Testbench Qualification for SystemC-AMS Timed Data Flow Models

Muhammad Hassan[1,2], Daniel Große[1,2], Hoang M. Le[2], Thilo Vörtler[3], Karsten Einwich[3], Rolf Drechsler[1,2]
[1]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[2]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[3]COSEDA Technologies GmbH, Dresden, Germany
muhammad.hassan@dfki.de {grosse,hle,drechsle}@informatik.uni-bremen.de
{thilo.voertler,karsten.einwich}@coseda-tech.com

***Abstract*—Analog-Mixed Signal (AMS) circuits have become increasingly important for today's SoCs. The Timed Data Flow (TDF) model of computation available in SystemC-AMS offers here a good tradeoff between accuracy and simulation-speed at the system-level. One of the main challenges in system-level verification is the quality of the testbench. In this paper, we present a testbench qualification approach for SystemC-AMS TDF models. Our contribution is twofold: First, we propose specific mutation models for the class of filters implemented as TDF models. This requires to analyze the Laplace transfer function of the filter design. Second, we present the mutation-based qualification approach based on the proposed specific mutations as well as standard behavioral mutations. This allows to find serious quality issues in the testbench. Our experimental results for a real-world AMS system demonstrate the applicability and efficacy of our approach.**

I. Introduction

Internet-of-Things (IoT) devices and *Cyber Physical Systems* (CPS) have escalated the demand for complex *System-on-Chips* (SoCs) comprising of analog, mixed-signal, and digital circuits tightly integrated on one die. As a consequence thereof methodologies are required to efficiently design, verify and produce high quality SoCs at lower costs. In particular, AMS verification face significant challenges due to the increasing design complexity. The digital parts are comparatively easier to verify as fault model based tests can be generalized, but unfortunately digital methodologies cannot be easily extended to AMS designs. The main reason for this are the differences between pure digital and mixed-signal designs: 1) continuous time signals for AMS systems highlight the need for parametric faults which affect the signal values even though the underlying design is unchanged, 2) there is a large number of possible test input signals (frequency, amplitude, sine wave, square wave, impulse etc), hence, careful application specific response checking is required, 3) frequency response, signal duration, amplitude variations, and post processing etc. are crucial in AMS verification, 4) AMS verification requires functional characterization, and 5) there are also no generic/-standardized fault models [1] such that each circuit topology requires a separate fault model. To summarize, these key differences highlight the compelling need for enhanced AMS verification methodologies.

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project CONVERS under contract no. 16ES0656, and University of Bremens graduate school SyDe, funded by the German Excellence Initiative.

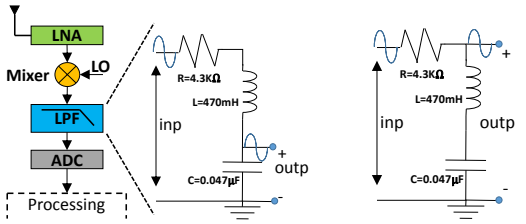
For an early design entry and early system verification, *Virtual Prototyping* is nowadays established industrial practice. The *Timed Data Flow* (TDF) *Model of Computation* (MoC) available in SystemC-AMS [2] offers a good tradeoff between accuracy and simulation-speed at the system-level. TDF is used to model the pure algorithmic or procedural description of the underlying design as well as different types of transfer functions.

The typical SystemC-AMS verification flow consists of either adapting the existing tests for the new circuit by manually modifying one of the tests from previous circuits with similar functionality, or by writing a new test from scratch. The methods used are based on the experience of the test engineer as well as the circuit specifications. This highlights one of the main challenges in system-level design verification: the quality of testbench. *Mutation testing* [3], [4], [5], [6] is an established approach for both software and digital hardware designs to determine the effectiveness of tests and testbenches, respectively. Essentially, the design is mutated based on a fault model and it is checked whether this mutation is detected by at least one test. Mutation testing heavily relies on the mutations which mimic potential faults. For pure digital designs this is well understood and strong approaches are available (see e.g. [7], [8], [9]).

