Verification as if your life depends on it

Rob Gerth

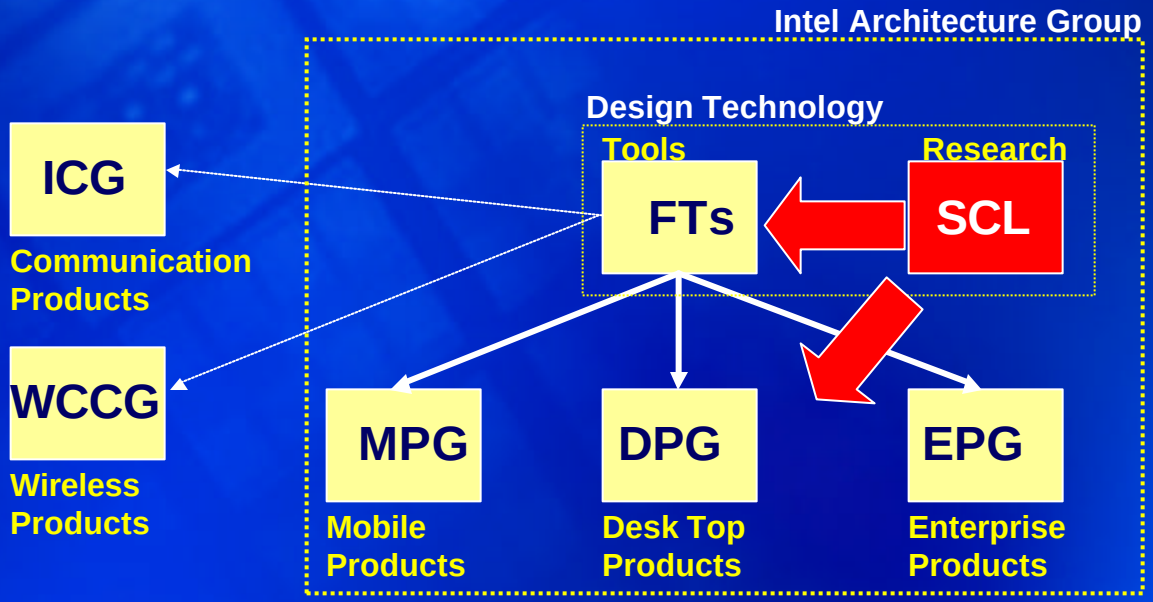
May 19, 2001

Where it all started (for Intel)

- In 1994, Intel took a $475M charge against revenues to cover replacement costs and inventory writedown due to the Pentium® FDIV problem
  - Lousy crisis management
  - But, post-silicon errata are extremely expensive
  - Nothing wakes up higher management better…
    - Strategic CAD Labs (SCL) founded as a result

Strategic CAD Labs (SCL)

http://intel.com/research/scl



Invent design technologies that will be critical to Intel

Horizon: 3+ years

33 researchers; 6 opens

Disaster

- In 1994, Intel took a $475M charge against revenues to cover replacement costs and inventory writedown due to the Pentium® FDIV problem
  - Lousy crisis management
  - But, post-silicon errata are extremely expensive
  - Also, nothing wakes up higher management better…
    - Strategic CAD Labs (SCL) was founded as a result

Seven years later

- Full formal verification against IEEE 754 floating-point standard
  - verify same gate-level RTL all traditional dynamic validation is performed on
  - mostly routine
  - easily within development timeframe
  - excellent ROI thru easy porting; even across micro-architectures
- Good reasons to be confident same can be achieved for
  - control- and memory-heavy designs
  - off-chip protocols/chipsets: Scalability Port, USB

intel

SCL
Strategic CAD Labs
Leading the way in CAD

Not so fast…

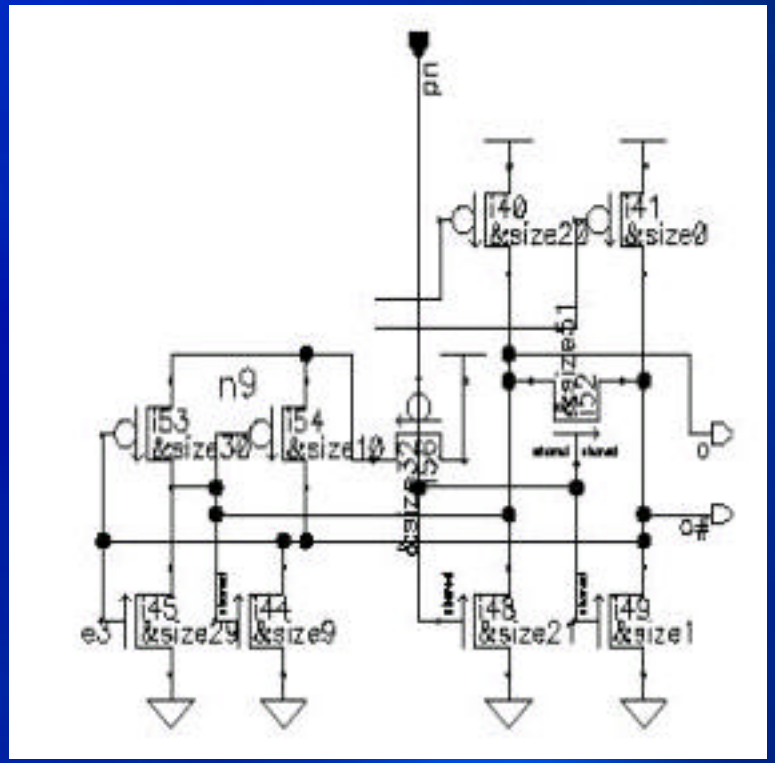
Transistor level problems

- Cannot be modeled on gate-level

Because *o* and *o#* are held to 0 at reset, it takes 2 unit delays for the *o* and *o#* to stabilize …

- what if the input data was available for only 1 unit delay?

- Still on discrete level!



