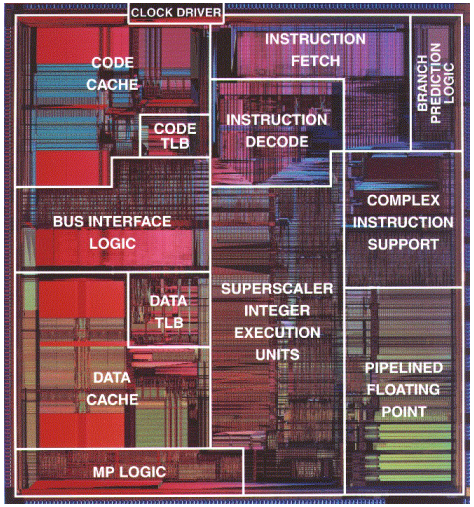
Chips with Confidence: Formal Datapath Verification

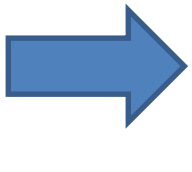
Dr John O'Leary
Principal Engineer, Intel Corporation
john.w.oleary@intel.com

Outline

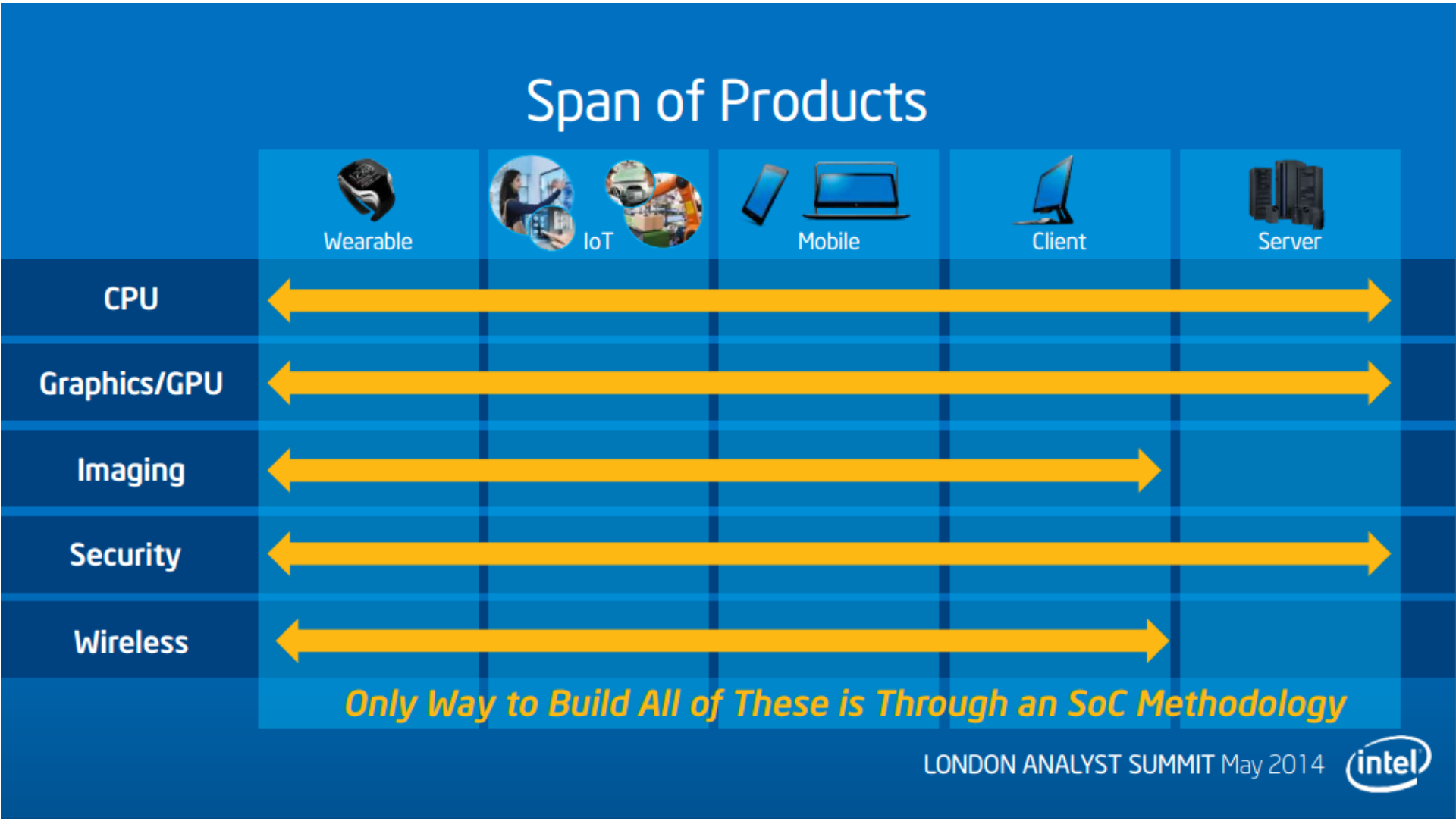
- What we are verifying and why
- Verification technology
- Verification technology + X = Impact
  - Specifications
  - Methodology
  - Reuse

