

Verification of SoC Designs

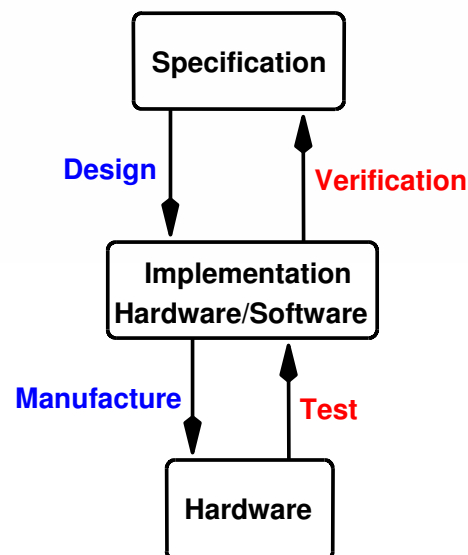
- Simulation-based techniques
- Formal analysis
- Dealing with state explosion
- Verification of embedded software

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
1

Verification versus Test



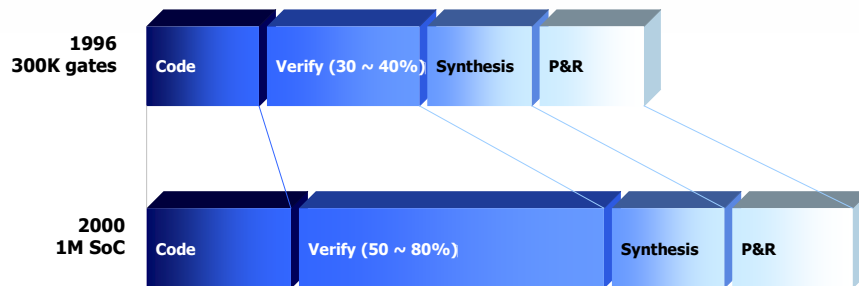
SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
2

Verification Effort

- **Verification** portion of design increases to anywhere from **50 to 80%** of total **development effort** for the design.



*Verification methodology manual, 2000-
TransEDA*

SoC Design - ICS, Fall 2010
November 13, 2010

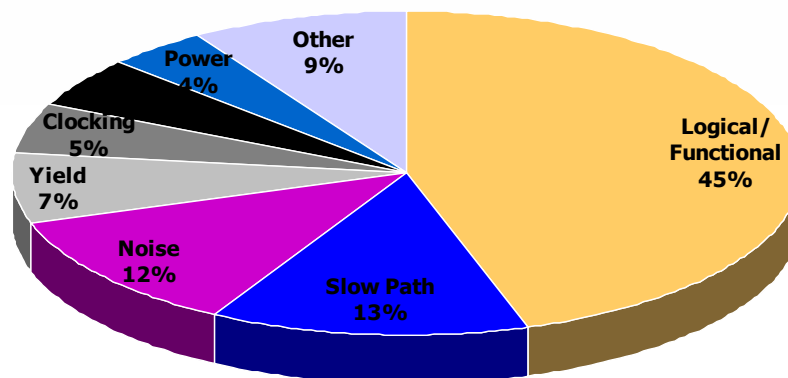
J. A. Abraham

Verification of SoC Designs

3

Percentage of Total Flaws

- About **50%** of flaws are functional flaws.
 - Need verification method to fix **logical & functional** flaws



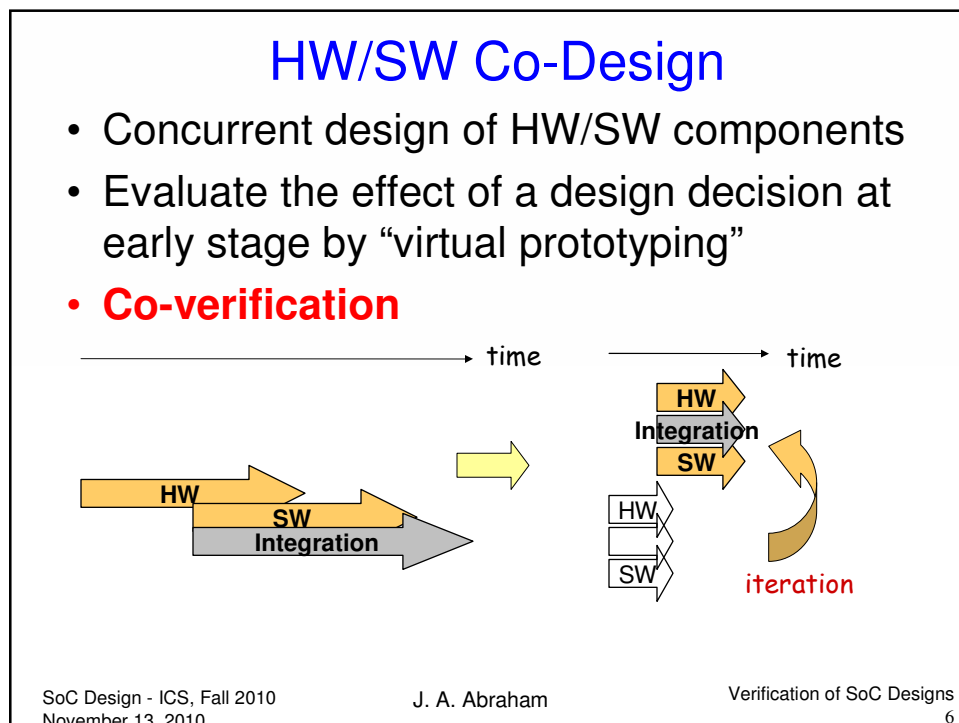
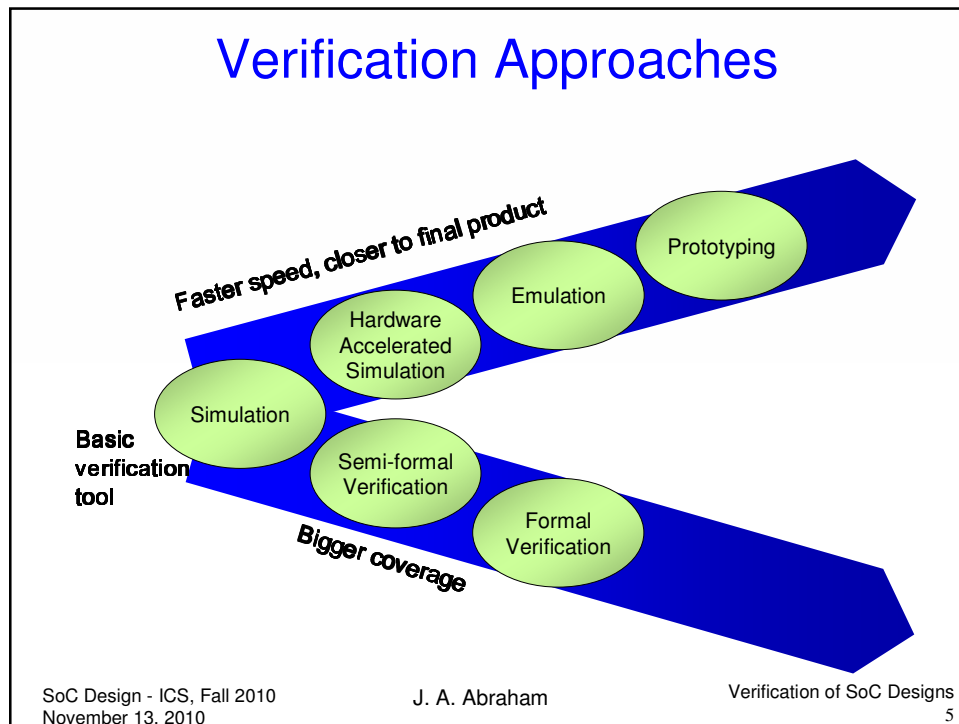
From Mentor presentation material, 2003

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs

4



Verification Options

- Simulation Technologies
- Equivalence Checking
- Formal Analysis (Model Checking)
- Physical Verification and Analysis

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
7

Simulation Technologies

- Event-based Simulators
- Cycle-based Simulators
- Transaction-based Simulators
- Code Coverage
- HW/SW Co-verification
- Emulation Systems
- Rapid Prototyping Systems
- Hardware Accelerators
- AMS Simulation
- Numerical Simulation (MATLAB)

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
8

Static Technologies

- “Lint” Checking
 - Syntactic correctness
 - Identifies simple errors
- Static Timing Verification
 - Setup, hold, delay timing requirements
 - Challenging: multiple sources

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
9

Formal Techniques

- Theorem Proving Techniques
 - Proof-based
 - Not fully automatic
- Formal Model Checking
 - Model-based
 - Automatic
- Formal Equivalence Checking
 - Reference design \leftrightarrow modified design
 - RTL-RTL, RTL-Gate, Gate-Gate implementations
 - No timing verification

