

# **Verification of digital circuits using Java**

Bachelor's project

*19th of May 2025*

**Rasmus Wiuff**

DTU

# Outline

Introduction

Problem specification

Design

Implementation

Defining tests

Running tests

Result

Further development

Conclusion

# Introduction

- Chip design requires verification
- Verification most commonly done using UVM
- Commonly used frameworks: Chisel, SpinalHDL, pyuvvm
  - pyuvvm requires UVM approach
  - Chisel, SpinalHDL requires adapting to Scala
- 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%

---

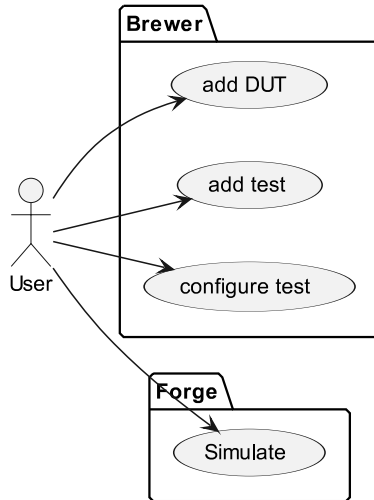
# Problem specification

**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

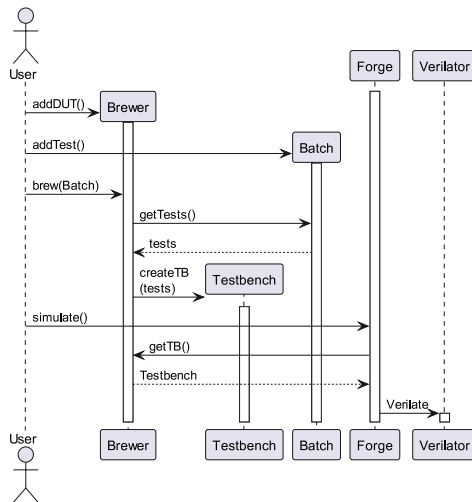
## The Brewer

- Adding devices
- Handling test logic
- Preparing testbenches

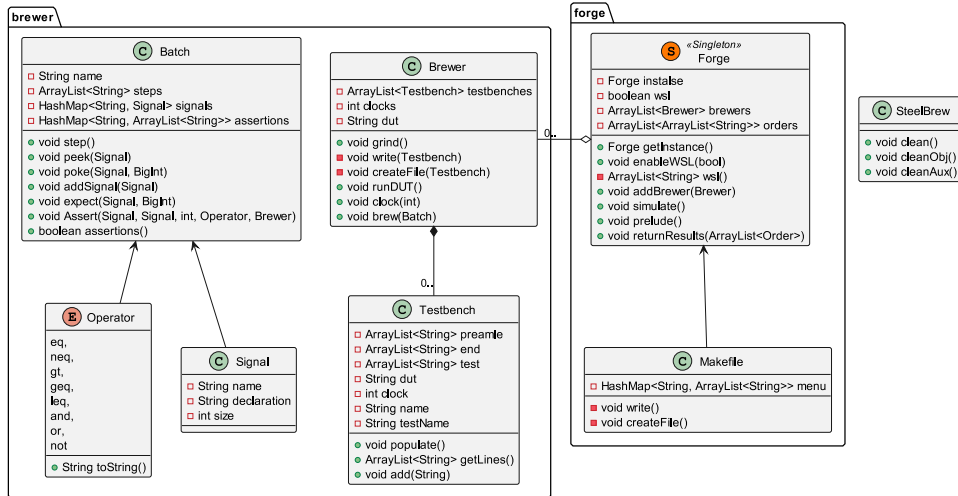
## The Forge

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

# The final workflow



# The program structure





# Batch

<div> <div></div> <div>Batch</div> </div>
<div> <div></div> String name           <div></div> ArrayList&lt;String&gt; steps           <div></div> HashMap&lt;String, Signal&gt; signals           <div></div> HashMap&lt;String, ArrayList&lt;String&gt;&gt; assertions         </div>
<div> <div></div> void step()           <div></div> void peek(Signal)           <div></div> void poke(Signal, BigInt)           <div></div> void addSignal(Signal)           <div></div> void expect(Signal, BigInt)           <div></div> void Assert(Signal, Signal, int, Operator, Brewer)           <div></div> boolean assertions()         </div>

```


public void peek(Signal signal) {
    steps.add("std::cout << \"\\n Peek on " + signal.getName()
        + ": \" << (int)(dut->" + signal.getName()
        + ") << \" @ simtime: \" << sim_time << std::endl;\\n");
}

public void poke(Signal signal, BigInteger integer) {
    steps.add("dut->" + signal.getName() + " = "
        + integer.intValue() + ";\\n");
    steps.add("std::cout << \"\\n Poke on " + signal.getName()
        + ": " + integer.intValue()
        + "\\n<< \" @ simtime: \" << sim_time << std::endl;\\n");
}

public void expect(Signal signal, BigInteger integer) {
    steps.add("std::cout << \"\\n Expected " + integer.intValue()
        + " on " + signal.getName()
        + ". Recieved \" << (int)(dut->" + signal.getName()
        + ") << \" @ simtime: \" << sim_time << std::endl;\\n");
}

```

# Brewer

 Brewer
<div> <div>□</div> ArrayList&lt;Testbench&gt; testbenches           </div> <div> <div>□</div> int clocks           </div> <div> <div>□</div> String dut           </div>