It used to be so simple

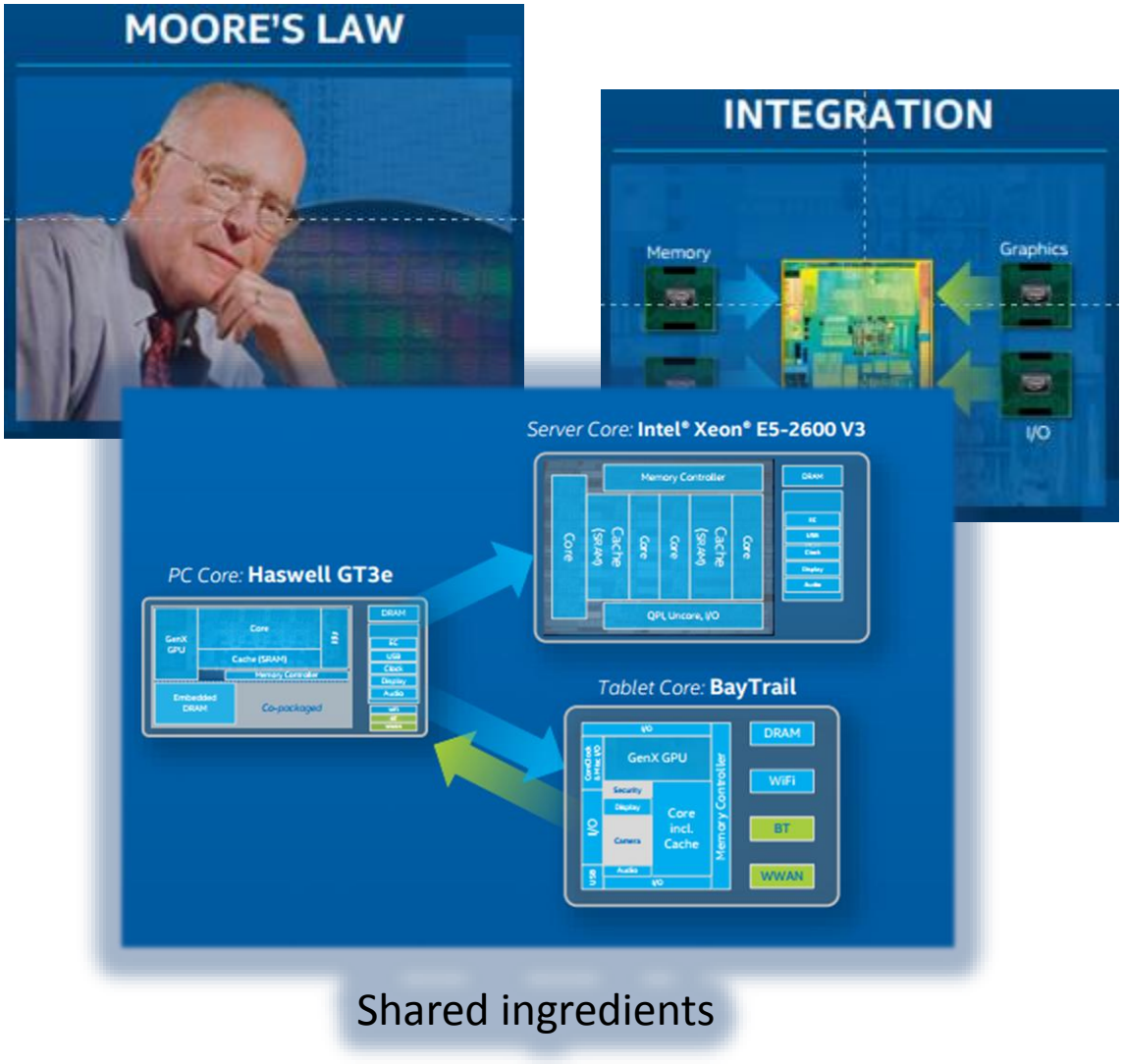




Now it is not so simple

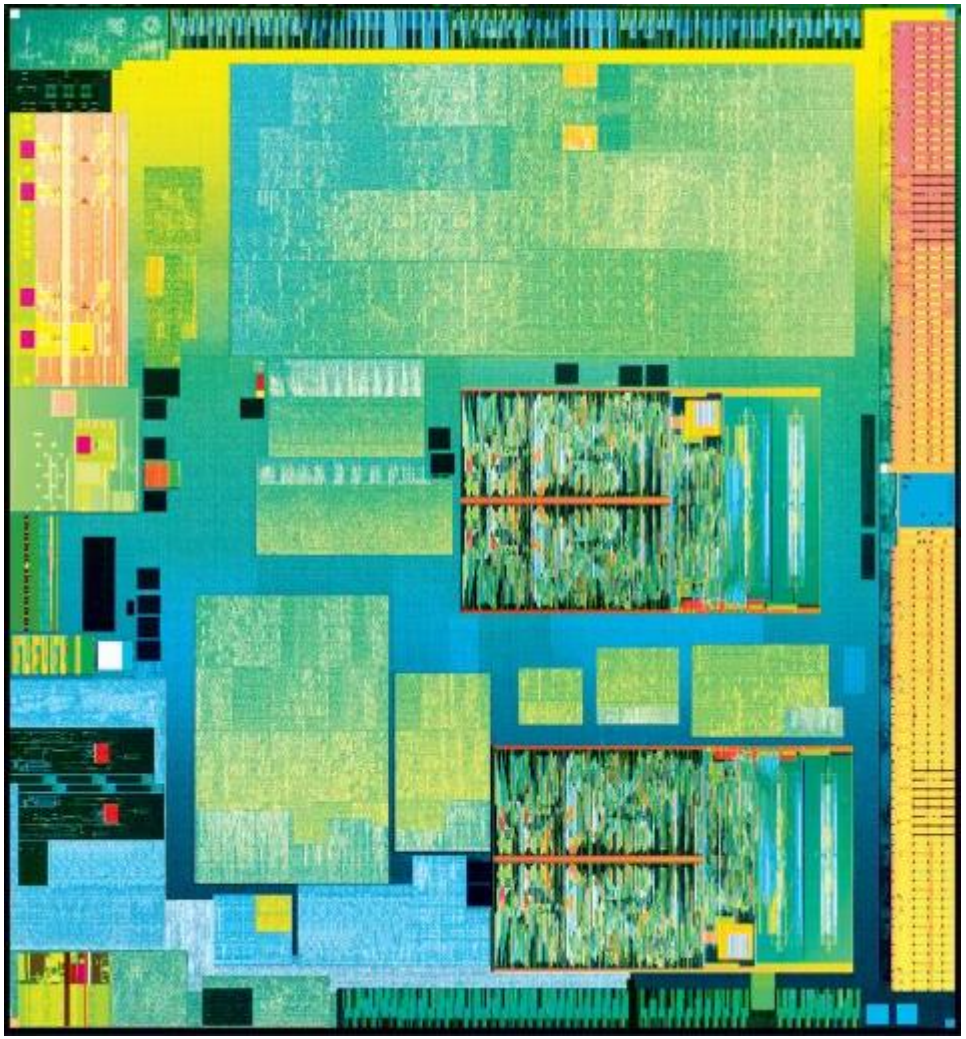


SoC methodology

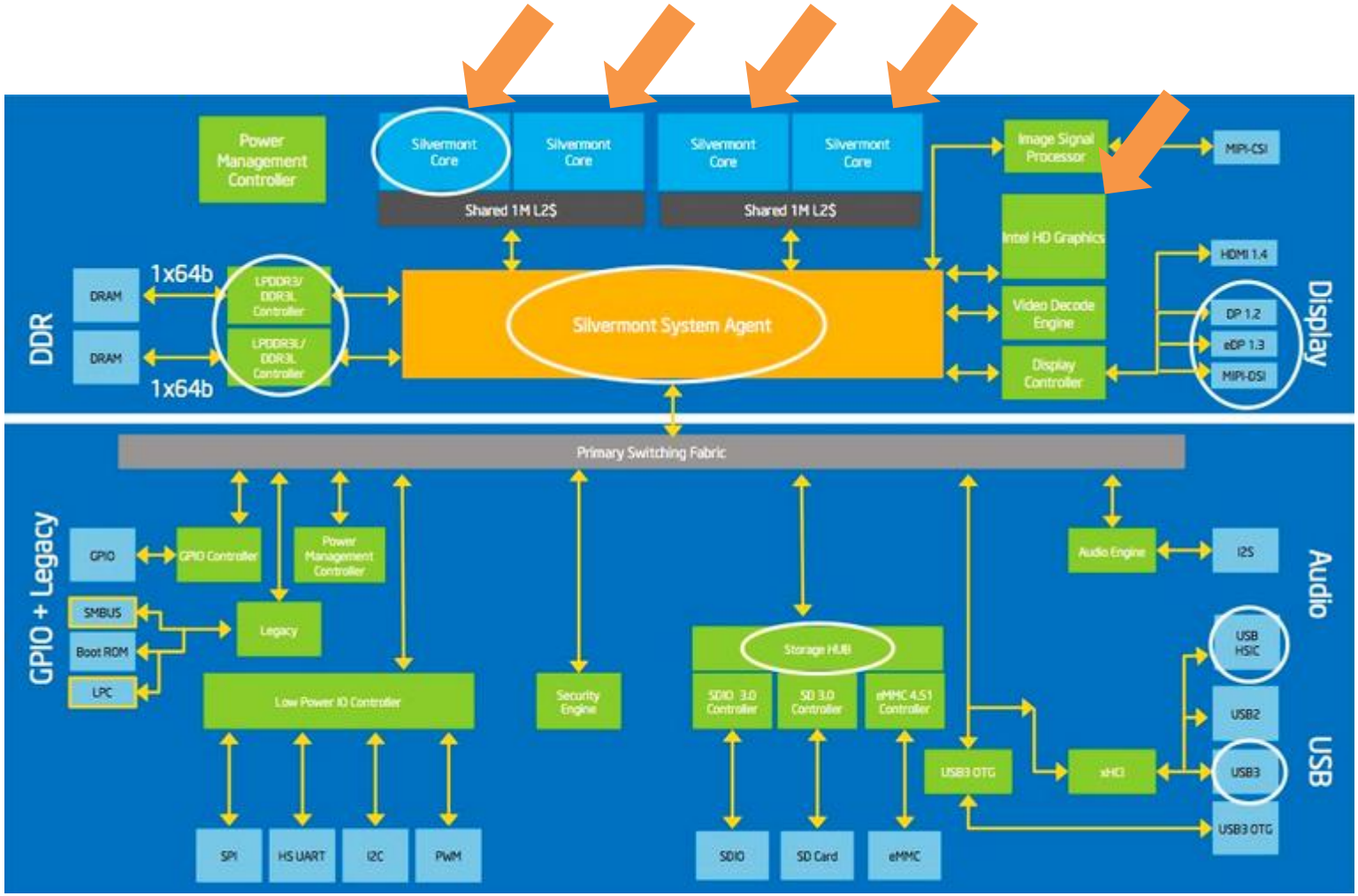


Shared ingredients

