14. Test Generation Algorithms and Emulation for Verification

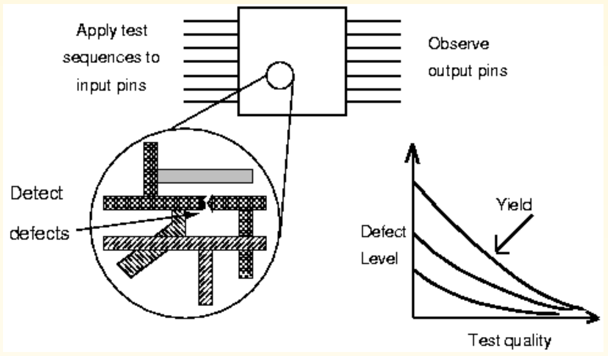
Jacob Abraham

Department of Electrical and Computer Engineering
The University of Texas at Austin

Verification of Digital Systems
Spring 2020

March 5, 2020

Manufacturing Test



- Manufacturing process may result in physical defects or parameter variations
- Tests for defects and variations – output will be different from the case when there is no defect

Process of searching for defects is similar to searching for bugs

Tests for Faults in a Circuit

- Approach to generating tests for defects is to map defects to (higher level) faults: develop fault model, then generate tests for the faults
  - Typical for defects: gate-level "stuck-at" fault model
  - For bugs: Line coverage, control state coverage, "coverage points", etc.
- As technology shrinks, other physical faults: bridging faults, delay faults, crosstalk faults, etc.
  - Some electrical faults (due to design or manufacturing) are only seen in post-silicon validation
- An interesting point: what is important is how well the tests generated (based on the fault model) will detect realistic defects or bugs
  - The accuracy of the fault model is secondary

Automatic Test Pattern Generation (ATPG)

- Map defects to (higher level) faults, and develop fault models (example logic-level "stuck-at" faults, or "path delay" faults)

Steps in Test Generation

- **Activate** fault (produce error at fault site)
- **"Sensitize"** path from fault to output (propagate error to output)
- **"Justify"** internal signals to primary inputs
- Choices may exist during sensitization and justification: if conflicts arise, need to **backtrack**
- If no test exists, fault is **redundant**

SAT solvers have been applied to generating tests for combinational blocks

Propagation of the error is not necessary for verification (any node can be monitored in simulation)

ATPG Complexity

Problem of generating a test for a stuck-at-fault in a combinational circuit is NP-Complete

Satisfiability is also NP-Complete

A lot of interesting problems belong to the class of NP-Complete problems

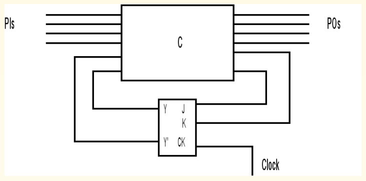
- Tovey, Craig A, “Tutorial on computational complexity,”, *INFORMS Journal on Applied Analytics*, vol. 32, pp. 30–61, 2002.
- Classic paper by Karp: Karp R .M., “Reducibility among Combinatorial Problems,” *In: Miller R. E., Thatcher J.W., Bohlinger J.D. (eds) Complexity of Computer Computations,* The IBM Research Symposia Series, 1972, Springer, Boston, MA.

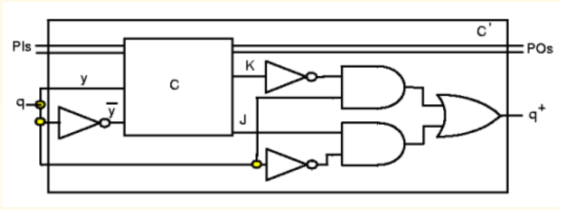
Observability and Controllability

- **Observability**: ease of observing a value on a node by monitoring external output pins of the chip
- **Controllability**: ease of forcing a node to 0 or 1 by driving input pins of the chip
- Combinational logic is usually easier to observe and control
  - Still, NP-complete problem
- Finite state machines can be very difficult, requiring many cycles to enter desired state
  - Especially if state transition diagram is not known to the test engineer, or is too large

Sequential Circuit Test Generation

"Unroll" sequential circuit into an iterative logic array model for one "time frame"

