A UVM Testbench for Analog Verification: A Programmable Filter Example

Charles Dančak
Betasoft Consulting, Inc.
Contents

• UVM-Compliant Testbench Organization
• Bandpass Filter DUT and Auto-Extract
• Fixture Stimulus/Response Subcircuits
• Test Plan: Three "Flat" Sequences
• Top-Level Module; TLM Pathways
• Analog Assertions: \(I_{dd}\) and \(V_{bn}\)
• Scoreboard to Evaluate Response
• Reaching 100% Coverage
• Conclusions and Questions

Objective:
To verify an AMS circuit block in a UVM framework. Bring all UVM-style enhancements into AMS domain, via XMODEL flow.
UVM Testbench Organization

- **UVM-Based Environment**
- **Coverage Component (Stand-Alone)**
- **Stock UVM Sequencer**
- **Scoreboard (Compare Gain)**
- **TLM-1 Path For Packets**
- **Virtual Interface**
- **Interface Port**

**Diagram Notes:**
- COVG, fINT, MODE, fINT x MODE
- Functional Coverage
- TLM Port
- AGNTD
- Database
- Agent, Monitor
- AGNTM
- MON
- VMIF
- uvm_config_db
- FIX
- DUT
- DIF
- Filter
- Bus
- Reference Model
- EXP GAIN
- ACT GAIN
- GOLD_REF
- SEQ{1,2,3}
RC Bandpass Filter with Op-Amp

Virtuoso Schematic Editor

Add-On Tools

Write XMODEL Netlist

MODELZEN Auto-Extract

Op-Amp Subcircuit

Bias Supply (nMOS)

Eight Modes

Digital Control Codes

DUT Digital Input Pins: {ctl_byp, ctl_r2, ctl_c1, ctl_c2}

<table>
<thead>
<tr>
<th>Binary Value</th>
<th>Enumeration Literal</th>
<th>( f_{LO} )</th>
<th>( f_{HI} )</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>M0_40–060x2</td>
<td>40</td>
<td>60</td>
<td>2</td>
</tr>
<tr>
<td>0001</td>
<td>M1_40–040x2</td>
<td>40</td>
<td>40</td>
<td>2</td>
</tr>
<tr>
<td>0010</td>
<td>M2_20–060x2</td>
<td>20</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>M3_20–040x2</td>
<td>20</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>M4_40–120x1</td>
<td>40</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>M5_40–080x1</td>
<td>40</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>M6_20–120x1</td>
<td>20</td>
<td>120</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>M7_20–080x1</td>
<td>20</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>1XXX</td>
<td>Bypass/Power-Down</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ctl1_byp Bit

ctl_\_byp Bit
MODELZEN:
Auto-extracts SystemVerilog model of the entire filter from process-specific CMOS circuit.

```systemverilog
//Auto-extracted nMOS transistor M0:
nmosfet #( .W(1.2e-05), .L(1.8e-07), .Vth(0.6), .Kp(0.001267), .Ro(1.084e+06), . . .
 .Cgb('{4.659e-10,0,0}),  . . . . . .
 ) M0 (.d(tail), .g(vbn), .s(vss), . . .);
```

Ready-to-Simulate SystemVerilog Model (xmodel_prims nmosfet)

Op-Amp Subcircuit

Bandpass Filter
**Fixture Stimulus Chain**

**Stimulus Chain**

**Converter Primitive**
(real to xreal)

**Guideline:**
Encapsulate analog functions and xreal/xbit signals inside a fixture.

**f = 12 kHz (Random)**

**12-kHz Sinusoid**

**1 kHz Clock**

**Bandpass Filter DUT**
Fixture Response Chain

Initial Block 1kHz Clock PPA_IN, _OUT to MIF Bus

Measure Primitives (meas_pp)
Compatibility with UVM-AMS

- Submodule **FIXTURE** plays the same role as UVM-AMS **harness**; contains DUT.
- **XMODEL** primitives act as **analog resources**, to apply or measure signals.
- Verilog-AMS co-simulation or user-defined RNM modeling not required.

Accellera Architecture (to/from UVM Hierarchy)

[Diagram showing Accellera Architecture with various modules and connections]

www.accellera.org/resources/videos/uvm-ams-workshop-2021
Basic Feature Test

- The fixture's resources can exercise the DUT in any of its modes or states.
- To test normal filter operation, apply a sinusoid to Vin in any given mode.
- Measure peak-to-peak amplitudes at input and output, to compute gain.

