HIGH ASSURANCE HARDWARE WITH REWIRE

Just Say No! to Semantic Archaeology

Bill Harrison, Adam Procter, Ian Graves, & Michela Becchi
University of Missouri

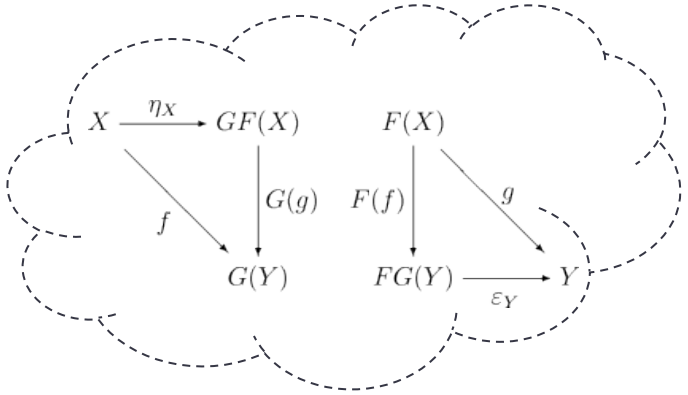
Gerard Allwein
US Naval Research Laboratory

Archaeology

Semantic Archaeology & Formal Methods

System Implementation

Mathematical System Model formulating precise system security properties



Semantic Archaeology as It Occurs in Nature

From Sarkar, et al., Semantics of x86-CC Multiprocessor Machine Code, POPL09

*"The key difficulty was to go from the **informal-prose vendor documentation**, with its **often-tantalising ambiguity**, to a **fully rigorous definition (mechanised in HOL)** that one can be **reasonably confident is an accurate reflection** of the vendor architectures (Intel 64 and IA-32, and AMD64)."*

x86 Instruction Semantics in HOL in terms of Monadic Microcode; e.g.,

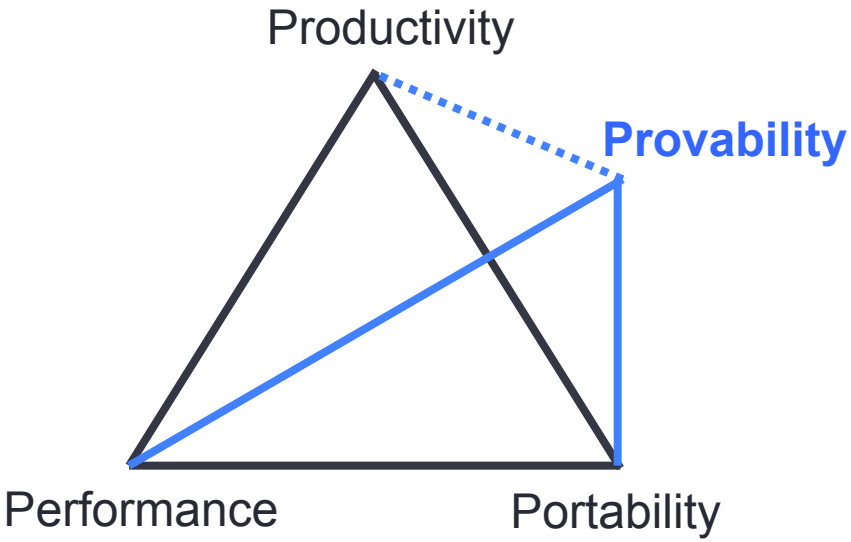
$$\begin{aligned}
&\mathrm{seqT} : {'a}\ \mathsf{M} \to ({'a} \to {'b}\ \mathsf{M}) \to {'b}\ \mathsf{M} \\
&\mathrm{parT} : {'a}\ \mathsf{M} \to {'b}\ \mathsf{M} \to ({'a} * {'b})\mathsf{M} \\
&\mathrm{constT} : {'a} \to {'a}\ \mathsf{M} \\
&\mathrm{failureT} : \mathsf{unit}\ \mathsf{M} \\
&\mathrm{mapT} : ({'a} \to {'b}) \to {'a}\ \mathsf{M} \to {'b}\mathsf{M} \\
&\mathrm{lockT} : \mathsf{unit}\ \mathsf{M} \to \mathsf{unit}\ \mathsf{M}
\end{aligned}$$

Security Flows in the Many Core Era*

- Highly (Re)configurable Architectures/FPGAs
- Many Specially Tailored, “One Off” Components
  - Reuse of Off-the-shelf components
  - “Mix and Match” comes to Hardware
- Challenge: High Assurance in this environment
  - Want the flexibility and speed of development
  - …but also want formal guarantees of security and safety for critical systems.

