Hardware/Software Co-verification of Cryptographic Algorithms using Cryptol

Levent Erkök, Magnus Carlsson, Adam Wick

November 18th, 2009
FMCAD’09, Austin TX

The Cryptol team, past and present:
Sally Browning, Magnus Carlsson, Levent Erkök, Sigbjorn Finne, Andy Gill, Fergus Henderson, John Launchbury, Jeff Lewis, Lee Pike, John Matthews, Thomas Nordin, Mark Shields, Joel Stanley, Frank Seaton Taylor, Jim Teisher, Philip Weaver, Adam Wick

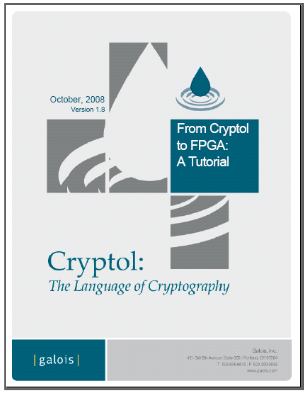
| galois |

Challenge: verifiably correct crypto

- From NIST's 2008 Annual Report (pg 15)
  - 48% of crypto-modules, and 27% of crypto-algorithms had flaws.
  - Without evaluation, about 50-50 chance of buying correct crypto
- Critical for information security
- Major goals:
  - Create verifiably correct crypto
  - Prove existing implementations correct
- Focus on algorithms (not waveform)

Approach:
Specifications and Formal Methods

- Executable specification language
  - Language tailored to the crypto domain
  - Designed with feedback from cryptographers
- Execution and Validation Tools
  - Tool suite for different implementation and verification applications
  - In use by crypto-implementers



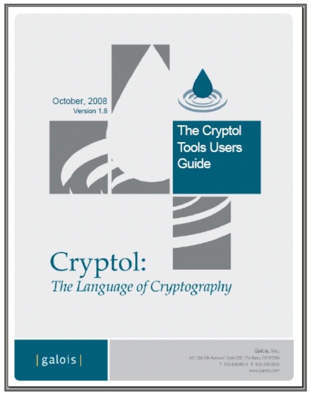
October, 2008
Version 1.8

From Cryptol to FPGA: A Tutorial

Cryptol:
The Language of Cryptography

| galois |

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, OR 97204
T 503.626.6616 | F 503.350.0833
www.galois.com



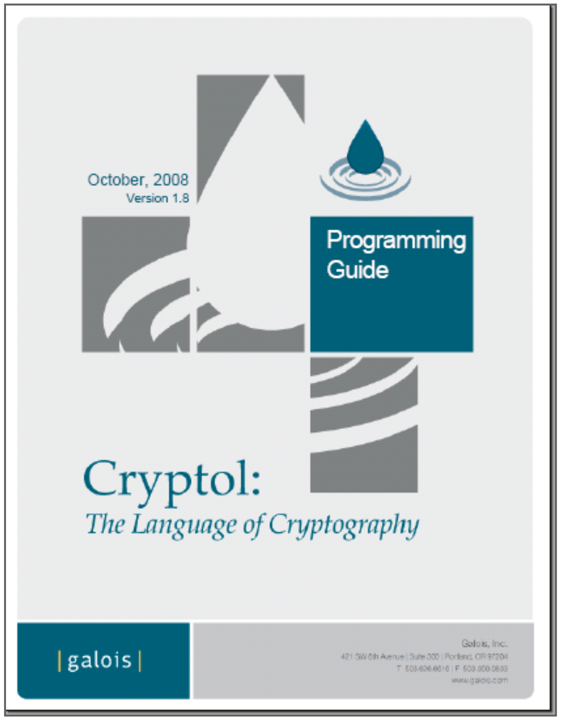
October, 2008
Version 1.8

The Cryptol Tools Users Guide

Cryptol:
The Language of Cryptography

| galois |

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, OR 97204
T 503.626.6616 | F 503.350.0833
www.galois.com



