Verification of an Image Processing Mixed-Signal ASIC

Kevin Buescher, EM Microelectronic-US, Colorado Springs, CO kevin.buescher@emmicro-us.com
Milos Becvar, EM Microelectronic-US, Colorado Springs, CO milos.becvar@emmicro-us.com
Greg Tumbush, PhD, PE, Tumbush Enterprises, Colorado Springs, CO, greg@tumbush.com
David Jenkins, FirstPass Engineering, Castle Rock, CO, david@firstpasseng.com

***Abstract* - This paper presents the verification of an image processing mixed-signal ASIC containing a custom CPU. The work was performed at EM Microelectronics-US, Inc.**

I. INTRODUCTION

This paper presents the verification of an image processing mixed-signal ASIC containing a custom CPU. This ASIC is a major enhancement to the ASIC reported in [1]. The work was performed at EM Microelectronics-US, Inc.

The ASICs that EM Microelectronics-US develops are typically very lower power with standby current in the uA range and operating current in the 100's of uA range. Low power modes are used extensively. Their ASICs are optimized for area, hence, memory is to be kept to a minimum. The ASICs are typically dominated by analog, and the design teams are small, 1 or 2 designers and a similar number of verification engineers. Their design cycles are short, concept to tapeout in less than 12 months. Their ASICs must sell in a market that is typified by low cost and high volume.

For this project Mr. Becvar, a co-author, was the architect, digital designer, and firmware engineer. Mr. Buescher was the analog designer. Dr. Tumbush was contracted to perform the mixed signal chip-level simulations and develop analog models in SystemVerilog using real number modeling. Mr. Jenkins, also a co-author, was contracted to integrate Dr. Tumbush's models and perform the digital top-level verification. This paper will focus on the mixed signal chip-level and digital top-level verification, while [1] focused on the custom CPU of the previous generation.

Three major improvements were made to the previous verification environment: migration to a UVM-based testbench, real number modeling of analog blocks, and rigorous chip-level AMS verification. These improvements will be expanded upon in the following sections along with results. In this paper the term top-level digital testbench will refer to the testbench testing the digital logic. This testbench will use SystemVerilog models of the analog blocks. The term chip-level testbench will refer to the testbench testing the entire ASIC including digital logic, analog, memories, and pads.

II. MIGRATION TO A UVM-BASED TESTBENCH

It was decided to migrate the digital top-level VHDL testbench used in [1] to a modern UVM-based testbench. There were many reasons for this decision. First and foremost, for re-use and to simplify the verification of follow-on devices. UVM provides a standardized methodology with industry wide support – both from a tool or IP availability and also resource accessibility. Working within a standardized methodology ensures a consistent approach, an approach that supports (1) re-use: (a) with-in the current project by the extension of classes, shared methods, code templates, etc. (b) simpler extension to the next project or (c) with the potential use of external vendor IP; (2) built-in constructs for constrained randomization; and (3) allows for the partitioning of work.

Alternative to migration to a UVM-based digital top-level testbench was modification and reuse of a VHDL-based testbench used in previous generation of devices. In many cases this is a very attractive short-term approach minimizing verification effort. In our case the underlying Device Under Verification (DUV) design changed in so many aspects that the level of testbench reuse was severely limited. In addition, the original verification team was not available and it was expected that the new verification team would spend significant time learning a non-standardized VHDL testbench before any real work could be done. Consequently, it was estimated that the effort associated with modifying the VHDL testbench was comparable with migration to a UVM-based testbench.

The new digital top-level testbench is depicted in Figure 1. Using randomization and functional coverage the number of tests was reduced from 165 to less than 50. The DUV is the yellow block. The real number models of the analog blocks are in dark purple. The Image Processing Logic (IPL) contains the custom CPU which was verified using a block-level UVM testbench described in [1]. The CPU itself was enhanced by new instructions which were easily integrated into the existing UVM-based ISA model. Integration of the new CPU was therefore a relatively straightforward task.

