A Verification Methodology for SoC-Controlled, Highly-Integrated, Mixed-Signal and RF ICs

D. B. Walker
Sirenza Microdevices, Inc.
CDNLive! 2007 Silicon Valley
Session 2.11
Outline

- Introduction
  - System
  - Problems
  - Example
- Sirenza Solutions
- Further Work
SoC System

- Big “A” - little “d” chip taped out in the DFII environment
- Large number (~300) of digital outputs to control integrated analog/RF circuits
- Simple SPI (4-wire serial) interface for configuration data
- Direct inputs to control the transceiver state (2-3 inputs)
- Firmware stored in mask ROM
- Firmware responsible for:
  - Calibration of analog blocks
  - Sequencing the turning on of the analog blocks
  - Reacting to configuration changes and transceiver state
  - Digitally controlled AGC
PHS Transceiver Block Diagram

RX RF Input → LNA → RF Mixer → Channel Selection LPF → IF Mixer → RX Output

Ref Freq → PLL

V_{tune} → PLL → \div N → Channel Selection LPF

Charge Pump Output → PLL

TX I/Q Data Input → Channel Selection LPF

PA → TX RF Output

SoC

Baseband Control
Problems

- Large number of internal outputs make verification difficult because there are a lot of signals to verify.
- How do you make sure everything is connected correctly?
- The complex operation of the firmware makes verifying its operation difficult.
- Program in mask ROM means the firmware must operate the first time.
- How do you know if you’ve tested the firmware sufficiently?
Sirenza Techniques

- Generate DUT netlist directly from the DFII environment
  - Ensures connectivity between analog and digital blocks
  - Provides stimuli to the firmware
  - Checks for correct operation
- Instruction trace GUI for firmware debug
  - Familiar, intuitive interface provides assembly source code debugging
  - Firmware operation tied to actual hardware
- Firmware code coverage utility
Example SoC System

- **Analog**
  - Differential amplifier with support for DC Offset correction

- **SoC**
  - BC6502 Processor ([www.birdcomputer.ca](http://www.birdcomputer.ca))
  - Responsible for controlling the DC Offset correction
  - P65 assembler

Email: [dwalker@sirenza.com](mailto:dwalker@sirenza.com) for demo files.
Netlist From DFII Procedure

Start

Build DFII Configuration File

Build a “verilog_model” view for each top-level block

Netlist using NCVerilog Tool

Build Verilog Code for DUT using command line tool

Devise analog block models and assertions

End

- Cadence DFII Tools
- Sirenza Developed Tool
- End-user task
Build “verilog_model” Cell View

- Select the top level cell in the DFII library manager
- Select the menu, File ▶ New ▶ Cell View…
- Fill out the form as shown and click “OK”
- An editor window opens. Save and quit. A SystemVerilog module will be bound to this stub later)
- Repeat for all of the top-level analog blocks
Create a configuration view for top level schematic

Use views in this order (preference to view “verilog_model”)

Stop at view “verilog_model” or “symbol” by default

Digital block has no “verilog_model” view

Override default view to expose module “ctl_logic”
Netlist Using Verilog Integration Tool

- From the CIW, select the menu: Tools ➤ Verilog Integration ➤ NC-Verilog…
- Set options (Setup ➤ Netlist…):
  - Netlist Explicitly
  - Support Escape Names
  - Drop port range
- Set simulator timescale

Press to initialize run directory

Press to netlist

Enter the configuration created in the last step
Build Composite Verilog File

- The Verilog Integration netlisting tool creates each module in various files
- The file “verilog.inpfiles” specifies all of the files that were netlisted
- To build the composite verilog file, execute the following commands:

  > cd <Run directory>
  > cat `sed 's//.*||' verilog.inpfiles` > <output filename>
Add Verilog Models and Assertions

- **SystemVerilog Models**
  - Generate digital inputs
  - Serve as a base for assertions to check behavior

- **Assertions**
  - Define the “Rules” for operation
  - Check for valid sequences

- **Types:**
  - Inner-module
    - Checking within a module
    - Test module bound to the analog block
  - Intra-module
    - Checking between modules
    - Test module bound to the DUT
Example: DC Offset Correction: Circuit

- Need to model:
  - Delay of the amplifier to the comparator with respect to the DAC inputs and block enables
  - A DC offset for the circuit to correct

- Need to assure:
  - The blocks are enabled correctly
  - The calibration result is correct
Example: DC Offset Correction: Model

// Model the analog circuit. Assume...
// Amplifier turn on time : 6us
// DAC turn on time : 5us
// Comparator turn on time : 3us
// Comparator delay from dac inputs : 2us
//
// Model turning on the sub-blocks
assign#(5us, 0) dac_on = EN_DAC;
assign#(6us, 0) amp_on = EN_AMP;
assign#(3us, 0) cmp_on = EN_CMP;

// If the all of the circuits are turned on, then schedule
// an update on the comparator output for 2us later.
// Otherwise, the output should be unknown.
always@(*)
    if (dac_on && amp_on && cmp_on) begin:setcomp
        CMP <= #(2us) DAC > cal_value;
    end
    else begin
        disable setcomp;
        CMP = 1'bx;
    end
Example: DC Offset Correction: Checking

// Some handy properties
property en_without_VDD(x);
   @(posedge clk_us) ($rose(x)) |-> VDD;
endproperty

property VDD_off_enable_check(x);
   @(posedge clk_us) ($fell(VDD)) |=> !x;
endproperty

// Check the enables with respect to the power
en_dac_without_VDD: assert property (en_without_VDD(EN_DAC));
VDD_with_en_dac: assert property (VDD_off_enable_check(EN_DAC));
... repeat for “EN_CMP” and “EN_AMP” ...

