Eldee Stephens – RAIM/ECC Hardware Designer

23 May 2013

IBM System zEnterprise Design: Better Memory RAS through Novel Verification

A special thanks to Pat Meaney and Alexey Lvov for their provided graphics, and to Alexey who conceptualized and developed Blueveri

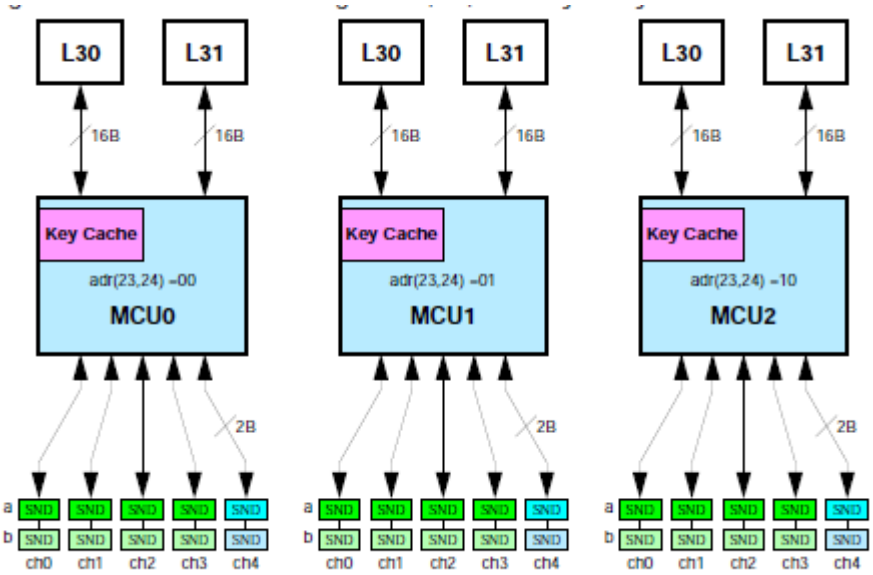
A quick introduction.....

- Joined IBM in 2002 after graduating from Duke University
- Started as member of POWER6 concept team, co-developed nest M2 model; various timing and logic roles in FXU and L3C; also worked on chip timing
- Became pL3 unit lead in 2005
- Punted on Komal, and joined p6 nest bringup team
- Moved to POK in 2007, joined z10 bringup team, focusing on IO and packaging issues
- Joined zG MCU team in 2008, responsible for RAIM, among other things; continued to work in bringup on zG, focusing on memory subsystem
- Continued MCU role for zHelix; appointed zNest bringup lead for zHelix
- Currently zNest bringup lead for zSphinx

- Married Leebah Nechama in 2004
- Daughter, Simcha Rivka, 8
- Son, Aharon Eliezer, 6
- Currently living in Waterbury, CT
- Hobbies include astronomy, entomology, and retrocomputing

What is RAIM?

- **RAIM is an acronym meaning 'Redundant Array of Independent Memory,' *á la* RAID**
- **It is a combination of an additional physical memory channel, a complex recovery architecture, and a very complicated ECC codepoint**
- **It takes up a large slice of real estate on the chip, verification is very difficult, and changes have been sometimes been prohibitive due to complexity**



- **But it's saved our bacon. More than once. Since zGryphon the rates of customer UIRA events in the memory space have dropped dramatically.**
- **It means customers are happier and IBM is saving money**
- **What next? Where can we improve?**

"RAIM is the greatest thing since sliced bread."

