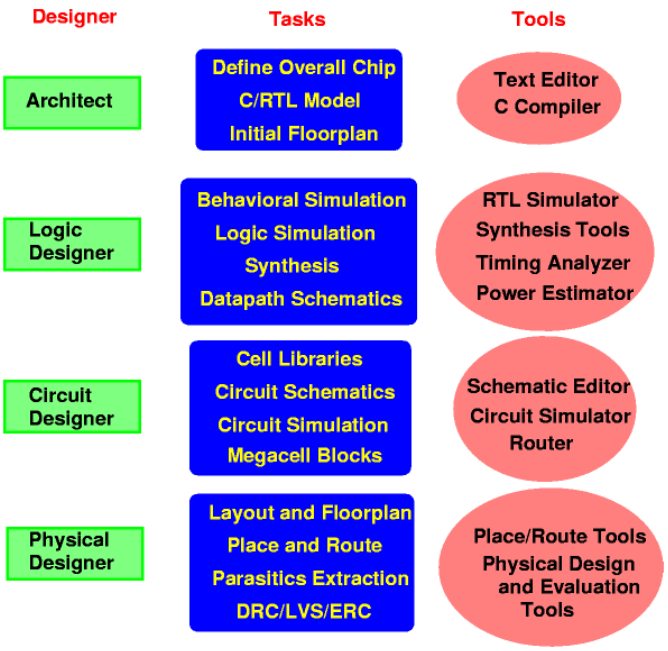
Hardware Verification
Application of formal techniques to chip designs

Contents

Steps in Designing Hardware



The Verification Problem

- Moore's Law: Complexity of integrated circuits doubles every 18 months
  - The International Technology Roadmap for Semiconductors (ITRS) estimates that there will be around 5,000,000,000 transistors on a single chip by 2010
- State–space explosion: a design with around 200 memory elements has more states than the number of protons in the universe

Hardware Bugs

What is a "bug"?

- When the design does not match the specification
  - Problem is that complete (and consistent) specifications may not exist for many products
  - The difficult aspect of making an X–86 compatible chip is not in implementing the X–86 instruction set architecture, but in matching the behavior with Intel chips
- Something which the customer will complain about

"It's not a bug, it's a feature"

Design Bug Distribution in Pentium 4

Source: EE Times, July 4, 2001

Type of Bug	Percentage
"Goof"	12.7
Miscommunication	11.4
Microarchitecture	9.3
Logic/microcode changes	9.3
Corner cases	8
Powerdown	5.7
Documentation	4.4
Complexity	3.9
Initialization	3.4
Late definition	2.8
Incorrect RTL assertions	2.8
Design mistake	2.6

42 Million transistors,
1+ Million lines of RTL

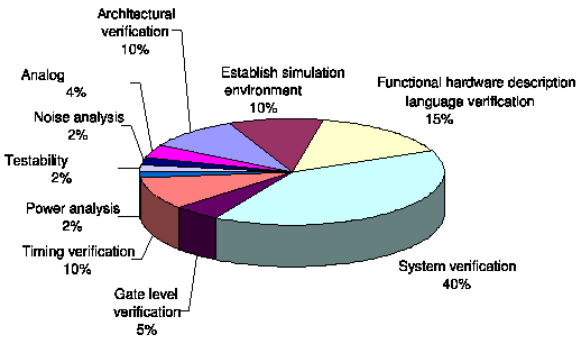
100 high–level logic bugs found through formal verification

Verification Effort

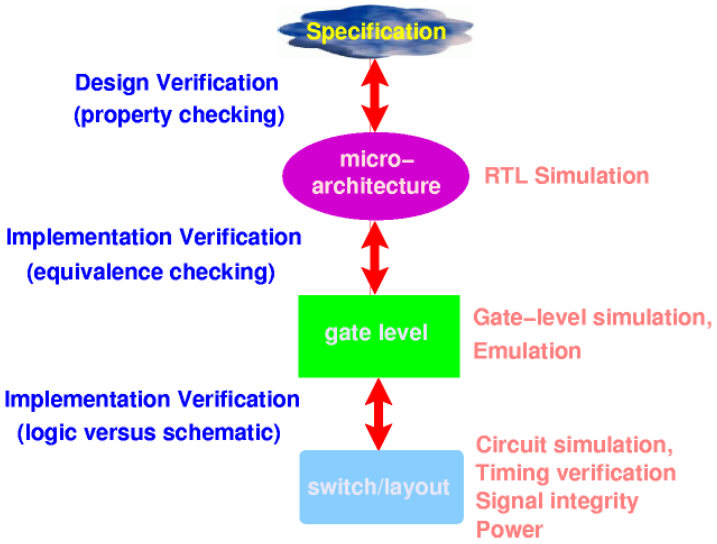
Source: 1999 ITRS

Over 50% of the design effort is in verification

- Many different aspects of verification



Design and Implementation Verification



Checking the Logic of the Entire Design

Is there a formal verification technique which can be applied to the entire chip?