October, 2008
Version 1.8

Programming Guide

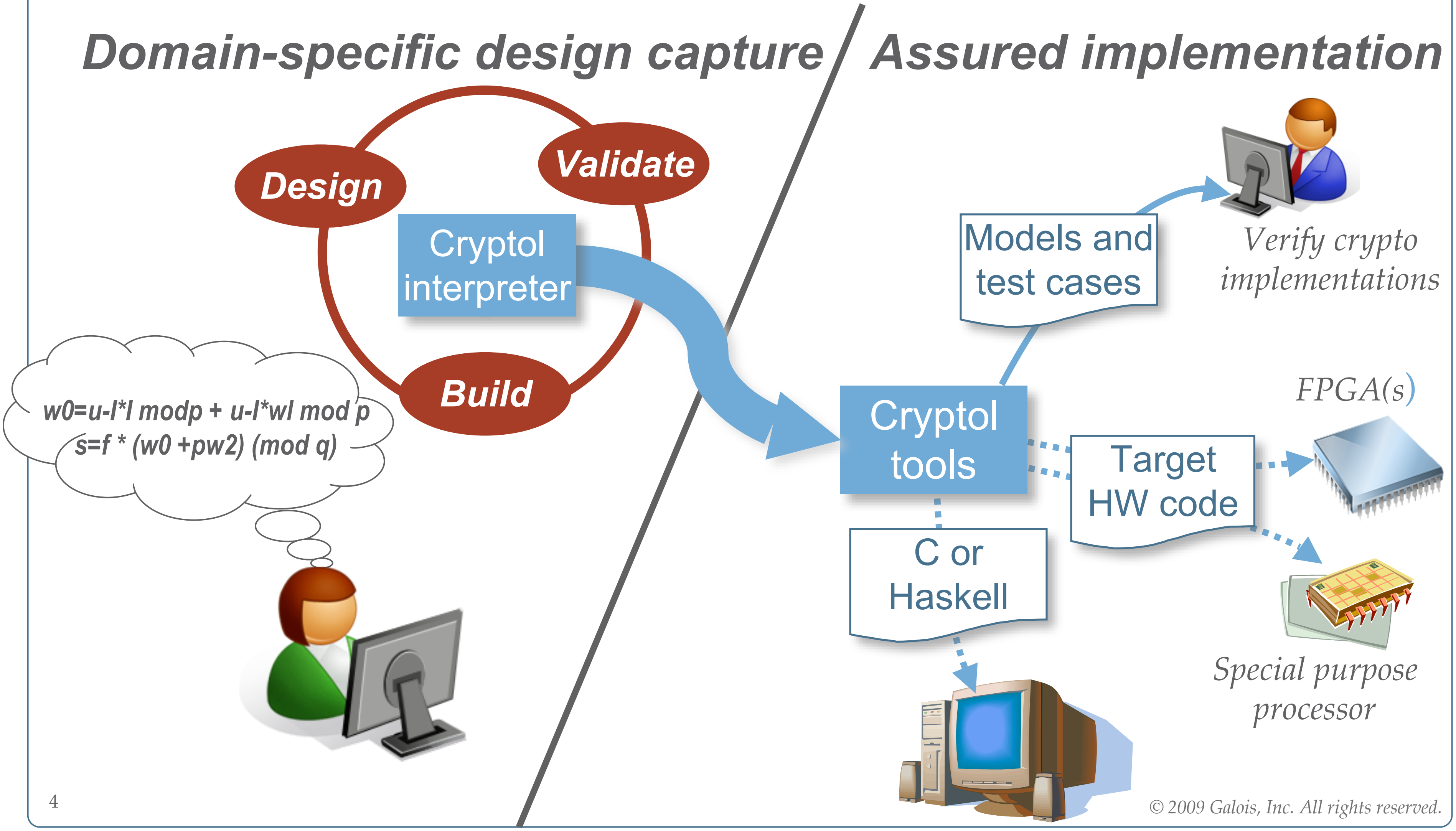
Cryptol:
The Language of Cryptography

| galois |

Galois, Inc.
421 SW 6th Avenue | Suite 300 | Portland, OR 97204
T 503.626.6616 | F 503.350.0833
www.galois.com

One Specification - Many Uses

Domain-specific design capture / **_Assured implementation_**



Design

Validate

Build

Cryptol interpreter

w0=u-l*l modp + u-l*wl mod p
s=f * (w0 +pw2) (mod q)