– Jim Murray

zHelix Memory Controller – RAS View

Layers of Memory Recovery

ECC
- Powerful 90B/64B Reed Solomon code

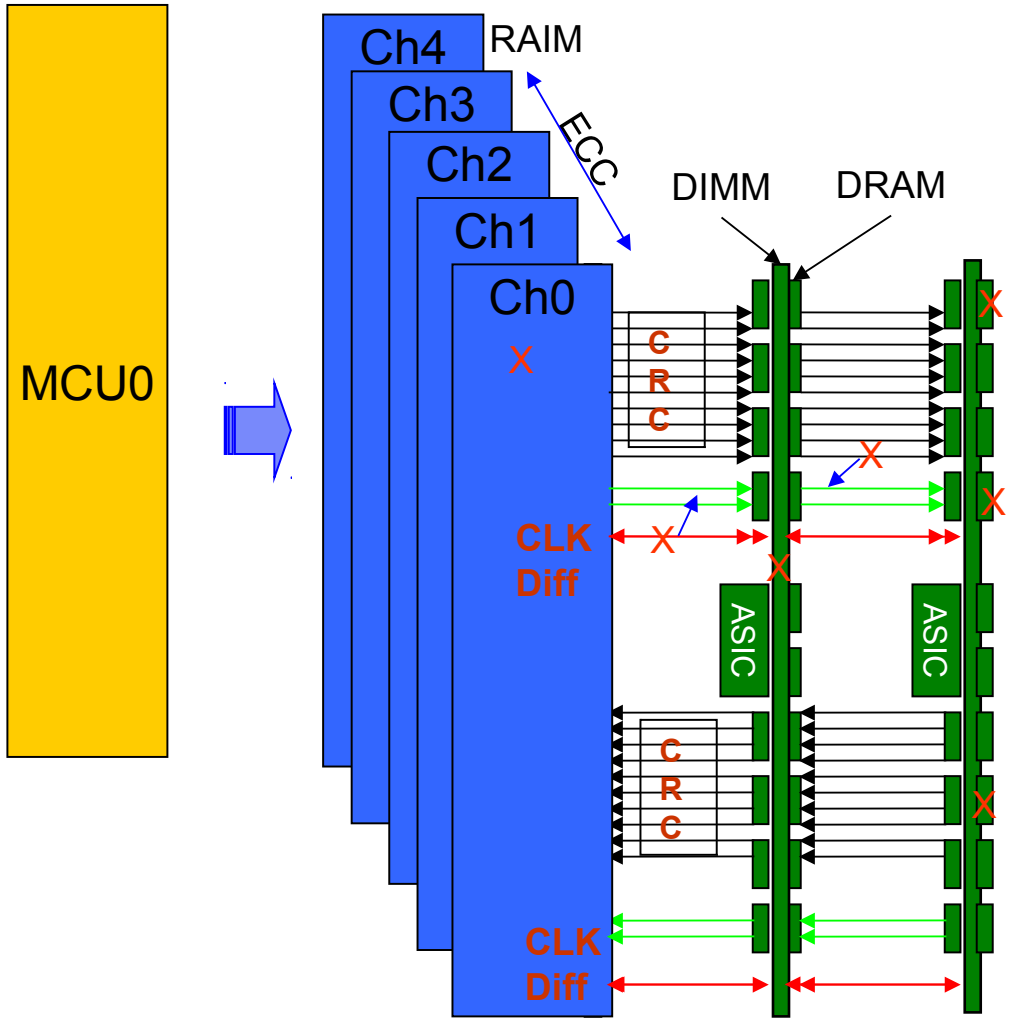
DRAM Failure
- Marking technology; no half sparing needed
- 2 DRAM can be marked
- Call for replacement on third DRAM
- DRAM errors in multiple channels can be corrected, even with mark present

Lane Failure
- CRC with Retry
- Data – lane sparing
- CLK – RAIM with lane sparing

DIMM Failure (discrete components, VTT Reg.)
- CRC with Retry
- Data – lane sparing
- CLK – RAIM with lane sparing

