Using Formal Applications to Create Pristine IPs

Lee Burns, Cypress Semiconductor
David Crutchfield, Cypress Semiconductor
Bob Metzler
Hithesh Velkoorru, Cypress Semiconductor
Agenda

• Why use Formal apps
• Starting point for introduction of Formal apps
• Implementation of Formal apps in the flow
  – Connectivity Checks
  – Coverage Closure
  – Register Verification
  – Code Quality
• Usage and results
Verification Pain Points - Connectivity

- Function routing through an IP
- Programmable interconnect
- Exhaustive verification is time consuming
Verification Pain Points - Connectivity

• Chip level interconnect
• Fully verified IP
• Exhaustive verification of all connections is time consuming
Verification Pain Points - Coverage

• Goal is 100% code and functional coverage
• Mining for unreachable code is time consuming
  – Iterative process
    • Develop tests
    • Review coverage
    • Consult IP owner
    • Create exclusions
    • Develop more tests
  – Repeated with code changes
Verification Pain Points - Registers

• Functional simulation using UVM_REG is time consuming
  – Read and understand spec
  – Not all bits are used within a register (requires non-contiguous masking)
  – Develop tests
  – Debug results
  – 4-6 weeks for 50K design
Verification Pain Points – Code Quality

• Design team does minimal testing
  – No formal test bench / may not compile
• Verification finds simple issues
  – Locked state machine
  – Combinational loop
  – Logic contention
  – Cycles of code quality
Existing Verification System

• VMS – Verification Management System
• Established a standard approach to:
  – Design and test bench organization
  – Specification of tools arguments
  – Test list creation
  – Regression status / coverage

INPUT

dut.files, tb.files, vms.cfg, test-list, command-line, pre/post scripts

VMS_RUN

OUTPUT

UCDB, HTML, Reports, logs, email, waves, debug, test-lists
Existing Verification System

- VMS manages compilation and simulation jobs through Mentor’s Questa VRM (Verification Run Management)
- VMS needs to provide
  - Additional steps for Formal applications
  - Additional inputs to feed Formal applications
  - Standard interface for Functional and Formal verification

INPUT
- dut.files, tb.files, vms.cfg, test-list, command-line, pre/post scripts

VMS_RUN

OUTPUT
- UCDB, HTML, Reports, logs, email, waves, debug, test-lists
Connectivity Checks

• Two new steps in VMS:
  – Generate connectivity properties
  – Verify properties with Formal tool (Questa PropertyCheck)
• New input to VMS: Connectivity spec (CSV)
• If all properties pass then connections are true
• No test bench required
Coverage Closure (Flow 1)

• One new step in VMS:
  – Generate exclusions with Formal tool (Questa CoverCheck)
• Exhaustive search for unreachable code (time consuming)
• Simulation optional to just generate exclusions
  – No test bench required / exclusions without test knowledge
Coverage Closure (Flow 2)

- No new steps in VMS / Normal exclusion flow
- Exclusions generated in previous run
- Coverage adjusted after regression run
Coverage Closure (Flow 3)

- One new step in VMS:
  - Generate exclusions with Formal tool (Questa CoverCheck)
- Seed code coverage to minimize search for unreachable code (less time consuming)
- Test bench and tests required
Register Checks

• One new step in VMS:
  – Verify registers with Formal tool (Questa RegisterCheck)
• New input to VMS: Register spec (CSV)
• RegisterCheck:
  – Generates assertions from CSV
  – Formally proves assertions against design
• No test bench required
Code Quality

- One new step in VMS:
  - Apply standard checks against design with Formal tool (Questa AutoCheck)
- AutoCheck:
  - Generates assertions based on desired checks
  - Formally proves assertions against design
- No test bench required
Connectivity Check Usage

```
define WidthParam_0 1
check_cond #( .width(`WidthParam_0) )
CHECK_COND_FROM_mod2_out_0_TO_ip_out_1 ( .cond( (mux1.sel == 1)), .src( mod2.out[0] ), .dest( ip.out[1]) );

module check_cond #(parameter width = 1) (input logic [width-1:0] src, dest, input logic cond);
connectivity_assert : assert property ( @$global_clock cond |-> src == dest );
endmodule : check_cond
```

type,src,dest,cond,delay
cond,mod2.out[0],ip.out[1], mux1.sel==1
Connectivity Check Results

- Completed routing verification of PLD like IP in 1 week
- Savings > 4 person weeks (exhaustive simulation)
- Identified major bug (likely missed in functional simulation)
- Automated chip level connectivity verification through CSV generation from spec

