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?

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...

**Pre-Silicon Logic Bugs per Generation**

- Pentium: 800
- Pentium Pro: 2,240
- Pentium 4: 7,855
- Next Generation: ?

**Bug Count Reflects Chip Complexity**

- Pre-Silicon Bugs
- Transistors

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/moore'slaw/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.
    -- Slow 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.
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
We can check for validation effectiveness with Coverage.

- This is similar to what is being done in pre-silicon verification \(\Rightarrow\) 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**

- **IBM POWER7**

→ 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
  – 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;
S3;
if (s4) begin
  SFlag1=1;
s5;
s6;
s7;
else
  SFlag2=1;
  s8;
s9;
end if;
s9;
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
    s5;
    s6;
    s7;
end else
    BFlag0=1;
else
    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 Agency (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.
SoC Platform at Block Diagram

LEON3 Processor

USB PHY
RS232
JTAG
PHY
LVDS
CAN

USB
Serial Dbg Link
JTAG Dbg Link
Ethernet MAC
Spacewire Link
CAN 2.0 Link

AHB AHB

AHB Controller
Memory Controller
AMBA APB

AHB/APB Bridge

VGA
PS/2
UART
Timers
IrqCtrl
I/O port

8/32-bits memory bus

PROM
I/O
SDRAM

Video DAC
PS/2 IF
RS232
WDOG

LOGAN

JTAG

GRMON

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki

2/22/2013
Xilinx University platform (XUP)
We instrumented 9 blocks from different clusters

<table>
<thead>
<tr>
<th>IP Block</th>
<th>Lines of Code</th>
<th>Basic Blocks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2cmst</td>
<td>107</td>
<td>15</td>
<td>I²C Master Controller from AMBA APB</td>
</tr>
<tr>
<td>div32</td>
<td>140</td>
<td>26</td>
<td>64-by-32 Bit Integer Divider</td>
</tr>
<tr>
<td>mmutw</td>
<td>179</td>
<td>28</td>
<td>MMU Table-Walk Logic</td>
</tr>
<tr>
<td>mul32</td>
<td>320</td>
<td>90</td>
<td>Signed/Unsigned 32-Bit Multiplier</td>
</tr>
<tr>
<td>uart</td>
<td>420</td>
<td>102</td>
<td>Asynchronous UART</td>
</tr>
<tr>
<td>mmutlb</td>
<td>421</td>
<td>54</td>
<td>MMU Translation Lookaside Buffer</td>
</tr>
<tr>
<td>svgactrl</td>
<td>472</td>
<td>104</td>
<td>VGA Controller</td>
</tr>
<tr>
<td>mmu</td>
<td>475</td>
<td>62</td>
<td>MMU Top-Level Entity</td>
</tr>
<tr>
<td>iu3</td>
<td>650</td>
<td>128</td>
<td>LEON3 7-Stage Integer Pipeline</td>
</tr>
</tbody>
</table>
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

- **iu3**: 95.00%
- **mmu**: 85.90%
- **svgactrl**: 90.50%
- **mmutlb**: 81.00%
- **uart**: 72.50%
- **mul32**: 35.70%
- **mmutw**: 94.70%
- **div32**: 73.30%
- **i2cmst**: 81.80%

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki
Post-Silicon vs. Pre-Silicon Branch Coverage

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki

2/22/2013
Leveraging Linux: Code Coverage for Post-Silicon Validation

Mehdi Karimibiuki

2/22/2013
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

• Journal Paper
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)

<table>
<thead>
<tr>
<th>Component</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>i2cmst</td>
<td>21.7%</td>
</tr>
<tr>
<td>div32</td>
<td>31.0%</td>
</tr>
<tr>
<td>mmutw</td>
<td>38.4%</td>
</tr>
<tr>
<td>mul32</td>
<td>65.0%</td>
</tr>
<tr>
<td>uart</td>
<td>60.0%</td>
</tr>
<tr>
<td>mmutlb</td>
<td>61.4%</td>
</tr>
<tr>
<td>svgactrl</td>
<td>21.9%</td>
</tr>
<tr>
<td>mmu</td>
<td>134.7%</td>
</tr>
<tr>
<td>iu3</td>
<td>9.6%</td>
</tr>
</tbody>
</table>

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki
Overhead -- LUTs (Percent)

- i2cmst: 3.5%
- div32: 5.6%
- mmutw: 21.7%
- mul32: 6.3%
- uart: 18.6%
- mmutlb: 4.5%
- svgactrl: 12.9%
- mmu: 4.2%
- iu3: 1.2%

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki
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
• 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-instrumentation by inspecting control flow graphs (CFG). Yet it preserves data accuracy.
module example
...
always @ (posedge clk)
  begin
    if(s1) then
      s2;
    else
      s3;
    endif;
    s4;
  end
endmodule

Agrawal’s method

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.

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

Pre-dominator tree

---
Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki

2/22/2013
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;
module example
...
always @ (posedge clk)
  begin
    if(s1) then
      s2;
    else
      s3;
    endif;
    s4;
  end
endmodule;

So far...50% overhead savings

Basic block dominator graph

CFG

Pre-dominator tree

Post-dominator tree

Superblock dominator graph

leaves

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibakht

2/22/2013
module example

... always @ (posedge clk)

begin
    if(s1) then
        s2;
    else
        s3;
    endif;
endmodule

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)

Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki

2/22/2013
Leveraging Linux: Code Coverage for Post-Silicon Validation --- Mehdi Karimibiuki
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 $a, b, c \in G$:
  - if $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry)
  - if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)
  - $a \leq b$ or $b \leq a$ (totality)
  - the order relation is translation invariant: if $a \leq b$ then $a + c \leq b + c$ and $c + a \leq c + b$. 
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.