- There is only one approach which will scale with the design: Simulation
- Most common technique used in industry today
- Cycle–based simulation can exercise the design for millions of cycles
  - Unfortunately, there are no good measures of coverage, and the question of when to stop simulation is an open one
- Emulation is a related approach to speed up the effort (used for verifying the first Pentium processor)
  - Developing another accurate model could be an issue

When Do You Tapeout?

Source: EE Times, July 4, 2001

Motorola criteria

- 40 billion random cycles without finding a bug
- Directed tests in verification plan are completed
- Source code and/or functional coverage goals are met
- Diminishing bug rate is observed
- A certain date on the calendar is reached

Functional Verification of Processors

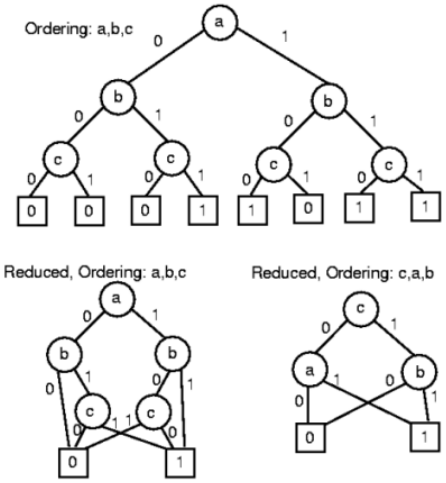
- PowerPC behavioral simulator has about 40,000 lines of C++ code
- IBM developed a model–based test generator, an expert system which contains a formal model of processor architecture, and a heuristic data base of testing knowledge
- Goal is to improve the quality of tests
  - Verification of previous RS/6000 processor involved 15 billion simulation cycles (large team with hundreds of computers during a year)
- Monitors can be compiled with the simulation model to make it easy to check whether a failure has occurred
- Commercial tools and environments are available to facilitate simulation
- Some efforts to automatically generate test cases

Equivalence Checking

- Most common technique of formal verification used in industry today
- Typically, the gate–level implementation is compared with the representation at a higher level (RTL)
- A canonical representation allows easy comparison of two functions
  - representation should be efficient in memory and time
  - problem of checking Boolean Equivalence is NP–complete
- Commercial tools available from most tool vendors
- Limitation is that some functions (such as multipliers) require exponential space for the representation

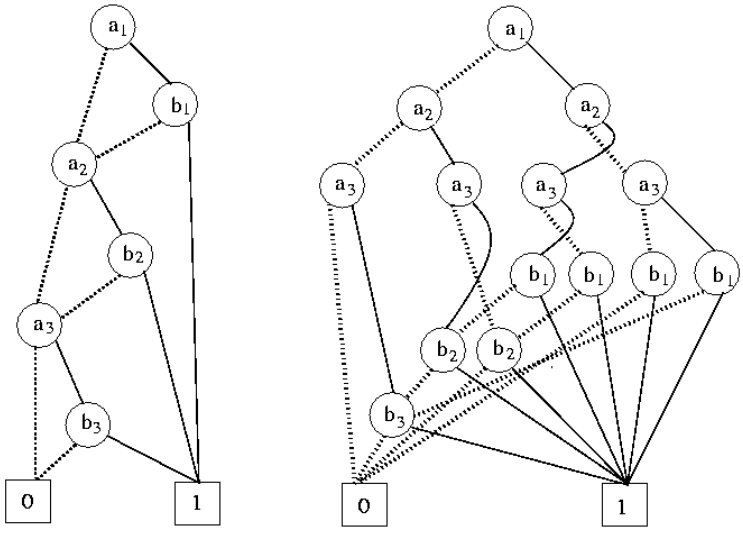
Binary Decision Diagrams (BDDs)

Represent and manipulate Boolean functions (sets of states, state–transition relationships, ...)