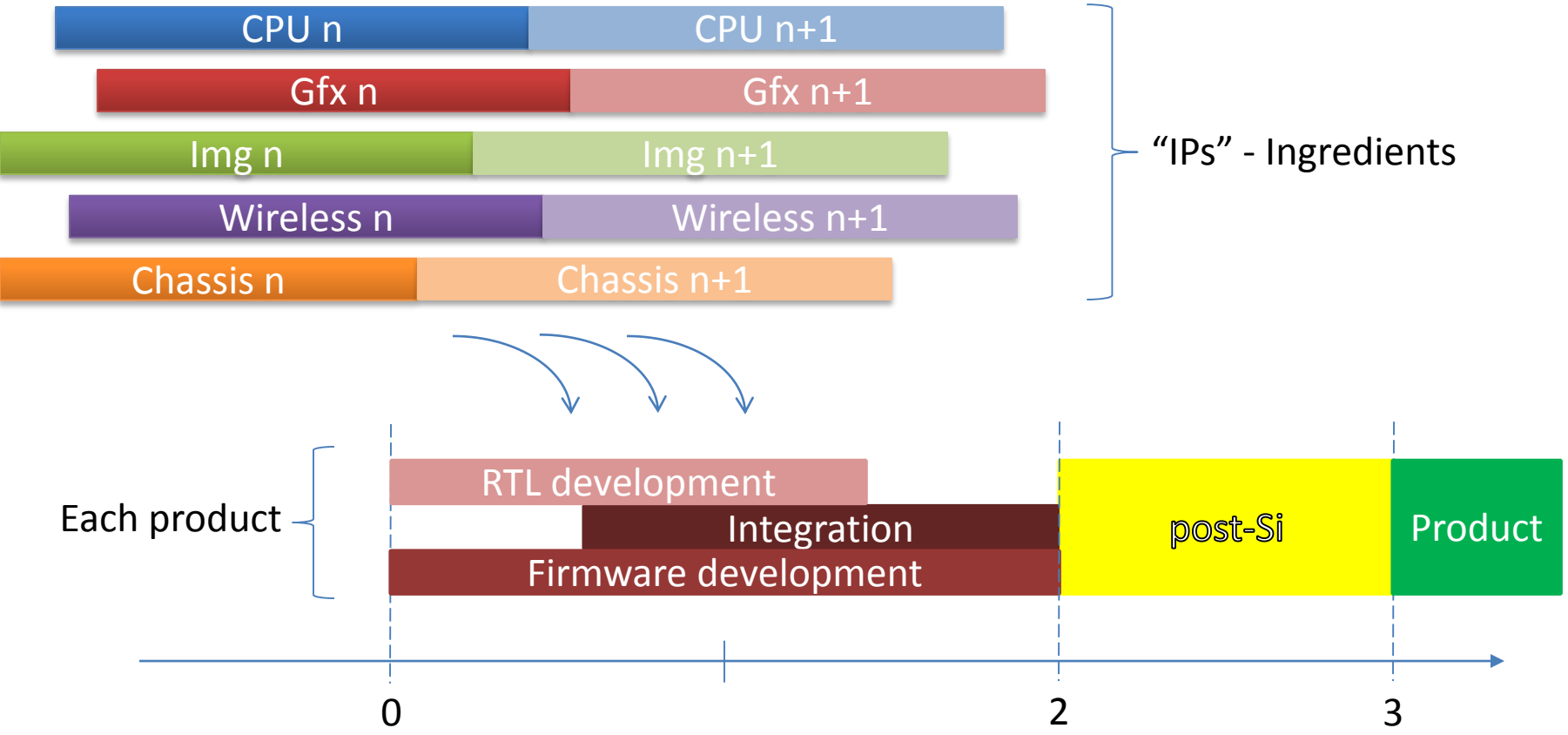
Ex: Bay Trail SoC



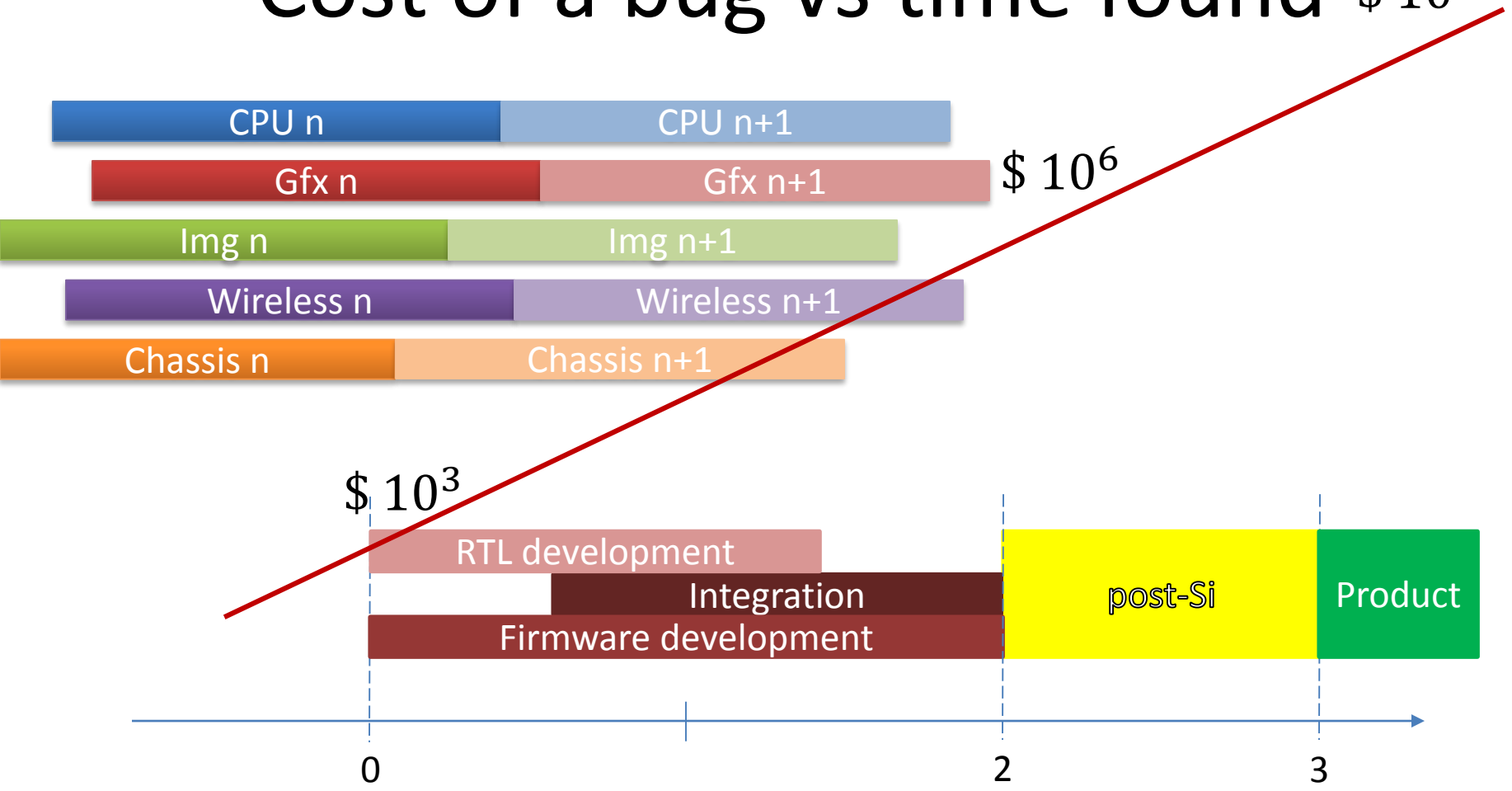
Bay Trail: User-visible datapaths



Development with shared IPs



Cost of a bug vs time found



Is a $1B bug possible?