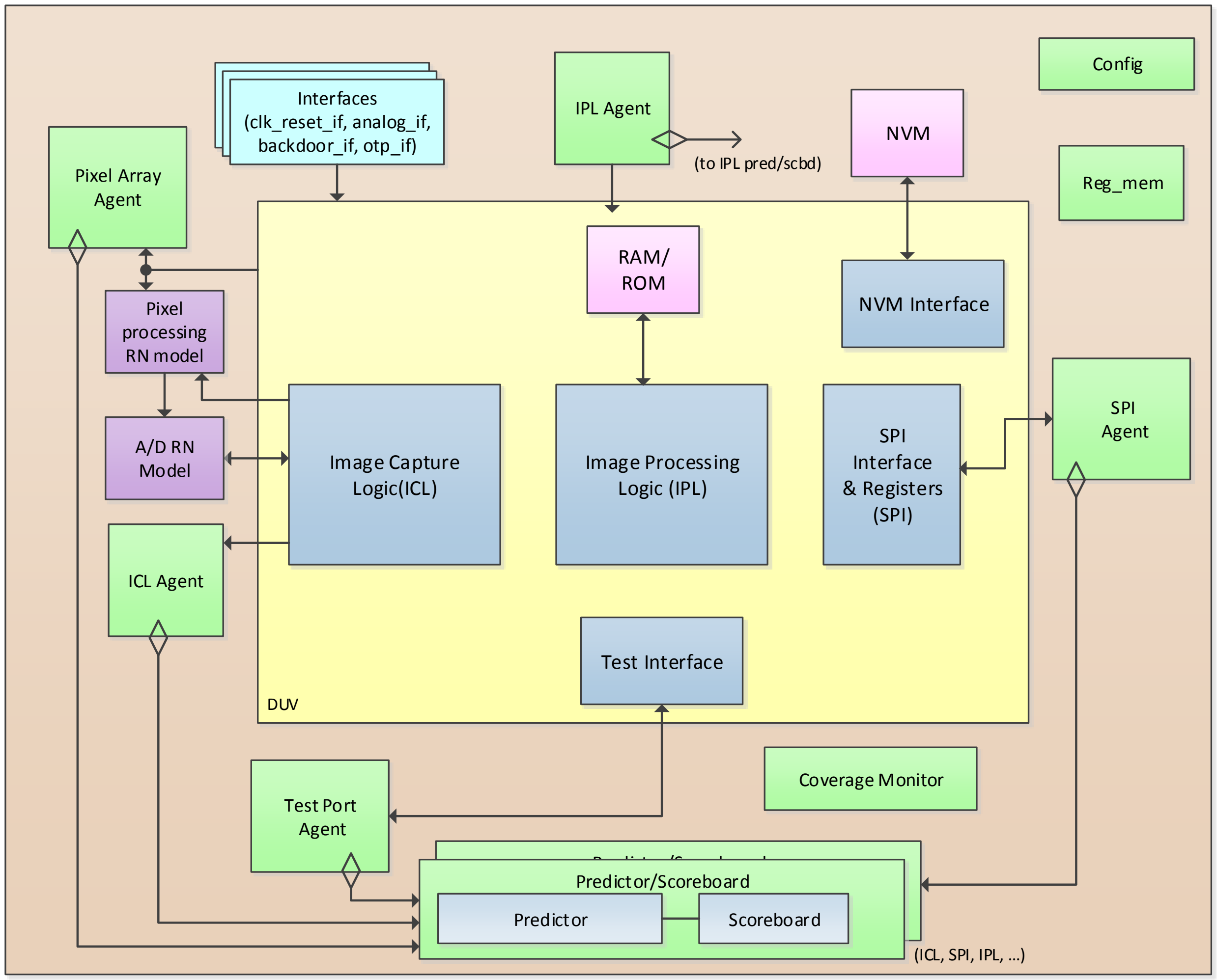

Figure 1: Digital Top-Level Testbench

A. Consideration and Pitfalls

Although the previous generation VHDL based test-bench will not be used for the top-level digital testbench it can still be of use. For example, it can be very useful to re-run simulations on the previous generation design to understand functionality. For this project the previous generation VHDL based test-bench was migrated to the new design and used to create production test patterns.

Management needs to be aware that the new UVM-based testbench will likely take longer than migrating the VHDL based testbench. The same education on the benefits of UVM and randomization needs to be provided as describe in [1]. In addition, the time to find the first bug will also likely be longer as shown in Figure 2. Directed testing finds the first bugs quicker because less time is spent on testbench infrastructure to support randomization. But once the testbench infrastructure is in place verification closure occurs sooner than with directed testing.