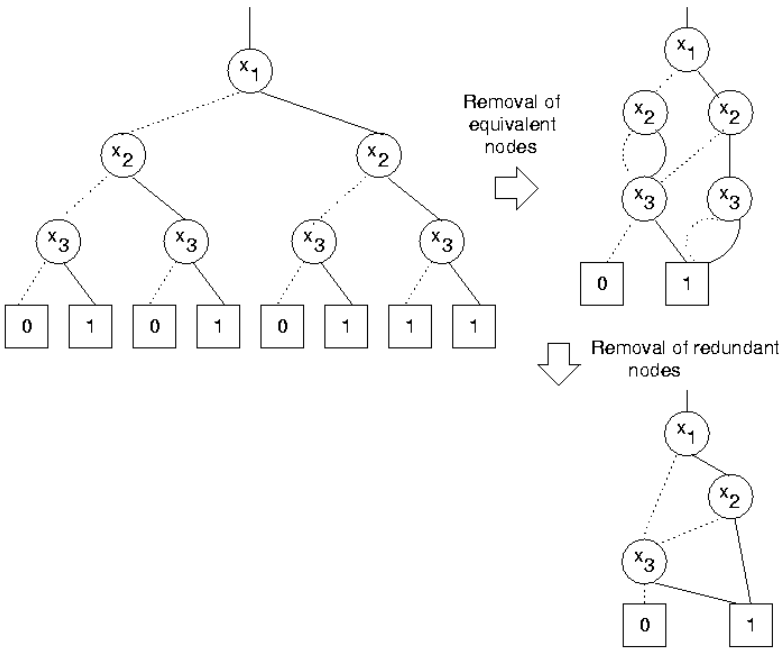
Ordering Dependence of BDDs

Function : $a_1b_1 + a_2b_2 + a_3b_3$



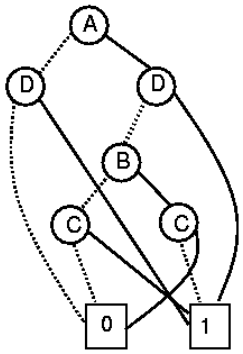
Ordering $a_1<b_1<a_2<b_2<a_3<b_3$ $a_1<a_2<a_3<b_1<b_2<b_3$

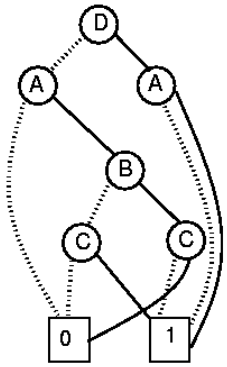
Reducing a BDD



Variable Reordering

(A and(B xor C)) or D



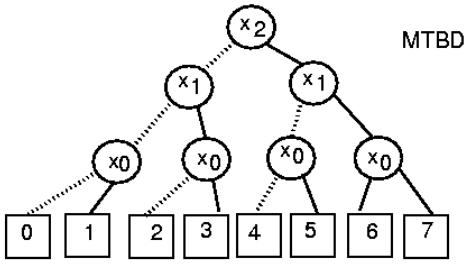


$F_{11} = 1$ F_{10} =graph at B

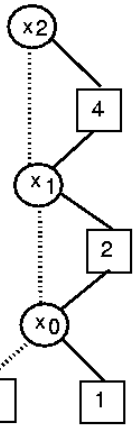
$F_{01} = 1$ F_{00} =0

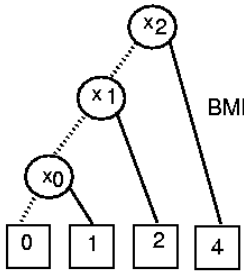
(D,(A,1,1),(A,B,0))

Different Representations



MTBDD EVBDD



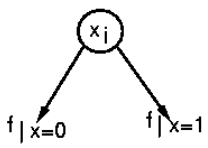


BMD,*BMD

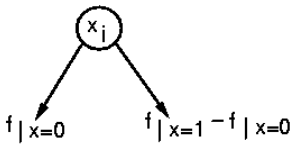
$X=4x_2+2x_1+x_0$

Hybrid Decision Diagrams

- Useful for verifying arithmetic circuits
- For state variables corresponding to datapath registers, behaves like a BMD
  - BMDs have linear representation for some functions whose BDD representation has exponential complexity
- For state variables corresponding to control registers, behaves like an MTBDD



control variables

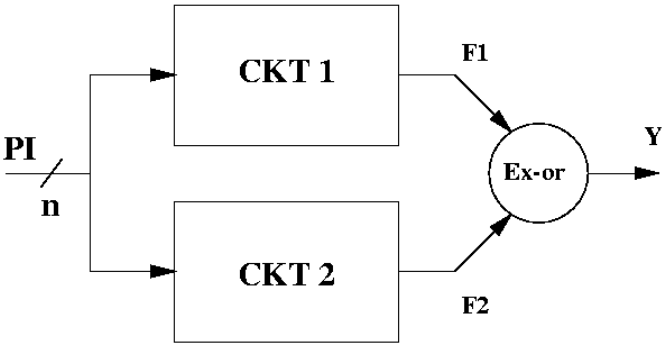


datapath variables

Manufacturing Test Techniques in Logic Verification

Check equivalence of states by generating a test for a "stuck–at 0" fault at the output