- Mid-1994 – FDIV flaw detected in the Intel Pentium® processor
- January 1995 – Intel announces a pre-tax charge of $475 million against earnings
- A 2003 analysis estimated that a similar escape could cost $12B, at then-current product volumes

Outline

- What we are verifying and why
- Verification technology
- Verification technology + X = Impact
  - Specifications
  - Methodology
  - Reuse

Formal tools at Intel

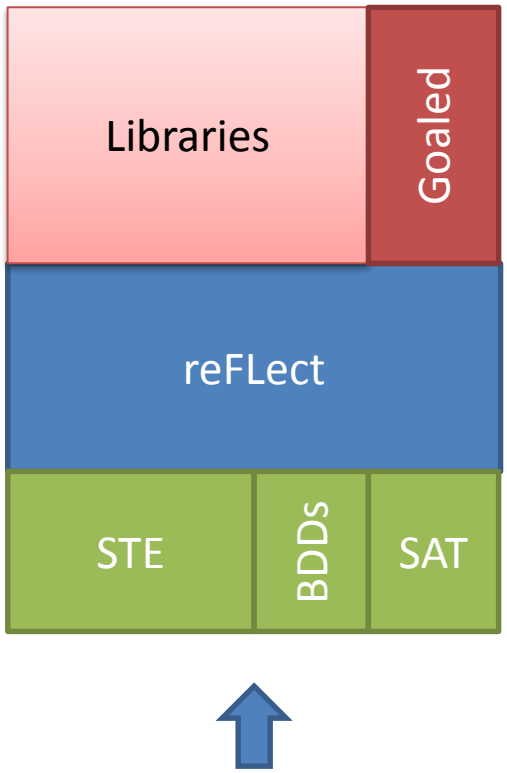
- RTL-vs-RTL and RTL-vs-schematic equivalence verification
  - Widely deployed across the industry
- RTL assertion checking
  - Broad-spectrum bug detection for IP blocks, interfaces, etc
  - Principally bounded model checking using SAT-solvers
- Datapath verification
  - Applied to virtually all Intel CPU + Gfx datapath designs
  - Similar approaches used in Centaur, IBM, AMD
- Architecture, microarchitecture (cache protocols, etc)
  - TLA+/TLC, Murphi, Spin
- Software and firmware
  - Static analysis widespread (e.g. Klocwork)
  - Some BMC, concolic testing, theorem proving

A brief history of datapath verification technology at Intel

- 1990-1994 – Intel/SRC funded academic research on BMDs, HDDs, word-level model checking (E.M.Clarke, R.E.Bryant); Symbolic trajectory evaluation –STE (Bryant, C.-J.H.Seger)
- Mid-1994 – FDIV flaw detected in the Pentium® processor
- Early 1995 – FV research group formed in Intel Strategic CAD Labs (SCL). Seger VF, O'Leary, Jones, Zhao hired.
- Mid-1995 – HDD-based word-level model checking demoed
- 1995-1996 – WLMC applied in SCL to verify properties of Pentium® Pro FPU functional blocks. Seger, Aagaard hired.
- 1997 – FIST bug detected in the Pentium® Pro processor
- 1997-1998 – Complete Pentium® Pro FPU verified against high-level specs, with machine-checked composition argument. Melham VF #1/N.
- 1999-present – production use of FP FV technology
- 2004 – fixpoint reached at our current tool suite

Intel's Forte system

Arithmetic operations & theorems
Intel CPU micro-instruction specs
Debugging routines



Interactive, deductive reasoning about reFLect programs

For writing formal specifications and verification scripting

Checks bounded LTL properties of RTL designs using symbolic simulation

(System) Verilog RTL

reFLect

- Higher-order, typed, lazy, functional
- Supports development of formal specifications, libraries, verification scripts, …

```
let bit_add (xv, yv) =
   letrec f [] [] = [F]
   /\     f (x:xv) (y:yv) =
            val [cin,res] = f xv yv in
            let sum = ( x XOR y ) XOR cin in
            let cout = ( x AND y ) OR ( x AND cin ) OR ( y AND cin ) in
            ( cout : sum : res )
   in
   f xv yv ;

add::(bool list # bool list) -> bool list
```

reFLect and BDDs

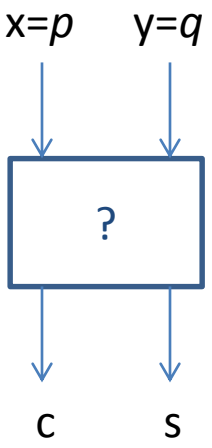
```
let a = variable_vector "a[2:0]";
let b = variable_vector "b[2:0]";

a;
[a[2], a[1], a[0]]::bool list

bit_add (a, a);
[a[2], a[1], a[0], F]::bool list

bit_add (a, b);
[…,
 b[1]&a[1]&b[2]&a[2] + b[0]&a[0]&a[1]&b[2]&a[2] +
    b[0]&a[0]&b[1]&b[2]&a[2] + !b[0]&!b[1]&!b[2]&a[2] +
    !a[0]&!b[1]&!b[2]&a[2] + ... ,
 b[0]&a[0]&b[1]&a[1] + !b[0]&!b[1]&a[1] + !a[0]&!b[1]&a[1] +
    !b[0]&b[1]&!a[1] + !a[0]&b[1]&!a[1] + ... ,
 !b[0]&a[0] + b[0]&!a[0]]::bool list
```

Symbolic trajectory evaluation (STE)



x=*p* /\ y=*q*

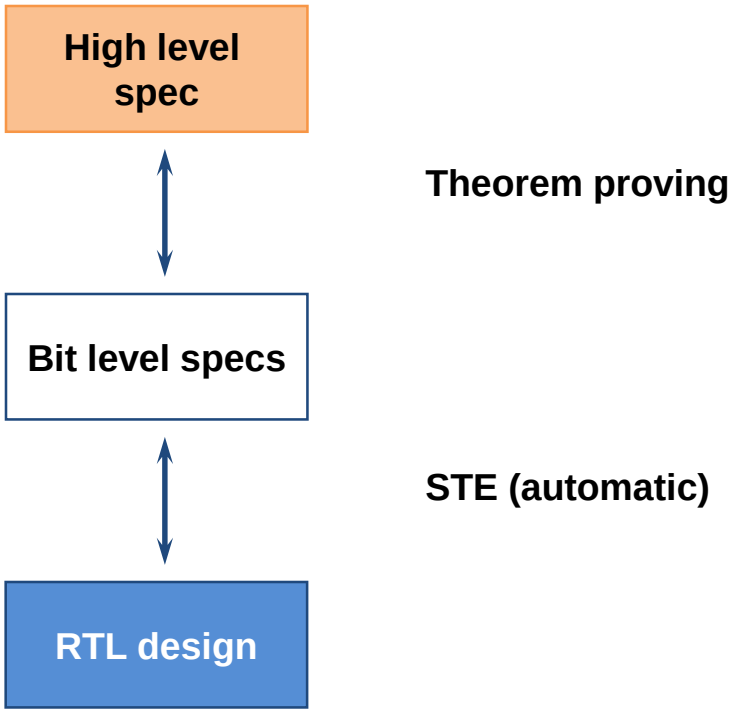
➔

[c,s] = bit_add([p], [q])