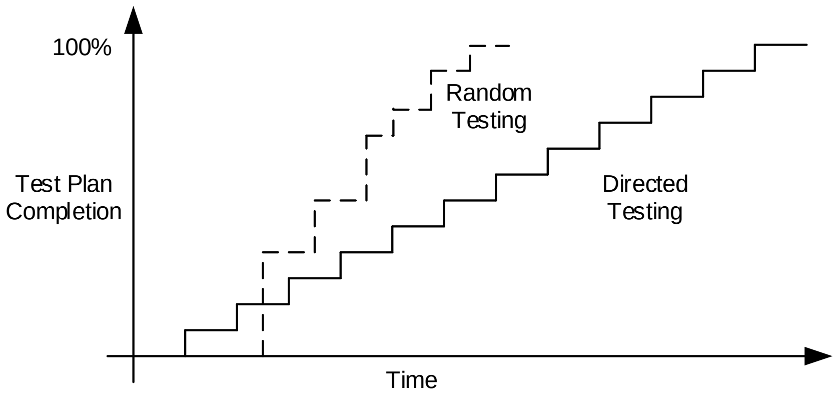

Figure 2: Directed vs random testing

III. REAL NUMBER MODELING OF ANALOG BLOCKS

This ASIC, being mixed-signal, required models of some of the analog blocks for the top-level digital testbench. The accuracy of the model needs to be appropriate. A very detailed model takes a long time to develop and verify and possibly to simulate. Too little detail may miss bugs. For example, should the model of an oscillator simply model the oscillator frequency? Which frequency should be modeled, slow, fast, or somewhere in the middle? Should the oscillator trim be modeled? What about the enable? In [1] VHDL models of the analog were used. These models were not verified against the "golden" analog schematics so their accuracy was questionable. Now, AMS simulations of the real-number model against the schematic are performed to ensure correctness as depicted in Figure 3.

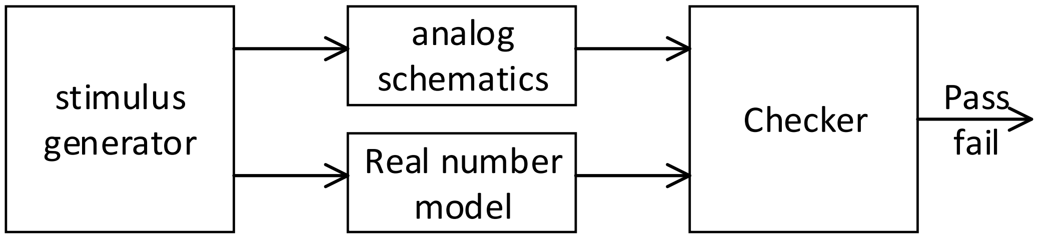

Figure 3: Verify Real-Number Models Against Analog Schematic

The development of the analog models (schematic and RN) was done using a "4 eyes approach". That is, the actual analog schematic circuits were designed completely independent from the RN models by different designers. This technique allowed for a unique ability to cross-check the schematic and the RN model design simultaneously and reduced the chance of misunderstanding the block specifications.

Since both the schematic and the analog models were designed from the device specification with little interaction between the individual designers the probability of catching design "bugs" was greatly improved. Each designer (the schematic and the RN model) had to interpret the specification correctly and create a "bug free" design in order for the final analog output streams to match. Thus any discrepancies in the output were due to one of four possibilities: 1) the schematic designer misinterpreted the spec, 2) the schematic designer made a design functionality error, 3) the RN model designer misinterpreted the spec, or 4) the RN model designer made a mistake in the RN model functionality.

This approach for verification definitely provided better analog fault coverage for the entire analog processing chain and improved test time efficiency. It allowed more structured test cases in addition to a better systematic method to evaluate the final analog output stream. The authors were able to directly compare the results between the analog schematics and the analog models for a few reference test cases to ensure that both paths produced the correct output. Once this step was completed the authors could rely solely on the models to complete the other test vectors in a significantly shorter timeframe.

A. Design of Pixel Processing Real Number Model

The Pixel Processing real number (RN) Model, shown in Figure 1, takes as input a 2-dimensional (2-D) array of pixel intensity values represented as real, processes them, and outputs a single 1-dimensional (1-D) array of processed values represented as real. A code snippet of the module declaration is below. Details have been obscured due to the proprietary nature of this ASIC.

```
module pixel_processing(
  input real pix_bus [NUM_ROWS-1:0] [NUM_COLUMNS-1:0],
  // Other inputs to control pixel processing
  output real pix_out [NUM_ROWS-1:0]
);
```

