24. New Directions in Verification

Jacob Abraham

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

Verification of Digital Systems
Spring 2020

April 23, 2020

Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology

Verification is a Fundamental Technology

- Dealing with the analysis of extremely complex systems
- Can answer questions about the behavior of systems
- Verification algorithms and abstraction techniques can be applied to a variety of application problems
  - Generating at-speed (functional/application-level) tests for faults in an embedded module in a VLSI chip
  - Identifying accurate application-level power consumption from module-level power information
  - Automatically identifying portions of a chip which can be gated off during a particular clock cycle (to reduce power)
  - Automatically finding "implications" of a given pattern in a data set
  - Improving statistical correlations in data mining
- The verification problem will never go away, as long as there are designs
  - Will always need to verify the correctness of designs

Look at the Big Picture

Verification can also be applied to the other areas involving not only integrated circuits, but also any problem which can be modeled using logic functions and where answers to logic questions are desired

Applicability of Verification Techniques

Simulation-Based Verification

- Can simulate very large designs
- Drawback: only a very small fraction of all possible inputs can be simulated in practice
- Generally use random or targeted sequences to achieve a coverage goal

Formal Verification

- Can only deal with small blocks
- Definite answer to whether a property holds, or a counter-example trace (assume the block is within the capacity of the tool)
- Useful for checking uncovered states, properties, etc., to achieve a coverage target

Application of Machine Learning

- Relatively new area of research
  - Lot of hype
- Techniques "learn" from data generated from the system, UVM tests, for example
- Genetic algorithms (discussed previously) are a type of "reinforcement learning"

Some questions to ponder when considering using machine learning for verification

- Can we use learning to find a bug, or prove the absence of a bug
- If massive amounts of data are needed for high quality learning, are there other techniques which would require less effort?
- Can the data point to areas not covered by the tests?
- Can additional information regarding the behavior of the system be deduced (learned)?

Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology

Software-Based (Native-Mode) Self Test for Processors

- Why not use functional capabilities of processors to replace BIST hardware?
  - No additional hardware
- Reduce test costs by using low-cost testers
- Increase coverage of delay defects and increase yield by testing native
- No issues with excessive power consumption during test

Developed at University of Texas (Int'l Test Conference 1998)

Application to processors at Intel (Int'l Test Conference 2002)

Are Random Tests Sufficient?

Intel implementation involved code in the cache which generated random instruction sequences

Interest in generating instructions targeting faults

Possible to generate instruction sequences which will test for an internal stuck-at fault in a module

In order to deal with defects in DSM technologies, **need to target small delay defects**

Recent work: **automatically generate instruction sequences which will target small delay defects in an internal module**

Tests for Small Delay Defects

Need to test paths in the circuit to detect small delay defects

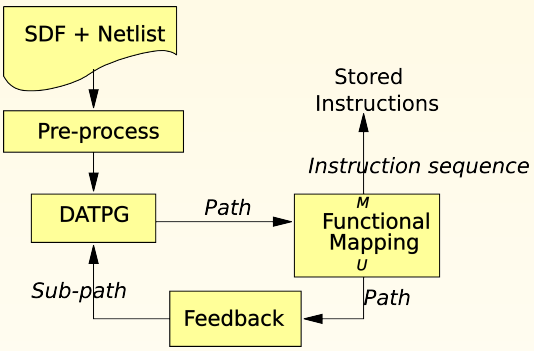
However, the number of paths in a circuit can be exponential in the number of nodes

Solution: test the longest path through every node

- This will detect the smallest possible delay increase which will cause the circuit to fail

Total number of tests is linear in the number of nodes

Automatic Generation of Instruction Sequences for Small Delay Defects



- Phase 1: all paths above a delay threshold
- Phase 2: longest paths through all nodes
- Delay-Based ATPG: generate "TRUE" paths above given delay threshold
- **Functional mapping: using verification engine**
- Feedback: heuristics to speed up search

Verification Property to Generate Test Sequence

Path

Net	Transition
InsA.Y	Rising
InsB.Y	Falling
insC.Y	Rising

Property

```
if ((InsA.Y = = 0) && (InsB.Y = =1) && (InsC.Y = = 0))
begin
  wait(1);
 if ((InsA.Y = = 1) && (InsB.Y = =0) && (InsC.Y = = 1))
begin
    wait(1);
    .....
```

Modified property with instructional constraints

```
#define `insn_legal = ((insn[31:26] == `insn_add) || (insn[31:26] ==`insn_sub)
                    || (insn[31:26] == `insn_shift));
always
   if (`insn_legal && (InsA.Y = = 0) && (InsB.Y = =1) && (InsC.Y = = 0))  begin
      wait(1);
      if (`insn_legal && (InsA.Y = = 1) && (InsB.Y = =0) && (InsC.Y = = 1)) begin
         wait(1);
            ......
```

Results on OR1200 processor

www.opencores.org, synthesized for 0.18μ TSMC process

Results for Phase 1 (paths > 80% of clock)

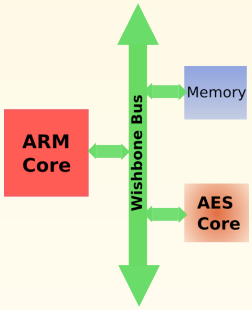
No. of Paths	Drop	Functionally Testable	Functionally Redundant	Time out
27424	12	15118	12106	200

Results for Phase 2
N: % nodes with test for longest path through them

Module	Functionally Testable	Functionally Redundant	Rejected Sub-paths	N (%)
or1200_ctrl	1826	29191	68087	90.6
or1200_alu	1427	16985	2716	100
or1200_lsu	970	4077	3744	100
or1200_wbmux	1146	2285	2118	100

