Data Flow Testing for SystemC-AMS Timed Data Flow Models

Muhammad Hassan[1,2] Daniel Große[1,2] Hoang M. Le[2] Rolf Drechsler[1,2]
[1]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[2]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
muhammad.hassan@dfki.de {grosse,hle,drechsle}@informatik.uni-bremen.de

***Abstract*—Internet-of-Things (IoT) devices have significantly increased the need for high quality Analog Mixed Signal (AMS) System-on-Chips (SoC). Virtual Prototyping (VP) can be utilized for an early design verification. The Timed Data Flow (TDF) model of computation available in SystemC-AMS offers here a good trade-off between accuracy and simulation-speed at the system-level. One of the main challenges in system-level verification of AMS design is to achieve full path coverage. In the software domain Data Flow Testing (DFT) has demonstrated to be a powerful testing strategy in this regard. In this paper we introduce a DFT approach for SystemC-AMS TDF models based on two major contributions: First, we develop a set of SystemC-AMS TDF models specific coverage criteria for DFT. This requires to consider the SystemC-AMS semantics of signal flow. Second, we explain how to automatically compute the data flow coverage result for given TDF models using a combination of static and dynamic analysis techniques. Our experimental results on real-world AMS VPs demonstrate the applicability and efficacy of our approach.**

I. Introduction

The *Internet-of-Things* (IoT) begins where the physical world meets the digital world. Sensors, converters, microprocessors, and transceivers, are what gather and transport the data that fuel the promise and potential of the IoT. This has introduced a new constraint in the *System-on-Chip* (SoC) design cycle: a multi-functional SoC, i.e., analog, mixed-signal, and digital circuits tightly integrated on one chip. In the past analog and digital parts have been verified by separate teams. However, this digital-analog divide fails at the SoC level. Malfunction of interfaces between the analog and digital parts is a common problem in AMS design, which becomes more complex due to the tight integration of digital and analog today. Hence, it is mandatory to develop methodologies which allow to consider analog and digital designs holistically. Another major challenge in AMS verification is the simulation speed of the SPICE (Simulation Program with Integrated Circuit Emphasis) models for the analog part of the SoC. Their simulation, while slow, is still considered a golden standard and cannot be ignored. But different levels of design abstractions can be used to achieve significantly better simulation performance, and earlier design verification.

That is why *Virtual Prototyping* (VP) at the abstraction of *Electronic System Level* (ESL) is nowadays an established industrial practice. The *Timed Data Flow* (TDF) *Model of Computation* (MoC) available in SystemC-AMS [1] offers a good trade-off between accuracy and simulation-speed at the SoC level. TDF defines time domain processing, and is used to model the pure algorithmic or procedural description of the underlying design. Multiple TDF models connect together to make a TDF cluster, i.e., a SoC. Because of earlier availability, the TDF model is used as a golden reference for system verification in a top-down design flow. Hence, its functional correctness is of utmost importance. Therefore, the complete AMS design as a whole is subject to rigorous verification.

Simulation is still the most widely used technique for functional correctness of VPs. Similarly, for SystemC-AMS TDF models, a verification environment is constructed by composing a testbench. To examine as much distinct functional behavior of the *Design Under Verification (DUV)* as possible, constrained random stimuli are applied [2], [3]. For each stimulus, the behavior of the TDF model(s) is compared against expected behavior (as given in specification sheet). Since a TDF model is essentially a software-based model, techniques from software testing domain can be leveraged for SystemC-AMS TDF models to improve the verification quality. One promising technique in the software domain is *Data Flow Testing* (DFT) [4], [5], [6] which experienced renewed interest due to its applicability in fault localization, security testing, and specification consistency checking [7].

DFT monitors the interactions of a piece of data (a variable's definition, and its uses). Essentially, the idea is that a bug is revealed only if the bug is exercised by a test input signal. The motivation is that the testsuite should be able to exercise program paths along which data flows, i.e., traversal of paths from a variable definition, to either some or all of its uses. They are generally termed as definition-use pairs or def-use pairs. Since, DFT brings the promise of a high quality testsuite in software domain, its importance towards AMS designs cannot be ignored.

Therefore, in this paper, we propose the first DFT approach for SystemC-AMS TDF models based on two major contributions: First, we develop a set of SystemC-AMS TDF models specific coverage criteria for DFT. This requires to consider the SystemC-AMS semantics of TDF signals, TDF cluster modeling, TDF ports, and dynamic TDF. Second, we explain how to automatically compute the data flow coverage result for given TDF models using a combination of static and dynamic analysis techniques. The coverage results guide the verification engineer to add new testcases to improve the coverage.

II. Related Work

AMS SoC verification has been an active research topic, both formal, and simulation based. Two symbolic model checking algorithms for the verification of AMS designs are proposed in [8]. The first model checker utilizes binary decision diagrams while the second is a bounded model checker that uses a satisfiability modulo theory solver. The analysis is done using VHDL-AMS description. [9], [10] also propose approaches to formally verify the AMS designs.