B. *Verification of Pixel Processing Real Number Model*

The Pixel Processing RN Model was verified against its analog schematic equivalent using the high-level methodology shown in Figure 3. A snippet of the extracted analog netlist is below. As can be seen from the snippet below, the schematic representation requires bias currents and voltages and pix_bus is no longer 2-D but is NUM_ROWS 1-D arrays.

```
subckt pixel_processing_ana
  pix_bus0\<0\> pix_bus0\<1\> ……
  Vdd Vss Ibias
  // Other inputs to control pixel processing
  pix_out\<0\> pix_out\<1\> …….
```

To provide the bias currents and voltages the schematic was wrapped with a Verilog-AMS wrapper.

```
`include "constants.vams"
`include "disciplines.vams"

 module pixel_processing_vams( input wreal  pix_bus0 [NUM_COLUMNS-1:0],
                               // pix_bus 1 to NUM_ROWS-1
                               // Other inputs to control pixel processing
                               output wreal pix_out [NUM_ROWS-1:0]);

  electrical Vdd, Vss,  Ibias;
  ground Vss;

  analog begin
    V(Vdd) <+ 2.0;       // Set Vdd=2.0V
    I(Ibias) <+ 0.001;   // Set Ibias=1mA
  end

pixel_processing_ana pixel_processing_ana(.pix_bus0(pix_bus0), . pix_bus1(pix_bus1), ….
                                          .Vdd(Vdd), .Vss(Vss), .Ibias(Ibias),
                                          // Other connections to control pixel processing
                                          .pix_out(pix_out));
endmodule
```

The testbench splits the 2-D arrays of reals into NUM_ROWS 1-D arrays of reals for input to the Verilog-AMS wrapper. This was done in the testbench instead of the Verilog-AMS wrapper because a 2-D array of *wreal* is not allowed [4]. Also, note that the input port of the wrapper is of type *wreal* as opposed to *real* due to the fact that ports cannot be of type *real* in a Verilog-AMS description. A detailed block diagram of the final block level testbench is in Figure 4.

The Stimulus Generator shown in Figure 4 is simply a NUM_ROWS by NUM_COLUMNS array of pixel values and some control signals. The pixel values are real and can be directed to test a particular function or random.

The SystemVerilog code for the checker is shown below Figure 4. The function *check_pix_out* takes a *pix_out* array from the analog schematic and the real number model. These two real arrays are passed as reference (as opposed to an input) to not make a copy of the real array. This is more memory efficient. The function *close_enough_real*, not shown, compares the two real values. Since the real values rarely match exactly a tolerance is also provided. The tolerance can be in terms of a percentage or in absolute terms. In this case the tolerance is specified to be 1mV. If the two reals are not within the tolerance an error is displayed and an error counter is incremented. In the case of the two reals being within tolerance a correct counter is incremented. It is important to count the number of times a checker was correct, as well as the number of times a checker was incorrect, for the test to not pass vacuously. At the end of the test the expected value of *error* and *correct* is checked.

