Leveraging Formal to Verify SoC Register Map
Agenda

• Problem/Background

• Introduction to Solution

• Application and Results
Problem (1)

• Problem
  • Comprehensive Register Verification is challenging:
    • Register map,
    • Default values,
    • Access policy, and
    • Connectivity: Bus or Bridge configuration.

• Verifying registers requires System Level environment
  • System Level testing needs more time to setup, and
  • its simulation is slow.
Problem (2)

- Problem
  - Simulation based testing is insufficient and not exhaustive
    - Hard to hit corner cases at system level.
    - Setting up complete coverage requires more engineering resources.
  - Bugs in registers are difficult to work around in software
  - Documentation is not in sync with the actual design
    - Access Policy,
    - Bit definitions,
    - Default values,
Solution

• Solution
  • Using Formal to Verify Registers

• Benefit
  • Easy to Setup.
    • We can start early, as soon as documentation is ready.
    • Using input directly from documentation. Verification and documentation share the same source of data
    • Minimal setup is required.
    • Maintenance is easy. No bench to modify or tests to rewrite.
      – Traditional simulation uses directed tests written in C-language,
      – Test updates are needed when the design or register definition is changed
  • Inputs: IPXACT, interface protocol information and design rtl
Solution

• Benefit
  • Exhaustive checking of access policy.
    • No need to spend efforts creating advanced testbench.
    • Automatically handles corner cases.

• Assertions can be ported and used by simulation
Agenda

• Problem/Background

• Introduction to Solution

• Application and Results
Check Generation

- IP-XACT Register Description
- Merge Utility (w/ BFM ext)
- Register Generator
- Formal BFM + checks
- Incisive Enterprise Verifier

- User specified or templates provided for common protocols
- Used to build the checks
- Debug waveforms
- Check failures

- RESET_SEQ
- READ_SEQ
- WRITE_SEQ

User form including protocol specific info

Documentation
Read and Write Sequences

- **APB Read Sequence**

  ![Diagram of APB Read Sequence]

  - PCLK
  - PADDR
  - PSEL
  - PENABLE
  - PWRITE

  **Conditions**
  
  - PSEL && !PENABLE && !PWRITE && (PADDR == REG_chk_addr)
  
  **Notes**
  
  - Pre-defined variable that can take on any address defined in IP-Xact
Read and Write Sequences

• APB Write Sequence

PCLK

PADDR

PSEL

PENABLE

PWRITE

PWDATA

PSEL && !PENABLE && PWRITE && (PADDR == REG_chk_addr) && PWDATA == REG_chk_data

Pre-defined variable: Can assume ALL data values
Read-Write Check

- Back door check

  - ALL activity

  - Start (Initialized)

  - Check (at register)

  - Controllable via command line arguments

- Front door check

  - ALL activity

  - Start (Initialized)

  - Check (via protocol)

  - Activity region enables detection of inter-register issues

  - Write_seq

  - Read_seq

  - Check (via protocol)

  - Controllable via command line arguments

  - Activity region enables detection of inter-register issues

  - Write_seq

  - Read_seq

  - Check (via protocol)

  - Controllable via command line arguments

  - Activity region enables detection of inter-register issues
Read-Write Check

• **RW check property**

```verilog
property rw_fd_chk;
  (... write_seq ##1
  !_start_read_after_write | REG_initiate_write)[*] ##1
read_seq |->
  chk_data(REG_act_fd_data, REG_chk_data, REG_exp_rw_mask)
);
endproperty
```

• **Check for address 0x10**

```verilog
d_reg_0x10_SPI_CTRL_rw_fd_chk : assert property
  (disable iff (reset_expression ||
    (REG_chk_addr != `REG_ADDR_WIDTH'h10)))
  rw_fd_chk);
```

- **Control existence of activity region**
- **Data check**
- **Ensures check only occurs for the register address and not in reset state**

**Read-write property**

**Control existence of activity region**

**Data check**

**Ensures check only occurs for the register address and not in reset state**
Reset and Read-Only Checks

- **Reset Check**
  - **ALL activity**
  - Start (Uninitialized)
  - reset_seq
  - Check (via protocol) (Controllable via command line arguments)
  - read_seq

- **Read-Only Check**
  - **ALL activity**
  - Start (Initialized)
  - read_seq
  - Check (via protocol) (Sample Data (via protocol))
  - read_seq

ALL activity EXCEPT writing to the address it is checking
Agenda

• Problem/Background

• Introduction to Register Solution

• Application and Results
IP Configurations

Faster to Setup

- Capability not utilized
- Back door access
- Front door
- Interface
- Formal BFM + checks
- Marvell IP under test

More Comprehensive

- Front door
- Interface
- BUS FABRIC
- IP 1
  - Reg0 (0x00)
  - Reg1 (0x01)
  - ...
Results

• ½ day to bring up environment
  – Leverage existing user form containing APB Read/Write sequences
    • Mapping APB interface signals
  – Compiling design
• 2-3 hour runs per IP
• Register maps for 5 IPs on two separate APB busses were validated using this flow
Bugs Found

• Many design versus specification mismatches
  – Field attribute: read-write vs. read-only.
  – Default Value,
  – Register Address Encoding
  – Duplicate register address error detected
    • 2 read-write registers at the same address (in IPXACT file) but with different reset values
    • Read from the address only reads one register
    • Reset check for one of the registers failed
    • Easily caught by the flow without the need to create stimulus
Reset Check detects design issue

- Reset Check for addr 0x00 (Register 1)
  - Expected data **0x00**

  ![Diagram showing successful check for addr 0x00 (Register 1)]

- Reset Check for addr 0x00 (Register 2)
  - Expected data **0xFF**

  ![Diagram showing unsuccessful check for addr 0x00 (Register 2)]
Summary

• Flow met goals in terms of efforts, speed, and quality:
  – Setup speed – Pre-existing APB Write Read sequences
  – Fast Formal tool execution - 2 to 3 hour runs
  – Automatic check generation from IPXACT description
  – Completeness
    • Found many design vs specification errors
    • Found bugs missed by directed tests on previous designs.