
Some types of testing



- Unit testing
- Integration testing
- System testing
- Retesting
- Randomised / constrained testing
- Directed testing
- Formal verification
- Functional testing
- Regression testing

What is regression testing?

- Re-running the same functional and non-functional tests after each change.
- Confirms that behaviour we already validated still works.
- When something that passed yesterday fails today, that’s a regression.
- Must be fully automated and repeatable.
- Usual split: quick sanity pack and full suite overnight.



Benefits of regression testing

- Early fault detection – re-run tests after every code or configuration change.
- Foundation for CI pipelines – CI tools rely on healthy regression suite.
- Lower cost of defects – issues fixed at commit-time are orders of magnitude cheaper than those found in the lab – or in the field.
- Confidence to refactor and innovate – clean up code or add features without silently breaking existing functionality.

Retesting vs regression testing

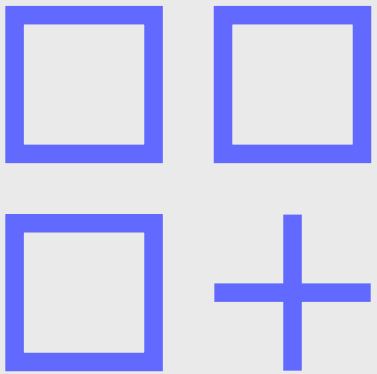
Retesting

- Focuses on a specific bug fix.
- Performed after fixing a known defect.
- Targets a specific failed test case.
- Usually manual.
- Confirms critical defects.

Regression testing

- Ensure recent changes haven’t broken existing functionality.
- Performed after any code change.
- Reruns a broader set of previously passed test.
- Often automated.

Regression testing in FPGA projects



- Code, config and constraints changes
- Recompile and simulation cycles are long
- Testbenches are complex and evolve with the design
- Changes in tooling and environment add more risk

When is a test case ready for regression?

- Defined purpose
- Deterministic behavior
- Self-checking
- Clean dependencies
- Passes consistently

Effective regression strategies

- Prioritise core functionality and recent changes
- Run different test scopes: fast checks, full regressions
- Plan for test reuse and maintenance
- Automate where possible

What to run daily and what to run at milestones

Daily regression

- Focus on functionality that is
  - Stable and unlikely to change often
  - Important for system correctness
  - Fast to simulate

- Examples
  - Reset and initialization
  - FSMs
  - Control registers
  - Communication
  - Interrupt logic

Milestone / nightly regression

- Include
  - Large tests or full system tests
  - Randomised or coverage-driven tests
  - Corner cases
  - Long simulations

- When to run
  - Approaching release
  - After major merges
  - During nightly / weekly CI

- Examples
  - System-level simulations
  - Long randomised sequences
  - Full code/functional coverage

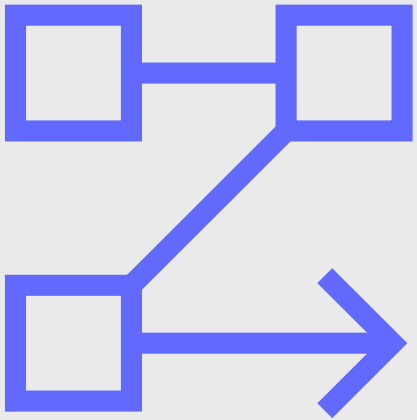
Common pitfalls

- Outdated test scripts
- Flaky tests: sometimes pass, sometimes fail.
- Poorly maintained testbenches
- Lack of ownership in large teams

Benefits of test automation

- Much faster than manual testing
- Reliable results
- Ensure consistency
- Saves time and cost
- Human intervention is not required while execution
- Increases efficiency
- Re-usable test scripts
- Test frequently and thoroughly

Continuous regression



- Run tests each time we push code
  → we spot errors right away.
- Just the needed tests run after a change
  → can save minutes / hours.
- Small quick set on every push to the server, full test at night or before milestone/release.
- Keep logs from every run
  → easy to see when a bug showed up.
- Gives developers trust to commit often without breaking old features.

What to look for in an HDL regression tool

Feature	HDLRegression
Simulator support	GHDL, NVC, Questa, Modelsim, Riviera-PRO
Easy setup	One pragma, one Python script
Test management	Filter by test case name or group
Test automation / CI integration	CLI + exit codes + logs
Debug support	Waveforms, logs, color coded PASS/FAIL
Scalable	Works for units and top-level
Framework-support	UVVM, OSVVM, Vunit, or in-house

What is HDLRegression?

