# Cryptol on FPGAs

John Launchbury
Galois Connections Inc
john@galois.com

Satnam Singh
Xilinx Inc
satnam@xilinx.com

GALOISCONNECTIONS

## Plan

- Cryptol domain-specific language
- The FPGA opportunity
- Lava and Jbits
- Review

GALOISCONNECTIONS

# Problem: Crypto-algorithm Verification and Validation


- **Managing assurance in face of exploding need and complexity**
  - Variety of target architectures
  - Legacy algorithms
  - Variety of requirements
  - Validation is laborious
  - Requires skills in both math and programming
- **Not just the NSA**
  - "25% of algorithms submitted for FIPS validation had security flaws"

Annabelle Lee, Director NIST CMVP
March 26, 2002



# C as a specification language

✔ Executable and less ambiguous than pseudo code

✘ But *not* formal and unambiguous
  - E.g. meaning of shifts in C is left up to the machine

✘ Forced to conform to machine model (word size, sequential, etc)

✘ Two words: buffer overflow!

✘ Lots of administrative overhead
  - Memory management
  - Working around word size limitations
  - …

# Overhead in C

```
static void Rc6ComputeKeySchedule(BYTE* key, int keyLengthInBytes,
                                  unsigned long* S)
{
  unsigned long L[(255+4-1)/4];
  const int t = 2+2*ROUNDS+2;
  int count;
  int u;
  int i,j;
  unsigned long A,B;
  int startOfExtraDword, numberOfExtraBytes;
  int c = (keyLengthInBytes+4-1)/4;
  if (c == 0)
    c = 1;
  for (count = 0; count < c; count++)
    L[count] = 0;
  for (count = 0; count < keyLengthInBytes/4; count++)
    L[count] = LoadDword(key+count*4);
  startOfExtraDword = keyLengthInBytes & 0xfffffffc;
  numberOfExtraBytes = keyLengthInBytes & 0x3;
  if (numberOfExtraBytes > 0) {
    L[c-1] = (unsigned long) key[startOfExtraDword];
    if (numberOfExtraBytes > 1) {
      L[c-1] |= (((unsigned long) key[startOfExtraDword+1]) << 8);
      if (numberOfExtraBytes > 2)
        L[c-1] |= (((unsigned long) key[startOfExtraDword+2]) << 16);
    }
  }
  S[0] = P32;
  for (count = 1; count < t; count++)
    S[count] = S[count-1]+Q32;
  u = 3*((c>t) ? c : t);
  i = j = 0;
  A = B = 0;
  for (count = 1; count <= u; count++) {
    unsigned long sum;
    sum = A+B;
    S[i] += sum;
    A = S[i] = ROTL(S[i], 3);
    sum = A+B;
    L[j] += sum;
    B = L[j] = ROTL(L[j], sum);
    i = (i+1) % t;
    j = (j+1) % c;
  }
}
```

This part corresponds to the Cryptol code on the right

Reference RC6 key schedule in C

RC6 key schedule in Cryptol

```
rc6ks key = split (rs >>> (v - 3 * nk))
  where {
    c = max (1, (width key + 3) / (w / 8));
    v = 3 * max (c, nk);
    initS =  [pw (pw+qw) ..] @@ [0 .. (nk-1)];
    padKey : [4*c][8];
    padKey = key # zero;
    initL : [c][w];
    initL = split (join padKey);

    ss = [| (s+a+b) <<< 3     || s <- initS # ss
                              || a <- [0] # ss
                              || b <- [0] # ls |];
    ls = [| (l+a+b) <<< (a+b) || l <- initL # ls
                              || a <- ss
                              || b <- [0] # ls |];
    rs = ss @@ [(v-nk) .. (v-1)];
  };
```