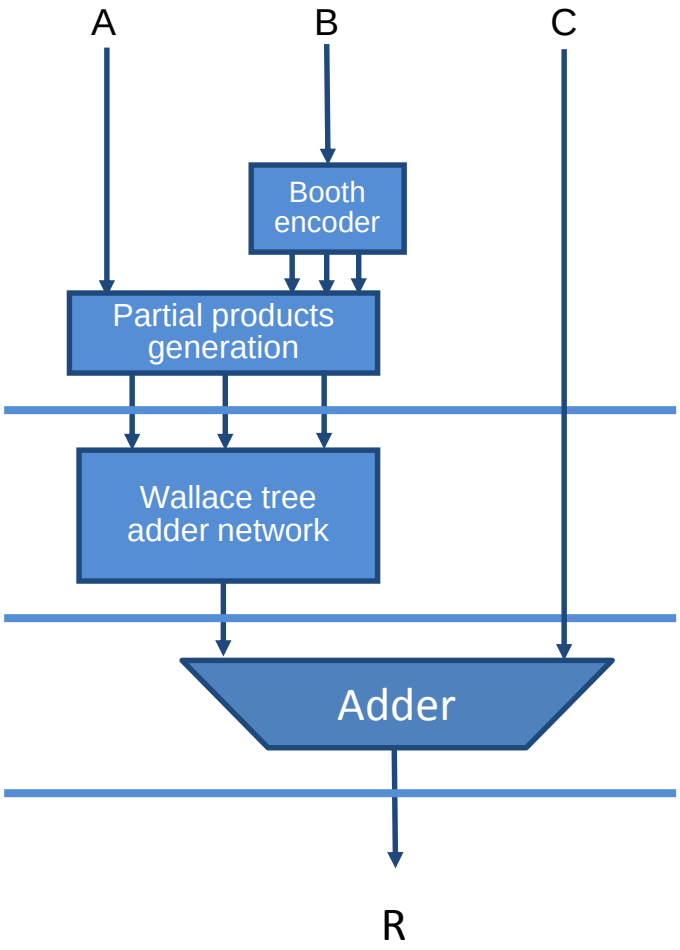
- Compute *C(p,q)*, *S(p,q)* via symbolic simulation
- Check this Boolean formula for validity:

  (C(p,q) = q&p) & (S(p,q) = !q&p + q&!p)
- Built-in abstraction: X ⊑ 0, X ⊑ 1

Approach


High level spec
Theorem proving
Bit level specs
STE (automatic)
RTL design

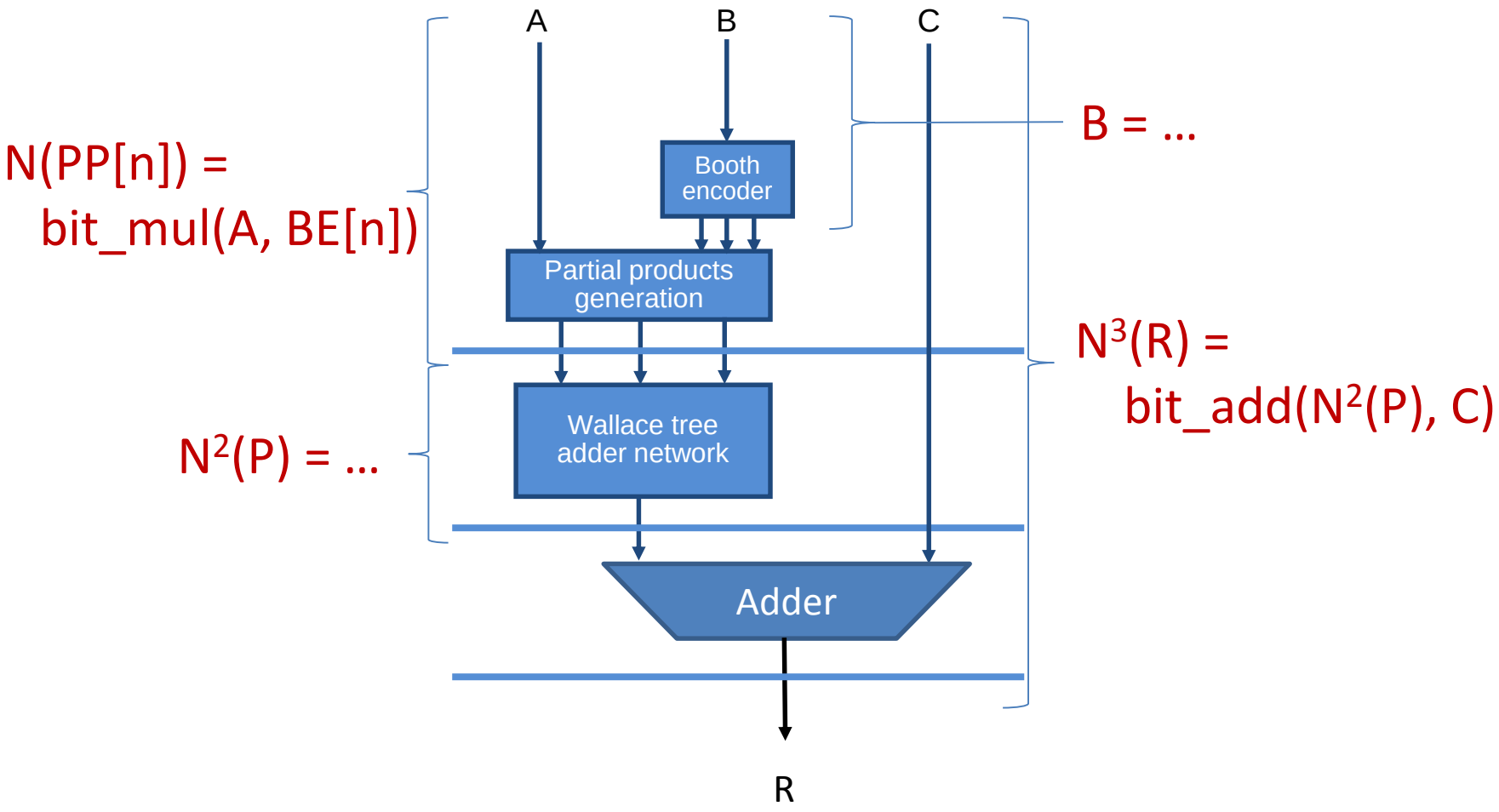
RTL design



$$i(N^3(R))) = i(A) * i(B) + i(C)$$

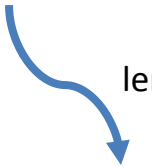
- $N^i(x)$ = "x in the i'th next cycle"
- i(x) = "bit-array x interpreted as an integer"
- *, + are the usual integer operations

Bit-level verifications with STE



Deduction connects bit-level operations to mathematics

$$\vdash \forall a, b.\, \mathrm{i}\big(\mathrm{bit\_add}(a, b)\big) = \mathrm{i}(a) + \mathrm{i}(b)$$



x=*A* /\ y=*B* ➔ N(z) = bit_add(*A*, *B*)

≡ { property of functions }

x=*A* /\ y=*B* ➔ i(N(z)) = i(bit_add(*A*, *B*))

≡ { theorem about bit_add }

x=*A* /\ y=*B* ➔ i(N(z)) = i(*A*) + i(*B*)