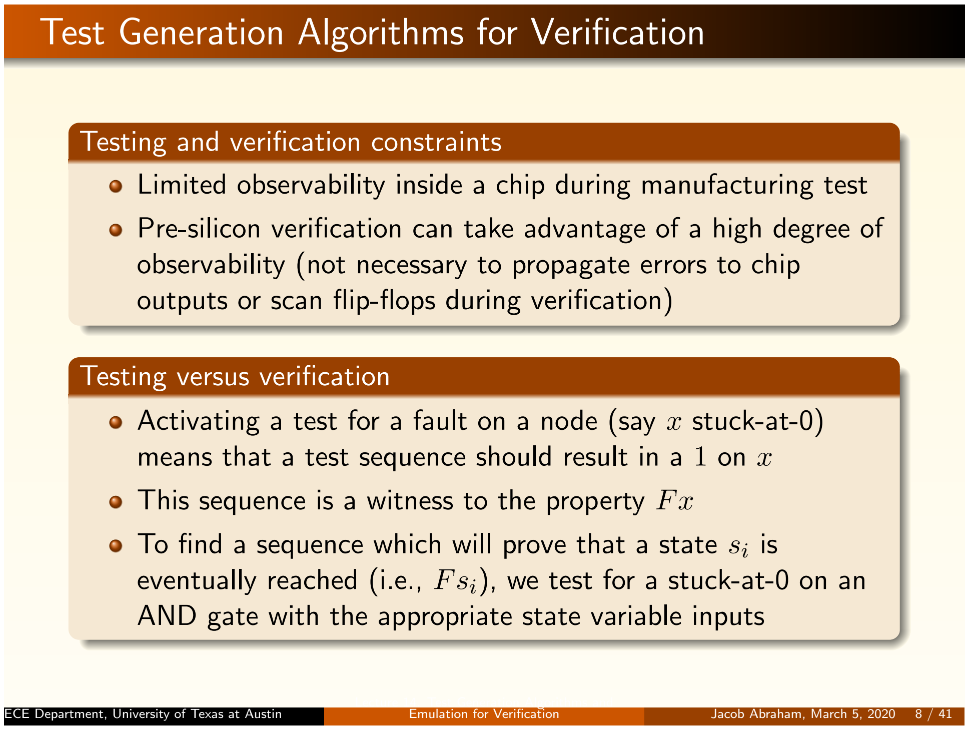




- Single stuck fault in circuit appears in every time frame (may affect propagation of errors)
- Usually assume that the single clock is fault-free
- In general, no prior knowledge of the number of time frames needed for propagation and justification

Appropriate Level to Verify Hardware

- Key requirement for designers: verification approach should fit into the design flow
- Verifying the high-level design: no guarantee that refinements will not introduce errors
- Abstractions to reduce complexity: may mask bugs
- Verify design at the lowest level possible: example, ATPG level
  - Can then deal with tri-states, multiple clocks, etc.



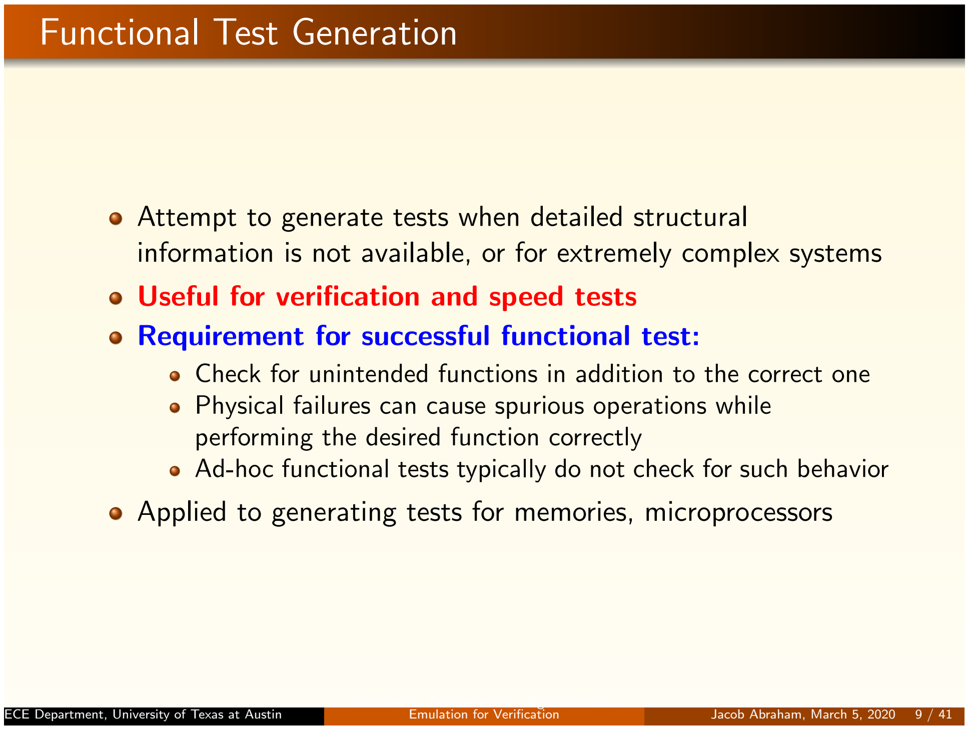
Test Generation Algorithms for Verification

Testing and verification constraints

- Limited observability inside a chip during manufacturing test
- Pre-silicon verification can take advantage of a high degree of observability (not necessary to propagate errors to chip outputs or scan flip-flops during verification)

Testing versus verification

- Activating a test for a fault on a node (say x stuck-at-0) means that a test sequence should result in a 1 on x
- This sequence is a witness to the property Fx
- To find a sequence which will prove that a state s_i is eventually reached (i.e., Fs_i), we test for a stuck-at-0 on an AND gate with the appropriate state variable inputs



Functional Test Generation

- Attempt to generate tests when detailed structural information is not available, or for extremely complex systems
- **Useful for verification and speed tests**
- **Requirement for successful functional test:**
  - Check for unintended functions in addition to the correct one
  - Physical failures can cause spurious operations while performing the desired function correctly
  - Ad-hoc functional tests typically do not check for such behavior
- Applied to generating tests for memories, microprocessors

Functional Fault Model for Memories

Memory Arrays

- One or more cells become stuck at 1 or 0
- One or more cells fail to undergo a 0 to 1 or a 1 to 0 transition
- There exist two (or more) cells which are **coupled**
  - A 0 to 1 (or a 1 to 0) transition in a cell (due to a write in that cell) changes the contents of another cell from 0 to 1 or from 1 to 0
  - If transition in cell I changes J, transition in J may not affect the state of cell I
- Multiple cells accessed during READ or WRITE