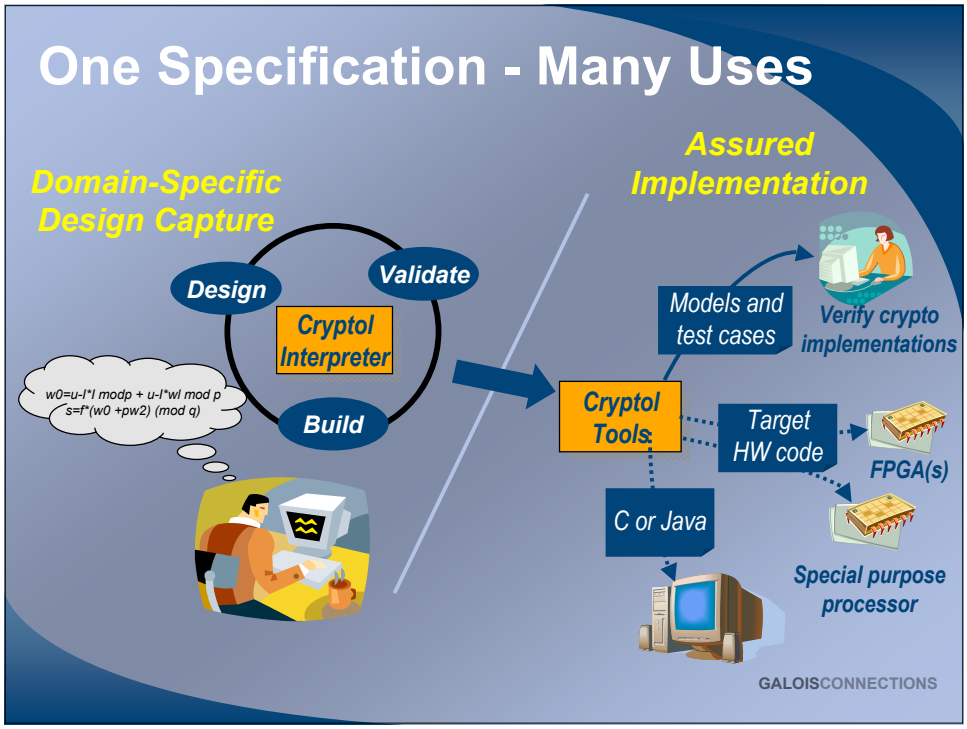
# Alternative: Specification Language and Formal Tools

- Domain-specific language for crypto specification
  - Model crypto algorithm
  - Natural – easy to express
  - Declarative
  - Clear and unambiguous

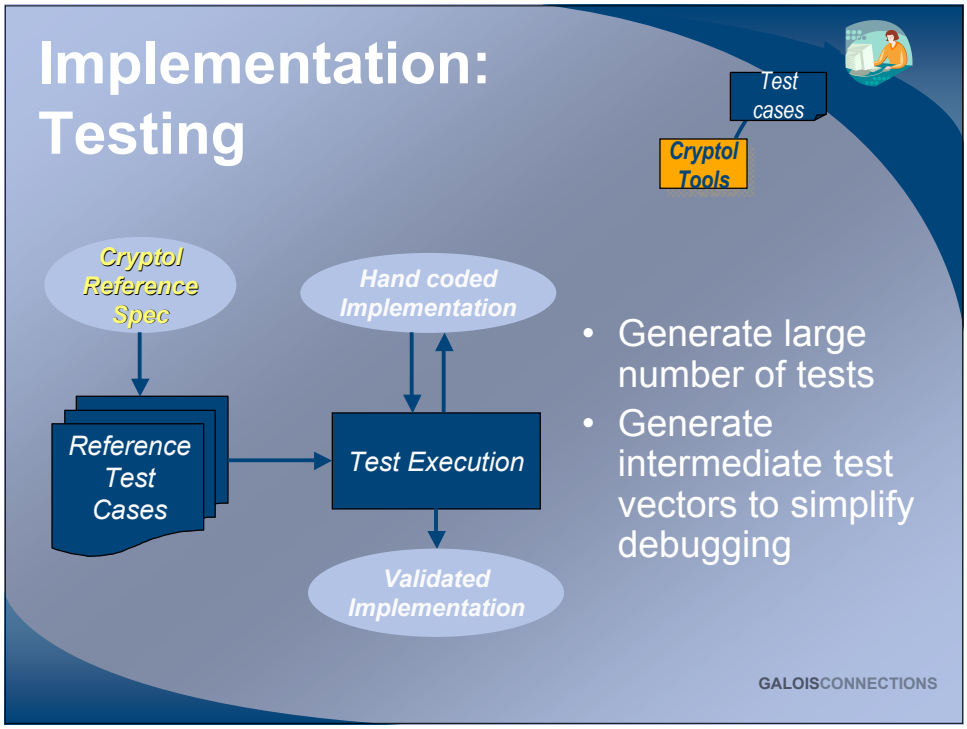
**Cryptol**
*The Language of Cryptography*