Cryptol tools

Models and test cases

Verify crypto implementations

Target HW code

FPGA(s)

Special purpose processor

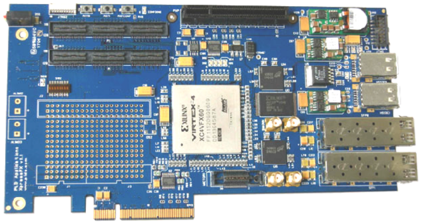
C or Haskell

Cryptol Project Mission: *To reduce the cost (in both time and money) of developing and certifying cryptographic applications*

Cryptol Specification

A Domain Specific Language

- Precise, Declarative Semantics
- High level design exploration
- Executable



Automated Synthesis down to FPGA and VHDL verification

- Evidence producing translation technique
- Verifying both generated and 3rd party VHDL
- SAT based equivalence checking

Certificate of Trust

This is to certify that

Cryptol Specification

can be trusted

Certificate Authority

Property specification and verification

- SAT/SMT based property verification
- Push button assurance
- Semi-automatic theorem proving (via Isabelle/HOL)

Cryptol Programs

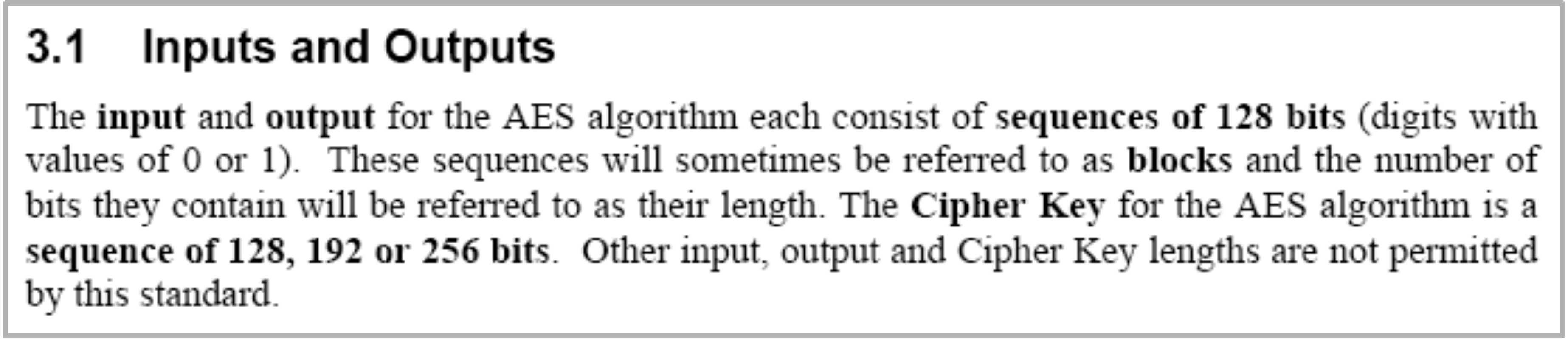
- File of mathematical definitions
  - Strong static typing, with type inference
  - Heavily influenced by functional languages (Haskell in particular)
- Definitions are computationally neutral
  - Think equations, not programming.
  - No assignments, no side effects.

```
x : [4][32];
x = [23 13 1 0];

F : ([16],[16]) -> [16];
F (x,x’) = 2 * x + x’;
```

Expressive Type system

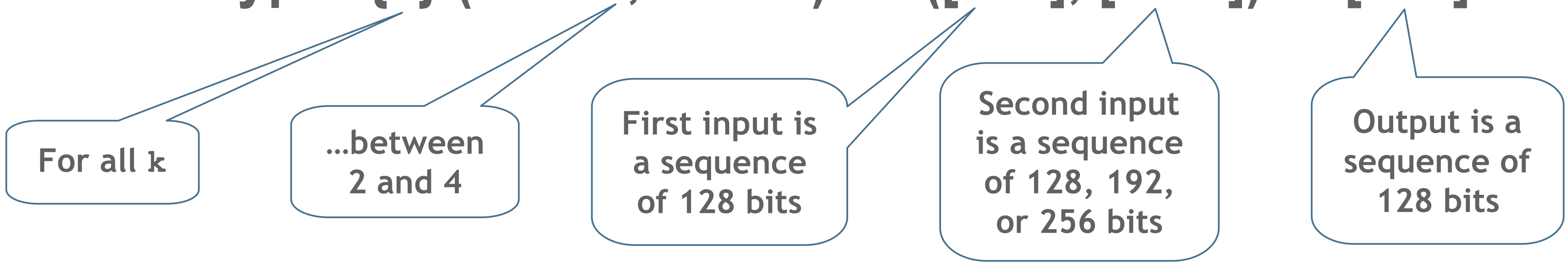
From the Advanced Encryption Standard definition[†]



