Application of Metamorphic Testing to Mixed Signal Systems with Behavioral Models

Daniel Cross
Cadence Design Systems
+1-512-342-5560
12301 Research Blvd Ste 200
Austin, TX 78759
danielcr@cadence.com

***Abstract*-Metamorphic Testing is a methodology originating within the software testing regime that allows testing of modules for which no good test oracle exists. An overview of metamorphic testing is provided with examples from software testing. Then, an application of metamorphic testing to a mixed-signal subsystem modeled with SystemVerilog real constructs is described. Simulation results are presented showing metamorphic testing of the mixed-signal system. The intent of this demonstration is to introduce the concept of metamorphic testing and encourage its application in mixed-signal verification.**

I. Introduction to Metamorphic Testing

A common testing methodology for new software modules is to present the module with a variety of inputs and verify that the module produces the correct or expected outputs in response. The resource relied upon to determine the correct or expected outputs is known as a *test oracle* [1]. Construction of a proper test oracle can sometimes be a time consuming, expensive, or ultimately impossible task. In these cases, a different testing methodology is needed.

To help address this need, the use of metamorphic testing in the software realm was introduced over 20 years ago [1]. This method employs the identification of *metamorphic relations* between different inputs and their corresponding outputs that must be maintained by the module under test if it is to perform its intended function.

As a simple example, assume a module that produces output $f(x)$ in response to input x, $f(y)$ in response to input y, and $f(x+y)$ in response to $x+y$. an example of a metamorphic relation (MR) for this module might be $f(x+y) = f(x)+f(y)$. A metamorphic test will simply assert that the relationship is true. Note that no knowledge of whether $f(x)$, $f(y)$, or $f(x+y)$ are the correct outputs for their respective inputs x, y, and $x+y$ is required.

A more concrete example (see [1]) might apply to a cryptographic application requiring math operations on very large numbers, for which existing math functions are unreliable. Custom functions written for the application must be tested, but any existing function that might be reliably used as a test oracle would be a perfect candidate to use in the application, and then testing would not be required. In the absence of a test oracle, metamorphic testing might be used as follows.

Assume a math function is required to calculate the value of e^x. A MR for this function is $e^x \times e^x = e^{2x}$. Once again, knowledge of the actual correct value of e^x is not required.

A more complex example of metamorphic testing from the realm of machine learning applications ([2], [3]) is as follows. Suppose the unit under test is an image classifier that counts the number of cats in an input image, as illustrated in Figure 1. If the input image is mirrored, the number of cats detected should be the same, therefore a metamorphic relation for a pair of tests on an image and a mirrored copy is $count_{MIRRORED}=count_{NORMAL}$. Similarly, if the input image is composed of four copies of the original image, the number of cats detected should be multiplied by four. Thus, the metamorphic relation for this pair of tests is $count_{REPEAT}=4 \times count_{NORMAL}$. No prior knowledge of the number of cats pictured is required for this type of testing. Construction of a traditional test oracle would require humans to manually count and classify all of the test images, a labor-intensive task. Metamorphic testing provides a reliable means of unit testing without preliminary classification of the test data by humans.

