CODE COVERAGE WITH TREADLE

DTU - Department of Applied Mathematics and Computer Science

OUTLINE

- What is Treadle and why do we need it?

- How does Treadle work?

- Our solution for implementing Code coverage

- How can we improve this solution?

WHAT IS TREADLE?

TREADLE: A FIRRTL EXECUTION ENGINE

- FIRRTL execution engine / circuit simulator.

- FIRRTL is an intermediate representation:
  - Generated by a Chisel source.
  - Used as an “optimisation layer” before generating the output Verilog.
  - loFIRRTL is an optimised and low-level version of the original FIRRTL code => this is what Treadle uses.

TREADLE: BACKEND FOR CHISEL TESTING FRAMEWORKS

- Treadle is used as the backend of current Chisel testing frameworks like Chisel-testers2.
- Treadle runs tests defined using the testing framework on the FIRRTL circuit.
- Treadle is good for testing but lacks code coverage.
  - Line/branch coverage needs to be implemented at the simulator level and not in the framework.
  - Treadle should be able to output a line coverage report.

HOW DOES TREADLE WORK?

TREADLE: HOW DOES IT WORK?

Treadle

BlackBoxes

- Built-in BlackBox Factory
- Clock Dividers
- EICG Wrapper
- Plus Arg
- Plus Arg Reader

Chronometry

- Timer
- UTC

REPL

- REPL Config
- REPL Options
- REPL Vcd Controller
- Script
- Treadle REPL CI
- Treadle REPL Stage

Executable

Big Prim Ops	Sensitivity Graph Builder
BlackBox Cycler	SnapShotter
Clock Info	Stop Op
Clock Stepper	Symbol
Data Store	Symbol Table
Data Store Plugin	Transition
Execution Engine	Treadle Exception
Expression Compiler	Vcd Mem. Logging Cont.
Expression View Builder	Waveform Values
Int Ops	Scheduler
Long Prim Ops	RollBack Buffer
Memory	Rendered Expression
Printf Op	

Driver | **Regression**

Scala BlackBox | **Treadle Options**

VCD

- VCD
- VCD Config
- VCD Diff

Diff

- VCD Comparator
- VCD Diff Option
- VCD Diff Stage

Stage

- Treadle Tester Phase

Phases

- Create Tester
- Get FIRRTL AST
- Prepare AST
- Set Implicit Output Info

Utils

- Augment Printf
- Bit Utils
- Find Module
- Fixup Ops
- Name Based Random Number Generator
- Number Helpers
- Render
- Vcd Runner

Treadle REPL

Treadle Tester

VCD Replay Tester

TREADLE: WHAT WE CARE ABOUT

Treadle

BlackBoxes

- Built-in BlackBox Factory
- Clock Dividers
- EICG Wrapper
- Plus Arg
- Plus Arg Reader

Chronometry

- Timer
- UTC

REPL

- REPL Config
- REPL Options
- REPL Vcd Controller
- Script
- Treadle REPL CI
- Treadle REPL Stage

Executable

Big Prim Ops	Sensitivity Graph Builder
BlackBox Cycler	SnapShotter
Clock Info	Stop Op
Clock Stepper	Symbol
Data Store	Symbol Table
Data Store Plugin	Transition
Execution Engine	Treadle Exception
Expression Compiler	Vcd Mem. Logging Cont.
Expression View Builder	Waveform Values
Int Ops	Scheduler
Long Prim Ops	RollBack Buffer
Memory	Rendered Expression
Printf Op	

Driver

Regression

Scala BlackBox

Treadle Options

VCD

- VCD
- VCD Config
- VCD Diff

Diff

- VCD Comparator
- VCD Diff Option
- VCD Diff Stage

Stage

- Treadle Tester Phase

Phases

- Create Tester
- Get FIRRTL AST
- Prepare AST
- Set Implicit Output Info

Utils

- Augment Printf
- Bit Utils
- Find Module
- Fixup Ops
- Name Based Random Number Generator
- Number Helpers
- Render
- Vcd Runner

Treadle REPL

Treadle Tester

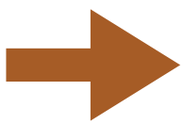
VCD Replay Tester

IMPLEMENTING COVERAGE

IMPLEMENTING COVERAGE: THE METHOD

- Based on a technique presented by Ira. D. Baxter[1]:
  - Modify source code before AST (Abstract Syntax Tree) construction.
  - For each multiplexer in design:
    1. Add 2 additional outputs (one for each mux path)
    2. Keep track of path taken in mux using the new outputs:

```
out <= mux(in$a, in$b$0, in$b$1)
```



```
io_cov_valid_0 <= in$a
io_cov_valid_1 <= mux(in$a, UInt<1>("h0"), UInt<1>("h1"))
out <= mux(in$a, in$b$0, in$b$1)
```

