What's the Matter with Kansas Lava?

Andrew Farmer, Garrin Kimmell, Andy Gill

The Information and Telecommunication Technology Center
The University of Kansas

May 18, 2010

KU

What do we want to do?

What do we want to do?

- Test circuit behavior.

KU

What do we want to do?

- Test circuit behavior.
- Generate documentation.

KU

What do we want to do?

- Test circuit behavior.
- Generate documentation.
- Sanity check code generation.

```
data Seq a = Seq (Stream a) AST
```

Do the two embeddings do the same thing?

KU

QuickCheck

Circuits are Haskell functions, so can we use QuickCheck?

KU

QuickCheck

Circuits are Haskell functions, so can we use QuickCheck?

```
prop_HAComm x y = (halfAdder x y) == (halfAdder y x)
    where types = (x :: Comb Bool, y :: Comb Bool)
```

KU

QuickCheck

Circuits are Haskell functions, so can we use QuickCheck?

```
prop_HAComm x y = (halfAdder x y) == (halfAdder y x)
    where types = (x :: Comb Bool, y :: Comb Bool)
```

- Hard to control input.
- Equality for `Seq`?
- No observable intermediate values.
- Doesn't test code generation.

KU

QuickCheck

Circuits are Haskell functions, so can we use QuickCheck?

```
prop_HAComm x y = (halfAdder x y) == (halfAdder y x)
    where types = (x :: Comb Bool, y :: Comb Bool)
```

- Hard to control input.
- Equality for `Seq`?
- No observable intermediate values.
- Doesn't test code generation.

```
instance Eq a => Eq (Comb a) where
    (Comb x _) == (Comb y _) = lhs == rhs
        where lhs = (unX x :: Maybe a)
              rhs = (unX y :: Maybe a)
```

```
instance Eq a => Eq (Seq a) where
    (Seq x _) == (Seq y _) = error "undefined"
```

KU

Observing Intermediate Values

```
fullAdder a b cin = (sum,cout)
  where (s1,c1) = halfAdder a a
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
```

Observing Intermediate Values

```
fullAdder a b cin = ((sum,cout),debug)
  where (s1,c1) = halfAdder a a
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
        debug = (s1,c1,c2)
```

Observing Intermediate Values

```
fullAdder a b cin = ((sum,cout),debug)
  where (s1,c1) = halfAdder a a
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
        debug = (s1,c1,c2)
```

- This changes the circuit interface, requiring all users of the `fullAdder` to change how they call it.

KU

Observing Intermediate Values

```
fullAdder a b cin = ((sum,cout),debug)
  where (s1,c1) = halfAdder a a
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
        debug = (s1,c1,c2)
```

- This changes the circuit interface, requiring all users of the `fullAdder` to change how they call it.
- It's also incredibly time consuming and error-prone.

KU

Probes

```
class Probe a where
    probe :: String -> a -> a
```

Probes

```
class Probe a where
    probe :: String -> a -> a
```

```
fullAdder a b cin = (sum,cout)
  where (s1,c1) = (probe "ha1" halfAdder) a a
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
```