In this paper, we present a testbench qualification approach for SystemC-AMS TDF models, which to the best of our knowledge has not been considered before. Our contribution is twofold: First, we propose specific mutation models for the class of filters (for both, active and passive filters) implemented as TDF models. This requires to analyze the Laplace transfer function of the filter design, and to perform the mutation accordingly. Second, we present the mutation-based qualification approach based on the proposed specific mutations as well as standard behavioral mutations. This allows to find serious quality issues in the testbench.

II. Background

For brevity, we refrain from giving a proper introduction to SystemC-AMS, and encourage the reader to go through the SystemC-AMS user guide [2]. Instead, we present here a simplified example SystemC-AMS program (Fig. 2) extracted from the RF receiver shown in Fig. 1a. The RF receiver includes a RF antenna, Low Noise Amplifier (LNA), a mixer with Local Oscillator (LO), a Low Pass Filter (LPF), and Analog-to-Digital Converter (ADC). The digitized signal is



(a) Second order LPF

(b) First order mutation of LPF - band stop filter

Fig. 1. RF receiver with filters – LNA: Low Noise Amplifier, LO: Local Oscillator, LPF: Low Pass Filter, ADC: Analog-to-Digital Converter

then sent out for further processing. The LPF block is expanded to show the RLC (Resistor, Inductor, Capacitor) circuit diagram. This will be used to showcase the main ideas of our approach throughout this paper.

The SystemC-AMS constructs and semantics necessary to understand the example will be explained as needed. We omitted the SystemC-AMS code required for instantiation and binding of components, i.e. the elaboration phase. The example (Fig. 1a) shows a single input (*inp*) and a single output (*outp*) analog second order passive LPF where L= 470 mH, C= 0.047 μF, and R= 4.3 KΩ, and cutoff frequency *fc* = 1 KHz. LPF is designed to allow signals with frequency lower than a certain cutoff frequency, and attenuate the signals with frequency higher than cutoff frequency. The filter is implemented in SystemC-AMS TDF MoC, where its Laplace transfer function (described by the numerator and denominator coefficients) is shown in Eq. 1.

$$H(s) = \frac{1}{LCs^2 + RCs + 1} \tag{1}$$

The coefficients are declared in Fig. 2, Line 3. SystemC-AMS provides dedicated solver for continuous time linear transfer functions in Laplace domain under the class *sca_tdf::sca_ltf_nd*, defined here as *ltf* (Line 4). The library function *initialize* (Line 7) sets the initial values of the member variables, i.e., *num* and *den* in this case (Line 8 - Line 13). This is also reflected in Eq. 2 with components values replaced. The callback method *processing* (Line 16) defines the time-domain behavior of the TDF module i.e., filter, in our case. The *num*, *den*, and *inp* are given to the solver (*ltf*) as inputs (Line 17), and *ltf* returns the continuous output and assigns it to *outp*.

$$H(s) = \frac{1}{22e^{-9}s^2 + 202.1e^{-6}s + 1} \tag{2}$$

III. Testbench Qualification Approach

In this section we first start with a motivating example for our approach. Then, we provide the ingredients like fault models, and qualification flow. At the end, the approach is illustrated to show its effectiveness.