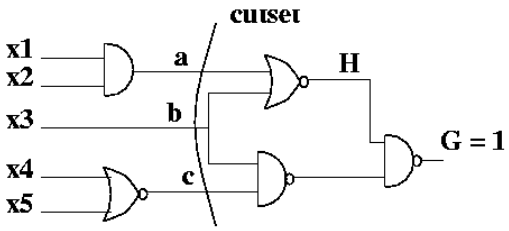
Though this is the "satisfiability problem", unlike using BDDs, process is usually not memory limited

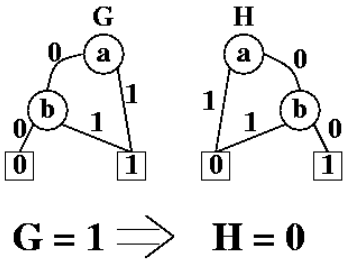


"Learning" Techniques to Speed Up Search

Automatic Test Pattern Generation (ATPG) may fail to find a solution in a bounded amount of time

Use information on internal nodes which correspond to each other





Model Checking in Industry

- Model checkers (most based on SMV from CMU) are beginning to be used to verify hardware designs
- Example, RuleBase at IBM
  - RuleBase incorporates reduction techniques which automatically reduce the design size before the model checker is called
  - "Irritators" define the input assumptions to the module under test
  - Sugar is used to write properties
- Most companies have a group dedicated to model checking
- Electronic Design Automation tool providers are beginning to release model–checking tools

Application of Model Checking to the IBM Power4

- Applied "functional formal verification" (equivalence checking and model checking) to some extent on approximately 40 design components, including portions of the instruction unit, floating point unit, control logic, memory subsystem and I/O chips
- Found more than 200 design flaws at various stages and of varying complexity
- At least one bug was found by almost every application of formal verification
- Estimate: 15% of bugs would have evaded simulation
  - Some of the bugs literally escaped 1–2 years of simulation
- Found that application of formal techniques by designers themselves, using formal team as consultants, was very fruitful

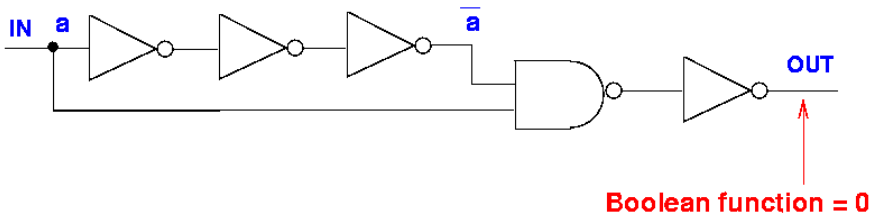
Application of Theorem Proving to Arithmetic Circuits

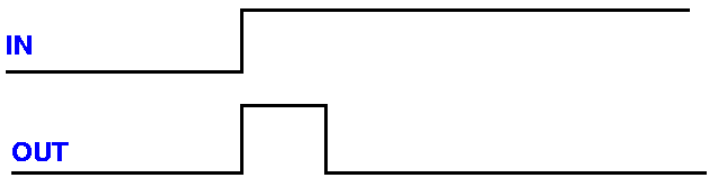
- ACL2, successor to Nqtm (Boyer–Moore Theorem Prover), used at AMD to formally verify floating point units
- First used by Moore et al. to check the proof of correctness of the Kernel of the AMD 5k86 floating point division algorithm
  - mechanically checked proof of correctness of the kernel
- Recently used to verify the RTL of the K7 Floating Point Unit
  - RTL primitives are logical operations on bit vectors
  - Developed theory of bit vectors to prove RTL correct with respect to the more abstract IEEE standard

Symbolic Simulation

- Equivalence checking between RTL and circuit schematics is difficult for some circuits (e.g., custom arrays
  - Critical timing and self–timed control logic components
  - Large number of bit–cells
  - Inherently complex sequential logic blocks
  - Dynamic Logic
- Traditional tools fail on such circuits
  - Very large state space and too many initial state/input sequence combinations for simulation–based tools
  - Boolean equivalence tools can only check static cones of logic, and do not capture dynamic behavior