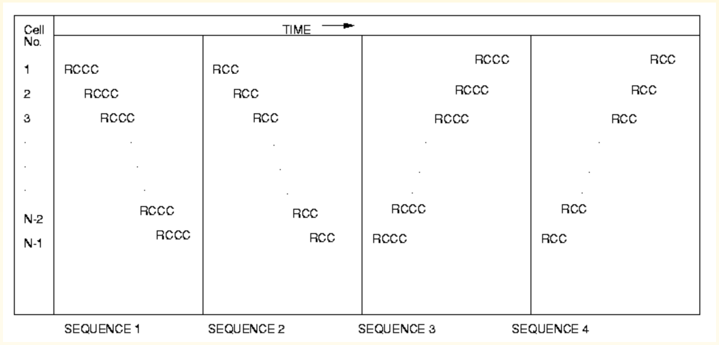
Memory Fault Model, Cont'd

Decoders

- The decoder will not access the addressed cell, and in addition may access non-addressed cells
- The decoder will access multiple cells, including the addressed cell

- Assumption that the combinational logic of the decoder will not be transformed into sequential logic
- **Decoder faults look like memory cell array faults**
- Fault model can be validated by simulating effects of faults at the transistor level

Example "March" Test for Memories



R: Read cell and verify
C: Complement cell

Complexity of Test: 14N (N cells)

This test will verify that writing any location will not disturb any other location

Functional Testing of Microprocessors

- Developed in the 80s for generating tests based on information about the instruction set
  - Tests for vendor parts without knowing details
- Tests based on **"functional fault models"** derived from analysis of the effects of low level faults on the behavior of modules
  - Based on functional tests for memories
- Fault models at control sequencing level
- Tests based on high-level information can also be used for **validating design correctness**

Data Storage and Transfer Tests

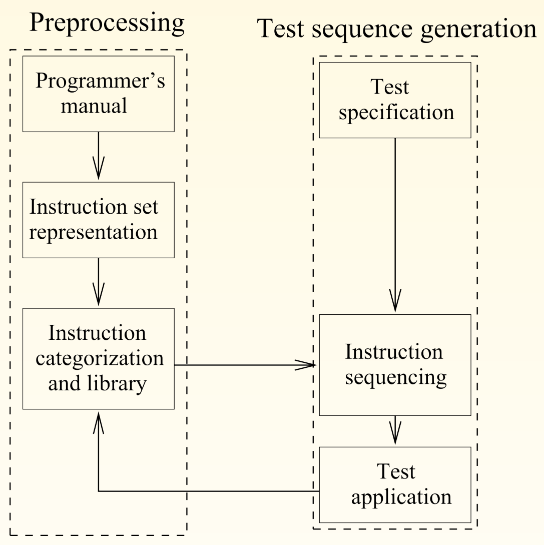
```
00000000000000000000000000000000
11111111111111111111111111111111

00000000000000001111111111111111
11111111111111110000000000000000

00000000111111110000000011111111
11111111000000001111111100000000

00001111000011110000111100001111
11110000111100001111000011110000

00110011001100110011001100110011
11001100110011001100110011001100

01010101010101010101010101010101
10101010101010101010101010101010
```

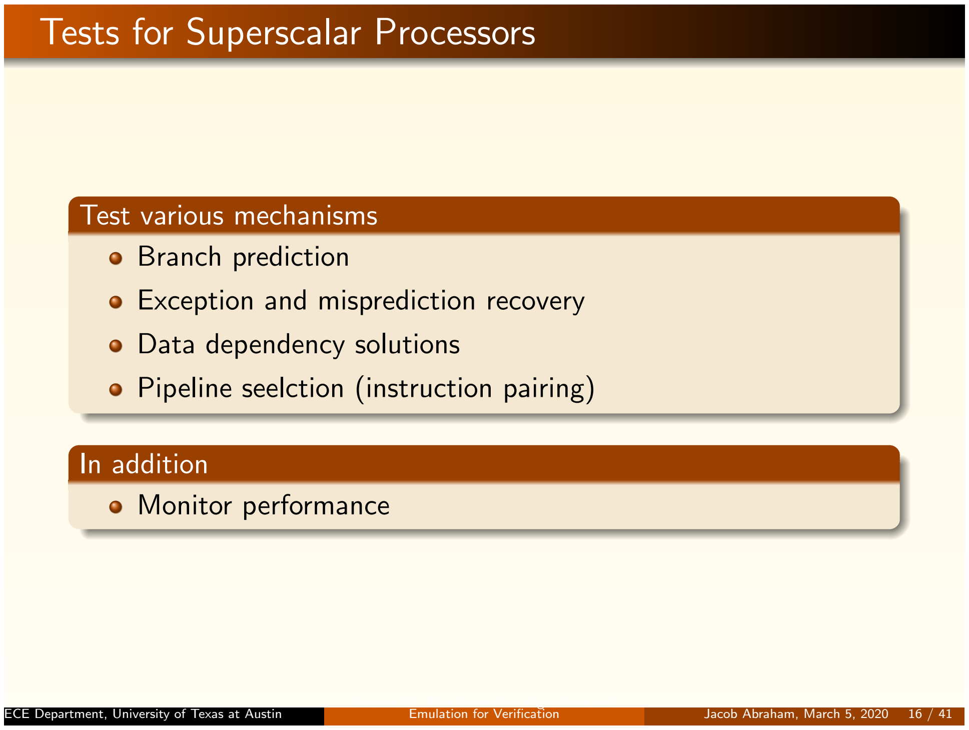
Data patterns to check all logical paths and all registers (32-bit transfer path)

- Test length logarithmic in number of bits (divide and conquer)
- Will such tests verify that there is no interaction between lines in a data path?