<table>
<thead>
<tr>
<th>Block</th>
<th>Route Inputs</th>
<th>Route Outputs</th>
<th>Control Register Bits</th>
<th># Lines in CSV File</th>
<th>Formal Runtime (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>408</td>
<td>308</td>
<td>962</td>
<td>5266</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>408</td>
<td>308</td>
<td>962</td>
<td>5266</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>312</td>
<td>256</td>
<td>786</td>
<td>5127</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>312</td>
<td>256</td>
<td>786</td>
<td>5127</td>
<td>2</td>
</tr>
</tbody>
</table>
Connectivity Check Concerns / Lessons

- Knowing when to use Formal connectivity checks
- How to combine Formal results with simulation coverage
- Good specifications of connectivity are needed
- Strict code modularity / limit modules to specific behaviors
- Automation of connectivity CSV is key for savings
Coverage Closure Results

Results of example IP without tests provided

- Approximately 1 hour execution
- 17783 unreachable items found
  - Branch 8489
  - Condition 22
  - Expression 1517
  - FSM States 0
  - FSM Transitions 0
  - Statement 7577
  - Toggle 178
  - Coverbin 0

- Exclusions generated for each
Coverage Closure Issue

• Parameter passing to highly configurable IP
  – Questa Formal can only process one configuration
  – Coverage exclusions can change for each configuration
  – Intersection is not sufficient
  – Union could cause errors
• Iterate over configurations and selectively apply exclusions based on configuration
Coverage Closure Issue – Potential Solution

- Questa CoverCheck analyze configuration parameters
- Select configuration
- Execute CoverCheck
- Repeat for more configurations
- Categorize exclusions based on configuration
Register Check Usage

- Register CSV and control file generated from internal specification

IP Specification

- ra
- register u_csr_bctl.$register_$field
- interface amba_ahb
- base_addr 0x00000000
- spec_type uvm
- signal_match nocase, prefix, postfix

- interface_port hready_in = mmio_hready
- interface_port hselx = mmio_hsel
- interface_port hwrite = mmio_hwrite
- interface_port haddr = mmio_haddr

----------
- interface_port hresetn = rst_hf_act_n
- interface_port hclk = clk_sys

Register Name, Register Description, Register Address, Register Width, Register Access, Register Reset Value, Register Reset Mask, Field Name, Field Description, Field Offset, Field Width, Field Access, Field Reset Value, Field Is Covered, Field Is Reserved, .memmap_write_internal botsel_l,"comment",0x00007808,32,RW,0x0,0xffffffff,clk_sel0,"comment",0,2,RW,0x0,0xffffffff,,
botsel_l,"comment",0x00007808,32,RW,0x0,0xffffffff,clk_sel1,"comment",2,2,RW,0x0,0xffffffff,,
botsel_l,"comment",0x00007808,32,RW,0x0,0xffffffff,clk_sel2,"comment",4,2,RW,0x0,0xffffffff,,
botsel_l,"comment",0x00007808,32,RW,0x0,0xffffffff,clk_sel3,"comment",6,2,RW,0x0,0xffffffff,,

3/1/2022
David Crutchfield, Cypress Semiconductor
Register Check Results

- PLD like IP (full configuration) functional simulation
  - 2781 registers = 27,105 register fields
  - Approximately 55,000 AHB accesses for full verification
  - > 50 CPU Hours
- Evaluating RegisterCheck
  - 10 register fields
  - 32 CPU min for design compilation
  - 76 CPU min for proving properties
Register Check Issues

• 7.6 min per / register field * 27,105 fields = 143 CPU days!
  – Black boxing could provide 20x improvement
  – More work / profiling to make performance reasonable

• Unique register access not understood
  – Address aliasing
  – Address ganging

• Non-uniform register variable naming
  – Makes automation of register specification difficult
Quality Check Results

- Approximately 30 min execution time
- Results of PLD like IP
  - Block Unreachable 1690
  - Bus Multiply Driven 2
  - Bus Undriven 2
  - Bus Value Conflict 2
  - Combo Loop 114
  - Declaration unused 36
  - Init X Unresolved 9067
  - Logic Unused 71
  - Port Unused 31
Conclusions

• Formal applications
  – Can be leveraged for automation
  – Good first step into Formal techniques
  – Can get started without a test bench

• Connectivity and Code Quality Checks show the most promise

• Coverage and Register Checks need more investigation and tool enhancements for flow integration