TREADLE: WHAT WE CHANGED

Treadle

BlackBoxes

- Built-in BlackBox Factory
- Clock Dividers
- EICG Wrapper
- Plus Arg
- Plus Arg Reader

Chronometry

- Timer
- UTC

REPL

- REPL Config
- REPL Options
- REPL Vcd Controller
- Script
- Treadle REPL CI
- Treadle REPL Stage

Executable

Big Prim Ops	Sensitivity Graph Builder
BlackBox Cycler	SnapShotter
Clock Info	Stop Op
Clock Stepper	Symbol
Data Store	Symbol Table
Data Store Plugin	Transition
Execution Engine	Treadle Exception
Expression Compiler	Vcd Mem. Logging Cont.
Expression View Builder	Waveform Values
Int Ops	Scheduler
Long Prim Ops	RollBack Buffer
Memory	Rendered Expression
Printf Op	

Driver | **Regression** | **Treadle Options**

Scala BlackBox | **Treadle REPL** | **Treadle Tester**

VCD

- VCD
- VCD Config
- VCD Diff

Diff

- VCD Comparator
- VCD Diff Option
- VCD Diff Stage

Stage

- Treadle Tester Phase

Phases

- Create Tester
- Get FIRRTL AST
- Prepare AST
- Set Implicit Output Info

Utils

- Augment Printf
- Bit Utils
- Find Module
- Fixup Ops
- Name Based Random Number Generator
- Number Helpers
- Render
- Vcd Runner

Coverage

- Coverage Parser

VCD Replay Tester

IMPLEMENTING COVERAGE: HOW?

- **Coverage Parser:**
    - Handles most of the work related to coverage.
    - Parses the FIRRTL source and adds the validators.
    - Handles the coverage report.

- **Get FIRRTL AST:**
    - Where the AST is constructed.
    - Add a call to the Coverage Parser to modify the source before AST construction.

IMPLEMENTING COVERAGE: HOW?

- **Treadle Tester:**
  - Where all the tests are run.
  - Create the coverage report when finished.

- **Driver:**
  - Where we specify simulation options.
  - Add an option allowing explicitly to activate/ deactivate coverage in order to improve performance.

RESULT: THE REPORT

```
COVERAGE: 50.0% of multiplexer paths tested
COVERAGE REPORT:

+ circuit Test_1 :
+   module Test_1 :
+     input in$a : UInt<1>
+     input in$b$0 : UInt<2>
+     input in$b$1 : UInt<2>
+     input clock : Clock
+     output io_cov_valid_0 : UInt<1>
+     output io_cov_valid_1 : UInt<1>
+     output out : UInt<2>
+
+     io_cov_valid_0 <= in$a
-     io_cov_valid_1 <= mux(in$a, UInt<1>("h0"), UInt<1>("h1"))
+     out <= mux(in$a, in$b$0, in$b$1)
```

HOW CAN WE IMPROVE?

WHAT'S LEFT?

- Coverage report could be more useful if it used the original Chisel source.
  - Map loFIRRTL back to Chisel.
  - Problem: Can only be done with simply combinatorial lines, no multiplexers. Would have to reconstruct the missing lines of code.

```
ORIGINAL REPORT:

+ circuit Test_1 :
+   module Test_1 :
+     input in$a : UInt<1>
+     input in$b$0 : UInt<2>
+     input in$b$1 : UInt<2>
+     input clock : Clock
+     output io_cov_valid_0 : UInt<1>
+     output io_cov_valid_1 : UInt<1>
+     output out : UInt<2>
+
+     io_cov_valid_0 <= in$a
-     io_cov_valid_1 <= mux(in$a, UInt<1>("h0"), UInt<1>("h1"))
+     out <= mux(in$a, in$b$0, in$b$1)
```

```
NEW REPORT:

+ class Test_1 extends Module {
+    val io = IO(new Bundle {
+        val a = Input(UInt(1.W))
+        val b = Input(Vec(2, UInt(2.W)))
+        val out = Output(UInt(2.W))
+    })
+    when(io.a) {
+        out := io.b(0)
-   }.otherwise {
-        out := io.b(1)
+    }
+ }
```

REFERENCES

- [1] Ira. D. Baxter, “Branch Coverage for Arbitrary Languages Made Easy”, 2002, Semantic Designs Inc., Austin Texas 78579 USA, http://www.semdesigns.com/Company/Publications/TestCoverage.pdf

- Current Project repository:

  https://github.com/chisel-uvm/treadle

QUESTIONS?