Leveraging Linux: Code Coverage for Post-Silicon Validation

Mehdi Karimi-biuki

Embedded Linux Conference 2013
San Francisco, CA

22 Feb, 2013

About me

- MASc (2012) and BASc (2009) from UBC
- 2 years of work experience at different companies Bcom, PMC, and Intrinsyc.
- Area of interest system validation and test + formal methods for debug + physical design.
- I am new in embedded software design
  - A firmware guy but mostly on the hardware side.

What we learn today?

- We look at the hardware examination of a very dominant test practice on today's industrial microprocessors
  - That is coverage of a popular test "Linux boot bring-up" on an industrial size system on chip called LEON3
- You will expect to learn how we did this (methodology) and how you can do it if interested.
- You will also see our interesting results achieved.
  - We conclude that Linux boot is a necessary test for today's state-of-the-art designs, but not enough.

Outline

- Motivation
  - Why post-silicon validation?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Conclusion and Future Work

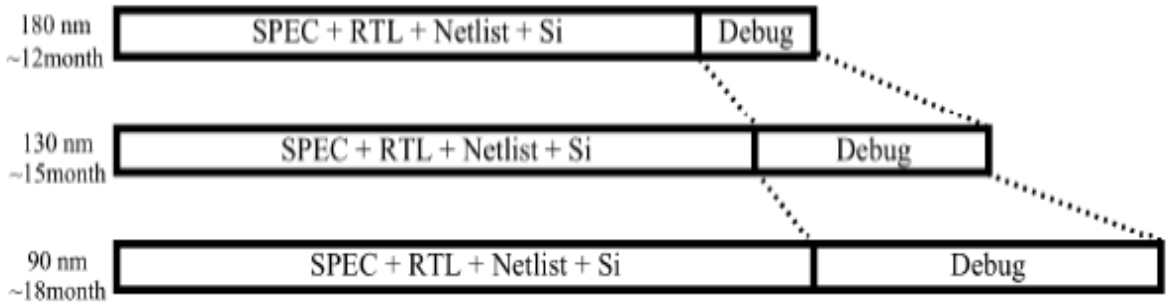
Outline

- Motivation
  - Why post-silicon validation?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Conclusion and Future Work

Why post-silicon validation?

- Post-silicon is everything that happens between tape-out and high-volume manufacturing.
- Fact: in a short time-to-market, SoC complexity continues to increase
  - Industry attempts to make more complex designs at shorter time to allow for:
    - Decreasing manufacturing and labor costs
    - Making low-power (clk-gating, power-gating, cloning, multi-bit register cells) designs while at higher speeds

Why post-silicon validation?



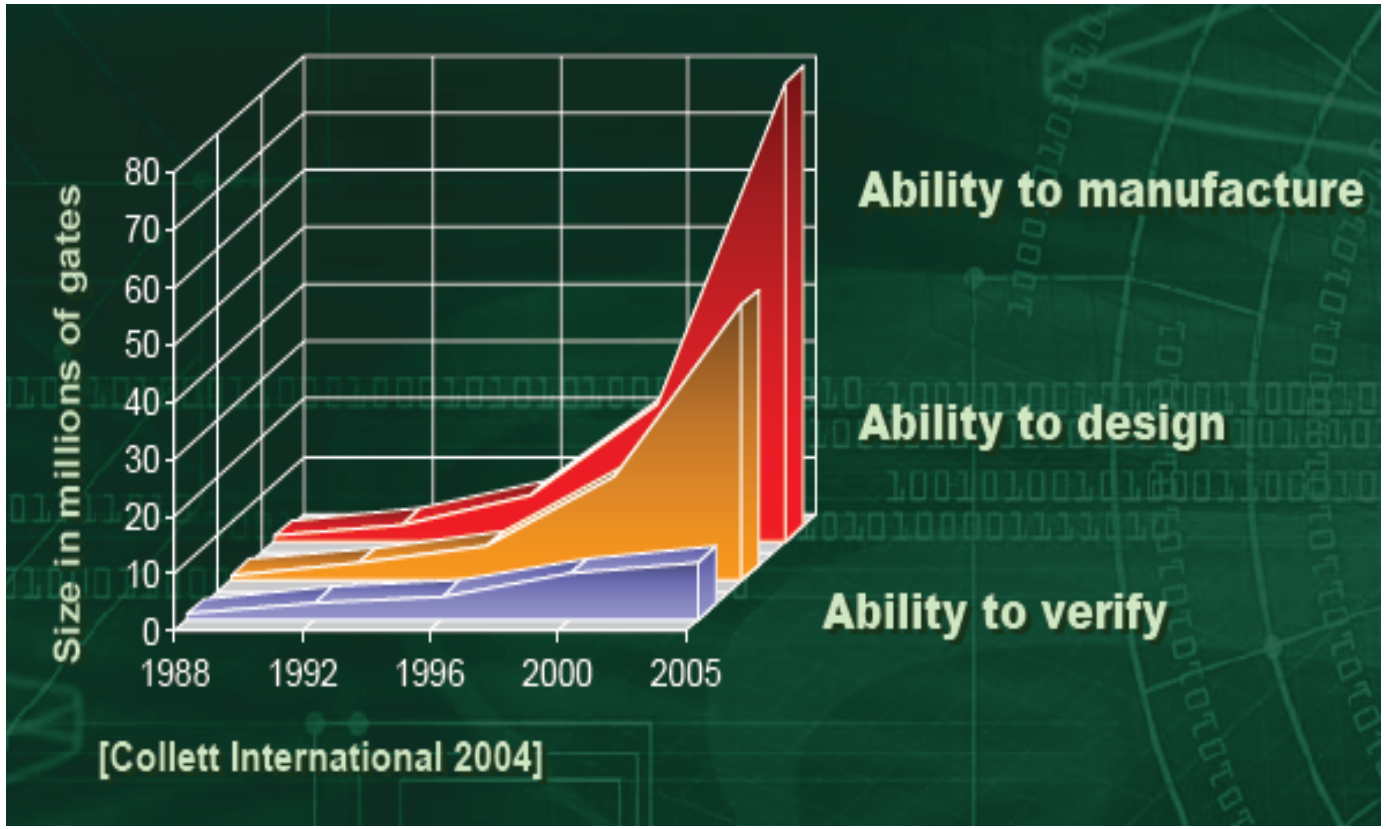
Miron Abramovici, Paul Bradley, Kumar Dwarakanath, Peter Levin, Gerard Memmi, and Dave Miller. 2006. A reconfigurable design-for-debug infrastructure for SoCs. In *Proceedings of the 43rd annual Design Automation Conference* (DAC '06). ACM, New York, NY, USA, 7-12.

Yet, all these come at a price

Bugs are harder to detect

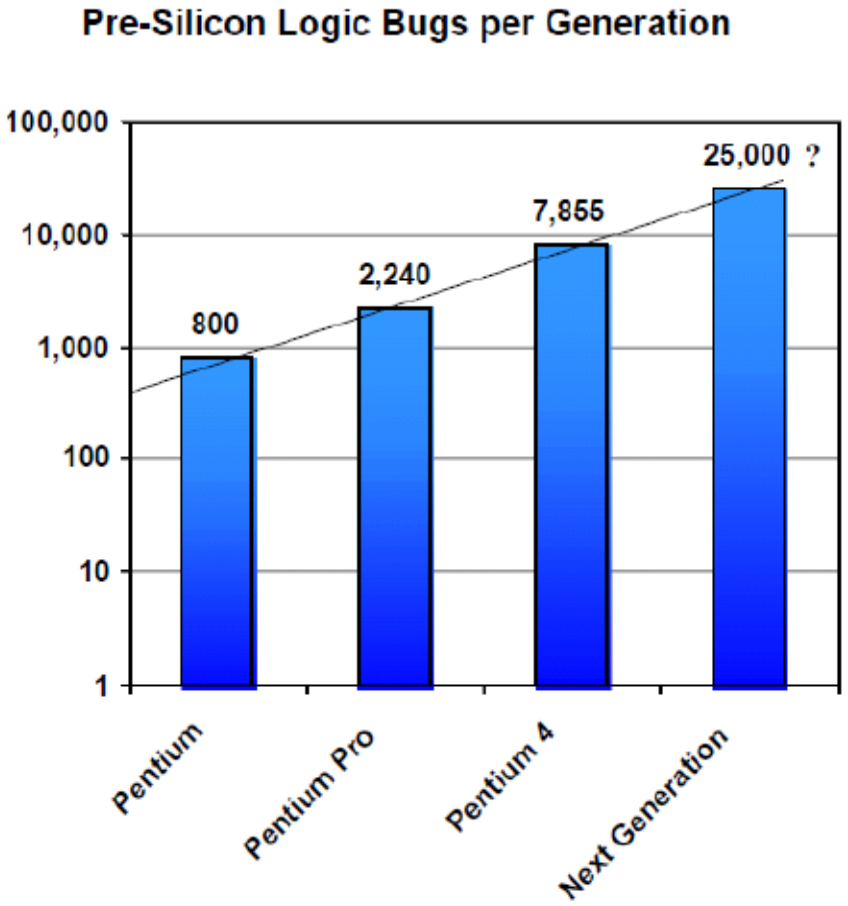
(It's easy to have bugs on silicon)