3.1 Inputs and Outputs

The **input** and **output** for the AES algorithm each consist of **sequences of 128 bits** (digits with values of 0 or 1). These sequences will sometimes be referred to as **blocks** and the number of bits they contain will be referred to as their length. The **Cipher Key** for the AES algorithm is a **sequence of 128, 192 or 256 bits**. Other input, output and Cipher Key lengths are not permitted by this standard.

```
blockEncrypt : {k} (k >= 2, 4 >= k) => ([128], [64*k]) -> [128]
```



For all k

...between 2 and 4

First input is a sequence of 128 bits

Second input is a sequence of 128, 192, or 256 bits

Output is a sequence of 128 bits

[†] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Language overview

- Bit-vectors as main data type
- Strong type system, with size and parametric polymorphism
  - Based on Hindley-Milner type inference
  - Extended with arithmetic predicates
- Pure
  - No side effects, no I/O
  - Waveform code is ***not*** our target
- Associated verification system
  - Functional correctness properties directly specifiable in code
  - Symbolic simulation + equivalence checking
- Cryptol is a happy marriage of research in:
  - Functional programming
  - (Automated) formal methods

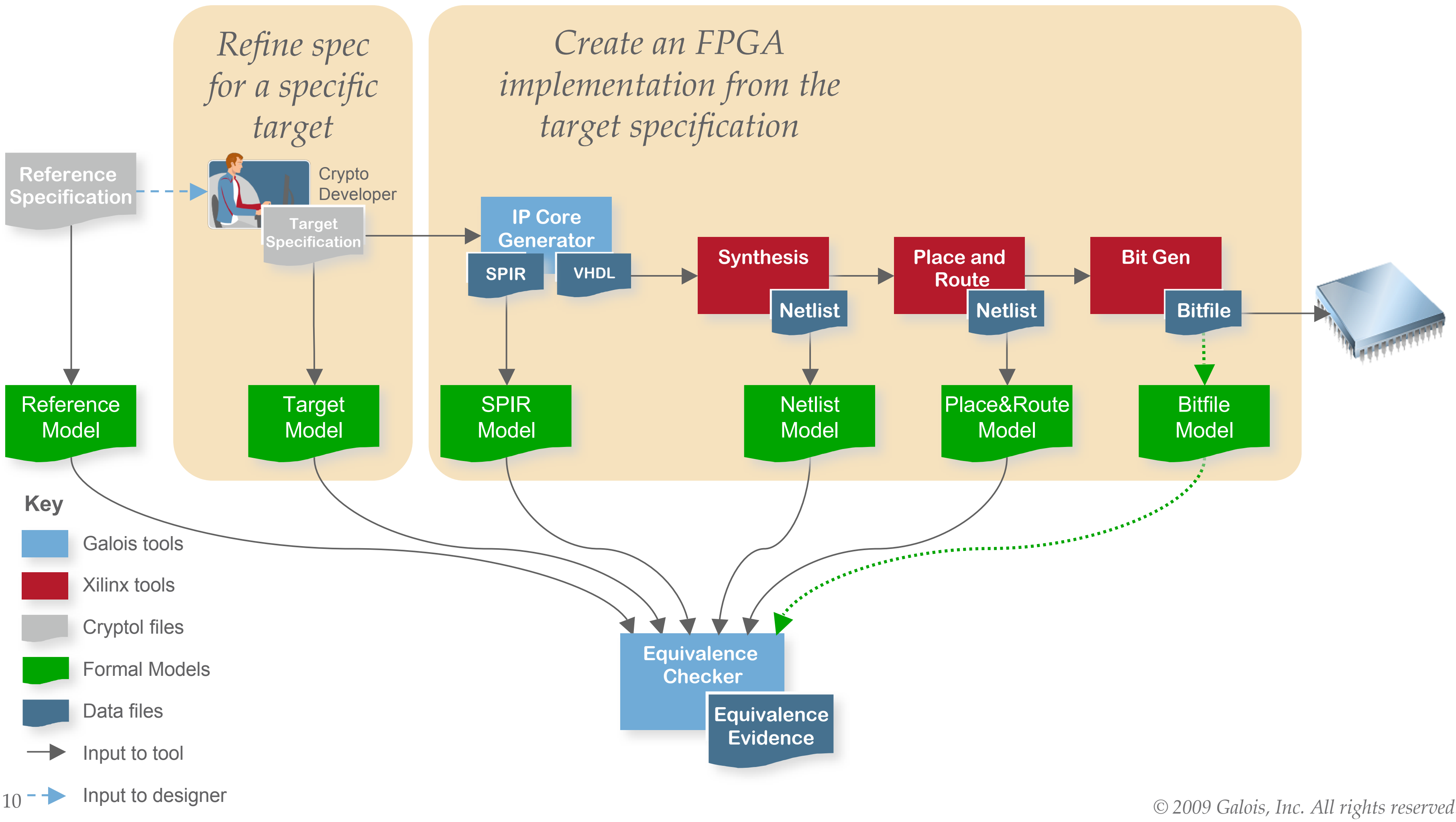
Generating and verifying FPGAs

- Cheaper to deploy in quantity
- Lack of trust in commodity hardware
  - Evaluators can see as much of the solution as possible
  - Do not have to ship designs off-shore
- Natural match between Cryptography and FPGAs
  - Highly-parallel stream processing
- And FPGAs are *fast…*