- Developed by Galois
  - Designed with feedback from NSA cryptographers
  - In use at NSA, Certicom and General Dynamics
  - Continued funding from NSA

# One Specification - Many Uses

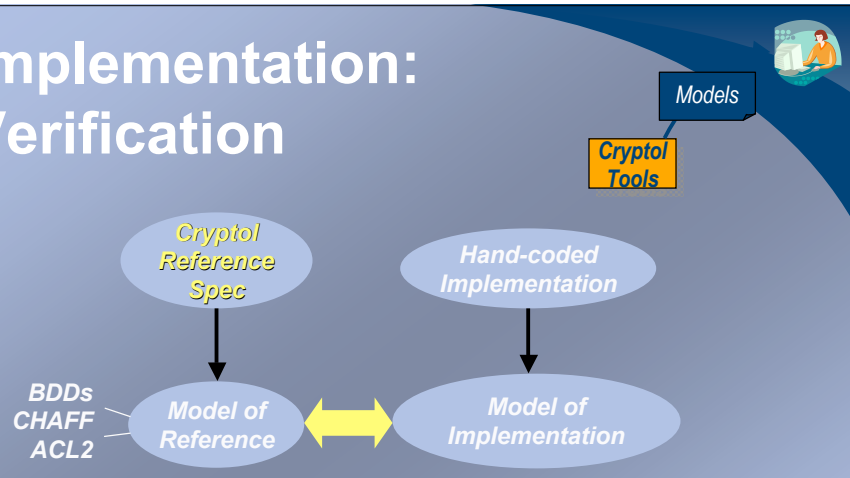



# Implementation: Testing




- Generate large number of tests
- Generate intermediate test vectors to simplify debugging

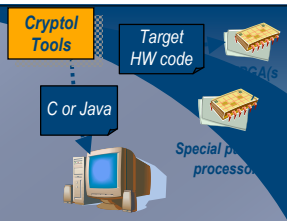
# Implementation: Verification



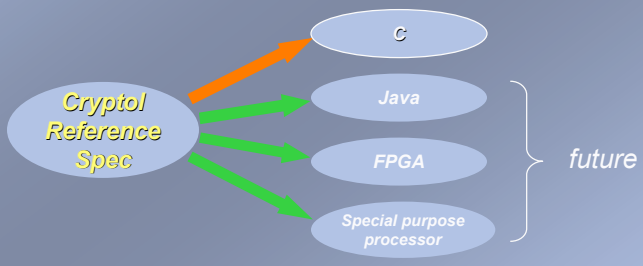

- Model checking now in development
- Will enable formal verification between Reference and Implementation
- Much higher assurance of correctness

GALOISCONNECTIONS

# Implementation: Code Generation




*A single correct, executable Cryptol specification can be deployed to a variety of target platforms…*




- **One specification to 'get right'**
- **Many targets for use**

GALOISCONNECTIONS

# Crypto-algorithm domain

- ***Crypto Domain Concepts***
  - Mathematical equational definitions
  - Data: bits, bit-vectors (words), vector structures, etc.
  - Flexible views of data, high-level operations
  - Coinductive streams
- Cryptol
  - Language of Sequences
    - Bit sequences, "words"
    - Word sequences, "blocks"
    - Sequences of sequences, "matrices"
  - *Finite* and *infinite* sequences
  - Rich set of operators

*Cryptol states what the sequence **is**. Computationally, sequences may exist in **time**, or in **space***

GALOISCONNECTIONS

# Sequence Operators

- Taking sequences apart
  - Into *n* segments, or
  - With *n* in each segment
- Joining sequences together
- Dropping a sequence prefix
- Transpose a sequence of sequences
- Reversing a sequence
- Sequence lookup

```
[1 2] # [4 6 3]
      = [1 2 4 6 3]
```

GALOISCONNECTIONS

# Multiple views

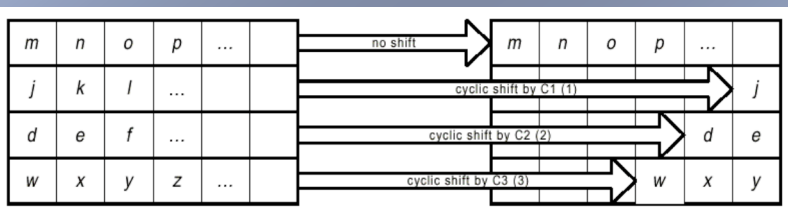



Figure 3: ShiftRow operates on the rows of the State.