Other works, like [11] focus on the abstraction of analog components implemented in Verilog-AMS and SystemC-AMS Electrical Linear Networks (ELN) to a C++ based models.

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.

This helps in faster simulations, and quicker system validation. But it is only limited to the circuits, and not the behavioral models. [12], [13] focus on the fault testing through abstraction of Verilog-AMS models to a C++ based model, where the faults are injected in to the Verilog-AMS model, abstracted to C++ model and finally simulated to check if the fault has been detected by the testsuite or not. It targets the analog design, and does not consider the complete AMS SoC.

[14] focuses on SystemC VPs essentially, and proposes different coverage criteria w.r.t. SystemC semantics i.e., non-preemptive thread scheduling, shared memory communication and event-based synchronization. But it is only limited to SystemC. In [15], the authors propose mutation analysis for SystemC-AMS TDF models for testbench qualification.

In [16], the authors propose data flow relations w.r.t., circuit components, for e.g., transistor configurations etc. It motivated us to look for similar flows at the system-level. As mentioned earlier, our approach is to the best of our knowledge the first attempt to leverage DFT for SystemC-AMS TDF models.

III. Background

A. Motivating Example

For brevity, we refrain from giving a proper introduction to SystemC-AMS, and encourage the reader to go through SystemC-AMS user guide [1]. Instead, we present here a simplified example SystemC-AMS program (Fig. 2) extracted from an IoT device, *sensor_system* (Fig. 1). This example will be used to showcase the main ideas of our approach throughout this paper. The *sensor_system* is a typical AMS system which includes a mixture of analog, and digital components; a temperature sensor (TS), a humidity sensor (HS), an analog signal delay (Z^{-1}), a 4×1 analog mux (AMUX), a gain element (G), a 9-bit analog-to-digital converter (ADC), a small digital control module, and two LEDs (Light-Emitting Diode). The *sensor_system* works as follows: TS/HS senses the temperature/humidity (time continuous analog signals), and if the sensed value crosses a certain threshold, the sensor generates an interrupt (digital signal) to the control module. The control module sets the AMUX select line corresponding to the sensor (Line 66), and reads the temperature/humidity value. It compares the value with a preset threshold (60°C for temperature, 45RH for humidity), and switches the corresponding LED (T_LED (Line 49)/H_LED (Line 62)) on, indicating too hot or too humid. Please note, if the temperature crosses 50°C, the controller halts the sensor output (Line 55), and reads the delayed value to ensure it read the value correctly in the first place. The whole system is implemented in SystemC-AMS TDF MoC, where the necessary parts of the design are shown in Fig. 2. We omitted the SystemC-AMS code required for instantiation and initialization, i.e. the elaboration phase. Please note, the variables with prefix *"ip_"/"op_"*, and *"m_"* refer to input/output ports, and member variables of the TDF model, respectively.

The TS is implemented in function *TS::processing()* (Line 1 - Line 16). It gives an output signal only when the input is between a certain threshold (between 30 mV and 1500 mV (Line 9)). HS behavior is implemented in *HS::processing()* (Line 18 - Line 30). Interrupts are additionally added to both the sensors. AMUX is implemented in *AM::processing()* (Line 32 - Line 39), and control in *ctrl::processing()* (Line 41 - Line 68). The control model translates incoming digital signal from ADC to a temperature reading by dividing the input (ip_DIN) by 10 (the scale factor). For e.g., a signal of 200 mV translates to 20°C. Gain (G) and analog delay (Z^{-1}) (SystemC-AMS library) are instantiated as shown in Fig. 2 at Line 76, and Line 73, respectively. ADC is instantiated at Line 79. In order to simplify the showcase of example, the ADC is assumed to output the same signal in digital form, it gets on its analog input. A required part of netlist (binding information) of the TDF cluster is shown in *sense_top::architecture()* (Line 70 - Line 82). This is important as will be shown in following sections.

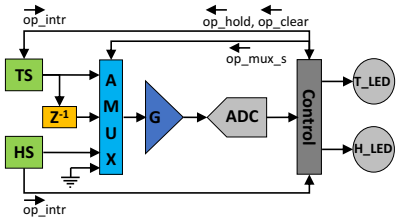

Fig. 1. Sensor system - T: Temperature, H: Humidity, XS: X Sensor (X=T,H), Z^{-1} = delay, AMUX: Analog mux, ADC: Analog-to-Digital Converter, X_LED: Light-emitting Diode, G: Gain

B. Data Flow Association and Testing

Data flow association, a generalization of definition-use (def-use) association, in general, is a tuple *(v, d, u)* where *d* is a program statement in which the variable *v* is defined, and *u* is a program statement in which *v* is used. Consider the variable *tmpr* defined in Fig. 2 - Line 4, and used in Line 9. It is considered a def-use association as the variable *tmpr* is not redefined between Line 4 and Line 9. A def-use association *(v, d, u)* can only be exercised by a testcase *t*, iff execution of *t* goes through definition *d* and then use *u* without re-definition of variable *v* in-between.