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
10

Physical Verification & Analysis

Issues for physical verification:

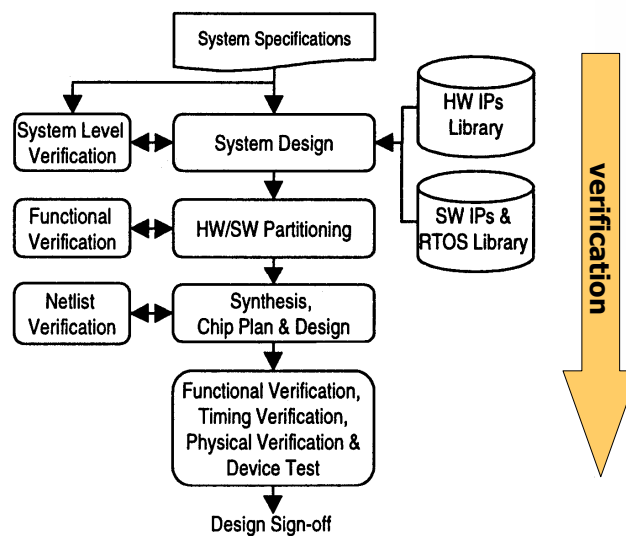
- Timing
- Signal Integrity
- Crosstalk
- IR drop
- Electro-migration
- Power analysis
- Process antenna effects
- Phase shift mask
- Optical proximity correction

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
11

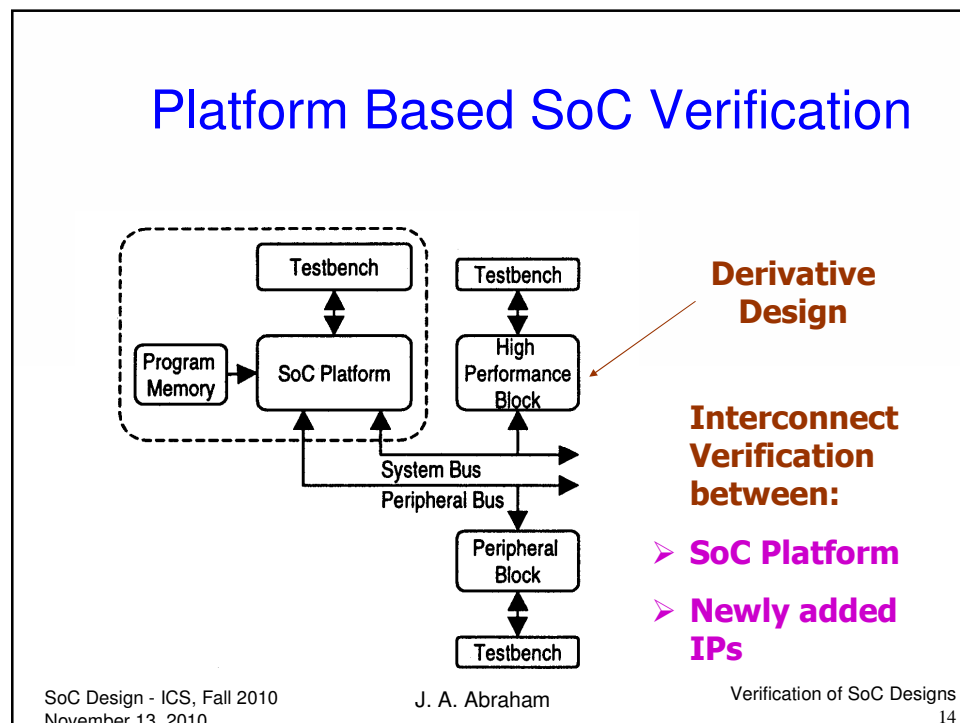
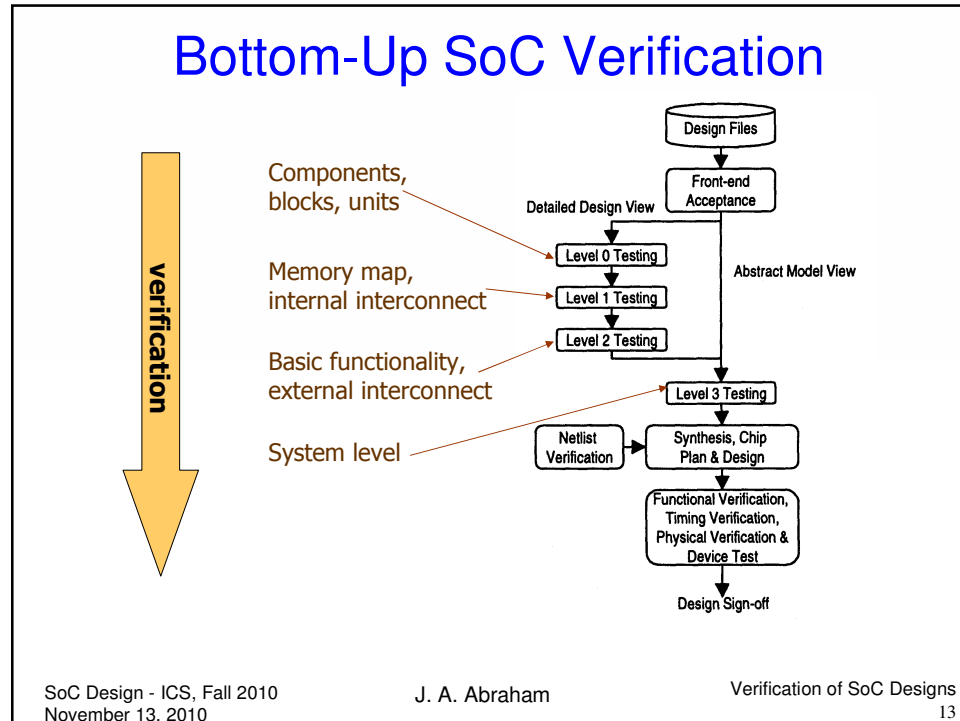
Top-Down SoC Verification



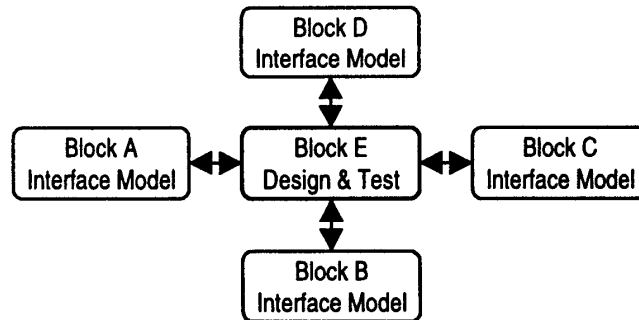
SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
12



System Interface-driven SoC Verification



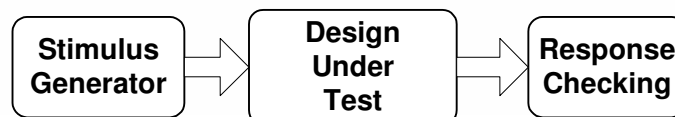
**Besides Design-Under-Test,
all others are interface models**

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
15

Traditional Testbench



- **Problems of Traditional Testbench**

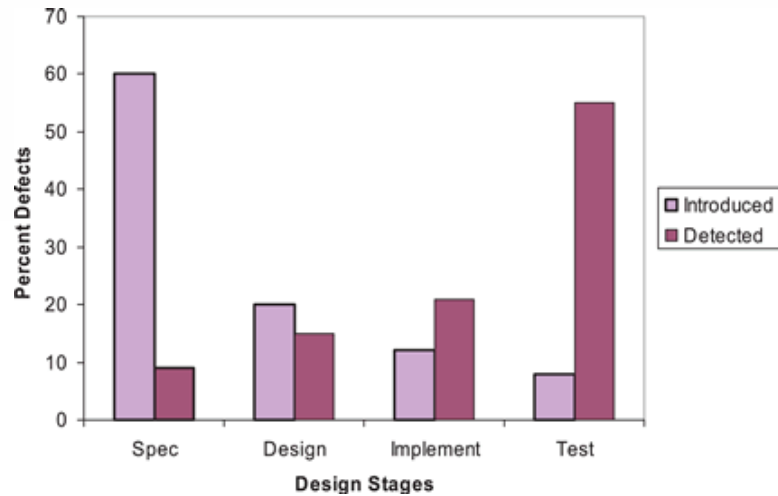
- Real-World Stimuli
- System-Level Modeling
- High-Level Algorithmic Modeling
- Test Automation
- Source Coverage

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
16

“Bug” Introduction and Detection



SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
17

Executable Specification

- Procedural Language for Behavioral Modeling
 - Design Productivity
 - Easy to model complex algorithm
 - Fast execution
 - Simple Testbench
 - Tools
 - Native C/C++ through PLI/FLI
 - Extended C/C++ : SpecC, SystemC
- Verify it on the fly!
 - Test vector generation
 - Compare RTL Code with Behavioral Model
 - Coverage Test

