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

Span of Products



SoC methodology



Ex: Bay Trail SoC



Bay Trail: User-visible datapaths



Development with shared IPs



Cost of a bug vs time found \$109



Is a \$1B bug possible?

 Mid-1994 – FDIV flaw detected in the Intel Pentium[®] processor



- January 1995 Intel announces a pretax 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

> Checks bounded LTL properties of RTL designs using symbolic simulation



Interactive, deductive reasoning about reFLect programs

For writing formal specifications and verification scripting

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)



- 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: $X \sqsubseteq 0, X \sqsubseteq 1$

Approach



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

 $\vdash \forall a, b. i(bit_add(a, b)) = i(a) + i(b)$

length a = length b

 $x=A \land y=B \Rightarrow N(z) = bit_add(A, B)$ $\equiv \{ property of functions \} \}$ $x=A \land y=B \Rightarrow i(N(z)) = i(bit_add(A, B))$ $\equiv \{ theorem about bit_add \} \}$ $x=A \land y=B \Rightarrow i(N(z)) = i(A) + i(B)$

Deduction ensures correctness and completeness of decomposition

```
i(B) = \sum i(BE[n]) * 2^{kn}
i(N(PP[n])) = i(A) * i(BE[n]), for each n
                                                          STE
i(N^{2}(P)) = \sum i(N(PP[n])) * 2^{kn}
i(N^{3}(R)) = i(N^{2}(P)) + i(C)
                            i(N^{3}(R)) = i(N^{2}(P)) + i(C)
                                       = \sum i(N(PP[n])) * 2^{kn} + i(C)
                                       = \Sigma (i(A) * i(BE[n])) * 2^{kn} + i(C)
                                       = i(A) * \sum i(BE[n]) * 2^{kn} + i(C)
                                       = i(A) * i(B) + i(C)
```

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

IEEE TRANSACTIONS ON COMPUTERS, VOL. C.(8, NO. 8, ADDRET 1986 Graph-Based Algorithms for Boo Manipulation

RANDAL E. BRYANT, MEMBER, 1920

A variety of met and manipulating I

may have a re

operation such

Abstruct—In this paper we present a new data structure for representing Boolean functions and an associated set of manipalation algorithms. Functions are represented by directed, acyclic representations si terion agouttents, reactions are represented by interest, accura-traphs in a manner similar to the representation's introduced by Lee [1] and Akers [2], but with further restrictions on the canonical sum-ofevery function of Let [1] and Asters [2], but with turner relations on the ordering of decision variables in the graph. Although a function or more. More pri requires, in memory carranges in my graph. Antropy a custom requires, in the ward care, a graph of day exponential in the member of arguments, many of the functions encountered in function or any second se at least for man typical applications have a more reasonable representation. Our Example represe algorithms have time complexity proparticent to the sizes of the [4] (or equivalent graphs being operated on, and hence are quite efficient as long as into unate functi trapas near operation on, and dence are quar convent a resp as the graphs do not give too large. We provide experimental result from applying these algorithms to problems in logic design verification that denoisestrate the precisitiy of our appears. several drawba require represent even and odd pa all of these repr

indez Terres-Boolean functions, binary decision diagrams, logic design verification, symbolic manipulation.

an exponential I. INTRODUCTION tations are can

BOOLEAN Algebra forms a cornerstone of computer science and digital system design. Many problems in many differe equivalence of digital logic design and testing, artificial intelligence, and combinatorics can be expressed as a sequence of operations on Due to the Boolean functions. Such applications would benefit from sequence of efficient algorithms for representing and manipulating Bool- erratic behav can functions symbolically. Unfortunately, many of the tasks one would like to perform with Boolean functions, such as to complete testing whether there exists any assignment of input variables such that a given Boolean expression evaluates to 1 (satisfiabilelic graphs. ity), or two Boolean expressions denote the same function (equivalence) require solutions to NP-complete or co NPdiagram no ined by Ak complete problems [3]. Consequently, all known approaches to performing these operations require, in the worst case, an the orderin amount of computer time that grows exponentially with the restriction lating the size of the problem. This makes it difficult to compare the relative efficiencies of different approaches to representing Our re and manipulating Boolean functions. In the worst case, all approache known approaches perform as pourly as the naive approach of commonl representing functions by their truth tables and defining all of tation. Fo the desired operations in terms of their effect on truth table and odd p entries. In practice, by utilizing more clever representations vertices and manipulation algorithms, we can often avoid these argume our alg degrade exponential computations.

Missionering received Nubrember 28, 7864; sevined June 11., 1885. This work was supported in part by the Defense Advanced Research Progens Agency table Orients 2711 and 3971. Barbar Description (Computer Science, Carangie Mellon University, Pathongh, PA, 1512). IEEE Log Support Science, Science, Carangie Mellon University, Pathongh, PA, 1512). single for the mentin function operation 0018-9340/86/0800-0677\$01.00

Formal Hardware Verifica Symbolic Ternary Trajectory

Randal E. Bryant Derek L. Beatty School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213 USA

Abstract

Symbolic trajectory evaluation is a new approach to formal hardware verification combining the circuit modeling capabilities of symbolic logic simulation with some of the analytic methods found in temporal logic model checkers. We have created such an evaluator by extending the symbolic switch-level simulator COSMOS. This program gains added efficiency by exploiting the ability of COSMOS to evaluate circuit operation over a ternary logic model, where the third value X represents an unknown logic value. This program can formally verify systems containing complex features such as switch-level models, detailed timing, and pipelining.

1. Introduction

Formal verification seeks to overcome the weakness of informal design testing by simulation. Using our verifier, one can prove that a switch-level model of the transistor circuit correctly implements a formal description of the desired behavior for all possible system operations. Formal verification becomes increasingly desirable as system designs become more complex. With the introduction of pipelining and concurrentlyoperating subsystems, it becomes increasingly difficult using informal methods to evaluate the many subtle interactions between logically unrelated system activities.

In this paper we describe a new approach to formal verification that augments the circuit modeling capabilities of symbolic, switch-level simulation with some of the analytic capabilities found in temporal logic model checkers. With this increased analytic capability, we can express such temporal aspects of circuit behavior as clocking methodology and the skewing in time caused by pipelining. This paper describes the operation and application of our evaluator using a number of circuit design examples. A detailed presentation of the formal logic is presented in [2].

*This research was supported by the Defense Advanced Research Project Agency, ARPA Order Namber 4976, and by the National Science Foundation under grant number MIP-8913667.

Permission to copy without fee all or part of this material is gran provided that the copies are not made or distributed for direct commerci advantage, the ACM copyright notice and the title of the publication a its date appear, and notice is given that copying is by permission of i Association for Computing Machinery To copy otherwise, or to republic requires a fee and/or specific permission. 28th ACM/IEEE Desi

P 1991 ACM 0-89791-395-7/91/0006/0397 \$1.50



24

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



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



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



Selected References

<u>Technology</u>

- 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
- 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.