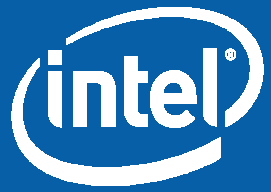


Smart-Lint:
Improving the Verification Flow

Itai Yarom

Design & Verification Technical Lead
Intel Corp.

Viji Patil

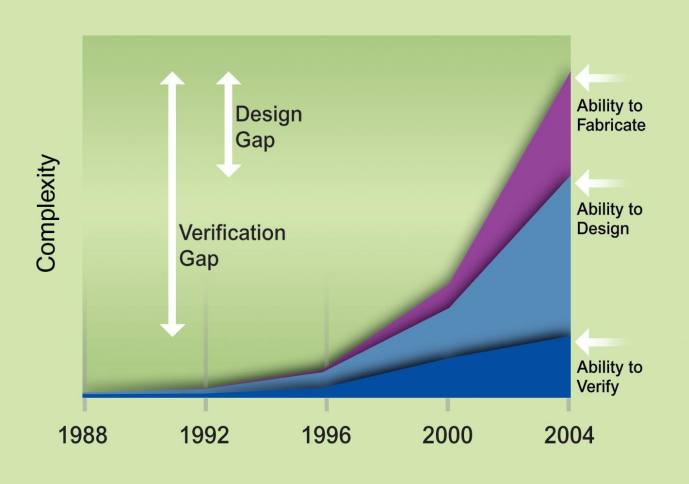
CAD Engineer

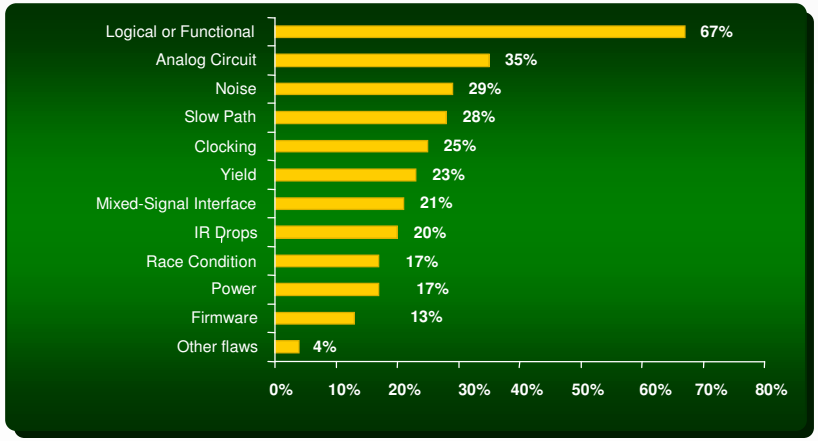
Intel Corp.

What are the Challenges we are Facing?

- How can I identify real problems from thousands of lint violations?
- How many paths with no synchronization are a real problem?
- How can I improve my level of guaranty in the verification environment?
- How can I verify false-paths and multi-cycle paths?

The Design & Verification Gap





Collett Intl. 2003 Survey

The number of transistors on a chip increases approximately 58% per year, according to Moore's Law.

The design productivity, facilitated by EDA tool improvements, grows only 21% per year.