DIMM Controller ASIC Failure
- RAIM Recovery

Channel Failure
- RAIM Recovery



zHelix MCU Recovery: ECC/RAIM Protection

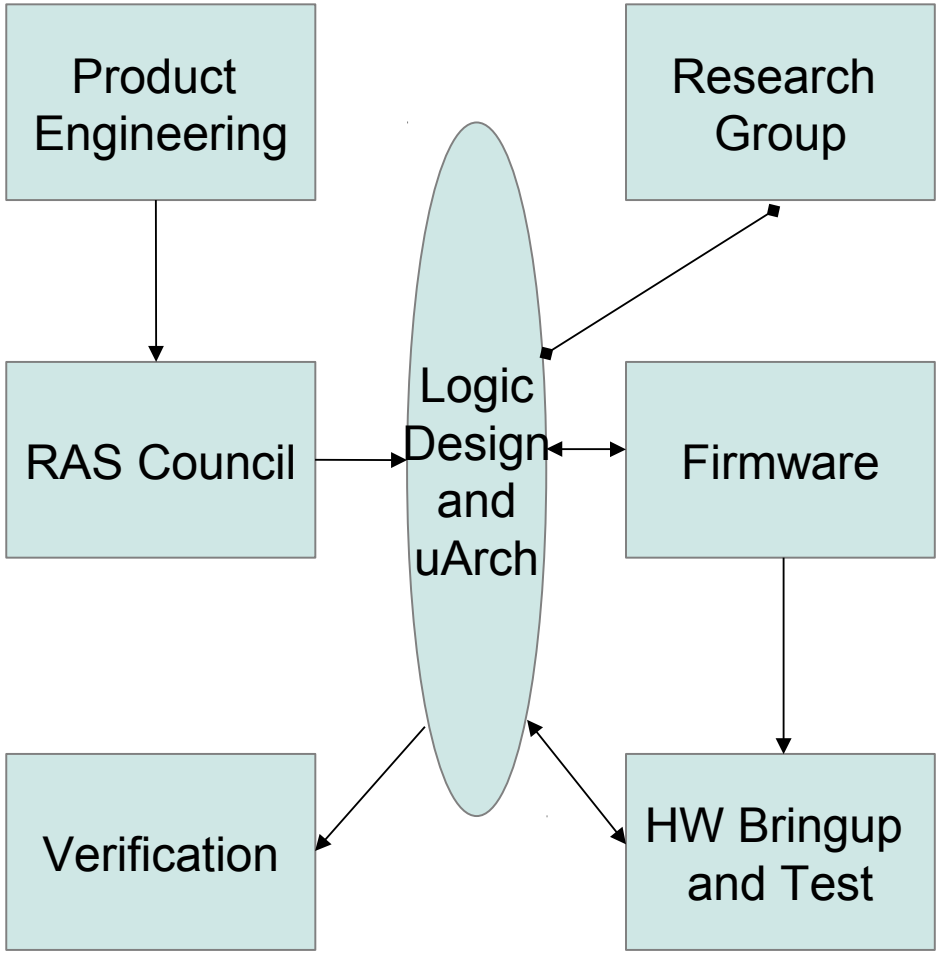
Marks/New Errors	No Marks	Single Chip Marked	Two Chips Marked	DIMM Marked or 3+ Errors
None	GOOD	GOOD	GOOD	GOOD
One Chip	CE	CE	CE	Service Request
Two Chips, Same or Two Channel(s)	CE	CE	CE	Call home for part replacement
Full Channel Error (CRC or other)	CE	CE	CE	Call home for part replacement

- No need for spare chips! If we 'know' in advance of a location where errors are extant, we 'mark' it. This can be applied on a DRAM or a channel basis. Marking ahead of time allows us to find new errors on top of the known ones.
- Marks are much more flexible than spare DRAMs, and can keep the card cost down as well, to say nothing of the reduced design cycle given simpler verification

- If we ever reach a threshold where the system can no longer correct dynamically, we 'call home' to let IBM know to replace the defective part; the system continues to operate without performance penalty in the interim
- Firmware is always looking at the health of DRAMs in the background

How big an effort was this thing?!

- **In a word, huge. Really, staggeringly, colossally *huge*.**
- **Nineteen unique RLMs; 42 instantiations at unit level**
- **58 VHDL components for performing GF(16) or GF(2^8) arithmatic operations**
- **Thousands of latches**
- **Tens of millions of gates**
- **Multiple cycles to encode and decode data; two parallel decoders**
- **Very significant PD deep dive to solve pervasive timing problems**
- **Two months verifying the code**
- **One month implementing initial logic**
- **Five months RTX effort**
- **Four months formal verification effort**



And then think of all the people involved! Three researchers, two verification engineers, two circuit and timing guys, a perpetually frustrated unit integrator, and an exhausted (but very happy) designer and his plucky Honda Civic racking up the miles between POK and Watson in Yorktown Heights.