Probes

```
class Probe a where
    probe :: String -> a -> a
```

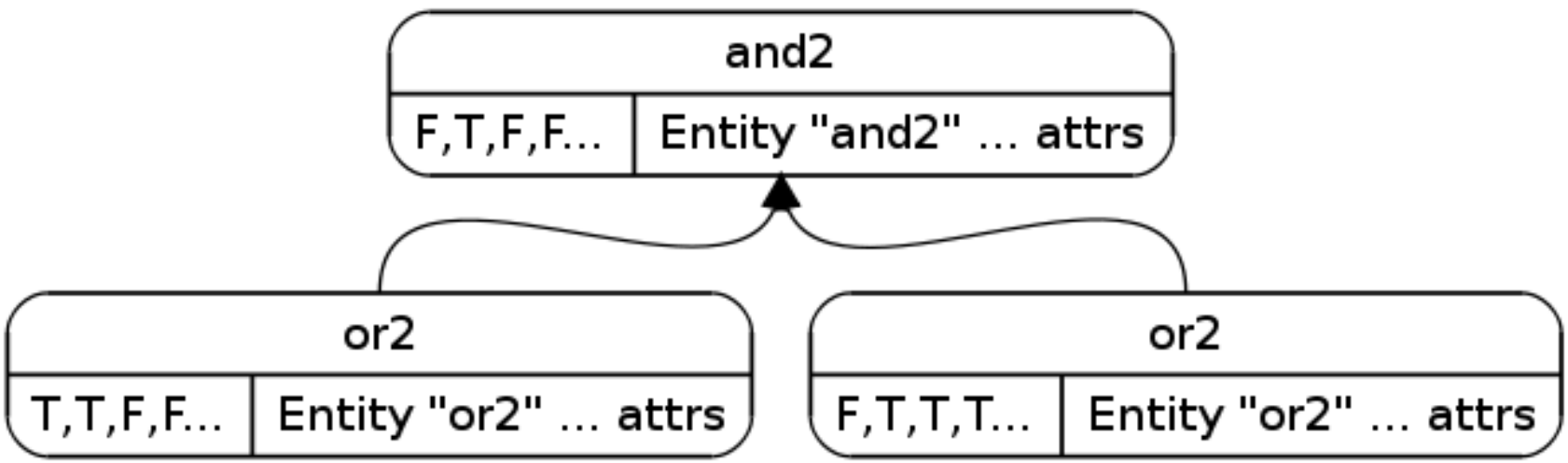
```
fullAdder a b cin = (sum,cout)
  where (s1,c1) = (probe "ha1" halfAdder) a a
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
```

```
> test $ fullAdder false true false
ha1_0: "F" :~ "F" :~ "F" :~ "F" :~ ...
ha1_1: "F" :~ "F" :~ "F" :~ "F" :~ ...
ha1_2: "(F,F)" :~ "(F,F)" :~ "(F,F)" :~ "(F,F)" :~...
```

KU

Probe Implementation

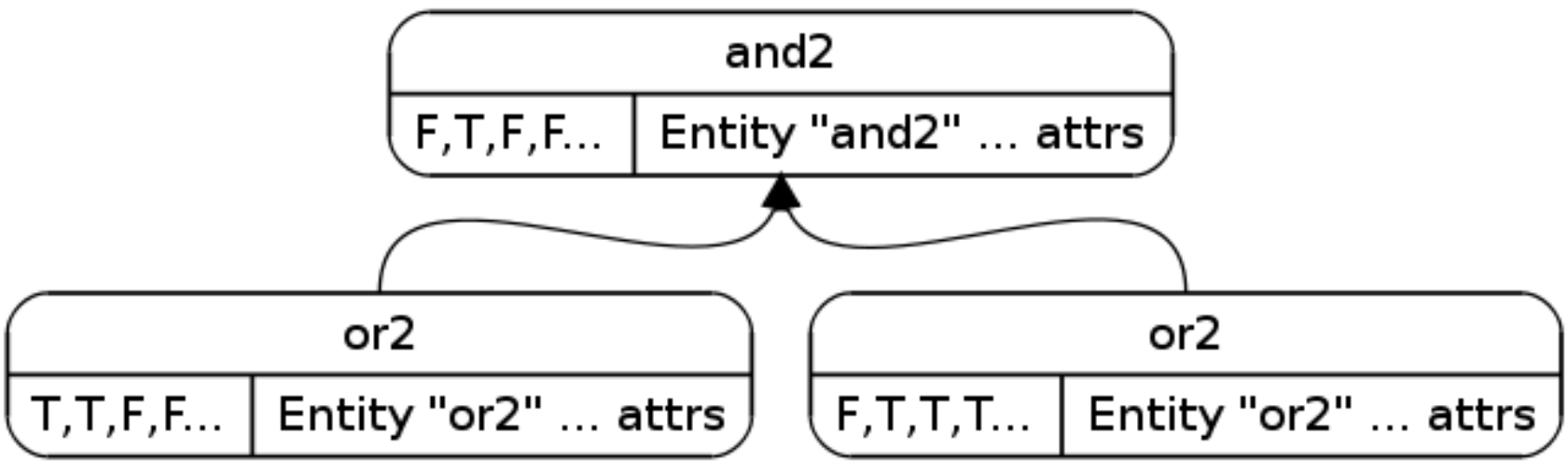
```
and2 :: (Signal sig) => sig Bool -> sig Bool -> sig Bool
and2 = liftSig2 (\ (Comb a ae) (Comb b be) ->
                          Comb (a && b)
                               (Entity "and2" [ae,be]))
```