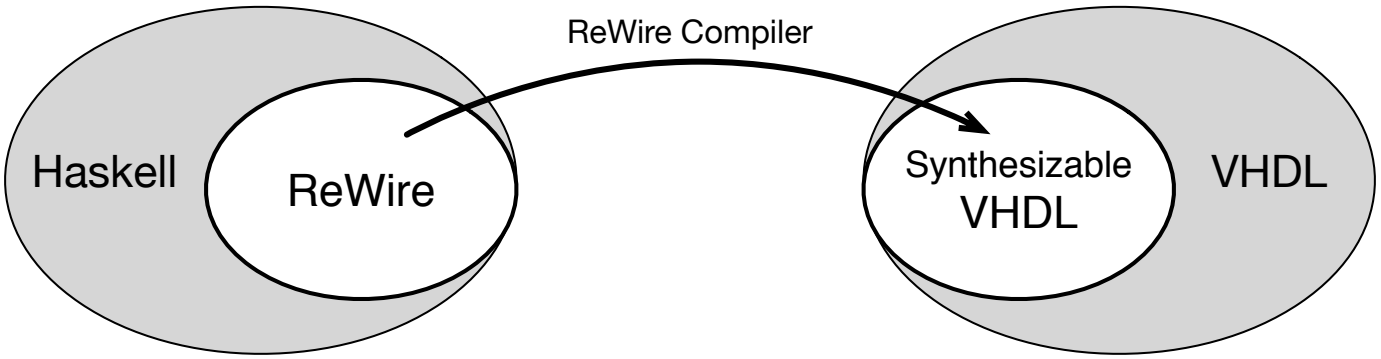
* Funded by the Office of the Assistant Secretary of Defense for Research and Engineering

Hardware Synthesis from Domain Specific Languages



- Delite [Olukotun, Ienne, et al.]
  - DSLs and Language Virtualization
  - "The Three P's"
- ReWire
  - Fourth P: *Provability*
  - DSL with rigorous semantics
    - Modular Monadic Semantics
  - High assurance
    - Security & safety properties
    - Formal methods Productivity

ReWire Language & Toolchain

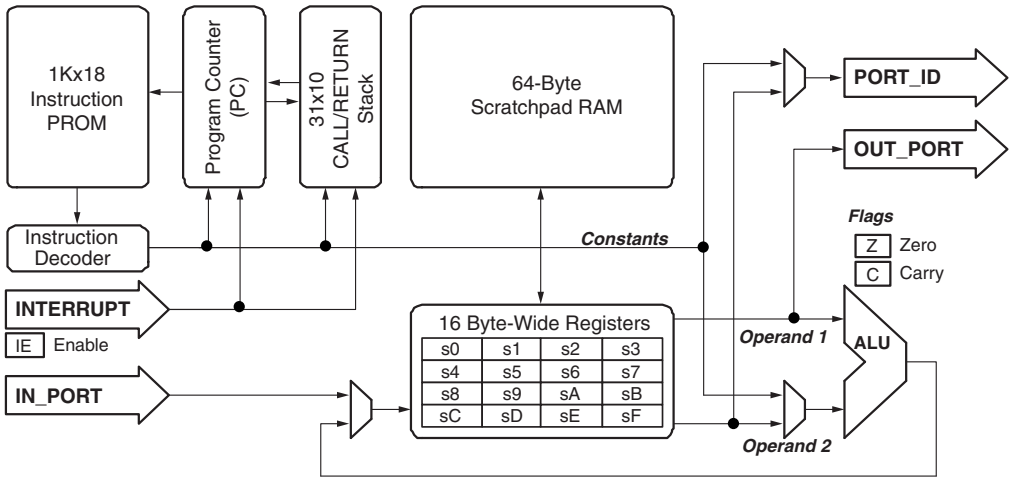


- Inherits Haskell's good qualities
  - Pure functions, strong types, monads, equational reasoning, etc.
  - Formal denotational semantics [HarrisonKieburtz05,Harrison05]
- Language design identifies HW representable programs
  - Mainly restrictions on recursion in functions and data
  - Built-in types for HW abstractions incl. clocked/parallel computations