What's the issue with existing verification technologies?

Exhaustive checking:

Check all possible combinations of input bits.

Normally ECC circuits have several hundreds of input bits. Checking of 2^100 cases will take time till the end of the Universe.

Random sampling:

Check 1 trillion of random combinations of input bits.

99% of bugs hide in “corner” cases. For example a bug may show up only when the first half of the inputs are Boolean inverses of the second half. Random sampling will never detect such a bug.

Formal verif at bit level:

Produce a formal proof of correctness for all combinations of inputs simultaneously.

The problem is exponentially hard in the number of inputs.

Each particular problem can (possibly) be solved in a reasonable time by trying a number of specialized methods.

SIXTH SENSE
is a really powerful tool utilizing this approach, but at BIT LEVEL ONLY.

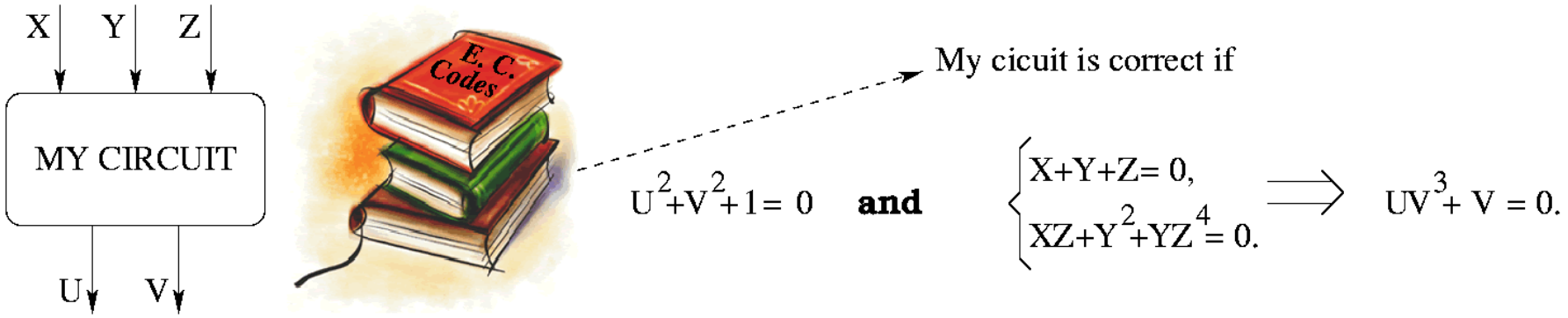
And that's the gate to innovation: “The Size of Verification”

- **The RAIM design has been with us, more or less unchanged, since zGryphon**
- **The massive effort involved in creating something new simply isn't worth it to the business: the time involved, the resources required**
- **Several points during the design cycle in both zGryphon and zHelix there were moments when we knew of deficiencies present or improvements that could be made, but the impact to the schedule was simply too great**

- **But we're IBMers! Innovation is our creed! We can not accept limitations of this sort.**
- **So what was the real gate? Verification. Every time.**
- **Minor changes to the code would require retooling the verification environments and rerunning testcases, especially in the case of the formal verification effort**
- **We could get around circuit issues, we could get around timing issues. But the 'time to verify' was simple too much. So we punted. More than once.**

- **But.... what if we could eliminate those issues? What if verifying the code took not months, but days? What if minor changes could be verified in not days or weeks, but minutes? What if we could do this without a dedicated formal verification engineer?**
- **It would mean we could prototype code implementations much, much more quickly. It would mean the designer could innovate to his/her heart's delight, being able to verify the changes on their own.**