The data flow testing, on the other hand, requires that the the testsuite should be comprehensive enough to cause the traversal of maximum paths from a variable definition to its uses. Meaning thereby, maximum number (all ideally) of data flow associations should be exercised. Testsuite refinement is done in the process i.e., addition of more test cases, until a preset coverage criteria is satisfied. This requires to detect data flow associations and measure the data flow coverage of the testsuite. In the following text, data flow associations and testing w.r.t. SystemC-AMS TDF models will be shown while considering the TDF semantics. Please, also note that the term data flow association will be used (alternatively) in place of def-use association in the following text.

IV. Data Flow Testing for SystemC-AMS TDF Models

In this section we start with the overview of our approach. Afterwards, we provide the ingredients like classification of data flow associations w.r.t. SystemC-AMS TDF models, and coverage criteria. At the end, the approach is illustrated to show its effectiveness.

A. Approach Overview

Our data flow testing approach for SystemC-AMS TDF models is shown in Fig. 3. It comprises of three stages; 1) static analysis, 2) dynamic analysis, 3) coverage analysis. All three

```
1  void TS::processing()
2  {
3    double sig_in = ip_signal_in; // volts
4    double tmpr = sig_in*1000; //millivolts
5    double out_tmpr = 0;
6    bool intr_ = false;
7    if (!ip_hold){
8     if (ip_clear) intr_ = 0;
9     else if ((tmpr > 30) && (tmpr < 1500 )){
10     out_tmpr = tmpr;
11     intr_ = true;
12    }
13    op_intr.write(intr_);
14    op_signal_out = out_tmpr;
15   }
16 }
17
18 void HS::processing()
19 {
20   double temp = ip_signal_in*1000; // mV
21   double Tdepend = (B1*42 + B2)*temp + (B3*42+B4);
22   double C = 153e-12; // capacitance
23   double BC = 150e-12; // bulk capacitance at 30%RH
24   double sensitivity = 0.25e-12;
25   bool intr_ = false;
26   double newRH = 30 + ((C - BC)/sensitivity) + Tdepend;
27   if (newRH > 30) intr_ = true;
28   op_intr.write( intr_);
29   op_signal_out = newRH;
30 }
31
32 void AM::processing()
33 {
34   double tmp_out = 0;
35   if (ip_select == 0) tmp_out = ip_port_0;
36   else if (ip_select == 1) tmp_out = ip_port_1;
37   else if (ip_select == 2) tmp_out = ip_port_2;
38   op_mux_out = tmp_out;
39 }
40
41 void ctrl::processing()
```

```
42 {
43   if(ip_intr0)
44    if((ip_DIN/10) < 60) {
45     op_clear = 1;
46     m_mux_s = 0;
47     op_hold = 0;
48    } else if (m_mux_s == 1 && (ip_DIN/10)>60){
49     op_T_LED = 1;
50     op_clear = 1;
51     op_hold = 0;
52     m_mux_s = 0;
53    } else if (m_mux_s == 0 && (ip_DIN/10)>50){
54     m_mux_s = 1;
55     op_hold = 1;
56    } else {
57     op_hold = 0;
58     op_clear = 1;
59     m_mux_s = 0;
60   }
61   else if (ip_intr1 && m_mux_s == 2){
62    if(ip_DIN > 45) op_H_LED = 1;
63    m_mux_s = 0;
64   } else if (ip_intr1)
65    m_mux_s = 2;
66   op_mux_s = m_mux_s;
67   if(ip_intr0==0) op_clear = 0;
68 }
69
70 void sense_top::architecture() // netlist
71 {
72   // .....
73   i_delay_tdf1 -> tdf_i.bind(op_signal_out);
74   i_delay_tdf1 -> tdf_o.bind(op_delay_out);
75
76   i_gain_tdf1 -> tdf_i.bind(op_mux_out);
77   i_gain_tdf1 -> tdf_o.bind(op_gain_out);
78
79   i_adc1 -> adc_i.bind(op_gain_out);
80   i_adc1 -> adc_o.bind(op_adc_out);
81   //...
82 }
```

Fig. 2. Sensor system - SystemC-AMS TDF model example (B1 = 0.0014/°C, B2 = 0.1325% RH/°C, B3 = -0.0317, B4 = -3.0876% RH)[17]

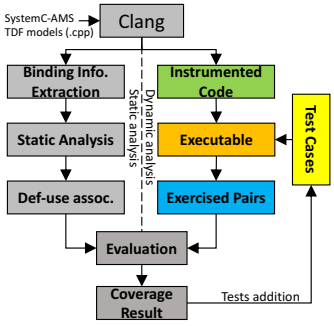

Fig. 3. Proposed Data Flow Testing Methodology Overview

stages work together to fully automatically compute SystemC-AMS TDF models specific data flow coverage results.

The static analysis identifies the set of all data flow associations computed in the TDF clusters using TDF models binding information ([1]). This step is executed only once at the beginning of the analysis. Due to static nature, the analysis computes an over-approximation of the associations which may also contain infeasible associations, i.e., dead code associations. On circuit level, dead code associations can be mapped to component isolation because of open circuit, wrong transistor configuration, or a very high/low passive component value etc. To guide testcase selection (with test input signal), associations are classified into different disjoint groups based on the likeliness of being infeasible.