# Multiple views

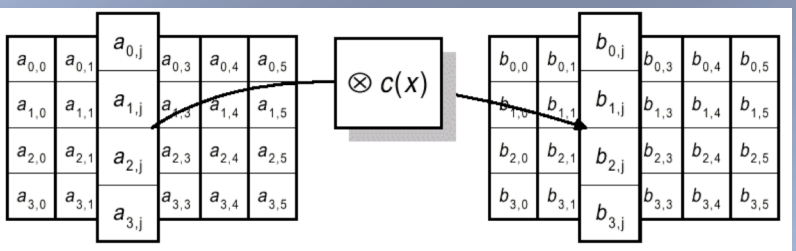



Figure 4: MixColumn operates on the columns of the State.

## Sequence Comprehensions

- Borrowed the comprehension notion from set theory

  **{ 2x+3 | x □ {1, 2, 3, 4} } = {5, 7, 9, 11}**

  Adapted to sequences

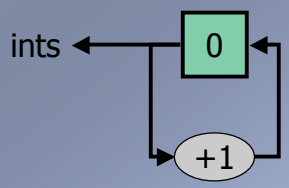
```
[| 2*x + 3 ||  x <- [1 2 3 4] |]
   =  [5 7 9 11]
```



## Recurrence

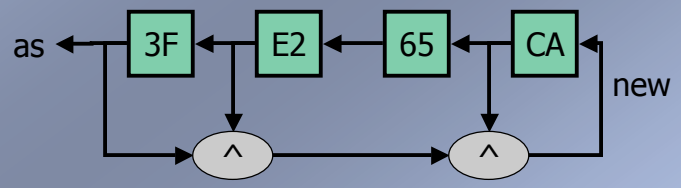
- Textual description of shift circuits
  - Follow mathematics: use stream-*equations*
    - Stream-definitions can be *recursive*

```
ints = [0] # [| y+1 || y <- ints |];
```



# More Complex Stream Equations

```
as  = [0x3F 0xE2 0x65 0xCA] # new;
new = [| a ^ b ^ c || a <- as
                   || b <- drop(1,as)
                   || c <- drop(3,as)|];
```





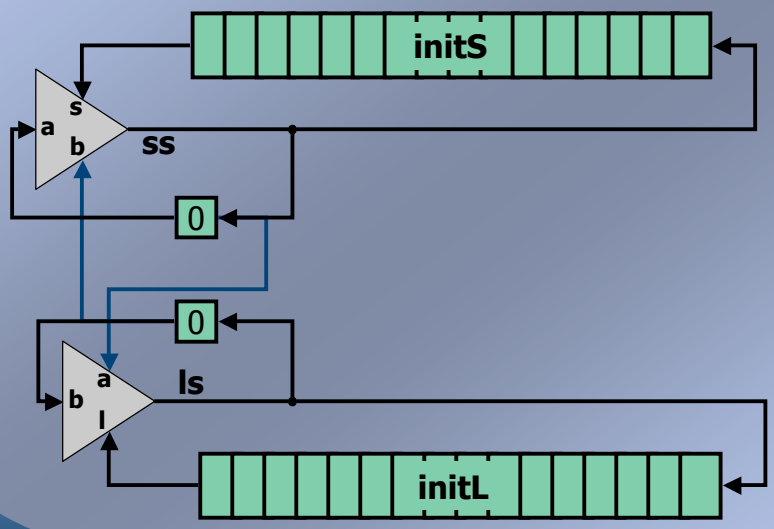

# RC6 Key Expansion

- Original specification is written in terms of arrays and updates
  - Key expansion code appears entirely symmetrical
  - Cryptol exposes non-symmetry

```
ss = [| (s+a+b) <<< 3 || s <- initS # ss
                      || a <- [0] # ss
                      || b <- [0] # ls |];

ls = [| (l+a+b)<<<(a+b) || l <- initL # ls
                        || a <- ss
                        || b <- [0] # ls |];
```

