



February 24-26, 2009

Adding Assertions At the Last Minute (lessons learned a little too late)

Stuart Sutherland
SystemVerilog Wizard
Sutherland HDL, Inc.



*Training engineers
to be HDL wizards*

www.sutherland-hdl.com

About the Presenter...

▶ Stuart Sutherland

■ SystemVerilog design and verification consultant

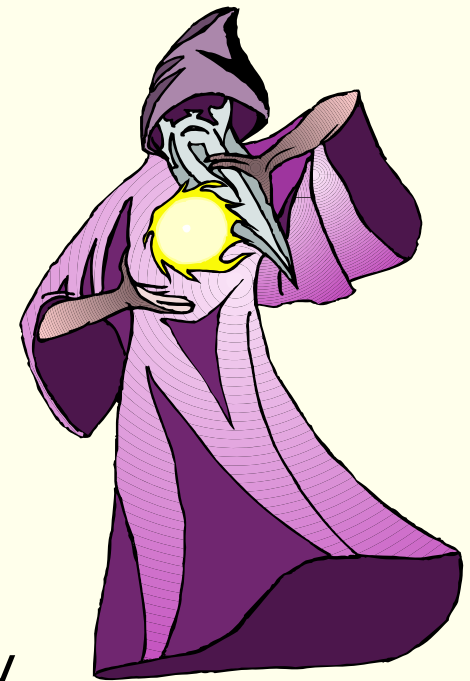
- Founder and President of Sutherland HDL, Inc.
- Specializes in providing Verilog/SystemVerilog training
- Involved in hardware design & verification since 1982
- Has been using Verilog since 1988
- Bachelors degree in Computer Science with minor in Electronic Engineering
- Master's degree in Education

■ Member of the IEEE Verilog and SystemVerilog standards groups since 1993

- Editor of IEEE 1364 Verilog and IEEE 1800 SystemVerilog Language Reference Manuals (LRMs)
- Author of multiple books on Verilog and SystemVerilog