// make sure that the amplifier is operating calibrated. The system is
// considered calibrated if
//  1) Calibration has been performed at least once
//  2) The amplifier and DAC are both on and the comparator is off
//  3) The DAC value matches the calibration value +/- 1 LSB
amp_out_of_calibration: assert property (
   @(tick_us) (EN_AMP && !EN_CMP && EN_DAC && calibrated) |=>
      (DAC == cal_value || DAC == cal_value+1)));

Instruction Trace Utility

- Instruction trace helper module
  - Bound to core with SystemVerilog “bind” statement
    - Additional top level signals:
      - Indication of program fetch
      - Internal register values
  - Writes binary file describing bus activity
    - Address
    - Read or write
    - Data
  - Writes binary file detailing execution of each instruction
    - Instruction time
    - Internal registers state (including condition codes)
    - Pointer to bus activity entry
- Intuitive Tcl/Tk GUI capable of running within SimVision or stand-alone
  - Uses concepts from other standard debugging GUI’s
Instruction Trace Helper Module: Instruction Activity

```haskell
// instruction record
struct packed {
    longint t; // time
    int recid; // pointer to the bus fetch
    byte a; // register a contents
    ... repeat for x, y, sp and pc registers
    byte sr; // status register contents
    byte sr_xmask; // mask indicating if any sr bits are 'x'
} inst_rec;

// for each valid bus cycle...
always@(posedge clk iff (stb && ack && !rst && trace)) begin
    if (fetch) begin
        inst_rec.t = $time;
        inst_rec.recid = recid;
        inst_rec.a = reg_a;
        ... repeat for x, y, sp, sr and pc registers

        // form a mask where a "1" indicates an unknown
        inst_rec.sr_xmask = reg_sr ^~ reg_sr;
        inst_rec.sr_xmask ^= 255;

        // write the record out in binary form
        fwrite(instfid, "%u", inst_rec);
        instid++;
    end
```
Instruction Trace Helper Module: Bus Activity

// bus activity record
struct packed {
    shortint adr;  // address of activity
    byte dat;     // data read/written
    byte dat_xmask;  // mask indicating ‘x values
    byte stat;  // status of bus activity
} bus_rec;

// for each valid bus cycle...
always@(posedge clk iff (stb && ack && !rst && trace)) begin
    logic [7:0] tb;  // 3-state temp byte

    bus_rec.adr = adr;
    tb ≡ we ? dat_o : dat_i;
    bus_rec.dat = ~tb;
    bus_rec.dat_xmask = tb ~^ tb;
    bus_rec.dat_xmask ^= 255;
    bus_rec.stat = {fetch, 6'b0, we};
    $fwrite(busfid, "%u", bus_rec);
    recid++;
end
Accessing Instruction/Bus Activity in Tcl/Tk

- Open the file as a binary file
  ```tcl
  set fid [open $filen r]
fconfigure $fid -translation binary
  ```

- Use the “seek” command to select a specific record
  ```tcl
  seek $fid [expr {$record * 20}]
  ```

- Use the binary “scan” function to read fields
  - Different for big-endian architectures
  - For little-endian (Linux)
    ```tcl
    binary scan $v scccccciw pc sr_xmask sr sp y x a recid t
    # convert to unsigned numbers
    set pc [expr {$pc & 0xFFFF}]
    ```
    ...

Instruction Trace Utility’s GUI

- Step backward into last instruction
- Step forward into next instruction
- Step forward over next instruction
- Search backward to highlighted line
- Search forward to highlighted line
- Time display:
  - Reset – set reference time to current time
  - Abs Time – set reference time to zero

Register Contents

Bus Activity
Firmware Code Coverage

- Use the instruction trace helper module to record coverage data:
  - Address of instruction executed
  - If the instruction executed was a conditional branch, record the state of the condition

- Use the “iccr” utility:
  - Aggregate data from several runs
  - Dump coverage data for analysis

- Use a Tcl/Tk GUI to analyze results
Recording Coverage Data

// Firmware code coverage
bit [15:0] fetched_addr;
bit [7:0] opcode;
bit after_fetch;

// SystemVerilog Code...
always@(posedge clk iff !rst && ack && stb) begin
  if (fetch) begin
    fetched_addr <= adr; opcode <= dat_i;
  end
  after_fetch <= fetch;
end

// Coverage statements for ROM
covergroup code_coverage @(posedge clk iff !rst && ack && stb);
  c_addr : coverpoint adr iff fetch {
    _ bins ROM[] = {[16'hFE00:16'hFFFF]};
  }
  c_bcnd : coverpoint {fetched_addr, takb} iff (after_fetch && opcode[4:0] == 'h10) {
    _ bins bcnd_ROM[] = {[17'hFE00 << 1):(17'hFFFF << 1) | 1]};
endgroup

code_coverage cc = new;

Keep track of Opcode and Address
Mark address of all instructions fetched
Mark each branch of conditional branch
Processing Coverage Data

- Use "iccr" commands to dump coverage information
  ```
  > iccr -COV58
  load_test <test name>
  indices -off
  set_legend -off
  report_detail -d -covered trace
  exit
  ```

- Parse coverage data using a Tcl/Tk GUI
  - Read coverage data
  - Read object file
  - Read source files
  - Display coverage data
Coverage GUI Display

This statement was executed 10 times

This statement branched 6 times and fell through 4 times

This statement branched 4 times and never fell through

This statement never executed

Summary Statistics

Total Coverage: 95.74% (45/47) Bcnd Coverage: 66.67% (4/6)
Further Work…

- Top level netlisting/modeling
  - VerilogAMS integration

- Instruction Trace
  - Support for multiprocessors
  - More modular code

- Firmware coverage
  - Integration into Design Team Manager
  - Firmware coverage in the verification plan