III. Testing Through Abstraction

The structure of the proposed testing framework generation flow is shown in Figure II. The steps of the flow are represented by green boxes while design models as blue boxes with a folded corner. The generation flow accepts as input model an AMS description of a DUT. Without lack of generalization, Verilog-AMS semantics is considered as reference for AMS descriptions [12]. The symbols \( \text{ddt} \) and \( \text{sd} \) are respectively the derivative and integral operators. The Verilog-AMS description shown in Listing I is used as a guiding example.

A. Fault injection

The proposed methodology acquires in the first step the circuit topology \( (i.e., \text{nodes and branches}) \) of the analog model.
A contribution statement is defined in Verilog-AMS the behavior of an analog model is defined as a list of differential equations implemented by using the branch contribution statement. A contribution statement is defined by using the branch contribution operator (i.e., \(<+\) which is composed by a left-hand side (LHS) and a right-hand side (RHS). The former specifies the physical quantity of a branch to which the value of the latter will be assigned. There are two categories of physical quantities associative to analog nodes: potential and flow. These can be accessed by means of an Access Function, which accepts as arguments two analog nodes and returns the potential difference or the flow between them (e.g., functions \(V\) and \(I\) in Listing I). Inside the Verilog-AMS semantics there is a rule that state that: whenever an access function used between two nodes, it implicitly defines a branch between them. For instance, the access function \(I(n_1, n_2)\) on line 8 of Listing I implicitly defines a branch between the nodes \(n_1\) and \(n_2\). The topology of the circuit can be retrieved by applying such a rule to all the access functions contained inside a Verilog-AMS model.

The knowledge about the topology allows to inject all types of faults represented by saboteurs and mutants that mutate the structure of the circuit. This is done by adding a new equation to the analog model. A short circuit can be modeled by injecting a 1 Ω resistor between the pair of nodes of an existing branch. An open circuit can be modeled by injecting a 10 MΩ resistor in series with an existing branch. A current pulse can be modeled by injecting a controlled current source with the equation

\[
I(n_1, n_2) <+ \text{pulse};
\]

where \(\text{pulse}\) is a variable used during the simulation to model the desired behavior of the pulse. Also components parameters can be injected by inducing a subtle deviation from designer’s specifications. The fault model chosen for this work is the current pulse, placed between each internal node and the ground node. If applied to the circuit of Listing I, the list of injectable nodes are: \(n_1, n_2\) and \(out\). The fault injection step generates a faulty analog circuit model for each fault.

**B. Analog model abstraction**

The second step of the generation flow implements the process of abstraction to both fault-free and mutated descriptions. By referring to the abstraction levels reported in [8], the abstraction used in this paper is a process that transforms an analog model at the circuit level to a model at the functional level. The guiding idea of the abstraction is to automatically transform an analog model to a signal flow description by finding the input-output relations of a subset of the model values. These values, will be referred from now on as values of interest. The automatic definition of such values is a key aspect for applying the abstraction to the faulty models.

1) *Equation system enrichment:* This step has the purpose of gathering all the branches equations implied by the two principles of conservation of energy and charge. Applied to the electrical domain these equations are described by the first and second circuit laws of Kirchhoff applied respectively to nodes and loops. The equations relating the same node or the same loop are in a relation of linear dependency.

2) *Cone of influence exploration:* This phase derives the signal flow description of each value of interest by using the system of equations enriched at the previous step. Such description is built by selecting one of the possible subsets of equations describing the behavior of the value of interest w.r.t. the inputs of the model. This is done by means of a graph representation of the system of equations. A node is associated to each equation and labeled with LHS variable of the equation. Then, an edge connects a node \(A\) to a node \(B\) whenever the LHS variable of the equation associated with the second node \(B\) appears on the RHS of the equation associated with the first node \(A\). The graph is then visited for each value of interest, by starting from a node that represents it. At each step, the equation represented by a visited node is stored inside the current subsets of equations and the node is disabled. All the nodes associated to equations belonging to the set of linearly dependent equations of the selected one are disabled. Disabled nodes cannot be visited again. The visit ends when all the nodes are disabled. The behavior of saboteurs is preserved by this step, since the visit always selects one equation for each branch (whether describing its potential or flow). Then, the differential equations belonging to the acquired subset are discretized by using state of art techniques of numerical differentiation and integration. Let us suppose that all derivative operators are discretized using the backward difference formula and that the access functions are replaced by symbols. If we apply the discretization with a

```c++
1 void process(const double &V_in)
2 // Evaluate the values of interest.
3 V_out_gnd = 0.546961 * V_in - 0.77348 * V_n1_out_ddt;
4 V_n1_out = 0.011049 * V_in - 0.99447 * V_n1_out_ddt;
5 // Update auxiliary variables.
6 V_n1_out_ddt = V_n1_out;
7 }
```