Test of SoC Cores using Embedded Processor (*Gurumurthu et al., 2008*)

Wishbone and 128-bit AES designs from opencores.org
Validation vectors: random values encrypted/decrypted



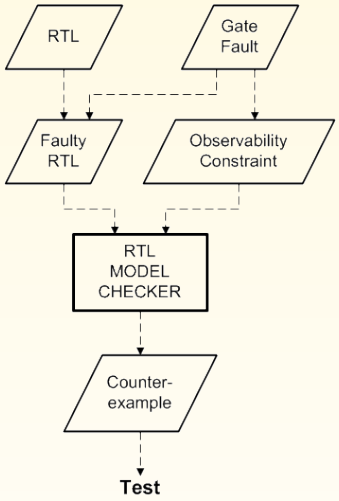
AES Core

Inputs	69
Outputs	33
Combinational primitives	9225
Sequential primitives	1119
Stuck-at faults	64070

Result of Mapping AES tests to ARM instructions (one case)

	Size (bytes)	Fault coverage(%)	Original Coverage(%)	No. of Cycles	Original Cycles
Test	9128	90.15	90.35	7816	7435

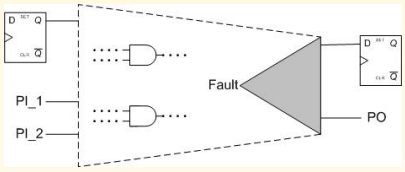
RT Level Test Generation Targeting Hard-to Detect Faults (*Prabhu et al., 2012*)



Overview

- Map gate level stuck-at fault to RTL
- Capture the propagation constraints as a LTL property
- Generate a witness for the LTL property using bounded model checking
- Use SMT based bounded model checking
- Scale with cone of influence reduction

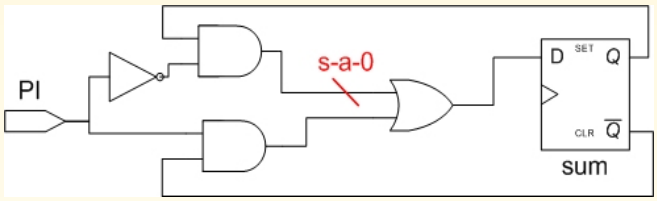
Modeling Stuck-at Faults in RTL



Approach

- Assume one to one match between flops in RTL, netlist
- Identify flops/primary outputs $o_1, o_2, ..., o_n$ in output cone of the fault
- Identify the boolean function for each of the output flops/primary outputs. Ex $o_k = f_k(i_1, i_2, ..., i_m)$
- Identify the boolean function for the output flops with the fault inserted. Ex $o_k^f = f_k^f(i_1, i_2, ..., i_m)$
- Fault condition : $fault_k = f_k(i_1, i_2, ..., i_m) \oplus f_k^f(i_1, i_2, ..., i_m)$

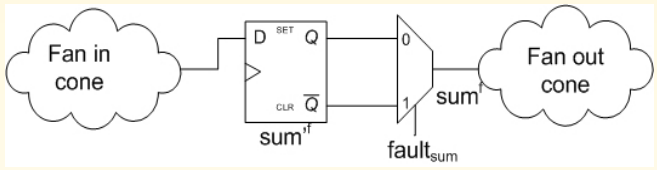
Modeling Stuck-at Faults in RTL



Example

- $always\ @(posedge\ clk)$
  $sum <= PI \oplus sum;$
- $sum = (PI \wedge \neg sum) \vee (\neg PI \wedge sum)$
- $sum^f = PI \wedge \neg sum$
- $fault_{sum} = sum \oplus sum^f = \neg PI \wedge sum$

Modeling Stuck-at Faults in RTL



Example

$always\ @(posedge\ clk)$
$sum'^f <= PI^f \oplus sum^f;$

$assign\ fault_{sum} = (\neg PI^f \wedge sum^f);$

$assign\ sum^f = (fault_{sum} ? \neg sum'^f : sum'^f);$

Experimental Setup

Process

- OR1200 RISC processor was DUT (included multiplier in data path)
- EBMC Model checker / Boolector SMT solver
- Bound of pipleine depth + 1
- Focused on hard to detect faults in control logic
- Commercial ATPG to seive out easy to detect stuck-at faults
- 78% Fault coverage with commercial ATPG

Experimental Results

Module	ATPG FC(%)	Flts.	SAT based method			Naive Observability Method		
			FC(%)	# TO	T(sec)	FC(%)	# TO	T(sec)
if	80.35	328	84.11	310	96.18	88.49	161	95.13
ctrl	63.21	832	65.97	817	83.12	97.15	59	69.72
oprmuxes	73.66	378	76.09	354	95.49	98.26	6	57.46
sprs	89.59	393	90.85	381	93.71	93.78	57	90.27
freeze	82.94	17	99.14	2	64.41	100	0	43.51
rf	78.59	7444	80.50	7268	97.57	90.21	463	69.83
except	72.69	1263	73.48	1209	98.63	92.79	128	96.19
Overall	78.05	10655	79.17	10343	96.23	93.86	874	76.11

FC(%) : Fault Coverage in %
 # Faults : # of Undetected Collapsed Faults
 # TO : # of Timed Out faults
T(sec) : Average Time for generating a test for a fault in seconds