- Formal equivalence verification issue

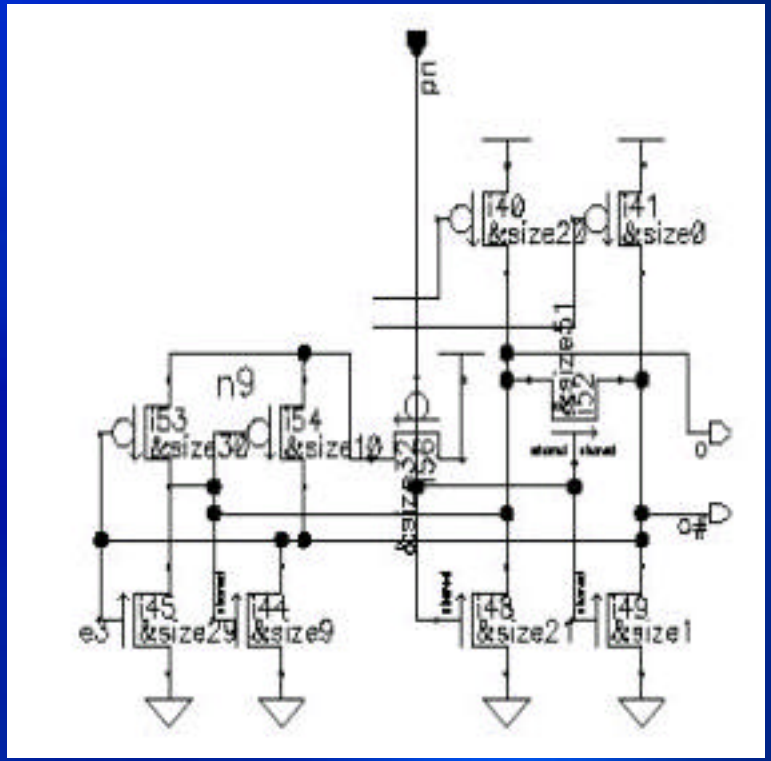
‘Analog’ errors

- speed-paths

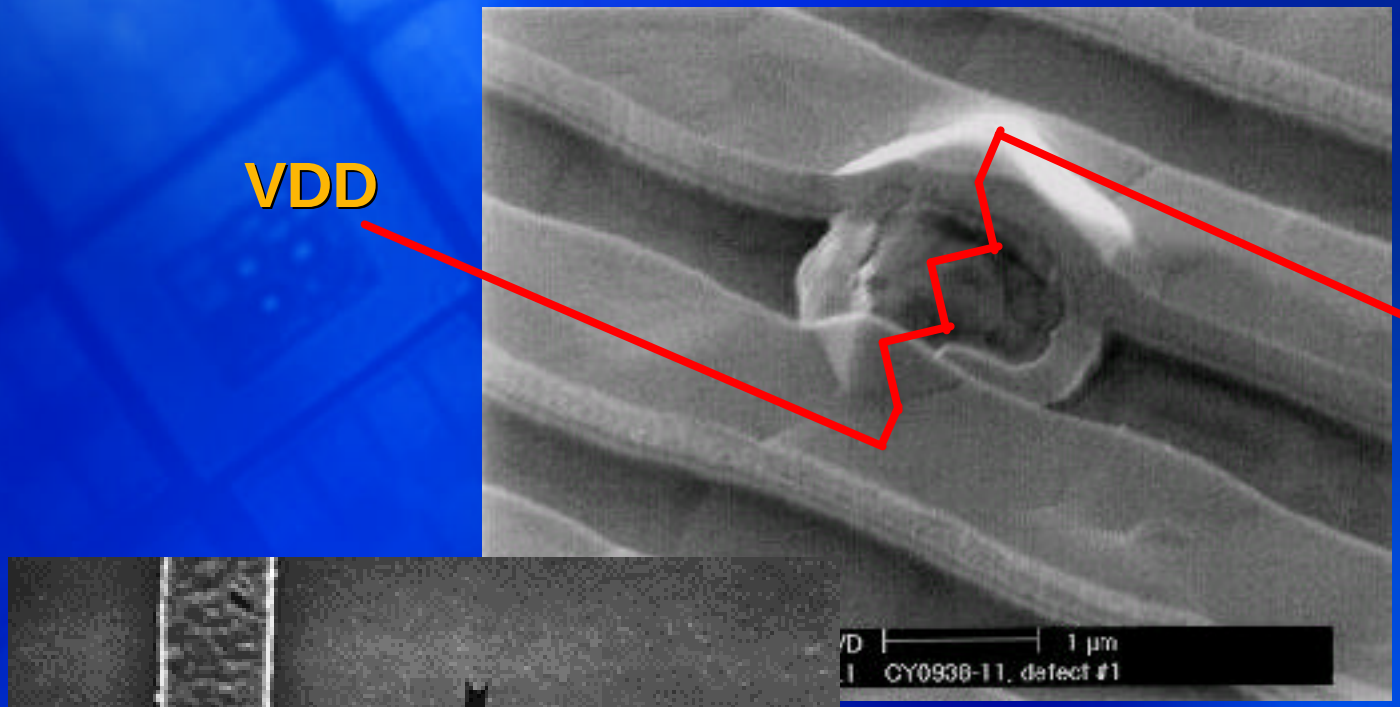
there is a scenario that

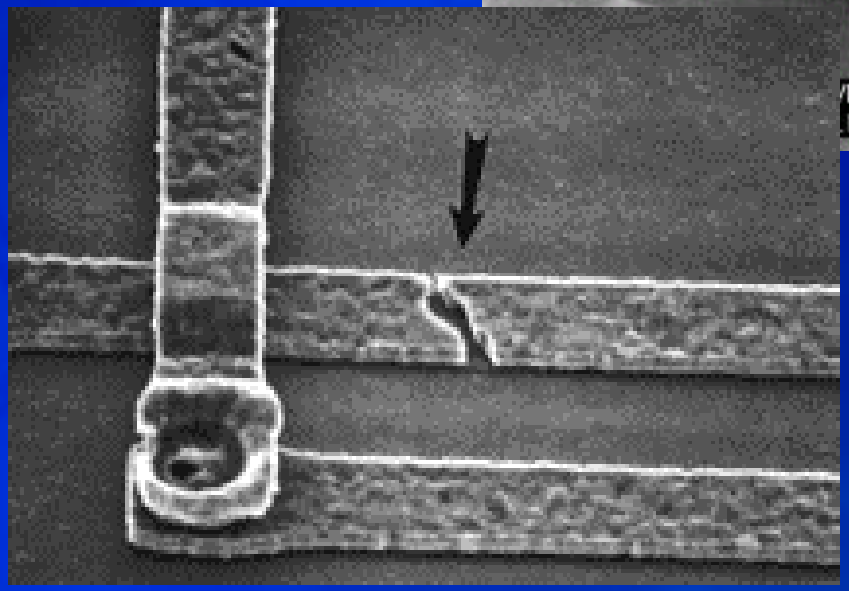
leads to a signal switching

2 pico seconds too late



Physical design issues





Functional correctness only one of many design worries

- Silicon errata extremely costly
  - highly developed classical validation technology which will find many bugs
- FV does not automatically acquires a place

