Moore’s Law v. Verification Complexity

Jason Baumgartner

Thanks for contributions: Wolfgang Roesner, Ralf Fischer

IBM Corporation
Outline

- What is Moore’s Law?
- Verification Complexity
- Does Moore’s Law hurt Verification Complexity?
  - No
  - Yes
- Can we cope with this complexity?
  - Yes
  - No
- Open problems: despair!!! vs hope???
What is Moore’s Law?

# transistors per IC for minimum cost has increased at roughly a factor of two per year

there is no reason to believe it will not remain nearly constant for at least 10 years

What *Isn’t* Moore’s Law?

- Attributed to virtually all exponentially-growing computing metrics
  - Circuit speed
  - Computing power (MIPS, GFlops, …)
  - Storage capacity
  - Network capacity
  - Pixel density
  - ...

- Strictly speaking, these are *not* part of Moore's original observation
  - IC complexity for *minimum cost*
What *Isn’t* Moore’s Law?

- Nonetheless, all refer abstractly to the same trend
  - Device miniaturization
  - Reliable integration “growth”
  - “Circuit and device cleverness”

- We deliberately abuse notation w.r.t. *“Moore's Law”* in this talk
Design Characteristics of “Moore’s Law”

- Smaller: *miniaturization*
  - Devices and transistors

- Cheaper
  - Per *transistor*
  - Not necessarily per *product*

- Faster
  - If software developers are willing :-)

![Graph showing trends in technology nodes (90nm, 65nm, 45nm, 32nm)]
Design Characteristics of Moore’s Law

- Bigger!
  - Chips, wafers, ICs, networked systems, ...

- More complex!!
  - More “smaller” devices crammed onto chip / IC
  - Scale-up of functionality: data width, memory size, ...

- Hotter!!!
  - Since bigger & faster!
Design Characteristics of Moore’s Law

- Comparable functionality in smaller / cheaper package?

- No! Cram *more* into a bigger package

- Harder to verify!!! ??? !?$%*#@!!
  - Thankfully, “device complexity” cannot *afford* to be “as great as possible”
Longevity: for at least 10 years, indeed!

Functionally correct transistors / IC?
The End is Near! (Is it?)

- Moore himself was one of his harshest critics
  - Disappearing “circuit cleverness” 1975
  - Lack of demand for VLSI 1979
    - The Death of Cost Effectiveness

- No exponential is forever - must hit limitations of physics?
  - Can we miniaturize (and design w.r.t.) quantum particles? Hmmm…

- Note trends on massive parallelization (e.g. BlueGene), 3D chips, biological computing, quantum computing, ...
  - Who knows?
  - Will global warming (or out-of-control particle accelerator) finitize “forever”?
What is “Hardware Verification”?

- Chips often designed in Hardware Description Language (HDL)
- HDL taken through compile, physical design steps to fabrication

- Many facets to verifying a chip
- *Logic verification* is the primary focus of this talk
- *Does the HDL of the chip produce correct computations?*
  - E.g., the FPU generate IEEE-compliant results?

- Will later touch upon correctness of post-HDL flow
What is “Hardware Verification”?

- May often represent verif problem using a *sequential netlist*
  - Correctness properties may be synthesized into simple assertion checks

- Netlist composed of primary inputs, combinational gates, sequential elements (latches, RAM, …)

- *Sequential* refers to ability to “remember” past computations
Verification Complexity

- A *state* is a valuation to the sequential elements of the design.

- Exhaustive (formal) verification generally requires analysis of *reachable* states.
Verification Complexity

Formal verification generally requires analysis of *reachable states*

- Falsification may only require exploring a *subset*

Some proof techniques leverage fast analysis of a *superset*
  - Induction: can design transition from a good state to a bad state?
Verification Complexity

- **Explicit** state enumeration is intractable
  - **Symbolic** analysis often superior, though still capacity-gated
Moore’s Law v. Verification Complexity

- # Components per IC doubles every ~2 years

- Verification thus *appears* to grow exponentially more complex
  - Compounded by use of *today’s* computers to verify *tomorrow’s* designs

- Is this *necessarily* the case?