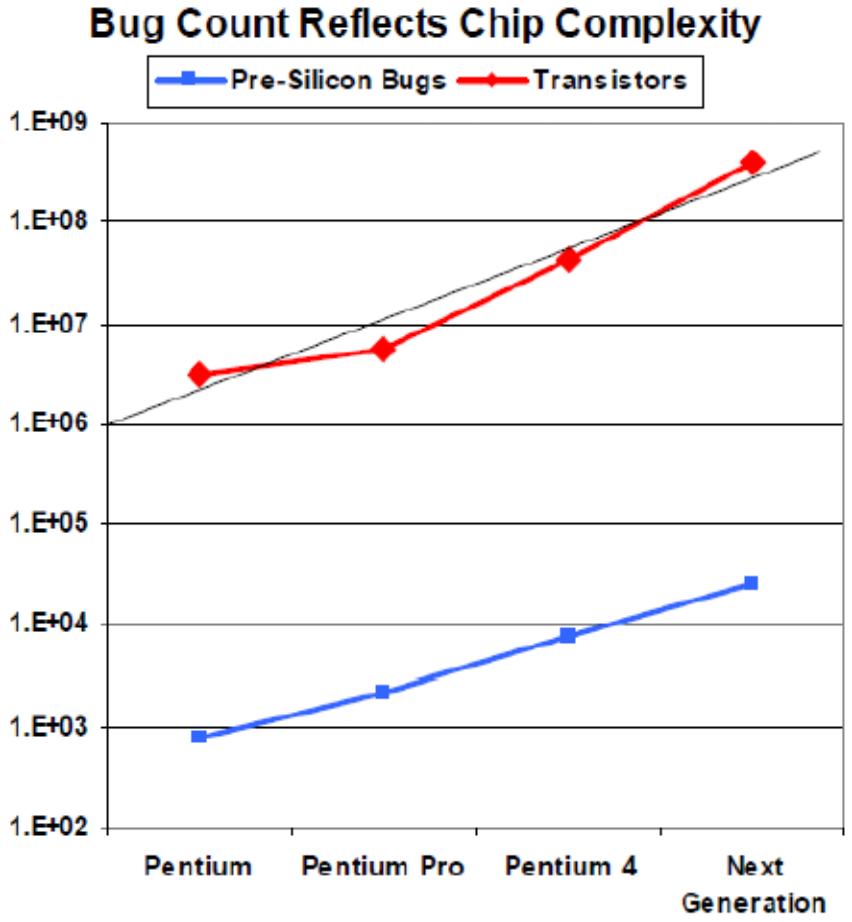
Data from Mentor Graphics…



Source: Harry Foster, Chief Technologist, Mentor Graphics

Data from Intel…





Source: Tom Schubert, Intel "High Level Formal Verification of Next –Generation Microprocessors" DAC 2003
Intel Corporate Web Site "Moore's Law … http://www.intel.com/technology/silicon/mooreslaw/index.htm

Post-Silicon vs. Pre-Silicon

- Pre-silicon techniques:
  - Advantages:
    - Very good visibility of internal events → better controllability → better feeling for debugging
      - Inexpensive bug fixing by setting breakpoints, etc.
      - Quick turnout time
  - **Disadvantages:**

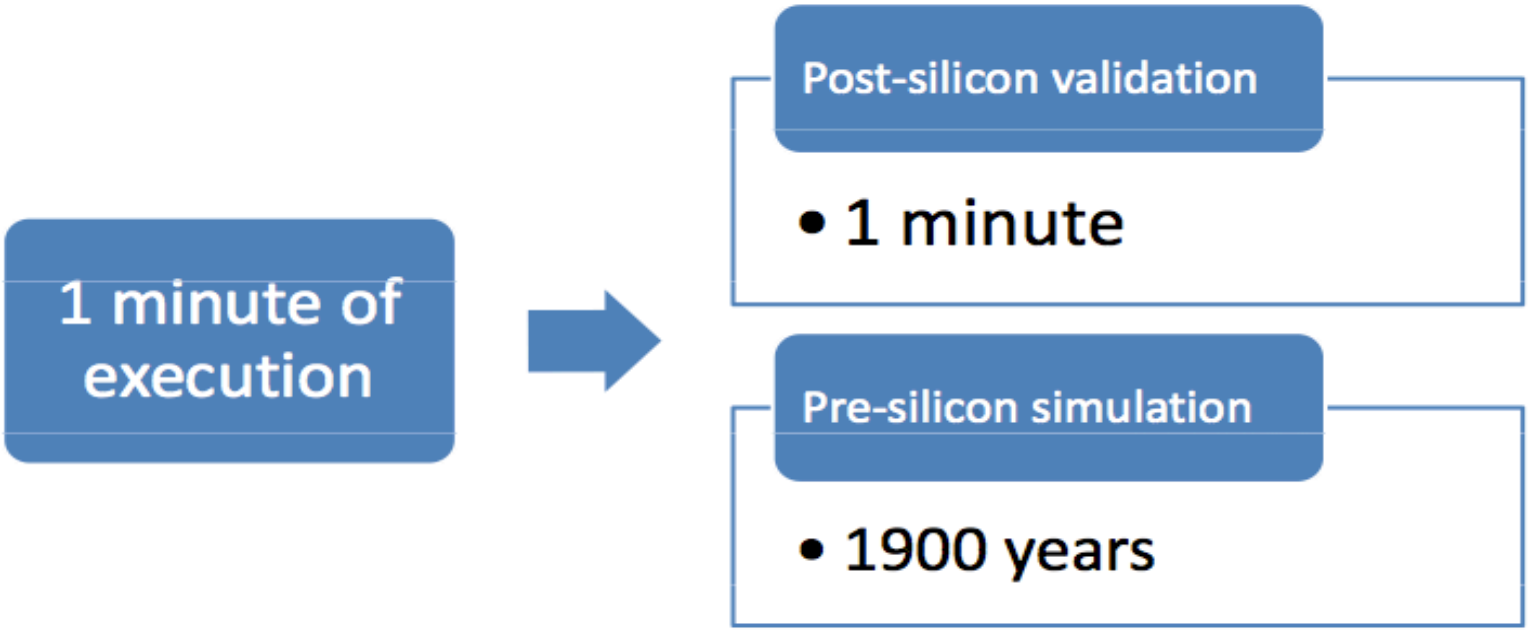
    -- Difficult to model complex electrical behaviors and off-chip interactions → requires very good understanding of the behavior of the chip for different functional modes… (some tools from Cadence/Synopsys/AtopTech)…

    3 million gates takes about 8 hours to simulate timing.

    -- 

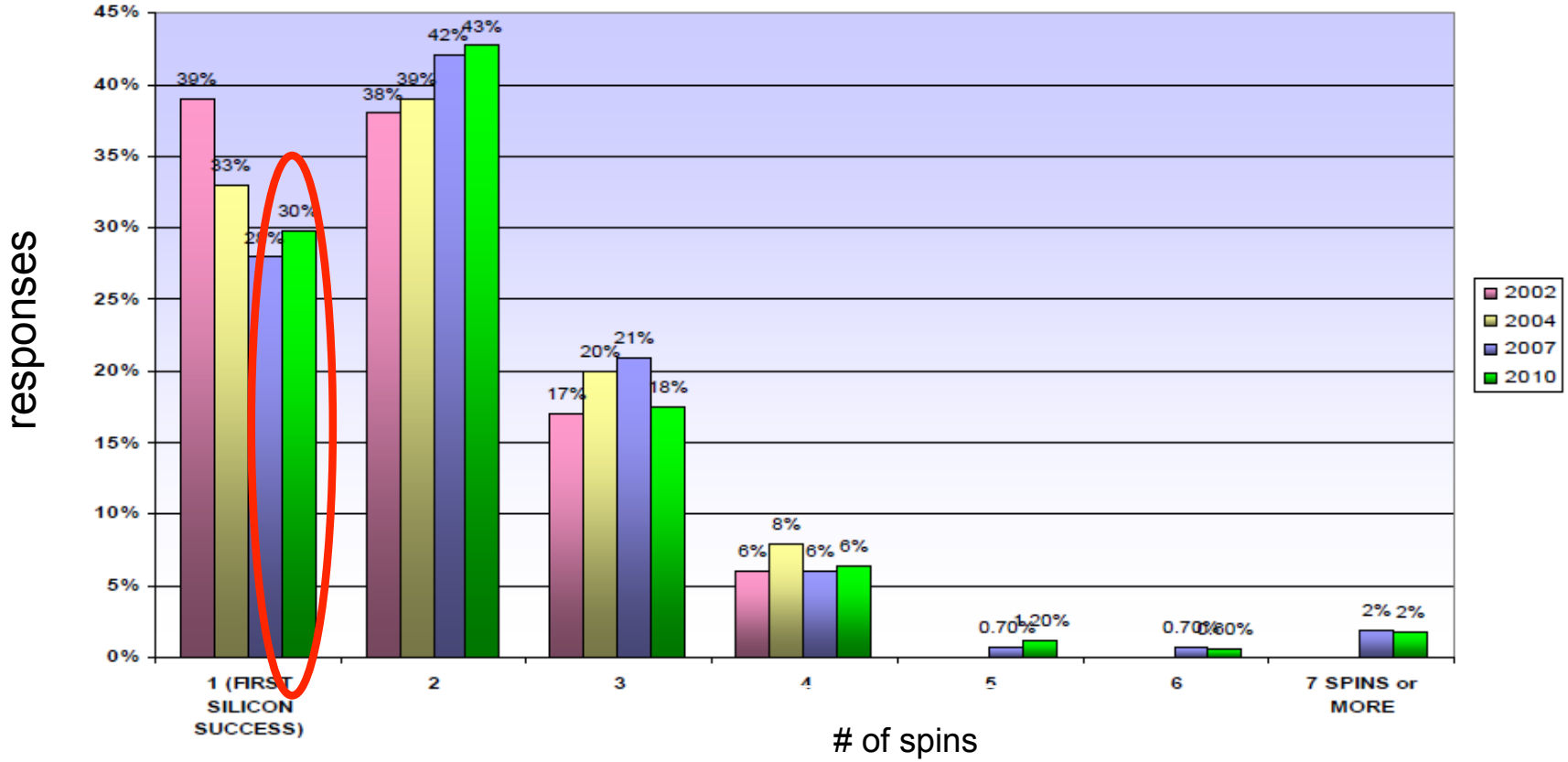
    up to nine orders of magnitude against real hardware

Post-Silicon vs. Pre-Silicon



Simulation is SLOW.
Consider Linux boot that takes 1 minute on actual hardware…it takes 1900 years in simulation!

1st Silicon success is only 30%



- More bugs are reported as complexity continues to increase
- Simulation is slow
- Therefore, there are more chances for bugs escaping into post-silicon (see figure above)
- Without a doubt, post-silicon validation is a must.

Wilson Research Group & Mentor Graphics 2010 Functional Verification Study, Used With Permission.

Outline

- Motivation
  - Why post-silicon validation?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Conclusion and Future Work

Why post-silicon coverage?

- Typical Post-Silicon Validation Process
  - In post-silicon, because it runs at full speed, we run real software applications to determine that the chip works as intended.
  - …run specialized test programs
  - Or random instruction streams
  - Check functionality while at different functional modes(func, scan_cap, rambist, etc.), and extreme process-voltage-temperature (PVT) corners.
  - Regression suites…
  - Prepare and run drivers…

And the question is…Did I do enough?????

I mean, if a software hangs, is it all from the software side, or hardware?