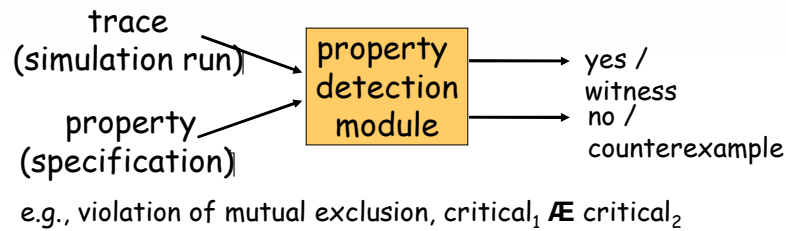
SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
18

Property Detection

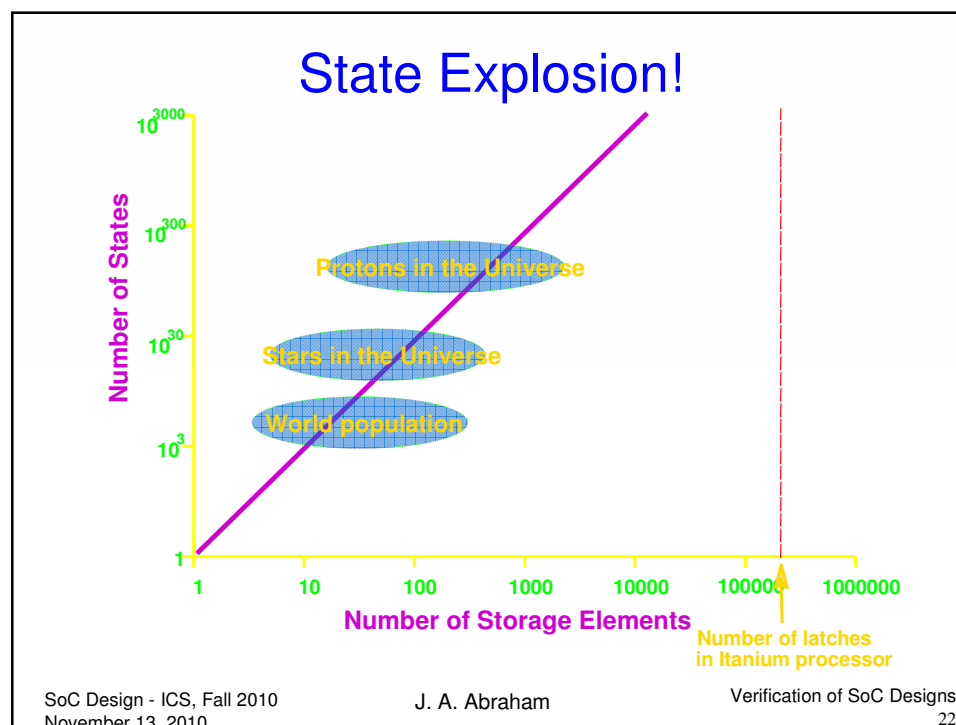
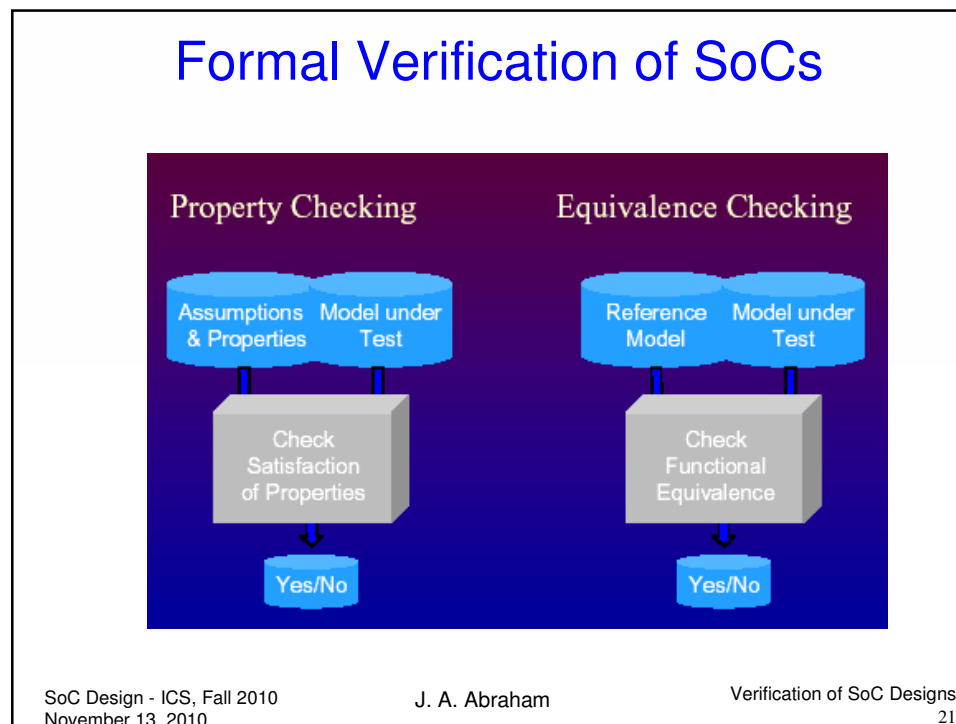
Property detection: to decide whether a simulation run (trace) of a design satisfies a given property



Example: Properties written in PSL/Sugar

Specifying Properties (Assertions) in Industry Tools

- Open Vera Assertions Language (Synopsys)
- Property Specification Language (PSL) (IBM, based on *Sugar*)
 - Accelera driving consortium
 - IEEE Std. 1850-2005
- Accelera Open Verification Library (OVL) provides ready to use assertion functions in the form of VHDL and Verilog HDL libraries
- SystemVerilog is a next generation language, added to the core Verilog HDL
 - IEEE Std. 1800-2005



Abstractions to Deal with Large State Spaces

- Model checking models need to be made smaller
- Problem: **State-Space Explosion**
- Smaller or “reduced” models must retain information
 - Property being checked should yield same result
- **Balancing solution: Abstractions**

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
23

Program Transformation Based Abstractions

- Abstractions on Kripke structures
 - Cone of Influence (COI), Symmetry, Partial Order, etc.
 - State transition graphs for even small programs can be very large to build
- Abstractions on Program Text
 - Scale well with program size
 - High economic interest

Static Program Transformations

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
24

Types of Abstractions

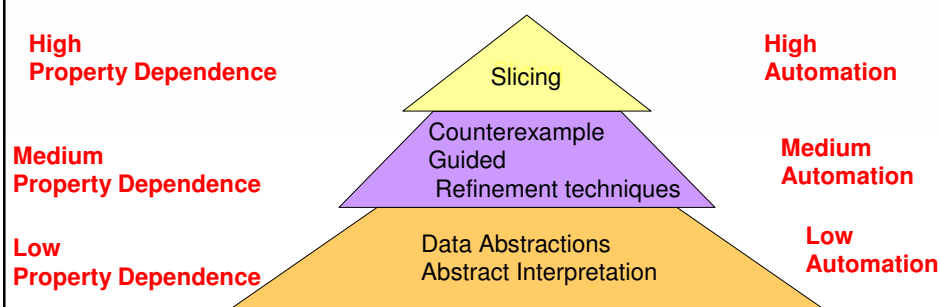
- Sound
 - Property holds in abstraction implies property holds in the original program
- Complete
 - Algorithm always finds an abstract program if it exists
- Exact
 - Property holds in the abstraction iff property holds in the main program

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
25

Abstraction Landscape



SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
26

Verification of challenging problems with high level static analysis

Antecedent Conditioned Slicing

- RTL abstraction technique
- Applied to LTL formulas
 - $G(a \Rightarrow c)$
- Theoretically complex, practically effective
- USB 2.0 protocol verification

Property checking

- High level symbolic simulation
 - Symbolic simulation of antecedent
 - Symbolic simulation of all CFG nodes
- Domain aware analysis
 - Function-wise case splitting
- Decision procedure
 - Model checker

Program Slicing

- Program transformation involving statement deletion
- “Relevant statements” determined according to *slicing criterion*
- Slice construction is completely *automatic*
- Correctness is *property specific*
 - Loss of generality
- Abstractions are sound and complete

Specialized Slicing Techniques

- Static slicing produces large slices
 - Has been used for verification
 - Semantically equivalent to COI reductions
- Slicing criterion can be enhanced to produce other types of slices
 - Amorphous Slicing
 - Conditioned Slicing