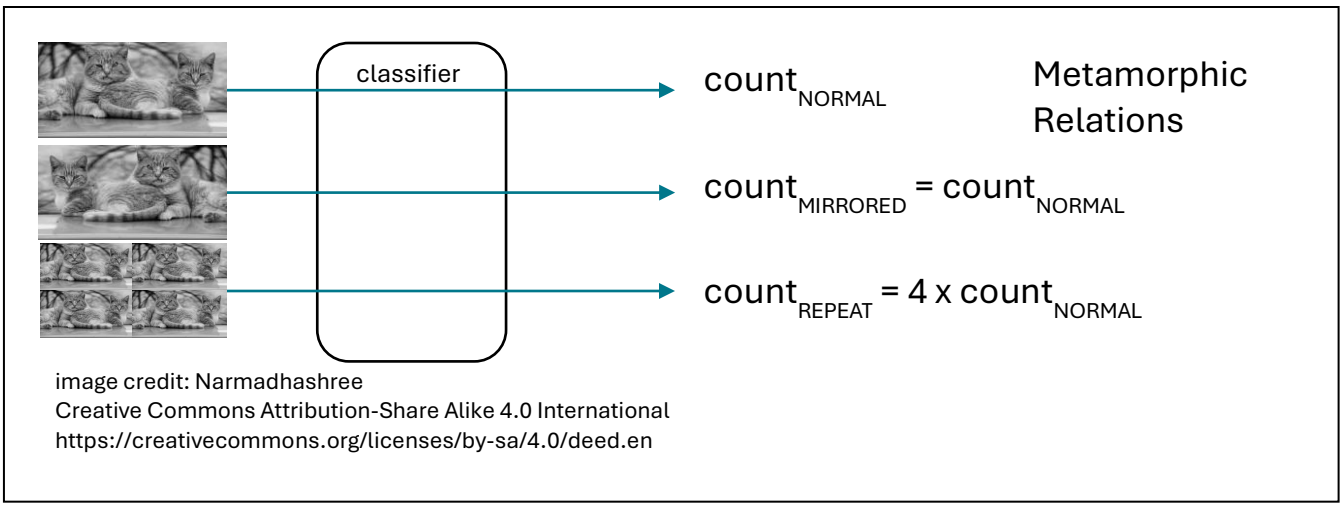

Figure 1. Machine learning image classifier application test with metamorphic relations

II. Metamorphic Testing in Hardware Verification

A. *Background for this work*

For integrated circuit verification, the Universal Verification Methodology (UVM) suggests that a Bus Functional Model (BFM) of the device should be used as a test oracle to check the system response to a stimulus. However, when verifying a complex mixed signal system the analog system components are often represented by behavioral models for simulation performance; and generation of a BFM would often necessitate reiteration of the models used for verification. In addition to compromising simulation performance (because of the need to simulate two instances of every model), this would reduce confidence in the verification results — since the BFM response will always match that of the device under verification and actual system function is left unverified. Metamorphic testing offers a way out of this conundrum.

As an example, refer to [4] in which the authors apply metamorphic testing principles to verification simulations of an industrial Phase-Locked Loop (PLL) subsystem, pictured in Figure 2. Rather than trying to determine the proper response to every input, they identify eight MRs that must be met for the PLL to properly function. One of these MRs states that if two similar tests are performed with reference frequencies related by a constant C that is close to unity, then the locked frequencies of the two tests should also be related by that same constant. More succinctly,

IF: Test 1: F_{REF} = F1
Test 2: F_{REF} = F2 = C x F1
THEN: F_{DIV} [Test 2] = C × F_{DIV} [Test 1]

The authors in [4] found that when C = 1.01, the PLL frequency locked to a very low subharmonic of F_{REF}. They investigated and found a dead-zone bug in the phase-frequency detector. This reported result demonstrates the applicability of metamorphic testing to mixed-signal hardware verification.

B. *Description of the subsystem used in the current work*

This work extends this application to a different type of mixed-signal subsystem in which a complex analog-digital interaction is verified using SystemVerilog real number modeling to represent the analog functions. The basis for this demonstration uses a receive baseband stage from a radio transceiver, analog to digital converter (ADC), and an Automatic Gain Control (AGC) block that acts upon the converted digital signals, represented in Figure 3. The analog I and Q baseband amplifiers have digitally controlled gain and filtering. The ADC is a successive approximation register (SAR) type, that interleaves samples from the I and Q amplifiers. A digitally controlled reference voltage generator within the ADC sets the center point and LSB size. The digital AGC block computes the $I^2 + Q^2$ signal magnitude and uses that to set the gain of the analog amplifiers such that the ADC dynamic range is maximized. It also computes a Received Signal Strength Indicator (RSSI) that represents the actual magnitude, in dB from a reference level, of the input analog signal (before gain control).