A. Motivating Example

The general idea of the qualification approach is that if a mutation is introduced in the LPF design (Fig. 2), the testbench should be able to detect that mutation. But there exists an interesting case, when a single mutation changes the complete behavior of the filter, i.e., transforms the filter from low pass to

```
1   ....
2   struct my_lpf::states{
3   sca_util::sca_vector<double>
        num,den;
4   sca_tdf::sca_ltf_nd ltf;
5   };
6
7   void my_lpf::initialize() {
8   s.num(0) = 1; //s^0
9   s.num(1) = 0; //s^1
10  s.num(2) = 0; //s^2
11  s.den(0) = 1; //s^0
12  s.den(1) = 202.1e-6;
        //s^1
13  s.den(2) = 22e-9; //s^2
14  }
15
16  void my_lpf::processing() {
17  outp = s.ltf(s.num, s.den,
        inp);
18  }
19  ....
```

Fig. 2. SystemC-AMS second order low pass filter implementation

high pass filter (HPF) or band stop filter (BSF). In this case, the basic tests included in the initial testbench, specially if testing the filter around *fc*, may not able to detect the mutation, and this can lead to serious problems in the design. For e.g., a LPF when mutated to BSF (Fig. 1b) will be able to pass all signals below and above *fc*, and only attenuate the signal with frequency equal to *fc*.

Therefore, careful test selection around *fc* is required to detect the mutation.

B. Approach Overview

The approach is shown in Fig. 3. It relies strongly on the fault models chosen for mutation testing. If the fault models are weak, or do not represent an interesting case, it is highly likely that the testbench quality will be low. Sound testbench qualification can only be carried out when accurate fault models (derived as closely as possible from underlying defects) are available.

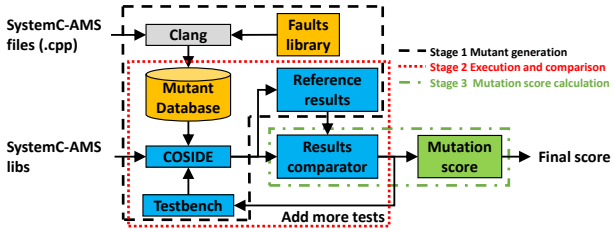


Fig. 3. Testbench qualification approach overview

C. Fault Modeling: Filters and More

We propose fault models for first and second order filters (both passive and active) implemented using SystemC-AMS TDF MoC as shown in Fig. 2. First order, and second order filters serve as primitive blocks for higher order filters, hence, fault modeling for these two types covers the vast spectrum of filters. One important thing to note is that the fault model is independent of the filter topology i.e., components connected in series or parallel.

Eq. 1 corresponding to LPF can be mutated in several ways, where each mutation changes the behavior of the filter completely. The mutations can be carried out in both the numerator and denominator. For e.g., a first order mutation in Eq. 1 is carried out by adding LCs^2 term in numerator. The result is shown in Eq. 3. This changes filter's behavior to band stop filter, which allows all frequencies to pass except around *fc*. The mutation is shown in Fig. 4, Line 4.

$$H(s) = \frac{LCs^2 + 1}{LCs^2 + RCs + 1} \tag{3}$$

Interestingly, the circuit topology stays unchanged i.e., the components, and their order of connection stays the same.

```
1    void my_lpf::initialize() {
2      s.num(0) = 1; //s^0
3      s.num(1) = 0; //s^1
4      s.num(2) = 22e-9; //s^2 <-- Mutation performed
5      s.den(0) = 1.0; //s^0
6      s.den(1) = 202.1e-6; //s^1
7      s.den(2) = 22e-9; //s^2
8    }
```

Fig. 4. First order mutation on Fig. 2

```
1    void my_lpf::initialize() {
2      s.num(0) = 0; //s^0 <-- First mutation
3      s.num(1) = 0; //s^1
4      s.num(2) = 22e-9; //s^2 <-- Second mutation
5      s.den(0) = 1.0; //s^0
6      s.den(1) = 202.1e-6; //s^1
7      s.den(2) = 22e-9; //s^2
8    }
```

Fig. 5. Second order mutation on Fig. 2

The corresponding circuit diagram is shown in Fig. 1b. This accentuates two problems, 1) visually the circuit looks the same as in Fig. 1a, the fault can be missed by test engineer, 2) if the testbench verifies the design around *fc*, the basic tests might not be able to detect the fault. Such mistakes are possible during copy-paste of the code and forgetting to update the equation coefficients. Hence, this makes a valid fault model. This also highlights the need for a high quality testbench.