Automatic Functional Test Generation for an ISA

Shen, 1998–99





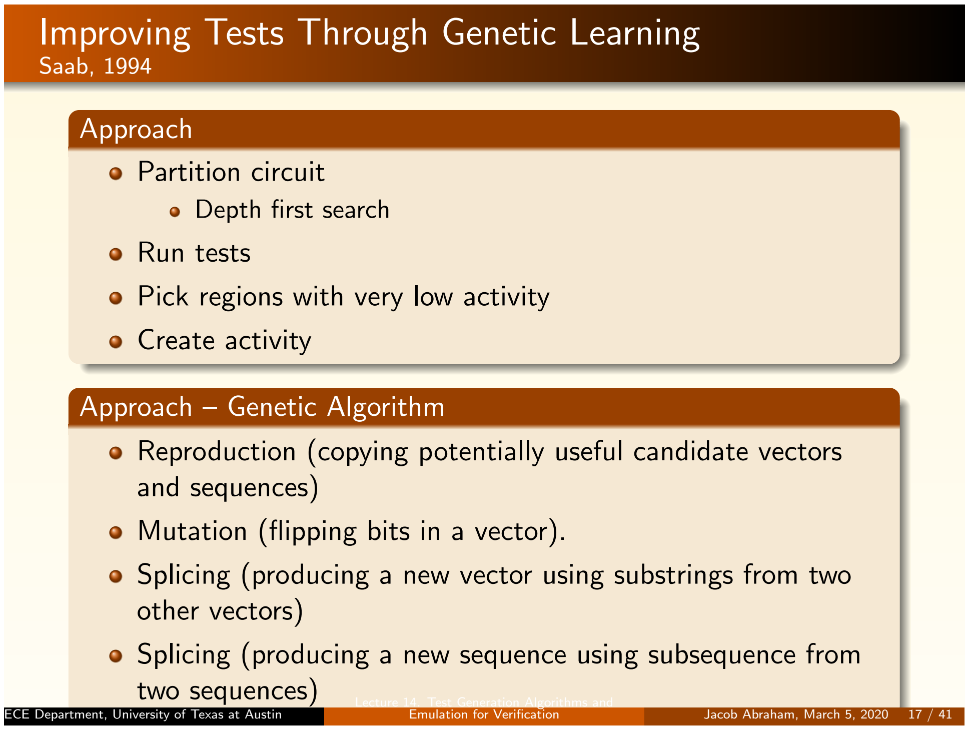
Tests for Superscalar Processors

Test various mechanisms

- Branch prediction
- Exception and misprediction recovery
- Data dependency solutions
- Pipeline seelction (instruction pairing)

In addition

- Monitor performance



Improving Tests Through Genetic Learning

Saab, 1994

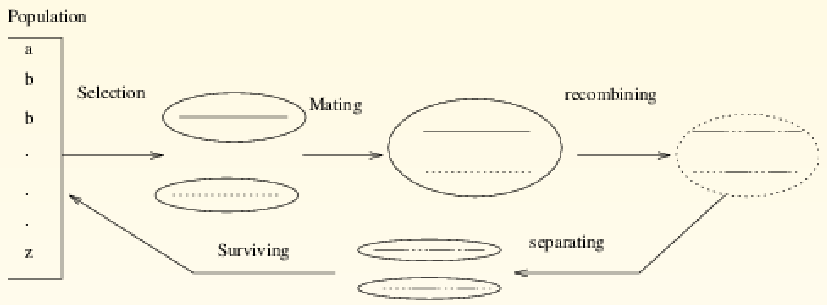
Approach

- Partition circuit
  - Depth first search
- Run tests
- Pick regions with very low activity
- Create activity

Approach – Genetic Algorithm

- Reproduction (copying potentially useful candidate vectors and sequences)
- Mutation (flipping bits in a vector).
- Splicing (producing a new vector using substrings from two other vectors)
- Splicing (producing a new sequence using subsequence from two sequences)

Genetic Algorithm

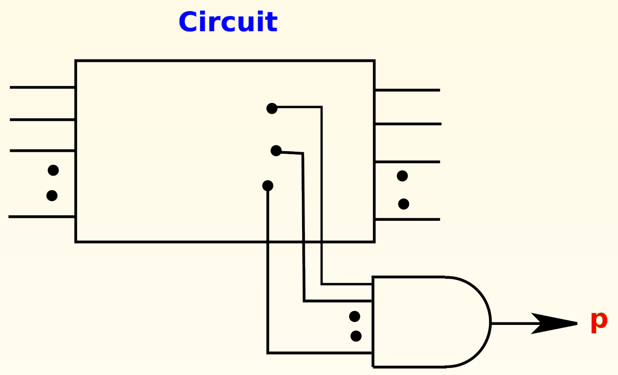


Found to be fast and efficient

- Can be used for very large circuits
- Applicable to different levels – RTL, gate, transistor

Verifying Properties Using Sequential ATPG

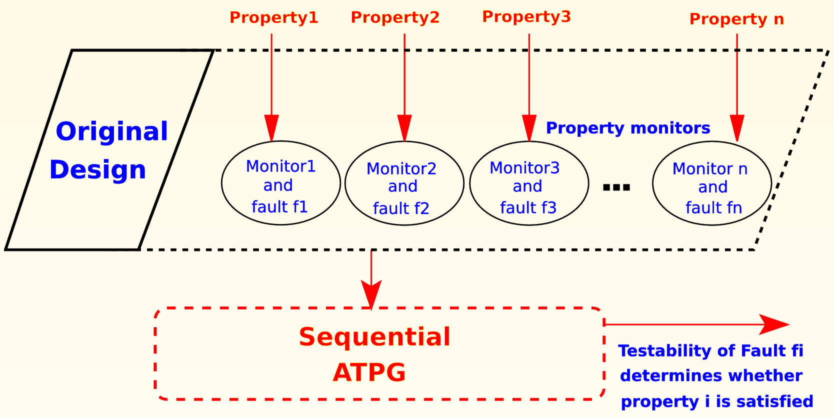
Prior work in checking *safety properties*; required custom ATPG or modifications to existing ATPG tools