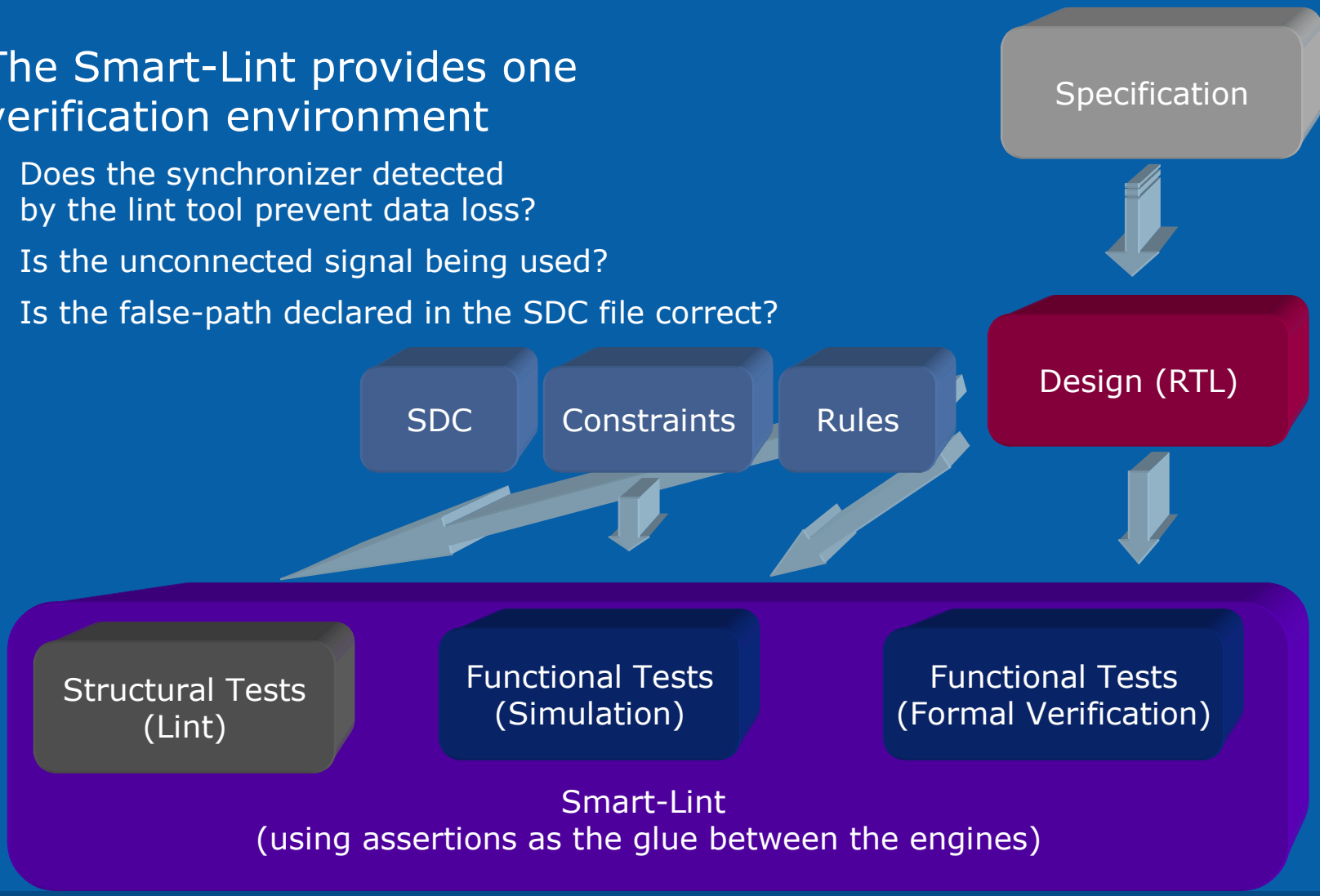
These numbers have held constant for the last two decades.

Roger Allan, Think Small: Can You Meet The Design Challenges At 90 nm And Below?, Electronic Design Journal, Feb 2005.

Smart-Lint in a Nutshell

The Smart-Lint provides one verification environment

- Does the synchronizer detected by the lint tool prevent data loss?
- Is the unconnected signal being used?
- Is the false-path declared in the SDC file correct?



2006

The Smart-Lint Flow

What is Smart-Lint and how can it help me?

Flow

- Lint tools go over the design and find issues that related to the structure of the design.
- The Lint tool generates checkers for further verification.
- The results of the lint and the functional verification flow are presented together.

Advantages:

- Provide enhance verification flow.
- Provide better lint environment.
  - Reduce the false violations
  - Handle additional checks
- Provide additional checkers.
  - Coverage and waivers

Examples for the Smart-Lint Benefits

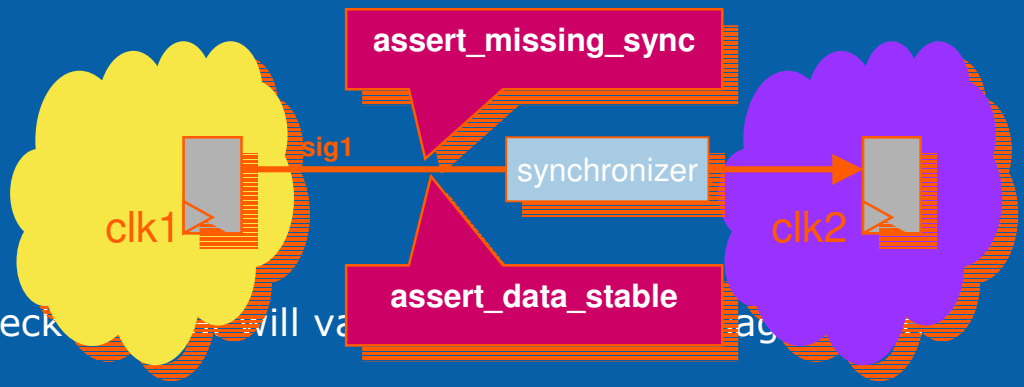
- Clock Domain Crossing (CDC)
- Improving the coverage
- Handling waivers
- Back to Lint

2006

Example – synchronization data lost

The Lint tool identifies clock domain crossing (CDC) paths.