Listing II: Abstracted Operational amplifier written in C++.
simulation step $h$ to the current-voltage relation of the equation shown on line 9 of Listing I, we obtain:

\[ I_{n1\_out} <+ \left( V_{n1\_out} - V_{n1\_out\_ddt} \right)/h \times c1; \]  

where $I_{n1\_out}$ and $V_{n1\_out}$ are respectively the current flowing through and the voltage across the capacitor at time $t$. While $V_{n1\_out\_ddt}$ is the voltage across it at $t-h$. All the values that were argument of a time-dependent function (e.g., $V_{n1\_out}$) are also considered as values of interest. They are required in order to evaluate at each simulation step the differentials and integrals auxiliary variables introduced by the discretization (e.g., $V_{n1\_out\_ddt}$).

3) Equation system symbolic solving: Each discretized system of equations has to be symbolically solved by considering as unknowns the values of interest. This operation requires a symbolic solver, i.e., an engine able to manipulate mathematical expressions. However, before solving the system of equations, all the symbols referring to the components variables are replaced by their numerical values (e.g., $x1$ with 400 in Listing I). The solution of the system is a set of linearly independent equations each of which has the following form:

\[ \text{lhs} = \sum_i \left( a_i \times c_i \right) + \sum_j \left( i_j \times c_j \right) \]  

where $\text{lhs}$ is a value of interest, $c_i$ and $c_j$ are numerical constants, $a_i$ is the i-th auxiliary variable and $i_j$ is the j-th input of the model. Listing II shows the C++ code generated by applying the abstraction flow to the running example with the voltage between nodes $\text{out}$ and $\text{gnd}$ as value of interest.

C. Testing framework building

A complete testing framework is built by using the previously solved fault-free and the mutated set of equations. Such a framework has the main role of orchestrator and is composed of two main functions: one executing the fault-free description and the other which executes the faulty one. The latter has two main tasks: 1) selects one of the faulty set of equations and 2) checks if the simulation time is inside the activation window of the selected fault. If it is outside that window, the fault-free set of equations is used instead. The two orchestrator’s functions are wrapped by a testing environment which: 1) Generates the input stimuli, 2) executes the two functions and provides the outputs of interest to an comparator (See Section III-D) and 3) finally updates all the auxiliary variables based on the discretization technique. If applied to the Equation 1, the update phase assigns the value $V_{n1\_out}$ to $V_{n1\_out\_ddt}$.

D. Fault simulation

The final phase starts the fault simulation and checks the presence of faults. At the end of each simulation step the outputs of interest are tested by means of a comparator function. Such a function evaluates the absolute value of the instantaneous difference between the response of the fault-free and faulty descriptions. Then, the value is compared with a threshold provided by the designer. If the value exceeds the threshold then the simulation is dropped and the fault is considered as detected. In conclusion, the proposed generation flow generates a C++ code which implements a complete testing framework able to simulate all modeled faults.

IV. EXPERIMENTAL RESULTS

The presented testing framework generation flow has been implemented in the ADVantage tool, i.e., the ADVanced Analog Testing frAmeWork GEneration flow. It exploits the tool HIFSuite [13] for parsing and manipulating the input descriptions. Verilog-AMS models have been simulated by using ELDO provided by Questa ADMS. All the experiments have taken place on a 64-bit machine running Linux with 16 GB of memory and Intel i7-5770 @ 3.40GHz.