Why post-silicon coverage?

Did I do enough?????

(And if not, am I making progress? What areas need more verification? …)

How can I get a feeling about the effectiveness of my validation scheme…on the chip

The Solution is……

We can check for validation effectiveness with Coverage.

- This is similar to what is being done in pre-silicon verification → they all use coverage.
- Coverage is any scheme that measures the thoroughness of validation process.
- In post-silicon validation, due to limited observability
  - Coverage instrumentation in post-silicon is done by adding some on-chip monitors.

Outline

- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Conclusion and Future Work

Proposed techniques for post-silicon coverage

Industry has integrated some coverage metrics onto their chips:

- Intel Core2 Duo Family
  - Bojan, T.; Arreola, M.A.; Shlomo, E.; Shachar, T.; , "Functional coverage measurements and results in post-Silicon validation of Core™2 duo family," High Level Design Validation and Test Workshop, 2007. HLVDT 2007. IEEE International , vol., no., pp.145-150, 7-9 Nov. 2007
- IBM POWER7
  - Adir, A.; Nahir, A.; Shurek, G.; Ziv, A.; Meissner, C.; Schumann, J.; , "Leveraging pre-silicon verification resources for the post-silicon validation of the IBM POWER7 processor," Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE , vol., no., pp.569-574, 5-9 June 2011

→ There is a need for a complete coverage technique.

Outline

- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types.
  - Instrumentation
  - Case study and results
- Conclusion and Future Work

We employ Linux boot as a standard test to examine our code coverage analyses

- The mostly known and used test for chip bring up
  - Linux boot is widely used, widely accepted as a good test for first silicon chip.
  - Code coverage is a standard, objective coverage technique.

Code Coverage Types

- Statement
- Branch
- Condition
- Expression
- Finite State Machine (FSM)

We instrument Statement and Branch

Outline

- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case study and results
- Conclusion and Future Work

Measuring Code Coverage on Chip

1. We instrument HDL code by adding flags per “basic block”.
2. Then run Linux.
3. Then count the number of flags that are set divided by the total number of flags in each block.

Instrumenting for Statement Coverage

- Add one flag per "basic block":

```
process (example)
S1;
S2;                 SFlag0=1;
S3;
if (s4) begin       SFlag1=1;
        s5;
        s6;         SFlag2=1;
        s7;
else
        s8;
        s9;         SFlag3=1;
end if;
s9;                 SFlag4=1;
end process;
```

"Basic Block" is a sequence of consecutive statements with a single branch or return statement at the end. The flow of control enters and leaves the basic block without any branching into or out of the sequence.

Instrumenting for Branch Coverage

- Add two flags per branch:

```
process (example)
S1;
S2;
S3;
if (s4) begin   BFlag0=1;
            s5;
            s6;
            s7;
else            BFlag1=1;
            s8;
            s9;
end if;
s9;
end process;
```

Outline

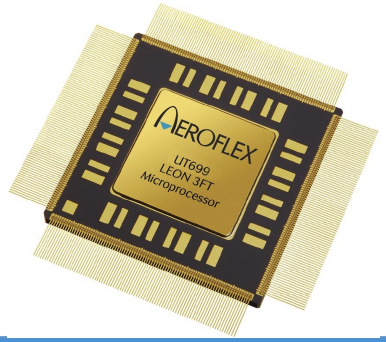
- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for Post-silicon coverage
- Post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case study and results
- Conclusion and Future Work

Case Study

- We pick an industrial-size SoC that is synthesizable to FPGA.
- Instrument code coverage in 9 blocks
- Measure post-silicon coverage
- Also compare with pre-silicon simulation results

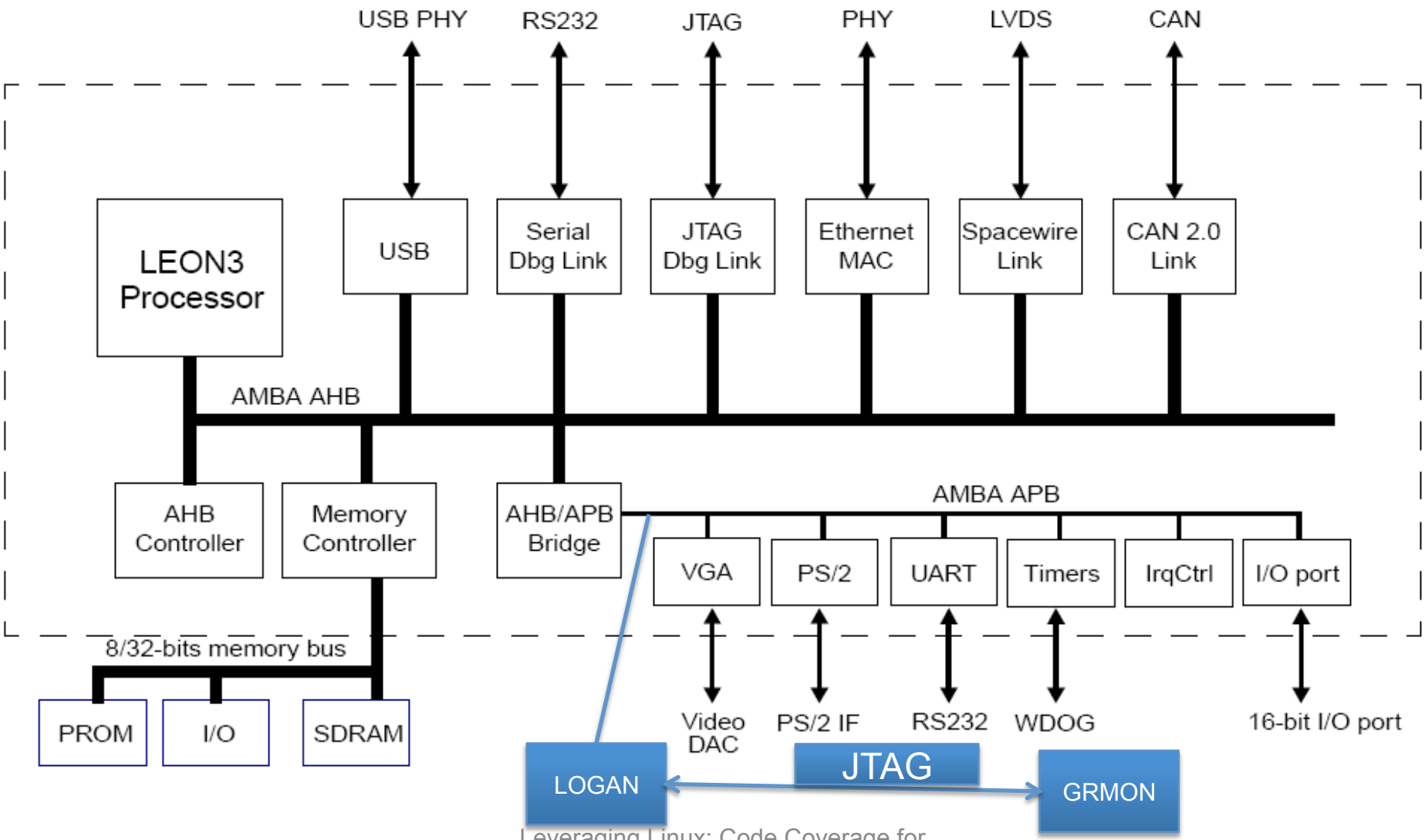
SoC Platform

- Built from Aeroflex Gaisler open-source IP
  - Aeroflex Gaisler IP used in real European Space Agengy (ESA) projects
  - All in VHDL
- Features:
  - Leon3 processor
    - OpenSPARC V8, 7-stage pipeline
  - IEEE-754 FPU
  - SPARC V8 reference MMU
  - Multiway D- and I-caches
  - DDR2 SDRAM controller/interface
  - DVI Display Controller
  - 10/100/1000 Ethernet MAC
  - PS2 Keyboard and Mouse
  - Compact Flash Interface
  - Can be fabricated to 0.18um ASIC technology.

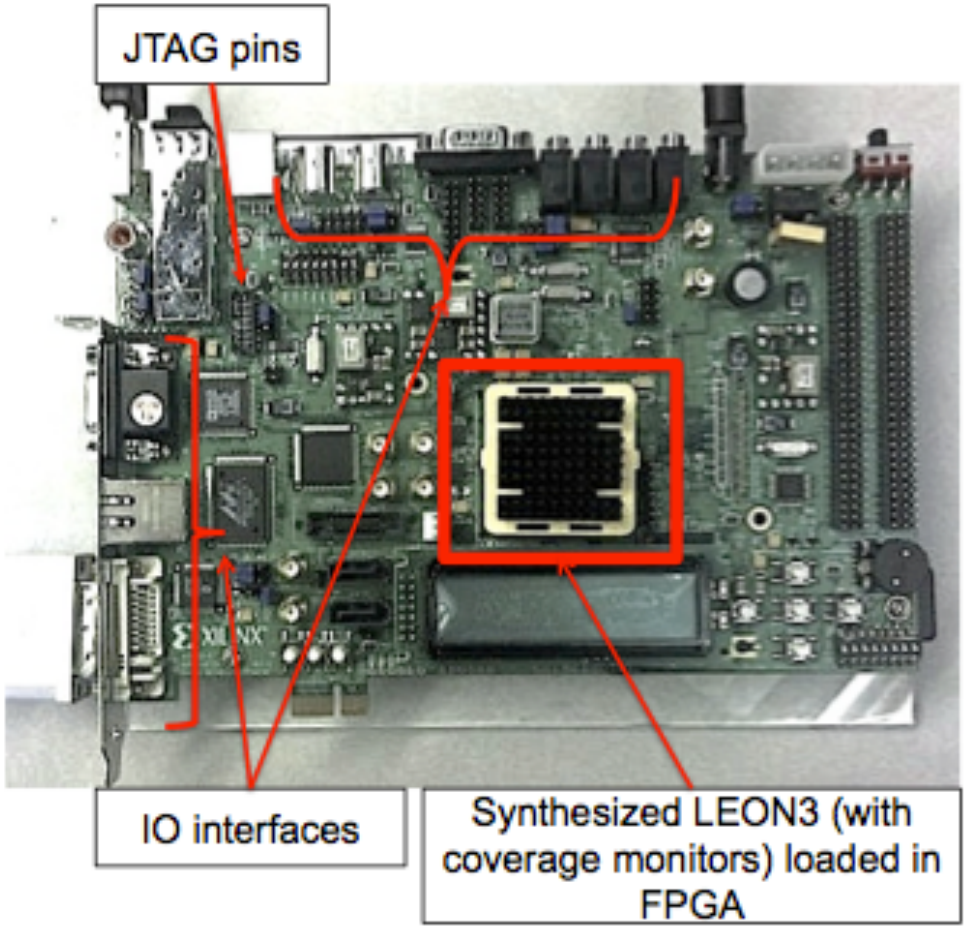


It's a notebook-on-chip

SoC Platform at Block Diagram



Xilinx University platform (XUP)