The dynamic analysis (Fig. 3 - right side) identifies the exercised data flow associations w.r.t. the testsuite. The SystemC-AMS source file with TDF models is instrumented so that relevant runtime information can be captured. The instrumented file is afterwards compiled and executed against every testcase from testsuite. For each testcase, different data (signal) flow information is recorded. The resulting logs are analyzed and combined to obtain the set of exercised data flow associations.

At the end, the static and dynamic analysis results are evaluated and combined to obtain a coverage result. Essentially, the result shows which data flow associations have been exercised by at least one testcase and which have been completely missed. An association can be missed due to, 1) The testsuite is insufficient to cover it. In this case a new testcase (test input signal with different parameters) needs to be added. 2) The association is infeasible. In this case it can be either inspected for correct binding, or ignored.

Our classification system, that ranks associations according to their likeliness of being infeasible, allows the testing engineer to focus his efforts on promising testcases (test input signals) to efficiently improve the coverage result. In this work, automated test generation has not been considered. In the following we describe our classification system and the coverage result in more detail and demonstrate both using the running example (Fig. 2).

B. Classification of Data Flow Associations

We define four classifications specific to TDF models of SystemC-AMS: Strong, Firm, PFirm, PWeak. They are defined in two categories, 1) within a TDF model, 2) in TDF cluster. The first category extends the classical notion of data flow testing that reason about variable definition and use within one function. First two classifications come under this category. In SystemC-AMS, a variable defined in one TDF model might flow to another TDF model for use. This happens when the defined variable is an output port, and used variable is an input port. In this case, the classical notion does not apply and new approach has to be devised. In this scenario, it is also possible that the defined variable (port) is redefined (signal amplification, signal delay etc.) outside the TDF model, before being used. Hence, the two classifications; PFirm, PWeak, are used to classify the definition and use w.r.t. the SystemC-AMS TDF model ports (hence P in PFirm and PWeak).

1) Def-Use Association: We define a def-use association as an ordered tuple *(v,d,dm,u,um)*. For a variable *v*, there exists a static path between definition *d* present in TDF model *dm* (*defining model*), and use *u* present in TDF model *um* (*using model*) without a redefinition of *v* in between. We define a *du-path* as a static path between *d* and *u* without a re-definition of *v*, when both *d* and *u* can be in same or different TDF models. Four classifications for def-use associations are proposed:

1) *Strong* - a) Variable *v* is an output port of a TDF model *dm*, and there exists a *du-path* between *dm* and *um*. b) Variable *v* is local to the TDF model, i.e., *dm* == *um*, and every static path between *def* and *use* is a *du-path*.
2) *Firm* - Variable *v* is local to the TDF model, and at least one static path between *d* and *u* is not a *du-path*.
3) *PFirm* - Variable *v* is an output port, and there exists at least one static path which is not a *du-path*.
4) *PWeak* - Variable *v* is an output port, and there exists no *du-path*.

Because local variable does not flow out of the TDF model, it is classified in first two associations; strong or firm. When ports of TDF models are considered, the data (signal) flowing towards the connecting TDF model might be a *du-path* (direct connection) to the *um*. It is also possible that the data (signal) is redefined. The re-definition could occur in the following two cases (limited to SystemC-AMS library, and only to single input single output (SISO) components): a) There exists a *delay* element in the *du-path*. The delay element delays the incoming data (signal) by the preset number of samples (seconds), and outputs an earlier value instead of the current value. Because of this reason, we consider it as a redefinition. b) There exists a *gain or buffer* element which amplifies the incoming signal, or regenerates it. More elements can also be added. Based on this general idea, three data flow associations are defined: a) If the output port directly connects to another TDF model, the association is *Strong*, b) if the output port connects to another TDF model, and there exists a re-definition of the port as well, i.e., there are two branches (one original, and one redefined) and both of them connect to the same TDF model, the association is called *PFirm*. Because we are not doing context aware analysis, we do not know which of the two definitions will be used inside the TDF model. Maybe both the definitions are used, or maybe one of them is used. It is context dependent (*if-else, while, for..*), and the condition cannot be evaluated on static time. For e.g., analog mux behavior, c) if both the branches are redefined, and connect to same TDF model, it is termed as *PWeak*. Because the original port is re-defined no matter which path is taken, or d) if the branches (original, and redefined) go to different TDF models, then they are classified according to the individual cases defined above (*Strong or PWeak*).

2) Coverage Result and Test Adequacy Criteria: Every classification defines a disjoint set of data flow associations. Please, note that the strengths of associations are based entirely on static characteristics of the underlying SoC. Hence, a coverage criterion of each classification is required, as follows:

1) *all-Strong* - the criteria is satisfied iff all data flow associations termed *Strong* have been covered.
2) *all- Firm* - all data flow associations termed *Firm* are covered.
3) *all- PFirm* - all data flow associations termed *PFirm* are covered.
4) *all- PWeak* - all data flow associations termed *PWeak* are covered.
5) *all- defs* - iff at least one def-use association *(v, d, dm, u, um)* is covered for every definition.
6) *all-dataflow* - iff all above criterion are satisfied.

