Verification Planning to Functional Closure of Processor-Based SoCs

February 2006  DesignCon 2006  Andrew Piziali
Overview

• Introduction
  • The Design Under Verification
  • Verification Planning
  • Verification Plan Automation
  • Functional Closure
  • Summary
Introduction

- Functional verification is consuming 70% of design labor
- Yet, more verification risk is assumed today!
- The path to functional closure must be quantified
- Is there a map to the verification goal?
- Yes! The map is the verification plan
- Coverage models quantify the verification problem
- Dynamic and static verification address the problem
- An executable verification plan facilitates process management
Overview

• Introduction
• **The Design Under Verification**
  • Verification Planning
  • Verification Plan Automation
  • Functional Closure
• Summary
The Design Under Verification

- Multiprocessor SoC
- Encrypted image read from MIIs
- Image decrypted by the DES engine
- Image rendered by the DSP
- Image written to the LCD/VGA interface
The Verification Problem

- Pre-verified components
  - RISC, DSP, MACs, DMA, USB, HDD
- Suspect components
  - DES engine
  - LCD/VGA block
  - Embedded RISC controller software
Overview

• Introduction
• The Design Under Verification
• Verification Planning
• Verification Plan Automation
• Functional Closure
• Summary
Verification Planning

• Analyze the design specifications
• Define the scope of the verification problem
  – Quantify the scope in coverage models
• Specify the solution to the verification problem
  – Dynamic verification: verification environment functional specification
  – Static verification: property set and theorem specifications
Verification Plan Structure

1. Introduction ........................................... *what does this document contain?*
2. Functional Requirements ....................... *opaque box design behaviors*
   2.1 Functional Interfaces ........................... *external interface behaviors*
   2.2 Core Features ................................. *external design-independent behaviors*
3. Design Requirements ............................. *clear box design behaviors*
   3.1 Design Interfaces .............................. *internal interface behaviors*
   3.2 Design Cores ................................. *internal block requirements*
4. Verification Views ................................. *time-based or functional feature groups*
5. Verification Environment Design ............. *functional specification of the verif env*
   5.1 Coverage ....................................... *coverage aspect functional specification*
   5.2 Checkers ....................................... *checking aspect functional specification*
   5.3 Stimuli ......................................... *stimulus aspect functional specification*
   5.4 Monitors ....................................... *data monitors functional specification*
Problem Scoping Begins with Specification Analysis

- Design intent is captured in the specifications
- Features need to be extracted from the specifications
- Top-down analysis
  - Required for large specifications
  - Need to distill behavioral requirements
  - Intent abstraction gap bridged by the human mind in brainstorm
- Bottom-up analysis
  - Suitable for small specifications
  - Document analyzed chapter by chapter
  - Amenable to analysis automation
Top-Down Analysis Contributors

- Are system performance and features as expected?
- Does HW support all SW functions?
- Will we get it all done in time?
- How do I capture system behaviors?
- What bugs are in the logic I build?
- Does HW support all SW functions?
- Are system performance and features as expected?
- Will we get it all done in time?
- How do I capture system behaviors?
- What bugs are in the logic I build?
Problem Scope Captured in Coverage Model Designs

• Top-level model design
  – Write semantic description
  – Select attributes and values
  – Choose model structure

• Detailed model design
  – What to sample for each attribute?
  – Where in the verification environment should we sample?
  – When should each attribute be sampled?
The DES engine receives data through the input FIFO consisting of encryption keys and cipher text. Sixteen cycles after the last datum is written into the FIFO, the clear text is written to the output FIFO. The input and output FIFOs are configured as 1Kx32. The data control block provides an interface to the AMBA bus for data and control flow for the encrypt/decrypt block.
The input FIFO manages flow control between the AHB interface and the data control block in order to accommodate data transfer rate differences between the two blocks. There are 1,024 32-bit entries in the FIFO.
## Coverage Model Design

<table>
<thead>
<tr>
<th>Feature</th>
<th>Attribute or Sampling Time</th>
<th>Values</th>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow Control Management</td>
<td>@fifo_clock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIFO depth</td>
<td></td>
<td>0, 1, 2..1022, 1023, 1024</td>
<td>DES input FIFO</td>
</tr>
<tr>
<td>AHB write</td>
<td></td>
<td>FALSE, TRUE</td>
<td>DES input FIFO</td>
</tr>
<tr>
<td>Data control block read</td>
<td></td>
<td>FALSE, TRUE</td>
<td>DES input FIFO</td>
</tr>
<tr>
<td>FIFO depth, AHB write, data control block read</td>
<td></td>
<td>C{}</td>
<td></td>
</tr>
</tbody>
</table>
Coverage Model Implementation in e

cover flow_ctrl_mgmt is {
    item FIFO_depth : uint = fifo_current_ptr$ using ranges = {
        range([0]); range([1]); range([2..1022]); range([1023]);
        range([1024]);
    };
    item AHB_write : bool;
    item DCB_read : bool;
    cross FIFO_depth, AHB_write, DCB_read;
};

event flow_ctrl_mgmt is rise(fifo_clock$);
Problem Solution: Verification
Environment Specification

• Dynamic verification
  – Coverage: specified after feature analysis; implemented using functional, code and assertion coverage
  – Checking: data and temporal
  – Stimulus: constrained random generation
  – Autonomous verification environment

• Static verification
  – Model checking requires property specifications
  – Theorem proving requires ... theorems
Overview

• Introduction
• The Design Under Verification
• Verification Planning
• **Verification Plan Automation**
• Functional Closure
• Summary
Verification Plan Automation

• Let’s use the plan like a GPS to answer questions
  – Where are we?  What is the next waypoint?
  – How far are we from functional closure?
  – When will we reach functional closure?

• Verification plan requirements
  – Written in a natural language for human communication
    – Ideas are conceived in our native tongue
    – Abstraction level and ambiguity must be modulated
  – Machine readable for verification environment linkage
    – Forward annotation: plan to verification environment
    – Backward annotation: verification environment back to the plan
Verification Plan Automation

• Verification plan view or perspective
  – Time-based and functional feature groups
• Session – a set of dynamic or static verification runs
• Session input file specifies run parameters
  – Simulation tests
  – Properties to be proven
  – Other run attributes
• Session output file
  – Runs passed and failed
  – Log files and trace files
Overview

• Introduction
• The Design Under Verification
• Verification Planning
• Verification Plan Automation
• **Functional Closure**
• Summary
Functional Closure

n. achieving the functional verification goals specified in the verification plan.

• Goals are specified using chosen metrics
  – Coverage, simulation failures, property proofs, code written

• Coverage analysis
  – Analysis of coverage holes: functional, code and assertion
  – Use coverage hole aggregation, projection and selection

• Failure analysis
  – Correlate failures with run parameters
  – Use Incisive Manager “First Failures” view
Overview

• Introduction
• The Design Under Verification
• Verification Planning
• Verification Plan Automation
• Functional Closure

• Summary
Summary

• Verification risk can be managed and mitigated today
• Start with good functional specifications
• Quantify the verification problem using useful metrics
• Migrate your verification plan from a project artifact to an executable specification
• Employ verification process automation technology to facilitate progress measurement and analysis