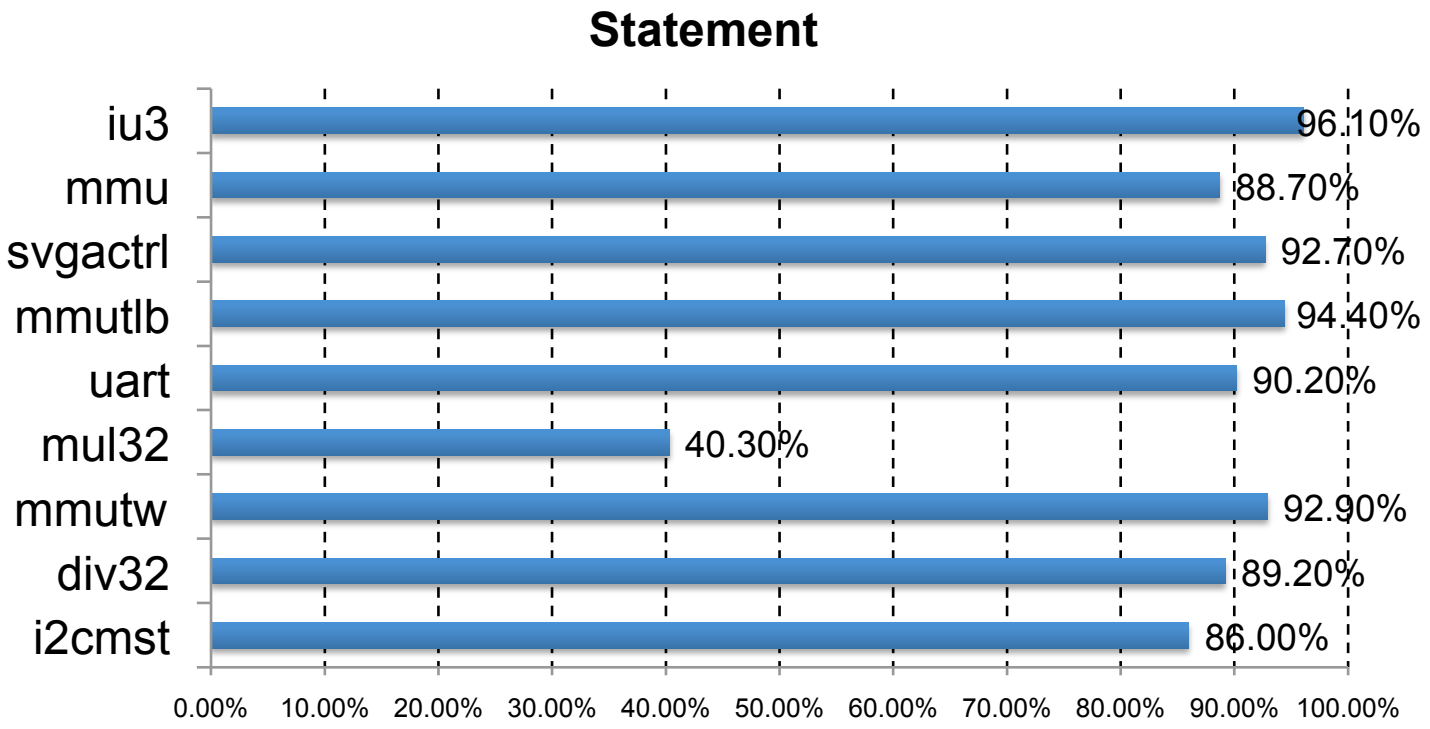


We instrumented 9 blocks from different clusters

IP Block	Lines of Code	Basic Blocks	Description
i2cmst	107	15	I^2C Master Controller from AMBA APB
div32	140	26	64-by-32 Bit Integer Divider
mmutw	179	28	MMU Table-Walk Logic
mul32	320	90	Signed/Unsigned 32-Bit Multiplier
uart	420	102	Asynchronous UART
mmutlb	421	54	MMU Translation Lookaside Buffer
svgactrl	472	104	VGA Controller
mmu	475	62	MMU Top-Level Entity
iu3	650	128	LEON3 7-Stage Integer Pipeline

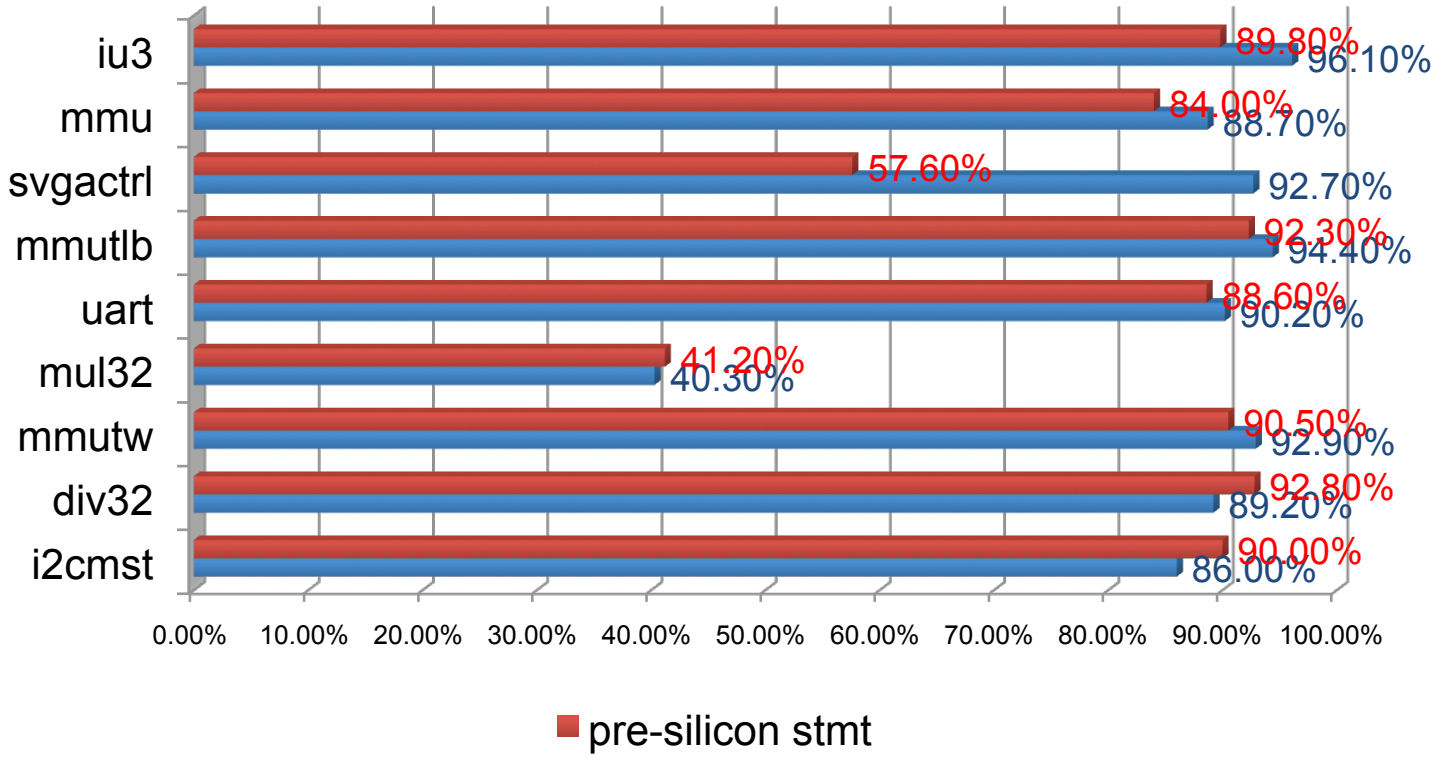
Post-Silicon Statement Coverage

- We boot Linux with kernel version 2.6.21.1, Debian etch distribution. It takes 45 seconds to boot up (at speed 75MHz) → about 3.4 billion clk cycles.