Positioning FV

- Multiple overlapping ?-processor projects
  - any time RTL changes, FV must be redone
  - formal verification must be done within the projects
  - FV requires more expertise than validation
  - separate verification teams
- Charged against project budget

Positioning FV ctd

- Expectation management
  - Initially: obtain *blessed* RTL
    - no amount of validation uncovers additionals bugs
    - FV of no interest if it applies to 'sanitized' RTL
  - Recently: also hunting for high-quality bugs
    - Jury still out

SCL's FV technology

STE symposium (CAV 2000)

http://intel.com/research/scl/stesympsite.htm

Requirements (Whig's history)

- Complete specifications usually not available
- Access to design engineers is always limited
  - Need exploration/debugging capability
  - Need versatile ways to build circuit APIs
- FV is expensive
  - Optimize common case: proof failure/debugging not success
  - Use API to isolate effort from RTL changes
  - Reuse/maintain verification artifacts
  - Amortize cost over changes and/or multiple designs
  - Must be sound
  - Clear what has (not) been proven

Forte verification system

- Functional Language: FL
  - What?
    - General purpose interpreted programming language
    - Everything is a function, i.e., no side effects
    - Lazy semantics (can deal with infinite objects).
  - Why?
    - Very high level (10 to 100 times smaller than equivalent C program)
    - Convenient to program APIs or FV scripts in
    - Easy to reason about (precise & simple semantics)
    - The preferred language type for writing theorem provers

Functional Language: FL ctd

- Main features:
  - **Strongly typed, i.e., safe**
  - Polymorphic type inference, i.e., convenient
  - Pattern matching built in, i.e., readable
  - Abstract data types, i.e., extremely safe
  - **BDDs first class objects**, i.e., easy to write symbolic algorithms
  - Garbage collection is built in, i.e., safe and convenient
  - Dynamic BDD variable reordering built in, i.e., convenient
  - **Efficient (basic) model checker available: STE**

Example non-traditional fl code

```
: let av = variable “a”;
a::bool

: let bv = variable “b”;
b::bool

: (av XOR bv) = av;
it::bool
b’

: (av XOR bv) == av;
it::bool
F

: !x.?y. ((x AND y) OR NOT (x XOR y));
it::bool
T
```

intel

SCL
Strategic CAD Labs
Leading the way in CAD

Example more complex fl code

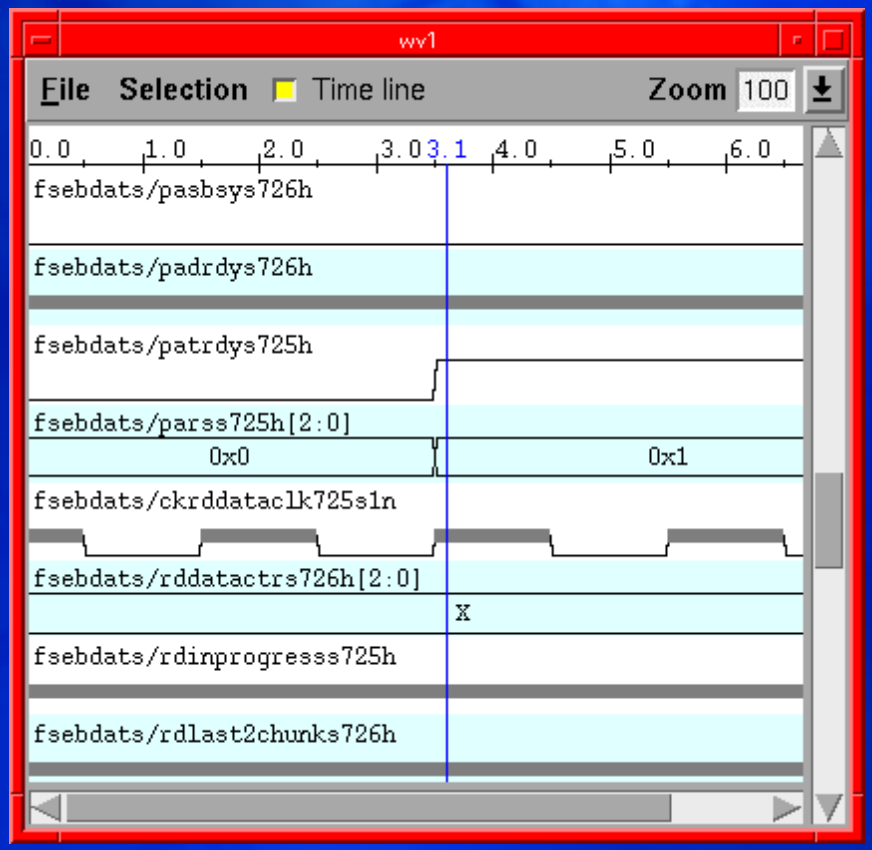
```
let cAX (Model R s s') set =
   let set' = bdd_cur_to_next set in
   quant_forall s' (R ==> set');

let cAG (Model R s s') set =
   letrec AGR cur =
      let new =
         let cur' = bdd_cur_to_next cur in
         quant_forall s' (set AND (R ==> cur')) in
      if new == cur then cur else AGR new in
   AGR T;
```

~50% of the code for a (simple) symbolic CTL model checker!