A. Testing framework generation flow validation

The testing framework generation flow has been validated by using a set of nine benchmarks taken from an online repository of Verilog-AMS models\footnote{www.designers-guide.org}. Table I reports the features of each benchmark that are: the number of equations, inputs, outputs, internal nodes, lines of code of the Verilog-AMS description and the time required to generate its testing framework. The generation time depends mainly on the number of dd $t$ and id $t$ that have been discretized and on the number of injected faults. The former increases the number of auxiliary variables and thus the complexity of the expressions contained inside the final model. The latter increases the number of faulty systems of equations that have to be solved. The negligible generation times reported in table highlight the efficiency of the proposed flow even with the increasing complexity of the models. The normalized root-mean-square-deviation (NRMSD) between the response of the Verilog-AMS circuit level and the generated functional models has been evaluated in order to measure the accuracy of the models. In the worst case the NRMSD is $10^{-3}$. Table II reports the simulation time required by Questa-ADMS and the proposed framework for simulating a fault-free description, a faulty version with one active fault and the fault injection campaign. The overhead caused by the mutation of the code is reported for the single active fault scenario. The simulation has been performed for 1 second of simulated time with a fixed timestep of 5 ns. Table III reports instead the results with a timestep which has been automatically set by Questa-ADMS to 100 ns during the simulation. The same timestep has been used with the C++ testing framework. All the faulty simulations do not drop the simulation if a fault is detected. A sequence of values generated from a sinusoidal function has been used as testbench in both the simulation environments. The generated framework is able to achieve from two to three orders of magnitude of speed-up, thus proving to be a valuable solution for the dependability evaluation of an analog device.

B. Testing analysis

The availability of a so efficient functional testing framework allows to perform some testing analyses that would be computationally unfeasible by using a circuit level simulator. The results of a threshold sensibility analysis performed by using the testing framework of the Accelerometer are shown in Table IV. It reports: the type, number and percentage of

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
Benchmark & Equations & Inputs & Outputs & Internal Nodes & LoC & Generation Time (s) \\
\hline
RC1 & 2 & 1 & 1 & 1 & 17 & 0.01 \\
\hline
IN2 & 5 & 2 & 1 & 2 & 21 & 0.01 \\
\hline
PDFilter & 4 & 1 & 1 & 2 & 21 & 0.01 \\
\hline
IN3 & 5 & 3 & 1 & 3 & 31 & 0.02 \\
\hline
Op-Amplifier & 6 & 1 & 1 & 3 & 31 & 0.03 \\
\hline
RC5 & 10 & 1 & 1 & 5 & 42 & 0.03 \\
\hline
RC10 & 20 & 1 & 1 & 10 & 67 & 0.17 \\
\hline
RC20 & 40 & 1 & 1 & 20 & 117 & 1.45 \\
\hline
Accelerometer & 66 & 10 & 8 & 25 & 123 & 0.30 \\
\hline
\end{tabular}
\caption{Benchmarks features and framework generation time.}
\end{table}
detected faults, the detection threshold and the time required to generate and simulate the framework. The framework has been simulation for 1 second of simulation time with a timestep of 100 ns. For these experiments the fault dropping feature has been used. For the purpose of the experiment the current pulse with a variable amplitude (in Ampere) has been used. The aim of this experiment is to find the maximum threshold that allows to detect all the faults. Table V reports the threshold sensibility analysis performed by using the testing framework of RC20. In this case the analysis has been extended to a wider range of faults including: a current pulse and both short and open circuits. The results of both the tables highlight the efficiency of both the generation flow and testing framework with different types of faults, allowing to perform an effective testing campaign. Thanks to the flexibility of the injection process, it was possible to inject different kinds of fault inside the more complex proposed test case (i.e., RC20).

### V. CONCLUDING REMARKS

A fault testing methodology based on the abstraction of analog and mixed-signal models has been described in details. The employed abstraction technique produces a signal flow representation that preserves the faulty behaviors. The speed-up of some orders of magnitude is an enabling factor for performing extensive fault campaigns which normally would require a large amount of time. The flexibility of the injection flows presented in this paper allows to apply it to a wide range of fault models. Future work will extend the flow to automatically generate also the set of input stimuli.

### REFERENCES