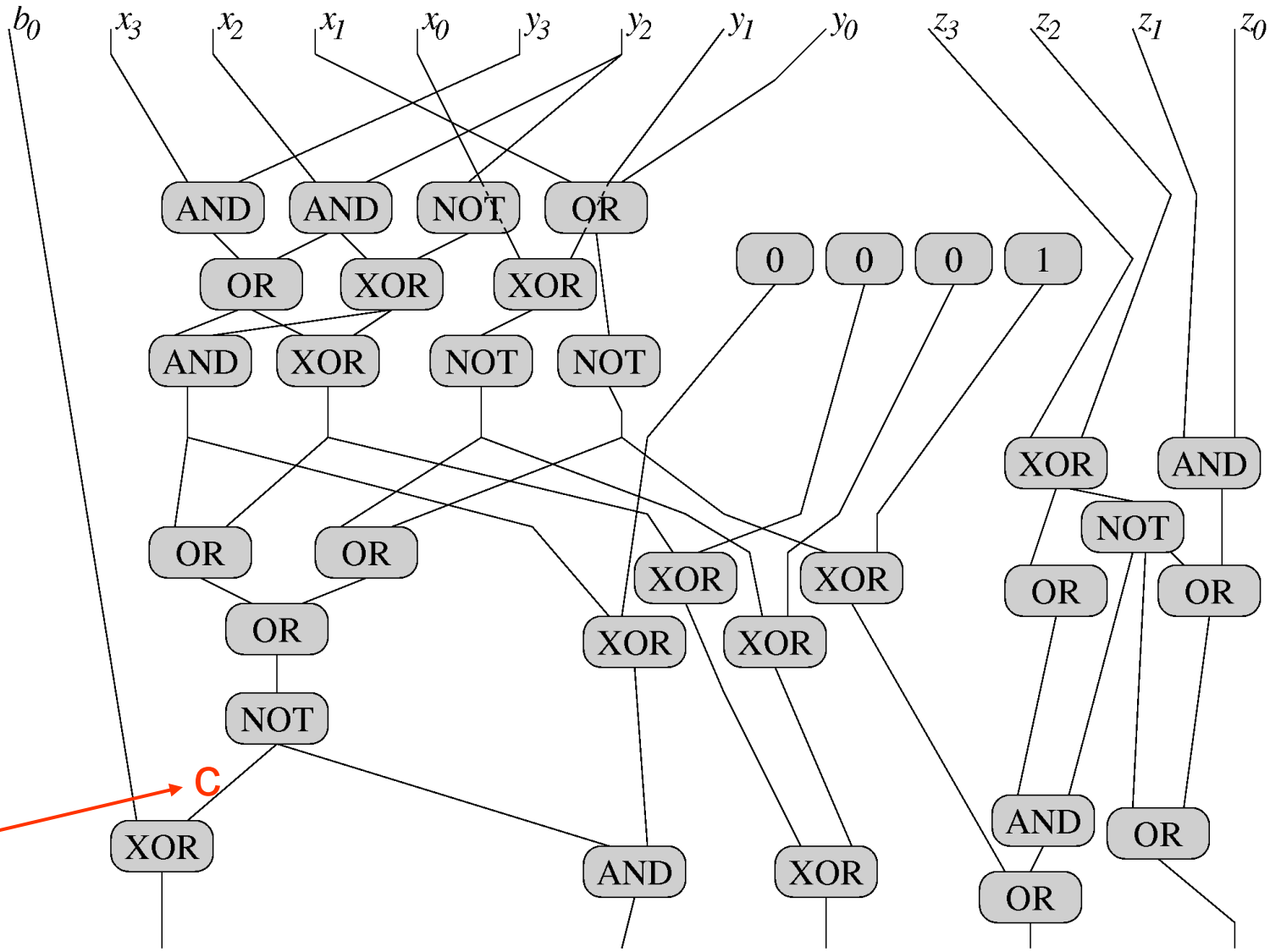
BLUEVERI:
Reason at the level of Galois Field elements themselves!