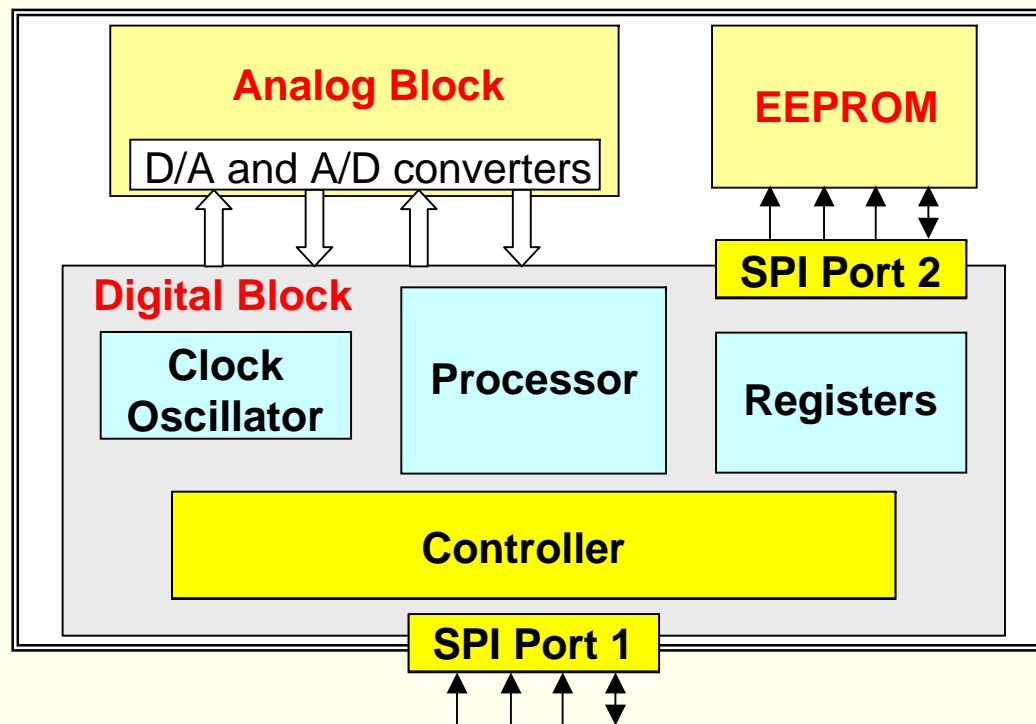
The Challenge

- ▶ I was hired as a consultant to assist a verification team that was behind schedule
 - The design was complete
 - Verification had started
 - A VMM test environment was in place
 - Constrained random transactions were being created
 - The work to be completed included:
 - A scoreboard to verify design functionality
 - Coverage to determine the effectiveness of the random transactions



Overview of the Design

- ▶ The designers were sure the design worked correctly!
 - The Analog Block was internal IP used in several designs
 - The EEPROM was third-party commercial IP
 - The Digital Block was partly re-used from another design
 - The Controller and SPI ports were new, plus "tweaks"



Verification Overview

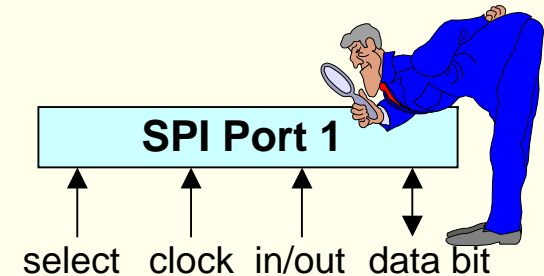
- ▶ **SystemVerilog VMM was used for the testbench**
 - The EEPROM was a commercial model
 - The testbench emulated the Analog Block
 - Randomized transactions for the external SPI port
- ▶ **My task was to add coverage to the verification**
 - Were all possible commands generated?
 - Did all possible responses occur?
 - Did all types of EEPROM operations occur?
 - Were errors from which the design should recover tested?
 - Invalid SPI command received
 - SPI Port transfer had too few bits or too many bits
 - Parity errors



External SPI Port Protocol

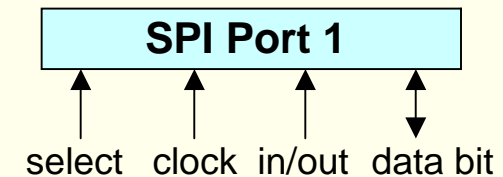
▶ The external SPI (Serial Peripheral Interface) Port:

- Proprietary protocol for this design — not a standard
- Uses a Master/Slave protocol
 - The **external device** is the Master
 - Originates all transfers
 - Sends a command to the design
 - Receives a response from the design
- Transfers data 1 bit at a time
 - A **gated clock** controls the transfer rate
 - The clock is supplied by the Master
 - Unrelated and asynchronous to the internal system clock
- There are no handshake signals
 - A transfer starts when the gated clock starts

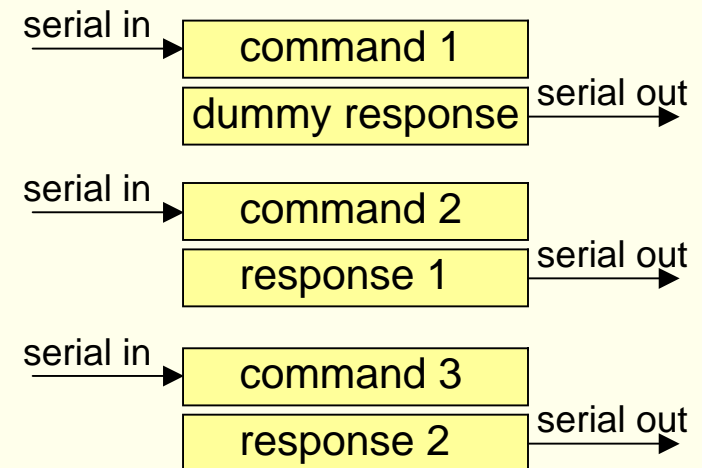


External SPI Port (continued)