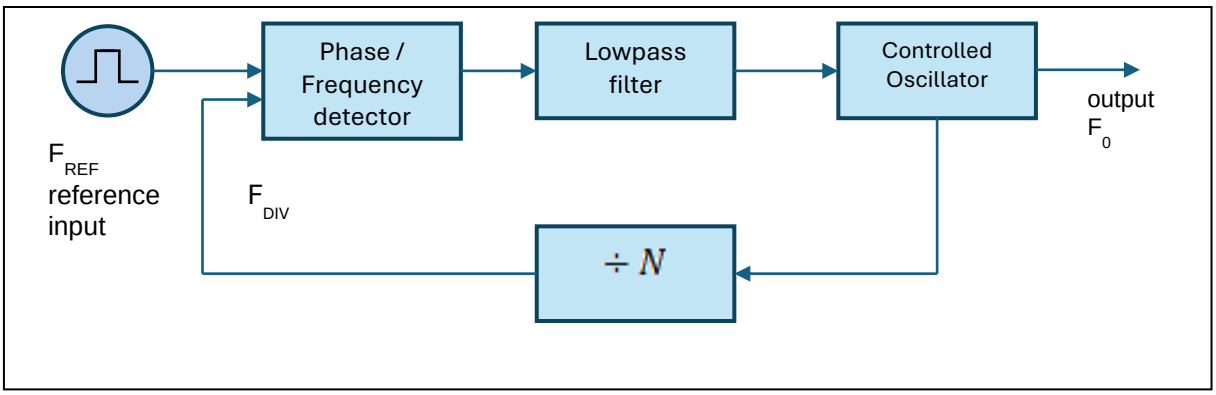

Figure 2. Phase Locked Loop (PLL) diagram

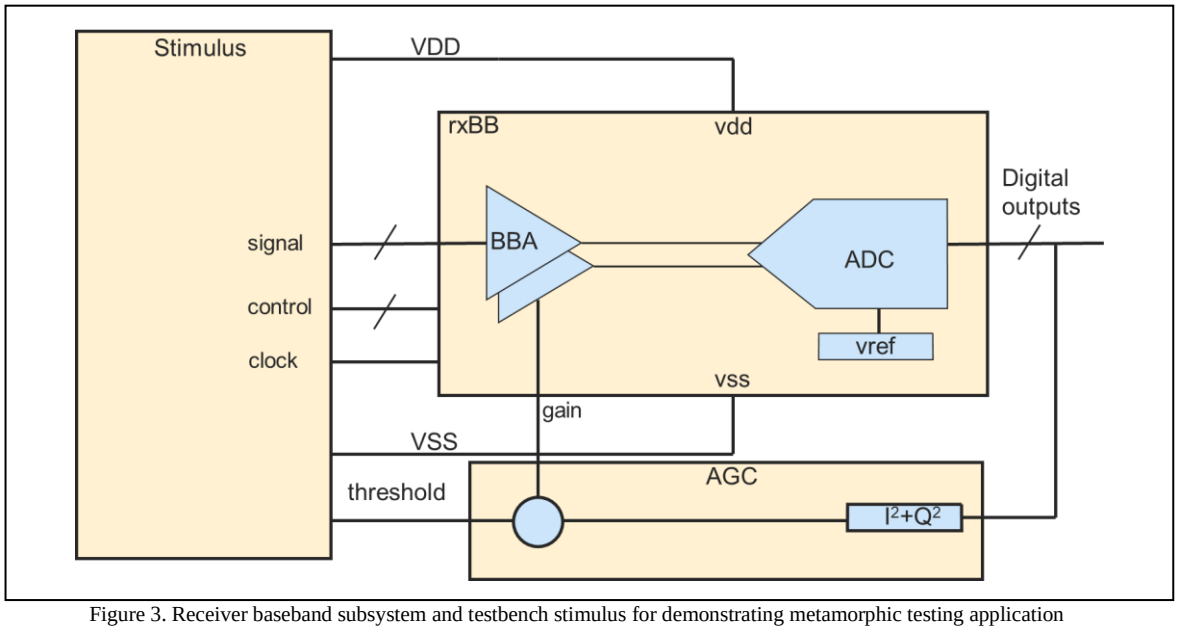

Figure 3. Receiver baseband subsystem and testbench stimulus for demonstrating metamorphic testing application