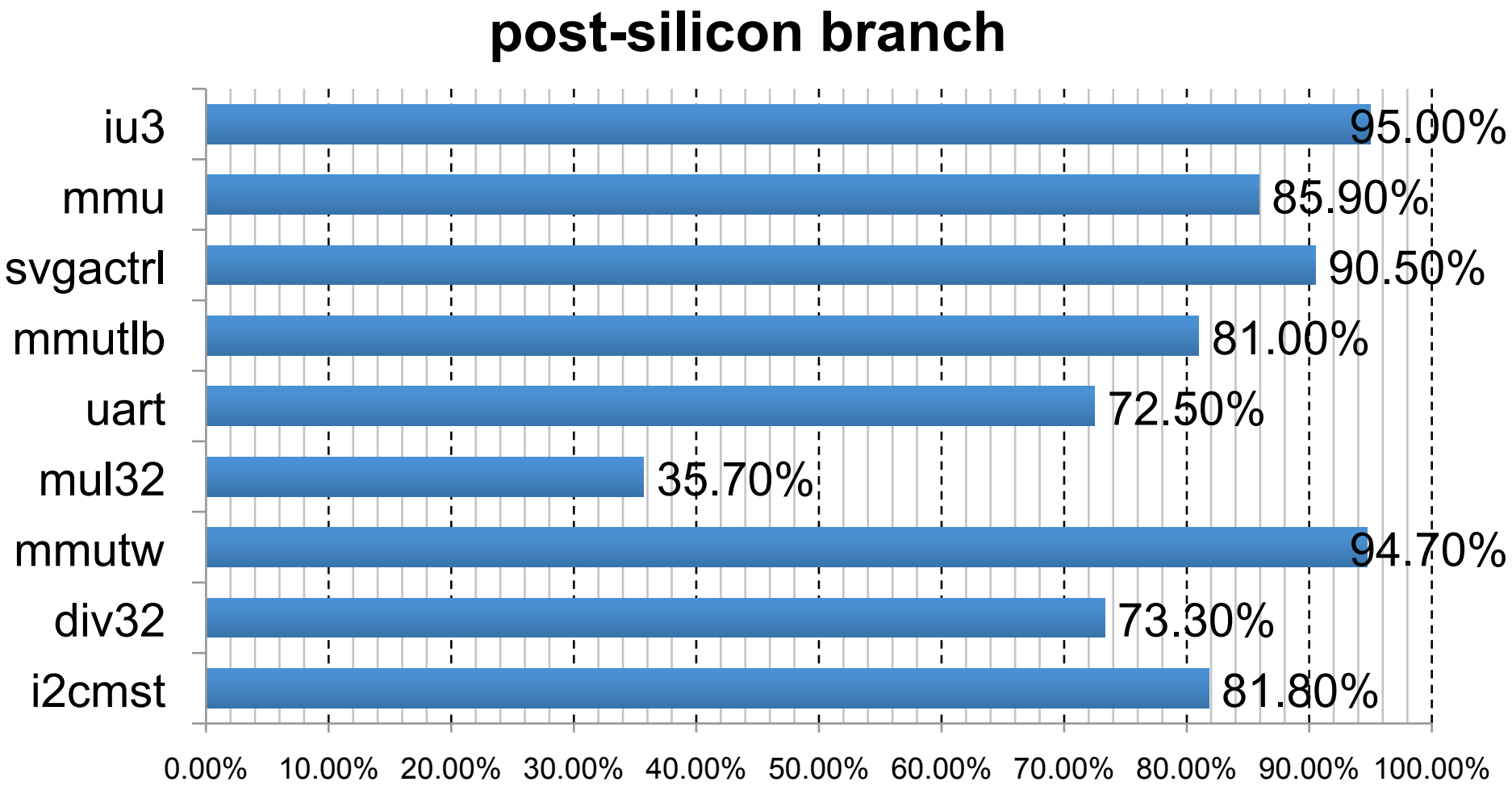


Post-Silicon vs. Pre-Silicon Statement Coverage

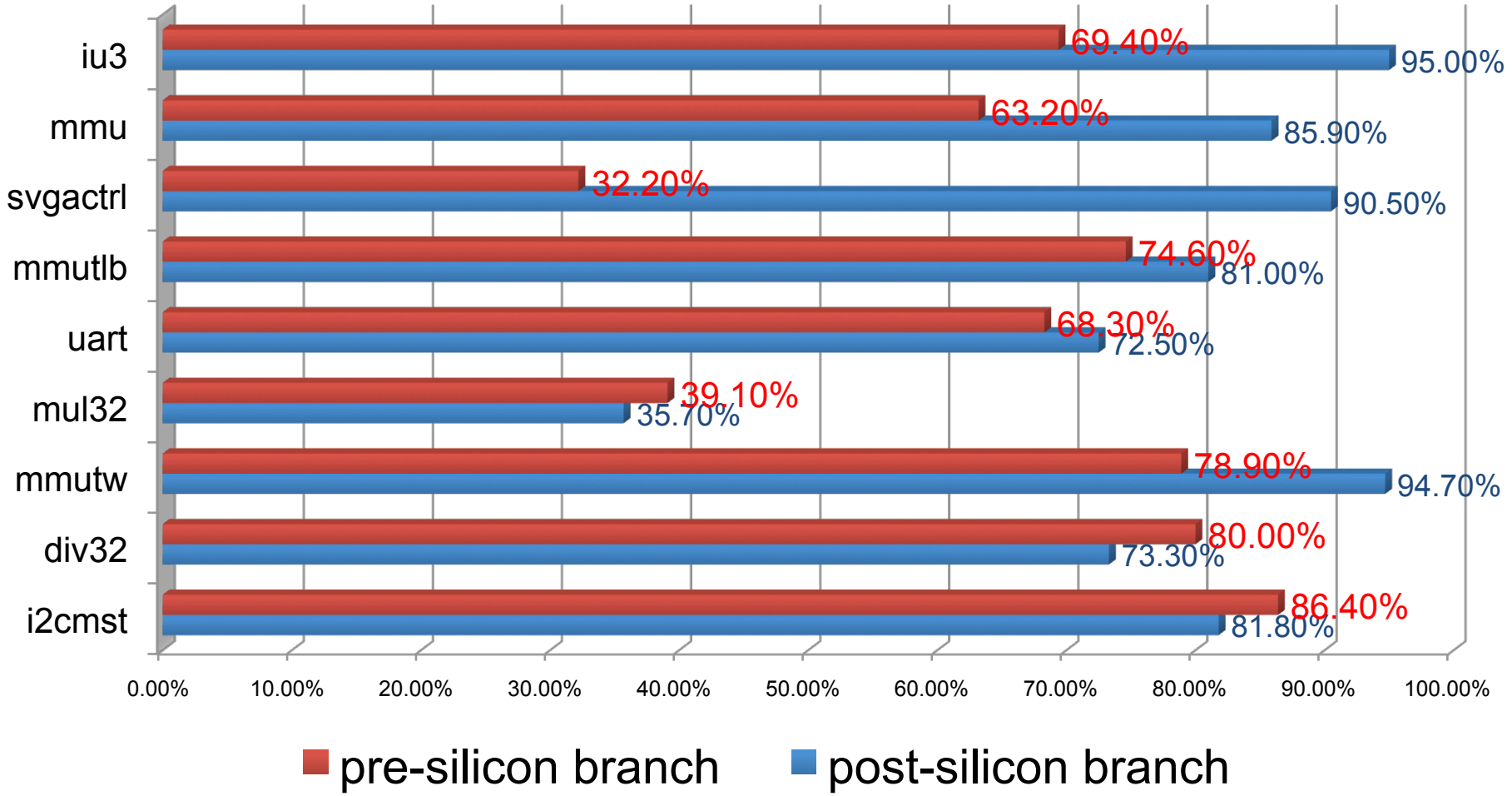
- Running Gaisler system level tests



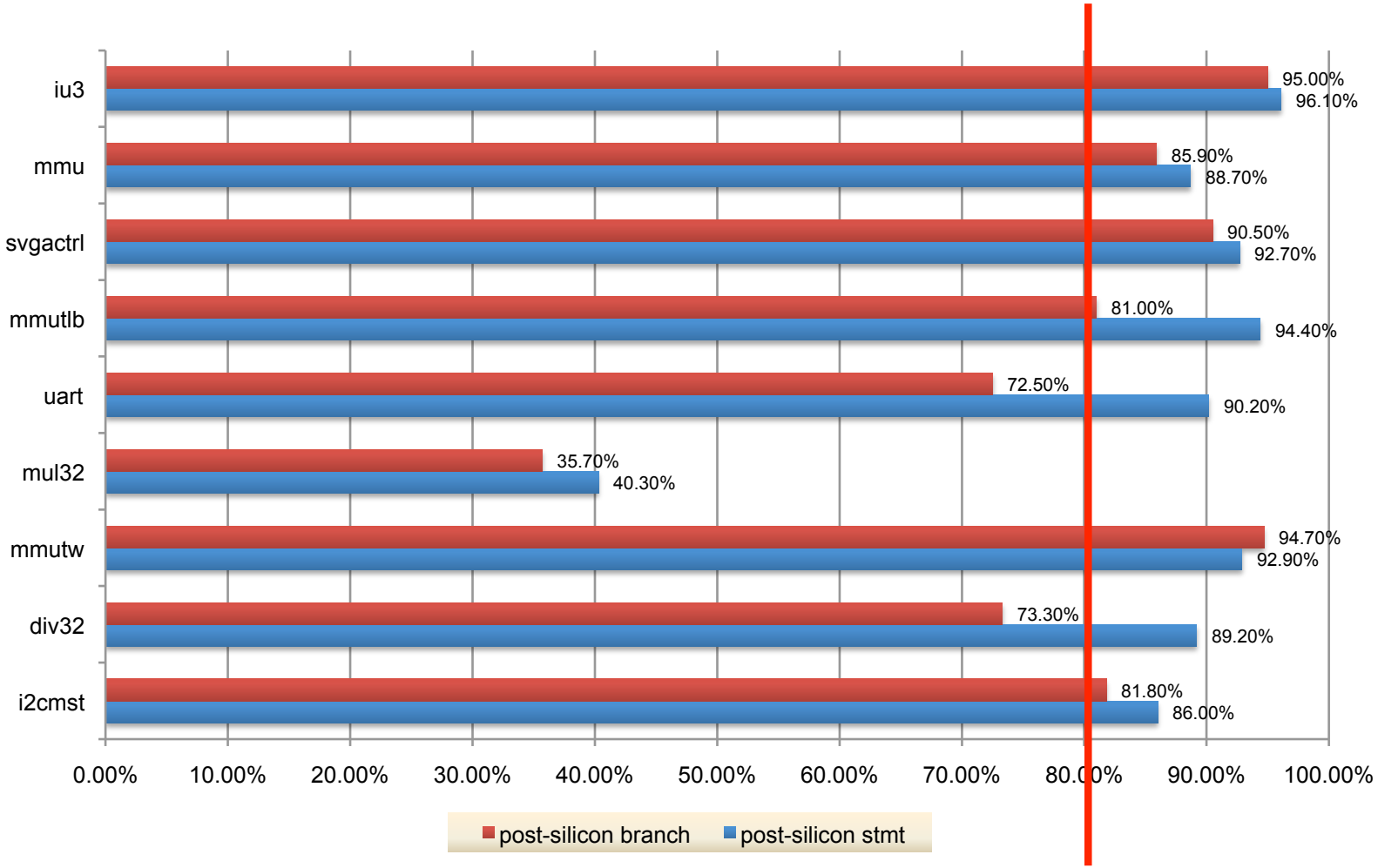
Post-Silicon Branch Coverage



Post-Silicon vs. Pre-Silicon Branch Coverage



Post-Silicon



Outline

- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Conclusion and Future Work

Conclusions

- Demonstrated a practical and an effective technique to measure coverage for post-silicon validation effectiveness.
  - Measured and compared pre- and post-silicon code coverage on a realistic SoC.
  - Results show Linux boot is a very good test to run in post-silicon, but the results also show that Linux boot is not a sufficient test to claim our chip is working completely fine.