- **Advantages: Unbelievably effective for error correcting codes verification. Easily verifies ECC with thousands of input bits. Simplifies logic implementation in VHDL substantially. Much, much faster than any other verification method to date.**
- **Disadvantages: Narrowly specialized. Only applicable to circuits based on Galois fields algebra. This, though, includes nearly all ECC codes and many schemes used for data encryption/decryption in hardware**

- ***Less time to verify, less logic to write, less resources required across the project. Novel codes can be implemented and verified far more quickly than in zGryphon!***

Verification in a bit-oriented world....



Prove that if

x_3 xor y_3 = 0,

x_2 xor y_2 = 0,

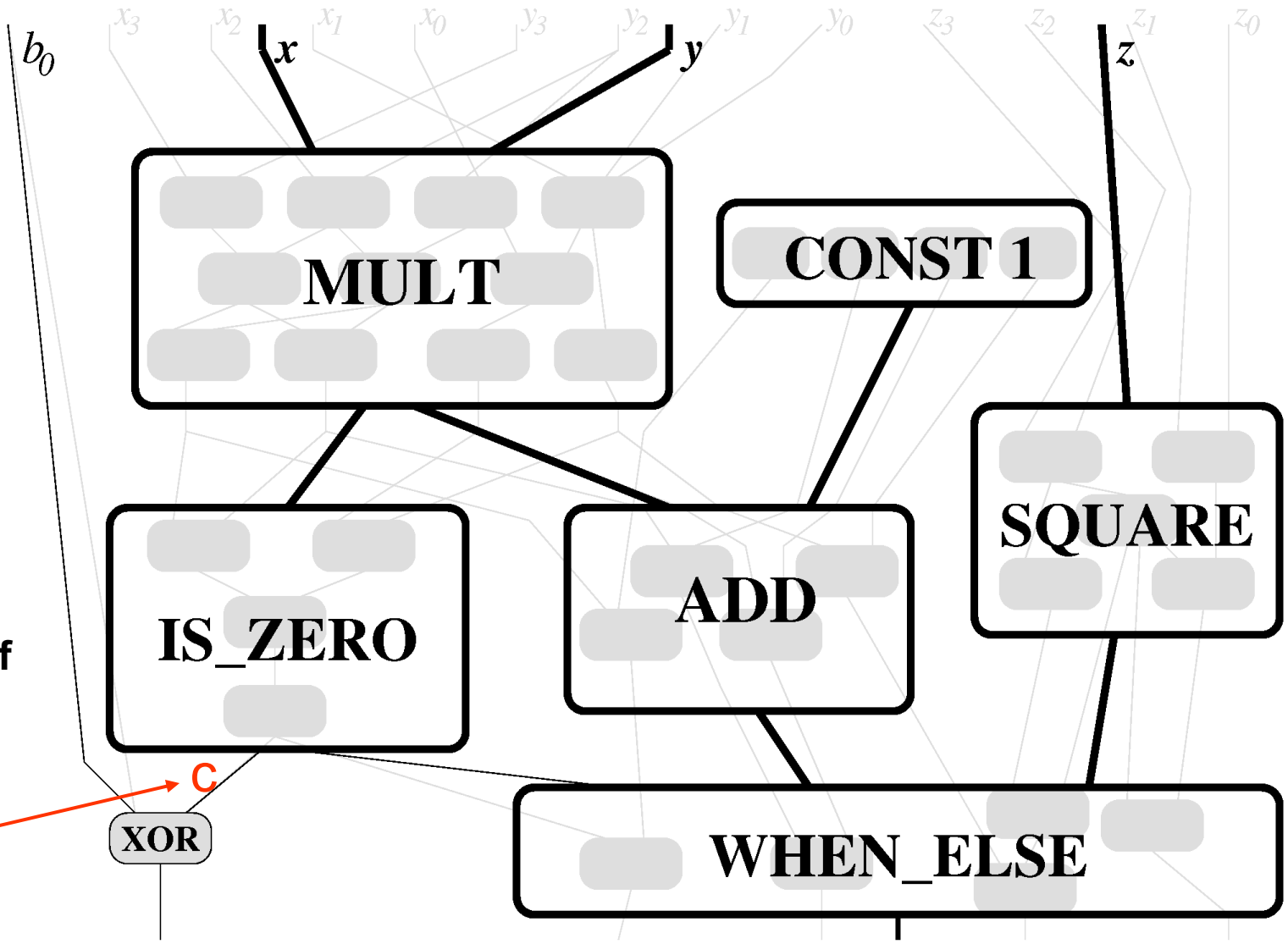
x_1 xor y_1 = 0,

x_0 xor y_0 = 0

then

c = 1.