Detecting a "stuck-at-0" fault on **p** is equivalent to proving EFp

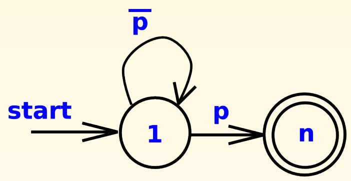
Approach to Property Checking

Abraham, Vedula and Saab, 2002



Bounded Model Checking

Monitor State Machine for EXp



The monitor machine moves to an accepting state if p is true

This is combined with the design, and the ATPG tool asked to find an input sequence to reach the state "n"

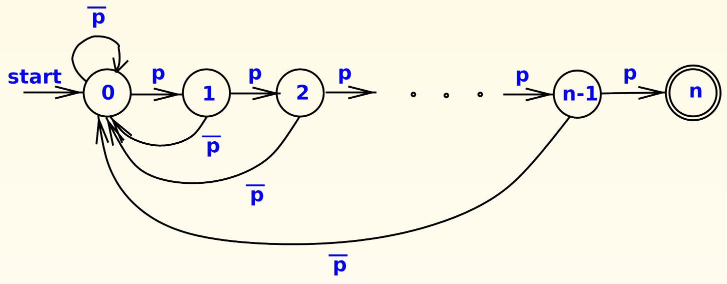
The result would be one of

- ATPG finds an input sequence – EXp is proved, and the sequence would be a *witness* to the property
- ATPG returns the result that a "test" is not possible – EXp is false
- ATPG *aborts* – the design was too complex to be analyzed

Checking EGp

Find an input sequence of length n for which the system will satisfy the property p

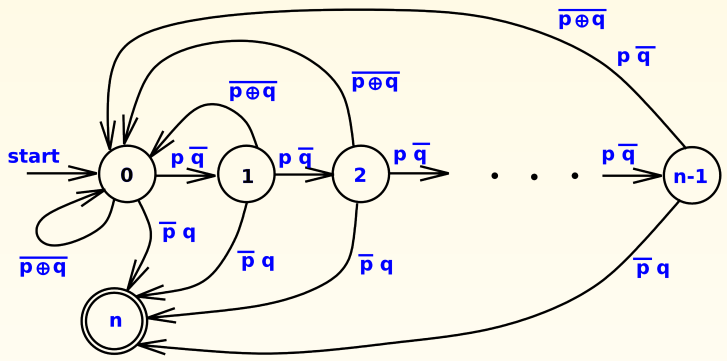
Bounded Liveness



If there is a loop back from state n to the starting state, we can deduce general liveness

Checking E pUq

For some path of up to n cycles, there is a state where q holds, and p holds in every previous state



Results on ISCAS89 Benchmark Circuits

- ATPG: commercial tool (Mentor *flextest*)
- BMC: Cadence research tool (SMV) with *zchaff* SAT solver
- s838.1 – 36 inputs, 1 output, 446 gates, 32 flops
- Property: output is 1 for a sequence of n clocks (n=5, 10, 15)
  - Result: true for n = 5, 10 (false for n = 15)

CPU seconds to check property
(SUN UltraSparc, dual 450MHz, 1 GByte)

Bound: 5		Bound: 10		Bound: 15	
BMC	ATPG	BMC	ATPG	BMC	ATPG
1.57	0.1	2.0	0.2	2.88	0.3

Checking Properties of GL85

- Clone of Intel 8085 designed by Alex Miczo (not pin-compatible)
- 10,084 gates, 238 flip-flops
- Properties (dealing with system reset)
  - ROIA: Reset on Interrupt Acknowledge
  - RORW: Reset on Read-Write
  - ROTF: Reset on Tstates Flow
  - ROIE: Reset on Interrupt Enable
  - TOPE: Trap on Priority Encoding
  - RWIO: Reset While Interrupt On

Example:
$G((TRAP = 1 \ \& \ TRAPFF = 1) \implies (P5/PIE(2:0) = 000B))$

Time for Checking GL85 Properties

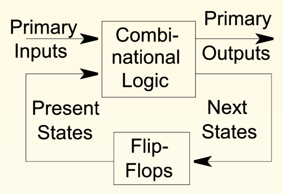
CPU seconds on SUN UltraSparc, dual 450MHz, 1 GByte