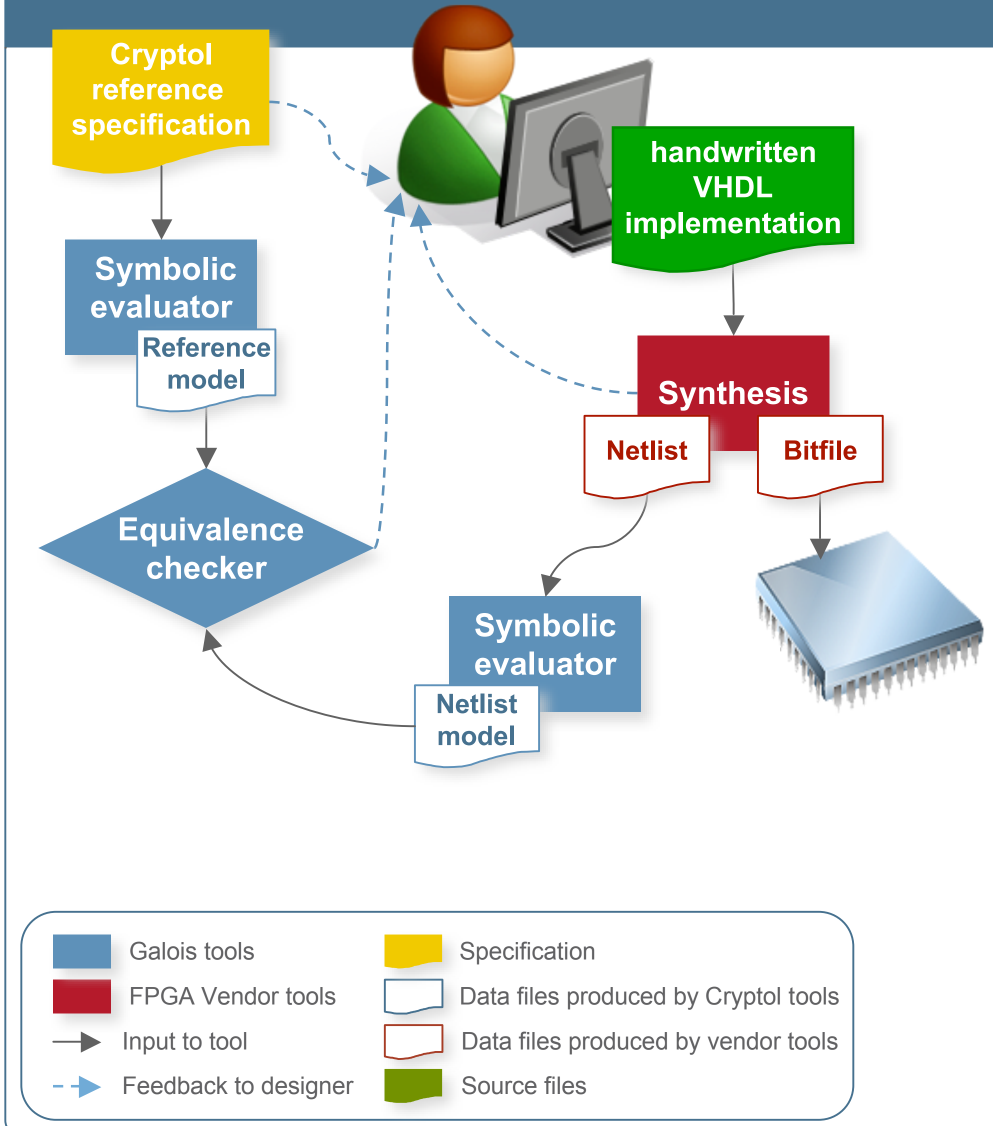
galois

Strategy

Maintain functional equivalence with the reference specification throughout the tool chain



Cryptol in the Development Process: Hand-written VHDL



A VHDL-FPGA engineer:

- Studies the reference specification to gain understanding
- Crafts a VHDL implementation by hand
- Uses the equivalence checker to certify
- Counter-examples are priceless!

Some Verification Results

- NIST Hash Competition (Skein)
  - Men Long (Intel)
  - Stefan Tillich (TU Graz)
- NIST AES Competition
  - Reference C
  - Optimized C

NIST Hash Competition

- “NIST has opened a public competition to develop a new cryptographic hash algorithm, which converts a variable length message into a short “message digest” that can be used for digital signatures, message authentication and other applications.”
- 51 submissions
- Galois has verified VHDL implementations of some
  - Against Cryptol “golden” specs
- We’ll look at Skein verification in detail:

http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

Verification Process

1. Develop a specification
2. Understand the implementation
3. Match the type signature of the implementation and specification
4. Use Cryptol to generate AIGs for both the implementation and specification
5. Call the equivalence checker