Conditioned Slicing

- Slices constructed with respect to set of possible input states
- Characterized by first order, predicate logic formula
- Augments static slicing by introducing condition
 - $\langle C, I, V \rangle$
 - Constrains the program according to condition C
- Canfora et al

Example Program

```

begin

1:      read(N);
2:      A = 1;
3:      if (N < 0) {
4:          B = f(A);
5:          C = g(A);
6:      } else if (N > 0) {
7:          B = f'(A);
8:          C = g'(A);
9:      } else {
10:         B = f''(A);
11:         C = g''(A);
12:      }

11:     print(B);
12:     print(C);

end

```

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
31

Example Program: Static Slicing wrt <11, B>

```

begin

1:      read(N);
2:      A = 1;
3:      if (N < 0) {
4:          B = f(A);
5:          C = g(A);
6:      } else if (N > 0) {
7:          B = f'(A);
8:          C = g'(A);
9:      } else {
10:         B = f''(A);
11:         C = g''(A);
12:      }

11:     print(B);
12:     print(C);

end

```

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
32

Example Program: Conditioned Slicing wrt $\langle (N < 0), 11, B \rangle$

```
begin

1:      read(N);
2:      A = 1;
3:      if (N < 0) {
4:          B = f(A);
5:          C = g(A);
6:      } else if (N > 0) {
7:          B = f'(A);
8:          C = g'(A);
9:      } else {
10:         B = f''(A);
11:         C = g''(A);
12:      }

11:     print(B);
12:     print(C);

end
```

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs

33

Verification Using Conditioned Slicing

- Slicing part of design irrelevant to property being verified
- Safety Properties of the form
– $G (\text{antecedent} \Rightarrow \text{consequent})$
- Use *antecedent* to specify states we are interested in

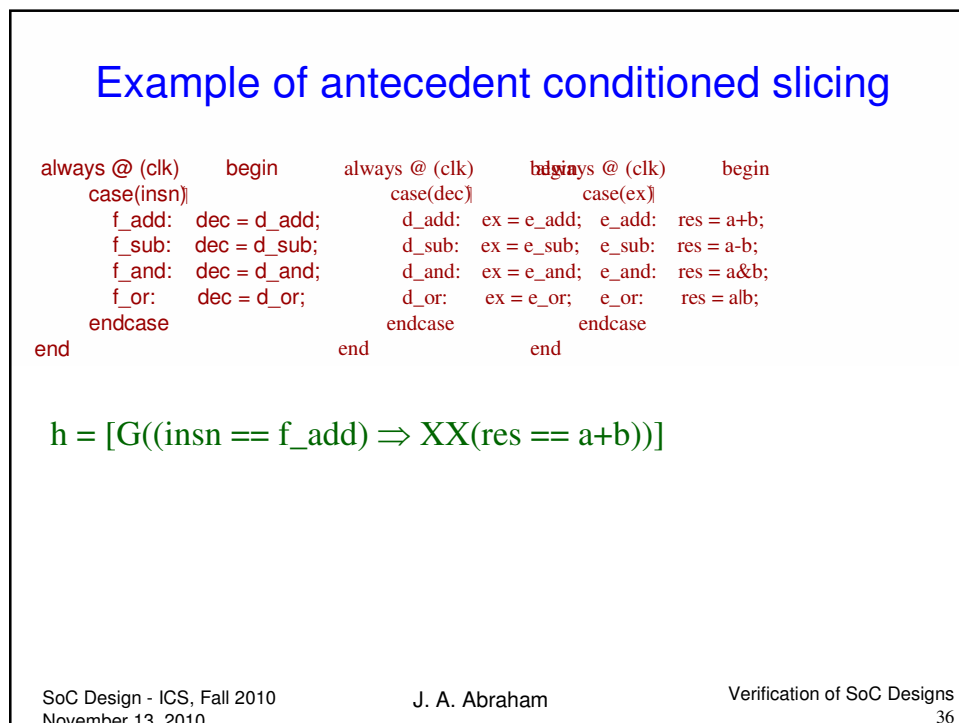
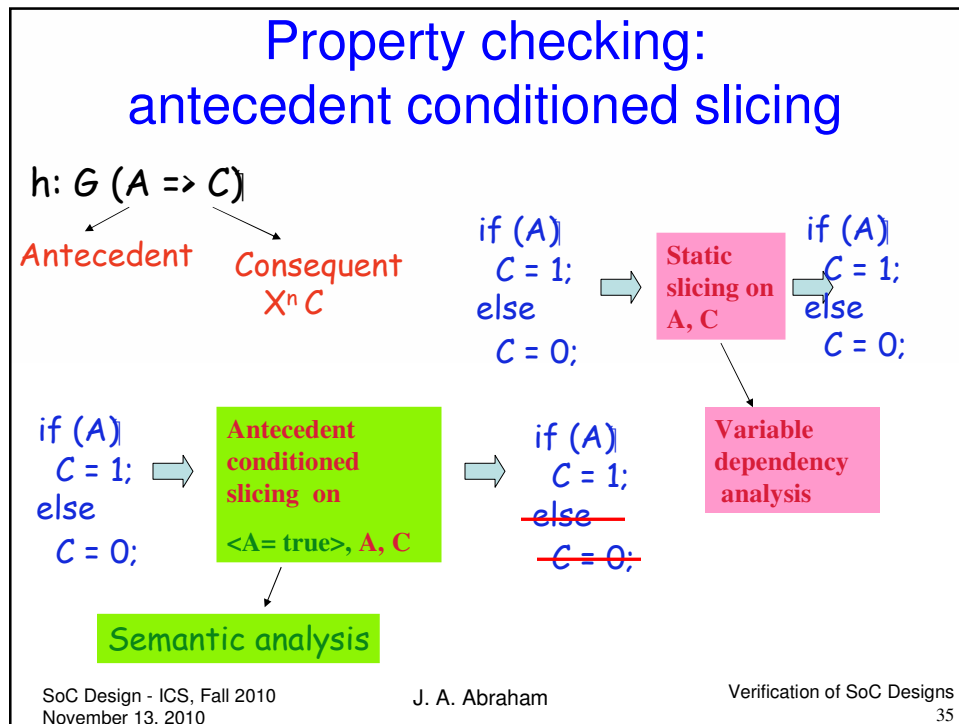
*We do not need to preserve program
executions where the antecedent is false*