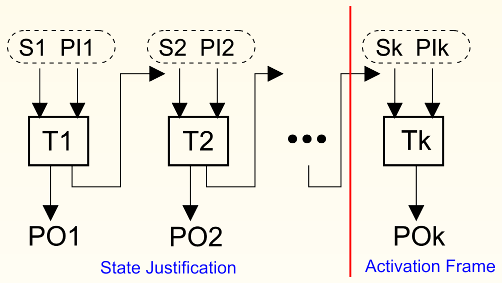
Bound = 25

Property	BMC	ATPG
RORW	7209	12.1
ROTF	4373	12.4
ROIA	6589	12.8
ROIE	7072	13.8
TOPE	10156	13.2
RWIO	6669	12.3

ATPG State Justification

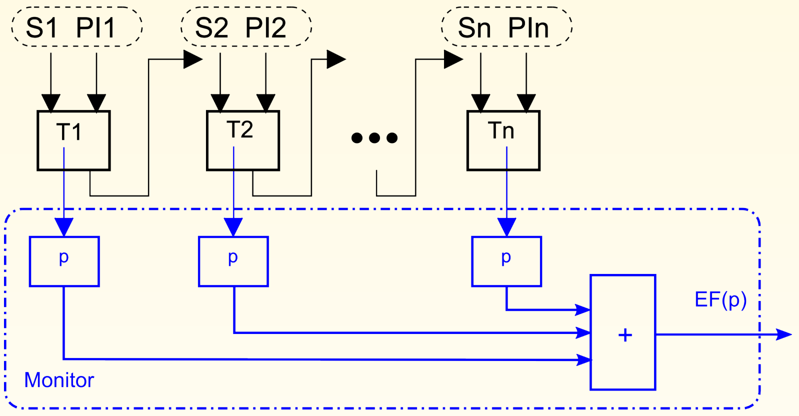
- ATPG activates the monitor faults in the last time frame
- Justify the activation state from unknown on known initial state





ATPG Model for EF(p)

EF(p) is valid if fault at line EF(p) stuck-at-1 is redundant



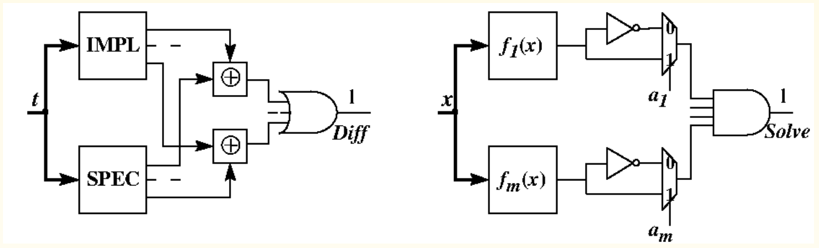
The fault is at the primary output; only fault-free values need to be justified

Satisfiability Solvers

(Abramovici and Saab, 2007)

SAT Solvers

- SAT solving is fundamental for many problems, including equivalence checking, testing, property checking, etc.
- SAT belong to the class of NP-complete problems



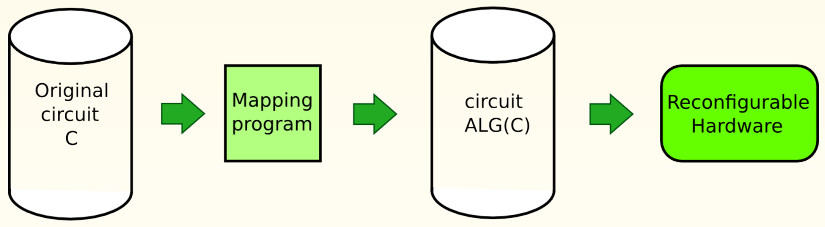
Circuit comparison using SAT

Solving system of Boolean equations using SAT

Satisfiability on Reconfigurable Hardware

Speeding up SAT solving

- Speed up an algorithm ALG working on a circuit C
- Map C to a new circuit ALG(C) which executes ALG for C
- Use reconfigurable hardware to "virtually" create the ALG(C) circuit
- Unlike a hardware accelerator, the hardware is specific to a single circuit



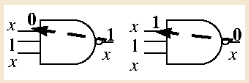
Circuit for SAT search

Objective

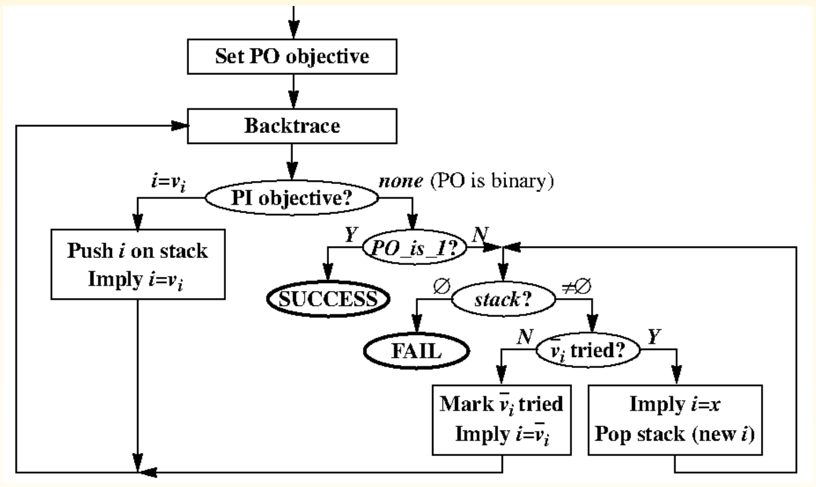
- Search based on a test generation algorithm, PODEM
- Uses circuit structure to speed up process
- Objective: desired assignment of a value to a line in the circuit
- Initially, all lines set to 'x', primary output (PO) to 1
- Objective achieved only by assignments to primary inputs (PIs)