Expressing Architectural Designs in ReWire

Details in "*Semantics-directed Architecture in ReWire*", Procter et al., ICFPT13

Xilinx PicoBlaze Architecture



PicoBlaze Data Layout in ReWire

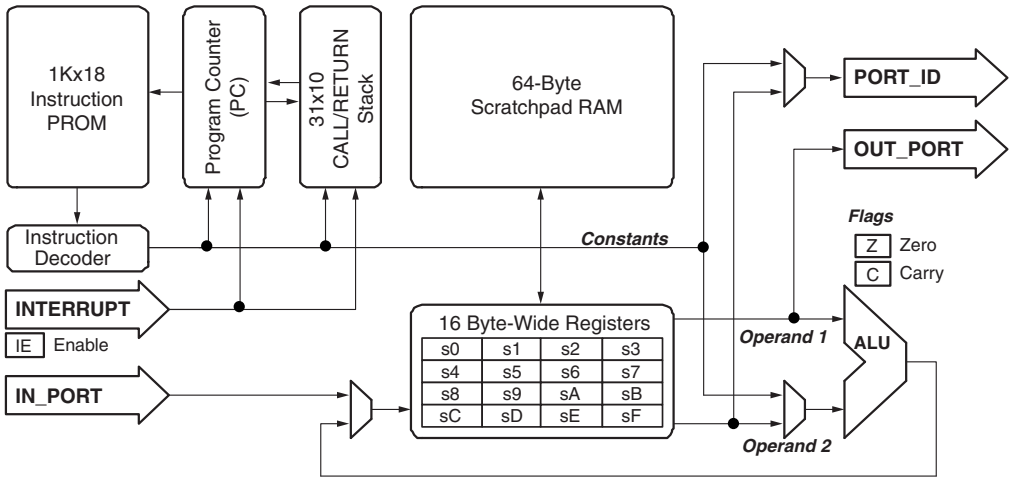
```
type RegFile   = Table W4 W8
type FlagFile  = (Bit,Bit,Bit,Bit,Bit)
type Mem       = Table W6 W8
data Stack     = Stack { contents :: Table W5 W10,
                         pos :: W5 }
data Inputs    = Inputs { instruction_in :: W18,
                          in_port_in     :: W8,
                          interrupt_in   :: Bit,
                          reset_in       :: Bit }
data Outputs   = Outputs { address_out       :: W10,
                           port_id_out       :: W8,
                           write_strobe_out  :: Bit,
                           out_port_out      :: W8,
                           read_strobe_out   :: Bit,
                           interrupt_ack_out :: Bit }
```

Expressing Architectural Designs in ReWire (cont'd)

Details in "*Semantics-directed Architecture in ReWire*", Procter et al., ICFPT13

- **`fde`** device is tail-recursive
- Clock timing is expressed in **`Dev`** monad

Xilinx PicoBlaze Architecture




```
fde :: Dev Inputs PicoState Outputs
fde = do s <- getPicoState
         let i     = inputs s
             instr = instruction_in i
         ie <- getFlagIE
         if reset_in i == 1
            then reset_event
            else if ie == 1 &&
                    interrupt_in i == 1
                       then interrupt_event
                       else decode instr
         fde
```

PicoBlaze Fetch-Decode-Execute in ReWire

Compare with PicoBlaze in VHDL

- Outermost VHDL Component for PicoBlaze

```
component KCPSM3
port (
      instruction   : in std_logic_vector(17 downto 0);  -- Inputs type
      in_port       : in std_logic_vector( 7 downto 0);
      interrupt     : in std_logic;
      reset         : in std_logic;
      clk           : in std_logic;
      address       : out std_logic_vector( 9 downto 0); -- Outputs type
      port_id       : out std_logic_vector( 7 downto 0);
      write_strobe  : out std_logic;
      out_port      : out std_logic_vector( 7 downto 0);
      read_strobe   : out std_logic;
      interrupt_ack : out std_logic;
     );
end component;
```