Experimental Results, Structural Observability

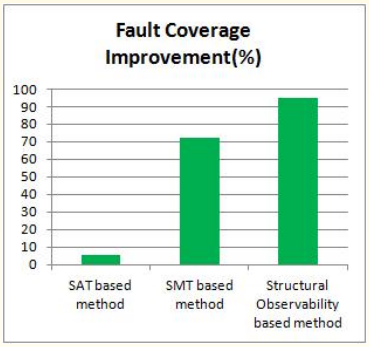
Module	FC(%)	# TO	T(sec)
if	98.17	25	23.14
ctrl	99.21	8	21.16
oprmuxes	100	0	19.33
sprs	97.53	12	18.39
freeze	100	0	10.48
rf	98.37	172	22.85
except	97.63	69	38.14
Overall	98.87	454	24.23

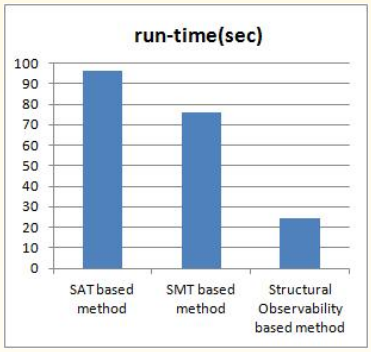
FC(%) : Fault Coverage in %
 # Faults : # of Undetected Collapsed Faults
 # TO : # of Timed Out faults
T(sec) : Average Time for generating a test for a fault in seconds

Summary of Results

- Functional fault coverage of 99% for OR1200 processor
- SMT based approach was 4x faster than SAT

Coverage and Run Time Comparisons (*Prabhu et al., 2012*)





Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology

Early Power Estimation (RTL and Above)

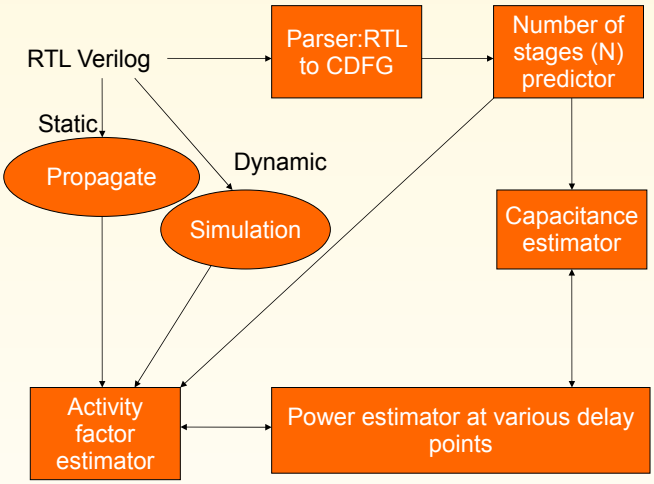
Activity factor estimation

- Logic functions do not change due to synthesis, only their implementations change
- Approximate the activity at the RT-Level
  - Get input-output activity by RT-Level simulation
  - Empirical observation to obtain activity in intermediate stages, $sf_i = (sf_{in} - sf_{out}) * (1 - \frac{i}{N})^2 + sf_{out}$
  - Quadratic variation with respect to logic depth

Logical effort, modified to extract capacitance for any delay target

- Stage effort from delay $f = F^{\frac{1}{N}} = \frac{\frac{D}{\tau} - P}{N}$
- Sizing of nodes $C_{in} = C_{out} * \frac{g}{f}$

Prototype Tool



Experiments

Estimated values vs. reference values
Reference values obtained at gate-level
Interconnect: wire-load model
Libraries:

- Artisan TSMC 0.18μm
- Virtual Silicon UMC 0.13μm

Library sets: (x1, x2, x4) (2ip, 3ip, 4ip)
Circuits

- OR1200 and FPU (opencores)
- ISCAS high-level models
- ISCAS sequential circuits

Results (robust with respect to technologies and libraries)

Combinational circuits (0.18μm)

Target gate library	Average abs. error	Average rel. error
1,2 ip	17.12	8.05
1,2,3 ip	18.95	11.81
1,2,3,4 ip	19.60	17.65

After accuracy improvement:

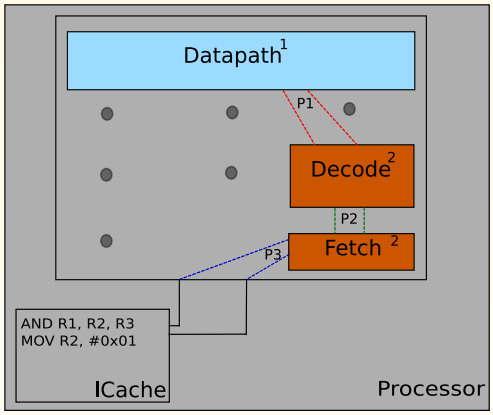
Relative error estimates for sequential circuits

Circuits	Target gate library	Err%_0.13	Err%_0.18
Behavioral	1,2 ip	6.48	6.80
	1,2,3 ip	5.75	7.89
	1,2,3,4 ip	5.47	6.79
Structural ISCAS	1,2 ip	8.26	10.19

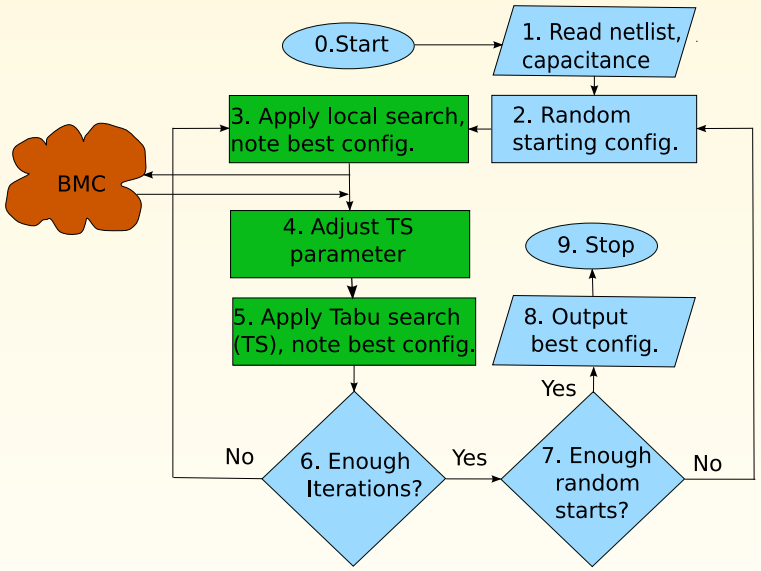
Peak Power Estimation

Objective

Finding an instruction stream which maximizes the dynamic power, given the gate-level description of the processor