Deduction ensures correctness and completeness of decomposition

$i(B) = \Sigma\, i(BE[n]) * 2^{kn}$

$i(N(PP[n])) = i(A) * i(BE[n])$, for each n

$i(N^2(P)) = \Sigma\, i(N(PP[n])) * 2^{kn}$

$i(N^3(R)) = i(N^2(P)) + i(C)$

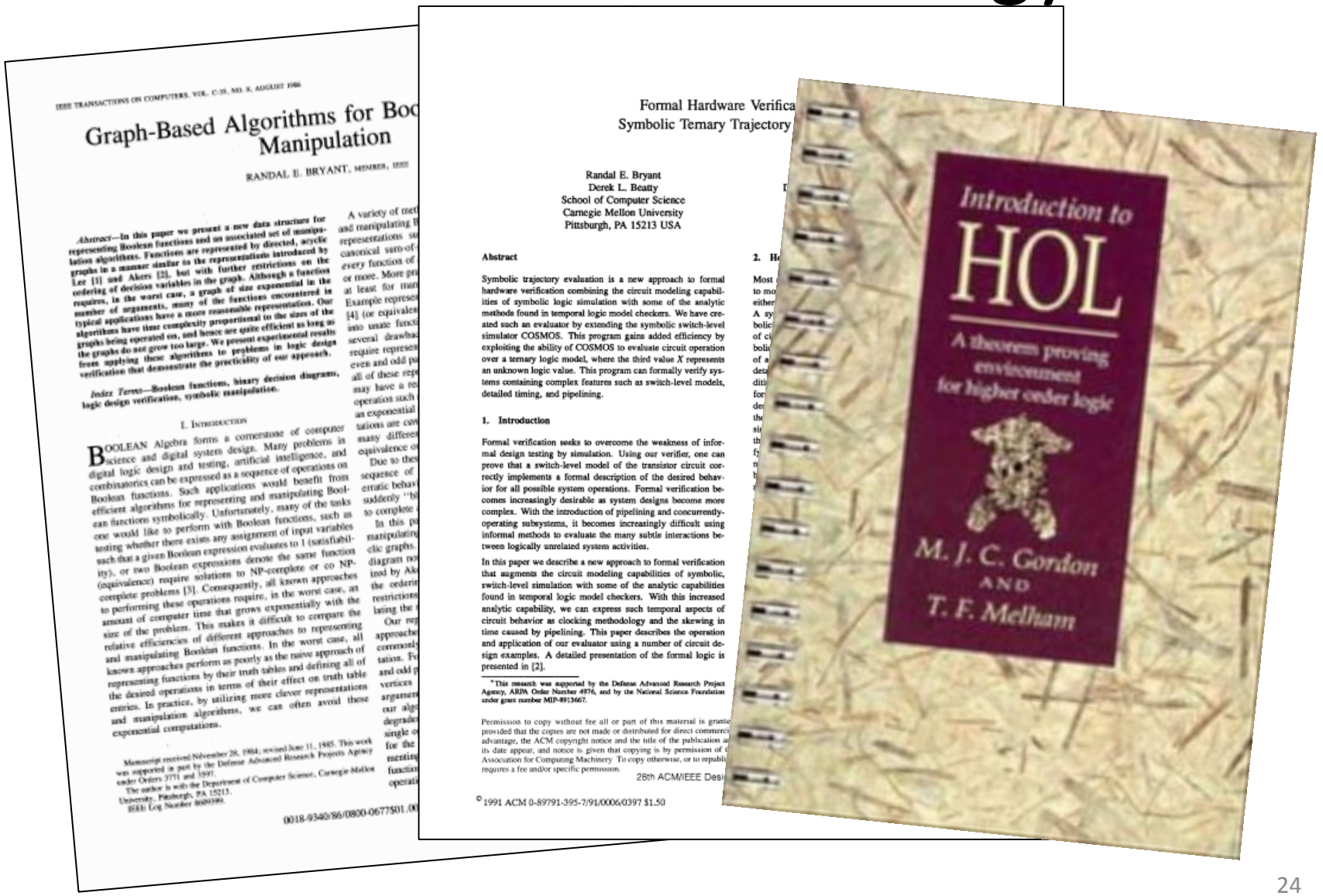
STE

$$
\begin{aligned}
i(N^3(R)) &= i(N^2(P)) + i(C) \\
&= \Sigma\, i(N(PP[n])) * 2^{kn} + i(C) \\
&= \Sigma\, (i(A) * i(BE[n])) * 2^{kn} + i(C) \\
&= i(A) * \Sigma\, i(BE[n]) * 2^{kn} + i(C) \\
&= i(A) * i(B) + i(C)
\end{aligned}
$$

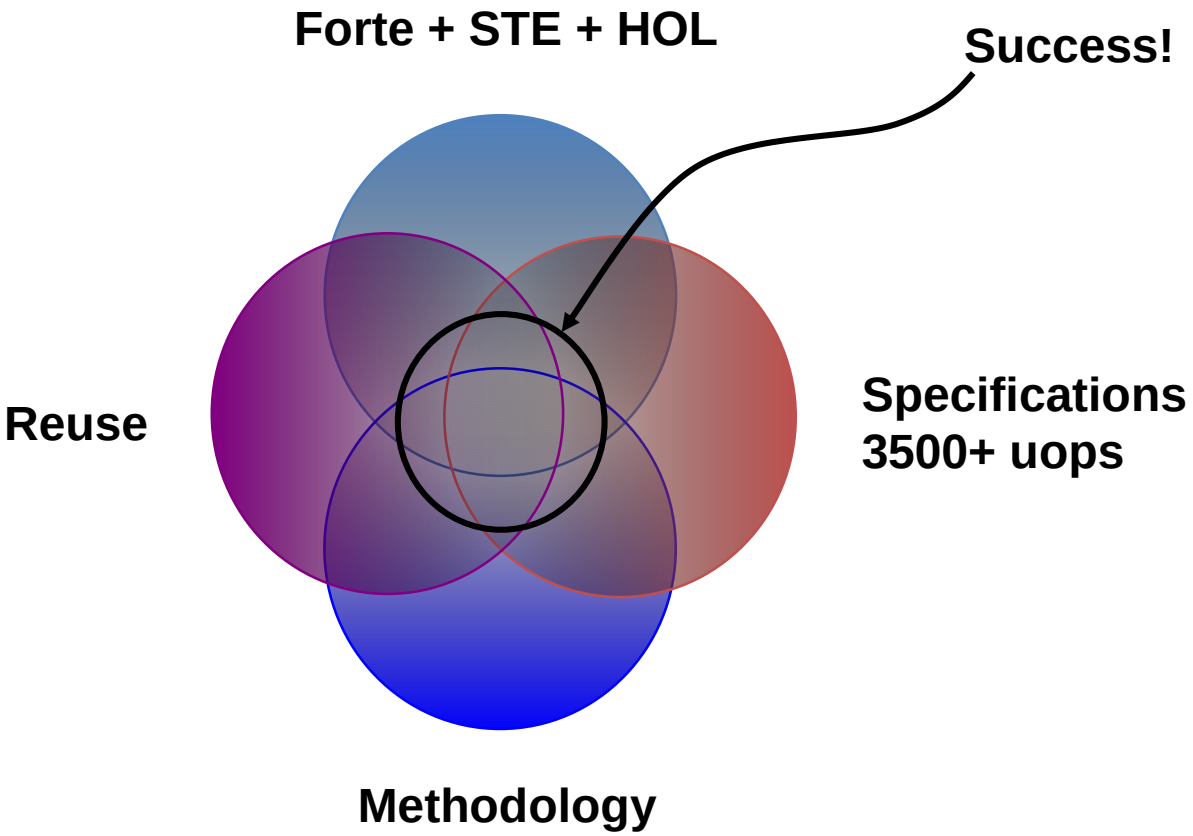
Goaled

- LCF-style interactive theorem prover, following in the footsteps of HOL and HOL Light
- Theories of reFLect data types
  - Natural numbers, integers, rationals
  - Lists, pairs, reFLect ADTs
  - Bitstring arithmetic
- Proof automation
  - Unconditional and conditional (contextual) rewriting
  - First order solver based on model elimination
  - Universal linear arithmetic over N, Z, Q