Random Inputs ($f$, Mode)

$f = 12$ kHz
$A = 100$ mV

$PPA_{IN} = 200$ mV

$PPA_{OUT} = 104$ mV

$f = 12$ kHz
$A = 52$ mV

**Transcript Snippet**

<table>
<thead>
<tr>
<th>$f_{INT}$</th>
<th>MODE</th>
<th>STATE</th>
<th>$g_{ACT}$ (OUT/IN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 kHz</td>
<td>M7</td>
<td>FILTER</td>
<td>0.5163 1803</td>
</tr>
</tbody>
</table>
Data Packet Objects

- Each packet represents one transaction, with stimulus and response data.
- Same packet type reused for both TX_PKT (to DUT) and RX_PKT (from DUT).
- A UVM sequence can generate a series of constrained randomized packets.

```lean
class PACKET extends uvm_sequence_item;
    `uvm_object_utils(PACKET)
    int TAG = 0; //Packet ID.
    //Random frequency (10--120 kHz):
    rand int fINT; //kHz.
    real fREAL; // Hz.
    //Random passband mode (M0..M7):
    rand MODE_t MODE;
    real PPA_IN, PPA_OUT; //V.
    //Constrain random variables:
    «Constrain fINT and MODE.»
    function void post_randomize();
    . . . . . 
endclass: PACKET
```

A Dynamic Object
Not Fixed Component

Post-Randomize:
Convert fINT to real; enforce cyclic MODE.
Test Plan: Sequence 1 and 3

**Seq1:** Seq_Filter
Perform basic feature test.

**Seq2:** Seq_Bypass
Power-down; bypass DUT.

**Seq3:** Seq_Resume
Resume basic feature test.

---

Our simple test suite will apply three consecutive (flat) sequences to the DUT.

```c
/* PACKET CODE SNIPPET */
//Constrain fINT to its range:
constraint fRANGE_con {
  fINT inside {[10:120]}; //kHz.
}
//Constrain MODE to unused values:
constraint XCLUDE_con {
  !(MODE inside {USED}); //Queue.
}

/* SCOREBOARD CODE SNIPPET */
* Compute actual gain: */
real PPA_IN, PPA_OUT;
.
.
case (TX_PKT.STATE)
  FILTER, RESUME:
    gACTUAL = PPA_OUT / PPA_IN;
  .
endcase
```
 Daw the DUT recover after long power-down?

Test Plan: Sequence 2

Check analog DUT properties during power-down, to enhance overall coverage.

Concurrent Assertion (SVA Syntax)

/* FIXTURE CODE SNIPPET: * Check nMOS bias level Vbn * after entering WAIT_1 state: */
CK_BIAS: assert property (BIASING_pro)
  uvm_report_info("PROPS", «pass_message», UVM_LOW);
else
  uvm_report_warning("PROPS", «fail_message», UVM_HIGH);

Are Analog Properties Valid SVA?

Four DUT States

Assert: VBN = 700 mV ± 50

xreal to real SVA Variable

SEQ2: SEQ_BYPASS
Power-down; bypass DUT.
• With analog resources encapsulated in fixture, higher layers are all digital.
• Compliant with UVM and compatible with Accellera UVM-AMS standard.
UVM Sequence: Purely Digital

Definition of SEQ1

Power On; Set State to FILTER

Randomize f and MODE

Details of pin-level timing at DUT left entirely up to driver/monitor code.
Test Suite: Sequences 1, 2, 3

//THREE-SEQUENCE TEST SUITE:
class TEST_SUITE extends uvm_test;
  .  .  .  .  .  .
«Get knobs from command-line processor.»
task run_phase(uvm_phase phase);
  SEQ_FILTER SEQ1; //Normal filtering.
  SEQ_BYPASS SEQ2; //Power-down, wait.
  SEQ_RESUME SEQ3; //Resume filtering.
//Finish up last clock cycle:
«scope».set_drain_time(this, 1000us);
phase.raise_objection(this);
//Launch sequence 1:
SEQ1 = SEQ_FILTER::type_id::create("SEQ1");
SEQ1.TRIALS = TRIALS;
SEQ1.start(E.AGNTD.SQR);
//«Continued at right...»