Backtracing

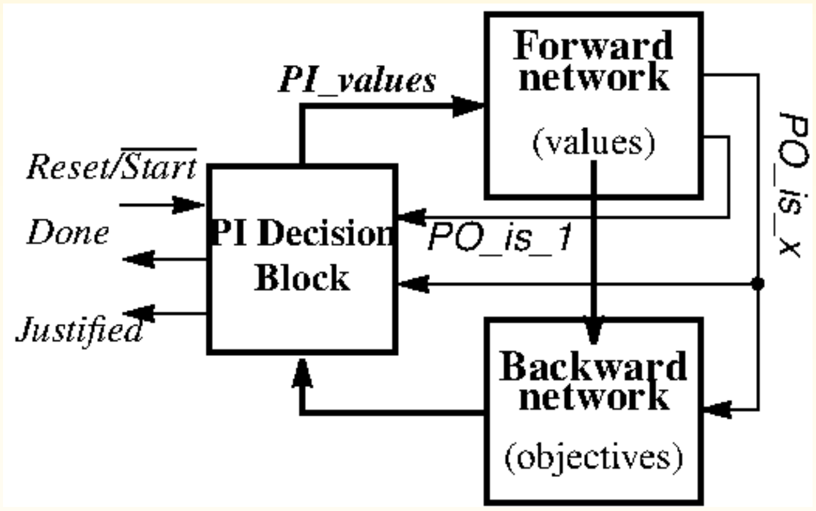
- Backtrace procedure propagates an objective along a single path from a line to a PI



Flowchart of Search Algorithm



Architecture of SAT Circuit



Results

Circuit	Gates	PIs/POs	SAT Gates	Size Incr.	SAT?	SAT Clocks	CPU (sec)	Speedup
C432A	160	36/7	2,285	14.3	Y	5	0.1	20,000
C499A	202	41/32	3,003	14.9	Y	49	0.1	2,041
C880A	383	60/26	4,137	10.8	N	21	0.2	9,524
C1355A	546	41/32	5,231	9.6	Y	476,676	226.0	474
C1908A	880	33/25	6,706	7.6	Y	5,021	2.0	398
C2670A	1,193	233/140	13,180	11.0	N	180,606	43.0	238
C3540A	1,669	50/22	12,365	7.4	N	132,204	188.9	1,429
C5315A	2,307	178/123	21,276	9.2	N	252	0.7	2,778
C6288A	2,416	32/32	22,174	9.2	N	1,601,943	2,782.6	1,737
C7552A	3,512	207/108	28,277	8.1	N	10,824	8.5	785

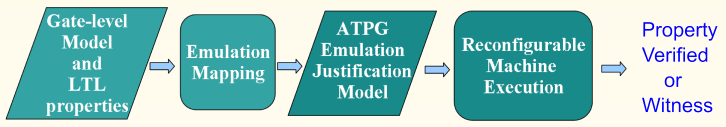
Benchmark outputs were ANDed together to produce a single output

CPU for software SAT was a 110 MHz processor, 1 MHz clock assumed for FPGA

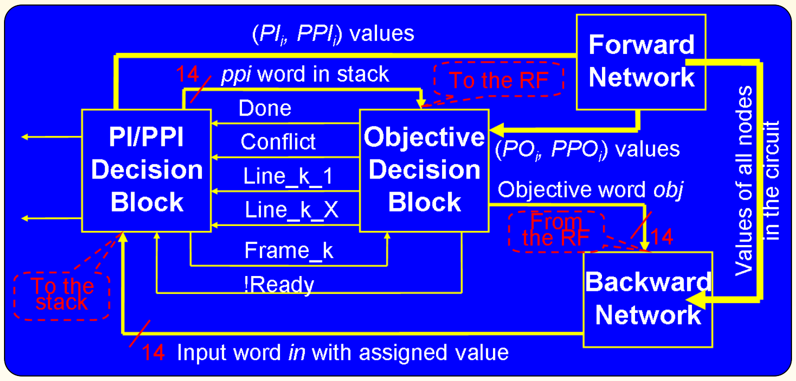
Emulation Model for Bounded Model Checking Using Sequential ATPG

Qiang et al., 2005

- Input consists of gate-level circuit and set of properties
- Develop an emulation model that verifies the property
  - ATPG Justification part specialized for circuit and properties


Gate-level Model and LTL properties
Emulation Mapping
ATPG Emulation Justification Model
Reconfigurable Machine Execution
Property Verified or Witness

Emulation Architecture

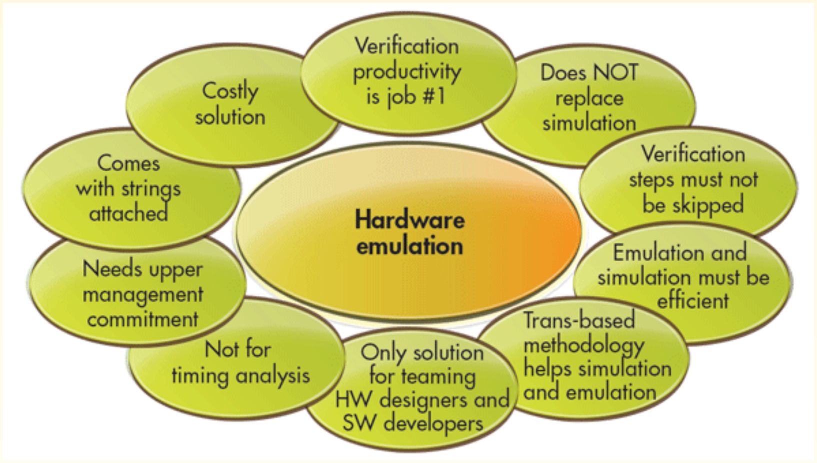


Model Sizes of ISCAS89 Circuits

	Original Model				ATPG Model	
Circuit	PIs	POs	PPIs	Gates	Gates	Increase
s1238	14	14	18	508	16011	30.5
s1423	17	5	74	657	17690	25.9
s1488	8	19	6	653	16327	24.0
s1494	8	19	6	647	16365	24.3
s5378	35	49	179	2779	30862	10.1
s9234	36	39	211	5597	42430	6.6
s13207	62	152	638	7951	63437	7.0
s15850	77	150	534	9772	67855	5.9
s35932	35	320	1728	16065	162415	9.1
s38417	28	106	1636	22179	143977	5.5
s38584	38	304	1426	19253	159249	7.3