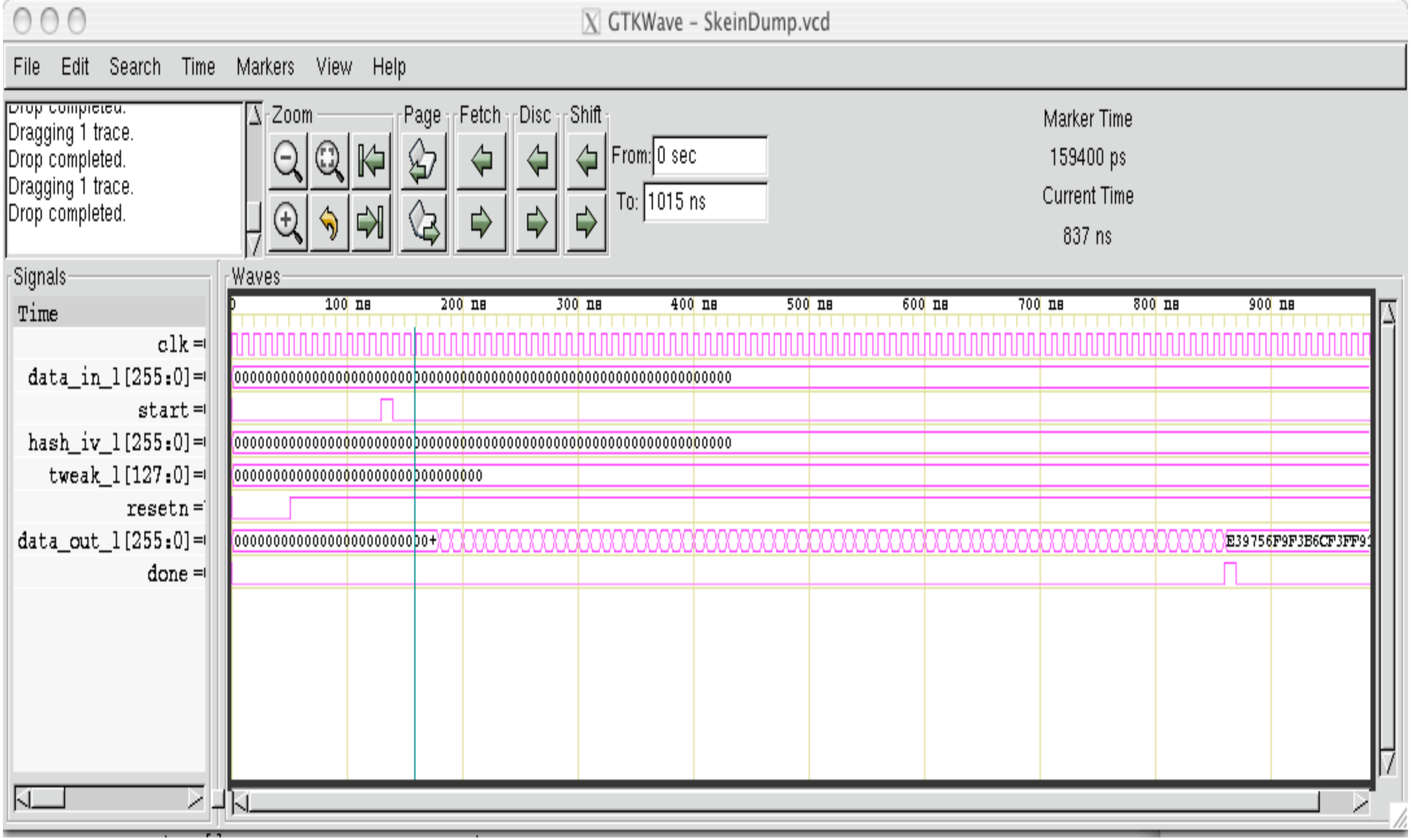
Develop a Specification

```
encrypt256 : [256] -> [256];
encrypt256 (key_tweak_pt) = vn + kn
  where {
   // Threefish-256 has 72 rounds:
   nr  = 72;
   nw  = 4;
   …
   key_words : [4][64];
   key_words = split(join key);
   tw_words : [2][64];
   tw_words = split(join tweak);
   pt_words : [nw][64];
   pt_words = split(join pt);
  };
```

http://www.galois.com/blog/2009/01/23/a-cryptol-implementation-of-skein/

galois

Understanding the Implementation



```
Drop completed.
Dragging 1 trace.
Drop completed.
Dragging 1 trace.
Drop completed.
```

From: 0 sec

To: 1015 ns

Marker Time

159400 ps

Current Time

837 ns

```
Time
clk =
data_in_l[255:0] =  0000000000000000000000000000000000000000000000000000000000000000
start =
hash_iv_l[255:0] =  0000000000000000000000000000000000000000000000000000000000000000
tweak_l[127:0] =    00000000000000000000000000000000
resetn =
data_out_l[255:0] = 0000000000000000000000000+  E39756F9F3B6CF3FF91
done =
```

Import the VHDL to Cryptol

```
extern vhdl("datatype.vhd”, "skein_mixcolumn.vhd", "skein_round.vhd”,
            "skein_shiftrow.vhd‟, "skein_add_round_key.vhd”, "skein.vhd‟,
             skein, clock=clk, reset=resetn, invertreset)
extern_menLong : [inf](start:Bit, data_in_L:[256], hash_iv_L:[256],
                       tweak_L:[128]) ->
                 [inf](done:Bit, data_out_L:[256]);
```

- Very closely matches the VHDL implementation
- Imports are done as stream processors
- Details such as start/reset signals are still present
- The imported function becomes a first-class citizen!