List of Publications, Demo, and Poster Presentations

- Demo and Poster Presentation
  - University Booth, DAC 2011, San Diego.
- Conference Paper
  - M. Karimibiuki, K. Balston, A.J. Hu, and A. Ivanov. "Post-silicon code coverage evaluation with reduced area overhead for functional verification of SoC". In *IEEE International High Level Design Validation and Test Workshop (HLDVT),* pages 92 –97, Nov. 2011.
- Journal Paper
  - Kyle Balston, Mehdi Karimibiuki, Alan J Hu, Andre Ivanov, and Steve Wilton. "Post-silicon code coverage for multiprocessor system-on-chip designs". *IEEE Transactions on Computers,* to appear (accepted June 17, 2012).

Future Work

- Explore monitoring for other code coverage metrics
  - Expression and condition
- Compare code coverage results with other techniques
  - Assertion, mutation, etc.
- Apply more expensive techniques to reduce monitoring overhead, without sacrificing accuracy.
  - Software techniques intended to be lightweight, only explore graph properties of CFG.
  - In post-silicon, overhead reduction more important, could try e.g., formal analysis of the code.

End.

Outline

- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case study and Results
- Area overhead investigation
  - Area Overhead Results
  - Area overhead reduction methodology
  - Reduction results
- Conclusion and Future Work

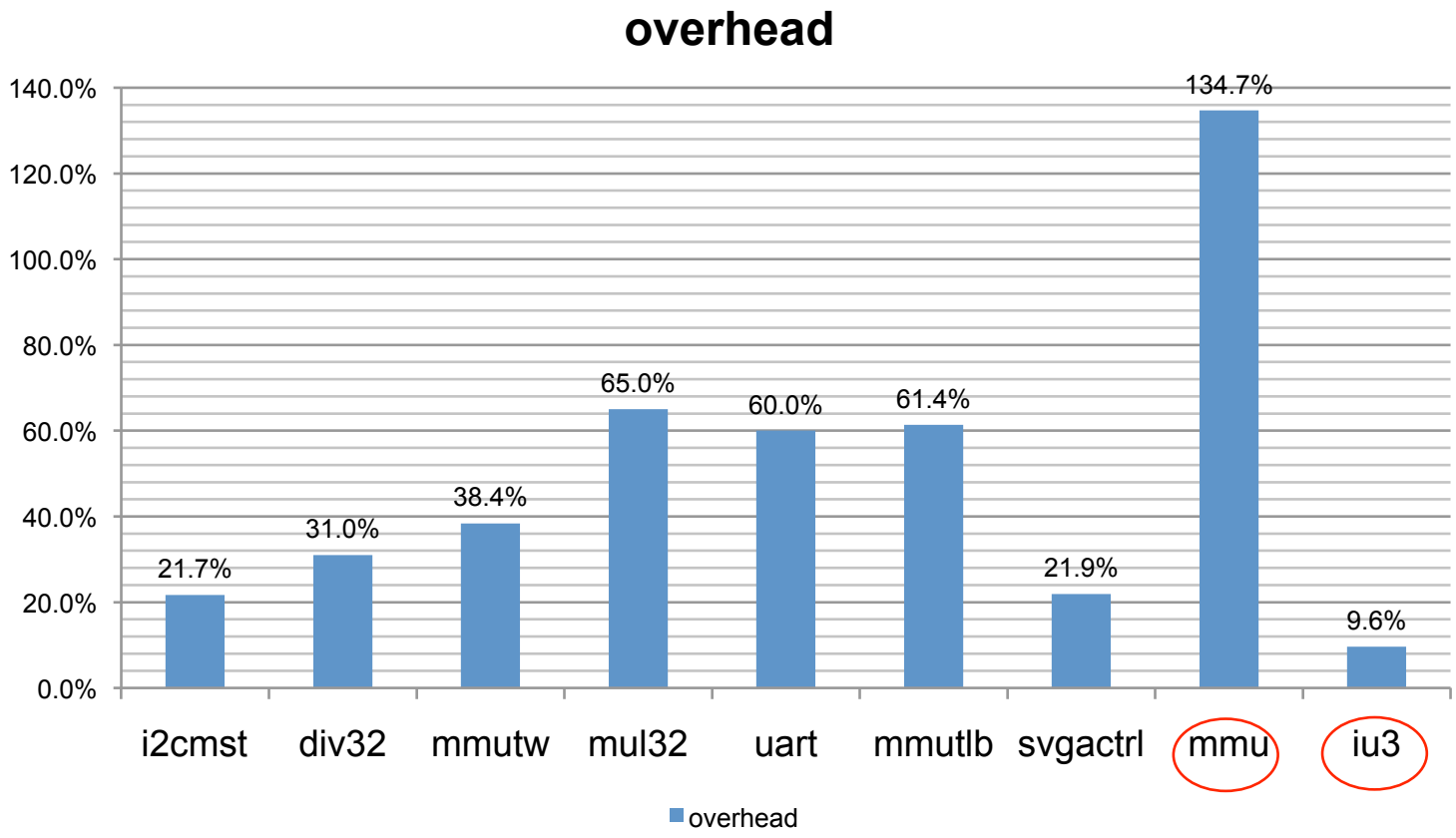
Area overhead calculation

- Area overhead is not reported in any coverage paper that we surveyed!!!
- Why area overhead is important?
  - Direct effect on cost
  - Direct effect on speed
  - We want minimal change to the intended functionality of the chip

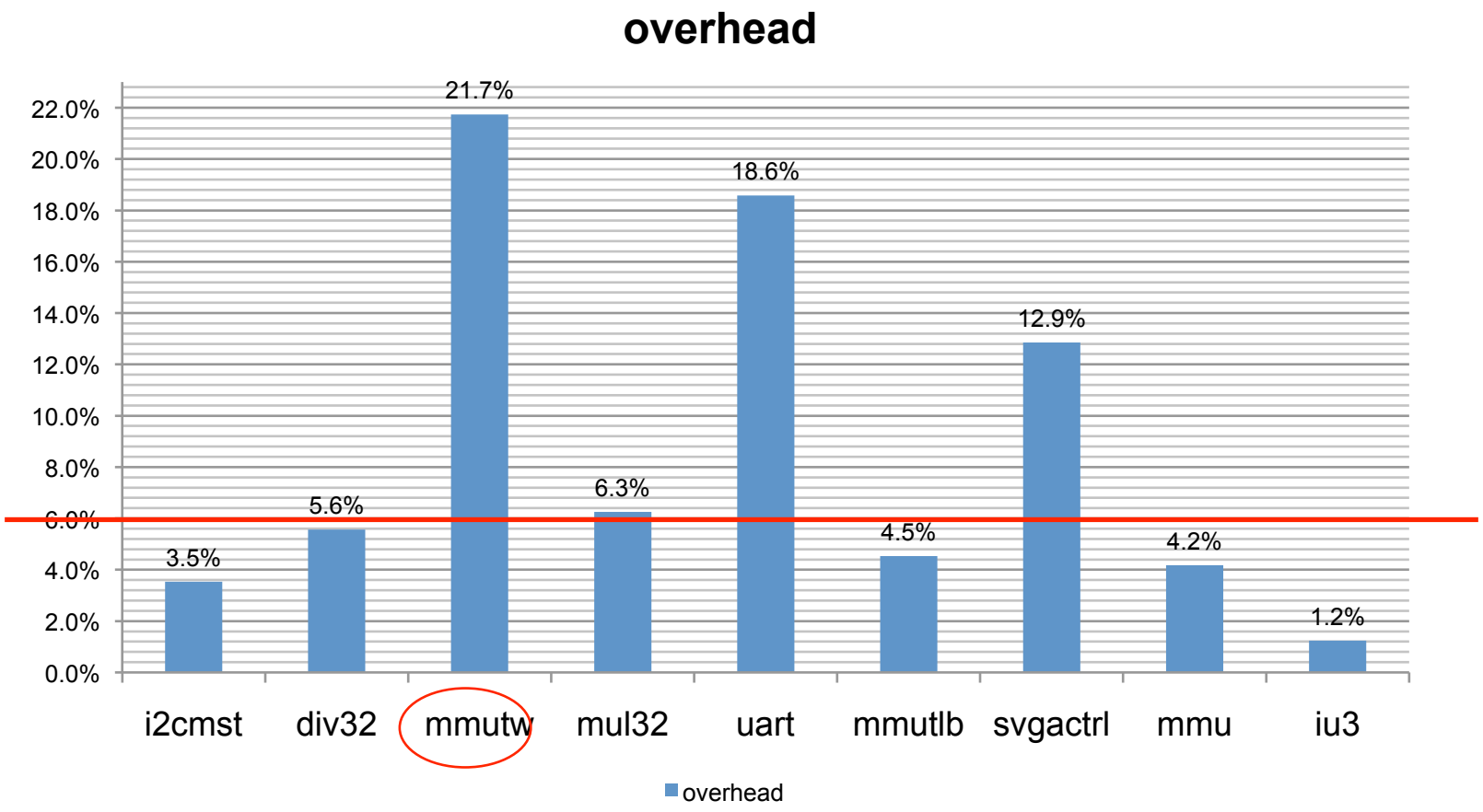
We calculate area based on two components:

- Based on FFs before routing and optimization but after synthesis
- LUTs after routing and optimization

Overhead -- FFs (Percent)



Overhead -- LUTs (Percent)



Outline

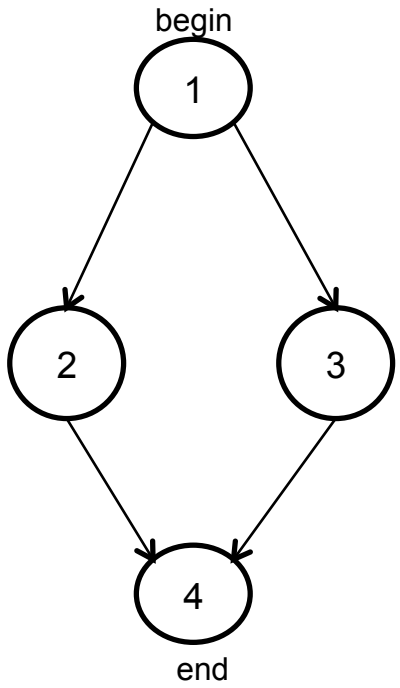
- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Area overhead investigation
  - Area Overhead Results
  - Area overhead reduction methodology
  - Reduction results
- Conclusion and Future Work

Agrawal's method

- Code coverage is a classic concept in software testing
- We use a classic technique devised by Agrawal for control flow graphs. (*Reference: Hiralal Agrawal. Dominators, super blocks, and program coverage. In Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL '94, pages 25–34, New York, NY, USA, 1994. ACM.*)
- How much overhead reduction can we achieve by using state-of-the-art techniques from the software testing world?
- The technique reduces the per-basic-block-instumentation by inspecting control flow graphs (CFG). Yet it preserves data accuracy.