Satisfying *all-dataflow* criteria is difficult due to limitations of static analysis, i.e., imprecisions or over-approximation. But because the associations are disjoint, satisfaction of each criteria is independent. Hence, it is possible that some of the (sub-)criteria can be fully satisfied - or at least up to a high degree, i.e. 90% of the associations have been exercised. In particular, we expect that *all-Strong*, *all-Firm*, and *all-PFirm* to be the primarily focused criteria. The *Strong*, *Firm*, and PFirm associations contain at least one *du-path*, hence, it is expected from the test input signal to cover them.

3) Illustration: To illustrate our DFT approach for SystemC-AMS TDF models, we use the example IoT device given in Fig. 1, and its code in Fig. 2. We show with the help of three different testcases applied one at a time how the data flow coverage increases. The final coverage results are shown in Table I. Table I can be interpreted in the following way: the column titled *Static Pairs* lists all the statically identified data flow associations, the columns titled *Testsuite* presents three testcases (TC); *TC1, TC2, TC3*. The table lists the data flow associations from *Strong* (top left), to *PWeak* (bottom right). If a TCX (X = 1,2,3) is able to exercise a data flow association, an "x" is marked in the corresponding TC column. The data flow associations which are not exercised are marked with a "-" in the corresponding TC column. The test input signals or TC used are: TC1) a constant time continuous signal of 0.1V, mimicking a temperature of $10°C$, TC2) a time continuous signal from 0V to 0.65V, i.e. $0°C$ to $65°C$, and back to 0V ($0°C$), TC3) a time continuous signal at 0.40V ($45°C$). TC1 and TC2 are applied to TS, while TC3 is applied to HS.

To give an idea, the def-use association *(tmpr, 4, TS, 9, TS)* is Strong as there exists only one path from Line 4 to Line 9, without any redefinition of *tmpr* (a local variable). The def-use pair is exercised by TC1, and TC2. The def-use pair *(op_intr, 13, TS, 43, ctrl)* is also a Strong association. Because, it is not redefined inside the TDF model *TS*, and being an output port, it is not redefined while flowing to the next TDF model *ctrl*. The def-use pair *(out_tmpr, 5, TS, 14, TS)* is Firm, because it is local to the TDF model *TS*, and there exists multiple paths from Line 5 to Line 14. The def-use associations *(op_signal_out, 14, TS, 35, AM)* is exercised by both TC1 and TC2, and *(op_signal_out, 74, sense_top, 36, AM)* is exercised by TC2 only. There are two paths originating from *op_signal_out* port. One path connects to the *AM* as original signal (defined at Line 14), and the second path is redefined through a delay element at Line 74. Because both the paths end up in *AM*, any of them can be used based on the mux *select* line. Since, this information cannot be deduced statically, it is termed PFirm. The def-use pair *(op_mux_out, 77, sense_top, 79, sense_top)* is termed PWeak, because the signal at port *op_mux_out* always goes through the gain element at Line 87, hence, it is redefined before it goes to ADC. It is exercised by all three testcases.

When TC2 was applied, it was expected that "T_LED" (Line 49) will be switched on (once temperature goes above

