Chalk

A tool for architecture design

Wouter Swierstra
joint work with Koen Claessen, Carl Seger, Mary Sheeran, and Emily Shriver

Aim

- Try to design an architecture description language that:
  - can work at different levels of abstraction;
  - is capable of early estimations of performance and power;
  - builds on our functional programming expertise.

Behavioural

Structural

Behavioural

- Hawk (Cook, Launchbury, Matthews)
- Lava (Bjesse, Claessen, Sheeran, Singh)

Structural

Behavioural

- Hawk (Cook, Launchbury, Matthews)
- Lava (Bjesse, Claessen, Sheeran, Singh)
- Wired (Axelsson, Claessen, Sheeran)

Structural

Behavioural

- Hawk (Cook, Launchbury, Matthews)
- Chalk
- Lava (Bjesse, Claessen, Sheeran, Singh)
- Wired (Axelsson, Claessen, Sheeran)

Structural

Lava

- A data type for primitive gates (and, not,...);
- Describe the **structure** of circuits:

```
mux (c,t,e) = or2 (and2 (not c) t)
                  (and2 c e)
```

Lava

- Haskell combinators to assemble circuits (sequential composition, butterfly circuits, ...)
- Simulation and testing using QuickCheck;
- VHDL generation for circuits;
- Hooks into automatic theorem provers.

Hawk

- **Idea:** use Haskell as an executable hardware specification language.
- "Shallow embedding" – there is no separate data type to represent the AST.

Hawk - Signals

Signals assign values to every clock cycle:

```
type Signal a = [a]
```

Hawk combinators – I

Haskell functions to manipulate signals:

```
constant :: a -> Signal a
constant x = repeat x

lift :: (a -> b) -> Signal a -> Signal b
lift f signal = map f signal
```

Hawk combinators – II

```
delay :: a -> Signal a -> Signal a
delay x s = x :: s

mux :: Signal Bool
  -> Signal a -> Signal a -> Signal a
mux cs ts es = zipWith3 cond cs ts es
  where
  cond c t e = if c then t else e
```

Slightly non-trivial examples

- Hawk has been used to describe microprocessors
  - ALU and register files;
  - pipelining;
  - branch prediction.

Hawk review

- **Pro**: easy to write down executable specs;
- **Con**: you can’t do anything with these specs besides execute them.
  - No generating VHDL;
  - No automatic theorem proving;
  - No “non-functional” analysis.

Chalk

- Chalk is a Hawkish specification language that aims:
  - to provide more functionality than just executable specifications;
  - to support hierarchical architecture descriptions that can be refined incrementally.

A deeper embedding

```
data Circuit a where
 Pure :: a -> Circuit a
 App :: Circuit (b -> a) -> Circuit b
      -> Circuit a
 Delay :: a -> Circuit a -> Circuit a
 Component :: String -> Circuit a ->
           -> Circuit a
```

A deeper embedding

```
data Circuit a where
  Pure :: a -> Circuit a
  App :: Circuit (b -> a) -> Circuit b
       -> Circuit a
  Delay :: a -> Circuit a -> Circuit a
  Component :: String -> Circuit a ->
            -> Circuit a
```

I'll use an infix operator `<*>` instead of `App`

Example - mux

```
mux :: Circuit Bool ->
  Circuit a -> Circuit a -> Circuit a
mux cs ts es = component “Mux” $
  pure (\c t e -> if c then t else e)
    <*> cs
    <*> ts
    <*> es
```

ALU

```
data Cmd = ADD | SUB | INCR

alu :: Circuit Cmd ->
       Circuit (Int,Int) ->
       Circuit Int

alu cmds args = component “ALU” $
  pure eval <*> cmds <*> args
  where eval ADD  (x,y) = x + y
        eval SUB  (x,y) = x - y
        eval INCR (x,_) = x + 1
```

Register file

```
data Reg = R0 | R1 | R2 | R3

type Regs = (Int,Int,Int,Int)

regFile :: Signal (Reg, Int) ->
  Signal Reg -> Signal Reg ->
  (Signal Int, Signal Int)
regFile = loop initRegs regStep
  where
  loop :: s -> (s -> (a,s)) -> Signal a
  regStep :: Regs -> ((Int,Int), Regs)
```

Simple Hawk Microprocessor

- We can assemble these pieces:

```
sham :: (Signal Cmd, Signal Reg,
         Signal Reg, Signal Reg)
         -> (Signal Reg, Signal Int)
sham (cmds, destReg, srcA, srcB) = ...
```

- ... by using our register file to lookup the state of the source registers;
- and passing this on to the ALU.

Simulation

- It is easy to extract original Hawk signal functions:

```
simulate :: Circuit a -> [a]
simulate (Pure x) = repeat x
simulate (Delay x h) = x : simulate h
simulate (App f x) =
  zipWith ($) (simulate f) (simulate x)
```

Recap

- Hypothesis: writing specs using these combinators is no harder than in Hawk;
- ...but we now have more structure at our disposal.
- We can use this info to do other analyses.

Future work

- Circuit size;
- Graph visualisation;
- Symbolic performance analysis;
- Type-directed analyses;
- Non-standard interpretations;
- ...