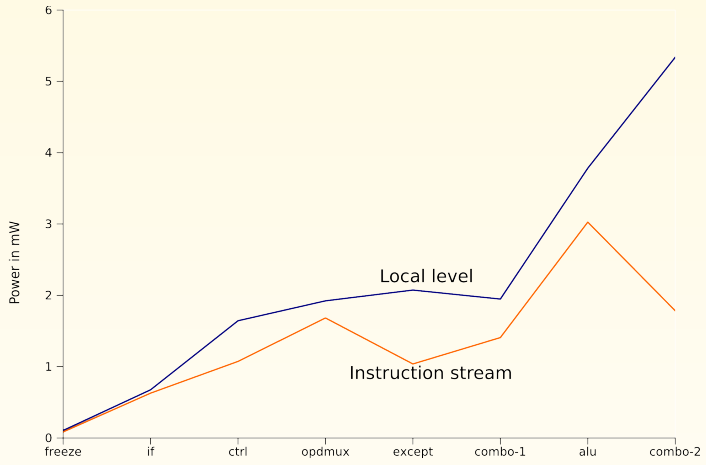


Algorithm



Power: Module-Level versus Processor-Level

Power in mW



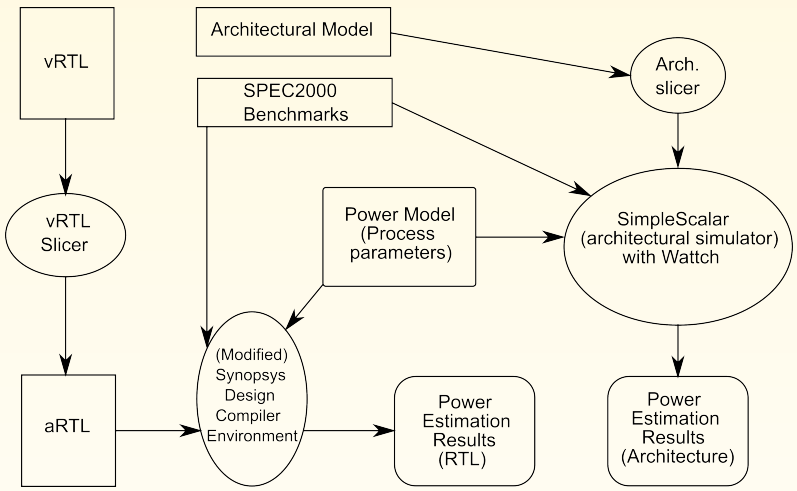
Automatic Annotation of RTL Code for Low Power

Instruction-driven slice of a microprocessor design

- All the relevant circuitry of the design required to completely execute a specific instruction
- Parts of the decode, execute, writeback etc. blocks
- Cone of influence of the semantics of the instruction

- Given a microprocessor design and an instruction
  - Identify the instruction-driven slice
  - Shut off the rest of the circuitry
- This might include
  - Gating out parts of different blocks
  - Gating out floating point units during integer ALU execution
  - Turning off certain FSMs in different control blocks since exact constraints on their inputs are available due to instruction-driven slicing

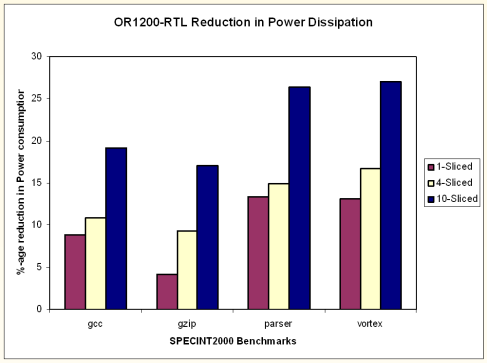
Approach



Results on a Simple Pipelined Processor (OR1200)

Single instruction issue pipelined RISC microprocessor

- Results shown after inserting annotations
  - Sliced on 1, 4, 10 instructions
  - For SPECINT2000 benchmarks

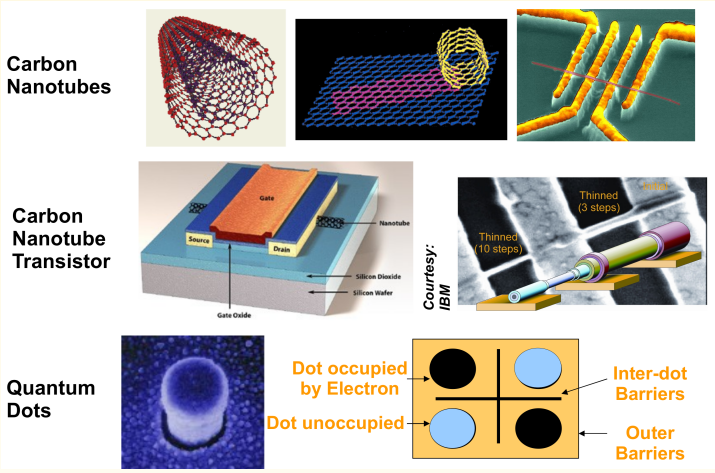


Similar results on PUMA (dual-issue, out-of-order super-scalar, fixed-point PowerPC core)