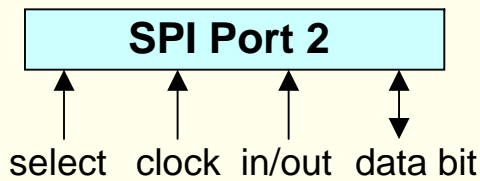
- ▶ The SPI data signal is bi-directional
 - A command input to the design
 - A response output from the design
 - The Master controls the transfer direction



- ▶ Each external SPI Port transfer consists of 2 16-bit words
 - The first word is an input
 - A new command to be executed
 - The second word is an output
 - A response to the previous command
 - Generated by the Controller
 - Each transfer must be exactly 32 clock cycles
 - Too few or too many cycles is bad!



EEPROM SPI Port Protocol

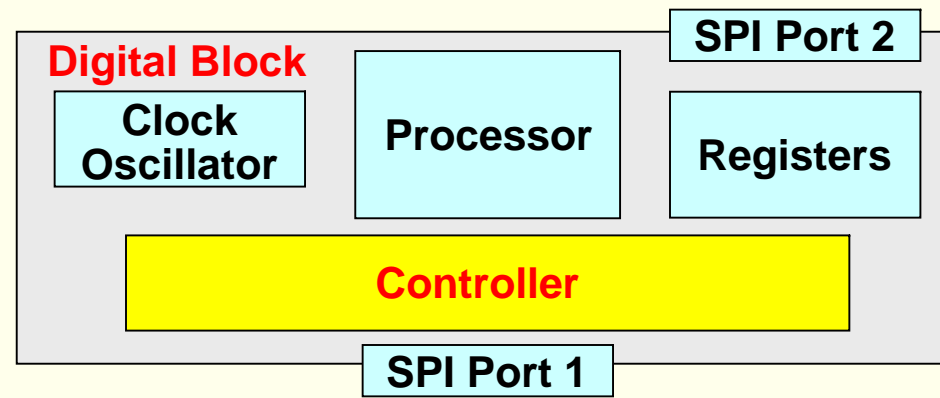


► The EEPROM SPI Port:

- Same signals as external SPI but very different protocol
 - The **design** is always the Master
 - Originates all transfers
 - Sends read/write commands to the EEPROM
 - Uses a **gated clock** derived from the system clock
- A transfer can be 16, 24 or 144 gated clock cycles
 - Read/write EEPROM status register transfer is **16 bits**
 - 8-bit command; 8-bit data word
 - Single word transfer is **24 bits**
 - 8-bit command; 8-bit address; 8-bit data word
 - Burst mode transfer is **144 bits**
 - 8-bit command; 8-bit start address; 16 8-bit data words

The Controller Does a Lot

- ▶ **The Controller block was entirely new for this design**
 - **Decodes commands received from the external SPI port**
 - Uses handshake signals to read/write the SPI port register
 - **Causes appropriate actions to occur within the design**
 - Sets, clears, and responds to various control lines
 - **Generates a response to each external command**
 - Valid responses if no problem occurred
 - Error responses when something is wrong
 - **Recovery from errors needs to be tested!**



Functional Coverage versus Assertion Coverage

▶ Functional coverage is value oriented

■ Can answer questions such as:

- Did the random stimulus generate all commands?
- Did the design respond with all possible response types?

■ Functional coverage cannot *directly* answer every question

- Was a SPI transfer attempted with too few clock cycles?
- Did a EEPROM burst read transfer exactly 16 words?

▶ SystemVerilog Assertions are sequence oriented



Did a SPI transfer occur that
was not exactly 32 clock cycles?