- Let us revisit how this capacity tends to be used
  - Moore's *Heirlooms*
Moore's Heirlooms: Integration

- Integration of more devices on chip
  - System on a Chip: more components+functionality moved on-chip
  - Caches are moving on-chip

- Lowers packaging costs and power, increases speed
- “Moving” components: no negative impact to verif complexity
Moore's Heirlooms: *Modularity*

- Additional execution units
  - Multiple FPUs, FXUs, LSUs, ...

- Additional cores
  - POWER4 is 2 core; POWER7 is 8 core

- No additional *component* verif complexity

- Overall *system* complexity may increase
  - Hardware, software, or both
  - More concurrency, # interfaces
  - Some aspects may be covered by higher-level verification
Moore's Heirlooms: Specialized Hardware

- SW function moves to hardware
  - Vector units, encryption

- Diversified modularity
  - Cell processor: 8 Synergistic Processing Elements in addition to a Power processor

- May not increase verif complexity
  - “Only” more components to verify

- Though nonetheless difficult to verify!
  - Move verification burden from SW to HW?
Moore's Heirlooms: *Increased Operand / Data Width*

- Operand width has grown substantially
  - Mainstream (vs mainframe!) processors

- Many processors have emulated 128-bit data support for decades
  - SW + specialized HW atomically manages narrower computations
Moore's Heirlooms: *Increased Operand / Data Width*

- Does increased data width increase verification complexity?
  - Sometimes “no” !!!

- Data routing checks are not necessarily more complex
  - Some checks may be **bit-sliced**; *linear* verification scaling
  - **Word / vector** reasoning techniques scale well *when applicable*
    - UCLID, SMT, uninterpreted functions
  - Verification **reduction techniques** have been proposed to automatically shrink widths to facilitate a broader set of algorithms
    - Control / token nets, Bjesse CAV’08
Moore's Heirlooms: *Increased Operand / Data Width*

- Does increased data width increase verification complexity?
  - Sometimes “yes” !!!

- What about correctness of *computations* on the operands?
  - *Optimized* arithmetic / logical computations are not simple + = * / < >

- Consider IEEE Floating Point Spec

<table>
<thead>
<tr>
<th></th>
<th>Single Precision</th>
<th>Double Precision</th>
<th>Quadruple Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Width</td>
<td>32</td>
<td>64</td>
<td>128</td>
</tr>
<tr>
<td>Exponent bits</td>
<td>8</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Significand bits</td>
<td>23</td>
<td>52</td>
<td>112</td>
</tr>
</tbody>
</table>
Floating-Point Verification

- Floating point number format: $S \times B^E$
  - S: Significand, e.g. 3.14159
  - B: Base, here B=2
  - E: Exponent, represented relative to predefined bias
    - Actual exponent value = bias + E

- A normalized FP number has Mantissa of form 1.?????
  - Aside from zero representation

- Fused multiply-add op: $A \times B + C$ for floating point numbers A,B,C
  - C referred to as addend
  - $A \times B$ referred to as product
Floating-Point Verification: Double Precision

- Optimized FPU: 15,000 lines HDL
- IEEE-compliant reference model for double precision: 500 lines HDL

\[ \begin{align*}
2^{e_{\text{addend}}} & \quad 2^{e_{\text{prod}}} & \quad \text{164 bit} \\
\text{significand addend} & \quad \ldots00 \\
+/− & \quad \text{significand product} & \quad \ldots00 \\
\text{cnt leading 0's} & \quad \text{copy and round} \quad \text{a+b} & \quad \text{Leading zero's may occur, e.g.,} \\
\text{S Exp} & \quad \text{Frac} & \quad 1.101011 \\
\text{− 1.101001} & \quad = 0.000010 \\
\end{align*} \]
Floating-Point Verification: Double Precision

- Direct equiv check between reference, implementation is computationally intractable
  - Use *case-splitting* strategy!

- Four distinct categories of case splits used
  - Based on difference between product, addend exponent
(a) Far out left: the addend is much larger than the product; there is no overlap, the product becomes a sticky bit.

(b) Overlap left: the addend is larger than the product; the product overlaps with the right part of the addend.

(c) Overlap right: the product is larger than the addend; the addend overlaps with the right part of the product.