Model checking in fl

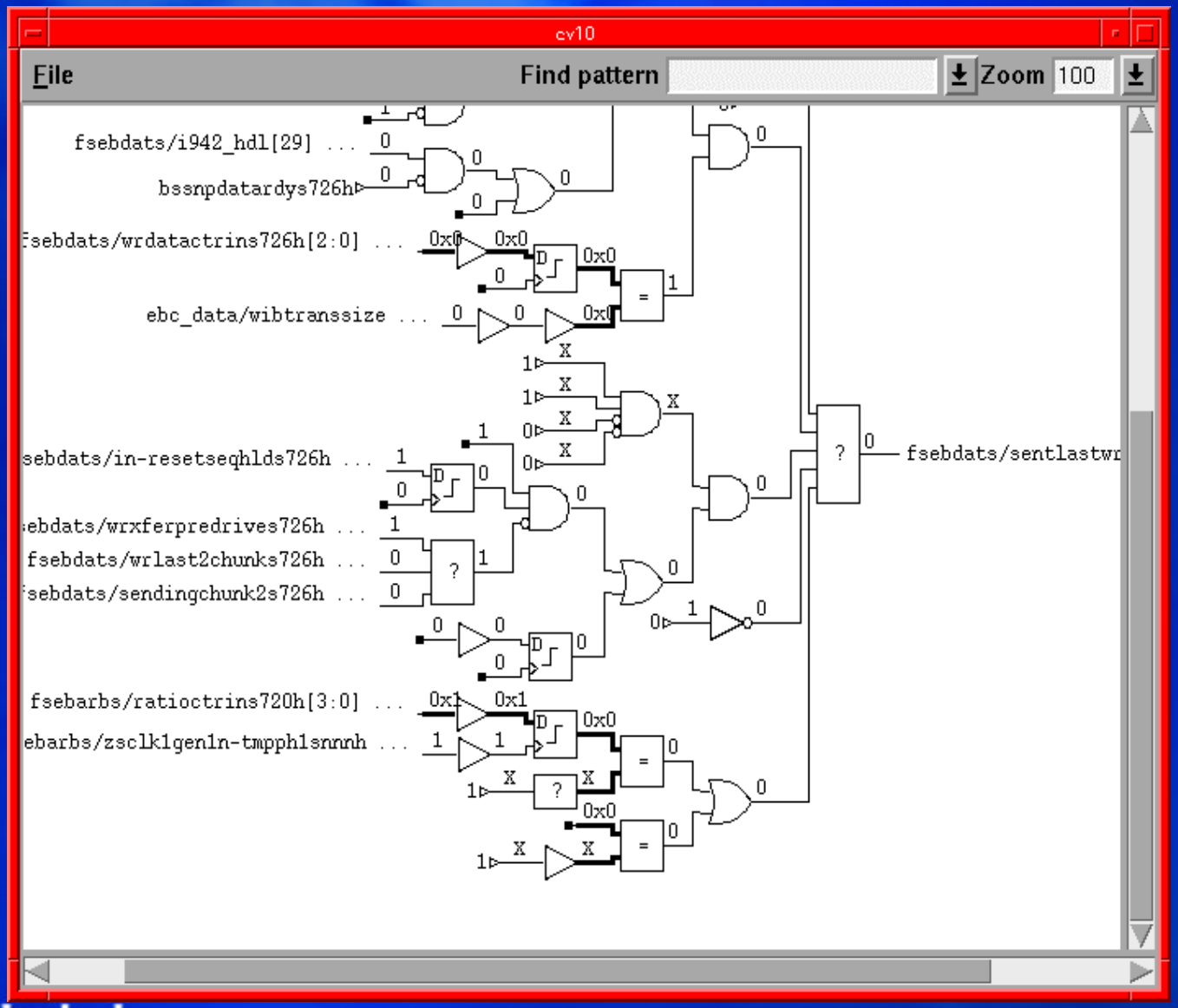
- Model checker (STE) is a built-in command in fl
- As a result: the specification and the commands to invoke the model checker are both fl programs.
  - Result:
    - Running large number of (smaller) model checking runs becomes practical
    - Reasoning about the verification results become reasoning about functional programs.
    - Mechanisms to manage and organize large programs are immediately available for large specifications/proofs.
    - The programming language provides powerful (textual) abstraction mechanisms making it easier to write high-level specification.

Forte Waveform display



- Draws single signals and/or vectors
- Can add, delete, duplicate, expand (vectors) any selected waveform
- Allows user to lock a time line and get values back annotation onto the circuit canvas
- Waveforms can be saved as PostScript code

Forte Circuit visualization



- Example of back-annotation of current circuit values
- For symbolic expressions a pull-up window is available for more details

Forte theorem prover

- Light-weight theorem prover
  - Model checking results as axioms
    - ensure that STE runs cover all cases
  - Decomposition of high-level specifications
    - model checkable lemma's

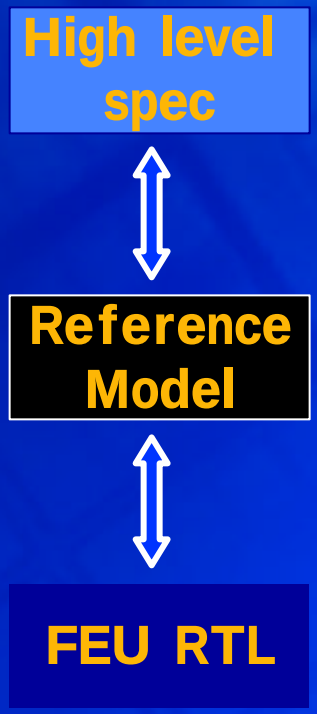
Floating point ADD example

- IEEE Std 754 says:
  - "... each of the operations shall be performed as if it first produced an intermediate result correct to infinite precision and unbounded range, and then [rounded] this intermediate result to fit in the destination's format." (section 5)
  - "When rounding toward negative infinity, the result shall be the format's value closest to (and no greater than) the infinitely precise result." (section 4.2)
- The top-level spec for FADD (rounding down) is

```
norm a AND norm b ==>
  (rounding_mode = TO_NEG_INF) ==>
    r <= a + b AND       // no greater than
    a + b < r + ulp      // closest to
    AND ...
```