Similarly, a second order mutation is performed on Eq. 1 by setting *num(0)* to 0 (first mutation, Fig. 5, Line 2), and setting *num(2)* to $22e{-}9$ (second mutation, Line 4). This is supported by the coupling effect hypothesis, where simple mutations couple to form a complex fault. Coupling effect can help in revealing very serious design issues.

A list of possible first and second order mutations on filter transfer functions is shown in Table I, when a certain filter type is implemented. For e.g., given a 2nd Order LPF, it can have four possible mutations shown under *first order mutations*, and *second order mutations* headings in Table I. The second order mutations are only applicable on the Laplace transfer functions in our case. For algorithmic descriptions, only first order mutations are used.

In addition to proposed filter fault models, SystemC-AMS TDF MoC models can be mutated using the operators and syntax from C++ library. Therefore, for sequential modeling faults, we adopt the comprehensive set of mutation operators as proposed in [10], where 77 C/C++ mutation operators are explained. They are primarily based on the competent programmer hypothesis, i.e. faults are syntactically small and only few keystrokes away from original program. In case of the constant mutation fault model (adding/subtracting a value), only small modifications of the original values are considered i.e., the mutation is done within +/-10% range. This is to keep the values within the defined tolerances of the components. The assumption here is that if this small variation is detected by the testbench, the larger variations will most likely be detected.

D. Qualification Flow

The qualification flow is divided into three stages as shown in Fig. 3. In the first stage, the mutants are generated and stored in a *Mutant Database*. Then, the original Design Under Verification (DUV) is simulated using the given testbench. The resulting output (all tests passing) is put aside as *Reference-results*. It helps in defining the operating ranges of the DUV, later to be compared with mutated design results. This stage needs to be executed only once in qualification flow.

In the second stage, mutants are picked one by one from the *Mutant Database*, compiled and executed using the same testbench used for the original DUV. The output is compared with *Reference-results*. If a test in testbench fails, that means the mutant has been detected, and it is termed as "killed". But if the mutant passes the tests, there could be several reasons, to name few; 1) not enough tests available, 2) equivalent mutant.

The third stage calculates the Mutation Score (MS) by using Eq. 4 on second stage results. MS of 1 is the highest, and signifies a sound testbench.

$$MS = \frac{\#mutants\_detected}{\#total\_mutants} \tag{4}$$

TABLE I
FIRST AND SECOND ORDER MUTATIONS OF FILTERS

Given Filter	1st Order Mutation	2nd Order Mutation
1st Order LPF	2nd Order LPF	1st Order HPF
1st Order HPF	2nd Order BPF	1st Order LPF
2nd Order LPF	2nd Order BSF	2nd Order HPF
	1st Order LPF	2nd Order BPF
2nd Order HPF	2nd Order BSF	2nd Order LPF
		2nd Order BPF
2nd Order BPF	1st Order HPF	2nd Order LPF
		2nd Order BPF
2nd Order BSF	2nd Order LPF	1st Order LPF
	2nd Order HPF	

LPF: Low Pass Filter — BPF: Band Pass Filter
HPF: High Pass Filter — BSF: Band Stop Filter

```
1    void lp_eln_tb_stim_tc1::stimulus_sequence() {
2      CHECK_RANGE_DYNAMIC(outp, -0.707, 0.707, sc_time(1.0,
           SC_MS),sc_time(10, SC_MS), sc_core::SC_ERROR); //TC1
3      CHECK_RANGE(s.s_fft_res, 0.7069, 0.7072, sc_core::SC_ERROR);
           //TC2
4      CHECK_RANGE_AFTER(outp, 0.105, 0.690, sc_time(50, SC_US),
           sc_core::SC_ERROR); //TC3
5      CHECK_RANGE_DYNAMIC(outp, sin_reference(outp_ref, 10),
           sin_reference(outp_ref, -10),sc_time(0.0, SC_SEC),
           sc_time(10, SC_MS), sc_core::SC_ERROR); //TC4
6    }
```