III. Testbench and Metamorphic Testing

A. *Metamorphic Relations*

Several metamorphic relations were identified for this subsystem. They apply for an arbitrary input waveform $w(t)$ with I and Q components $w_i(t)$ and $w_q(t)$ given the following reasonable assumptions:

- $w(t)$ is continuous
- $w(t)$ is band-limited
- the $\pi/2$ phase relationship between $w_i(t)$ and $w_q(t)$ is maintained
- $|\, w(t) \,|$, $|\, 2w(t) \,|$, and $|\, 0.5w(t) \,|$ are within the AGC range

Taking $H(w(t))$ as the output of the ADC with input $w(t)$, and $R(w(t))$ the RSSI level with input $w(t)$, the MRs are as follows:

MR1. $H(2w(t)) = H(w(t))$
MR2. $R(2w(t)) = R(w(t)) + 6$ dB
MR3. $H(0.5w(t)) = H(w(t))$
MR4. $R(0.5w(t)) = R(w(t)) - 6$ dB

These MRs appear to be trivial. MRs 1 and 3 can be stated in words as "the action of the AGC should keep the output of the ADC at a constant level." Likewise, MRs 2 and 4 can be described as "the RSSI calculation should take into account the gain correction applied when indicating the input signal strength." However, rendering these natural language specifications in a mathematical way that can be coded in an automated test is not so obvious. Metamorphic testing gives us a paradigm for embodying the expected system behavior in a way that does not

necessitate construction of a test oracle in our testing framework. As stated above, these MRs only hold given the assumptions enumerated. Outside those conditions, other MRs, or different testing methods, are needed.

B. Stimulus and random constraints

To keep this demonstration simpler, sinusoidal input was used instead of attempting to generate truly arbitrary waveforms. Parameters of amplitude and frequency were varied randomly (see Figure 4). The reference setting that adjusts LSB size was also randomized.

```
class c_testVals;

   rand real testAmpl = 0.1;
   rand logic unsigned [3:0] testReftrim2 = 0.0;
   rand real testFreq = 500e3;

   constraint cAmpl   {testAmpl inside {[50e-3:0.25]};}  // peak amplitude of input
   constraint cFreq   {testFreq inside   {[100e3:2e6]};}   // Frequency of input in Hz
   constraint cRef2   {testReftrim2 inside {[0:7]} ;}    // trim LSB size

endclass : c_testVals
```

Figure 4. SystemVerilog class for randomizing stimulus parameters, with constraints

Randomized values were applied within the stimulus block to generate stimulus. Some of the SystemVerilog used to generate the stimulus is shown in Figure 5. The $\pi/2$ phase relationship between the I and Q signals is enforced by using `$sin()` and `$cos()` to generate values. A 40ns strobe clock triggers computation of the next sample of the I and Q sinusoids, using the currently set values for amplitude and frequency. These sample values are treated as differential quantities, and single-ended signals that will drive the differential input baseband amplifiers are generated by dividing the differential values by 2 and adding to or subtracting from a common mode value set by parameter. The test pattern begins by randomizing an instance of the c_testVals class (shown in Figure 4). The randomized values are assigned to amplitude, frequency, and reference value variables. Later in the simulation, the amplitude variables are set to 2X the randomized value. Still later (not shown in the figure), the amplitude variables are set to one half the randomized value.

```
  parameter real vCM = 0.75;
  parameter real sampRate = 40e-9;
  c_testVals testVals=new;
  always #(sampRate/2.0) clk = ~clk;
  always @ (posedge clk) begin
     iDiff = iAmpl * $sin(`M_TWO_PI * iFreq * $realtime);
     qDiff = qAmpl * $cos(`M_TWO_PI * qFreq * $realtime);
  end
  assign inip = vCM + iDiff/2;
  assign inim = vCM - iDiff/2;
  assign inqp = vCM + qDiff/2;
  assign inqm = vCM - qDiff/2;
  initial begin
     if (testVals.randomize()) begin
       refTrim2 = testVals.testReftrim2;
       iFreq = testVals.testFreq;
       qFreq = testVals.testFreq;
       iAmpl = testVals.testAmpl;
       qAmpl = testVals.testAmpl;
       .  .  .
       iAmpl = testVals.testAmpl*2.0;
       qAmpl = testVals.testAmpl*2.0;
       .  .  .
     end
end
```