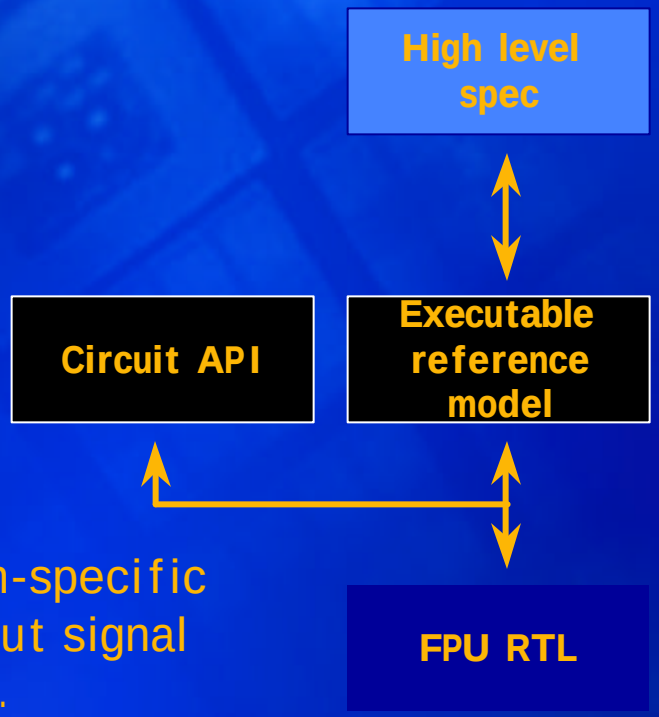
FADD FL Ref Model

```
norm a AND norm b ==>
  (rounding_mode = TO_NEG_INF) ==>
    r <= a + b < r + ulp  ...
```




```
// Adapted from: Feldman and Retter,
// Computer Architecture (McGraw-Hill, 94)
// pp. 489-491
let ADDmodel pc rc in1 in2 =
  ...
  // Find the amount of shift needed
  let diff = ex2 '-' ex1  in
  let rsh = MINv diff m in
  // Do the shift
  let sgf1' = srshift m rsh (sgf1@[F]) in
  let sgf2' = sgf2@[F] in
  // Perform the sum (or subtract)
  let add = (sign fp1 = sign fp2) in
  let sum = IF add THENv (sgf2' '+' sgf1')
                    ELSEv (sgf2' '-' sgf1')
  // Now perform rounding
  ...
```

A closer look at the ref model

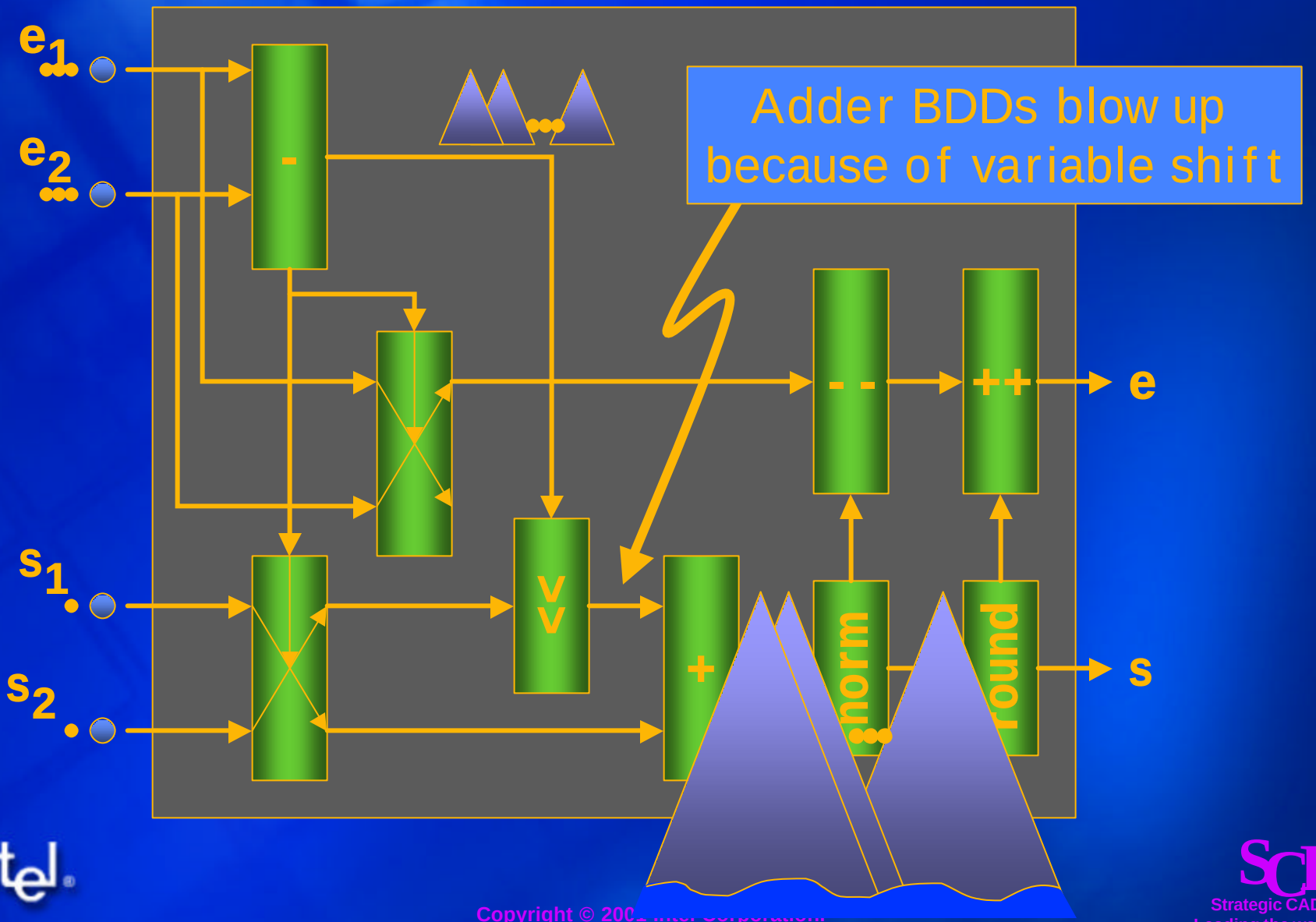


Theorem proving + model checking (manual, highly reusable)

STE (automatic, reusable)

API adds design-specific information about signal names, timing, …
Isolates ref model from RTL changes

Verifying RTL against Ref Model



Theorem prover support

- Need to split STE runs in cases with fixed mantissa shifts
  - theorem prove to make sure all cases are covered
- On high level
  - decompose IEEE spec into properties of rounding, exponent/mantissa values etc.
  - Model check these against FL reference model
  - **Invaluable to keep verifier honest**