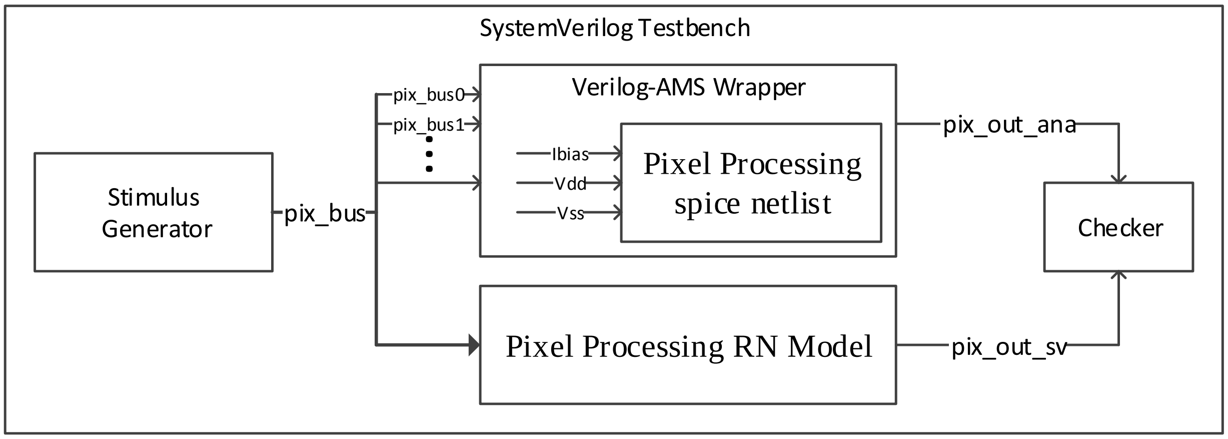

Figure 4: Testbench for Verifying Pixel Processing real number model

```
function automatic void check_pix_out(ref real pix_out_sv [NUM_ROWS-1:0],
                                       ref real pix_out_ana [NUM_ROWS-1:0]);
  for (int row=0; row<NUM_ROWS; row++) begin
    if (close_enough_real(.real1(pix_out_sv[row], .real2(pix_out_ana[row], .tolerance(0.001))
      correct++;
    else begin
      $display("ERROR: %0t: For pix_out[%0d] SV=%1.3f and ANA=%1.3f", $time, row, pix_out_sv[row],
                                                                              pix_out_ana[row]);
         error++;
    end // if
  end   // for
endfunction
```

C. Consideration and Pitfalls

During verification of the pixel processing model an issue was found with the accuracy of signals *pix_out* from the *pixel_processing_ana* spice netlist. The analog value in *pixel_processing_ana* and *pixel_processing_vams* would be as expected but when passed through the Verilog-AMS wrapper to the testbench the *pix_out* values did not accurately represent the voltages. The voltages were often slightly off (10's of mV). An example is seen in Figure 5. The analog value of pix_out<0> is ~108mV both in the spice netlist and the Verilog-AMS wrapper. But when passed to the testbench as pix_out[0] the value is ~102mV. The discrepancy of 6mV is not acceptable.

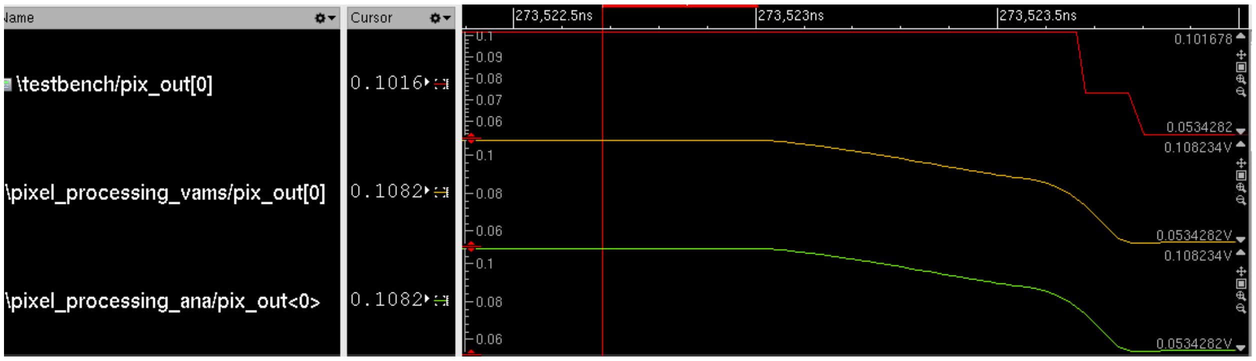

Figure 5: pix_out[0] error with default connrules

The solution was to override the *vdelta* parameter in the electrical to real (E2R) connect rule. By default the *vdelta* parameter is the supply (2.0V) divided by 64 which allows a resolution of only 31mV. The syntax to override this parameter for the Cadence AMS simulator is below. These directives would be placed in a simulation control file. As can be seen in Figure 6 the value of pix_out[0] is the same from analog schematic to SystemVerilog testbench.

```
amsd {
    ie vsup=2.0 connrules=E2R vdelta=0.001
    config cell=pixel_processing_ana use=spice
}
```