Outline

- Introduction
- Testing circuits after manufacture and in the field
  - Test generation for small delay defects
  - Tests for hard-to-detect faults
- Low-power systems
  - Early power estimation
  - Peak power estimation
  - Automatic annotation of RTL code for low power
- Application of verification to other domains
  - Verifying system security properties
  - Verifying safety-critical systems
  - Verification technology applied to biology

Longer Term Technologies



The basic principles of design do not change – just using different building blocks

Verification solutions are still relevant

Design Bugs

Logic bugs

- Verification is dominating the design cycle
- Unlikely that all design bugs are caught before deployment
- Diversity is necessary to deal with design bugs

Design margins

- Effects of real bugs are not easy to duplicate (in many cases, error latencies of many millions (or billions) of cycles)
- Gray: concepts of **Bohr bugs** (repeatable) versus **Heisenbugs** (not seen to be repeatable)

Bugs and design margins could be exploited by an attacker

Security Attacks

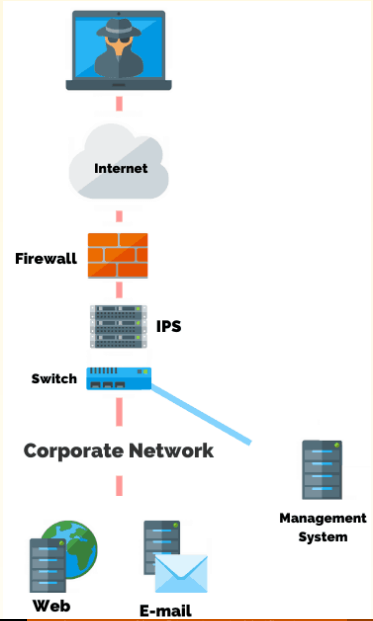
Hardware Trojans

- Malicious modification of designs
- Example of analog circuitry modifying a digital chip – extremely difficult to identify
- Design diversity may be a solution

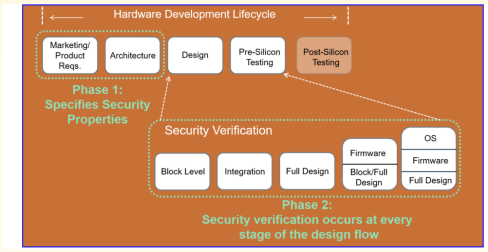
External attacks

- Classic work (Abadi) suggested control flow checking to detect execution of undesired code
- Effects of attacks could include modification of data, execution sequences, denial of service, etc.
  - Require data checks in addition to control-flow checks
  - Need to detect DoS attacks during operation – example, shutting down GPS system (or spoofing GPS position)