Agrawal's method

```
module example
…
always @ (posedge clk)
    begin
       if(s1) then
              s2;
       else
              s3;
       endif;
              s4;
endmodule;
```

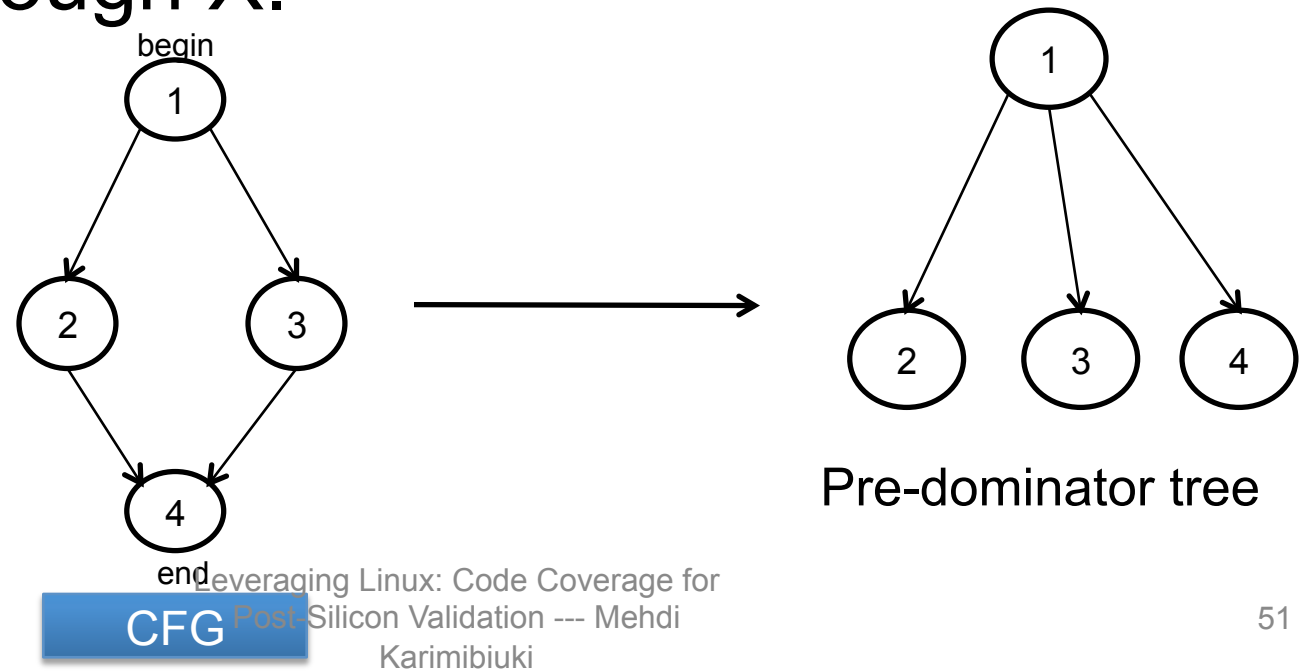


Control Flow Graph (CFG)

How does it works: two relations: pre-dominance and post-dominance

- Definition 1: Basic block X *pre-dominates* basic block Y if every path from begin to Y goes through X.

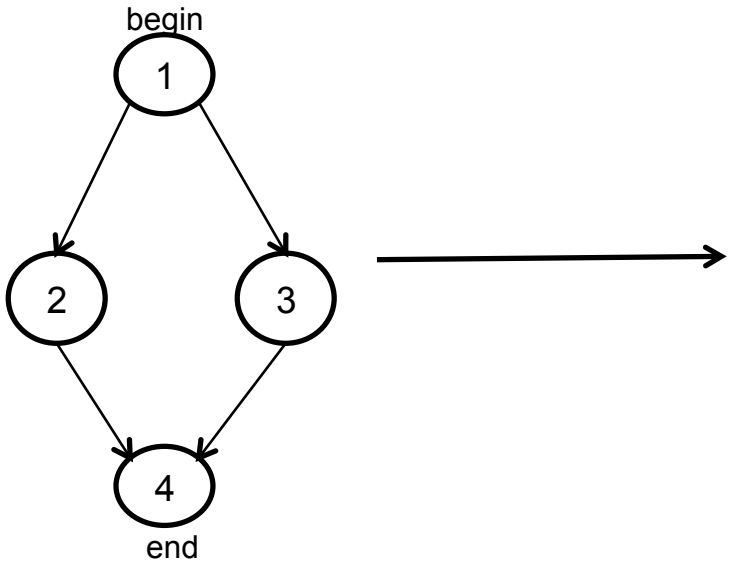
```
module example
…
always @ (posedge clk)
    begin
      if(s1) then
            s2;
      else
            s3;
      endif;
            s4;
endmodule;
```



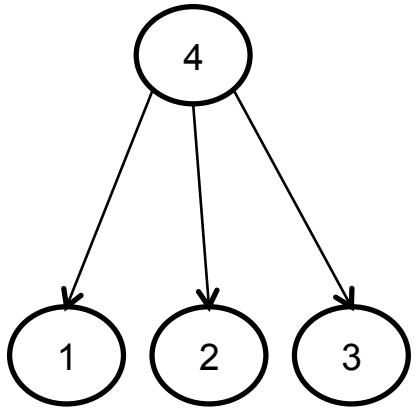
How does it works:

- Definition 2: Basic block X *post-dominates* basic block Y if every path from Y to exit goes through X.

```
module example
…
always @ (posedge clk)
    begin
      if(s1) then
            s2;
      else
            s3;
      endif;
            s4;
endmodule;
```

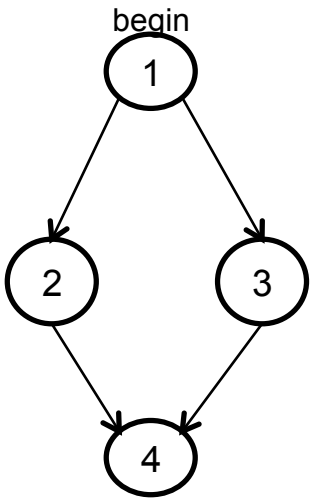


CFG

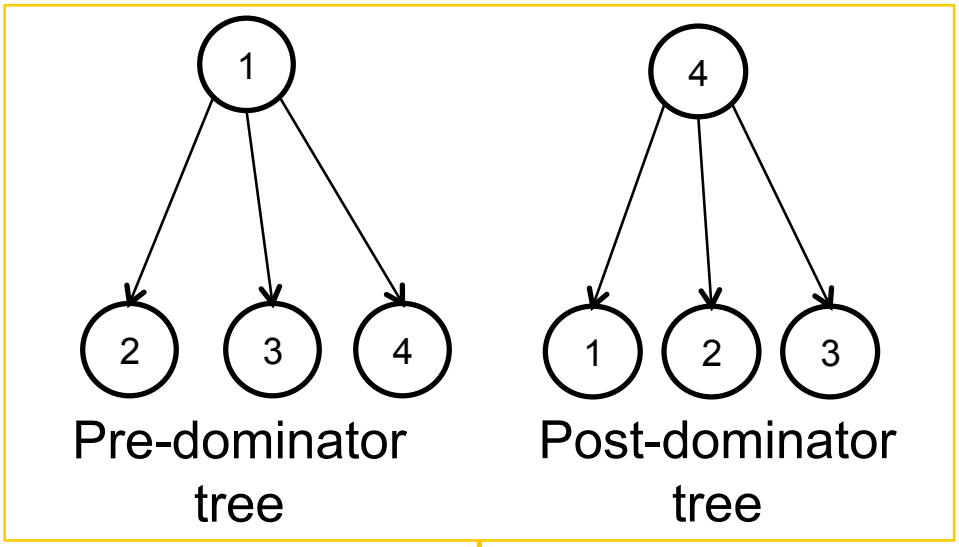


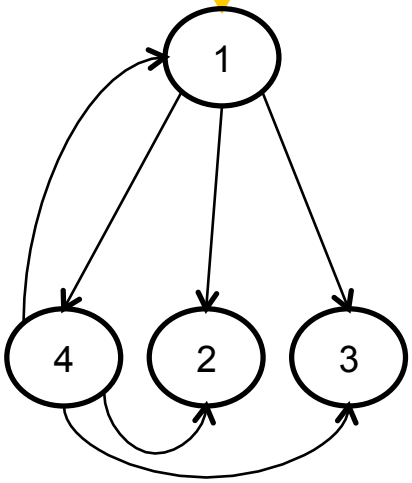
Post-dominator tree

```
module example
…
always @ (posedge
clk)
    begin
      if(s1) then
            s2;
      else
            s3;
      endif;
            s4;
endmodule;
```