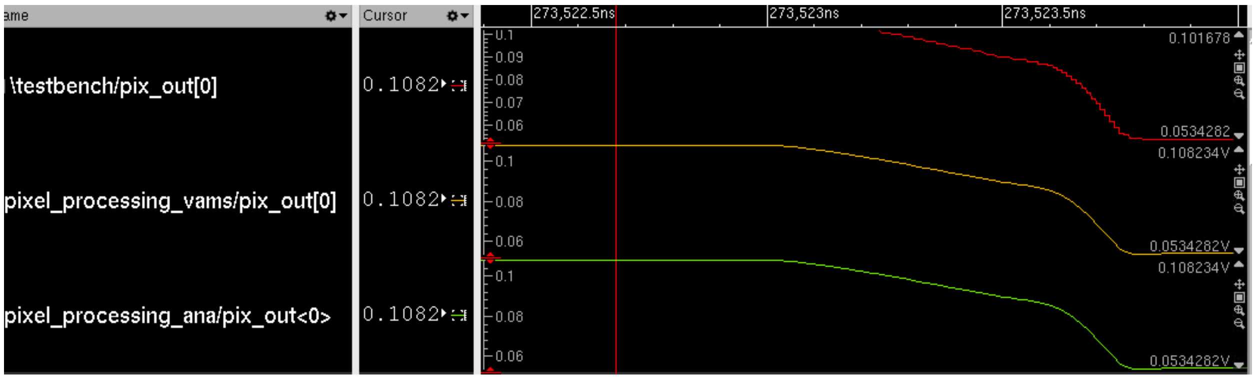

Figure 6: pix_out[0] with vdelta=0.001 in connrules

IV. RIGOROUS CHIP-LEVEL AMS VERIFICATION

Previous Chip-Level AMS verification in [1] was ad-hoc, unable to be run as regressions, and not self-checking. The testbench shown in Figure 7 eliminates these 3 deficiencies by adding self-checking transactors on the spi and test bus. All chip-level AMS tests are self-checking and run as part of regressions.

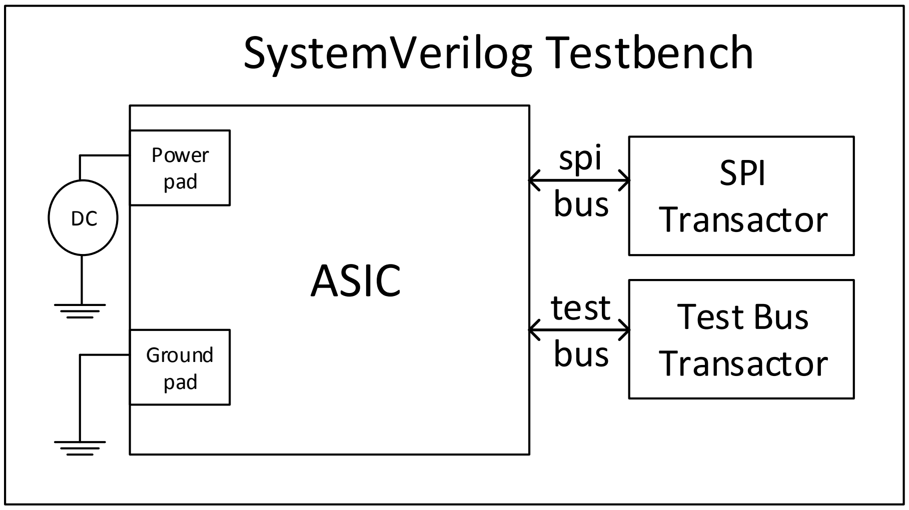

Figure 7: Chip-Level Testbench

A. *Chip-level Verification Methodology*

Due to the very long simulation times of chip-level AMS simulations it is important for all interested parties to agree on what functionality will be tested. The goal is not to verify analog functionality. That is what analog simulations are for. Likewise, there is no reason to verify digital functionality in an AMS simulation. The focus should be on verifying connectivity between digital and analog and device startup.

Another important decision is at what level the analog blocks will be represented. For example, schematic models of oscillators lead to long simulation times due to the continuous and high frequency oscillations. Keeping as many oscillators as possible in the digital domain will result in greatly improved simulation times. Usage of memory schematics may also lead to long simulations times or non-convergence. An RTL level model of the digital logic will simulate faster than a gate level representation. For a startup test, though, perhaps all analog blocks and pads will be represented as a schematic.

It is not necessary to go through the startup sequence for the regulator, POR circuitry, etc for every test. Use an initial condition to set the regulator output to the expected regulated voltage and toggle POR.

Due to the simulation time being dominated by SPI and test bus transactions it is imperative to get these correct prior to running an AMS simulation. Be sure to have a switch to run a quick digital simulation to debug these transactions. All chip-level tests were executed at the command line and were self-checking, a necessity for regression testing.

For this ASIC a Full Chip Test Plan document was created. An example of the description of 2 tests is below.