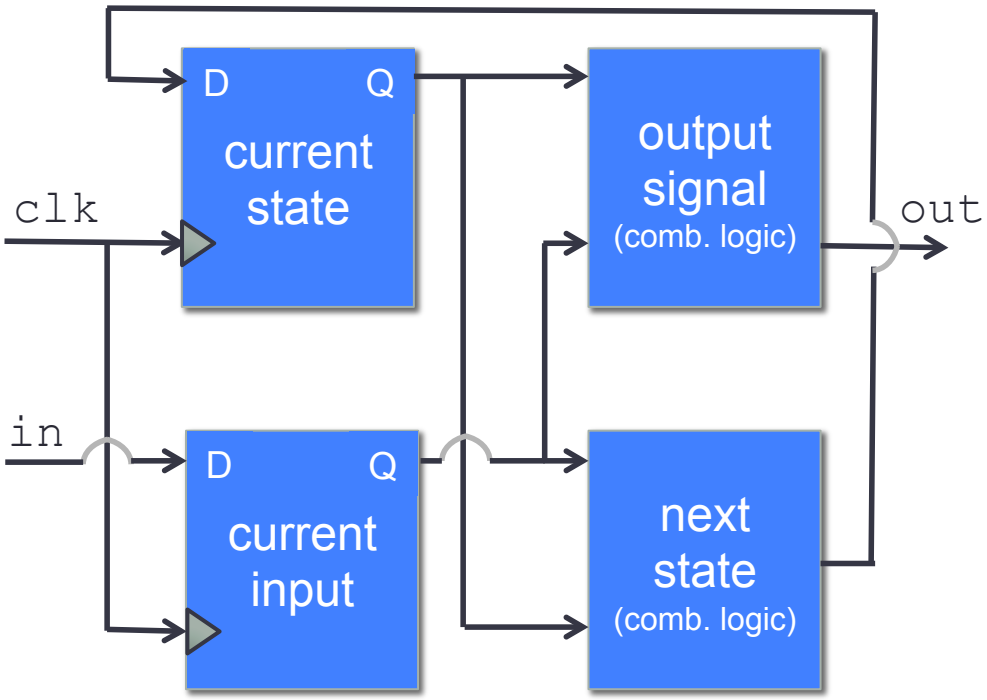
- Corresponds to ReWire term of monadic type
  - **`Dev Inputs PicoState Outputs`**

Crucial Distinction:

`Dev` is a formal object we can reason about.

Types for Devices

Terms of type `(Dev i s o)` compiled to…




```
newtype ReT i o m a = ReT (m (Either a (o, i → ReT i o m a))
newtype StT s m a   = …
```

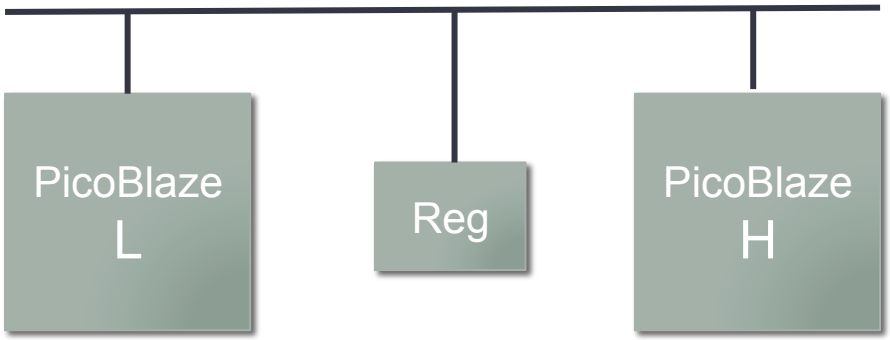
```
type Dev i s o = ReT i o (StT s Identity) ()
```

Performance

- Prototype ReWire compiler vs. Hand-coded VHDL implementation by experienced Xilinx engineer.
  - XST synthesis tool for Spartan-3E XC3S500E, speed -4
  - configured to optimize for speed, not space.