SoC Design - ICS, Fall 2010
November 13, 2010

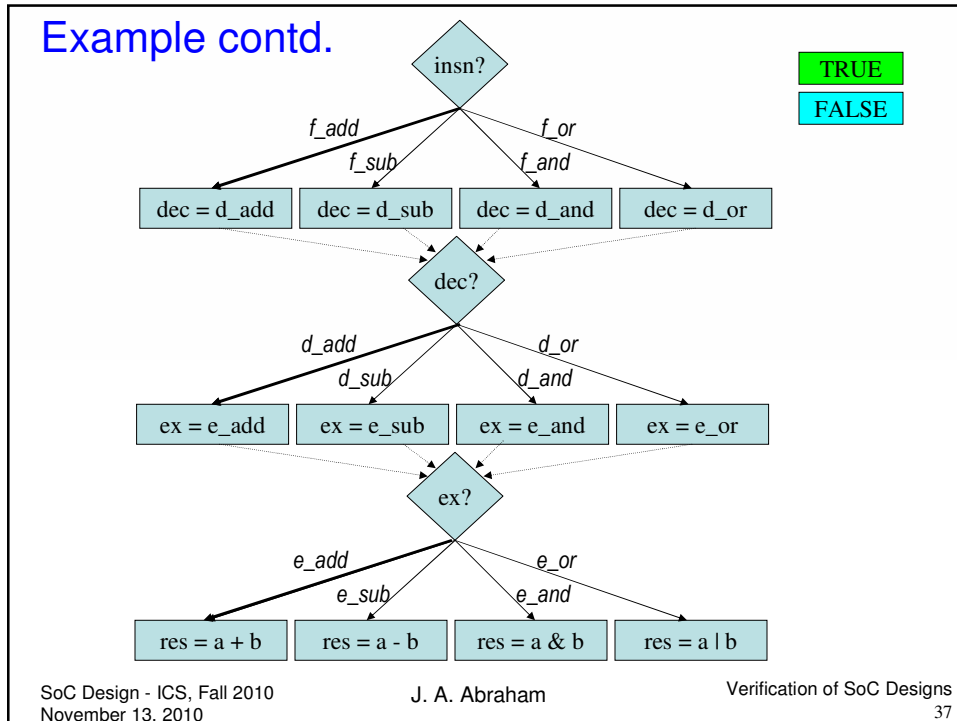
J. A. Abraham

Verification of SoC Designs

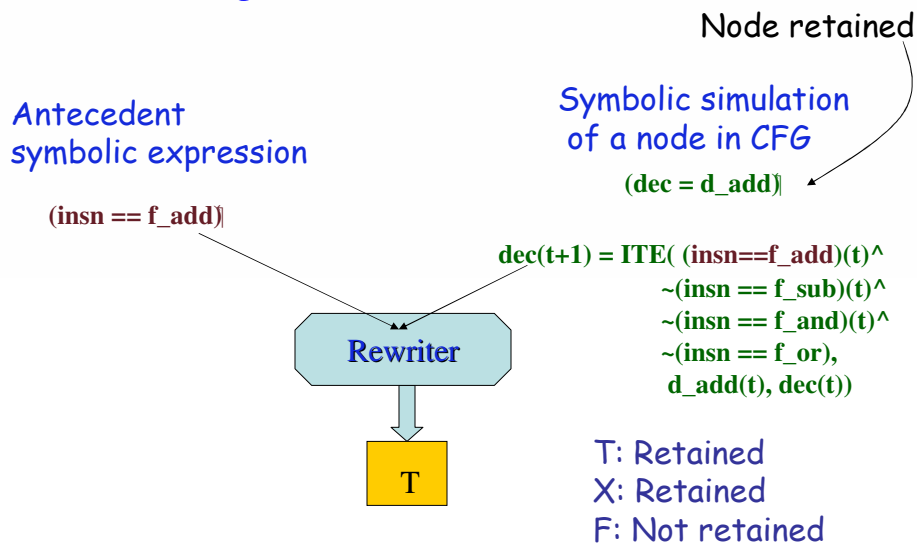
34



Example contd.



Checking the truth of the antecedent



Complexity of Antecedent Conditioned Slicing

- Symbolic simulation of all nodes in each process
- Expression computation over all processes in the program
 - Handles global predicates
- Symbolic simulation of the antecedent
- Looking forward in time
 - Depends on n in $(A \Rightarrow X^n C)$
- Decision procedure for checking truth of antecedent
 - Could be arbitrarily hard
- Path traversal of all processes
 - Pruning non-retained nodes
- **Worst case: retain all nodes**

Correctness of Antecedent Conditioned Slicing

Theorem: An LTL formula h of the type, where h is

$$G(a \Rightarrow c)$$

$$G(a \Rightarrow X^n c)$$

$$G(a \Rightarrow F^{\leq k} c)$$

holds on the original program iff it holds on the antecedent conditioned slice.

Proof intuition:

For a Kripke structure of the slice, all states satisfy $a \Rightarrow c$.

These include states of the original Kripke structure that satisfy a .

Thus all states of the original that satisfy a must satisfy h .

All states of the original that satisfy $\neg a$, satisfy $a \Rightarrow c$ vacuously.

Example of Antecedent Conditioned Slicing

```
always @ (clk)    begin    always @ (clk)    begin    always @ (clk)    begin
  case(insn)      case(dec)      case(ex)
    f_add: dec = d_add;    d_add: ex = e_add; e_add: res = a+b;
  endcase
end                end                end
```

Single instruction behavior for f_add instruction

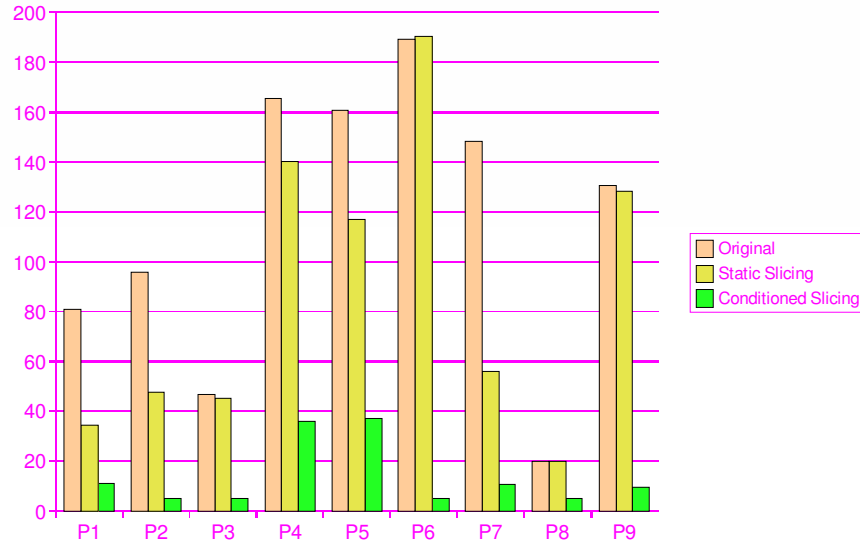
$h = [G((insn == f_add) \Rightarrow XX(res == a+b))]$

Experimental Results

- Verilog RTL implementation of USB 2.0 function core
- Properties taken from specification document
 - Safety properties expressed in LTL
 - Mostly control based, state machine related
- Used Cadence SMV-BMC
 - Circuit too big for SMV
 - Used a bound of 24
- 450 MHz, Ultra Sparc dual processor with 1 GB RAM

Results on USB $G(a \Rightarrow c)$ Properties

CPU Seconds, 450 MHz dual UltraSPARC-II with 1 GB RAM



SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Results of Antecedent Conditioned Slicing

- Temporal property verification for USB 2.0
- Safety properties of the form
 - $G(a \Rightarrow Xc)$
 - $G(a \Rightarrow a \cup_s c)$
- Liveness Properties
 - $G(a \Rightarrow Fc)$
- USB has many interacting state machines
 - Approximately 10^{33} states
- Bound of 50

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Example Properties of the USB

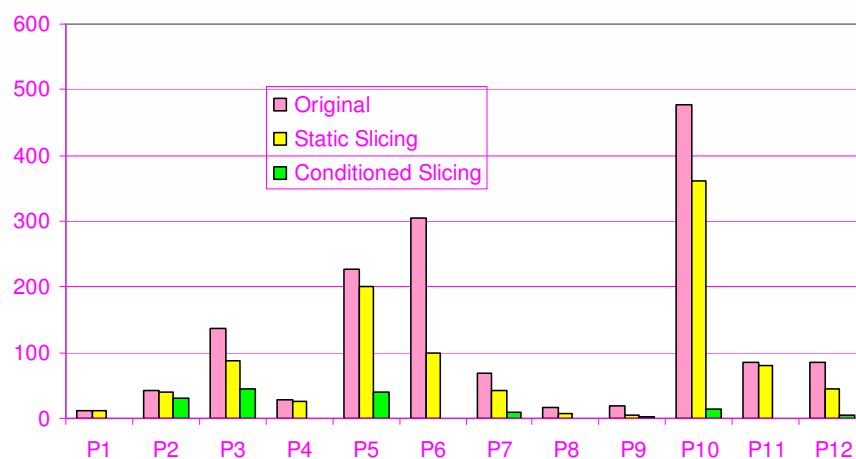
- $G((\text{crc5err}) \vee \neg(\text{match}) \Rightarrow \neg(\text{send_token}))$
 - If a packet with a bad CRC5 is received, or there is an endpoint field mismatch, the token is ignored
- $G((\text{state} == \text{SPEED_NEG_FS}) \Rightarrow X((\text{mode_hs}) \wedge (\text{T1_gt_3_0ms}) \Rightarrow (\text{next_state} == \text{RES_SUSPEND})))$
 - If the machine is in the speed negotiation state, then in the next clock cycle, if it is in high speed mode for more than 3 ms, it will go to the suspend state
- $G((\text{state} == \text{RESUME_WAIT}) \wedge \neg(\text{idle_cnt_clr}) \Rightarrow F(\text{state} == \text{NORMAL}))$
 - If the machine is waiting to resume operation and a counter is set, eventually (after 100 mS) it will return to normal operation

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Results on Temporal USB Properties

CPU Seconds, 450 MHz dual UltraSPARC-II with 1 GB RAM



SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of challenging problems with high level static analysis

Pipelined Processor Verification

- Reason with the entire state of the machine (Burch and Dill)
 - Enhancements use theorem proving techniques
 - Significant manual component
 - Construct complicated invariants
 - High-level model based
- Antecedent conditioned slicing
 - Domain aware analysis
 - Instruction wise case splitting
 - Decision procedure
 - Model checker

Automatic techniques do not
scale to instruction level
verification

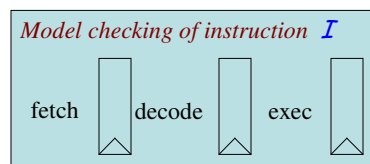
SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
47

Single instruction verification

- Obtain single instruction machine by antecedent conditioned slicing
 - Antecedent is instruction word
- Property is $G(I \Rightarrow R)$ where
 - $I = i_1 \wedge Xi_2 \wedge \dots \wedge Xi_n$
 - i_t represents the antecedent in pipeline stage t
 - R is the result of I in terms of its target register values



SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

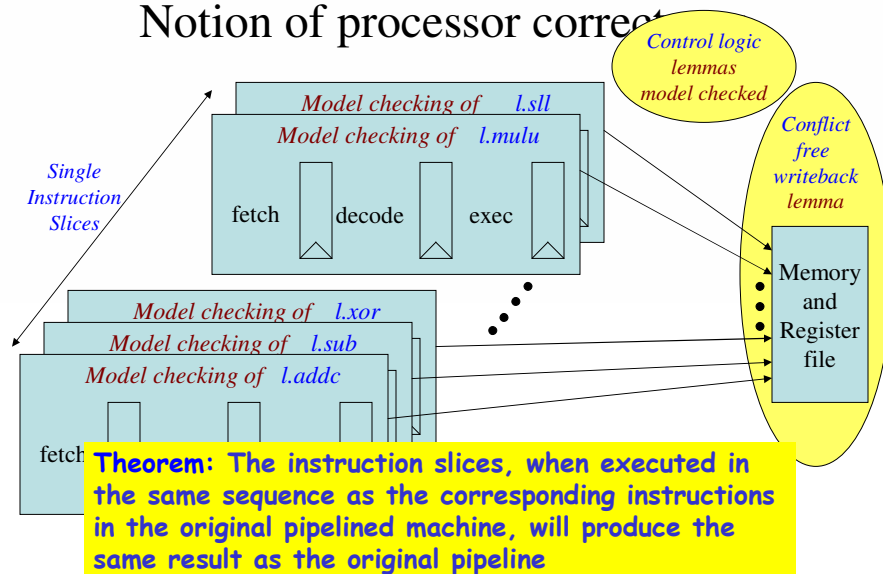
Verification of SoC Designs
48

Interaction between instructions



Lemma: Instructions should write back only to target register only on writeback stage

Notion of processor correctness



Results of OR1200 verification

Class	Insn	SMV Time(s) SLICED	Memory usage (KB)	SMV Time(s) UNSLICED
ALU	<i>l.add</i>	25.65	23796	DNF
ALU	<i>l.sub</i>	24.7	24018	DNF
ALU	<i>l.addi</i>	21.6	19658	DNF
ALU	<i>l.xor</i>	24.84	24831	DNF
ALU	<i>l.and</i>	23.28	21727	DNF
ALU	<i>l.or</i>	24.01	22761	DNF
MAC	<i>l.mul</i>	25.28	49831	DNF
MAC	<i>l.mulu</i>	26.63	22801	DNF
BRANCH	<i>l.bf</i>	132.63	44281	DNF
BRANCH	<i>l.bnf</i>	139.47	46350	DNF
BRANCH	<i>l.j</i>	57.36	31969	DNF
BRANCH	<i>l.jalr</i>	54.09	31073	DNF
BRANCH	<i>l.cmov</i>	159.64	30094	DNF

- OpenRISC 1200
- 32-bit scalar RISC processor
- 5 stage integer pipeline
- Publicly available
- Intended for portable/embedded applications

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
51

Results of OR1200 verification

Class	Insn	SMV Time(s) SLICED	Memory (KB)	SMV Time(s) UNSLICED
COMPARE	<i>l.sfeq</i>	157.29	30004	DNF
COMPARE	<i>l.sfne</i>	183.01	51731	DNF
COMPARE	<i>l.sfgt</i>	194.43	53801	DNF
LSU	<i>l.ld</i>	35.85	63112	DNF
LSU	<i>l.lws</i>	33.91	29104	DNF
LSU	<i>l.sd</i>	38.32	30941	DNF
SHF/ROT	<i>l.sll</i>	26.81	23771	DNF
SHF/ROT	<i>l.srl</i>	27.83	24865	DNF
SHF/ROT	<i>l.ror</i>	27.93	26919	DNF
SPRS	<i>l.mfspr</i>	226.97	50696	DNF
SPRS	<i>l.mtspr</i>	212.27	48627	DNF

- 3GHz Pentium4
- 1GB RAM
- Bolstered use of several Boolean level engines
 - Model checkers, SAT, BDD based engines

All instructions of a pipelined processor were verified

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
52

Verification of challenging problems with high level static analysis

Sequential equivalence checking

- High level symbolic simulation of RTL implementation
- High level symbolic simulation of System level spec
- Domain aware analysis
 - Sequential compare points obtained using heuristics
- Decision procedure
 - SAT solver

SoC Verification

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
53

Term Rewriting for Arithmetic Circuit Checking

- Significant success with RTL Term level reductions
- Verification of arithmetic circuits at the RTL level using *term rewriting*
- **RTL to RTL equivalence checking**
- Verified **large multiplier designs** like Booth, Wallace Tree and many optimized multipliers using this rewriting technique

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Term Rewriting Systems: Example

- Terms: $\text{GCD}(x,y)$
- Rewrite rules:
 - $\text{GCD}(x,y) \rightarrow \text{GCD}(y,x)$ if $x > y, y \neq 0$
 - $\text{GCD}(x,y) \rightarrow \text{GCD}(x,y-x)$ if $x \neq y, y \neq 0$
- Initial term: $\text{GCD}(\text{initX}, \text{initY})$

$$\begin{array}{l} \text{GCD}(6, 15) \xRightarrow{R_2} \text{GCD}(6, 9) \xRightarrow{R_2} \text{GCD}(6, 3) \xRightarrow{R_1} \\ \text{GCD}(3, 6) \xRightarrow{R_2} \text{GCD}(3, 3) \xRightarrow{R_2} \text{GCD}(3, 0) \end{array}$$

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

VERIFIRE

- Dedicated Arithmetic Circuit Checker
- Vtrans: Translates Verilog designs to Term Rewriting Systems
- Vprover: Proves equivalence of Term Rewriting Systems
 - Iterative engine
 - Returns error trace if proof not found
 - Maintains an expanding rule base for expression minimization
 - Incomplete, but efficient engine

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Results on Multipliers

Wallace Tree	VERIFIRE	Commercial Tool 1	Commercial Tool 2
4 X 4	14s	10s	9s
8 X 8	18s	18s	16s
16 X 16	25s	Unfinished	Unfinished
32 X 32	40s	Unfinished	Unfinished
64 X 64	60s	Unfinished	Unfinished

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

System Level Equivalence Checking

- Sequential equivalence checking
 - Verifying two models with different state encodings
- System specifications as system level model (SLM)
 - Higher level of abstraction
 - Timing aware models
- Design concept in RTL needs checking
 - Retiming, power, area modifications
 - Every change requires verification against SLM
- Simulation of SLM
 - Tedious to develop
 - Inordinately long running times

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
58

Equivalence Checking Using Sequential Compare Points

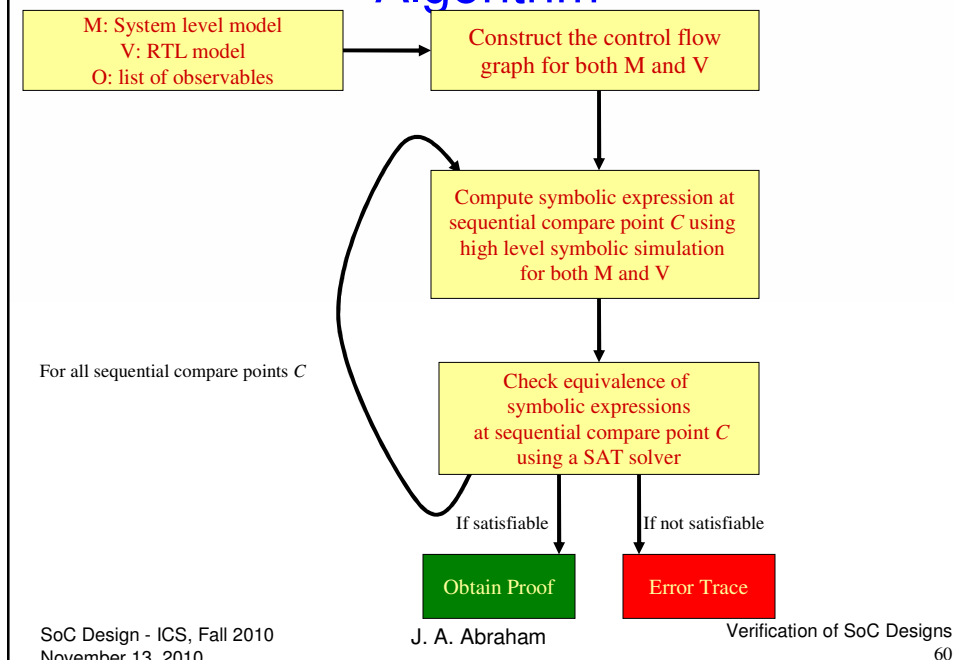
- Variables of interest (observables) obtained from user/block diagram
 - Primary outputs / Relevant intermediate variables
- Symbolic expressions obtained for observables assigned in a given cycle (high level symbolic simulation)
- Introduce notion of sequential compare points
 - Identification with respect to relative position in time
 - Identification with respect to space (data or variables)
- Symbolic expressions compared at sequential compare points
- Comparison using a SAT solver in this work
 - Other Boolean level engines can also be used