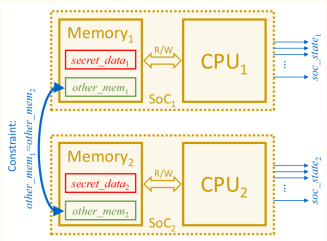
Detecting and Preventing Intrusions



Verifying Hardware Security Properties

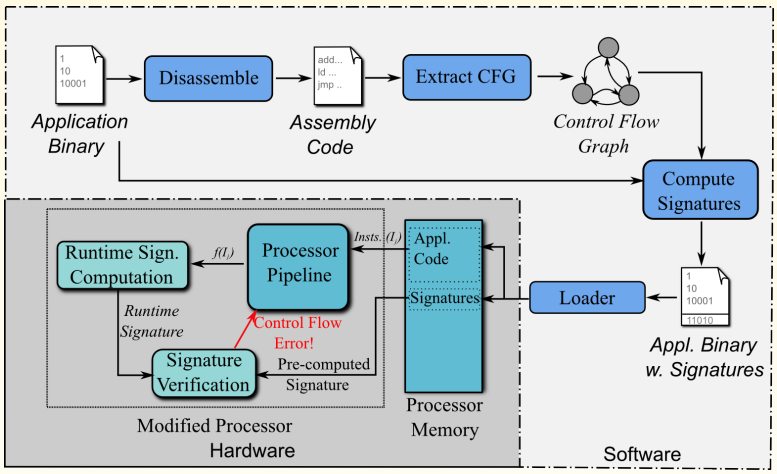


Source: Tortuga Logic

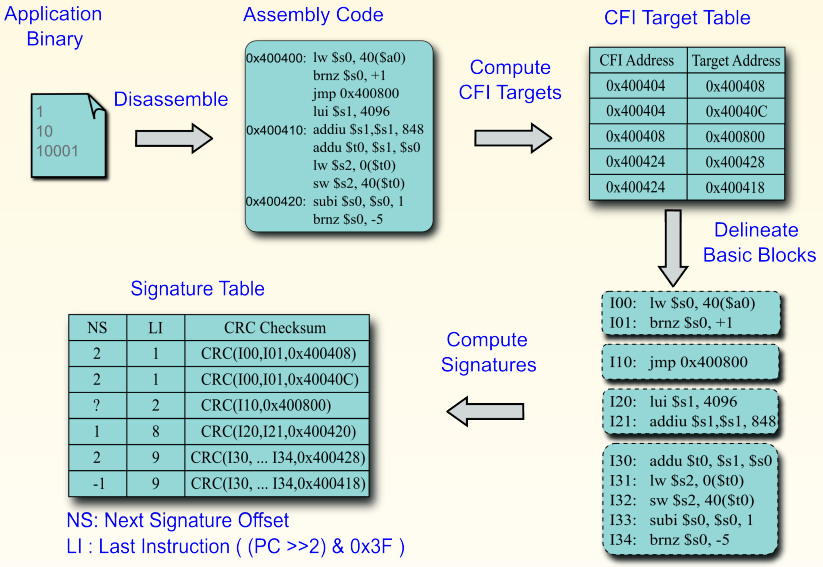


Source: Fadiheh et. al, DATE 2019

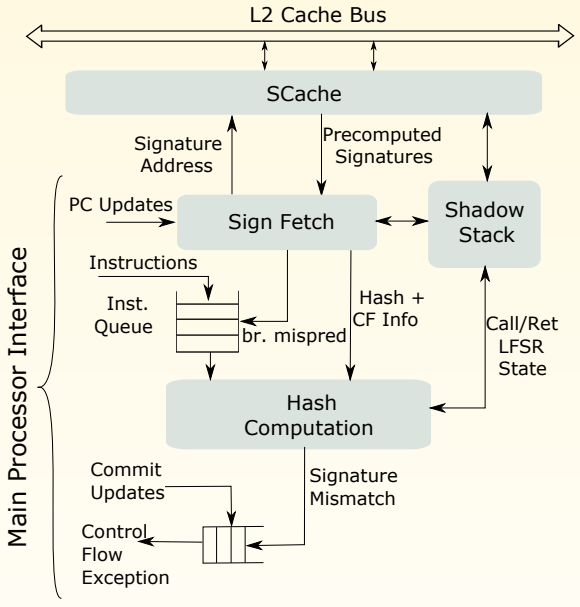
Framework for Hardware Control Flow Monitoring (*Chaudhari et al., 2012*)



Signature Computation

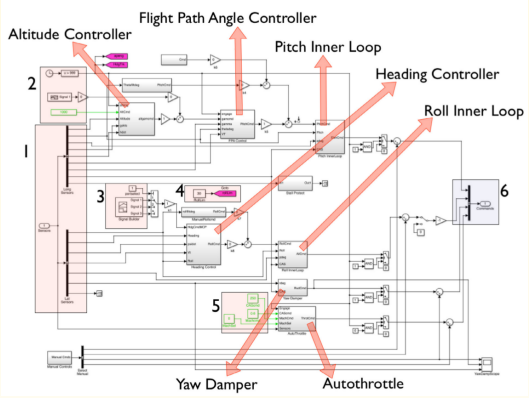


Control Flow Co-Processor Architecture



Verifying Safety of a Flight-Critical System

Simulink "Transport Class Model" (TCM) (from NASA) of twin-engine aircraft



- Properties derived from higher-level safety requirements

Brat *et al.*, "Verifying the Safety of a Flight-Critical System," LNCS 9109, 2015

Summary of Verified TCM Properties

#	Property	Assumptions	Original Requirement
1	G-250	G-260	The heading control mode, when selected, sends roll commands to turn to and maintain the commanded heading.
2	G-110	G-220,G-260	The guidance system shall be capable of steering to and following a specified heading.
3	G-120	G-180,A1,A2, FPA1	The guidance shall be capable of climbing at a defined rate, to be limited by minimum and maximum engine performance and airspeed.
4	G-130	G-180,A1,A2	The guidance shall be capable of descending at a defined rate, to be limited by minimum and maximum engine performance airspeed.
5	G-140	G-120,G-200	The guidance shall be capable of climbing at a specified rate to a specified altitude, to be limited by maximum engine performance for a set airspeed
6	G-150	G-180,A1,G-120, A2,G-200	The guidance shall be capable of descending at a specified rate to a specified altitude, to be limited by maximum engine performance for a set airspeed
7	G-170 (Mode)	–	The altitude control shall engage when the altitude control mode is selected and when the FPA control mode is not selected, and when there is no manual pitch or manual roll command from the stick.
8	G-180 (Mode)	–	The FPA control shall engage when the FPA mode is selected, and when there is no manual pitch or manual roll command from the stick.
9	G-100	–	The Guidance system shall be capable of maintaining a steady speed in the normal flight envelope.
10	G-200	–	If the altitude control is engaged, once the plane is within 250 ft of the commanded altitude, the plane will remain within 250 ft of the commanded altitude.

Summary of Verified TCM Properties, Cont'd

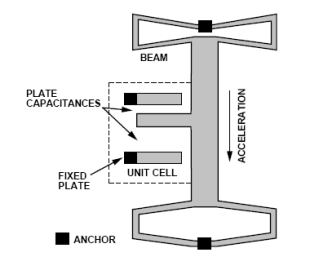
#	Property	Assumptions	Original Requirement
11	G-210 (Mode)	–	If the FPA control and the altitude control are both selected, the FPA control will disengage and the altitude control will engage once the lane is within 200 ft of the commanded altitude.
12	G-220 (Mode)	–	The heading control shall engage when the heading control mode is selected, and when there is no manual pitch or manual roll command from the stick.
13	G-230	–	If the altitude control is engaged with no active speed control, the speed control shall engage and the speed command shall synchronize to the current speed, which shall become the new altitude's target speed.
14	G-240	–	The bank angle limit is established by the Bank Angle Limit Selector.
15	G-260 (Mode)	–	When the heading control mode is engaged, roll commands are given to turn in the nearest direction to the selected heading.
16	G-270 (Mode)	–	Manually positioning the thrust levers does not cause autothrottle disengagement.
17	G-290	–	The autothrottle will be limited by the max and the min throttle.
18	G-160	–	The guidance function shall be able to automatically deploy spoilers to limit speed in a descent, or when a significant reduction in airspeed is requested by the pilot, deactivating at low speed.
19	G-280	–	The FCCs shall issue a warning when the commanded altitude disagrees with the stored commanded altitude stored in the FCCs.
20	G-190	–	If any control surface actuator loses hydraulic pressure, the autopilot shall disengage.