	Slices	Flip Flops	4-LUTs	F_{max} (MHz)
PicoBlaze	99	76	181	139.919
ReWire	451	110	866	69.956

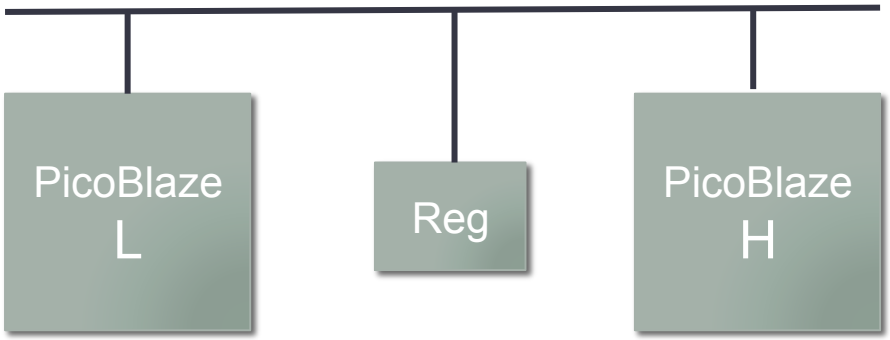
Designing a Secure Dual-core PicoBlaze*



- Two PicoBlazes (L ≤ H) with a shared register Reg
  - Reg is read-only by H; read+write by L
- Proved a non-interference style security specification
  - Equational proof based on “by-construction” properties of monads
  - Verifies ReWire code directly
  - Just say NO! to Semantic Archaeology.

* Details in Procter, et al., “Semantics Driven Hardware Design, Implementation and Verification in ReWire”, LCTES 2015 (to appear).

Designing a Secure Dual-core PicoBlaze*



- Type of Dual-Core constructor function:

```
dualcore :: Dev Inputs PicoState Outputs ->
            Dev Inputs PicoState Outputs ->
            Dev2 Inputs PicoState Outputs
```

* Details in Procter, et al., “Semantics Driven Hardware Design, Implementation and Verification in ReWire”, LCTES 2015 (to appear).

Security Theorem

$$\texttt{pull os is (dualcore lo } \mathbf{\texttt{hi}}\texttt{) >>= } \kappa_0$$
$$\texttt{= pull os is (dualcore lo } \mathbf{\texttt{nop}}\texttt{) >>= } \kappa_0$$

where

$$\kappa_0 = \lambda \texttt{os. mask}_H \texttt{ >> return os}$$
$$\texttt{nop} = \texttt{(skip } o_0 \ i_0\texttt{)}$$

Proof follows closely:

Harrison & Hook, “Achieving Information Flow Security Through Monadic Control of Effects”, Journal of Computer Security 2009

Proof Sketch of Security Theorem

$$
\begin{aligned}
&\mathit{pull}\ \mathit{os}\ [i_1,\ldots,i_n]\ (\mathit{dualcore}\ \mathit{lo}\ \mathit{hi})\ \texttt{>>=}\ \lambda \mathit{os}.\ \mathit{mask}_H\ \texttt{>>}\ \texttt{return}\ \mathit{os} \\
&\quad = (\mathtt{lh}_1\ ;\ldots;\ \mathtt{lh}_n)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— } \mathit{mask}_H \text{ idempotent} \\
&\quad = (\mathtt{lh}_1\ ;\ldots;\ \mathtt{lh}_n\ ;\ \mathtt{mask}_H)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— assoc.} \\
&\quad = (\mathtt{lh}_1\ ;\ldots;\ \mathtt{l}_n\ ;\ \mathtt{mask}_H)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— clobber} \\
&\quad = (\mathtt{lh}_1\ ;\ldots;\ \mathtt{mask}_H\ ;\ \mathtt{l}_n)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— atomic nonint.} \\
&\quad = (\mathtt{lh}_1\ ;\ \mathtt{mask}_H\ ;\ \ldots;\ \mathtt{l}_n)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— atomic nonint.} \\
&\quad = (\mathtt{l}_1\ ;\ \mathtt{mask}_H\ ;\ \ldots;\ \mathtt{l}_n)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— clobber} \\
&\quad = (\mathtt{l}_1\ ;\ \ldots;\ \mathtt{l}_n)\ \texttt{>>=}\ \lambda \mathtt{os}.\ \mathtt{mask}_H\ \texttt{>>}\ \texttt{return os} && \text{— “reversing previous steps”} \\
&\quad = \mathit{pull}\ \ \mathit{os}\ \ [i_1,\ldots,i_n]\ \ (\mathit{dualcore}\ \mathit{lo}\ \mathit{nop}) \qquad \texttt{>>=}\ \ \lambda \mathit{os}.\ \ \mathit{mask}_H\ \ \texttt{>>}\ \ \texttt{return}\ \ \mathit{os}
\end{aligned}
$$