(d) Far out right: the product is much larger than the addend; the addend becomes a sticky bit.
Floating-Point Verification: Double Precision

- Case split to fix product exponent
  \[ C_\delta := (e_a + e_b - \text{bias} = e_c + \delta) \]
  \[ \delta = e_{\text{prod}} - e_c \] where \( e_{\text{prod}} = (e_a + e_b - \text{bias}) \) is the product exponent 
  and \( e_c \) is the addend exponent

- Normalization shifter is used to yield a normal result
  - Depends upon # number of leading zeros of intermediate result

  \[ C_{\text{sha}} := (\text{sha} = X) \] for all 106 possible shift amounts;
  \[ C_{\text{sha/rest}} := (\text{sha} > 106) \] to cover the remaining cases (trivially discharged)

- Define a secondary case-split on normalization shift
  - Constraint defined directly on shift-amount signal (sha) of Ref-FPU
  - Sha is 7-bit signal (double-precision) to cover all possible shift amounts
Floating-Point Verif: Double to Quad Precision

- **Double precision**
  - ~585 total cases to check
  - Each tractable using BDDs

- **Quad precision**
  - ~1244 total cases
  - *None* are practical using BDDs

- Denormal operands require additional case split on input normalization shifter
  - # Cases increases three orders of magnitudes double-to-quad
Moore's Heirlooms: *Increased Operand / Data Width*

- Error Code Detection / Correction (ECC) logic becomes substantially more complex w.r.t. data width
  - Byproduct of transistor miniaturization: *soft errors!*
  - Increasingly mandate ECC logic
  - Along with increasingly elaborate ECC algos to handle *more* error bits

- Emerging encryption HW similarly explodes in complexity w.r.t. data width

- Overall: *Does* increased data width increase verif complexity?
  - Sometimes dramatically !?$%*#@!!
Moore's Heirlooms: *Increased RAM Depth*

- Often *not* a substantial cause of verification complexity
  - Most of the design is insensitive to this metric

- Verification algorithms can often treat such arrays more abstractly with *memory consistency constraints*
  - Efficient Memory Model, BAT ICCAD’07, Bjesse FMCAD’08

- Though with larger caches and *more elaborate associativity schemes* comes increased complexity
  - Sometimes the logic *adjacent to* memory array becomes more complex
Moore's Heirlooms: *Circuit and Device Cleverness*

- Countless tricks behind increasing MIPS and computing power
  - Some of these are HUGE causes of verification complexity

- First consider techniques for circuit speed
  - Integration, interconnect speedup, miniaturization, datapath widening all eliminate speed barriers
  - Natural push to *speed up* core processing circuitry
  - How is this achieved?
Moore's Heirlooms: *Circuit Speed*

- Decades of synthesis research to reduce logic area, delay, ...
  - E.g., simple redundancy removal and rewriting
Moore's Heirlooms: *Circuit Speed*

- Decades of synthesis research to reduce logic area, delay, ...
  - E.g., simple redundancy removal and rewriting
Decades of synthesis research to reduce logic area, delay, ...

- E.g., simple redundancy removal and rewriting
- Max clock speed ~ max combinational delay between state elements
Moore's Heirlooms: *Circuit Speed*

- Decades of synthesis research to reduce logic area, delay, ...
  - Redundancy *introduction* may also speed up circuit
Moore's Heirlooms: *Circuit Speed*

- Retiming may be used to enable higher circuit speed
  - Max clock speed ~ max combinational delay between state elements
Moore's Heirlooms: *Circuit Speed*

- Pipelining (+retiming) may be used to break lengthy computations
Moore's Heirlooms: *Circuit Speed*

- Multi-phase latching/clocking better distributes propagation delays
  - Convert edge-sensitive to level-sensitive state elements; retiming
Moore's Heirlooms: \textit{Circuit Speed}

- Do these techniques \textit{necessarily} hurt verification complexity?
  - Some may, others not

- Luckily, many are \textit{reversible} by verification-helping transforms
Moore's Heirlooms: *Circuit Speed*

- Logic minimization techniques often *help* verification
  - Though *word-level techniques* may suffer
  - Optimizations often done at *bit-level*