TABLE I
SYSTEMC-AMS TDF MODELS SPECIFIC DATA FLOW ASSOCIATIONS - REFERENCE FIG. 2

Static Pairs	Testsuite			Static Pairs	Testsuite			Static Pairs	Testsuite		
	TC1	TC2	TC3		TC1	TC2	TC3		TC1	TC2	TC3
Strong				(op_intr, 13, TS, 67, ctrl)	x	x	-	(C, 22, HS, 26, HS)	-	-	x
(m_mux_s, 65, ctrl, 66, ctrl)	-	-	x	(op_hold, 55, ctrl, 7, TS)	-	x	-	(out_tmpr, 10, TS, 14, TS)	x	x	-
(m_mux_s, 65, ctrl, 48, ctrl)	-	-	-	(op_hold, 57, ctrl, 7, TS)	-	x	-	(ip_signal_in, 1, TS, 3, TS)	-	-	-
(m_mux_s, 65, ctrl, 53, ctrl)	-	-	-	(op_clear, 45, ctrl, 8, TS)	x	x	-	(BC, 23, HS, 26, HS)	-	-	x
(m_mux_s, 65, ctrl, 61, ctrl)	-	-	x	(op_clear, 50, ctrl, 8, TS)	-	-	-	(intr_, 27, HS, 28, HS)	-	-	x
(m_mux_s, 54, ctrl, 66, ctrl)	-	x	-	(op_clear, 58, ctrl, 8, TS)	-	x	-	(temp, 20, HS, 21, HS)	-	-	x
(m_mux_s, 54, ctrl, 48, ctrl)	-	x	-	(op_clear, 67, ctrl, 8, TS)	x	x	-	(newRH, 26, HS, 27, HS)	-	-	x
(m_mux_s, 54, ctrl, 53, ctrl)	-	x	-	(op_hold, 47, ctrl, 7, TS)	x	x	-	(newRH, 26, HS, 29, HS)	-	-	x
(m_mux_s, 54, ctrl, 61, ctrl)	-	-	-	(op_hold, 51, ctrl, 7, TS)	-	-	-	(Tdepend, 21, HS, 26, HS)	-	-	x
(m_mux_s, 59, ctrl, 66, ctrl)	-	x	-	**(op_intr, 13, TS, 43, ctrl)**	x	x	-	(op_signal_out, 29, HS, 37, AM)	-	-	x
(m_mux_s, 59, ctrl, 48, ctrl)	-	-	-	(adc_out, 47, adc, 44, ctrl)	x	x	-	(sensitivity, 24, HS, 26, HS)	-	-	x
(m_mux_s, 59, ctrl, 53, ctrl)	-	-	-	**(adc_out, 47, adc, 48, ctrl)**	-	x	-	(ip_signal_in, 18, HS, 20, HS)	-	-	x
(m_mux_s, 59, ctrl, 61, ctrl)	-	-	x	(tmpr, 4, TS, 9, TS)	x	x	-	(adc_out, 47, adc, 53, ctrl)	-	x	-
(m_mux_s, 63, ctrl, 66, ctrl)	-	-	x	(op_mux_s, 66, ctrl, 35, AM)	x	x	x	(adc_out, 47, adc, 62, ctrl)	-	-	x
(m_mux_s, 63, ctrl, 48, ctrl)	-	-	-	(op_mux_s, 66, ctrl, 36, AM)	-	x	x	**Firm**			
(m_mux_s, 63, ctrl, 53, ctrl)	-	-	-	(op_mux_s, 66, ctrl, 37, AM)	-	-	x	(intr_, 6, TS, 13, TS)	-	x	-
(m_mux_s, 63, ctrl, 61, ctrl)	-	-	-	(sig_in, 3, TS, 4, TS)	x	x	-	(tmp_out, 34, AM, 38, AM)	-	-	-
(m_mux_s, 46, ctrl, 66, ctrl)	x	x	-	(tmpr, 4, TS, 10, TS)	x	x	-	**(out_tmpr, 5, TS, 14, TS)**	x	x	-
(m_mux_s, 46, ctrl, 48, ctrl)	-	x	-	(intr_, 8, TS, 13, TS)	x	x	-	(intr_, 25, HS, 28, HS)	-	-	x
(m_mux_s, 46, ctrl, 53, ctrl)	-	x	-	(op_intr, 28, HS, 64, ctrl)	-	-	x	**PFirm**			
(m_mux_s, 46, ctrl, 61, ctrl)	x	x	-	(op_intr, 28, HS, 61, ctrl)	-	-	x	**(op_signal_out, 74, sense_top, 36, AM)**	-	x	-
(m_mux_s, 52, ctrl, 66, ctrl)	-	-	-	(intr_, 11, TS, 13, TS)	x	x	-	**(op_signal_out, 14, TS, 35, AM)**	x	x	-
(m_mux_s, 52, ctrl, 48, ctrl)	-	-	-	(tmp_out, 35, AM, 38, AM)	x	x	x	**PWeak**			
(m_mux_s, 52, ctrl, 53, ctrl)	-	-	-	(tmp_out, 36, AM, 38, AM)	-	x	-	**(op_mux_out, 77, sense_top, 79, sense_top)**	x	x	x
(m_mux_s, 52, ctrl, 61, ctrl)	-	-	-	(tmp_out, 37, AM, 38, AM)	-	-	x				

TC: Testcase (test input signal) (x) = data flow pair exercised (-) = data flow pair not exercised

60°C). But it did not switch on, and the data flow associations related to lines between Line 49 and Line 52 were never exercised. Upon careful inspection, an interface problem was found between ADC and control. The output of ADC was saturating because of 9-bit resolution. Any signal above 512mV was saturated to 512mV at ADC output.

TC1, and TC2 alone were not sufficient to achieve a reasonable data flow coverage. They were not able to exercise many associations specific to HS. Hence, TC3 is used additionally. There is still room for coverage improvement. This example demonstrates that standard def-use coverage criterion alone are not sufficient for extensive testing of SystemC-AMS TDF designs. Our proposed SystemC-AMS specific Strong, Firm, PFirm, and PWeak coverage criterion are important for a high quality testsuite.

V. IMPLEMENTATION DETAILS

In this section we describe the implementation of our DFT framework. The framework is implemented using the LibTooling library for Clang compiler [18]. Clang generates an Abstract Syntax Tree (AST) of the SystemC-AMS TDF model's source code. The AST is parsed to extract the required information to perform the data flow analysis. We next discuss important implementation details.