Figure 5. Testbench stimulus SystemVerilog showing randomization of parameters and input signal generation

C. Measurement and applying metamorphic relations

A measurement SystemVerilog module extracts the peak-to-peak amplitude, in LSBs, of the digital ADC outputs. This module also performs a moving average of the RSSI indication from the AGC. A measurement window is set

by parameter. Measurements start at a strobe signal generated by the stimulus, and end when the measurement window expires. Upon receipt of a new strobe, the measurement module resets the min and max counters of the peak-to-peak measurement, and restarts the RSSI moving average. Measurements are retained in arrays (**`ampl[0:2]`** and **`rssi[0:2]`**) until the end of the simulation. Within the arrays, element 0 represents the initial measurement, element 1 represents the measurement at +6dB input, and element 2 represents the measurement at -6dB input. Key portions of the measurement module are shown in Figure 6.

```
parameter WINDOW = 200e-6;
always @ (posedge strobe) begin
   timer = $realtime + WINDOW;
   max = 0;
   min = 255;
   rssiCount = 0;
   while ($realtime < timer) #(1us);
   ampl[measCount] = (max - min);
   measCount++;
end
always @ (posedge iClk) begin
   if ($realtime < timer) begin
     if (iVal > max) max = iVal;
     if (iVal < min) min = iVal;
   end
end
always @ (rssiVal) begin
   if ($realtime < timer) begin
      rssi[measCount] = (rssi[measCount]*rssiCount + real'(rssiVal/256.0))/(rssiCount + 1.0);
      rssiCount++;
   end
end
```

Figure 6. Measurement module SystemVerilog showing min/max peak-to-peak extraction and RSSI moving average calculation

Also within the measurement module, a task applies the metamorphic relations and produces a text report on the stdout and/or the log file. The SystemVerilog that implements this task and the metamorphic relation enforcement is shown in Figure 7. Note that the test comparisons in this task could just as easily be embodied in SystemVerilog assertions or any other standard test construct used in a verification testing flow.

```
   task measurementReport ();
      bit amplTest, rssiTest;
      $display("******* Measurement Report *******");
      $display("Peak-to-peak magnitudes           ");
      for (int i=0;i<3;i++)
         $display("Measurement %d           %d",i,ampl[i]);
      if ((ampl[0] == ampl[1]) && (ampl[0] == ampl[2]))
      $display("******* TEST PASSED        *******");
      else
      $error("******* TEST FAILED        *******");
      $display("*******  ******* *******   *******");
      $display("RSSI Values           ");
      for (int i=0;i<3;i++)
         $display("Measurement %d           %3.3f",i,rssi[i]);
      if (((rssi[0]-rssi[2]) > 5.5) && ((rssi[0]-rssi[2]) < 6.5) &&
          ((rssi[1]-rssi[0]) > 5.5) && ((rssi[1]-rssi[0]) < 6.5))
      $display("******* TEST PASSED        *******");
      else
      $error("******* TEST FAILED        *******");
   endtask
```

Figure 7. Metamorphic testing SystemVerilog

IV. Simulation Results

A typical simulation result is shown in Figure 8. The trace “inip” is one of the four inputs to the I and Q baseband amplifiers. “gain” is the gain control feedback signal from the AGC. It can be seen reacting to and tracking the changes in amplitude of inip. iVal and qVal are analog representations of the digital ADC outputs. After AGC convergence, they can be seen to have a consistent peak-to-peak magnitude even as the input signal doubles and

then is divided in half. “ampl” is an array in the measurement module that holds the peak-to-peak amplitude measurements. In the figure it can be observed that all three measurements match, as expected.