KU

Probe Implementation

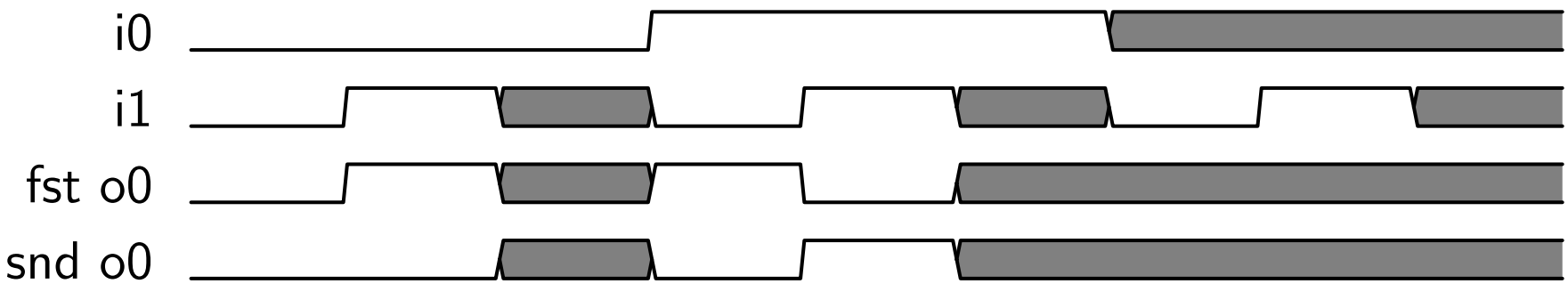
```
and2 :: (Signal sig) => sig Bool -> sig Bool -> sig Bool
and2 = liftSig2 (\ (Comb a ae) (Comb b be) ->
                          Comb (a && b)
                               (Entity "and2" [ae,be] toDyn(a && b)))
```