- Indeed, often desirable to *leverage* such minimization techniques explicitly *to enhance* verification
  - Minimize suboptimally synthesized designs for enhanced verification
  - Exploit optimization potential created by testbench
  - A powerful synergy between synthesis and verification
Moore's Heirlooms: *Circuit Speed*

- Redundancy *introduction* often hurts verification
  - More state elements with high correlation, more logic to reason about
    - Naive BDD, SAT algorithms may become highly inefficient
    - Induction becomes less effective with correlated state element bloat
- May be reversed through redundancy *removal*
Moore's Heirlooms: *Circuit Speed*

- *Min-period* retiming often hurts verification
  - Often increases state element count, correlation
  - Similar complexities as redundancy introduction

- May be reversed through *min-area* retiming
Moore's Heirlooms: *Circuit Speed*

- Multi-phase latching often hurts verification
  - Increase in state element count, correlation; increase in *diameter*
- May be reversed through *phase abstraction*
  - Unfold next-state functions modulo 2

![Diagram of circuit elements](image)
Moore's Heirlooms: *Circuit Speed*

- Pipelining often hurts verification
  - Increase in state element count; increase in diameter
- May be reversed through peripheral retiming
  - And state-folding abstraction (like phase abstraction)
Moore's Heirlooms: *Circuit Speed*

- Overall message: correct-by-construction synthesis transforms can often be reversed to avoid significant verification penalty
  - Will be revisited later in context of “equivalence checking”

- Such transforms often *help* verification in themselves
  - ABC (UC Berkeley) won the 2008 HWMCC largely due to such transforms
  - SixthSense (IBM) uses such a transformation-based verification paradigm
Moore's Heirlooms: *Circuit Speed*

- Can *all* circuit speedups be reversed to avoid verif complexity?
  - Unfortunately (and obviously), no!

- E.g., pipelining is often accompanied by *bypass* / *stall* circuitry
  - Only “correct” under specific assumptions / handshaking with adjacent logic

- Entails significant design complexity
  - Cannot generically be reversed without expensive *global* reasoning
Moore's Heirlooms: *Design Cleverness*

- Many techniques critical to preserving Moore’s Law momentum
  - Yet very complex for verification
  - Difficult to automate or generically “reverse”

- Superscalar + out-of-order execution
- Prefetching
- Speculative execution
- Reconfigurable hardware
- *Holistic design*
  - The simultaneous optimization of:
    - materials, circuits, cores, chips, system architecture, software, …
Moore's Heirlooms: *Design Cleverness*

*Design innovation has become key to performance gains*
Must the Verif Engineer suffer Moore's Prodigy?

- Discussed circuit / design tricks which keep Moore's Law moving
  - Impact on verification
  - Ways to attempt to cope

- For HDL verification, why are (some of) these even pertinent??
  - Does the HDL need to reflect circuit optimizations?

- Let us briefly discuss path from HDL to fabrication
From HDL to Fabrication

HDL Design
valout <= (A and NOT B) or (NOT A and B);

Layout matches schematic

"trusted"

Gate-level netlist

Layout

Transistor-level schematic

equiv checking

LVS
Must the Verif Engineer suffer Moore's Prodigy?

- Logic synthesis includes numerous optimizations to improve schematic quality
  - Technology-independent optimizations
  - Technology-dependent optimizations
  - Technology mapping

- Why not use a higher-level HDL as basis of verification?
  - Leave the ugly circuit optimizations to synthesis and equiv checking!!!
Use Higher-level HDL Basis for Verification?

+ Easier to design
+ Easier to verify
  - Simpler; amenable to word-level techniques
+ Likely to be less buggy
  - Bugs / lines of HDL roughly constant across design generations

- Any drawbacks??
Combinational Equivalence Checking (CEC)

- Very well-established technology
  - ~Every chip fabricated today leverages this technology

- Requires 1:1 state elt mapping between Pre / Post Synthesis models
  - Does not truly analyze sequential behavior of the design
  - Validate next-state functions, outputs with respect to cutpoints
Combinational Equivalence Checking (CEC)

- Combinational optimizations easily pushed into synthesis flow
  - No need to hand-tune HDL for combinational optimization
  - (As long as synthesis is powerful enough)

- Though not applicable for sequential optimizations
  - Retiming, pipelining, multi-phase latching, clock gating, sequential redundancy elimination/introduction, ...