Performance

- Comparing the single core PicoBlaze to the dual core:

	Slices	Flip Flops	4-LUTs	F_{max} (MHz)
2-Core	907	258	1735	67.867
1-Core	451	110	866	69.956
Ratio	2.011	2.345	2.003	0.970

Hardware vs. Program Verification

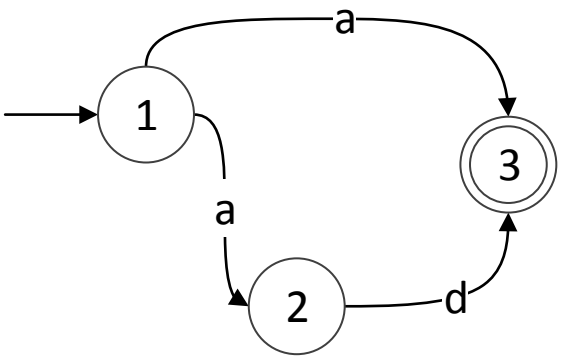
Traditional HW Verification

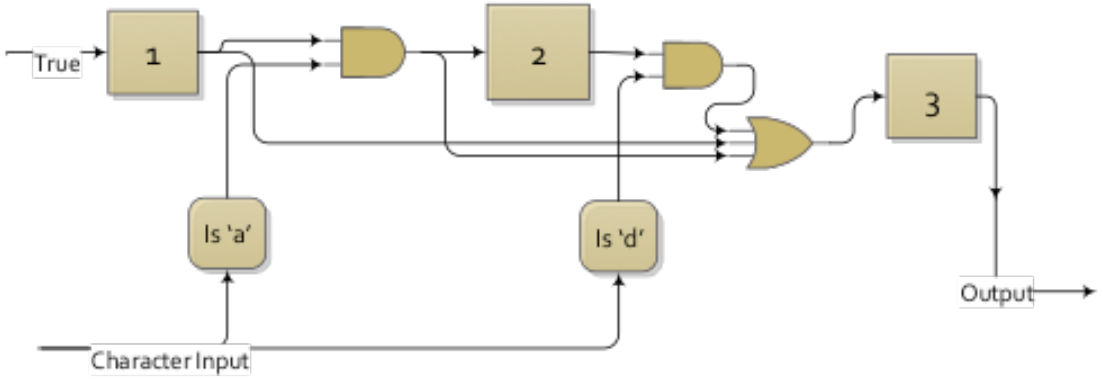
- HW Verification has been around for many, many years…
  - HOL (Cambridge), Boyer-Moore (Texas), Isabelle (Cambridge & Munich), BDD's, etc., etc.
- Basic Recipe
  1. Start with circuit,
  2. Produce formal model capturing its essence,
  3. Encode in theorem prover logic & verify!
- How do you check the faithfulness of Step 2?
  - Does the model capture the artifact?
  - Can you prove that it is faithful?

Program Verification

- Say you have a programming language,
- **IF** you have:
  - a compositional semantics for the language, and
  - a trusted compiler,
- **THEN** you can:
  - verify programs
  - verify compiler's semantic faithfulness, and
  - produce high assurance implementations.
- Canonical example: Hoare semantics for procedural languages.
- This is the approach ReWire takes.