KU

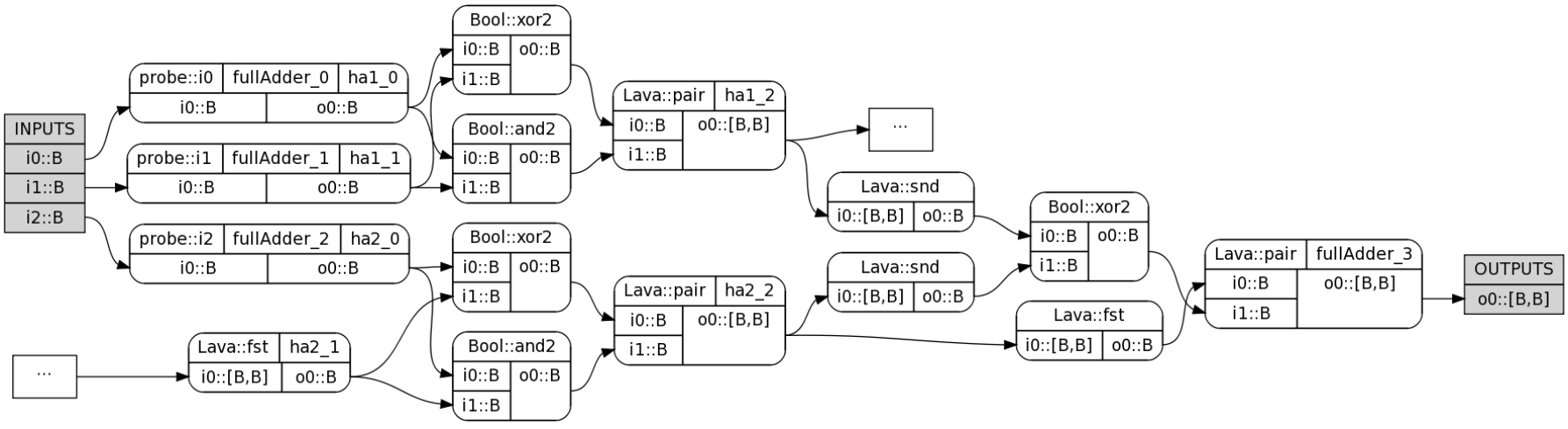
Putting Probes to Work

```
fullAdder a b cin = (sum,cout)
  where (s1,c1) = (probe "ha1" halfAdder) a b
        (sum,c2) = halfAdder cin s1
        cout = xor2 c1 c2
```

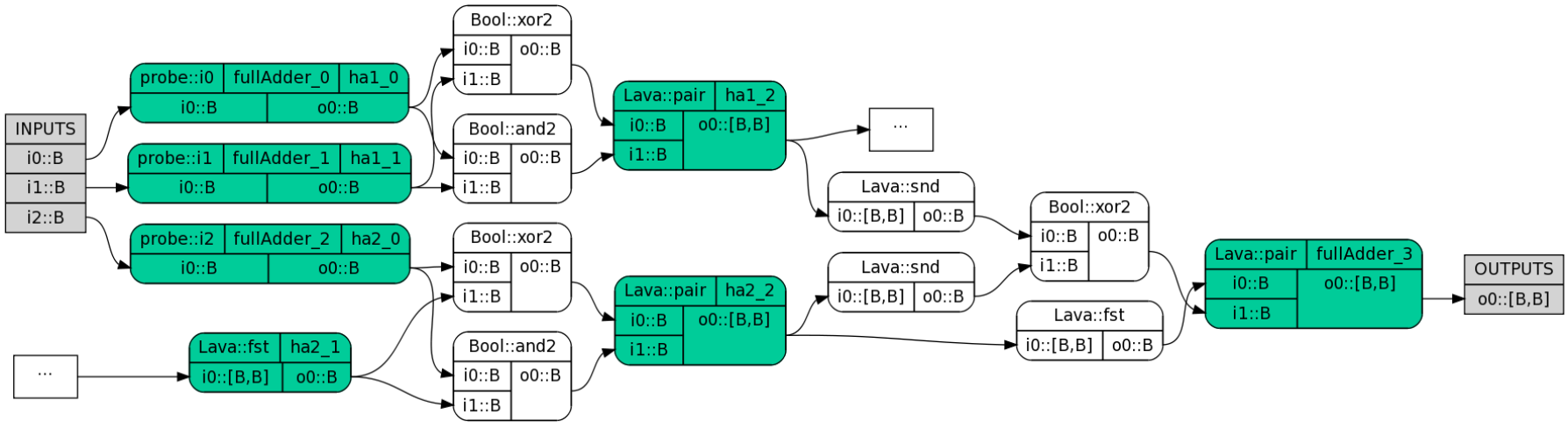



```
halfAdder ::  Seq Bool → Seq Bool → (Seq Bool, Seq Bool)
```