Verification technology



X: Arithmetic FV's road to El Dorado



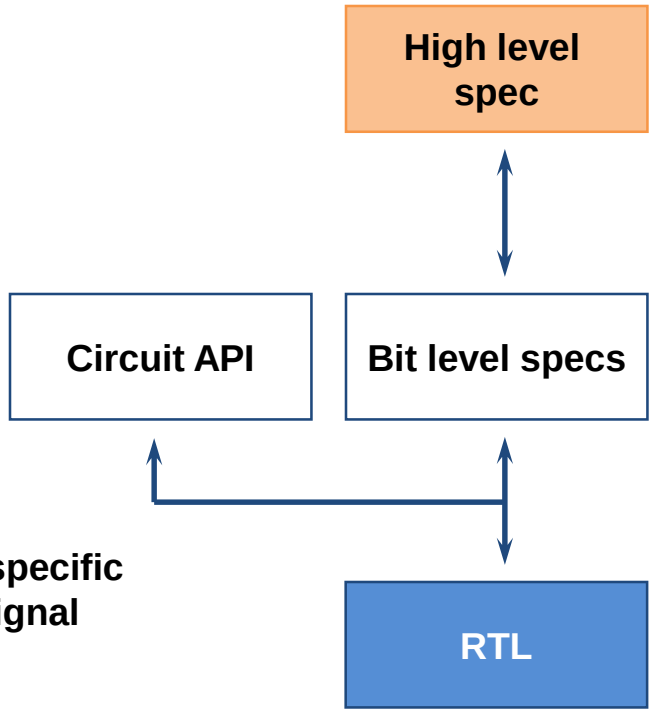
Specifications

- Clear, abstract, unambiguous specifications are more valuable than gold
  - Even if the specification is prone to change
  - Even if there is no fully-formal link to implementation
- For arithmetic datapaths we can write such specifications
- For other functionality, design intent is expressed by English text and other artifacts:
  - Tables
  - Diagrams (bubble diagrams, block diagrams, message sequence charts)
  - Pseudo-code
- Unanswered questions:
  - How can we improve specification quality for non-arithmetic?
  - Can we practically check
    - self-consistency of specifications
    - Firmware/software/hardware implementations against specifications?

Environment specifications

- Part of every spec is an accurate-enough environment model
  - For hardware blocks, this means modeling neighboring blocks
  - For software routines, this means modeling the caller, subroutines, library functions, etc
  - For firmware the environment might be both software and hardware
- The effort of writing environment models limits the uptake of formal methods
- Research needed:
  - Automatic abstraction of environment models from interfaces and code
  - Synthesis of environment models from simulation traces
  - Environment modeling at the hardware/firmware interface (esp timing)

Methodology

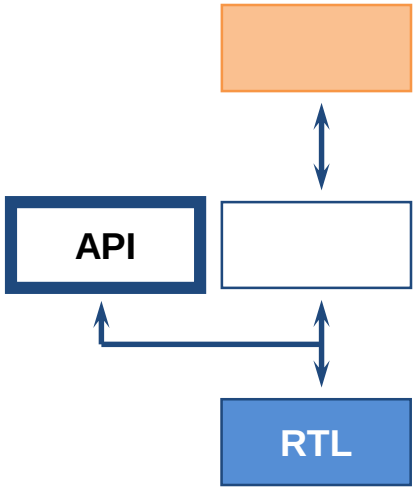


Theorem proving

STE (automatic)

API factors design-specific information about signal names, timing, ...

Circuit API



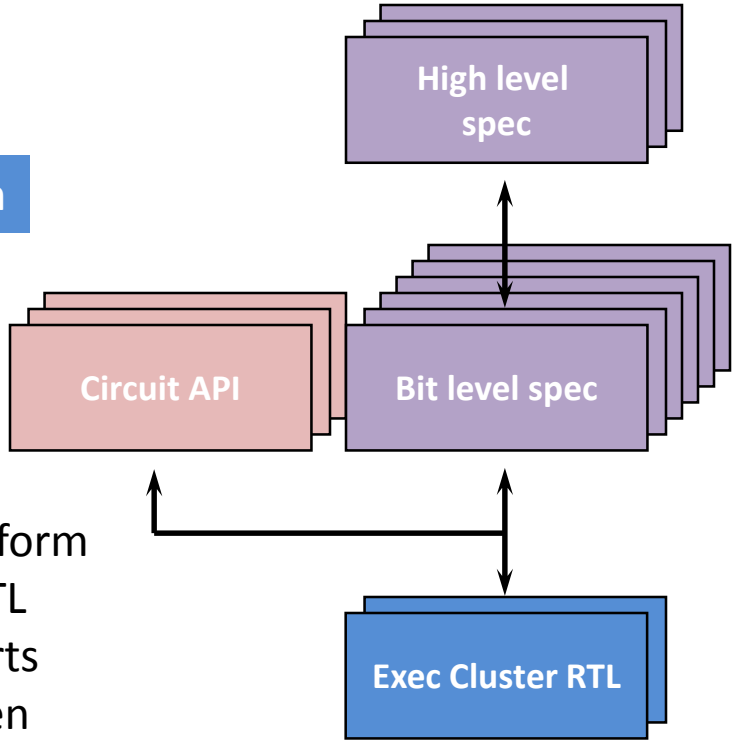
- The glue between bit-level specs and RTL
- Isolates signal names, timing, …
- Developed per-design during the initial phase of verification and evolves with RTL changes

Exec cluster verification methodology

Functionality

Environment

Implementation



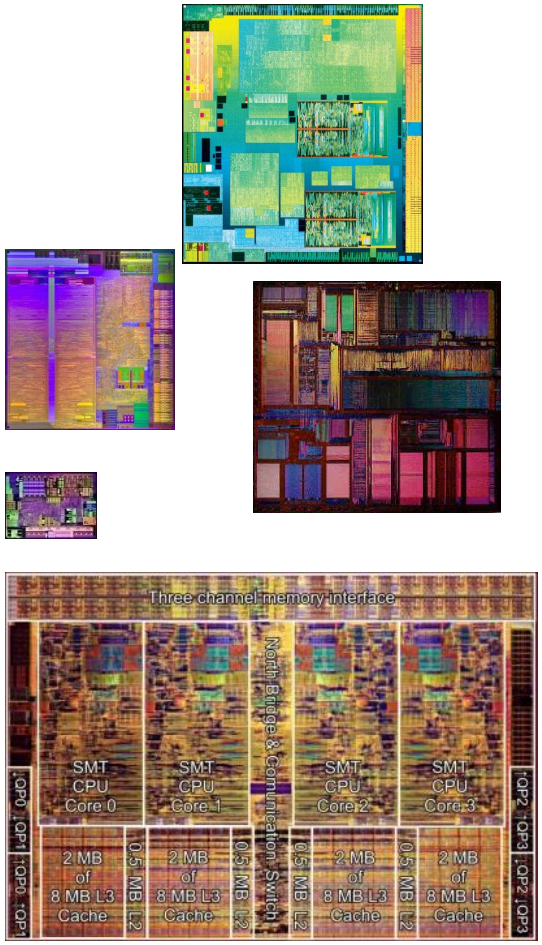
Thousands of micro-operations

API presents a uniform interface to the RTL design and supports portability between design projects

See Roope Kaivola, *et al*. Replacing testing with formal verification in Intel Core™ i7 processor execution engine validation. CAV 2009.