Using Assertions for Coverage

- ▶ Assertions are typically used to check for design failures
 - Report an error if an expected condition does not occur

```
property p_req_gnt;  
    @(posedge clock) // when to sample values  
    request ##3 grant ##1 !request ##1 !grant;  
endproperty: p_req_gnt
```

A sequence of events is defined as a *property*

```
assert property (p_req_gnt) passed_count++; else $error;
```

optional pass statement

optional fail statement

- ▶ Properties can also be used to report coverage

- If the property never completed, then that aspect of the design was not tested

```
cover property (p_req_gnt);
```

Reporting Property Coverage

▶ Three ways to get property coverage information:

```
property p_req_gnt;  
    @(posedge clock) // when to sample values  
    request ##3 grant ##1 !request ##1 !grant;  
endproperty: p_req_gnt  
  
assert property (p_req_gnt) passed_count++; else $fatal;  
cover property (p_req_gnt);
```

■ SVA cover property statements

- Useful, but not integrated into functional coverage report

■ Simulator-specific reports

- Might integrate property coverage with functional coverage, but different for each simulator

■ Functional coverage on the optional pass statements

- Integrates directly into functional coverage reports
 - Needs a proposed SV-2009 enhancement to be useful

The Use of Assertions

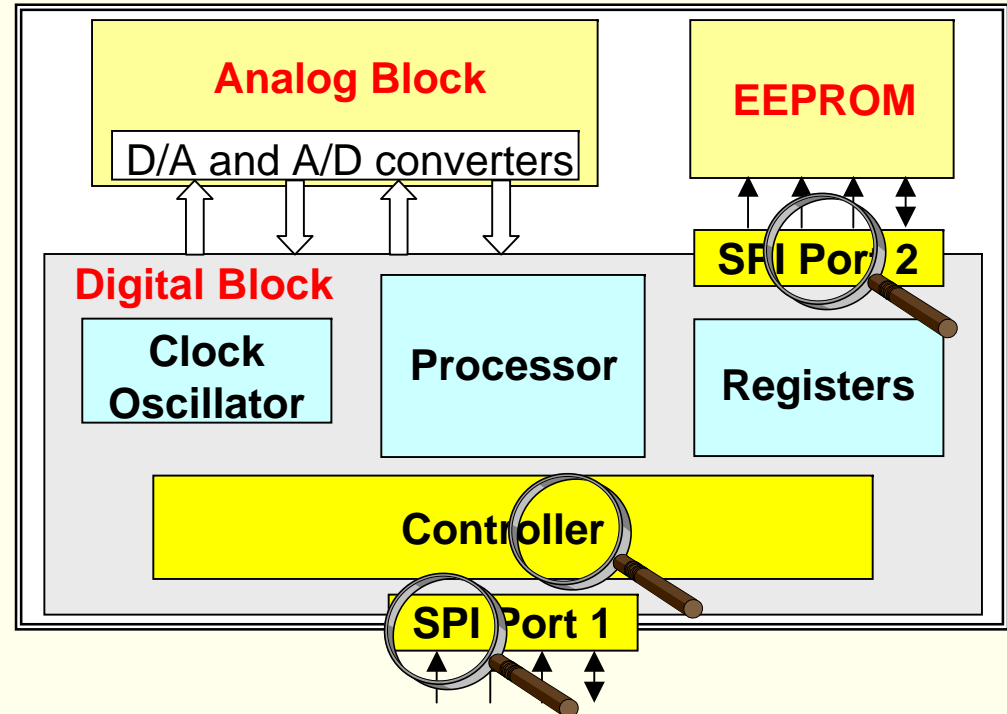
▶ Assertions were used to supplement coverage for:

■ **The two SPI Ports**

- Control lines
- Correct number of transfer clock cycles

■ **The Controller**

- Handshake sequences

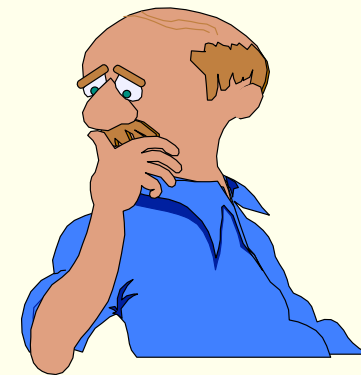


■ **This presentation only gives an overview of why and how assertions were used for coverage**