Verification in Blueveri....



Prove that if

x + y = 0

then

c = 1.

Verification in Blueveri (part 2)

- **Blueveri is capable of looking at a complex design and 'understanding' what the various gates actually do mathematically.**
- **It also understands basic combinatorial logic which is not operating on GF(2k) symbols, such as basic control logic and dataflow controls**

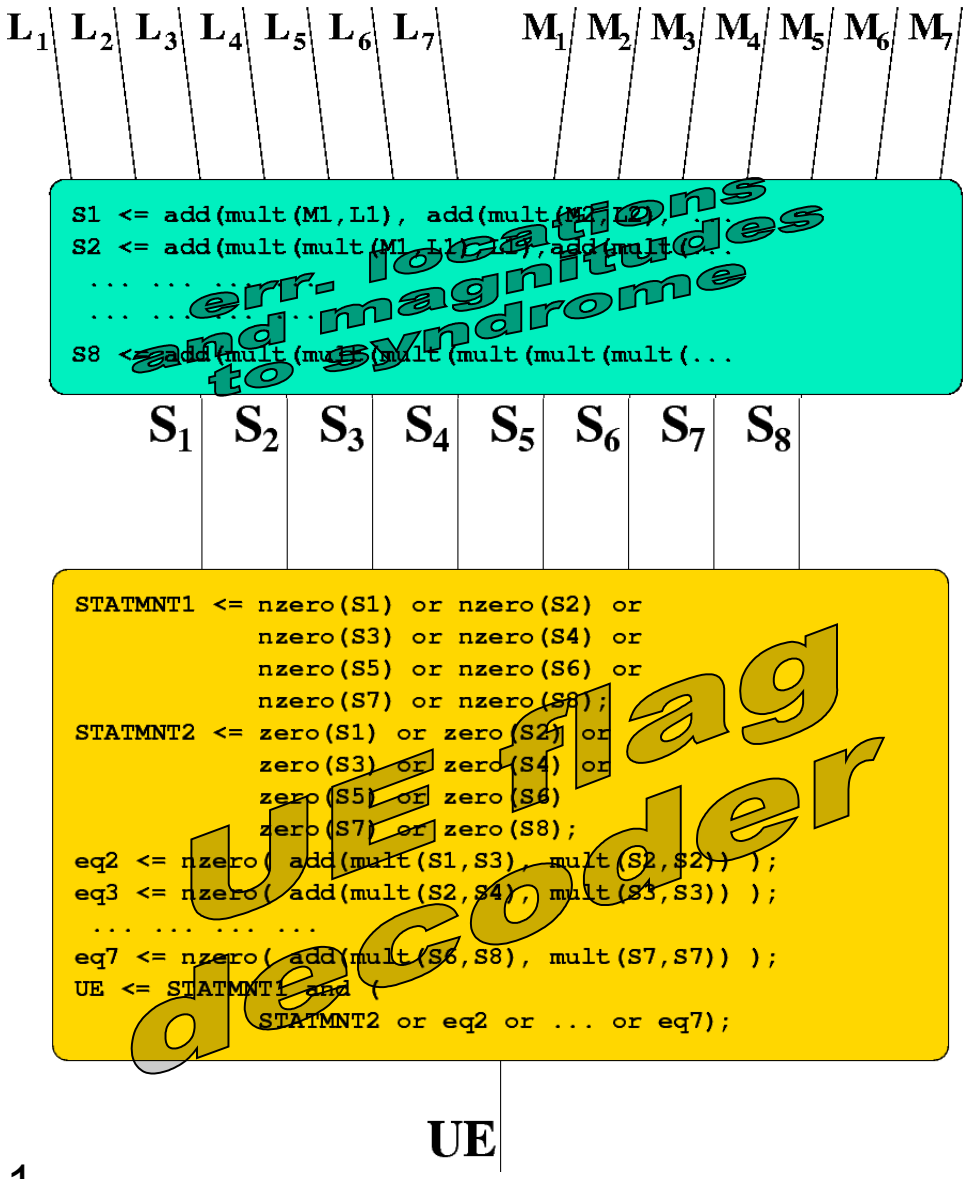
Galois Field operations

- **ADD, MULT, DIV operations. In fact any fixed set of unary operations on GF(2k) symbols which are linear over GF(2)**
- **So that means things like SQUAREing operations. Or roots. Or bit-level permutations.**
- **It means it can digest conditional gate logic using when/else statements or simple AND/OR logic**
- **It understands the concepts of dynamic symbols and constants**

Boolean operations

- **Boolean operations are broken down into elementary operations such as AND, OR, NOT, and XOR gates**
- **Tests on signals for constant values are also supported, so it 'sees' signals as being constant zero or constant non-zero when building the graph used internally as it walks through the design to minimize runtime**

Example Using Blueveri: Simple Circuit and Check




```
BEGIN_CHECK "THREE ERRORS"
  ALGEBRAIC_CONSTRAINTS_ON_GF_INPUTS;
     M1 != 0;
     L1 != 0;
     M2 != 0;
     L2 != 0;
     M3 != 0;
     L3 != 0;
     L1 != L2;
     L1 != L3;
     L2 != L3;
     M4 == 0;
     M5 == 0;
     M6 == 0;
     M7 == 0;

  BIT_EXPECTED_VALUES
     UE  must be  '1';

END_CHECK
```

Note: L, M, and S in this example are all multi-bit GF symbols

Example Using Blueveri: Performance Comparison

UE Flag Decoder

symbol errors	expected UE	symbol size	input bits	Blueveri	Sixth Sense
1	no	8 bits	16	Success after 0.1sec	Success after 14sec
2	yes	8 bits	32	Success after 1sec	Gives up after 24hours
3	yes	8 bits	48	Success after 1sec	n/a
4	yes	8 bits	64	Success after 33min	n/a