The Lint tool checks the presence of synchronizers in those paths.



The Lint tool generates assertions (check … will va… ag…
synchronizer

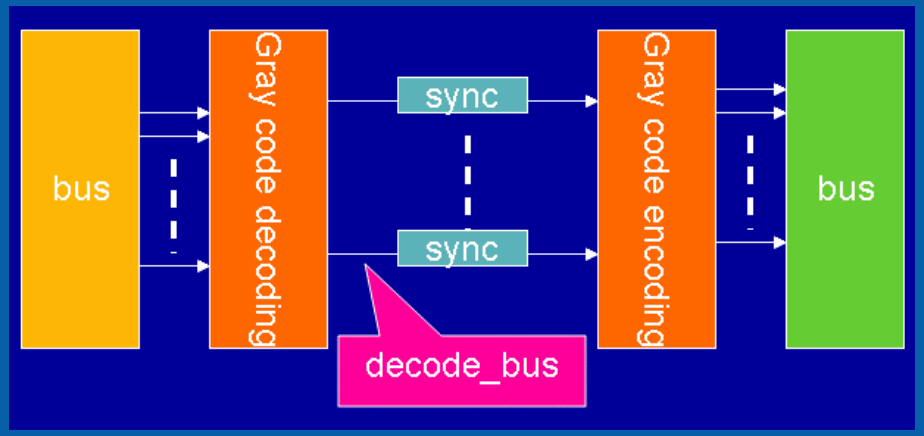
assert_missing_sync sig1_check (.clk(clk1), .rst(!rst_l), .in(sig1), .out(sig1_new))

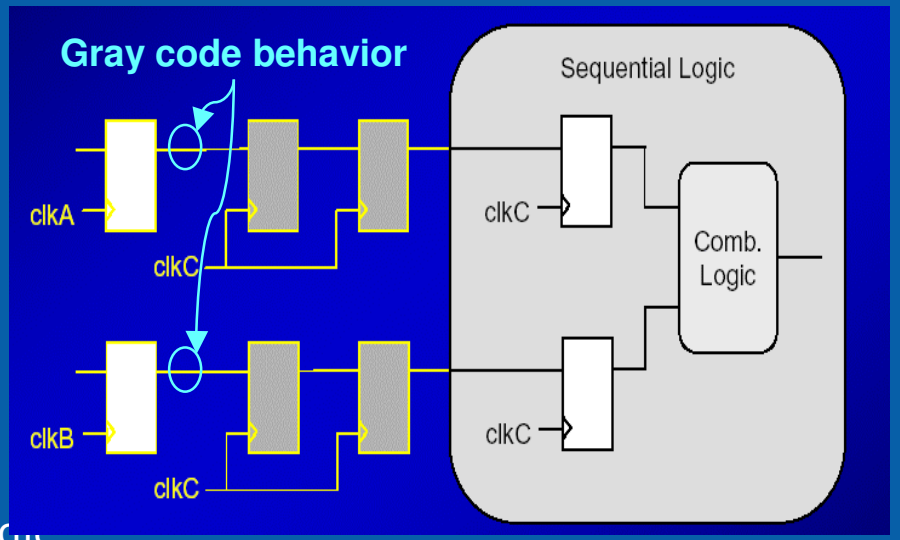
- This checker checks whether signal sig1 is static and therefore doesn't need synchronizer.

assert_data_stable #(0,`clk1_clk2_min) sync1 (.clk(clk1), .rst(!rst_l), .data(sig1))

- The checker makes sure that the input of the synchronizer is stable for `clk1_clk2_min clock cycles.