The connectivity (binding) of TDF models is possible in different ways depending on the implementation of the SystemC-AMS design, for e.g., using *bind* keyword. It is important to extract it knowing the correct semantics. By default, the implementation of SystemC-AMS TDF models reside in *module::processing()* function, but it could also be in a user defined function. This is registered in the elaboration phase using *register_processing()* library function. This information is required for proper analysis, as the framework needs to know where to look for the TDF model. Afterwards, the AST of each TDF model is parsed again while the signal definition and use information to perform static analysis is extracted. Static analysis is performed in two steps: 1) analysis within a TDF model, 2) analysis of the TDF cluster. Please note, in step one, the output ports defined in a TDF model are assigned *X* in place of use *u*, and use model *um*. In step 2, binding information is used to map the output ports with *X* to correct TDF models. If the use exists inside the TDF model, the *X* is replaced with the new use location. Otherwise, it is left as it is. Similarly, the input ports are assigned the start location of their TDF model initially, or location of *initialize()* function. Later, they are also resolved using binding information.

Finally, dynamic analysis is executed by instrumenting the TDF model while parsing AST. For every definition/use detected, a print instruction is written just before the statement. This print instruction logs the information related to the variable/port (location, TDF model name). But this only applies to the statements inside a TDF model. As soon as TDF cluster is analyzed, a print instruction has to be placed either inside the TDF component (could be a library component), or inserted in parallel as a separate TDF model. By parallel insertion, it means that the data (signal) flowing into the redefinition element (gain, delay etc.) also flows into the parallel TDF model, termed *parallel_print()*. We found *parallel_print()* to be less intrusive as the library components remain unchanged. Once the TDF cluster is instrumented, a testsuite is executed. Each testcase (test input signal) exercises different definitions and uses, and data flow associations can be created using the following way. Each definition is mapped on to a corresponding use as soon as it is encountered. If there exists a use, but not definition, it is notified as a warning. The data flow associations are reported to the verification engineer.

VI. EXPERIMENTAL RESULTS

In this section we present a case study to demonstrate the proposed DFT approach for SystemC-AMS TDF models. The experiments were carried out on two real-world AMS systems, 1) Car window lifter system, 2) Buck-boost converter [19]. AMS system design implementation and simulations were carried out in COSIDE SystemC-AMS tool environment [20]. Both AMS systems are implemented using SystemC-AMS TDF models. The results are summarized in Table II. In the following, both experiments are briefly explained.

A. Car Window Lifter System

In the first experiment, we consider a windows lifter system for cars. The AMS system controls the windows movements (up and down), while ensuring the passengers are not harmed.

TABLE II
CASE STUDY: CAR WINDOW LIFTER SYSTEM, AND BUCK-BOOST CONVERTER DATA FLOW ASSOCIATIONS

AMS Systems	Iter.	Tests	Data Flow Associations					
			Static (#)	Dynamic Def-Use Pairs				
				T (#)	S (%)	F (%)	PF (%)	PW (%)
Car Window Lifter	0	17	573	446	86	81	0	67
	1	20		467	87	82	0	76
	2	23		487	90	84	0	81
	3	26		525	93	89	0	93
Buck Boost Converter	0	10	362	243	70	65	100	100
	1	15		268	76	72	100	100
	2	20		282	81	75	100	100
	3	24		307	89	81	100	100

T: Total S: Strong F: Firm PF: PFirm PW: PWeak

Current flowing through the lifter motor is measured continuously as the window moves. In case of an obstacle (e.g., passenger's hand), the current flow changes, signaling the controller to stop. The system contains an Electrical Control Unit (ECU), and a complete window environment containing the motor, the mechanical parts including the window, and the control buttons. The ECU model includes a motor current filter to remove noise from current measurement, ADC for the motor current conversion, a current detector for over-current detection, the button logic (updown_decoder) and the micro-controller. During the simulation, the obstacle is inserted (and removed) at different times, and different window positions in to the system.

The car window lifter system has 17 testcases in the testbench initially, achieving 78% data flow coverage. There were 573 def-use pairs identified by our static analysis, out of which 446 were exercised by the dynamic analysis. Out of 446 exercised pairs, 86% Strong, 81% Firm, and 67% PWeak def-use pairs were exercised. There were no PFirm def-use pairs identified. The *all-defs*, *all-uses* criterion are not satisfied, hence, *all-dataflow* is also not satisfied. Table II shows four iterations where 9 testcases were added, and coverage increased. During analysis, two types of bugs were discovered, 1) use of ports in TDF models without definitions, 2) dynamic TDF induced failures. In this case, the timestep was reduced to accurately determine the hindrance while closing the window. Due to the change, the threshold comparisons failed in certain cases (specially current feedback loop) leading to def-use pairs being not exercised. These sorts of bugs can prove fatal when undefined behavior goes unchecked.

B. Buck-Boost Converter

In the second experiment, an energy efficient buck-boost converter [19] is used. It is a type of a DC/DC converter, and can operate in two modes, 1) step-down converter (buck), 2) step-up converter (boost). It is commonly used in IoT devices that are often powered with a battery. The main challenge of buck-boost converter is the switching frequency control algorithm. The algorithm monitors the current flowing. The controller sets the mode of the converter (buck or boost), the expected output value, and the maximum current allowed to flow through the converter. 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 is applied and a target voltage is programmed via the controller.