- An open-source regression runner for VHDL/Verilog TB – written in Python 3.
- Ultra-easy start-up – only a pragma comment in the TB; no refactor.
- Framework independent – works with UVVM, OSVVM, Vunit, or in-house code.
- Auto-detects ModelSim, Questa, Riviera-PRO, GHDL, NVC, and drives them for you.
- Scales from quick unit tests to large, multi-library projects.
- Goal: simplicity and efficiency → focus on writing good tests.

Testbench integration and test cases

- One pragma comment to mark testbench.
- No renaming or restructuring required.
- Test cases defined by generics, entity-architecture, and sequencer built-in test cases.

```
--HDLRegression:TB
entity tc_tb is
  generic (
      GC_TESTCASE     : string    := "UVVM TB";
      GC_GENERIC_1    : natural   := 1;
      GC_GENERIC_2    : natural   := 2;
      GC_PATH         : string    := ""
      );
end tc_tb;

architecture tb_arch of tc_tb is
begin

  p_seq : process
  begin
      if GC_TESTCASE = "tc_1" then
          ...
      elsif GC_TESTCASE = "tc_2" then
          ...
      else
          ...
      end if;

       report_alert_counters(FINAL);
      std.env.stop;
      wait;
  end process;

end architecture tb_arch;
```

Simple Python script

```
from hdlregression import HDLRegression

hr = HDLRegression()

hr.add_files("../src/*.vhd", "design_lib")
hr.add_files("../tb/*.vhd", "test_lib")

hr.start()
```

- 6-12 lines of plain Python.
- Import HDLRegression, list files & libraries, and call start().
- Same script runs RTL and netlist simulations.
- Use the same script to run locally and in CI pipelines.
- Full Python language available.

Select TC to run

```
sim % python ../script/sim.py -ltc

========================================================================
  HDLRegression version 0.61.2
  See /doc/hdlregression.pdf for documentation.
========================================================================

Scanning files...
Building test suite structure...
TC:1 - tc_tb.tb_arch.random_write_and_read
TC:2 - tc_tb.tb_arch.tc_read_empty
```

```
sim % py ../script/run.py -tc 2

========================================================================
  HDLRegression version 0.61.2
  See /doc/hdlregression.pdf for documentation.
========================================================================

Scanning files...
Building test suite structure...
Simulator: NVC

Starting simulations...
Running 1 out of 2 test(s) using 1 thread(s).
Running: test_lib.tc_tb.tb_arch.tc_read_empty (test_id: 2)
Result: PASS (0h:0m:0s).

------------------------------------------------------------------------
Simulation run time: 0h:0m:0s.
SIMULATION SUCCESS: 1 passing test(s).
```

- Default regression mode:

  runs any new tests and any tests that failed the last time.
- Run a single test case:

  `-tc <tc id>` or

  `–tc <entity>.<arch>.<tc>`
- Use wildcards to filter:

  `-tc uart_tb.*.rx*`

Select TG to run

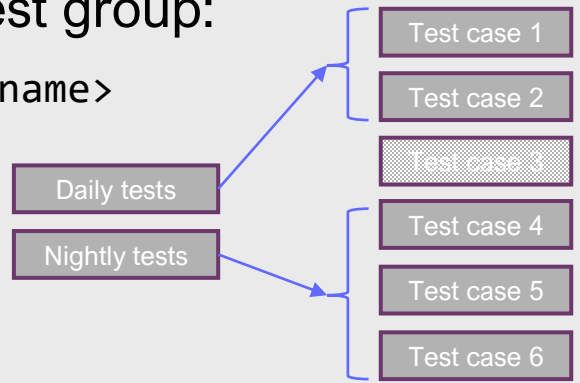
```
hr.add_to_testgroup("nightly_tests", "tc_tb", "tb_arch", "random_write_and_read")
hr.add_to_testgroup("daily_tests", "tc_tb", "tb_arch", "tc_read_empty")

hr.start()
```

```
Sim % python ../script/sim.py -ltg
================================================================================
HDLRegression version 0.61.2
See /doc/hdlregression.pdf for documentation.
================================================================================

Scanning files...
Building test suite structure...
|---- nightly_tests
|   |--  tc_tb.tb_arch.random_write_and_read
|---- daily_tests
|   |--  tc_tb.tb_arch.tc_read_empty
```