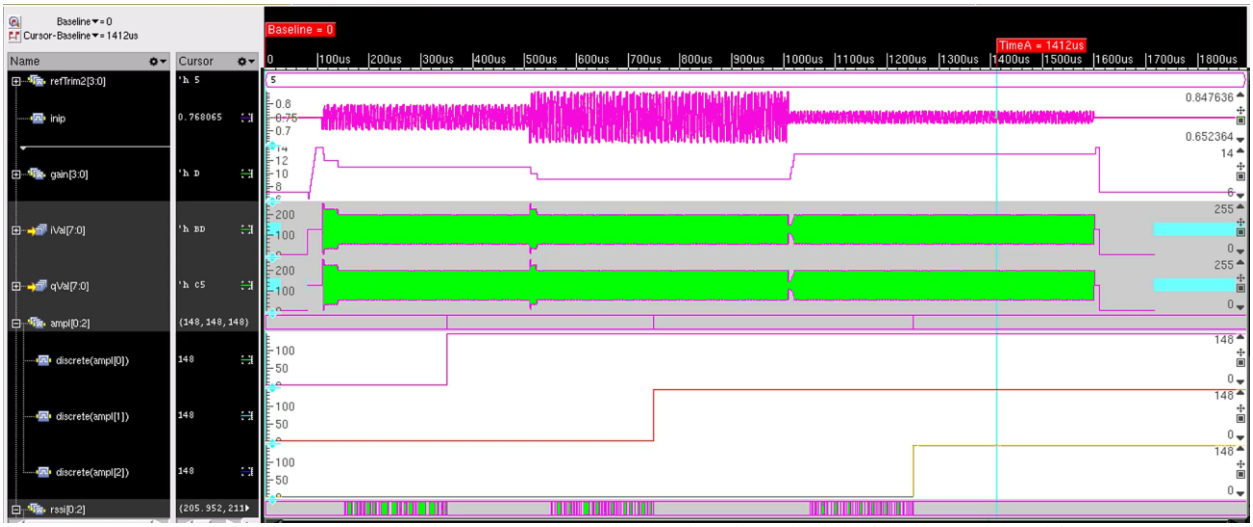

Figure 8: Simulation waveforms showing metamorphic testing in progress. “iVal” and “qVal” are analog representation of the ADC outputs. “gain” indicates the AGC response to the changing input signal strength (“inip”). As the simulation progresses, the measurement array “ampl” is filled with ADC values at certain measurement points. Near the end of the simulation, the marker (at 1412us) reveals that the three measurements are identical (at 148).

Typical output of the measurement report task is shown in Figure 9. Measurements 0 represents initial measurements, Measurements 1 are compared to Measurements 0 to test MR1 and MR2. Measurements 3 are compared to Measurements 0 to test MR3 and MR4. The tests passed because the three peak-to-peak magnitude Measurements matched, and RSSI Measurements 1 and 2 were plus and minus 6 dB from Measurement 0.

```
******* Measurement Report *******
Peak-to-peak magnitudes
Measurement            0                   148
Measurement            1                   148
Measurement            2                   148
******* TEST PASSED         *******
*******  ******* *******    *******
RSSI Values
Measurement            0           205.952
Measurement            1           211.989
Measurement            2           199.947
******* TEST PASSED         *******
```

Figure 9. Metamorphic test result report

V. Conclusion

This demonstration shows how the principles of metamorphic testing can be applied to verify a mixed signal subsystem. Implementation of tests that check subsystem behavior against the defined metamorphic relations in SystemVerilog was presented. Some simulation results were shown, with metamorphic test results. Although SystemVerilog real number models were used in this demonstration, application of the metamorphic testing paradigm is not limited to that type of modeling. In most circumstances, device-level schematic representations can be substituted for the DUT with no loss of applicability (though the simulations will likely take more time). However, the use of SystemVerilog in the testbench allows inclusion of all of the built-in verification aids of the language. Constrained random stimulus can generate test inputs (as we have shown). Coverage constructs such as the one shown in Figure 10 can monitor functional signals to assess how completely the design operating space has been tested. Assertions can determine whether the MRs are maintained for valid inputs. And all aspects of the methodology are compatible with Universal Verification Methodology (UVM) construction. Metamorphic tests contribute to design coverage and may provide means of testing portions of the input space otherwise inaccessible.