  - Many CEC tools have (very) limited support for some of these
Sequential Equivalence Checking (SEC)

- Is there a technology for equiv checking *sequential* optimizations?
  - Of course! SEC has been proposed 16 years ago
  - Only beginning to see industrial application

- Analyzes true sequential I/O behavior
  - Supports *arbitrary functionality-preserving optimizations*

- With this generality comes computational complexity
Sequential Equivalence Checking (SEC)

- Can we push sequential optimizations into synthesis flow?

- Due to SEC, *in theory* the answer is “yes”

- If synthesis is *powerful enough*, in practice the answer is “yes”
  - (Or is it?)
Sequential Equivalence Checking (SEC)

- Can we push sequential optimizations into synthesis flow?
  - *Optimal synthesis is as computationally expensive as verification*
    - Holistic design: synthesis alone is not yet *nearly* sophisticated enough
    - Many global optimizations require pseudo-manual “leap of faith” to perform at component level, clever methodology to validate
  - SEC is also computationally expensive!
    - Sequential reasoning over *two* designs
  - “Synthesis history”-aware approaches are promising to flatten this concern
    - Brayton’s FMCAD’07 keynote, Brayton/Mishchenko FMCAD’08

- *Post-silicon analysis is an open challenge*
Post-Silicon Analysis

- Various *pervasive logic* in chip aside from core functionality

- E.g., scan chains enable reading the contents of a chip

- Critical to debug silicon failures
  - Also used for purposes such as initialization, self-test, ...
Post-Silicon Analysis

- If synthesis moves the state elements, what exactly will be scanned?
  - Sequential optimizations must be done before scan insertion (else ineffective)
  - Need to remap scanned values to state elements familiar to the designer

- Can mapping be done through sim + SAT using “synthesis history”?
  - May entail additional constraints on synthesis; de Paula FMCAD’08, Hunt

- What about trace/debug buses, which monitor the chip real-time?
  - Cannot algorithmically post-process that trace!

- What about Engineering Change Orders?
  - Last-minute changes made directly to post-synthesis design
Post-Silicon Analysis

- If we can reconcile pervasive design w.r.t. sequential synthesis

- We still need to verify the pervasive logic
  
  1) Verify that primary functionality is unaltered through this injection
     - Easy SEC-style task, constraining to functional analysis
     - Though now synthesis does conditionally alter behavior!

  2) Need to verify that pervasive logic works properly!
     - Often more critical than main functionality; chip rendered useless otherwise

- What if not correct (or inadequate): how debugged?
  
  - Does the designer need to learn post-synthesis design ??
  - Somewhat defeats the purpose !!!
Coping with Moore's Heirlooms

- How can verification cope with increased design complexity?
  - A good methodology is often key overcoming technology shortcomings

1) Abstraction?

- Maybe, though intrinsic complexity need to be modeled+verified somewhere
  - Leaky abstractions have their limits!
Coping with Moore's Heirlooms

- *Leaky abstractions* have their limits

- Push for *holistic design* makes these limits fundamental

- May seem crazy, though a cost-effective path to high-end systems
  - Suppressing design detail too deeply risks catastrophe
    - Failure to meet timing / performance / power / reliability goals
    - *Lost time to market*
    - *Product failure*
BlueGene/L System

**System**
64 Racks, 64x32x32

**Rack**
32 Node Cards

**Node Card**
(32 chips 4x4x2)
16 compute, 0-2 IO cards

**Compute Card**
2 chips, 1x2x1

**Chip**
2 processors

- **2.8/5.6 GF/s**
- **5.6/11.2 GF/s**
- **90/180 GF/s**
- **180/360 TF/s**

- **16 GB**
- **512 GB**
- **32 GB**
- **32 TB**

- **2.8/5.6 TF/s**
- **5.6/11.2 TF/s**
- **90/180 TF/s**

- **1.0 GB**
- **16 GB**
- **32 GB**
- **32 TB**

**Moore’s Heirlooms:**

**Circuit Speed**

- BlueGene/L System
  - 2.8/5.6 GF/s
  - 4 MB
  - 2 processors
  - 2 chips, 1x2x1