- Gather sanity tests, overnight regressions, corner case tests, interface tests etc. in groups.
- Define test groups in the regression script

  `<tg name>, <entity>, <arch>, <tc>`
- Run a test group:

  `-tg <tg name>`



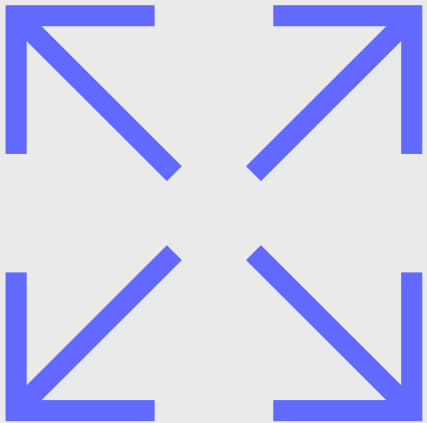
Fast local feedback

- Incremental re-run→ only tests touched by changed files run.
- Unit-level tests finish in seconds on a laptop.
- Instant PASS/FAIL summary with exit code.

```
Starting simulations...
Running 2 out of 2 test(s) using 1 thread(s).

Running: test_lib.tc_tb.tb_arch.random_write_and_read (test_id: 1)
Result: FAIL (0h:0m:0s)
Test run: sim_errors=1, sim_warnings=0.

Running: test_lib.tc_tb.tb_arch.tc_read_empty (test_id: 2)
Result: PASS (0h:0m:0s).

------------------------------------------------------------
Simulation run time: 0h:0m:0s.
SIMULATION FAIL: 2 tests run, 1 test(s) failed.
```

Grows with the project

- Auto dependency scan builds compile order as libraries grow.
- Thread flag `-t N` fans out over all cores/licenses.
- Rarely need to touch the script during the project.



End-to-end Run

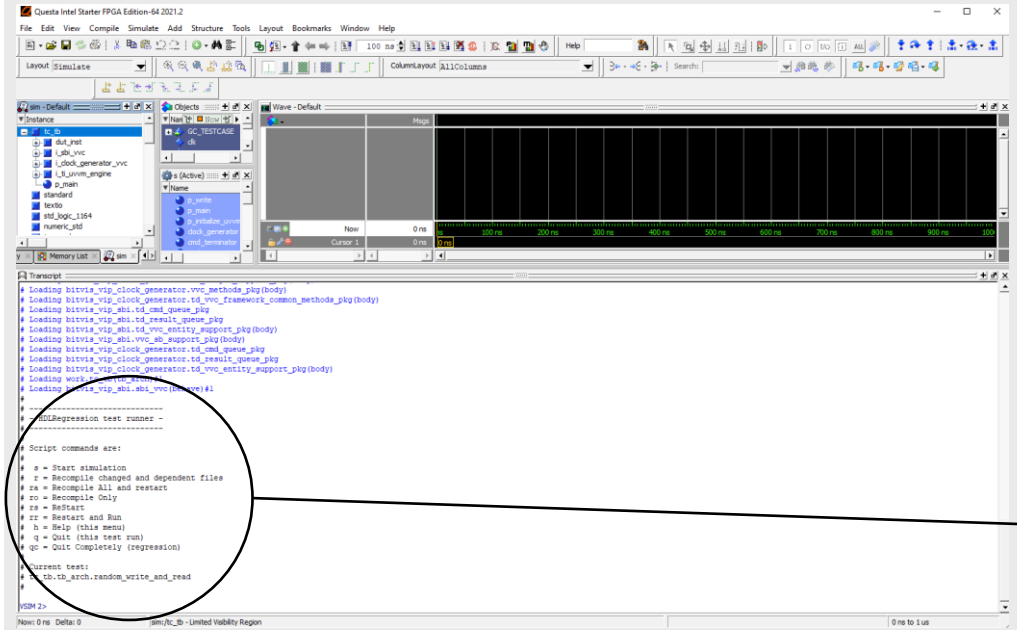
- Scan → Detect unchanged → Skip → Run changed tests
- Failing tests stay *red* until fixed
- Full regression:

  `-fr / --fullRegression`

```
sim % python ../script/sim.py

================================================================================.
 HDLRegression version 0.61.2
 See /doc/hdlregression.pdf for documentation.
================================================================================

Scanning files...
Building test suite structure...
Simulator: NVC

Starting simulations...
Test run not required. Use "-fr"/"--fullRegression" to re-run all tests.
```

```
sim % touch ../src/mem_block.vhd

sim % python ../script/sim.py

================================================================================
 HDLRegression version 0.61.2
 See /doc/hdlregression.pdf for documentation.
================================================================================

Scanning files...
Building test suite structure...
Simulator: NVC
Compiling library: design_lib  - OK -

Starting simulations...
Moving previous test run to: ./hdlregression/test_2025-04-30_11.46.58.379030.
Running 2 out of 2 test(s) using 1 thread(s).
Running: test_lib.tc_tb.tb_arch.random_write_and_read (test_id: 1)
Result: PASS (0h:0m:0s).

Running: test_lib.tc_tb.tb_arch.tc_read_empty (test_id: 2)
Result: PASS (0h:0m:0s).

--------------------------------------------------------------------------------
Simulation run time: 0h:0m:0s.
SIMULATION SUCCESS: 2 passing test(s).
```

GUI and waves

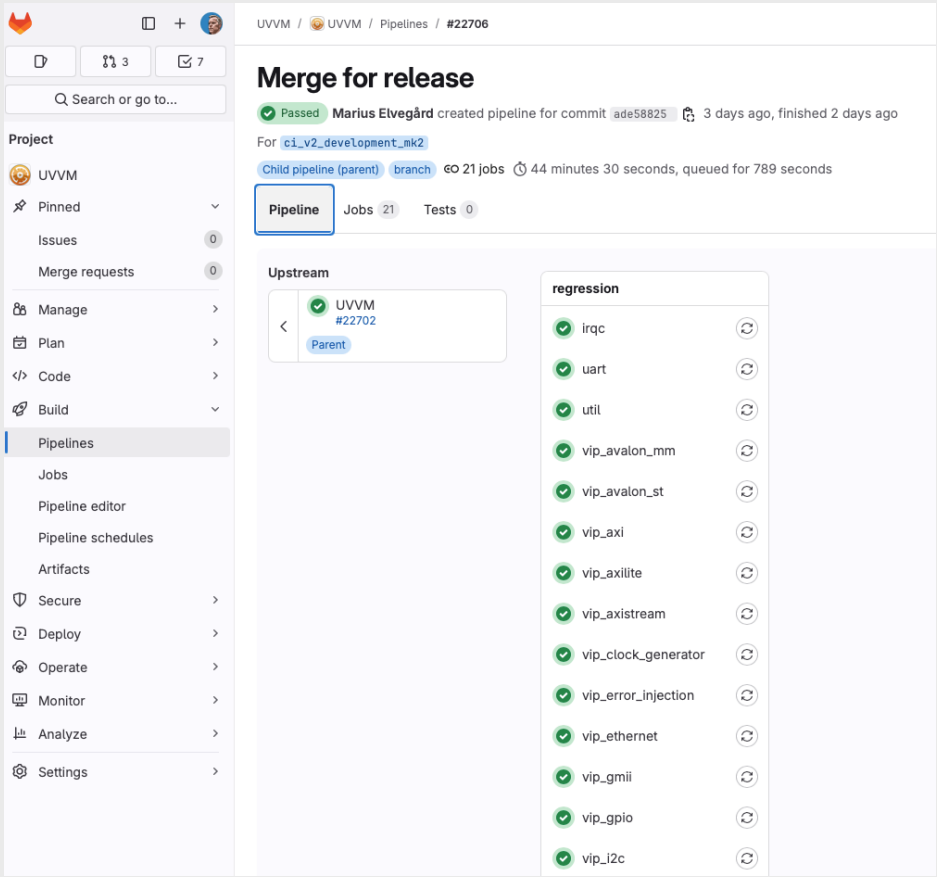


- `-g / --gui` CLI options opens ModelSim/Questa with project precompiled and test case loaded.
- Work interactively in GUI while debugging.
- GHDL / NVC dumps VCD/FST files.