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
59

Algorithm



SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
60

Correctness theorem

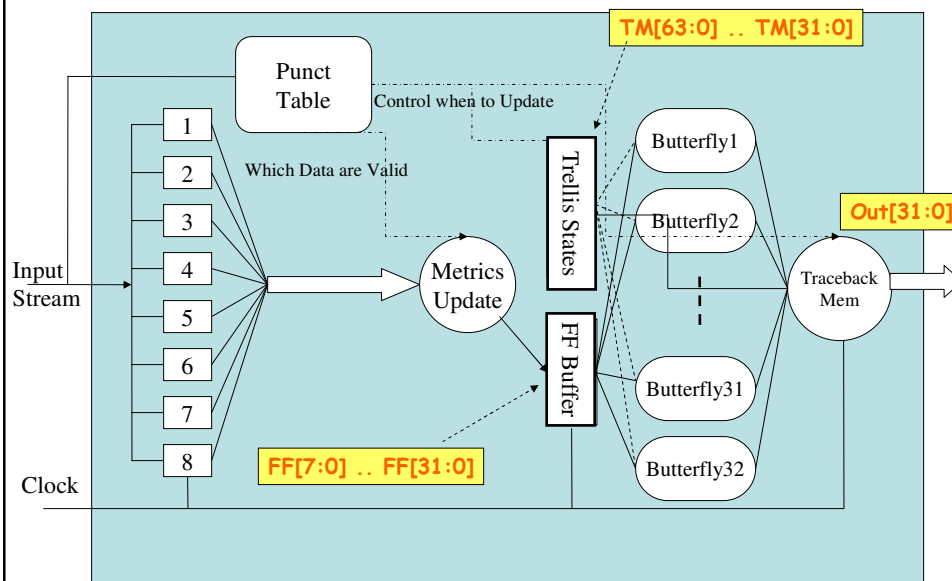
Theorem:

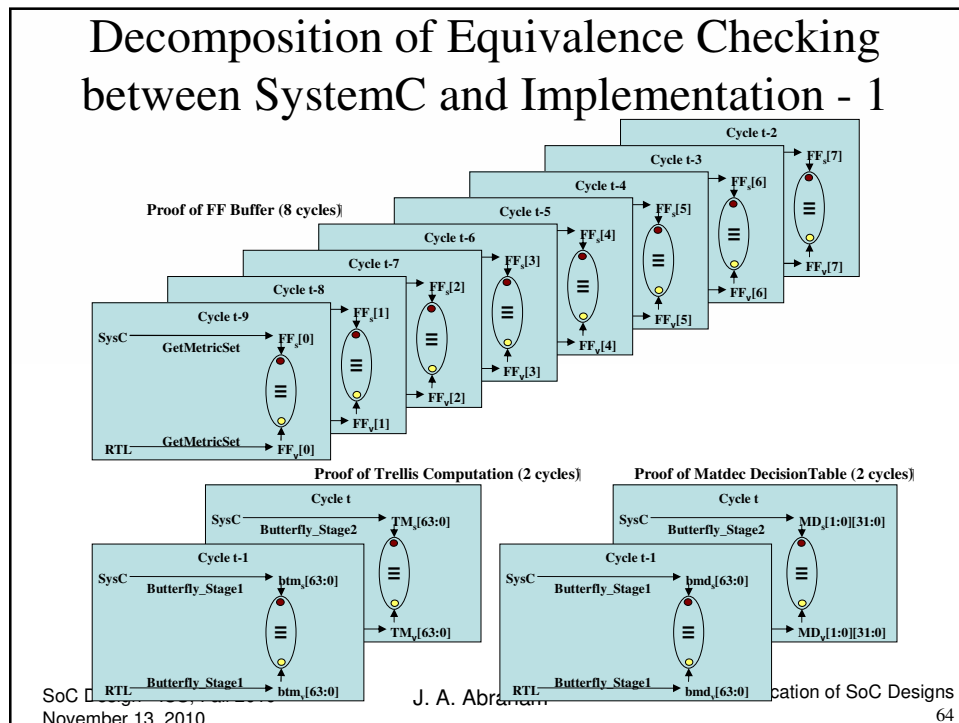
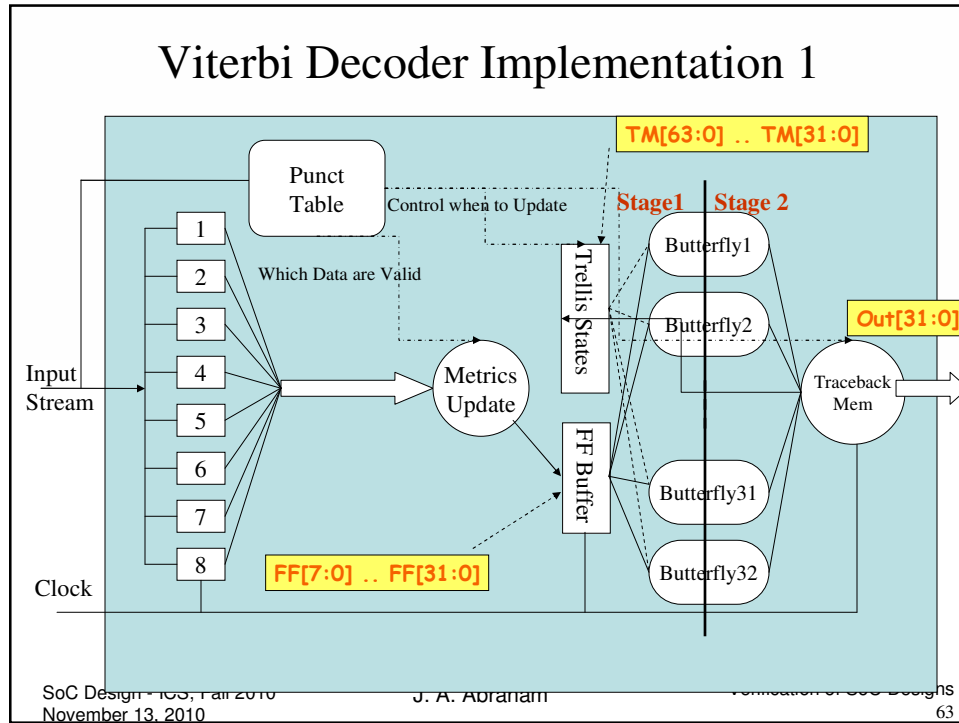
Let two systems M and V such that, $PI(M) = PI(V)$ and $PO(M) = PO(V) = PO$.
Let n be the longest cycle length taken to obtain all primary outputs in both systems.
Let M and V be compared at every point $C = (t, d)$ such that $t \leq n$.
Let \sim_c be the simulation relation that denotes the symbolic expression equality at C .
Then, for all C , $V \sim_c M \Rightarrow V \sim_{PO} M$.