1) startup_gates
    a. Objective: Show that the ASIC will startup, release reset, and the regulator will supply the correct regulated voltages
    b. Views: gate level views of digital, schematic views for analog and pads
    c. Steps:
        i. Ramp VPWR
    d. Correctness:
        i. POR release
        ii. Regulator regulates at 2.0V
        iii. Oscillators startup when needed
        iv. Note current consumed by logic, memory, and analog
2) pixel_processing
    a. Objective: Show that the pixels are processed correctly
    b. Views: RTL level views of digital, high level model of oscillator in digital domain, high level model of regulator and memory, schematic views for remaining analog and pads.
    c. Steps:
        i. Set initial condition of 2.0V on regulator output
        ii. Toggle POR
        iii. Send SPI bus transaction to enable pixel processing
        iv. Read processed pixels through Test bus transactions
    d. Correctness:
        i. Processed pixels are read as expected.

An example of setting an initial condition on the regulator output is below:

```
ic chip_top_tb.chip_top.regulator.vpwr=2.0
```

B. Chip-level Regressions

Just as it is important to run periodic regressions on the digital logic, periodic regressions must be run at the chip-level to ensure a change late in the project does not introduce a bug. Automatically executing regressions at the chip level is more challenging than for the digital due to the need to check out data from disparate revision control systems. For example, SVN for the digital, and ICManage for the analog. Also, a spice netlist will need to be extracted from the schematic.

Unfortunately for this project the analog schematics were revision controlled with a home grown revision control system. This system was gui based and did not support a script to check out schematics. Schematic updating and spice netlist extraction was done manually which relied on good communication with the analog designer to know when a schematic changed. Because the chip-level tests were run from the command line executing a regression script with a *cron* job was easy.

V. RESULTS

In this chapter the impact of each improvement, both positive and negative, will be presented. First, the most important result, is that the ASIC was found to be a first-pass success.

A. Migration to a UVM-based testbench

The VHDL-based testbench was the evolution of 3 previous generations of image processing chips. A block diagram of the testbench is shown in Figure 8. The testbench utilized simplified VHDL models of analog blocks instantiated inside duv_top together with the digital logic (dig_top) and pad I/O models. Duv_top was converted from ASIC top-level schematics in order to verify top-level interconnection of logic and analog blocks. Duv_top was instantiated inside tb_top containing models of the external environment (model of SPI master – spi_mod and external clock generator). The testbench was controlled from a generator/monitor (gen_mon) block which was unique to every testcase. In order to control the analog models across hierarchy, gen_mon used a VHDL package instantiated signal bus to send model specific commands (package based signals in VHDL are connected to each block including the package alleviating the need to connect such information through the block interface).

The testbench supported stimulating pixel_array from images stored in a file as well as pseudorandom image generation. Unlike the new UVM-based testbench a top-level model predicting DUV responses does not exist in this testbench. Logic to pixel array protocol is checked using a FSM-based model inside pixel_array_mod – allowing autonomous checking of this functionality.

Each of the 165 testcases corresponded to a unique architecture of the gen_mon block – expected DUV responses were modeled in each testcase (in some cases they were hardwired constants, more frequently the VHDL behavioral modeling was used to calculate DUV response from hand generated or pseudorandom stimuli).

There are many similarities between the legacy VHDL-based testbench and the UVM-based testbench, however many features are missing. Pseudorandom generation is severally limited in VHDL and lacks constraining capabilities. There is very limited use of assertions (checking mostly clear invalid states). Requirement coverage is collected by a script parsing testcases gen_mon source code. Each verification engineer was responsible for including covered requirement numbers in the testcase header.

A downside to creating the UVM-based testbench was that more time was spent in the up-front development of the testbench instead of developing directed tests. The first reporting of bugs occurred later in the project than with the original VHDL-based testbench (see Figure 2). Another difficulty is finding verification engineers fluent in SystemVerilog and experienced in UVM.