Modeling Errors Affecting Verification Results

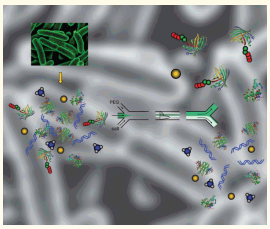
- Some components produced output when disabled (e.g., the altitude controller)
  - TCM model provided was incomplete
- Manual inputs from the pilot did not override the outputs of the autopilot for all three axes
  - Incompleteness in the TCM model
- Some inputs were not variables but appeared as xed constant values in the model (e.g., the bank angle limit of G-240)
  - Modeling error
- Conict of G-180 with with G-210 and the implicit assumption that only the flight path angle control or the altitude control can be active at any moment in time
  - G-180 had to be rened

Applications of Verification Technology in Other Domains

- Analysis of complex systems
  - Analyzing the power grid for "green" power
  - Analyzing DNA sequences
- Analysis of emerging systems
  - Micromechanical systems
  - Microfluidics systems

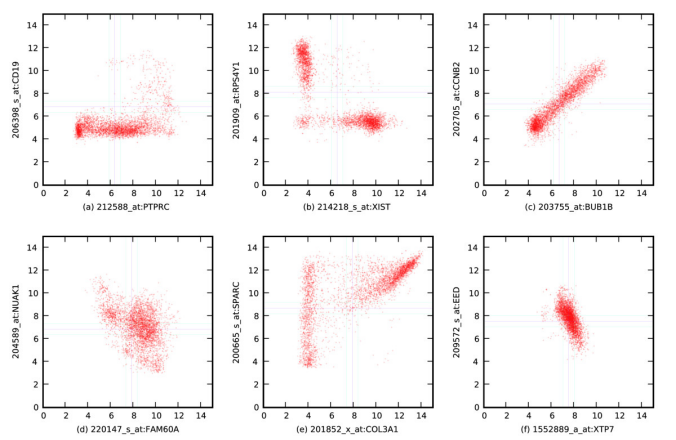


Analog Devices ADXL204 MEM Accelerometer



Sandia Labs: purification of proteins in a microfluidic device using genetically-engineered partition tags

Finding "Implications" in DNA Sequences



Six different types of Boolean relationships between pairs of genes taken from the Affymetrix U133 Plus 2.0 human dataset
Found using Boolean verification tools The two axes correspond to the expression levels of two genes

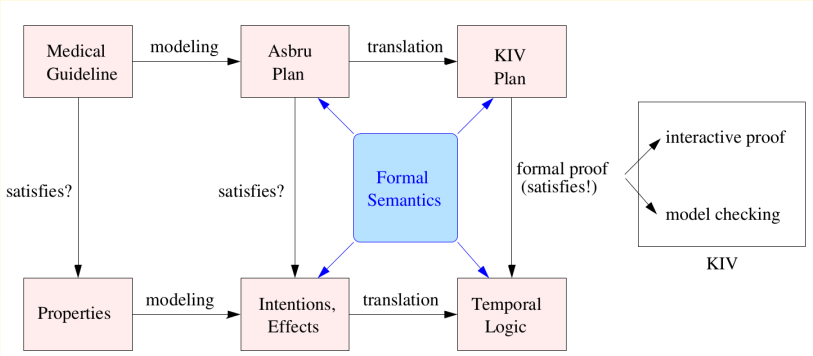
Source: Sahoo et al., Genome Biology 2008

Application to Systems Biology – Example Tools

- BAM, LDL degradation pathway
- BIOCHAM, Mammalian cell cycle control, G protein-coupled receptor kinases
- BoolNet, Genetic networks
- COPASI, Biochemical networks
- GreatSPN, Signal transduction pathways for angiogenesis
- IBM Rational Rhapsody, T-cell activation with statecharts
- PRISM, Biological signaling pathways, bone pathologies
- Simulink, Heart model for pacemaker verification
- S-TaLiRo, Modeling insulin-glucose regulatory system

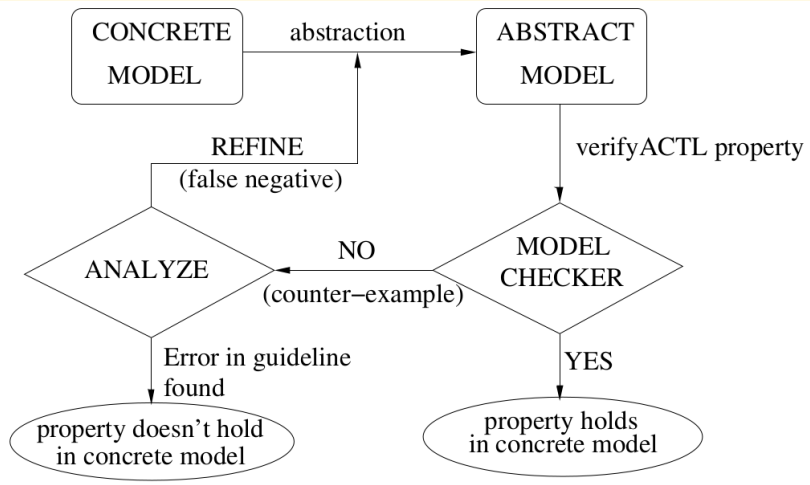
Bartocci and Lio, "Computational Modeling, Formal Analysis and Tools for Systems Biology," *PLOS Comutational Biology*, Jan. 21, 2016