Control For Custom Array Structures

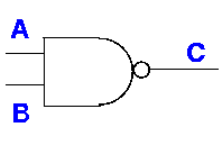


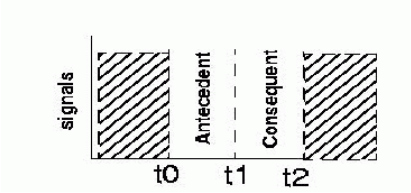


- OUT pulse fans out to array READ/WRITE control signals
- Equivalence checking does not work

Scalar Simulation

To prove that circuit is a NAND gate, exhaustive simulation requires 2^n vectors



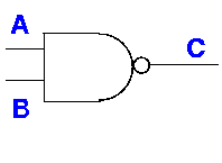


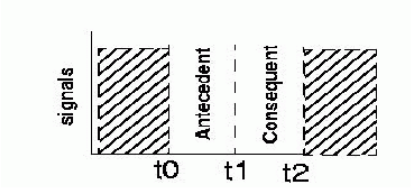
Antecedent	Consequent
A = 0 (t0,t1) and B = 0 (t0,t1)	C is 1 (t1,t2)
A = 0 (t0,t1) and B = 1 (t0,t1)	C is 1 (t1,t2)
A = 1 (t0,t1) and B = 0 (t0,t1)	C is 1 (t1,t2)
A = 1 (t0,t1) and B = 1 (t0,t1)	C is 0 (t1,t2)

Table could be viewed as "Antecedent => Consequent"

Ternary Simulation

Using three values (0,1,X), N–input NAND requires N+1 vectors to verify

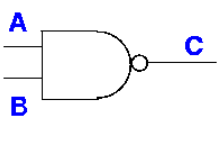


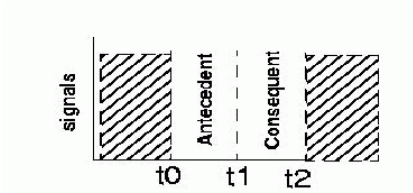


Antecedent	Consequent
A = 0 (t0,t1) and B = X	C = 1 (t1,t2)
A = X and B = 0 (t0,t1)	C = 1 (t1,t2)
A = 1 (t0,t1) and B = 1 (t0,t1)	C = 0 (t1,t2)

Symbolic Simulation

Exhaustive verification: N–input NAND requires 1 vector and N variables





Antecedent: A = "a" (t0,t1) and B = "b" (t0,t1)

["a" and "b" are Boolean variables]

Consequent: C = [~(a AND b)](t1,t2)

Symbolic Trajectory Evaluation

- VERSYS symbolic trajectory evaluation tool
  - Based on VOSS tool (from UBC/CMU)
- Trajectory formulas
  - Boolean expressions with the temporal "next–time" operator
  - Ternary valued states represented by a Boolean encoding
- Properties of the type Antecedent implies Consequent
  - Antecedent and consequent are trajectory formulas
  - Antecedent sets up the stimulus and the state of the circuit
  - Consequent specifies the constraint on the state sequence

Array Verification at Motorola

- All arrays on latest Power–PC processor verified using VERSYS
- Complete array verification took about 8%–10% of array design time
- Properties proved on RTL and transistor–level models

- Bugs found during custom array equivalence checking
  - Incorrect clock regenerators feeding latches
  - Control logic errors in READ/WRITE enables
  - Violation of "one–hot" property assumptions
  - Scan chain hookup errors
  - Potential circuit–related problems such as glitches and races

Verifying Compliance of Intel Floating–Point Hardware

Used internal formal verification system, Forte, an evolution of the VOSS system

- Seamlessly integrates several types of model–checking engines with lightweight theorem proving and extensive debugging capabilities
- Model checkers:
  - Based on symbolic trajectory evaluation
  - Word–level model checker (linear–time checker tailored to verifying arithmetic circuits, based on Hybrid Decision Diagrams
- Design partitioned to manage complexity, with lightweight theorem prover used to guarantee that no mistakes were made in the partitioning process
- Forte system includes significant support for efficient debugging

Conclusions

Formal techniques are finding increasing use in verifying hardware

- Beginning to see many success stories in real designs
  - by applying the appropriate approach to parts of the design
- Lot of opportunity for developing improved methods
  - Integrating different techniques
  - Better abstractions to deal with large designs