What did we obtain

- IEEE level functional specs for FP operations of the FEU
- Proved correct datapath functionality for these operations
  - all variants, precision and rounding modes
- Compliance verified of the **same** gate-level descriptions
  - on which all pre-silicon dynamic validation is performed
  - from which the actual silicon is derived
- Extensive reuse (see next slide)
- Floating point FV accepted within Intel

Reuse

- P Pro -> P III
  - ?-arch change: power saving modes
  - minor API changes
    - Forte's waveform and circuit viewers essential
  - complete reuse of FL models and Thm proving effort
- P III streaming SIMD instructions
  - redesigned API
  - moderate redesign of reference models (1 float ? 2 packed floats/ints)
  - after that: high reuse
- P Pro -> P IV
  - completely new ?-architecture
  - reuse of FL models and Thm proving effort

Industrial strength FV

- Theory development and practical system building and use should go hand-in-hand
  - Clean theory makes system building manageable
  - Use of system drives theory development
- Developing a general solution is much more efficient than developing an “optimal” point tool
- No assumption that all FV can be “push button”
- In developing practical FV tools, think BIG
  - Today’s circuit are extremely large; even minor inefficiencies become bottlenecks

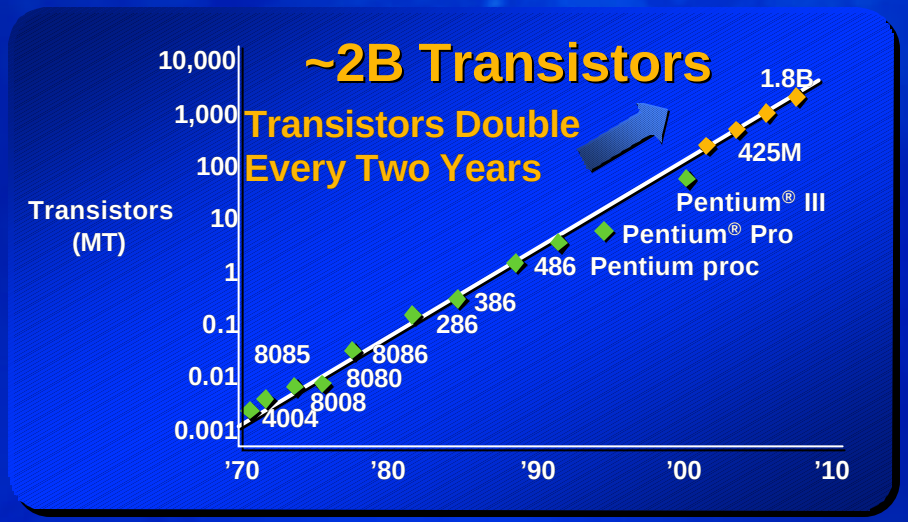
software is

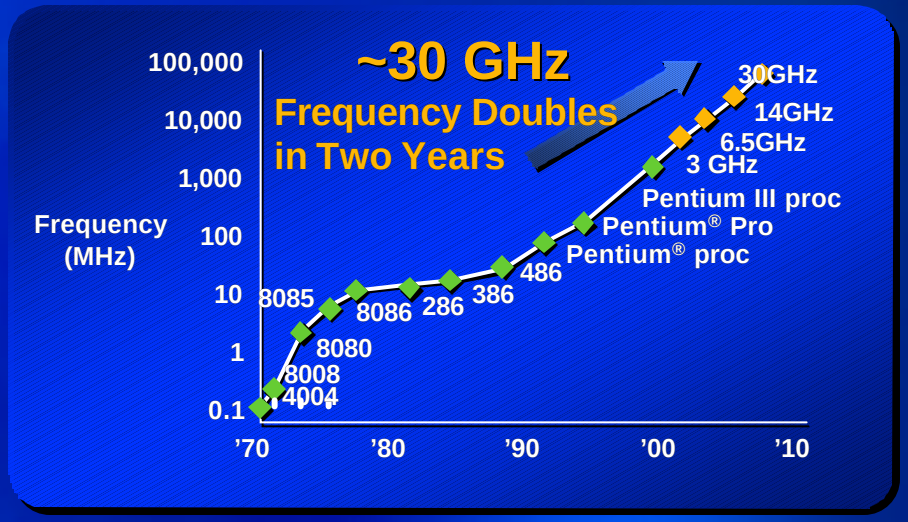
intel

SCL
Strategic CAD Labs
Leading the way in CAD

Trends

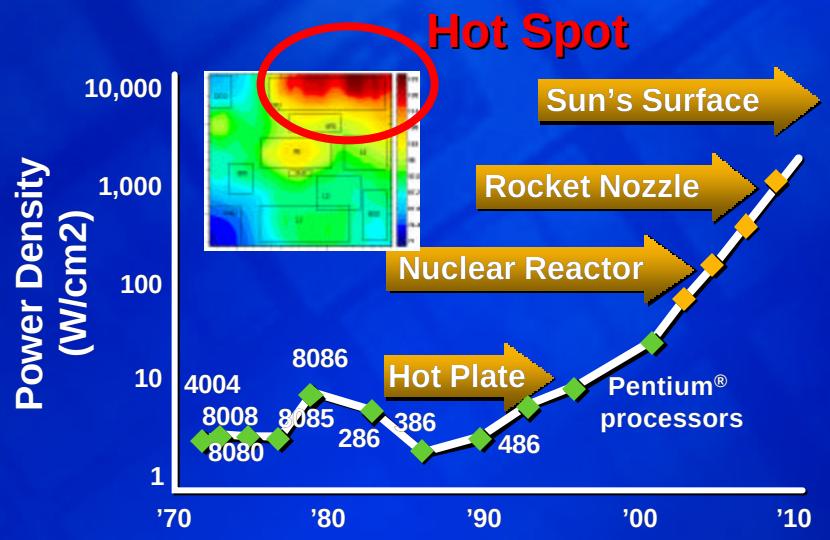
Moore's law



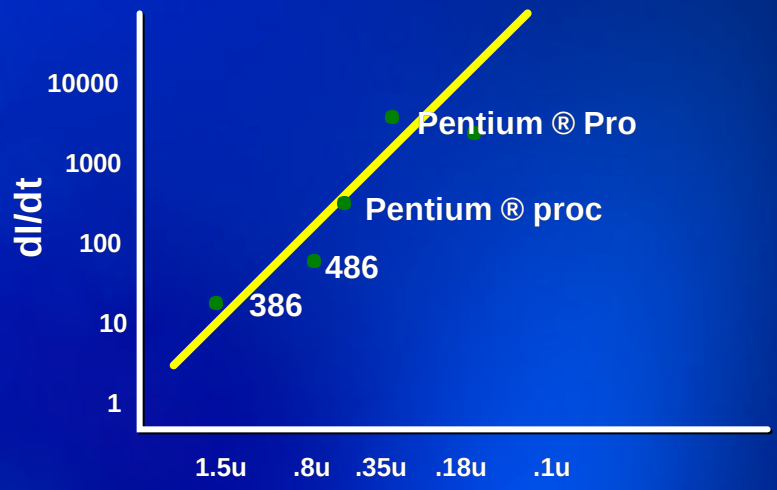


Power…

Density



Delivery



Obviously, this trend cannot continue

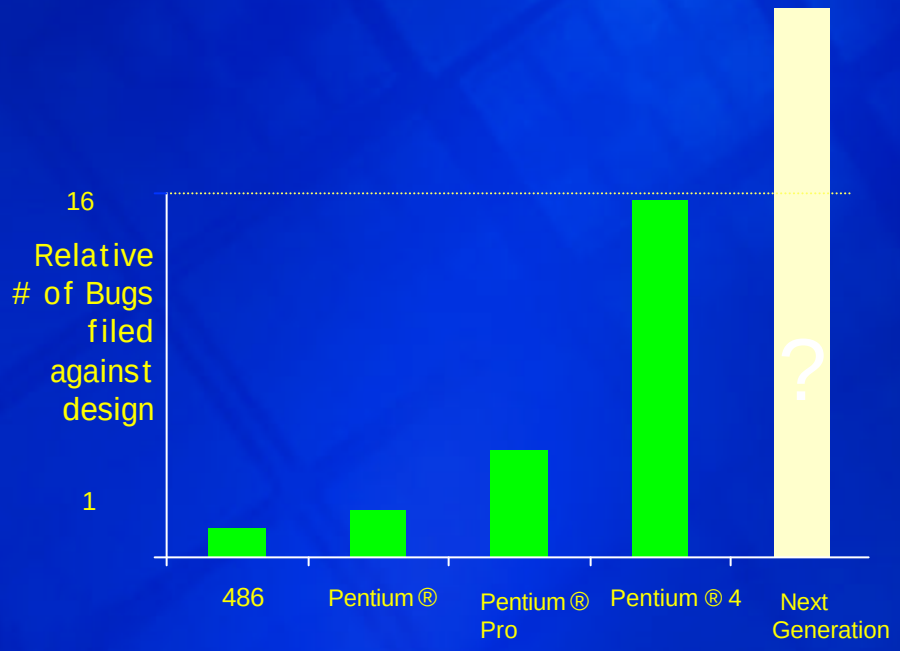
Consequences

- Marketing drive to use full transistor budget for performance increase
  - super scalar, out of order, branch prediction, speculation, …
- Power density trend forces change in micro-architectural design
  - clock gating, throttling, dynamic frequency & dI/dt control, …
- Ever more complex micro-architecture

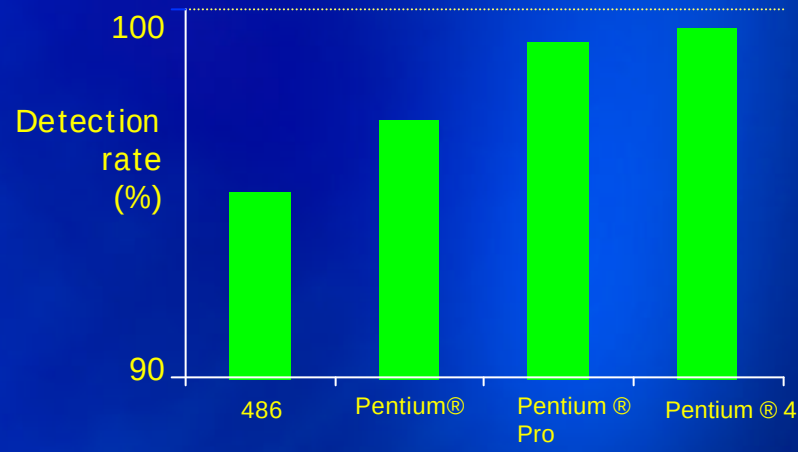
Moore's curse

Moore's curse

- # execution paths roughly exponential in number of gates (transistors)
- Classical validation (and design) methods are fighting a loosing battle