- The paper discusses the use assertions in more detail

Last Minute "Challenges"

- ▶ The use of assertions was not part of the verification plan
 - Project management had assumed that using a scoreboard and functional coverage would be sufficient
- ▶ Adding assertions late in the project was not easy
 - Several road blocks were encountered
 - The task took longer due to these road blocks
- ▶ Some of the "challenges" (read "frustrations") were:
 - Ambiguous design specifications
 - Design changes after assertion written
 - Inconsistent naming conventions
 - Awkward design partitioning
 - Gated clocks



Ambiguous Design Specs

▶ Example:

"chip_select is an active low signal and is used to enable a transfer on the SPI port."



▶ What this specification does not say:

- Does chip_select need to stay active throughout a transfer, or just long enough to start the transfer?
- Must chip_select go high after each transfer, or can back-to-back transfers be done without toggling chip_select?
- Is it OK for the gated SPI clock to run if chip_select is not active?

▶ The verification manager's answer...

"I don't know — Go ask the designers how they implemented it in the design."

Ambiguous Specs (continued)

▶ My response to the verification manager...

"I can write the assertion to match the design, but all that proves is that the design does whatever it does — It does not prove that the design does what it is supposed to!"

▶ The designer's answers (paraphrased)...

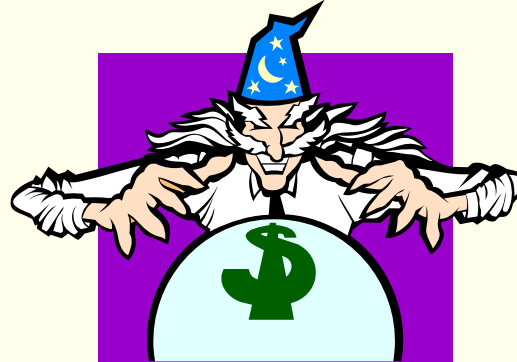
- "chip_select must remain low throughout a transfer — it is used as an enable for an internal counter"
- "chip_select must transition to low at the start of each transfer — it is used to transition a state machine out of its wait state since the gated clock is not yet running"
- "The gated clock running without chip_select could be a problem — we assumed it would never happen"

Other Challenges



- ▶ **Changes to the design after the assertions were written**
 - An active high signal became an active low signal
 - Functionality in one module moved to a different module
- ▶ **Inconsistent naming conventions**
 - Signal names in the design did not match the specification
 - Same clock called by a different name in every block
 - Inconsistent naming of active low signals
- ▶ **Awkward design partitioning**
 - The spec would describe functionality in one block
 - Designers split the functionality across multiple modules
- ▶ **Gated clocks**
 - SVA is cycle based — triggers on clock edges
 - If a gated clock is off, the assertion will not fire

Was It Worth The Effort?



► **YES!** Using assertions for coverage:

- **Revealed shortcomings in the existing verification tests**
 - Additional tests needed to be added
- **Brought out design specification ambiguities**
 - Writing the assertions forced re-examining the spec
 - Found cases where the designers had interpreted the spec incorrectly
- **Found bugs in the design**
 - E.g., A handshake bug passed functional tests, but failed an assertion check

Lessons Learned



- ▶ **Assertions are useful for verification coverage**
 - Provide coverage that can supplement functional coverage
 - Find bugs that might otherwise go undetected
- ▶ **Adding assertions at the last minute is challenging**
 - Poor naming conventions can make assertions more difficult
 - Design partitioning can make assertions more difficult
 - Gated clocks pose problems with assertions
 - Design spec might not have details needed for assertions
 - Ambiguities should be resolved before implementing the design, not after!

Conclusion



Design for Verification!

*Plan to use assertions early
in the design process*

"Imprecise specifications lead to
mysterious problems"

Ken McElvain, DVCon-2009,
"Prototyping: Where Hardware and Software First Meet"