Fig. 6. SystemC-AMS testbench checkers

E. Illustration

To illustrate the qualification approach, consider the LPF shown in Fig. 2 with *fc*= 1 KHz. For brevity, we only show two interesting mutants with initial MS of 0.5. A sine wave with frequency *f*= 1 KHz (equal to *fc*), and amplitude 1.0 V is used as the input test signal to the LPF. The basic testbench shown in Fig. 6 containing three test cases (TC) is executed to get *Reference-results*. TC1 (Line 2) checks upper and lower bounds of *outp*, and also overshoots and undershoots during the specified time duration. TC2 (Line 3) checks the amplitude of *outp* exactly at *fc* by taking Fast-Fourier Transform (FFT). TC3 (Line 4) checks if *outp* has crossed a threshold (0.105 V) after 50 μS.

Case one, a first order mutant shown in Fig. 4 is used, and the tests are executed. The mutant behaves like a band stop filter, and attenuates the signal completely at *fc*, which is $1e3$ for the LPF. TC2 kills the mutant as the amplitude of *outp* goes below 0.7 V.

Case two, a second order mutant shown in Fig. 5 is used, and surprisingly all the test cases pass. The reason being, the

mutant is behaving like a high pass filter, and at *fc* it has the same magnitude response as the LPF (DUV) i.e., *outp* = 0.707 V. Hence, a new test case TC4 is added (Fig. 6, Line 5) which checks the phase deviation of *outp* in range $\pm 10°$ wrt. the original signal (*outp_ref*). Now, the mutant is killed because of phase difference, and the MS is 1.

IV. Implementation Details

The framework's first stage is mutant generation, done by an in-house standalone command line tool from the input SystemC-AMS source files. The underlying infrastructure is the LibTooling library of Clang, and the tool compiles new mutants by traversing different entities of the Abstract Syntax Tree (AST). The tool supports all mutation operators described in Section III-C. Second stage integrates the COSIDE [11] environment for execution and simulation, and the last stage performs post-processing on results.

V. Experimental Results

In this section we present a real-world AMS system; an energy efficient buck-boost converter [12], as a case study. A buck-boost converter can operate in a step-down converter (buck) or step-up converter (boost) mode.

An overview of the circuit is shown in Fig. 7. The input voltage at node *v1* of the battery is transformed to the voltage *v2* at the output and drives a load. The main principle of the conversion is to store energy *lx* in the inductor (*i_l1*) and to control the current flowing through it. Using switches within the buck-boost converter, the battery (*i_battery1*) is connected from *v1* to *lx* to charge *i_l1*, or from *lx* to *v2* to discharge it. Depending on the switching ratio the voltage is regulated. The main challenge when implementing a buck-boost converter is thereby the control algorithm which controls the switching frequency, depending on the current flowing through the nodes *v1* and *v2*. A digital controller sets the mode of the buck-boost converter and the expected output voltage. In addition, the maximum current can be set, which flows through the converter. The converter also uses low pass filters, to filter out the high frequencies. The buck-boost converter and its controller are implemented as SystemC-AMS TDF models.

To test the buck-boost converter model it is checked how fast the expected output voltage is reached and how stable it is. Therefore, an input voltage *v1* is applied and a target voltage *v2* is programmed via the controller. In Fig. 8 a test case is shown. Using a checker framework, part of the COSIDE® SystemC-AMS tool environment [11], an arbitrary function $f(t)$ can be specified as expected value. This enables the creation of analog regression runs with automatic checks. In the example, it is checked that after 2.5 ms the output voltage is between 1.8V and 2.2V. It can be seen that the voltage is not stable enough and the test would fail. The SoC has 10 test cases in the testbench initially.