Example – Gray code validation





The lint tool can identify the gray code logic

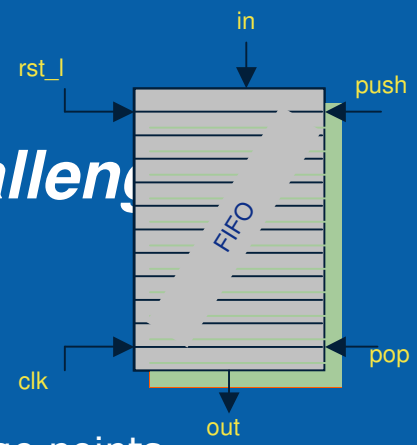
- But it cannot verify its correct behavior

Correct behavior can be verified using formal or simulation tools:

```
assert_one_hot #(0,SIZE) aoh1 (clk, !rst_l,
                               decoded_bus ^ prev_decoded_bus);
always @(posedge clk) prev_decoded_bus <= decoded_bus;
```

Structural Coverage

A different way to address the coverage challeng



How would you measure the verification progress?

- Code coverage – measuring the level of code and statement exercising.
- Functional coverage – measuring the coverage of the functional coverage points.

Is this enough?

- How can we measure the FIFO usage in a system with an arbitration of 3 clients?
- How Can we measure whether the FIFO is full/empty?

Structural coverage is adding coverage points based on the structure of the design

- Has FIFO been empty? Has FIFO been full?
- Has FIFO reached high-water mark?
- Number of enqueues/dequeues
- Maximum FIFO entries

How to Handle Waivers?

How can I verify that the waiver is correct?

For example:

- This signal is set by the configuration, therefore we don't need to synchronize it.
- We don't need to assign a value to this signal, since this only can occur in this scenario.

The flow:

- Generate a checker that verifies the waiver assumption.
- Associate the checker to the waiver.
  - This will provide me the connection between the checker, the waiver and the waived violation.
  - Failure in the checker will mean a problem with the waiver.

Lint & Smart-Lint

The traditional questions:

- Which of the violations I should debug first?
- How can I reduce the amount of violations?

With smart-lint we can verify that the lint violations will cause a functional error (using simulation and formal technique).

Automatic Assertions	Class	Checker	Monitor	Fully automatic
Dead code	Lint	Yes	No	Yes
Bus contention	Lint	Yes	No	Yes
FSM deadlock	Lint	Yes	No	Yes
Uninitialized FSM variables	Lint	Yes	No	Yes*
Unreachable FSM states	Lint	Yes	No	Yes
Full-case synopsys pragma violation	Lint	Yes	No	Yes
Parallel-case synopsys pragma violation	Lint	Yes	No	Yes
Uninitialized memory	Lint	Yes	No	Yes
FIFO monitor	Monitor	No	Yes	No
SRAM monitor	Monitor	No	Yes	No
Missing synchronizers	CDC	Yes	Yes	Yes
Gray-code	CDC	Yes	Yes	Yes*
Handshake	CDC	Yes	Yes	Yes*
Multi-cycle path	TCV**	Yes	No	Yes
False-path	TCV**	Yes	No	Yes

* Sometimes user intervention is needed.
** Timing Constraint Verification (TCV)

What about violations that we haven't report because of the high amount of false violations?

- Do we have a contention on the bus?
- Does the case is full? Parallel?
- Is it ok that the FSM does not have a defined default state?

Can we use lint to improve the probability to identify bugs?

- Un-initialized FSM variable.
- Unreachable FSM state. FSM deadlock.

Experience @ Intel

Example: Sunrise lake FSENG unit

Issue: There were about 1000 violations of different kinds starting from two signals, namely, *protocol* and *mmr_proto_rst_n*.

- "Combinational logic before synchronizer"
- "Signal feeds to the asynchronous reset port of the receiving register have no synchronizer"

Design data:

- *mmr_proto_rst_n* is intentionally OR'ed before driving the flop's reset to maintain an asynchronous assertion. However synchronous deassertion is guaranteed.
- The signal *protocol* is static (it is a fuse option).There was a need to filter these out.

Solution: We want to:

- Waive those signals from the report of those particular violations.
- Be sure that the waiver is correct.

```
assert_guaranty_by_design deassertion_ok (.in(mmr_proto_rst_n_in), out(mmr_proto_rst_n));
assert_missing_sync protocol_sig (.in(protocol_in), .out(protocol),.clk(clk),.rst(!rst_l));
```

Summary

- Smart-Lint provides one environment for addressing the design & verification gaps
- Smart-Lint provides a solution that is greater than its parts (the verification tools)
- Good experience in Intel
  - Reduce TTM by automatically identifying the major violations
  - Improve the level of guaranty in the correctness of the design
- More details in the paper


intel
Leap ahead™