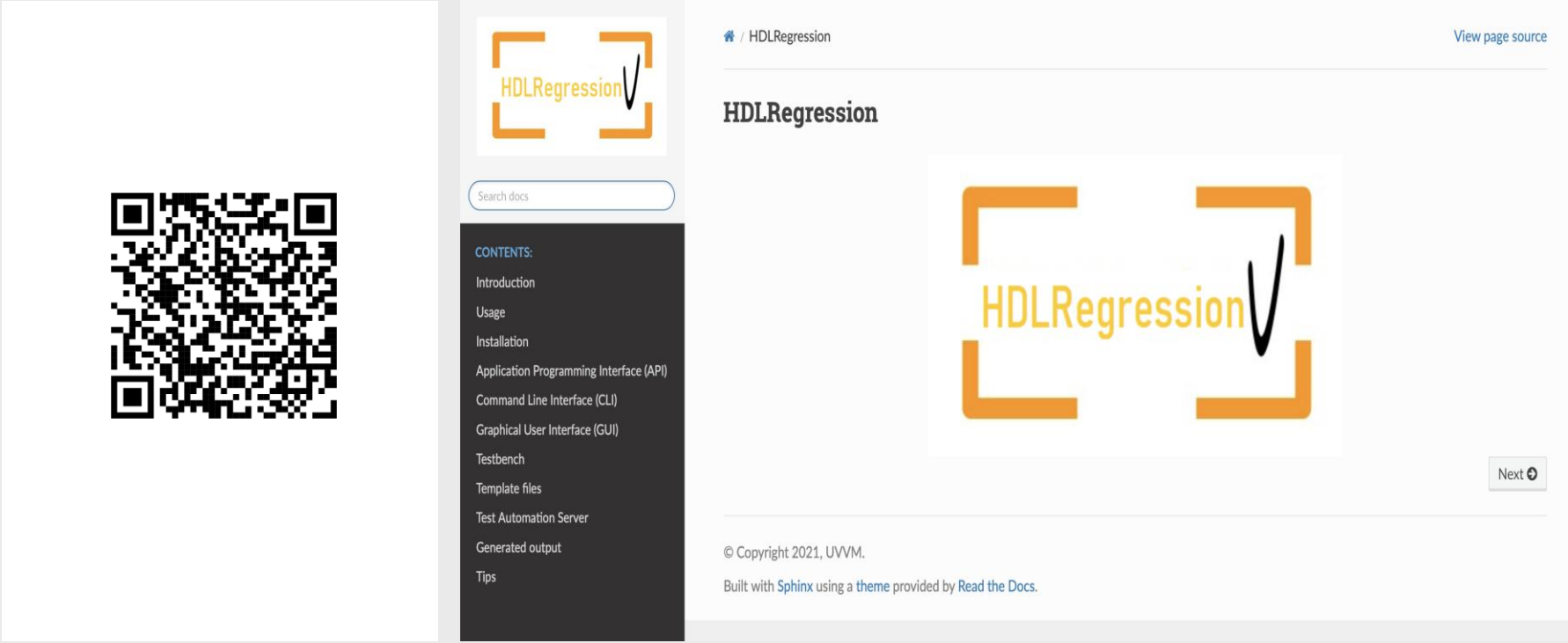
```
# ------------------------------
# - HDLRegression test runner -
# ------------------------------
#
# Script commands are:
#
#  s = Start simulation
#  r = Recompile changed and dependent files
# ra = Recompile All and restart
# ro = Recompile Only
# rs = ReStart
# rr = Restart and Run
#  h = Help (this menu)
#  q = Quit (this test run)
# qc = Quit Completely (regression)
#
# Current test:
# tc_tb.tb_arch.random_write_and_read
#
```

CI pipelines

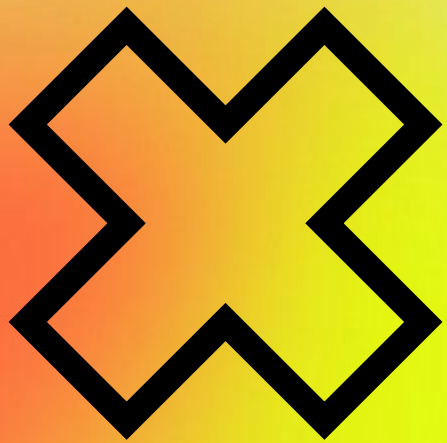
- Easy integration with CI.
- Same script, check in with rest of the code.
- PASS/FAIL badge from result (exit code).
- Stores logs and coverage as artifacts.



Documentation



Key take-aways



- Regression improves the verification flow.
- Use CI pipelines for running regression on each push.
- A reliable regression tool and self-checking TB is key.
- HDLRegression is easy to use
  - One pragma in the TB.
  - One small Python script.
  - Works with any verification FW.
  - Fast feedback, CI-ready, scales with your projects.
  - Open-source and free on GitHub