Of course, MRs are only valid over a given range of inputs and test conditions, as we have shown. Outside of those conditions, other MRs or different testing methodologies are required.

The only limitation to the applicability of Metamorphic Testing we see is the difficulty of determining appropriate MRs. For example, the present example could be extended to include an entire Radio Frequency receive signal path, including frequency translation and selectivity. However, MRs that describe the relationship between system responses to different RF inputs might be impractically complex.

```
covergroup RXBB_coverage
 @ ( posedge (sampleClkp & !sampleClkn)
      iff ((en === 1'b1) && (refEn === 1'b1)) );

 cov_gain:    coverpoint gain { bins gainSets[13] = {[1:14]};
                         ignore_bins unused = {0,15};
                         }
 IAMPL:      coverpoint iAmpl {
                bins vIn0 = {[0.0:0.05)};
                bins vIn1 = {[0.05:0.15)};
                bins vIn2 = {[0.15:0.2)};
                bins vIn3 = {[0.2:0.25)};
         ignore_bins vIn4 = {[0.25:$]}; }
 ADCI:        coverpoint outi { bins cADCI[8] = {[0:$]}; }
 ADCQ:        coverpoint outq { bins cADCQ[8] = {[0:$]}; }
 BBBW:        coverpoint bwTrim { bins rxBW[] = {[0:$]}; }
 REF1:        coverpoint refTrim1 { bins rxREF1 = {8};
                             ignore_bins unused = {[0:7],[9:15]};
                           }
 REF2:        coverpoint refTrim2 { bins rxREF2 = {[0:7]};
                             ignore_bins unused = {[8:15]};
                           }
 option.comment = "******* RXBB Block Coverage ***************";
endgroup
```

Figure 10. Sample covergroup showing functional coverage reached by Metamorphic Testing

In general, there is no algorithmic method for deriving the Metamorphic Relations that this type of testing relies upon. Instead, identification of appropriate and useful MRs requires close analysis of the system, its operational requirements, and its expected behaviors by skilled experts familiar with the system. While system specifications and requirements will often help determine which MRs must be true for proper function, the MRs differ from requirements in their mathematical specificity and applicability to testing in the absence of a proper test oracle.

This demonstration can serve as a template for propagating metamorphic testing across many more mixed signal designs for which creation of sufficiently detailed test oracles is impractical or impossible. Files used in this demonstration are available for download at [5].

REFERENCES

[1] T. Y. Chen, T. H. Tse and Zhiquan Zhou, "Fault-based testing in the absence of an oracle," *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*, Chicago, IL, USA, 2001, pp. 172-178.

[2] X. Xie, Z. Zhang, T. Y. Chen, Y. Liu, P.-L. Poon, and B. Xu, "METTLE: A METamorphic Testing Approach to Assessing and Validating Unsupervised Machine Learning Systems," *IEEE Transactions on Reliability*, vol. 69, no. 4, pp. 1293–1322, Dec. 2020.

[3] H. Wayne, "Metamorphic Testing," https://www.hillelwayne.com/post/metamorphic-testing/, March 2019.

[4] M. Hassan, D. Große and R. Drechsler, "System Level Verification of Phase-Locked Loop using Metamorphic Relations," *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, 2021, pp. 1378-1381.

[5] Cross, D., "Application of Metamorphic Testing to Mixed Signal Systems with Real Number Behavioral Models," https://support.cadence.com/apex/ArticleAttachmentPortal?id=a1OPP000001scZB2AY