The buck-boost converter has 10 testcases in the testbench initially, achieving 67% data flow coverage. The static analysis found 362 data flow associations in total, out of which only 243 were exercised by the testcases. The *all-defs* criteria is not satisfied, hence, *all-dataflow* is also not satisfied. Out of 243 exercised pairs, 70% Strong, 65% firm, 100% PFirm, and 100% PWeak def-use pairs were exercised. *all-PFirm*, and *all-PWeak* criterion are satisfied. Table II shows four iterations with addition of 14 more testcases, which increase coverage. We found that in some cases, the ports were not defined, but still used in a different TDF model. This is undefined behavior according to SystemC-AMS standards [1]. This cannot be detected by line coverage, as it will still be satisfied.

VII. CONCLUSION

In this paper we presented the first DFT approach for SystemC-AMS TDF models and a coverage criteria specific to it. At the heart of the proposed work is a scalable static analysis which operates directly on the SystemC-AMS TDF models. It considers the semantics of TDF signal flow, and dynamic TDF. Four data flow associations w.r.t. semantics of TDF models are proposed (Strong, Firm, PFirm, PWeak). In addition, using static and dynamic analysis, automatic computation of data flow coverage results is explained for a given DUV. Improvement of coverage results by adding tests for uncovered data flow pairs is also discussed. We have demonstrated the applicability in a real world VP showing the results of one IP model. Furthermore, we plan to investigate our proposed methodology on system-level verification of mixed-signal platforms using the RISC-V VP from [21].

REFERENCES

[1] M. Barnasconi, C. Grimm, M. Damm, K. Einwich, M. Louërat, T. Maehne, F. Pecheux, and A. Vachoux, "Systemc ams extensions user's guide," *Accellera Systems Initiative*, 2010.

[2] T. Vörtler, K. Einwich, M. Hassan, and D. Große, "Using constraints for SystemC AMS design and verification," in *DVCon Europe*, 2018.

[3] F. Haedicke, H. M. Le, D. Große, and R. Drechsler, "Crave: An advanced constrained random verification environment for systemc," in *System on Chip (SoC), 2012 International Symposium on*. IEEE, 2012, pp. 1–7.

[4] J. W. Laski and B. Korel, "A data flow oriented program testing strategy," *IEEE Trans. Softw. Eng.*, vol. 9, no. 3, pp. 347–354, May 1983.

[5] S. Rapps and E. J. Weyuker, "Selecting software test data using data flow information," *IEEE Trans. Softw. Eng.*, vol. 11, no. 4, pp. 367–375, Apr. 1985.

[6] L. Copeland, *A practitioner's guide to software test design*. Artech House, 2004.

[7] T. Su, K. Wu, W. Miao, G. Pu, J. He, Y. Chen, and Z. Su, "A survey on data-flow testing," *ACM Computing Surveys (CSUR)*, vol. 50, no. 1, p. 5, 2017.

[8] D. Walter, S. Little, C. Myers, N. Seegmiller, and T. Yoneda, "Verification of analog/mixed-signal circuits using symbolic methods," *TCAD*, vol. 27, no. 12, pp. 2223–2235, 2008.

[9] K. Lata, S. K. Roy, and H. Jamadagni, "Towards formal verification of analog mixed signal designs using spice circuit simulation traces," in *ASQED*. IEEE, 2009, pp. 162–172.

[10] S. Gupta, B. H. Krogh, and R. A. Rutenbar, "Towards formal verification of analog designs," in *ICCAD*, 2004, pp. 210–217.

[11] M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *TCAD*, vol. 37, no. 2, pp. 378–391, 2018.

[12] E. Fraccaroli and F. Fummi, "Analog fault testing through abstraction," in *DATE*, 2017, pp. 270–273.

[13] E. Fraccaroli, F. Stefanni, F. Fummi, and M. Zwolinski, "Fault analysis in analog circuits through language manipulation and abstraction," in *FDL*, 2017, pp. 1–7.

[14] M. Hassan, V. Herdt, H. M. Le, M. Chen, D. Große, and R. Drechsler, "Data flow testing for virtual prototypes," in *DATE*, 2017, pp. 380–385.

[15] M. Hassan, D. Große, H. M. Le, T. Vörtler, K. Einwich, and R. Drechsler, "Testbench qualification for SystemC-AMS timed data flow models," in *DATE*, 2018, pp. 857–860.

[16] M.-M. Bidmeshki, A. Antonopoulos, and Y. Makris, "Information flow tracking in analog/mixed-signal designs through proof-carrying hardware ip," in *DATE*, 2017, pp. 1707–1712.

[17] A. Devices, "Relative humidity measurement system," http://www.analog.com/media/en/reference-design-documentation/reference-designs/CN0346.pdf.

[18] C. Lattner, "Llvm and clang: Next generation compiler technology," in *The BSD Conference*, 2008, pp. 1–2.

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

[20] C. Technologies, "Coside®," http://www.coseda-tech.com.

[21] V. Herdt, D. Große, H. M. Le, and R. Drechsler, "Extensible and configurable RISC-V based virtual prototype," in *FDL*, 2018, pp. 5–16.