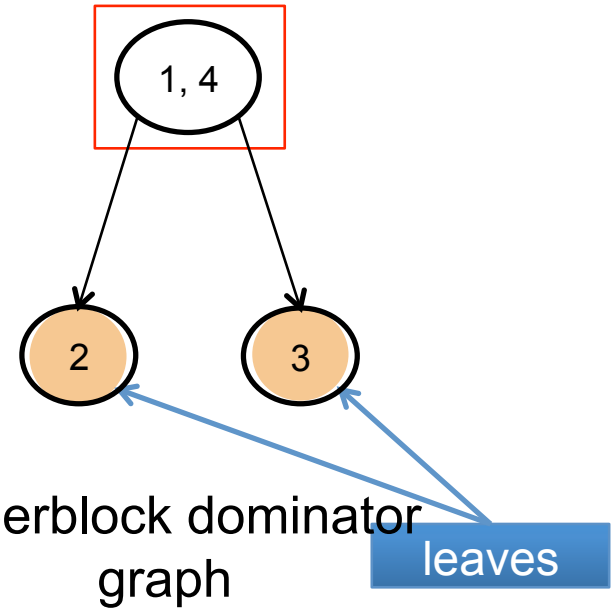


CFG





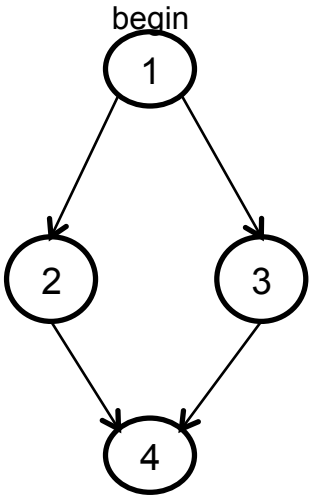
Basic block dominator graph



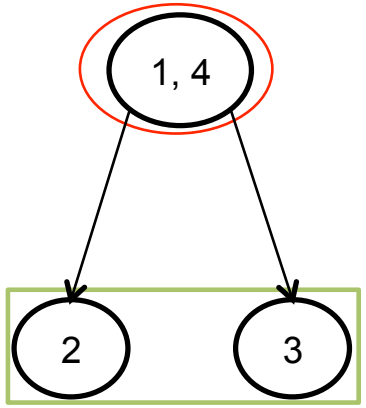
Superblock dominator graph

So far…50% overhead savings

```
module example
…
always @ (posedge
clk)
    begin
      if(s1) then
            s2;
      else
            s3;
      endif;
            s4;
endmodule;
```



CFG



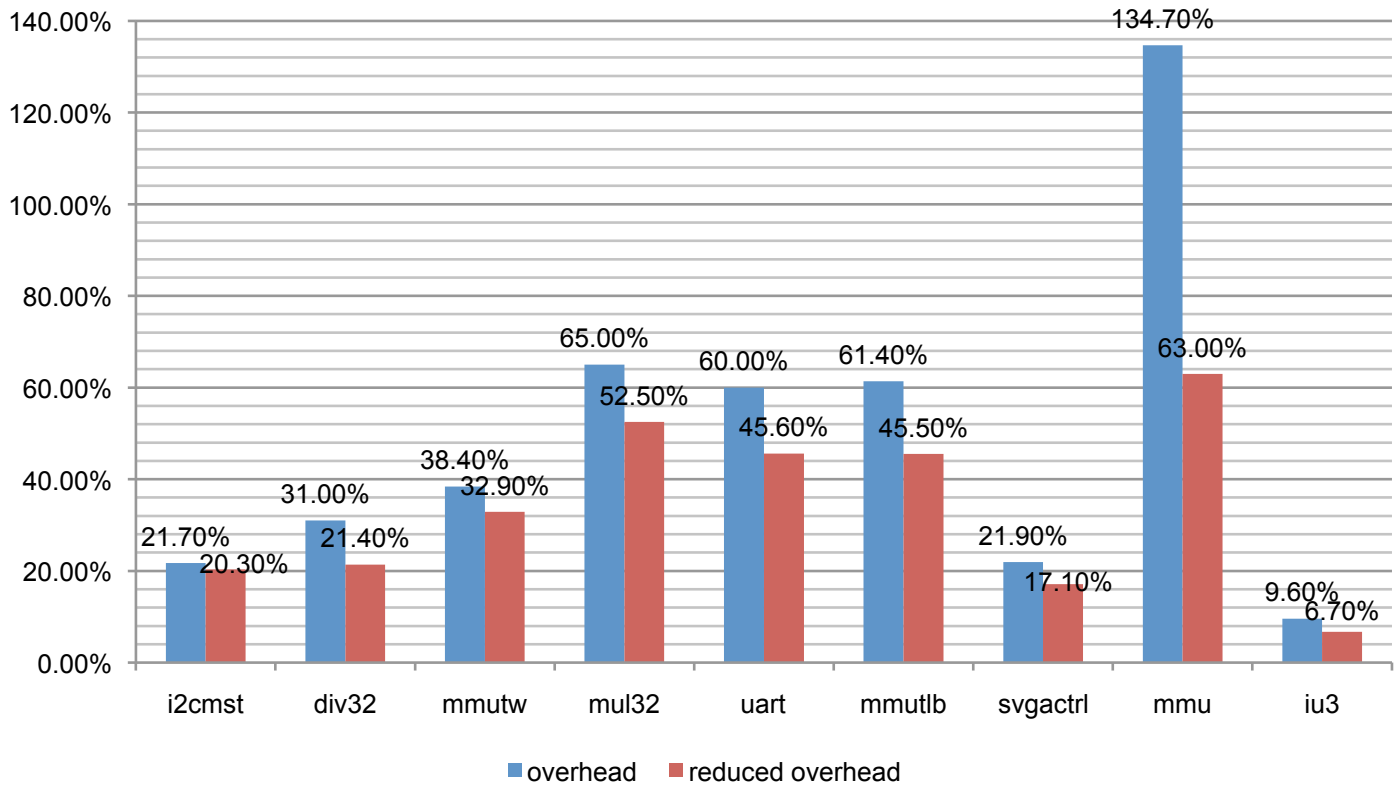
Superblock dominator graph

Overall, 50% area saving in this example

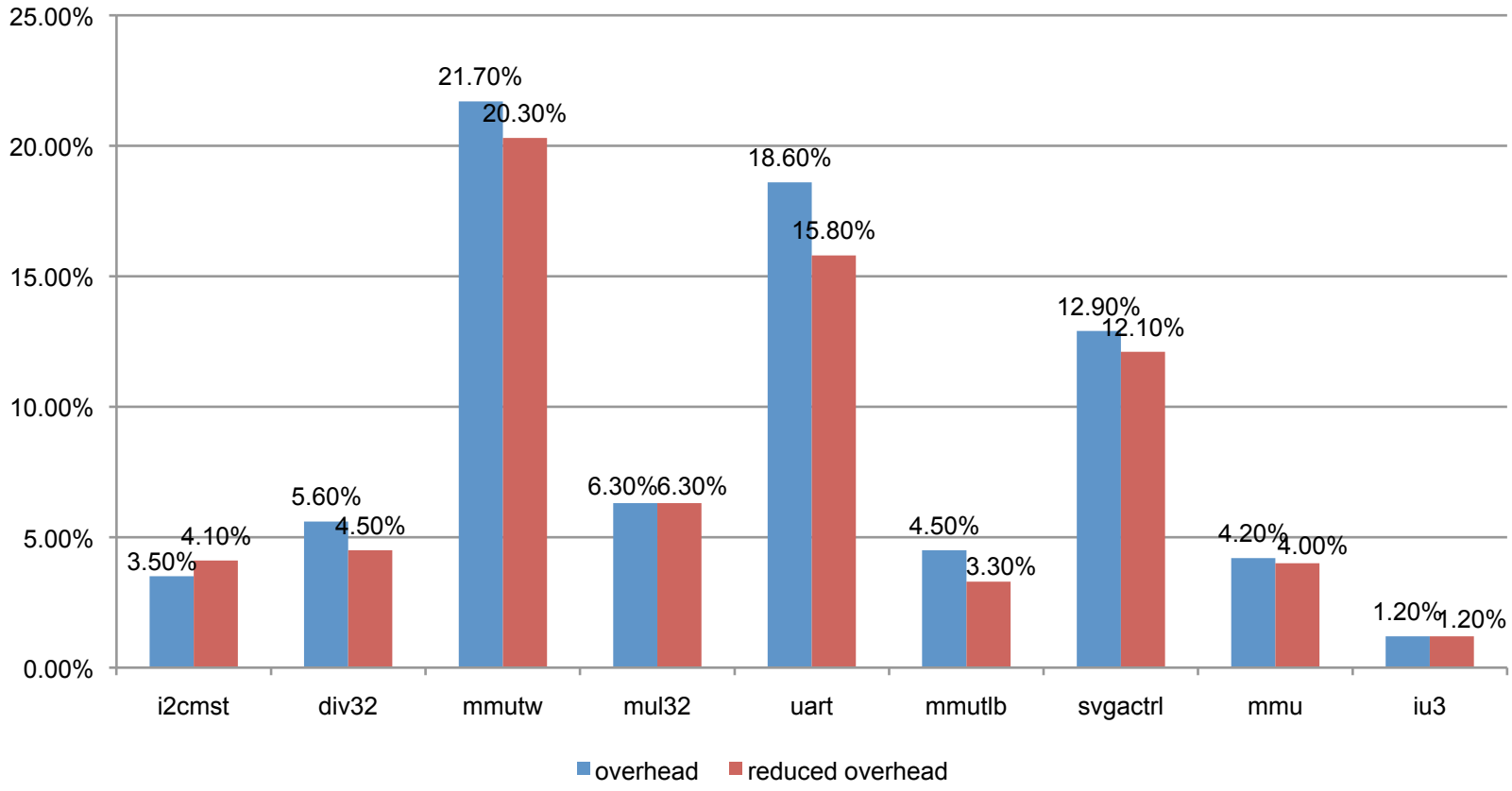
Outline

- Motivation
  - Why post-silicon validation needed?
  - Why post-silicon coverage?
- Proposed techniques for post-silicon coverage
- Using Linux for post-silicon code coverage
  - Why code coverage? Code Coverage Types
  - Instrumentation
  - Case Study and Results
- Area overhead investigation
  - Area Overhead Results
  - Area overhead reduction methodology
  - Reduction results
- Conclusion and Future Work

FF overhead reduced (Percent)



LUT overhead reduced (Percent)



Agrawal's Algorithm (POPL 1994)

- Merge to form dominance graph.
- Find strongly connected components in graph
  - Every basic block in SCC dominates others in SCC.
  - Therefore, basic block covered iff others covered.
  - Therefore, one flag per SCC.

tree

- All nodes are minimally connected
- N nodes and n-1 edges
- No more than one edge to a node

Linearly ordered

- a linearly ordered or totally ordered group is an ordered group G such that the order relation "≤" is total. This means that the following statements hold for all a, b, c ∈ G:
- if a ≤ b and b ≤ a then a = b (antisymmetry)
- if a ≤ b and b ≤ c then a ≤ c (transitivity)
- a ≤ b or b ≤ a (totality)
- the order relation is translation invariant: if a ≤ b then a + c ≤ b + c and c + a ≤ c + b.

Partial order vs total order

- Partial order elements not comparable in general but comparable to each other
  - A relation R on a set S is called a partial order if it is reflexive, antisymmetric and transitive. A set S together with a partial ordering R is called a partially ordered set or poset for short and is denoted
- Total order elements are comparable in terms of less than, greater than, etc.