

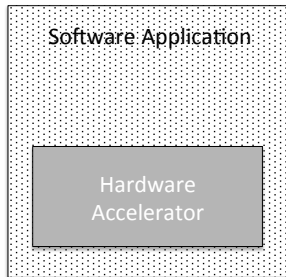
The background of the slide is a silhouette of a city skyline at sunset. The sky is a warm orange color, and the city buildings are dark silhouettes against it. The skyline includes several prominent buildings, including a tall, thin tower on the left and a large, multi-towered building on the right. The water in the foreground reflects the orange light of the sunset.

Model-driven Design & Synthesis of the SHA-256 Cryptographic Hash Function in ReWire

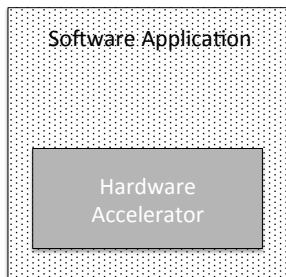
Bill Harrison *University of Missouri* Adam Procter *Intel Corp.*
Gerard Allwein *US Naval Research Laboratory*

October 7, 2016

Challenge: **High Assurance** Hardware Accelerators



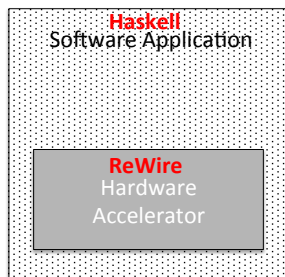
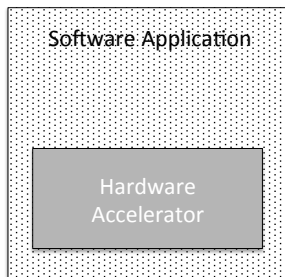
Challenge: **High Assurance** Hardware Accelerators



“Challenge”?

- ▶ Two different languages: SW & HDL
- ▶ Neither (typically) with formal semantics supporting verification

Challenge: **High Assurance** Hardware Accelerators



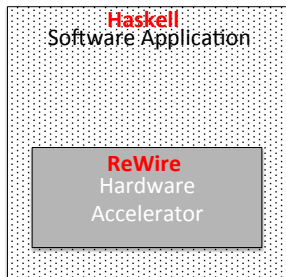
“Challenge”?

- ▶ Two different languages: SW & HDL
- ▶ Neither (typically) with formal semantics supporting verification

Approach

- ▶ Write in Haskell
- ▶ Transform acceleration target into ReWire
- ▶ Verify accelerator with Haskell semantics

Case Study: **High Assurance** SHA-256 HW Accelerator



Approach

- Write in Haskell
- Transform acceleration target into ReWire
- Verify accelerator with Haskell semantics

- Crypto-algorithms good candidates for both
 - hardware acceleration
 - formal verification
- SHA-256 (Secure Hash Algorithm) defined as pseudo-code [NIST02]:

Preprocessing

⟨Parse/Pad as N 512 bit blocks⟩

Main Loop

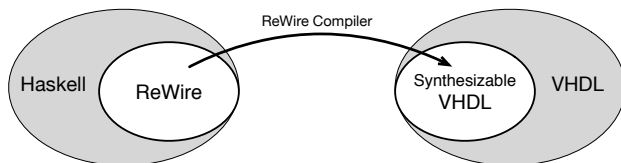
For 1 to N :

⟨do some stuff⟩

Inner Loop

For 0 to 63 : ⟨other stuff⟩

ReWire Functional Hardware Description Language



- ▶ Inherits Haskell's good qualities
 - ▶ Pure functions & types, monads, equational reasoning, etc.
 - ▶ Formal denotational semantics [HarrisonKieburz05,Harrison05]
- ▶ Types & operators for HW abstractions & clocked/parallel computations.
 - ▶ Organizing principle: monads, esp. "reactive resumption monad"
 - ▶ Very familiar ideas to functional programming community

