LCTES 2022 Artifact

This document provides evaluation guidelines for the following paper:

- paper #14: *Implicit State Machines*

It will help evaluate the following claims in the paper (Section 5):

- We implement an embedded DSL in Scala based on implicit state machines.
- The DSL examples given in the paper work as expected.
- We implement a simple microcontroller in the DSL.

Getting Started Guide

The artifact is a Docker image. The reviewer is invited to install docker and make sure it is working normally. The following official guide can help with the installation of Docker: https://docs.docker.com/get-docker/ .

1. Get the Docker image:

You may pull the docker image with the following command:

```
docker pull liufengyun/lctes22:1.0
```

Or, if you already have the container image, you can import it as follows:

```
docker load < lctes22-1.0.tar.gz
```

2. Run the Docker image:

```
docker run -it liufengyun/lctes22:1.0
```

3. Play with examples

Start the sbt:

```
cd /home/zeno && sbt
```

Compile the project:

```
compile
```

Now you can run the following example:

```
1 run asm/hello.s
```

You should be able to see “hello, world!” printed.

Step-by-Step Instructions

In this section, the following claims are supported by the artifact:

- We implement an embedded DSL in Scala based on implicit state machines.
- The examples given in the paper work in the paper.
- We implement a simple microcontroller in the DSL.

1. Implementation of DSL

The implementation is located in the following source files:

```
src/main/scala/zeno
├── lang.scala
├── core
│   ├── Trees.scala
│   ├── Types.scala
│   └── Values.scala
├── examples
│   ├── Adder.scala
│   ├── Controller.scala
│   ├── Filter.scala
│   └── Misc.scala
├── rewrite
│   ├── Interpreter.scala
│   ├── Phases.scala
│   ├── TreeAccumulator.scala
│   ├── TreeMap.scala
│   └── Verilog.scala
└── util
    ├── printing.scala
    └── Tracing.scala
```

The ASTs (abstract syntax trees) and compiler phases are defined in the following directories respectively:

- `src/main/scala/zeno/core`
- `src/main/scala/zeno/rewrite`

The file `src/main/scala/zeno/lang.scala` provides APIs for programmers to construct circuits using the DSL.

2. DSL Example: Adder

The source code for the example `Adder` can be found below:

- `src/main/scala/zeno/examples/Adder.scala`

First start the interactive Scala console:

```
cd /home/zeno && sbt console
```

Now import packages:

```
import zeno.lang.*
import zeno.examples.*
import scala.language.implicitConversions
```

We may create an instance of the adder as follows:

```
val a = input[Vec[2]]("a")
val b = input[Vec[2]]("b")
val circuit = Adder.adder2(a, b)
```

We may inspect the representation of the circuit as follows:

```
println(show(circuit))
```

We can generate the Verilog:

```
circuit.toVerilog("Adder", a, b)
```

We may create a simulator:

```
val add2 = circuit.eval(a, b)
```

Now, we can simulate the circuit with input:

```
add2(Value(1, 0) :: Value(0, 1) :: Nil)
add2(Value(1, 0) :: Value(1, 1) :: Nil)
```

3. DSL Example: Filter

The source code for the example `Filter` can be found below:

- `src/main/scala/zeno/examples/Filter.scala`

First start the interactive Scala console:

```
cd /home/zeno && sbt console
```

Now import packages:

```
import zeno.lang.*
import zeno.examples.*
import scala.language.implicitConversions
```

We may create an instance of the adder as follows:

```
val a = input[Vec[8]]("a")
val circuit = Filter.movingAverage(a)
```

We may inspect the representation of the circuit as follows:

```
println(show(circuit))
```

We can generate the Verilog:

```
circuit.toVerilog("Filter", a)
```

We may create a simulator:

```
val avg = circuit.eval(a)
```

Now, we can simulate the circuit with input:

```
avg(10.toValue(8) :: Nil)
avg(20.toValue(8) :: Nil)
avg(20.toValue(8) :: Nil)
```

You should be able to see the following values:

```
VecV(List(0, 0, 0, 0, 0, 0, 1, 0))
VecV(List(0, 0, 0, 0, 1, 0, 1, 0))
VecV(List(0, 0, 0, 1, 0, 0, 0, 1))
```

4. Micro-controller

The implementation of the controller can be found in `src/main/scala/zeno/examples/Filter.scala`. The test assembly programs can be found in `asm/`.

The simplest way to test the microcontroller is to run it with simulation.

First start the interactive Scala console:

```
cd /home/zeno && sbt console
```

Now import packages:

```
import zeno.lang.*
import zeno.examples.controller.*
import scala.language.implicitConversions
```

Then we can run the simulation with a test assembly file as follows:

```
Controller.test("asm/jump.s")
```

You should be able to see the following string gets printed:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz
```

The method `Controller.test` performs the following steps:

1. Use the assembler to translate the test into machine code.
2. Contruct an instance of the microcontroller with the machine code in the ROM for instructions.
3. Create a circuit simulator for the microcontroller.
4. Drive the simulator bus with a simulated memory.
5. Accumulate all memory stores to address 0.
6. When program finishes or maximum iterations reached, return the accumulated writes to address 0.

You can play with other assembly files located under `asm/`.

We can generate Verilog for the micro-controller as follows:

```
val busIn = input[Controller.BusIn]("busIn")
val instructions = Assembler.assemble("asm/jump.s")
val circuit = Controller.processor(instructions, busIn)
circuit.toVerilog("Controller", busIn)
```

The generated code is large, and the output is truncated.