//Launch sequence 2:
SEQ2 = «Call factory-create.»
SEQ2.TRIALS = TRIALS;
SEQ2.INACTV = INACTV;
SEQ2.start(E.AGNTD.SQR);
//Launch sequence 3:
SEQ3 = «Call factory-create.»
SEQ3.TRIALS = TRIALS;
SEQ3.INACTV = INACTV;
SEQ3.start(E.AGNTD.SQR);
phase.drop_objection(this);
endtask: run_phase
endclass: TEST_SUITE

Start SEQ1 on Base Sequencer

TRIALS
INACTV
Settling Time
Let Phasing System Halt the Simulation
Top module **UVM_TB** is the block actually *elaborated* by a simulator such as **VCS**.

```bash
bash> xmodel -f man.f
```

**Method Invoked at Time 0**

**Clock Bundled Into DIF and MIF**

**Never Directly Instantiated**

**XMODEL Compile Command**
Transcript with Max $|g\text{ERROR}|$

<table>
<thead>
<tr>
<th>TX_TAG</th>
<th>fINT kHz</th>
<th>MODE Name</th>
<th>STATE Name</th>
<th>RX_TAG</th>
<th>gACT (OUT/IN)</th>
<th>gEXP (HSPICE)</th>
<th>gERROR (gACT-gEXP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34</td>
<td>M7 FILTER</td>
<td>FILTER</td>
<td>1</td>
<td>0.80081911</td>
<td>0.79615200</td>
<td>0.00466711</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td>M0 FILTER</td>
<td>FILTER</td>
<td>2</td>
<td>1.18788707</td>
<td>1.17873200</td>
<td>0.00915507</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>M4 FILTER</td>
<td>FILTER</td>
<td>3</td>
<td>0.24548713</td>
<td>0.24592900</td>
<td>0.00044187</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>M1 FILTER</td>
<td>FILTER</td>
<td>4</td>
<td>0.86157682</td>
<td>0.85269000</td>
<td>0.00888682</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>.. FILTER</td>
<td>.. FILTER</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
<tr>
<td>37</td>
<td>XX</td>
<td>MX BYPASS</td>
<td>BYPASS</td>
<td>37</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
</tr>
<tr>
<td>38</td>
<td>XX</td>
<td>MX BYPASS</td>
<td>BYPASS</td>
<td>38</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
</tr>
<tr>
<td>39</td>
<td>XX</td>
<td>MX BYPASS</td>
<td>BYPASS</td>
<td>39</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
</tr>
<tr>
<td>40</td>
<td>XX</td>
<td>MX WAIT_1</td>
<td>WAIT_1</td>
<td>40</td>
<td>0.00000000</td>
<td>0.00000000</td>
<td>0.00000000</td>
</tr>
<tr>
<td>41</td>
<td>110</td>
<td>M4 RESUME</td>
<td>RESUME</td>
<td>41</td>
<td>0.69637779</td>
<td>0.68144600</td>
<td>0.01493179</td>
</tr>
<tr>
<td>42</td>
<td>76</td>
<td>M7 RESUME</td>
<td>RESUME</td>
<td>42</td>
<td>0.70367702</td>
<td>0.69537100</td>
<td>0.00830602</td>
</tr>
<tr>
<td>43</td>
<td>12</td>
<td>M6 RESUME</td>
<td>RESUME</td>
<td>43</td>
<td>0.51965732</td>
<td>0.51948100</td>
<td>0.00017632</td>
</tr>
<tr>
<td>44</td>
<td>63</td>
<td>M1 RESUME</td>
<td>RESUME</td>
<td>44</td>
<td>0.89859033</td>
<td>0.88735100</td>
<td>0.01123933</td>
</tr>
</tbody>
</table>

Maximum $|g\text{ERROR}|$ per run: 0.01713050

Enhance coverage during bypass?

XMODEL Accurate to 2% of HSPICE

Simulation run for 36 trials, printed in UVM tabular format as SCORECARD object.
//Bias recovery leads:
  xreal VBN_x;  real VBN;
//Bias node inside of DUT:
  assign VBN_x = DUT.vbias;
//Bias voltage at TICK edge:
  meas_value M_VAL( //Real-valued output:
    .in(VBN_x),.out(VBN), .trig(TICK_x)
  );
//Check nMOS bias during WAIT_1:
  property BIASING_pro; //700 mV ± 50.
    $rose(FREQ_IN.STATE == WAIT_1) |->
    (VBN >= 0.650) && (VBN <= 0.750);
endproperty: BIASING_pro
CK_BIAS: assert property (BIASING_pro)
  «Report pass; else report failure.»