Match the Type Signatures

```
menLongVHDL : [256] -> [256];
menLongVHDL inp = res
  where { wait     = (False, inp, zero, zero);
          start    = (True, inp, zero, zero);
          rest     = [wait] # rest;
          (_, res) = extern_menLong([wait start] # rest) @ 74;
        };
```

- Put a “functional” view over the imported VHDL
- Signal and timing details are resolved
- Matches the signature of the reference spec
- Compare with:

```
extern_menLong : [inf](start:Bit, data_in_L:[256], hash_iv_L:[256],
                       tweak_L:[128]) ->
                 [inf](done:Bit, data_out_L:[256]);
```

Use the equivalence checker

- Generate AIGs from:
  - Reference Cryptol implementation
  - Imported VHDL implementation
- Symbolic simulation based technique
  - Both for Cryptol and Netlist descriptions
- Call an external SAT solver
  - Potential models become “bugs” found
- Cryptol mediates the interaction
  - No specific knowledge of external tools needed

Men Long Equivalence Check

- VHDL Implementation of the Skein UBI Block
- Skein UBI Block AIG Sizes
  - Cryptol Reference, 118156 nodes
  - Men Long, 653963 nodes
- Found one ambiguity issue
- Used ABC (UC Berkeley) Equivalence Checker
- In ~1 hr VHDL code proved equivalent to spec
- Quite good for 256 bits of input
  - 2^{256} is a big number!

Stefan Tillich Equivalence Check

- Full Skein VHDL Implementation
- Skein AIG Sizes (256 bits input/output)
  - Cryptol Reference, 301342 nodes
  - Stefan Tillich, 900496 nodes
- Used ABC (UC Berkeley) Equivalence Checker
- Time: ~17.5h
- VHDL code is equivalent to Cryptol spec.

http://www.iaik.tugraz.at/content/research/

Synthesis from Cryptol

- High-level design exploration helps
  - Much easier to code in Cryptol than in VHDL
- Experiment: Synthesized core Threefish rounds
- Speed comparison:

Men Long	Tillich	Cryptol
0.409 Gb/s	1.75 Gb/s	1 Gb/s

- Further optimizations certainly possible

Why does this work?

- SAT based equivalence checking is an ideal fit for crypto
- Properties of symmetric-key encryption algorithms:
    - Very regular structure; no fancy operations
        - AES can be implemented just using array-lookup and XOR
    - No data-dependent control flow, to avoid timing attacks
        - Means no if-then-else splits!
    - All loops have upper bounds known statically
- SAT-sweeping very effective
    - Simulation based node-equivalence guesses are likely to be very accurate
- Bottom line: symmetric-key crypto is mostly bit-shuffling, and SAT is good at that
- NB. Doesn’t apply to ECC (Elliptic-Curve Cryptography)

Beyond equivalence checking

- Equivalence checking shows functional equivalence
  - The input/output behaviors are “precisely the same”
  - Or, they both have the exact same bugs..
- Property verification goes further
  - Allows “correctness” properties to be specified and proved automatically
- Classic crypto example:
  - For all values of key and plain-text, encryption followed by the decryption using the same key returns the plain-text
  - In Cryptol:

    ```
    theorem encDec: {key, pt}. dec (key, enc(key, pt)) == pt;
    ```
- Cryptol theorems are first class citizens of the language
  - Not just documentation!

Other Cryptol Assurance tools

- “Quickcheck” property-based testing
  - User gives a property, Cryptol automatically tests it on random inputs.
- Safety checking
  - Automatically checks that a Cryptol function will never raise an exception
  - Some possible exceptions: Divide-by-zero, Out-of-bounds array access, assertion failures
- SMT based property verification
  - Allows direct use of arrays
  - Uninterpreted functions (QF_AUFBV)
- Semi-automatic theorem proving
  - Translator from Cryptol to Isabelle theorem prover
  - User can specify arbitrary Cryptol properties, but proof may need human guidance
- Last two are essential for non-symmetric key systems
  - ECC (esp. large word multiplication) is a soft-spot for SAT

Questions?

www.cryptol.net

Contact: cryptol@galois.com