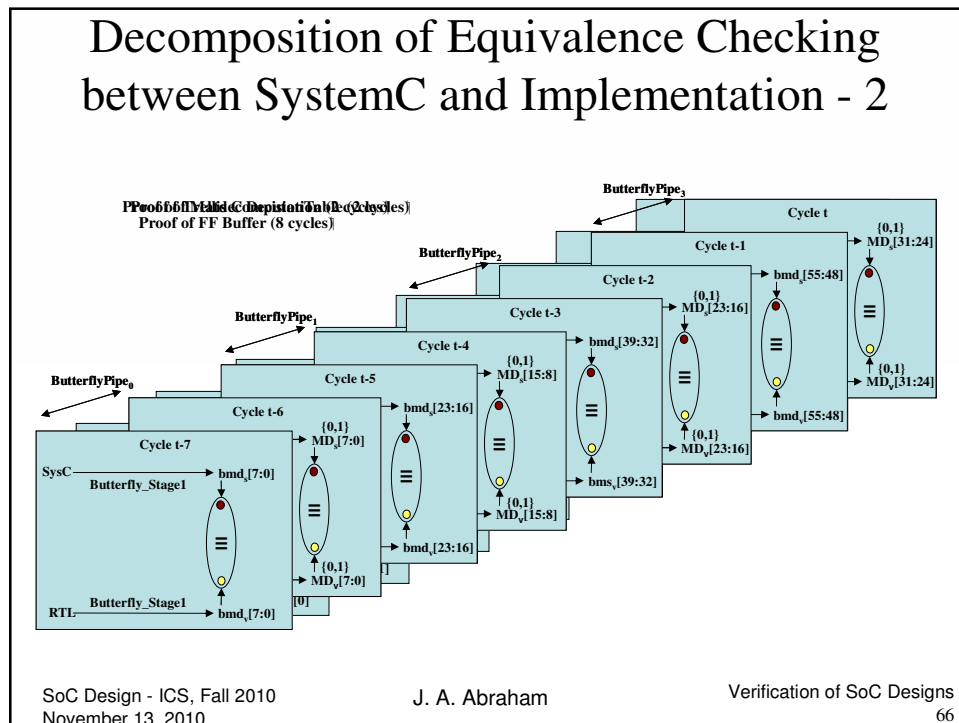
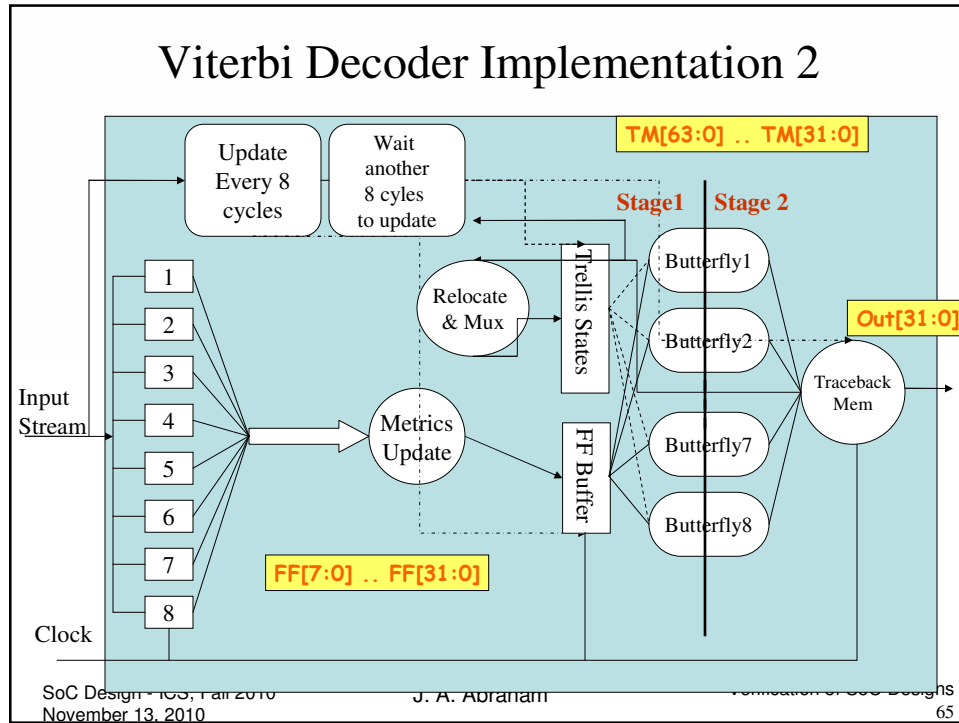
Proof intuition:

The base case is at time $t=0$, at initial state.
The induction hypothesis is relieved using a lemma that proves that
at any cycle t , if the two systems have
the same value for the symbolic expression of all variables d ,
 \sim_c holds at that cycle.
If all primary outputs are generated by cycle n , the relation holds.

Viterbi Decoder: SystemC Specification







Results of using a SAT solver

Block/Function	Number of clauses in the CNF formula	Design	Number of clauses in the CNF formula
PLUS	448	Monolithic Trellis	1892352
LESSTHAN	32	RTL decomposition (Design 1)	59136
Trellis Condition in the butterfly	14336	RTL decomposition (Design 2)	59136
Trellis computation in each stage of butterfly	28672		
Trellis per butterfly	57344		
MatDec each stage of butterfly	896		
MatDec per butterfly	1792		

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
67

Results

Block/Function	Number of variables	Number of symbolic variables generated
PLUS	64	2
Butterfly	128	66
Trellis (monolithic)	2304	2112
Trellis (decomposed)	128	66

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
68

Verifying Embedded Software

- Software Testing
 - Execute software for test cases
 - Analogous to simulation in hardware
- Testing Criteria
 - Coverage measures
- Formal analysis of software
 - Model Checking
 - Theorem Proving

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
69

Path Testing

- Assumption: bugs affect the control flow
- Execute all possible control flow paths through the program
 - Attempt 100% path coverage
- Execute all statements in program at least once
 - 100% statement coverage
- Exercise every branch alternative during test
 - Attempt 100% branch coverage

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
70

Software Verification

- Formal analysis of code
- Result, if obtained, is guaranteed for all possible inputs and all possible states
- Example of software model checker:
SPIN
- Problem: applicable only to small modules
) State Explosion

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
71

Data Abstractions

- Abstract data information
 - Typically manual abstractions
- Infinite behavior of system abstracted
 - Each variable replaced by abstract domain variable
 - Each operation replaced by abstract domain operation
- Data independent Systems
 - Data values do not affect computation
 - Datapath entirely abstracted

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
72

Data Abstractions: Examples

- Arithmetic operations
 - Congruence modulo an integer
 - k replaced by $k \bmod m$
- High orders of magnitude
 - Logarithmic values instead of actual data value
- Bitwise logical operations
 - Large bit vector to single bit value
 - Parity generator
- Cumbersome enumeration of data values
 - Symbolic values of data

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
73

Abstract Interpretation

- Abstraction function mapping concrete domain values to abstract domain values
- Over-approximation of program behavior
 - Every execution corresponds to abstract execution
- Abstract semantics constructed once, manually

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
74

Abstract Interpretation: Examples

- Sign abstraction
 - Replace integers by their sign
 - *Each integer K replaced by one of $\{> 0, < 0, =0\}$*
- Interval Abstraction
 - Approximates integers by maximal and minimal values
 - *Counter variable i replaced by lower and upper limits of loop*
- Relational Abstraction
 - Retain relationship between sets of data values
 - *Set of integers replaced by their convex hull*

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
75

Counterexample Guided Refinement

- Approximation on set of states
 - Initial state to bad path
- Successive refinement of approximation
 - Forward or backward passes
- Process repeated until fixpoint is reached
 - Empty resulting set of states implies property proved
 - Otherwise, counterexample is found
- *Counterexample can be spurious because of over-approximations*
- Heuristics used to determine spuriousness of counterexamples

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
76

Counterexample Guided Refinement

- Predicate Abstraction
 - Predicates related to property being verified (User defined)
 - Theorem provers compute the abstract program
 - Spurious counterexamples determined by symbolic algorithms
 - Some techniques use error traces to identify relevant predicates

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
77

Counterexample Guided Refinement

- Lazy Abstraction
 - More efficient algorithm
 - Abstraction is done on-the-fly
 - Minimal information necessary to validate a property is maintained
 - Abstract state where counterexample fails is “pivot state”
 - Refinement is done only “from the pivot state on”

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
78

Specialized Slicing for Verification

- Amorphous Slicing
 - Static slicing preserves syntax of program
 - Amorphous Slicing does not follow syntax preservation
 - Semantic property of the slice is retained
 - Uses rewriting rules for program transformation

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
79

Example of Amorphous Slicing

```
begin
  i = start;
  while (i <= (start + num))
  {
    result = K + f(i);
    sum = sum + result;
    i = i + 1;
  }
end
```

LTL Property: $G \text{ sum} > K$
Slicing Criterion: $(\text{end}, \{\text{sum}, K\})$

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
80

Example of Amorphous Slicing

Amorphous Slice:

```
begin
  sum = sum + K + f(start);
  sum = sum + K + f(start + num);
end
```

Program Transformation rules applied

- Induction variable elimination
- Dependent assignment removal
- Amorphous Slice takes a fraction of the time as the real slice on SPIN

Amorphous Slicing for Verification

- Similar to term rewriting
 - Used by theorem provers for deductive verification
- What is different?
 - Theorem provers try to prove entirely by rewriting
 - Hybrid approach
 - Rewriting only part of the program, based on slicing criterion
 - Model checking the sliced program

Conditioned Slicing

- Theoretical bridge between static and dynamic slicing
- Conditioned Slices specify initial state in criterion
 - Constructed with respect to set of possible inputs
 - Characterized by first order predicate formula
- Yields much smaller slices than static slices

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
83

Example Results – Conditioned Slicing

- Group Address Registration Protocol (GARP) and X.509 authentication protocol
- SPIN model checker
 - Memory limit of 512 MB given
 - Max search depth of 2^{20} steps
- All properties were in the form
Antecedent \Rightarrow Consequent

SoC Design - ICS, Fall 2010
November 13, 2010

J. A. Abraham

Verification of SoC Designs
84

Experimental Results

Property	Unsliced*	Conditioned Sliced	Property Proved
P1	91.65	1.72	Yes
P2	145.78	8.44	Yes
P3	145.36	8.41	Yes
P4	154.96	1.95	Yes
P5	117.81	10.23	Yes

*Static slicing in SPIN was enabled