Formalization and Verification of Medical Guidelines



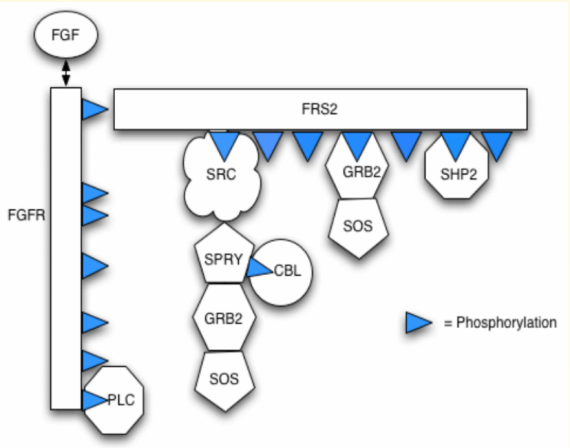
Baumler, LNCS 3925, 2006

Verification Process



Probabilistic Model Checking of Complex Biological Pathways

Fibroblast Growth Factor (FGF) pathways, modeled in PRISM

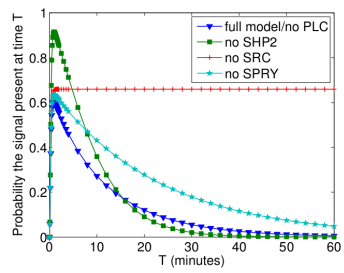


Heath, Computational methods in system biology, 2006

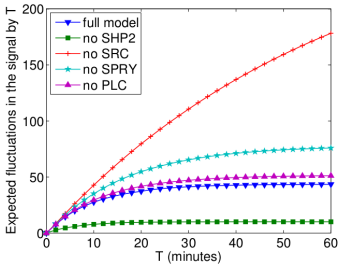
Partial Reaction Rules for Pathway

1. FGF binds to FGFR
FGF+FGFR ↔ FGFR:FGF (k_{on} = 5e+8M^{-1}s^{-1}, k_{off}=1e−1s^{-1})
2. Whilst FGFR:FGF exists
FGFR Y653 → FGFR Y653P (k_{cat}=0.1s^{-1})
FGFR Y654 → FGFR Y654P (k_{cat}=0.1s^{-1})
3. When FGFR Y653P and FGFR Y654P
FGFR Y463 → FGFR Y463P (k_{cat}=70s^{-1})
FGFR Y583 → FGFR Y583P (k_{cat}=70s^{-1})
FGFR Y585 → FGFR Y585P (k_{cat}=70s^{-1})
FGFR Y766 → FGFR Y766P (k_{cat}=70s^{-1})
4. FGFR binds FRS2
FGFR+ FRS2 ↔ FGFR:FRS2 (k_{on} = 1e+6M^{-1}s^{-1}, k_{off}=2e−2s^{-1})
5. When FGFR Y653P, FGFR Y654P and FGFR:FRS2
FRS2 Y196 → FRS2 Y196P (k_{cat}=0.2s^{-1})
FRS2 Y290 → FRS2 Y290P (k_{cat}=0.2s^{-1})
FRS2 Y306 → FRS2 Y306P (k_{cat}=0.2s^{-1})
FRS2 Y382 → FRS2 Y382P (k_{cat}=0.2s^{-1})
FRS2 Y392 → FRS2 Y392P (k_{cat}=0.2s^{-1})
FRS2 Y436 → FRS2 Y436P (k_{cat}=0.2s^{-1})
FRS2 Y471 → FRS2 Y471P (k_{cat}=0.2s^{-1})
6. Reverse when Shp2 bound to FRS2:
FRS2 Y196P → FRS2 Y196 (k_{cat}=12s^{-1})
FRS2 Y290P → FRS2 Y290 (k_{cat}=12s^{-1})
FRS2 Y306P → FRS2 Y306 (k_{cat}=12s^{-1})
FRS2 Y382P → FRS2 Y382 (k_{cat}=12s^{-1})
FRS2 Y436P → FRS2 Y436 (k_{cat}=12s^{-1})
FRS2 Y471P → FRS2 Y471 (k_{cat}=12s^{-1})
FRS2 Y392P → FRS2 Y392 (k_{cat}=12s^{-1})
7. FRS2 effectors bind phosphoFRS2:
Src+FRS2 Y196P ↔ Src:FRS2 Y2196P (k_{on} = 1e+6M^{-1}s^{-1}, k_{off}=2e−2s^{-1})
Grb2+FRS2 Y306P ↔ Grb2:FRS2 Y306P(k_{on} = 1e+6M^{-1}s^{-1}, k_{off}=2e−2s^{-1})
Shp2+FRS2 Y471P ↔ Shp2:FRS2 Y471P(k_{on} = 1e+6M^{-1}s^{-1}, k_{off}=2e−2s^{-1})
8. When Src:FRS2 we relocate/remove
Src:FRS2 → relocate out ($t_{1/2}$=15min)

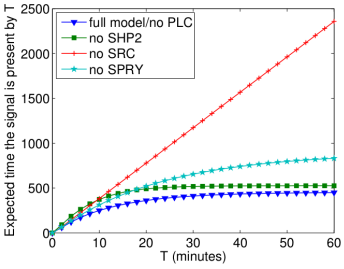
Transient Numerical Results



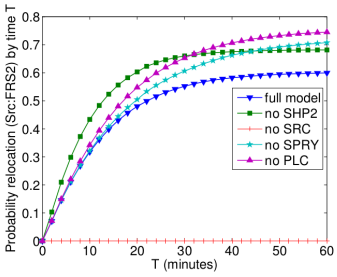
(a) Probability bound (Grb2:FRS2)



(b) Expected bindings (Grb2:FRS2)



(c) Expected time bound (Grb:FRS2)



(d) Probability relocated (Src:FRS2)