Verification of digital circuits using Java

Bachelor’s project

19th of May 2025

Rasmus Wiuff

DTU

Outline

Introduction

Problem specification

Design

Implementation

Result

Further development

Conclusion

Introduction

- Chip design requires verification
- Verification most commonly done using UVM
- Commonly used frameworks: Chisel, SpinalHDL, pyuvm
- ABV and formal verification improves the verification step

Verification cycles reduced by 25-30%
Pre-silicon bug detection rates improved by 20%
Security vulnerability detection increased by 40%

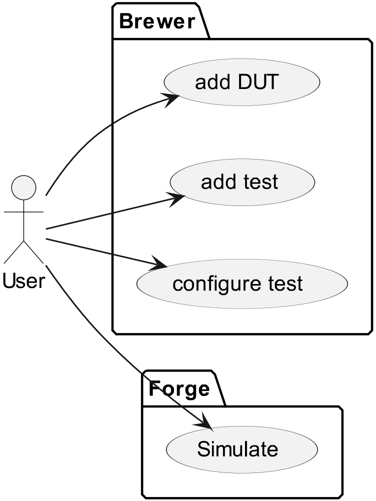
Problem specification

A chip verification framework written in Java, supporting SystemVerilog and core ideas from ABV, thus making it easy for designers to write their designs in SystemVerilog and use a well known language to implement assertion based tests.

Challenge	Success Criteria
Simulation driver	Launching and handling output from Verilator
Peek-poke-step	Basic verification tests
Assertions	SVA assertions
Test-translation	Translate tests into a testbench
Concurrency	Concurrent execution of the tests

Usecases

- Adding devices
- Adding tests
- Configuring tests
- Run simulations



Separation of responsibility

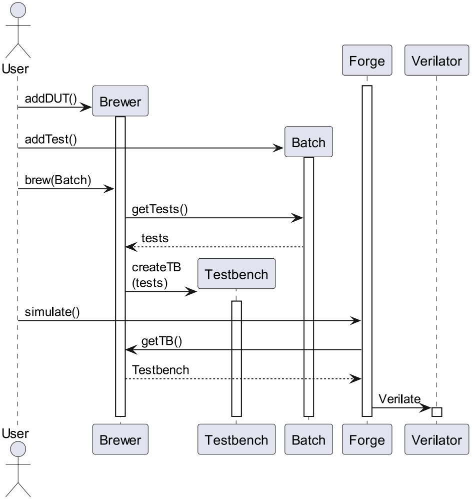
The Brewer

- Adding devices
- Handling test logic
- Preparing testbenches

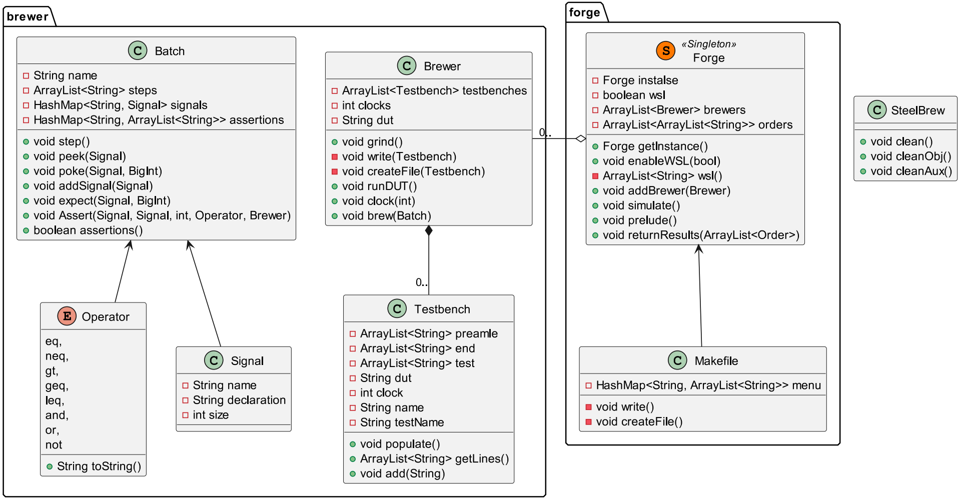
The Forge

- Define command arguments
- Launch Verilator
- Collect Verilator output
- Handle concurrency

The final workflow



The program structure



brewer

C Batch
□ String name
□ ArrayList<String> steps
□ HashMap<String, Signal> signals
□ HashMap<String, ArrayList<String>> assertions
● void step()
● void peek(Signal)
● void poke(Signal, BigInt)
● void addSignal(Signal)
● void expect(Signal, BigInt)
● void Assert(Signal, Signal, int, Operator, Brewer)
● boolean assertions()

C Brewer
□ ArrayList<Testbench> testbenches
□ int clocks
□ String dut
● void grind()
■ void write(Testbench)
■ void createFile(Testbench)
● void runDUT()
● void clock(int)
● void brew(Batch)

E Operator
eq,
neq,
gt,
geq,
leq,
and,
or,
not
● String toString()

C Signal
□ String name
□ String declaration
□ int size

C Testbench
□ ArrayList<String> preamle
□ ArrayList<String> end
□ ArrayList<String> test
□ String dut
□ int clock
□ String name
□ String testName
● void populate()
● ArrayList<String> getLines()
● void add(String)

forge

S «Singleton» Forge
□ Forge instalse
□ boolean wsl
□ ArrayList<Brewer> brewers
□ ArrayList<ArrayList<String>> orders
● Forge getInstance()
● void enableWSL(bool)
■ ArrayList<String> wsl()
● void addBrewer(Brewer)
● void simulate()
● void prelude()
● void returnResults(ArrayList<Order>)

C Makefile
□ HashMap<String, ArrayList<String>> menu
■ void write()
■ void createFile()

C SteelBrew
● void clean()
● void cleanObj()
● void cleanAux()

Defining tests

Running tests

Using the project

Further development

Conclusion

Q&A

Questions?