Error Magnitude Computation

errors	symbol size	input bits	Blueveri	Sixth Sense
2	8 bits	32	Success after 2sec	Gives up after 24hours
3	8 bits	48	Success after 2.1sec	n/a
4	8 bits	64	Success after 2.1sec	n/a
5	8 bits	80	Success after 2.3sec	n/a
6	8 bits	96	Success after 3.1sec	n/a
7	8 bits	112	Success after 49.4sec	n/a
8	8 bits	128	Success after 8min	n/a
9	8 bits	144	Success after 53min	n/a

- **Hours become minutes; minutes become seconds; solutions previously not possible are now reachable**
- **This means changes, minor or major, to both the underlying code and implementation can be verified much, much quicker**
- **The check syntax is easy for the designer to use and understand and the simplified VHDL syntax now possible is also a time-to-completion boon**

Summary

- **RAIM has been of major benefit to IBM, both financially and in terms of customer satisfaction and competitive advantage over Intel and HP**
- **To duplicate our effort with RAIM would take a designer with knowledge of linear algebra and finite field analysis, formal and random verification efforts, and significant resistance to new features added or changed during the design cycle due to impact to verification**
- **Blueveri solves all of this: it makes the design easier for the designer, it allows rapid and formal proofs of a code and its underlying implementation, and it reduces the personnel requirements**
- **Blueveri can be a key component to further radical RAS innovation in the memory space going forward**

- **Blueveri is already being used in ECC logic delivered by IBM Research including in POWER8 and in the L4 quad ECC for zSphinx**
- **In addition as a side project Blueveri is being used to formally verify the zSphinx RAIM implementation to find areas where Blueveri can improve; new features were added to the tool during this exercise and it continues the deep cooperation on RAIM between IBM Research and IBM STG begun back in 2008**