Fixure Primitive

Assorted Property

Trigger (xbit)

//Default clocking event:
  @(posedge FREQ_IN.PKT_CLK)

Supplements UVM test sequences; localized to specific circuit nodes or branches.
Filter Waveforms with Assertions

Analog/Mixed-Signal Assertions for Filter Properties

<table>
<thead>
<tr>
<th>Assertion Label</th>
<th>Variable Checked</th>
<th>Antecedent Condition</th>
<th>Property Expression</th>
<th>XMODEL Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK_LEAKG</td>
<td>Idd (Leakage)</td>
<td>STATE==BYPASS</td>
<td>Idd(max) ≤ 5 nA</td>
<td>iprobe, meas_max</td>
</tr>
<tr>
<td>CK_BIAS</td>
<td>Vbn (nMOS Bias)</td>
<td>STATE==WAIT_1</td>
<td>Vbn = 700 ± 50 mV</td>
<td>meas_value</td>
</tr>
</tbody>
</table>

TRIALS = 6
INACTV = 4

Analog Assertions (Idd, Vbn)
TLM Broadcast Pathway

- TLM raises level of abstraction for data transfers between components.
- This path broadcasts packets to two distinct destinations: SCB and COVG.
- FIFOs enable SCB to get time-offset TX_PKT, RX_PKT transaction streams.
The Scoreboard

- Scoreboard compares actual versus expected gain for each transaction.
- Computes and tracks discrepancy in gain: \( g_{ERROR} = g_{ACTUAL} - g_{EXPECT} \).
Coverage Collection

```cpp
/* STAND-ALONE COVERAGE OBJECT */
class COVERAGE extends uvm_subscriber #(PACKET);

covergroup CVG;
//Applied input-frequency bins:
  FREQ_cvg: coverpoint TX_PKT.fINT
    {
      //kHz
      bins B10  = {[ 10: 19]};
      bins B20  = {[ 20: 39]};
      ....
      bins B100 = {[100:120]};
    }
//Applied input-mode values:
  MODE_cvg: coverpoint TX_PKT.MODE;
//Cross-coverage:  MODE x fINT
  CROSS_cvg: cross MODE_cvg, FREQ_cvg;
endgroup: CVG

No guarantee a test suite covers filter operation over its entire range of \( f \), modes.
```
Reaching 100% Cross-Coverage

VCS Benchmark Runs

- Probability of a transaction covering an \( f \times MODE \) bin is 1/48, or about 2%.
- Can reduce run length by merging coverage data for two random seeds.
- Effective method when nightly regression suites are being run anyway.

bash>

xmodel -f man.f

bash>
simv +TRIALS=152 +INACTV=4 \n +ntb_random_seed=3947

bash>

mv simv.vdb simv1.vdb

bash>

«Repeat for second seed 9573.»

bash>

urg -dir simv1.vdb \n -dir simv2.vdb \n -dbname merged.vdb

VCS

bash

Script

Uncovered Bins at 96%

Uncovered bins

<table>
<thead>
<tr>
<th>MODE_cvg</th>
<th>FREQ_cvg</th>
<th>COUNT</th>
<th>AT LEAST</th>
<th>NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>[auto_M1_40-040x2][B40]</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>[auto_M2_20-060x2][B100]</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions and Questions

- An XMODEL-based UVM-compliant testbench to verify programmable AMS DUT.
- Brings UVM enhancements—modular components, configurable test length, phasing, TLM pathways, functional-coverage objects—into the AMS domain.
- The testbench utilizes standard UVM components to verify filter's transfer gain.
- During power-down mode, $I_{dd}$ and $V_{bn}$ are checked using analog assertions.
- XMODEL-based flow is accurate to 2% of HSPICE, with no sacrifice in speed.

Questions or comments?

For more details on XMODEL primitives, driver/monitor code, scoreboard methods, analog assertions:
C. Dančak, “SystemVerilog OOP Testbench, Parts {1,2}:
www.researchgate.net/publication/346061868_SystemVerilog_OOP_Testbench_for_Analog_Filter_A_Tutorial_Part_1
www.researchgate.net/publication/350412143_SystemVerilog_OOP_Testbench_for_Analog_Filter_A_Tutorial_Part_2