<div> <div>●</div> void grind()           </div> <div> <div>■</div> void write(Testbench)           </div> <div> <div>■</div> void createFile(Testbench)           </div> <div> <div>●</div> void runDUT()           </div> <div> <div>●</div> void clock(int)           </div> <div> <div>●</div> void brew(Batch)           </div>

```

public void brew(Batch batch) {
    Testbench testbench = new Testbench(dut, batch.getName(), clocks);
    if (batch.assertions()) {
        HashMap<String, ArrayList<String>> assertions =
            batch.getAssertions();

        Set<String> functions = assertions.keySet();
        functions.forEach(f -> assertions.get(f).forEach(
            s -> testbench.addFunc(s)));

        testbench.add("    while (sim_time < MAX_SIM_TIME) {\n");
        ...
        functions.forEach(f -> testbench.add(f + "\n"));
        ...
    }
    ArrayList<String> steps = batch.getSteps();
    for (String step : steps) {
        testbench.add(step);
    }
    testbenches.add(testbench);
}

```

# Testbench

## C Testbench

- ArrayList<String> preamble
- ArrayList<String> end
- ArrayList<String> test
- String dut
- int clock
- String name
- String testName

- void populate()
- ArrayList<String> getLines()
- void add(String)

```
public void populate() {
    preamble.add("#include <verilated.h>\n");
    preamble.add("#include \"V\" + testName + ".h\"\n");
    ...
    main.add("int main(int argc, char** argv, char** env) {\n");
    main.add("    V\" + testName + " *dut = new V\" + testName + ";\n");
    main.add("    Verilated::traceEverOn(true);\n");
    ...
    main.add("    m_trace->open(\"waveform\" + testName + ".vcd\");\n");
    ...
    end.add("    exit(EXIT_SUCCESS);\n");
}

public ArrayList<String> getLines() {
    ArrayList<String> returnList = new ArrayList<>();
    returnList.addAll(preamble);
    ...
    returnList.addAll(test);
    returnList.addAll(end);
    return returnList;
}
```

# Forge

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

```

public static void simulate() {
    prelude();
    ArrayList<String> tests = new ArrayList<>();
    brewers.forEach(b -> b.getTestbenches().forEach(
        t -> tests.add(t.getName())));
    for (String test : tests) {
        ArrayList<String> list = new ArrayList<>();
        if (wsl) list = wsl();
        list.add("make");
        list.add(test);
        orders.add(list);
    }
    new Barista(orders);
}

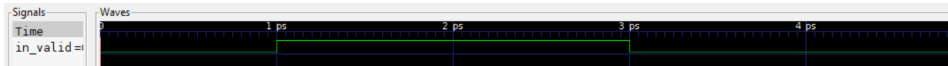
public static void prelude() {
    brewers.forEach(b -> b.grind());
    new Makefile(brewers);
}