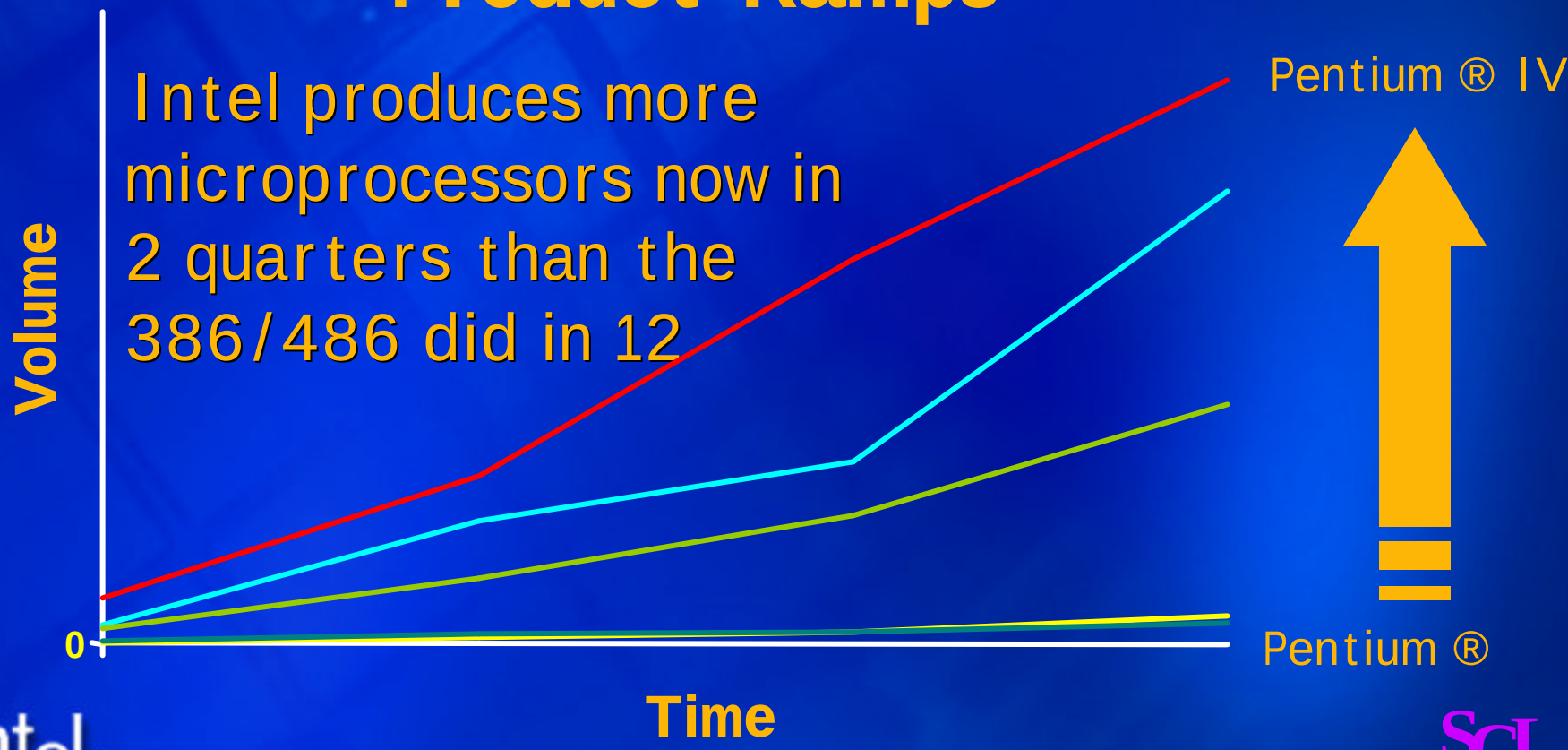
Bug creation



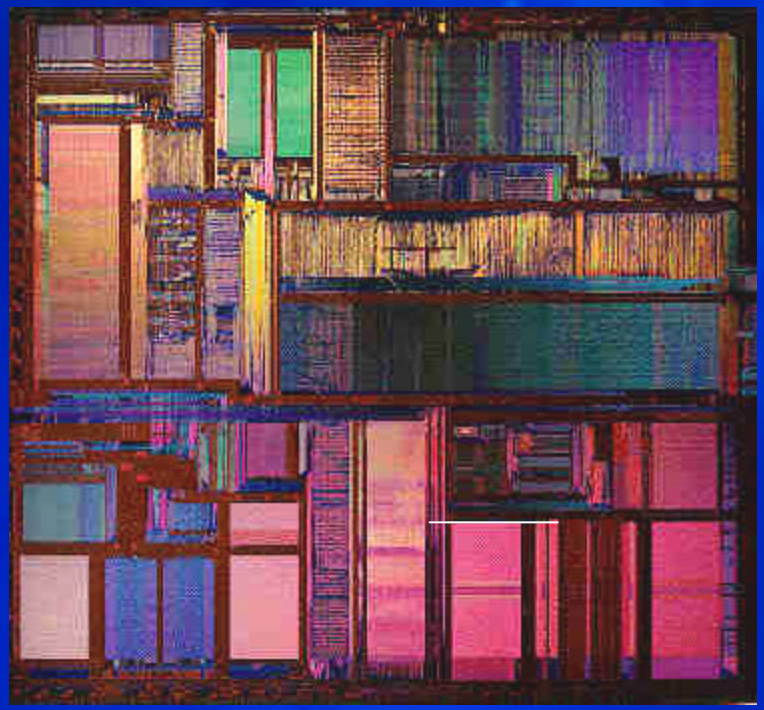
detection

& escapes can have an increasing $ impact

Product Ramps



Basic Formal Verification technology not really keeping up



- Symbolic Model Checking:
  - ~400 state variables with rich control logic
- Symbolic Trajectory Eval.
  - ~8,000 state variables with limited control logic
  - Generalized STE for rich control??
- **Certainly not on similar exponential growth curve**

intel

SCL
Strategic CAD Labs
Leading the way in CAD

Although trend data for software design are hard to come by, I would maintain they follow the same pattern

Ducking the trend

- Deploy FV on more 'abstract' models
  - Hence, integrate FV in the design process
    - otherwise very difficult to migrate FV results to implementation
    - implies FV must evolve from arcane art to mature science
      - methodology
      - versatile tool support

you can do worse than a forte-like system

Ducking the trend ctd

- Leverage abstract interpretation/static analysis in FV tools

Crude syntactic analysis ---- **Abstract interpretation** ---- Model checking

  - Recent examples
    - Bebob; Ball/Rajamani; Microsoft research
    - AX and successors; Holzmann, Bell labs
  - Research suggestion
    - Develop abstract interpration FV tool bench + methodology for **clean** language (say functional)

we can do worse than taking our cue from such research

intel

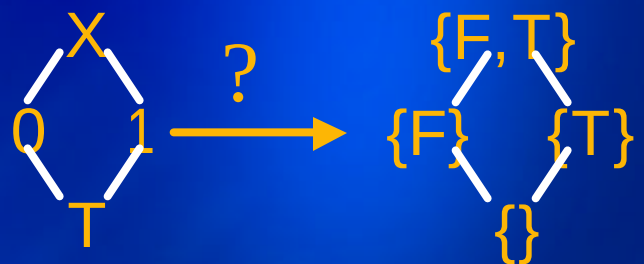
QUESTIONS?

Backup

Two key STE ideas

- Transition relation ultra-partioned
  - graph of boolean gates + their transfer functions
  - post images partitioned in same way
- Compute over below lattice
  - X is a fixed point for every gate
  - **automatic dynamic pruning (slicing)**
    - as long as a gate is not relevant for computation its inputs (and outputs) will have value X

GSTE far reaching generalization of STE and formally defined as an abstract interpretation



intel

SCL
Strategic CAD Labs
Leading the way in CAD

SCL
Strategic CAD Labs
Leading the way in CAD