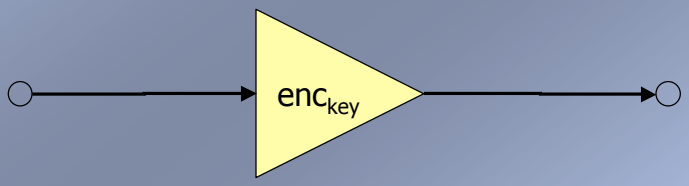
## "Circuit" Diagram




GALOISCONNECTIONS

## Modes: Electronic code book

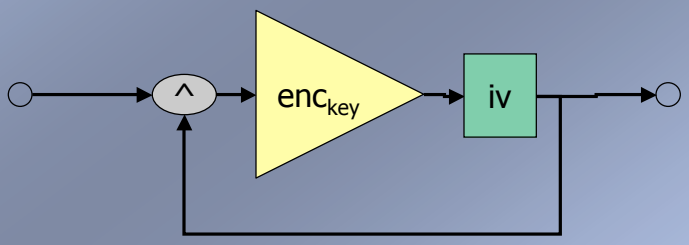
```
ct = [| encrypt (x, key) || x <- pt |]
```




GALOISCONNECTIONS

# Cipher Block Chaining

```
ct = [iv]
     # [| encrypt (x^y, key) || x <- pt
                             || y <- ct |]
```






# Key Observation

- Sequences are descriptions only
- Implementation of sequences can be:
  - Laid out in time
    - Loops and/or state machines
  - Laid out in space
    - Parallel and/or pipeline
  - Or a mixture of both
    - The mathematical specification is the same

***Sequentialization in Cryptol comes only from data-dependency — just like hardware***

## Cryptol Types

- Types express size and shape of data

```
[[0x1FE 0x11] [0x132 0x183]
 [0x1B4 0x5C] [0x26 0x7A]]   has type   [4][2][9]Bit
```

- Strong typing
  - The types provide mathematical guarantees on interfaces
- Type inference
  - Use type declarations for active documentation
  - All other types computed

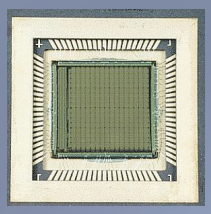


## Current Cryptol System

- Programming environment
  - Interpreter-based
  - Tracing support
  - Test-vector generation
- Compilation
  - Via Haskell with interface to C
  - Partial Compilation directly to C

# FPGAs: The Opportunity

- Configurable hardware
  - Very fast
  - Hugely parallel resource
  - Grows with Moore's Law



- Ubiquitous FPGAs in the future?
  - Large FPGA farms connected to network servers
  - General FPGA resource-board attached to computation engines



# The Crypto-domain

- FPGAs offer tremendous potential in cryptography
  - Highly parallel encryption/decryption
  - Highly parallel crypto-analysis
    - Key search

  . . .

- Natural match between Crypto and FPGAs
  - Manipulation of bit sequences
  - Parallelism

# The Problem

***FPGAs are difficult to use:***

- FPGA tools are designed for “hardware-engineering” not “software-engineering”
- Traditional software languages have inappropriate sequentialization built-in

*Mismatch in both directions*

GALOISCONNECTIONS

# Key Idea

*Computationally neutral*

*Natural for crypto-mathematicians*

*Take the declarative Crypto-specification language Cryptol, and compile right down to FPGAs*

*Complete control of compilation*

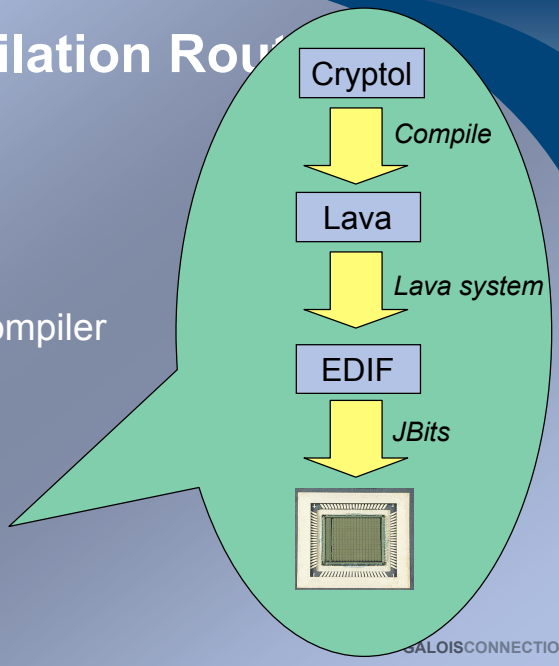
GALOISCONNECTIONS

## FPGA Compilation Rou

Prime choice.