- 5.6/11.2 GF/s
  - 1.0 GB
  - 32 chips 4x4x2
  - 16 compute, 0-2 IO cards

- 90/180 GF/s
  - 16 GB
  - 32 Node Cards

- 180/360 TF/s
  - 32 TB
  - 64 Racks, 64x32x32

Coping with Moore's Heirlooms

- *Leaky abstractions* have their limits
- Though abstraction clearly has a substantial role in verification
  - May enable a *hierarchical verification process*
  - *Protocol* verification $\rightarrow$ *HDL* proof obligations
  - *Unoptimized design* verification $\rightarrow$ *optimized design* proof obligations

- Sometimes requires *decomposition* of verification process
  - E.g. functionality vs pervasive vs timing vs power
  - Much harder to decompose *design* process in this manner, unfortunately

- Manually maintaining abstractions is *expensive*
  - Higher-level HDL, automated optimizations are “free abstractions”
POWER5 Chip

- Dual pSeries CPU
- SMT core (2 virtual procs/core)
- 64 bit PowerPC
- 276 million transistors
- 8-way superscalar
- Split L1 Cache (64k I & 32k D) per core
- 1.92MB shared L2 Cache >2.0 GHz
- Size: 389 sq mm
- 2313 signal I/Os

- >>1,000,000 Lines HDL
  - How big would its abstractions be?

POWER architecture:
- Symmetric multithreading
- Out-of-order dispatch and execution
- Various address translation modes
- Virtualization support
- Weakly ordered memory coherency
Coping with Moore's Heirlooms

- How can verification attempt to cope with increased design complexity?

2) Compositional reasoning

- Isolate complex logic as precisely as possible for dedicated verification

- When verifying adjacent logic (or higher-level model), abstractly model these complex components

- *Assume-guarantee reasoning* is sound and complete
  - Substantial work in automating this process
  - Still largely manual for realistic designs
Coping with Moore's Heirlooms

- Compositional verification is often manually intensive
  - Unless design is well-partitioned with well-defined interfaces

- For highest performance, *global optimization* is often performed
  - Subtle signal / timing relations are exploited between blocks
  - Intricate functionality decentralized

- Optimization to the point that designs *almost don't work properly*

- Overall: Even clever methodologies become cost-ineffective to cope with technology shortcomings
  - In cases need to resort to incomplete techniques; risk verification gaps
Example: *Power Saving Logic*

- Higher speed, increased density → need for power reduction!

- Karen already spoke about power-gating logic
  - Reduce power consumption by disabling voltage to *inactive* components

- Methodologically can cope with this complexity
  1) (Compositionally) verify that it does not meaningfully alter functionality
  2) Disable power-gating logic for ease of functional verification
     - A simple “abstraction”
  3) Also need to verify that it works as intended!

- *Decompose and abstract*
Example: *Power Saving Logic*

- Asynchronous issues arise due to clock domain decoupling
  - Even if running at identical frequencies

- *Metastability* may propagate if not properly guarded against

- Communication channels may suffer sequential delays

- Synchronization logic used to protect design from these dangers
  - Another set of verif tasks to ensure that this logic works properly

- “Abstract design” attained by not injecting metastability, delays
  - Though still includes the complexity of the synchronization logic
  - Manual abstraction / higher-level design may be needed
Conclusion

- Does Moore’s Law hurt the verification community?
  - Yes!
  - Though not always!

- Many open problems in design + verification
  - *HW verification is not a solved problem*

- *Old* open problems:
  - Improve bit-level verification algorithms!
  - Improve bit-level synthesis algorithms!
  - Improve equivalence checking techniques!
Conclusion

- **Newer open problems**
  - Improve higher-level verification algorithms !!
  - Improve high-level synthesis techniques !!
  - Improve higher-level equivalence checking techniques !!

- **Newest open problems**
  - Improve design+verification methodologies w/ focus on holistic design !!!
  - Address post-Si issues through sequential synthesis !!!

- “Someday Moore’s Law will work *for*, not *against*, the verification community” Allen Emerson
  - Requires substantial innovation! *Help achieve this goal !!!!