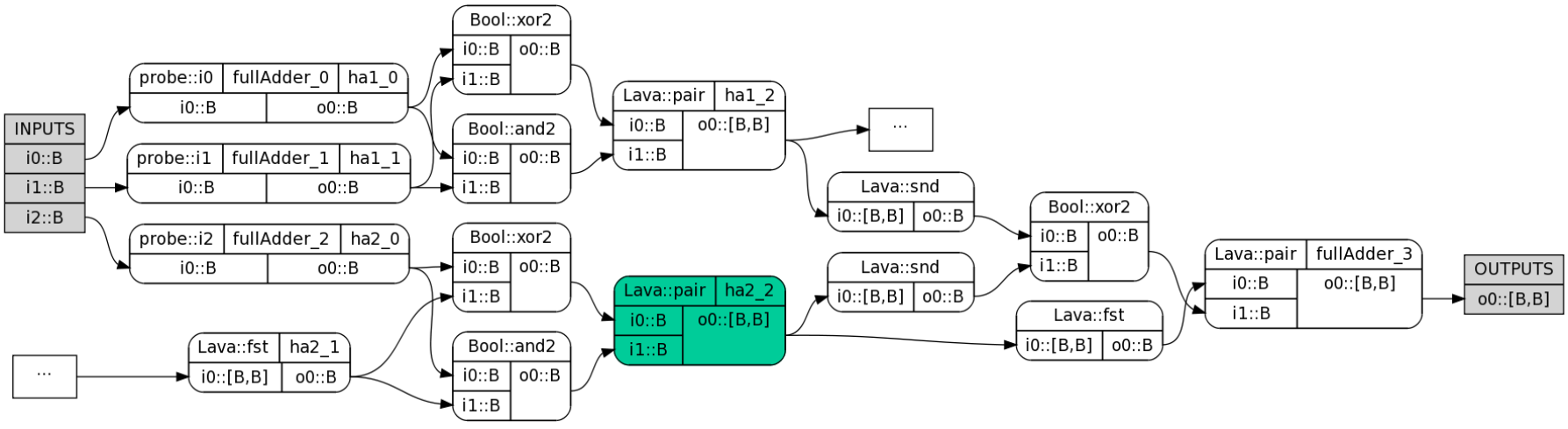
Comparing Deep and Shallow



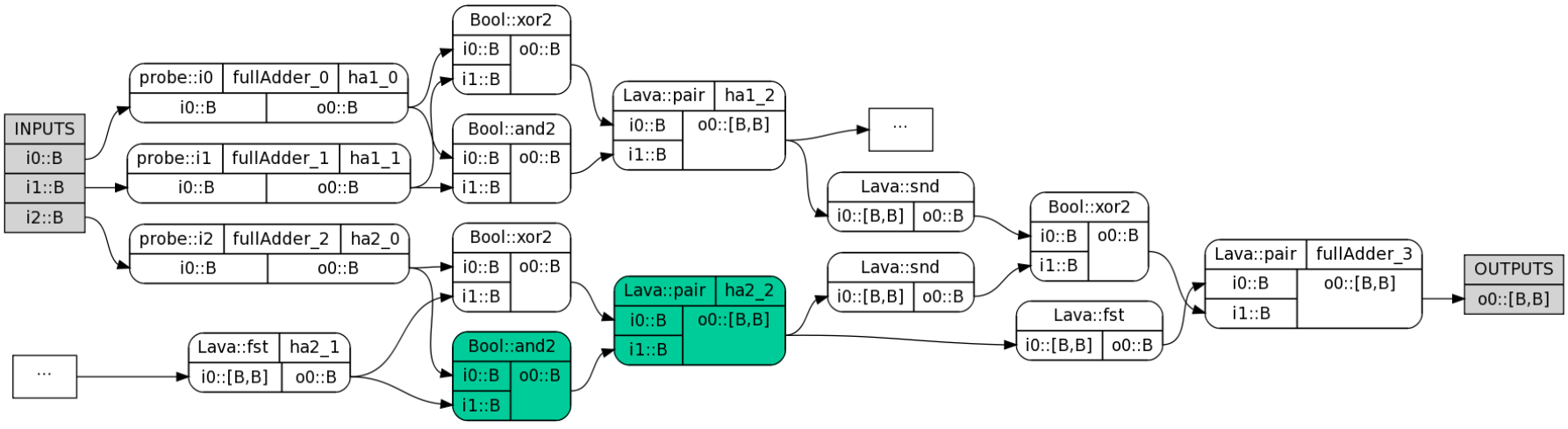
Comparing Deep and Shallow



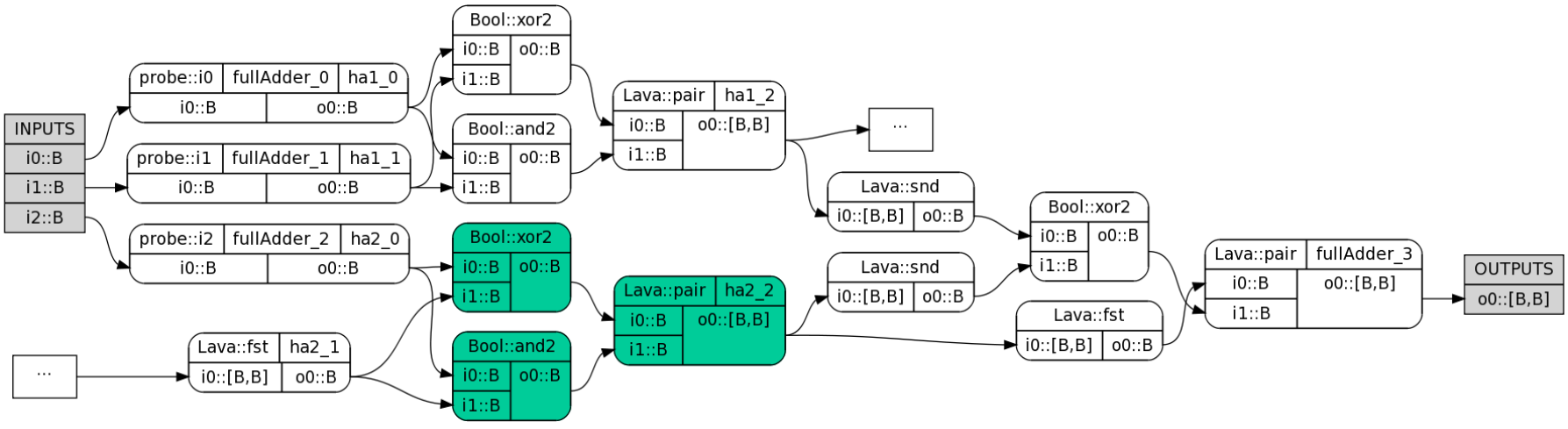
Comparing Deep and Shallow



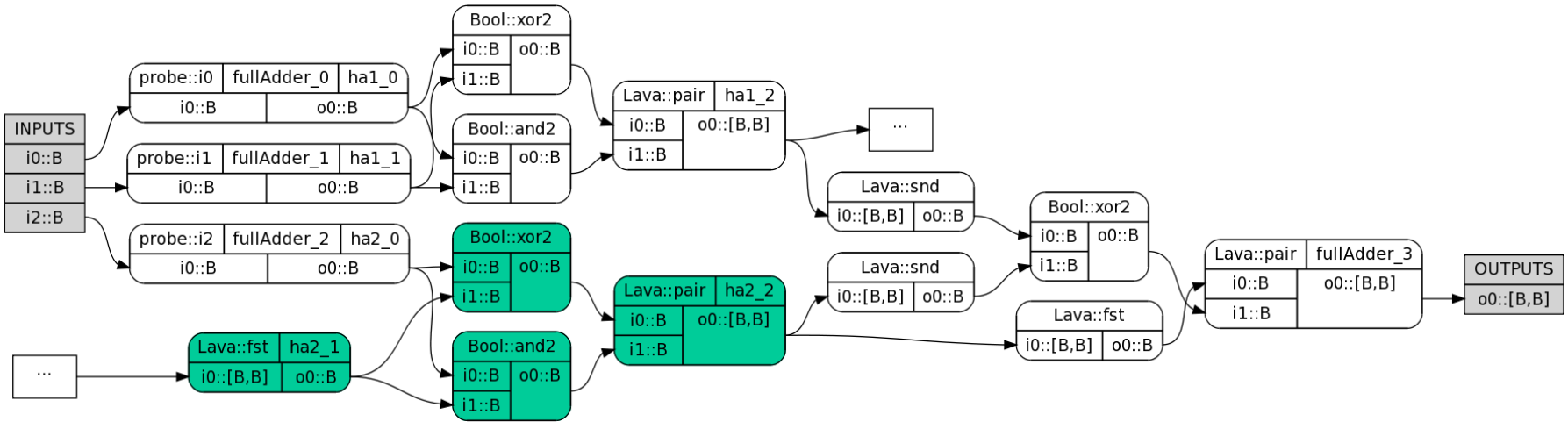
Comparing Deep and Shallow



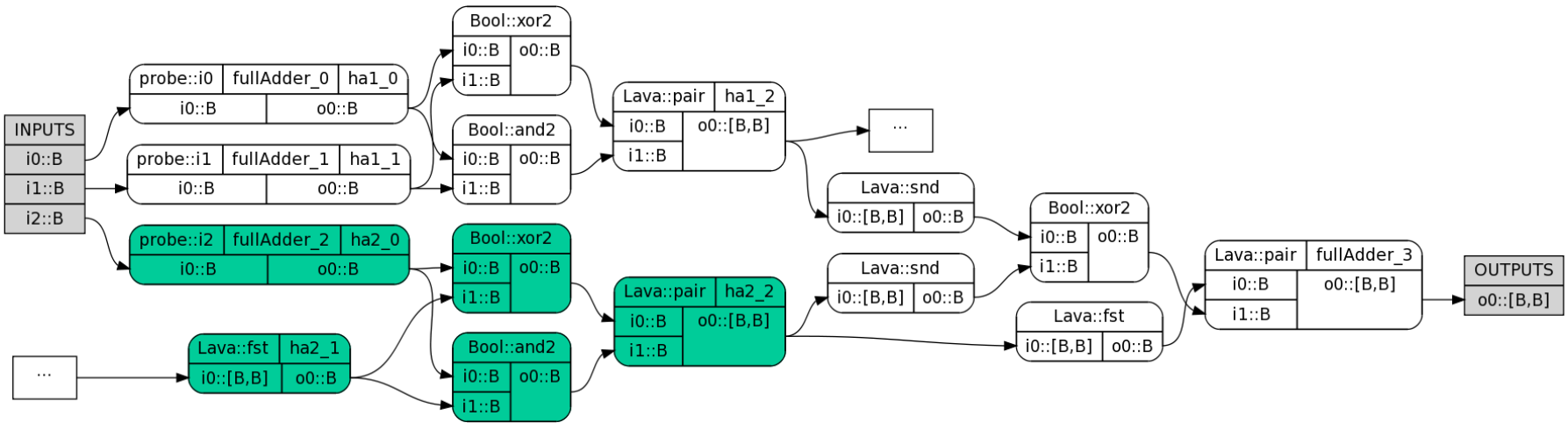
Comparing Deep and Shallow



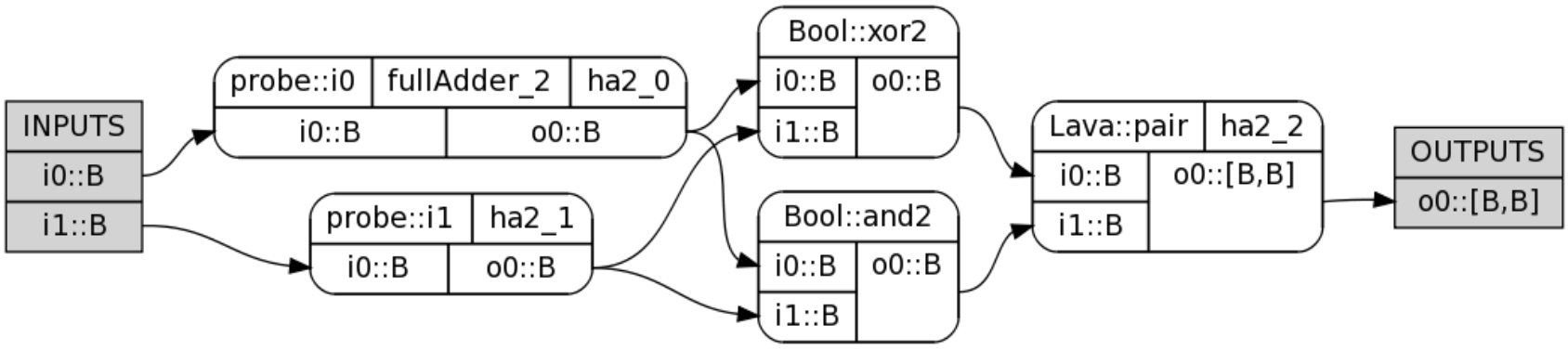
Comparing Deep and Shallow



Comparing Deep and Shallow



Comparing Deep and Shallow



Testbench

Using the probe data and the extracted subcircuit, we can generate a testbench in VHDL and run it with modelsim:

Testbench

Using the probe data and the extracted subcircuit, we can generate a testbench in VHDL and run it with modelsim:

ha2.shallow

```
0000
1110
0101
0000
1001
0101
0101
...
```

Testbench

Using the probe data and the extracted subcircuit, we can generate a testbench in VHDL and run it with modelsim:

ha2.shallow

```
0000
1110
0101
0000
1001
0101
0101
...
```

ha2.info

```
(0) F/0 -> F/0 -> (F,F)/00
(1) T/1 -> T/1 -> (F,T)/10
(2) F/0 -> T/1 -> (T,F)/01
(3) F/0 -> F/0 -> (F,F)/00
(4) T/1 -> F/0 -> (T,F)/01
(5) F/0 -> T/1 -> (T,F)/01
(6) F/0 -> T/1 -> (T,F)/01
...
```

KU

Future Work

Future Work

- It's cumbersome to examine large circuits at the level of primitive entities. Probes would allow us to group parts of the circuit and visualize at a higher level.

KU

Future Work

- It's cumbersome to examine large circuits at the level of primitive entities. Probes would allow us to group parts of the circuit and visualize at a higher level.
- Kansas Lava allows the importation of existing VHDL blocks. We could use this system in reverse to check the shallow embedding.

KU

Future Work

- It's cumbersome to examine large circuits at the level of primitive entities. Probes would allow us to group parts of the circuit and visualize at a higher level.
- Kansas Lava allows the importation of existing VHDL blocks. We could use this system in reverse to check the shallow embedding.
- A semi-automated algorithmic debugger.

KU

Future Work

- It's cumbersome to examine large circuits at the level of primitive entities. Probes would allow us to group parts of the circuit and visualize at a higher level.
- Kansas Lava allows the importation of existing VHDL blocks. We could use this system in reverse to check the shallow embedding.
- A semi-automated algorithmic debugger.
- Testing optimizations.

KU

Future Work

- It's cumbersome to examine large circuits at the level of primitive entities. Probes would allow us to group parts of the circuit and visualize at a higher level.
- Kansas Lava allows the importation of existing VHDL blocks. We could use this system in reverse to check the shallow embedding.
- A semi-automated algorithmic debugger.
- Testing optimizations.
- A dataflow visualizer.

KU

Questions?

KU