```

# Peeking and poking

```
Forge.enableWSL(true);
Brewer alu = new Brewer("alu");
Batch batch = new Batch("PeekPokeStep");
Signal signal = new Signal("in_valid", 1);
batch.addSignal(signal);
batch.peek(signal);
batch.poke(signal, BigInteger.ONE);
batch.step();
batch.peek(signal);
batch.step();
batch.poke(signal, BigInteger.ZERO);
batch.step();
batch.peek(signal);
batch.step();
batch.step();
alu.brew(batch);
Forge.simulate();
```

```
Peek on in_valid: 0 @ simtime: 1
Peek on in_valid: 1 @ simtime: 2
Peek on in_valid: 0 @ simtime: 4
#####
```



# Expect

```
Forge.enableWSL(true);  
Brewer alu = new Brewer("alu");  
Batch batch = new Batch("Expect");  
Signal signal = new Signal("in_valid", 1);  
batch.addSignal(signal);  
batch.step();  
batch.expect(signal, BigInteger.ONE);  
batch.peak(signal);  
batch.step();  
alu.brew(batch);  
Forge.simulate();
```

```
Exptected 1 on in_valid. Recieved 0 @ simtime: 2  
Peek on in_valid: 0 @ simtime: 2
```

# Concurrency

```
Forge.enableWSL(true);
Brewer alu = new Brewer("alu");
Brewer alu2 = new Brewer("alu2");
Batch batch = new Batch("PeekPokeStep");
Signal signal = new Signal("in_valid", 1);
batch.addSignal(signal);
batch.peek(signal);
batch.poke(signal, BigInteger.ONE);
batch.step();
batch.peek(signal);
batch.step();
batch.poke(signal, BigInteger.ZERO);
batch.step();
batch.peek(signal);
batch.step();
batch.step();
alu.brew(batch);
alu2.brew(batch);
Forge.simulate();
```

java.exe	30108	1,11	310.11 kB/s	615.36 MB	BOBBOT14\vwuif	OpenJDK Platform binary
wslexe	27628			1.35 MB	BOBBOT14\vwuif	Windows Subsystem for Linux
conhost.exe	11620			1.16 MB	BOBBOT14\vwuif	Console Window Host
wslexe	26316			2.5 MB	BOBBOT14\vwuif	Windows Subsystem for Linux
wsihost.exe	22280			2.29 MB	BOBBOT14\vwuif	Windows Subsystem for Linux
conho...	27932			1.77 MB	BOBBOT14\vwuif	Console Window Host
wslexe	25324			1.21 MB	BOBBOT14\vwuif	Windows Subsystem for Linux
conhost.exe	9856			1.18 MB	BOBBOT14\vwuif	Console Window Host
wslexe	16236			2.5 MB	BOBBOT14\vwuif	Windows Subsystem for Linux
wsihost.exe	8928			2.23 MB	BOBBOT14\vwuif	Windows Subsystem for Linux
conho...	14520			1.74 MB	BOBBOT14\vwuif	Console Window Host

# Assertions

**No success (as such)**

But why?

- Assertions are mostly time independent
- Testbench works, but lacks sectioning
- Including a run loop for assertions exclude peek-poke-step
- The devil is in the details

```
assertions.put(methodName + "(dut, sim_time)", assertion);
```



# Further developments

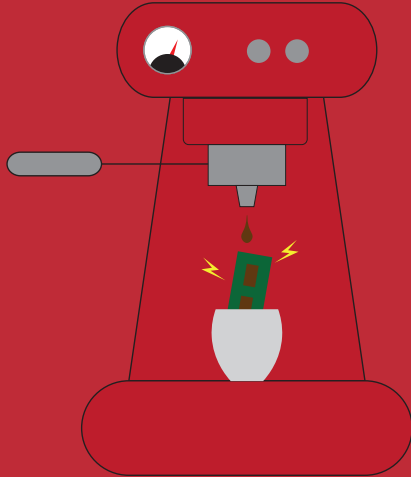
- Implement working assertions
  - Redo testbench for time independency ✓
  - Fix assert ✓
- Employ Verilator multithreading ✕
- Expand assertions with assume and cover logic ✕

# Conclusion

- Did not reach final goal: Implementing Assertion Based Verification.
- SteelBrew communicates with Verilator
- SteelBrew uses Makefiles
- Basic testing methods
- Handles concurrent execution
- Handles multiple DUT's and test routines for each

# Q&A

Questions?



# Assertion fix

1. Do a while in each testbench
2. Assertion methods defined before main method
3. Each test inclosed in simulation-time condition
4. Assertion method runs every cycle