Methodology enables wide deployment

- Tens of designs in progress
- 100’s or 1000’s of operations per design
- Live RTL, changing frequently until a few weeks before tapeout
- Specs and scripts > 1M LOC
- “Verification engineering”



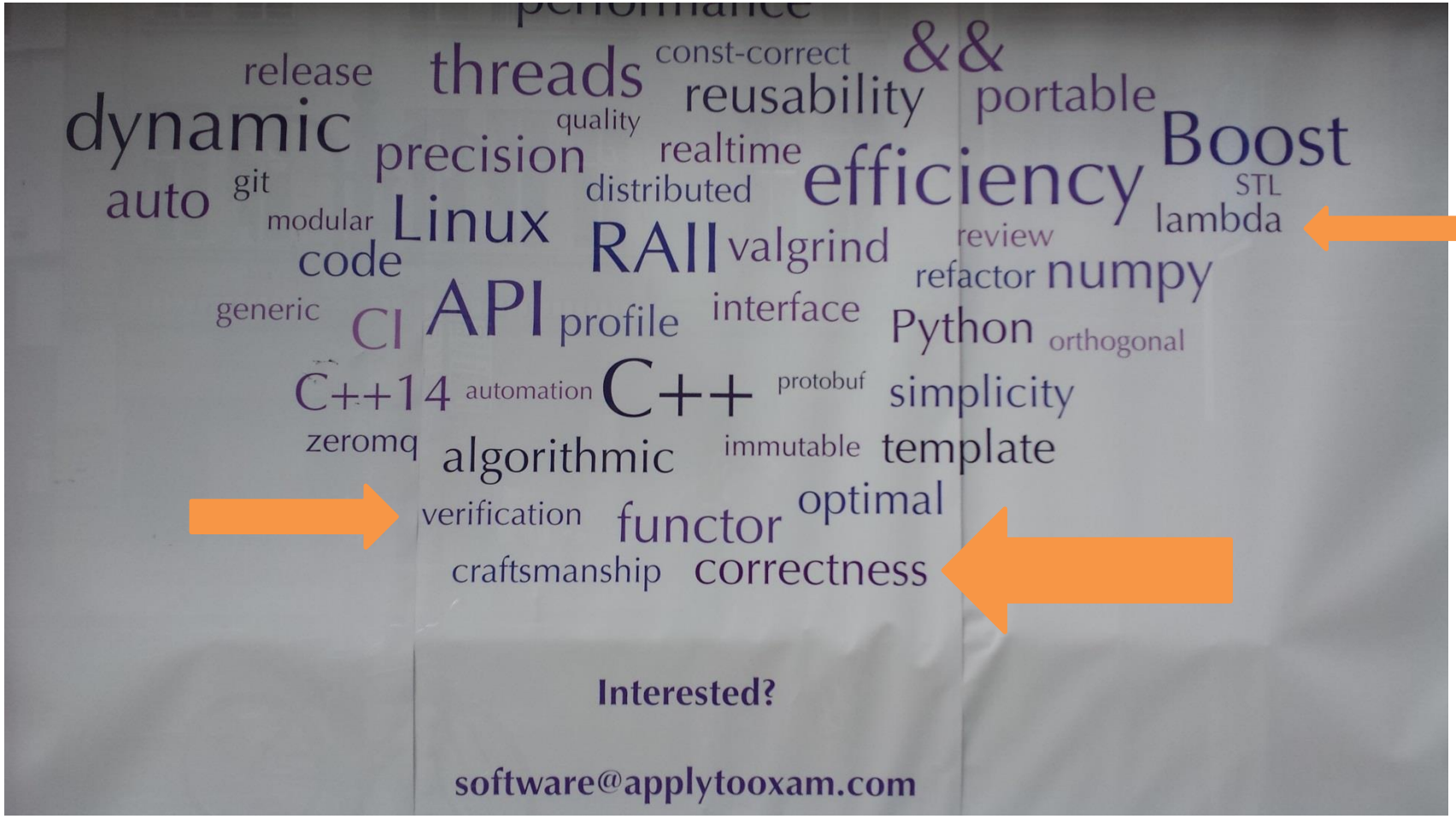
Re-use: Specs and proofs

- Products come in related families and generations
  - Families: server, desktop, mobile and ultramobile parts
  - Generations: Intel® Core™2 Duo Processor, Intel® Core™ i5 Processor
- Robust reusable proofs
  - Allow the cost of verification to be amortized
  - Certify common functionality across generations and families
- Many Intel datapath proofs are descended (with modification) from the Intel Pentium® 4 processor generation
  - "The cost of verifying is less important than the cost of re-verifying"
- An analogous scenario in software:
  - Pick a key component of Linux version N
  - Develop a specification and verify the component against it
  - *Do it again* for version N+1
  - *Do it again* for version N+2
  - *Port* to the equivalent BSD component
- We need to better understand how to reuse proofs and verification results (and validation collateral in general)

Summary

- Datapath verification widely deployed at Intel
- Verification technology necessary, not sufficient
- As important:
  - Specifications
  - Methodology
  - Re-use

Q&A


performance
release
threads
const-correct
&&
dynamic
quality
reusability
portable
precision
realtime
Boost
auto
git
distributed
efficiency
STL
modular
Linux
RAII
valgrind
review
lambda
code
refactor
numpy
generic
CI
API
profile
interface
Python
orthogonal
C++14
automation
C++
protobuf
simplicity
zeromq
algorithmic
immutable
template
verification
functor
optimal
craftsmanship
correctness
Interested?
software@applytooxam.com

Selected References

Technology

1. C.-J. H. Seger, and R. E. Bryant, Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories, Formal Methods in System Design, Vol. 6, No. 2 (March, 1995), pp. 147-190
2. J. Yang and C.-J.H. Seger, "Introduction to Generalized Symbolic Trajectory Evaluation", *IEEE Transactions on VLSI Systems*, vol. 11, no.3 (June 2003), pp. 345-353.
3. J.W. O'Leary, J. Grundy and T.F. Melham, "A Reflective Functional Language for Hardware Design and Theorem Proving", *Fifth Workshop on Designing Correct Circuits,* Barcelona, Spain, March 2004.
4. C.-J.H. Seger, R.B. Jones, J.W. O'Leary, T. Melham, M.D. Aagaard, C. Barrett, and D. Syme, "An Industrially Effective Environment for Formal Hardware Verification", *IEEE Transactions on Computer-Aided Design*, vol. 24, no.9 (September 2005), pp. 1381-1406.
5. J. Grundy, T.F. Melham, and J.W. O'Leary, "A Reflective Functional Language for Hardware Design and Theorem Proving", *Journal of Functional Programming,* vol. 16, no. 2 (March 2006).
6. J. O'Leary, R. Kaivola, and T.F. Melham, "Relational STE and Theorem Proving for Formal Verification of Industrial Circuit Designs", FMCAD'13.

Methodology and applications:

1. J.W. O'Leary, X. Zhao, R. Gerth, and C.-J.H. Seger, "Formally Verifying IEEE Compliance of Floating-Point Hardware", *Intel Technology Journal* (First Quarter, 1999).
2. R. B. Jones, J.W. O'Leary, C.-J. H. Seger, M. D. Aagaard, and T. F. Melham, "Practical Formal Verification in Microprocessor Design", *IEEE Design & Test of Computers*, vol. 18, no. 4 (July/August 2001), pp. 16–25.
3. Roope Kaivola, Katherine R. Kohatsu: Proof engineering in the large: formal verification of Pentium® 4 floating-point divider. STTT 4(3): 323-334 (2003)
4. Roope Kaivola, Naren Narasimhan: Formal Verification of the Pentium® 4 Floating-Point Multiplier. DATE 2002: 20-27.
5. Roope Kaivola: Formal Verification of Pentium® 4 Components with Symbolic Simulation and Inductive Invariants. CAV 2005: 170-184
6. Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, Sudhindra Pandav, Anna Slobodova, Christopher Taylor, Vladimir Frolov, Erik Reeber and Armaghan Naik. Replacing testing with formal verification in Intel Core™ i7 processor execution engine validation. CAV 2009.