The in-house tool (Section IV) generated 46 mutants of the converter, out of which 4 were filter mutants. 18 mutants were killed successfully by the testbench, whereas 28 were undetected. MS using Eq. 4 is 0.39. This low MS highlights the poor quality of the testbench, and emphasizes the need to write more test cases to kill the remaining 28 mutants.

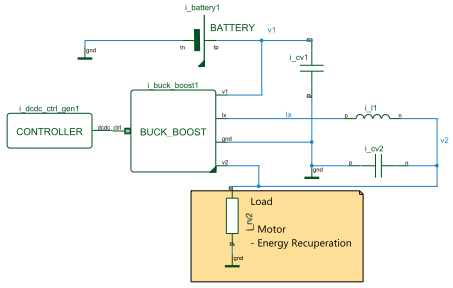


Fig. 7. Buck-boost converter system overview

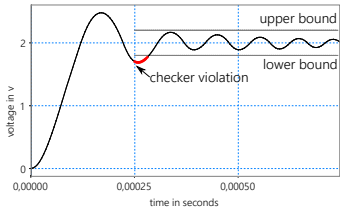


Fig. 8. Buck-boost test case with expected output voltage of 2V and allowed range

VI. Conclusion

In this paper we have presented the first testbench qualification approach for SystemC-AMS TDF models. The approach is based on mutation analysis. Due to the specifics of SystemC-AMS TDF models, we have devised new mutation models which analyze the Laplace transfer function of filters. Combining these new mutation models with standard behavioral mutations allows us to check the effectiveness of a SystemC-AMS testbench. We showed the effectiveness of our approach on a real-world SystemC-AMS system. In future, we plan to investigate constrained random techniques e.g., [13], for testcase improvement.

References

[1] M. Soma, "Challenges in analog and mixed-signal fault models," *IEEE Circuits and Devices Magazine*, vol. 12, no. 1, pp. 16–19, 1996.

[2] M. Barnasconi, C. Grimm, M. Damm, K. Einwich, M. Louërat, T. Maehne, F. Pecheux, and A. Vachoux, "Systemc ams extensions users guide," *Accellera Systems Initiative*, 2010.

[3] R. G. Hamlet, "Testing programs with the aid of a compiler," *IEEE Trans. Softw. Eng.*, no. 4, pp. 279–290, 1977.

[4] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," in *Mutation testing for the new century*. Springer, 2001, pp. 34–44.

[5] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *TSE*, vol. 32, no. 9, pp. 733–752, Sep. 2006.

[6] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *TSE*, vol. 37, no. 5, pp. 649–678, 2011.

[7] N. Bombieri, F. Fummi, G. Pravadelli, M. Hampton, and F. Letombe, "Functional qualification of TLM verification," in *DATE*, 2009, pp. 190–195.

[8] D. Große, H. M. Le, M. Hassan, and R. Drechsler, "Guided lightweight software test qualification for IP integration using virtual prototypes," in *ICCD*, 2016, pp. 606–613.

[9] A. Sen, "Concurrency-oriented verification and coverage of system-level designs," *TODAES*, vol. 16, no. 4, p. 37, 2011.

[10] H. Agrawal, R. A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. Spafford, "Design of mutant operators for the C programming language," Purdue University, Tech. Rep., 1989.

[11] "Coside®," http://www.coseda-tech.com.

[12] E. Lefeuvre, D. Audigier, C. Richard, and D. Guyomar, "Buck-boost converter for sensorless power optimization of piezoelectric energy harvester," *IEEE Transactions on Power Electronics*, vol. 22, no. 5, pp. 2018–2025, 2007.

[13] F. Haedicke, H. M. Le, D. Große, and R. Drechsler, "CRAVE: An advanced constrained random verification environment for SystemC," in *SoC*, 2012, pp. 1–7.