Reference Semantics

```
sha256 :: [Hex Word32] -> M (Oct Word32)
sha256 hws = do
  putDigest initialSHA256State
  mainloop hws
  getDigest
```

```
mainloop :: [Hex Word32] -> M ()
mainloop [] = return ()
mainloop (hw32 : hw32s) = do
  hi1 <- getDigest
  putIntDig hi1
  putBlock hw32
  putCtr C0
  innerloop
  mainloop hw32s
```

```
innerloop :: M ()
innerloop = do
  c <- getCtr
  s <- sched
  compress (seed c) s
  putCtr (incCtr c)
  case c of
    C63 -> intermediate
    _    -> innerloop
```

- Straightforward Formalization of Pseudocode from NIST Document
- Can be tested:

```
GHC> run_sha256 msg1
Oct 3128432319 2399260650
    1094795486 1571693091
    2953011619 2518121116
    3021012833 4060091821
GHC> hashed1
Oct 3128432319 2399260650
    1094795486 1571693091
    2953011619 2518121116
    3021012833 4060091821
    :
    :
```

Reference Semantics

```
sha256 :: [Hex Word32] -> M (Oct Word32)
```

```
sha256 hws = do
  putDigest initialSHA256State
  mainloop hws
  getDigest
```

```
mainloop :: [Hex Word32] -> M ()
mainloop [] = return ()
mainloop (hw32 : hw32s) = do
  hi1 <- getDigest
  putIntDig hi1
  putBlock hw32
  putCtr C0
  innerloop
  mainloop hw32s
```

```
innerloop :: M ()
innerloop = do
  c <- getCtr
  s <- sched
  compress (seed c) s
  putCtr (incCtr c)
  case c of
    C63 -> intermediate
    _    -> innerloop
```

Lifted Semantics

```
dev :: Inp -> ReT Inp Out M ()
dev (Init hw32) = do
  lift (do putDigest initialSHA256State
    hi1 <- getDigest
    putIntDig hi1
    putBlock hw32
    putCtr C0)
```

```
  signal Nix
  innerloop
dev (Load hw32) = do
  lift (do hi1 <- getDigest
    putIntDig hi1
    putBlock hw32
    putCtr C0)
```

```
  signal Nix
  innerloop
dev DigestQ = do
  hn <- lift getDigest
  i <- signal (DigestR hn)
  dev i
```

```
innerloop :: ReT Inp Out M ()
innerloop = do
  c <- lift (do c <- getCtr
    s <- sched
    compress (seed c) s
    putCtr (incCtr c)
    return c)
  i <- signal Nix
  case c of
    C63 -> lift intermediate >> dev i
    _    -> innerloop
```


Reference Semantics

```
sha256 :: [Hex Word32] -> M (Oct Word32)
```

```
sha256 hws = do
```

```
    putDigest initialSHA256State
```

```
    mainloop hws
```

```
    getDigest
```

```
mainloop :: [Hex Word32] -> M ()
```

```
mainloop [] = return ()
```

```
mainloop (hw32 : hw32s) = do
```

```
    hi1 <- getDigest
```

```
    putIntDig hi1
```

```
    putBlock hw32
```

```
    putCtr C0
```

```
    innerloop
```

```
    mainloop hw32s
```

```
innerloop :: M ()
```

```
innerloop = do
```

```
    c <- getCtr
```

```
    s <- sched
```

```
    compress (seed c) s
```

```
    putCtr (incCtr c)
```

```
    case c of
```

```
        C63 -> intermediate
```

```
        _ -> innerloop
```