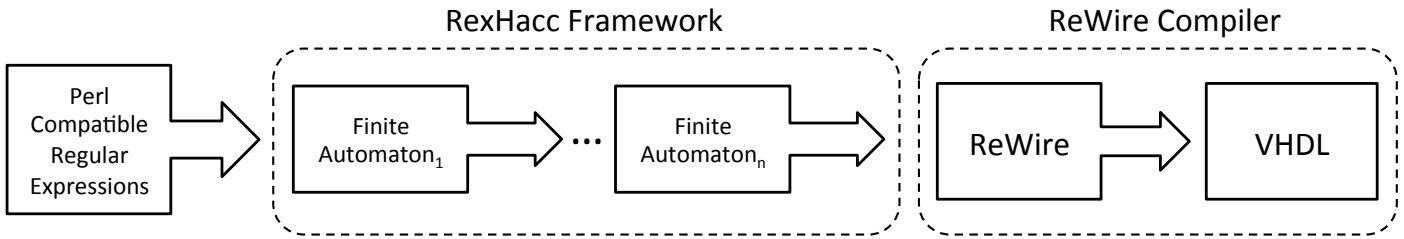
Fast Regular Expression Matching Using FPGAs





- Deep Packet Inspection for detecting malware
- Use HW Parallelism to Represent Non-determinism
- Sidhu & Prasanna 2001
- Becchi & Crowley 20[07|08|09|10]
  - Handwritten regular expression compiler in C
  - State of the art performance

Regular EXpression HArdware Compiler-Compiler



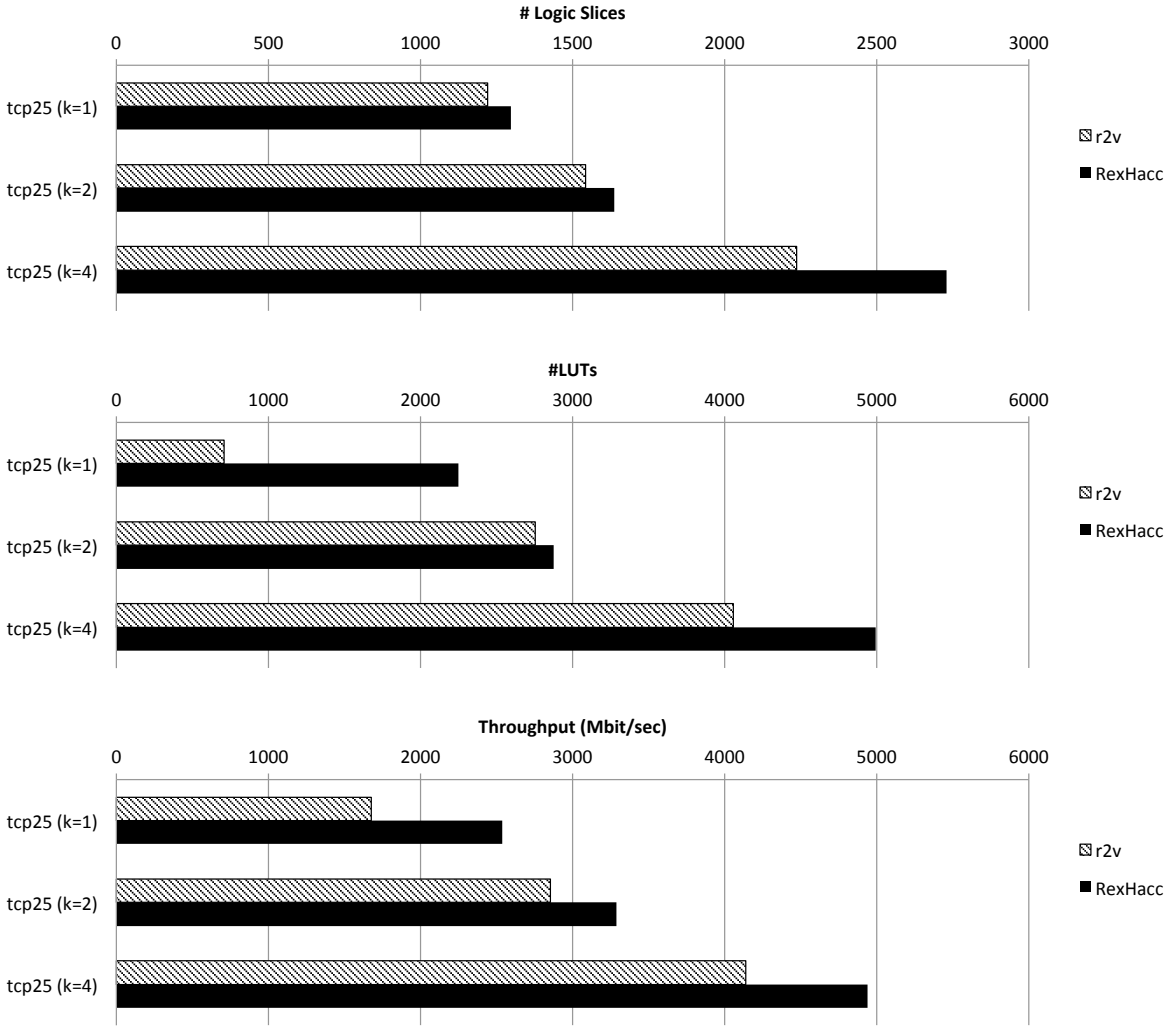

```
rexhacc :: (NFA a -> NFA a) -> RegEx a -> ReWire

compiler :: RegEx a -> ReWire
compiler = rexhacc opt
        where opt = (o1 o … o  on)
```