Adapt current Cyptol-to-C compiler to produce

1. Lava, and
2. JBits




GALOISCONNECTIONS

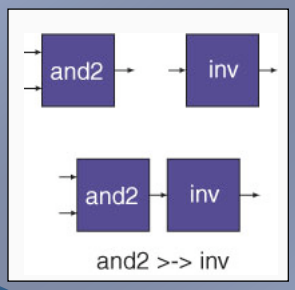
## Naturally Matched Technologies

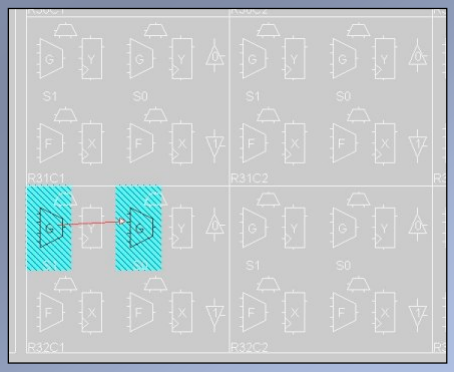
- Cryptol
  - Language designed for crypto-mathematicians
  - Generate finite-state machine descriptions
  - Formal semantics
- Lava
  - Language designed for 2D FPGA specification
  - High-level and declarative
  - Powerful placement computation and control
  - Formal semantics
  - Generates fully mapped EDIF
- Jbits
  - Java API designed for packing and routing FPGAs
  - Produce FPGA configuration bit-stream
  - Lightweight system