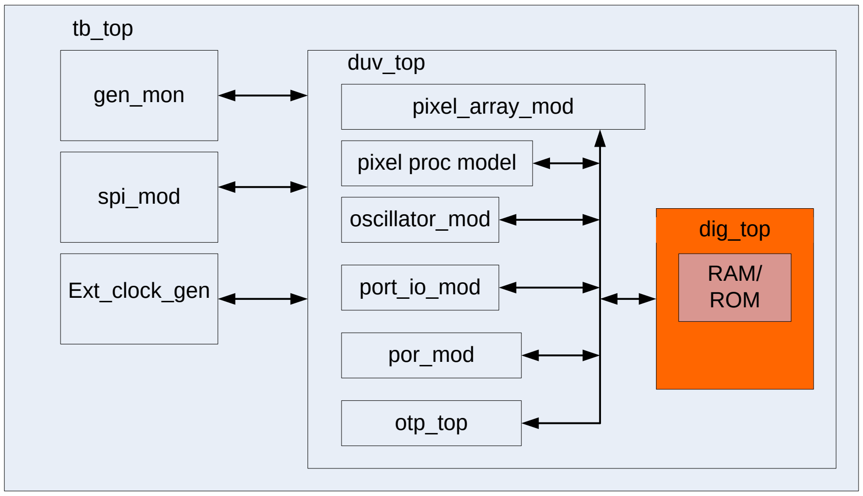

Figure 8: Original VHDL-based Digital Top-Level Testbench

B. *Real number modeling of analog blocks*

Once the RN models for the analog blocks were available, they were integrated into the top-level digital testbench. Configuration data was fed to the predictors and RN models along with transactions at stable points in the processing path. These points were picked to gather data at intermediate points or the final results. Resulting transactions were generated and passed to the scoreboard. These ‘predicted transactions’ were queued by the scoreboard to be compared with the ‘observed transactions’ when they arrived. The scoreboard is quite simple, really just a checker because of the use of predictors. A code snippet of the scoreboard is shown below:

```
`uvm_analysis_imp_decl(_expected)
`uvm_analysis_imp_decl(_observed)
class tb_scoreboard #(type T=uvm_sequence_item) extends uvm_scoreboard;
  `uvm_component_param_utils(tb_scoreboard #(T))

   uvm_analysis_imp_expected#(T, tb_scoreboard #(T)) expected_ap; // Analysis port for expected transactions
   uvm_analysis_imp_expected#(T, tb_scoreboard #(T)) observed_ap; // Analysis port for observed transactions
   uvm_tlm_analysis_fifo #(T) expected_results_af;                 // Analysis fifo

   // Other attributes to track count, enable logging, etc.
   …
   virtual function void write_expected (input T t);
     … simplified…
      expected_results_af.try_put(t);  // Put an expected transaction in the analysis fifo
   endfunction

   virtual function void write_observed (input T t);
      T expected_transaction;
    …simplified…
      expected_results_af.try_get(expected_transaction);  // Get expected transaction from analysis fifo
      compare_results(expected_transaction, t);   // Call compare function for expected transaction and observed
   endfunction

   virtual function void compare_results (T expected, T observed);
     .. simplified…
      if (observed.compare(expected)) begin
         // MATCH
      else
         `uvm_error({ … notification … })
   endfunction
endclass : tb_scoreboard
```

From an analog designers point of view the analog RN model was an extremely valuable tool which allowed more test cases to be examined and ultimately greatly improved the confidence in the design. In a design such as this with a large repetitive array there are hundreds if not thousands of nodes that, while are typically predictably connected, there are many cases such as on the array edges or in the multiplexers where mistakes can occur. These mistakes can be hidden until the right signal pattern comes along to expose them and use of analog RN modeling allowed a much better probability to uncover these bugs due to larger test case and structured output analysis.

C. Rigorous chip-level AMS verification

Having a rigorous methodology for chip-level AMS verification provided confidence in the results, as opposed to an ad-hoc methodology. By documenting in a full-chip test plan the functionality that would be tested along with the views of all blocks (schematic, high level, etc) the chip-level verification task was transparent and methodical. A self-checking testbench allowed regressions to be run which enhanced confidence that a late change did not introduce a bug.

VI. CONCLUSIONS

As was stated before, this ASIC was found to be a first pass success. This is quite an accomplishment for a full custom mixed signal ASIC. The UVM-based testbench along with the RN models of the analog blocks allowed the verification team to concentrate on finding bugs and generating interesting stimulus. The chip-level verification ensured that the analog to digital connections were correct and that the ASIC would startup.

VII. ACKNOWLEDGEMENTS

Thanks to EM Microelectronics-US, Tumbush Enterprises, and FirstPass Engineering for supporting the publication of this paper.

VIII. REFERENCES

[1] G. Tumbush and M. Becvar, “Design and Verification of an Image Processing CPU using UVM”, DVCon 2013

[2] G. Tumbush and C. Spear, “SystemVerilog for Verification: A Guide to Learning the Testbench Language Features”, 3rd Edition, 2012

[3] Verilog-AMS Language Reference Product Version 14.2, Cadence Inc., Jan 2015

[4] IEEE Standard for SystemVerilog, IEEE Computer Society, 2009