Details in "*Hardware Synthesis from Functional Embedded Domain-Specific Languages: A Case Study in Regular Expression Compilation*", Graves, et al., Applied Reconfigurable Computing (ARC15).

RexHacc Performance Evaluation

Details in “Hardware Synthesis from Functional Embedded Domain-Specific Languages:

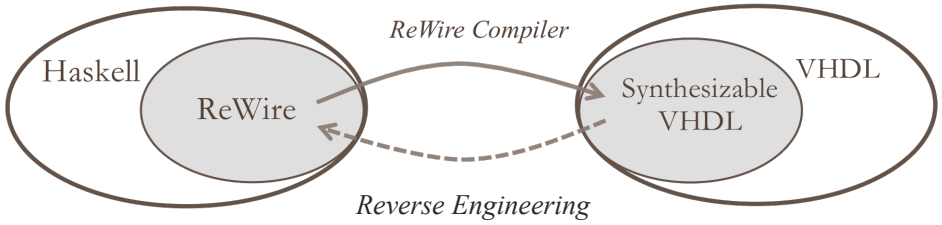


ReWire & Proof Engineering

- Proof Engineerig
  - Rewire both…
    - Computational λ-calculus
    - Expressive Fun. Lang.
  - Unifies specification, design & implementation languages
- ARM spec. [Fox/Myreen10,…]

```
arm_instr :
    iid                          →
    encoding×bool[4]×instr       →
    unit M
```

- Collaboration with Australian DSTO laboratory



THANKS!

Joint work with Dr. Gerry Allwein of US Naval Research Laboratory and Dr. Michela Becchi, Dr. Adam Procter, and Ian Graves of MU

Papers

- **Semantics Driven Hardware Design, Implementation, and Verification in ReWire**, Procter, et al. Languages, Tools and Compilers for Embedded Systems (LCTES) 2015 (to appear).
- **Hardware Synthesis from Functional Embedded Domain-Specific Languages: A Case Study in Regular Expression Compilation.** Graves, et al. Applied Reconfigurable Computing (ARC) 2015.
- **Semantics Directed Machine Architecture in ReWire.** Procter et al. Int. Conf. on Field Programmable Technology (FPT) 2013.
- **The Confinement Problem in the Presence of Faults**. Harrison et al. 14th International Conference on Formal Engineering Methods (ICFEM), 2012
- **Simulation Logic.** Allwein and Harrison. Logic and Logical Philosophy. Volume 23, No. 3, 2014
- **Distributed Logic.** Allwein and Harrison. NRL Memo Report, 2014
- **Modal Distributed Logic.** Allwein and Harrison. Book Chapter: Papers in Honor of J. Michael Dunn, 2015