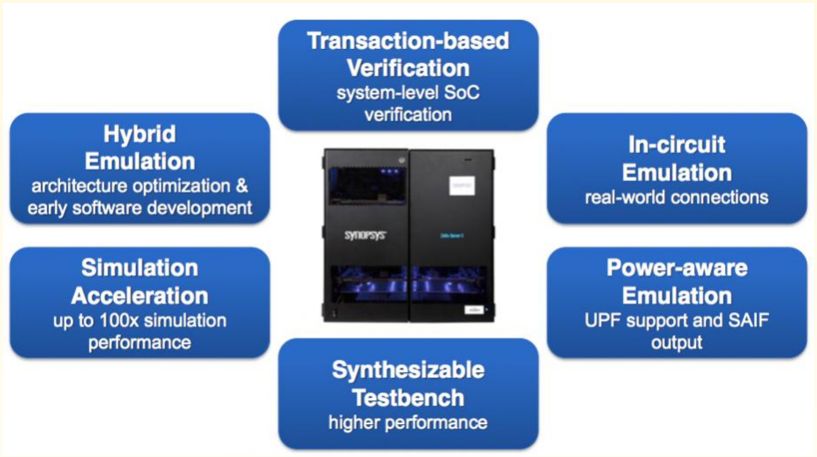
2 – 3 orders of magnitude speedup over software by using emulation hardware

When Should Emulation be Used for Verification?

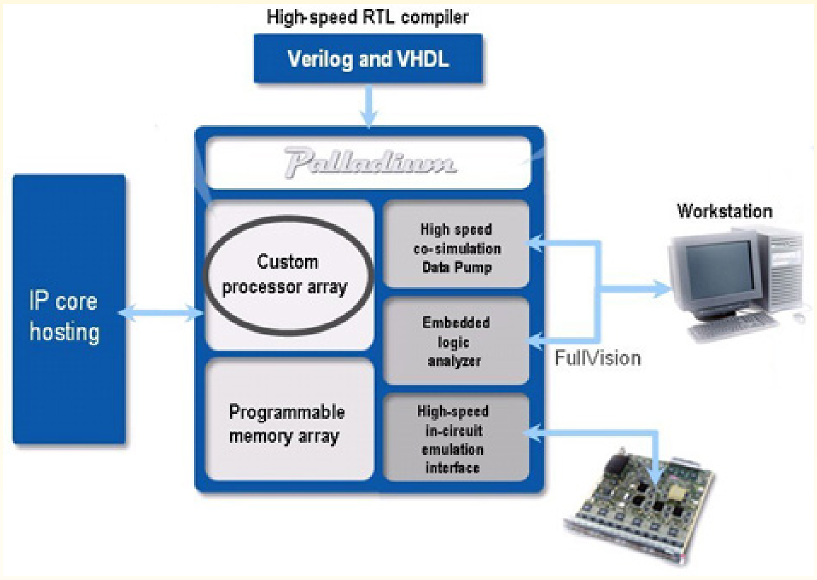

Hardware emulation
Costly solution
Verification productivity is job #1
Does NOT replace simulation
Verification steps must not be skipped
Emulation and simulation must be efficient
Trans-based methodology helps simulation and emulation
Only solution for teaming HW designers and SW developers
Not for timing analysis
Needs upper management commitment
Comes with strings attached

Source: L. Rizzatti, "10 Best Verification Practices for Hardware Emulation", *Electronic Design*, June 29, 2016

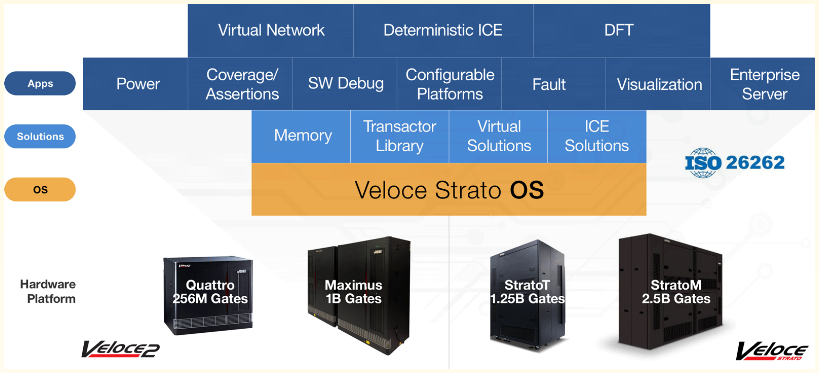
Synopsys Emulation (ZeBu)


Transaction-based Verification
system-level SoC verification
Hybrid Emulation
architecture optimization & early software development
In-circuit Emulation
real-world connections
Simulation Acceleration
up to 100x simulation performance
Power-aware Emulation
UPF support and SAIF output
Synthesizable Testbench
higher performance
SYNOPSYS

Cadence Special-Purpose Emulator (Palladium)



Mentor Emulation (Veloce)


Virtual Network
Deterministic ICE
DFT
Apps
Power
Coverage/ Assertions
SW Debug
Configurable Platforms
Fault
Visualization
Enterprise Server
Solutions
Memory
Transactor Library
Virtual Solutions
ICE Solutions
ISO 26262
OS
Veloce Strato OS
Hardware Platform
Quattro 256M Gates
Maximus 1B Gates
StratoT 1.25B Gates
StratoM 2.5B Gates
Veloce2
Veloce