Lifted Semantics

```
dev :: Inp -> ReT Inp Out M ()
```

```
dev (Init hw32) = do
```

```
    lift (do putDigest initialSHA256State
```

```
            hi1 <- getDigest
```

```
            putIntDig hi1
```

```
            putBlock hw32
```

```
            putCtr C0)
```

```
    signal Nix
```

```
    innerloop
```

```
dev (Load hw32) = do
```

```
    lift (do hi1 <- getDigest
```

```
            putIntDig hi1
```

```
            putBlock hw32
```

```
            putCtr C0)
```

```
    signal Nix
```

```
    innerloop
```

```
dev DigestQ = do
```

```
    h_n <- lift getDigest
```

```
    i <- signal (DigestR h_n)
```

```
    dev i
```

```
innerloop :: ReT Inp Out M ()
```

```
innerloop = do
```

```
    c <- lift (do c <- getCtr
```

```
                s <- sched
```

```
                compress (seed c) s
```

```
                putCtr (incCtr c)
```

```
                return c)
```

```
    i <- signal Nix
```

```
    case c of
```

```
        C63 -> lift intermediate >> dev i
```

```
        _ -> innerloop
```

Evaluation: Testing, Formal Specification, & Performance

► Testing

```
GHC> run_dev256 msg1
      Oct 3128432319 2399260650
      ...
GHC> hashed1
      Oct 3128432319 2399260650
      ...
```

Evaluation: Testing, Formal Specification, & Performance

► Testing

```
GHC> run_dev256 msg1
      Oct 3128432319 2399260650
      ...
GHC> hashed1
      Oct 3128432319 2399260650
      ...
```

► Formal Specification

For all finite `str :: String`,
DigestR (run_sha256 str)
= run_dev256 str

- Proof not in paper; similar specs proved in [TECS16], [FPT15], [LCTES15]

Evaluation: Testing, Formal Specification, & Performance

► Testing

```
GHC> run_dev256 msg1
      Oct 3128432319 2399260650
      ...
GHC> hashed1
      Oct 3128432319 2399260650
      ...
```

► Performance

- For Spartan-3E w/ Xilinx ISE, max clock rate = 60 MHz.
Total throughput = 404 Mbps.

Slices	Flip-Flops	LUTs	IOBs
1424 (30%)	1106 (11%)	2716 (29%)	134 (57%)

- In line with published, hand-written VHDL implementations of SHA-256: [Sklavos 2005]-[Kahri et al. 2015]


► Formal Specification

For all finite `str :: String`,
`DigestR (run_sha256 str)`
`= run_dev256 str`

- Proof not in paper; similar specs proved in [TECS16], [FPT15], [LCTES15]

Summary; Related & Future Work

- ▶ Appel [TOPLAS15] verifies an entire C implementation of SHA-256
 - ▶ We have only formally specified HW accelerator
 - ▶ Need “Foreign Device Interface” to link Haskell & ReWire
- ▶ High assurance relies on semantically-faithful compiler
 - ▶ Mechanization in Coq; Compiler Verification
- ▶ Functional Hardware Description: Chisel, Lava, etc.; Synchronous & Imperative: Esterel
- ▶ Rewire is open source:
<https://github.com/mu-chaco/ReWire>

 = Future Work

*This research supported by the US National Science Foundation CAREER Award #0746509 and the US Naval Research Laboratory.

Recent ReWire Publications



I. Graves, A. Procter, W. Harrison, M. Becchi, and G. Allwein.
Hardware synthesis from functional embedded domain-specific languages:
A case study in regular expression compilation.
In Proceedings of Applied Reconfigurable Computing 2015.



I. Graves, A. Procter, W. Harrison, and G. Allwein.
Provably correct development of reconf. HW designs via eq. reasoning.
In Proceedings of Field-Programmable Tech. 2015.



W. Harrison, A. Procter, I. Graves, M. Becchi, and G. Allwein.
A programming model for reconf. computing based in funct. concurrency.
In Proceedings of ReCoSoC 2016.



A. Procter, W. Harrison, I. Graves, M. Becchi, and G. Allwein.
A principled approach to secure multi-core processor design with ReWire.
ACM Trans. on Embedded Computing Systems (to appear), 2016.



A. Procter, W. Harrison, I. Graves, M. Becchi, and G. Allwein.
Semantics driven hardware design, implementation, & verif. with ReWire.
In Proceedings of LCTES 2015.