GALOISCONNECTIONS

## Lava specs based on *cells*

- Family of operators for connecting cells
- Cells have relative locations

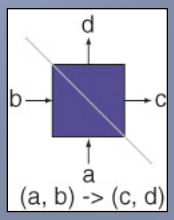


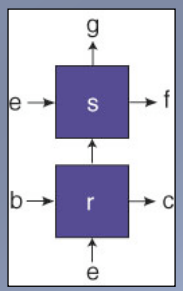



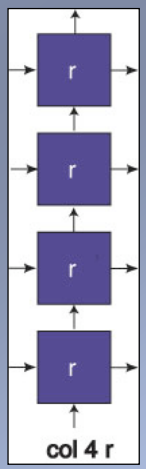
GALOISCONNECTIONS

## Lava 2D Cells

- 2D relative positions
- Cells have types
- Powerful data abstraction

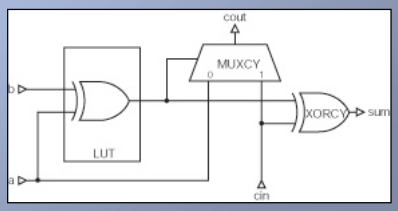


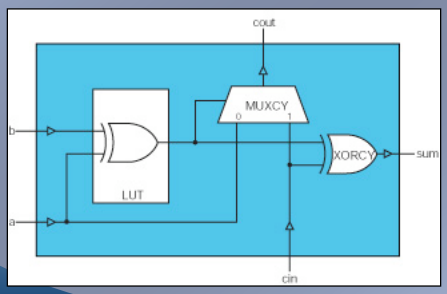



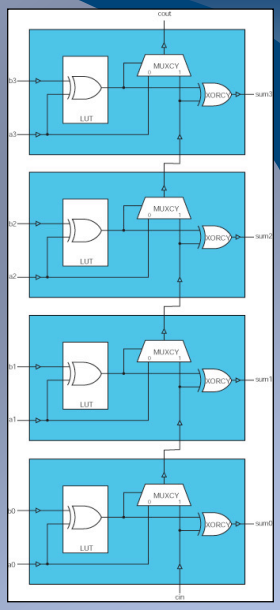




GALOISCONNECTIONS

# Apply to larger pieces

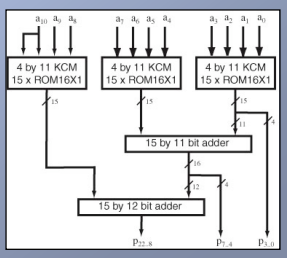


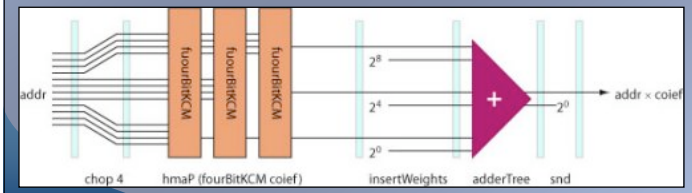



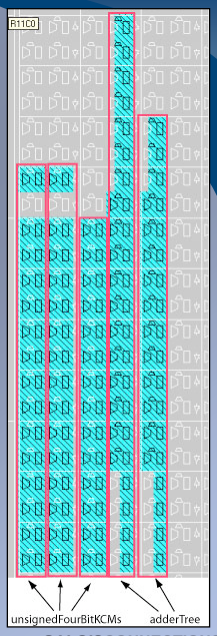




GALOISCONNECTIONS

# Memory-Sum Circuit



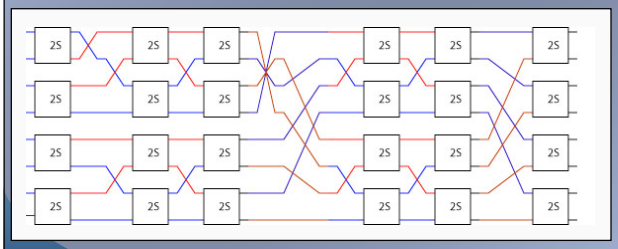



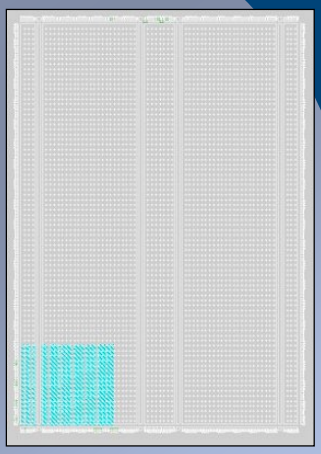




GALOISCONNECTIONS

## Butterfly

- Complex recursively-defined networks of cells
- Provably correct



GALOISCONNECTIONS

## JBits Performance

- Lava to Bitstream via JBits
  - 300,000 logic gates in less than 10 seconds for XCV300
- Modifying single gate at run-time
  - 172 ms

GALOISCONNECTIONS

## The missing piece

- Cryptol to Lava compiler
  - Decisions on parallelism of sequences
    - Space/time tradeoffs
  - Constant folding and partial evaluation
  - Pipeline-introduction transformations
  - Gate-level realization of primitive operations
- Algebraic techniques applicable throughout
  - Provides formal justification for compilation



## Why it can be built…

- **Constrained domain of source language**
  - Finite designs
- **No artificial sequentiality to be removed from the source language**
  - FPGAs provide a way to realize Cryptol's parallelization potential
  - Cryptol provides a way to realize FPGA's parallelization potential
- **Algebraic transformation methods are applicable throughout**
  - Formal (semantics-based) technique for reshaping programs and circuits

## Benefits

- **FPGA resources become available to crypto-mathematicians**
  - Not just to hardware engineers
- **Low barrier-to-entry for FPGA use**
  - Cryptol spec may have been developed for other purposes
  - Standard libraries of Cryptol specifications
  - FPGA implementation is a small delta for the user
- **Cross-compilation development scenario**
  - Develop specs on conventional hardware
  - Execute on FPGA



## Domain Specific Language Approach to V & V

- **Domain Specific**
  - Naturally understandable to developers
  - Simplifies expression, inspection, reuse
- **Executable**
  - Run tests and debug for correctness
  - Generate test cases
- **Declarative**
  - Not implementation-specific, concise
  - Useful for multiple purposes – test, generation, model building, etc.
  - Highly retarget-able to any architecture
- **Unambiguous**
  - Formal basis
  - Precise syntax and semantics
  - Independent of underlying machine models