An Efficient Behavioral Description Frontend Tool for Mixed-Mode SPICE Simulation

Álvaro Gómez-Pau, Luz Balado, Joan Figueras
Universitat Politècnica de Catalunya - UPC
Departament d'Enginyeria Electrònica
Barcelona, Spain

Abhijit Chatterjee
Georgia Institute of Technology
Atlanta, USA

***Abstract*—Analog and mixed-signal circuit design demands high amounts of knowledge, expertise and intuition. To that purpose, it is not deniable electrical CAD tools play a crucial role assisting designers in such complex scenario. SPICE-like electrical simulators have been established through the years as the de facto standard for analog and mixed-signal circuit simulation. Such simulation programs work at low level circuit description dealing with explicit component instantiations and electrical connections between them with no possibilities of description at a higher level. In this work, a software tool partially addressing this issue is proposed. The developed tool allows the designer to simulate a signal flow graph (SFG) given its behavioral description. The proposed tool efficiently transforms a behavioral SFG description into a netlist suitable for SPICE. It also generates the required graphical commands for rendering the graph as well as the listing of node equations to be solved algebraically. In order to evaluate the tool, it has been applied to behaviorally describe a transient error and fault tolerant scheme applied to a Biquad filter. The tool has been proven to be efficient in terms of its capabilities for mixing structural and behavioral descriptions in a single netlist, as well as for its simplicity and usefulness when dealing with large linear systems, therefore validating the proposal.**

***Index Terms*—Signal Flow Graph Simulation, Behavioral Description, Behavioral Simulation, Electrical Simulation, Electrical CAD, SPICE, Analog Checksums, Fault Tolerance, Error Cancellation, Linear Analog Filters, Biquad Filter**

I. Introduction

Design of analog and mixed-signal circuits is a complex task requiring high doses of expertise and intuition, specially when dealing with current submicron devices. In order to cope these difficulties, electrical simulators have assisted analog designers since the firsts pieces of software were written for such purposes [1]–[3]. Nowadays, there exist a vast number of electrical simulators in the market, each of them oriented to concrete applications such as digital and analog/RF domains or even featuring different system/circuit description approaches. Among this variety, one of the best well known simulators is SPICE, developed at Berkeley in 1973 [2] and still maintained and distributed today.

The SPICE program reads a circuit netlist and outputs simulation results according to the indicated analysis type. Simulator's syntax is pretty simple, although several commercial versions of SPICE have implemented extra features and language variants [4]. Basically, each circuit component is instantiated by explicitly indicating its electrical connections to other circuit components. By means of the modified nodal analysis, the simulator writes the nodal equations which are solved using advanced numerical integration methods.

Despite its tremendous utility, SPICE-like simulators are mainly oriented to deal with low level circuit descriptions, not allowing to implement functional descriptions. In the last years, some options have appeared to achieve such purpose. Similarly to digital hardware description languages, extended HDL versions have been developed to cope the necessities of analog and mixed-signal circuits [5] description and simulation. For instance, Verilog-A and Verilog-AMS [6] compact modeling languages allow circuit's description in a higher level of abstraction, therefore concentrating efforts in defining its functionality rather than its structure, hence improving design and simulation timings.

Circuit functionality can be described in many ways, but one of the most used, specially in linear analog circuits is using signal flow graphs (SFG) [7]. Such representation provides a functional, understandable and compact way of describing circuit's behavior. Control theory makes an intensive use of block diagrams and specific tools have been developed in the field, for instance, the well known MATLAB Simulink software [8]. Although it presents vast advantages for high level simulation, it does not allow the mix of behavioral and structural descriptions. Also, the GUI interface may result inefficient for large systems. Quite often, the use of strictly electrical simulation tools is mandatory because of the need to accurately simulate a certain circuit, but it would be desirable to describe some parts using a higher level of abstraction by means of signal flow graphs.

In this work, a Perl [9] tool that allows a behavioral SFG description to be simulated in SPICE is presented. The developed tool complements SPICE simulator by extending its capabilities to simulate behavioral SFG descriptions. This allows the user to mix structural and behavioral descriptions in the same netlist, therefore facilitating circuit design, analysis and validation. The developed tool also generates the listing of graphical commands to render the graph using GraphViz [10] and the node equations to be solved algebraically using a computer algebra system [11].

The work is organized as follows. Section II describes how the `sfg2hsp` tool parses a behavioral description of a signal flow graph and generates a low level description suitable for SPICE. Section III focuses on the `tf2sfg` tool which extends the functionality of the first tool to arbitrary transfer functions branches. In section IV, the developed tool is applied to simulate and analyze a Biquad filter provided with a transient error and fault tolerant circuitry proposed in [12]–[14] using analog checksums. The analog checker/correction circuitry is described using a SFG and simulated by means of the developed tool. Finally, section V highlights the main benefits of the tool and concludes the paper.

978-1-4799-5743-9/14/$31.00

II. Behavioral Signal Flow Graph Description and Its Equivalent Structural Description

A linear signal flow graph can be described by a set of p nodes/signals $X = \{x_1, \ldots, x_p\}$, and a set of q directed branches $A = \{a_1, \ldots, a_q\}$. Each branch $a_k \in A$ is comprised of two nodes, $(x_i, x_j) \in X^2$, here after referred as *departure node* and *destination node* respectively, and a certain gain, $\alpha_{ij}(s)$, between them. In the general case, $\alpha_{ij}(s)$ may be an arbitrary transfer function, though let us assume by now, α_{ij} represents a scalar or an integrator. Each branch describes a certain additive contribution to the *destination node* as a function of its corresponding *departure node*, as the following equation states, $\Delta_k x_j = \alpha_{ij} x_i$. Notation $\Delta_k x_j$ indicates the contribution to node x_j by k-th branch. The assumption of scalar or integrator branches does not restrict the use of arbitrary transfer functions as will be shown in section III.

The developed tool, `sfg2hsp`, reads a signal flow graph expecting a single branch description in each line following the format `xi:alphaij:xj`, where `xi` is the *departure node*, `xj` is the *destination node* and `alphaij` is the gain between the nodes considered in the branch. As mentioned earlier, string `alphaij` may represent a scalar or an integrator. For the case of integrators, it is mandatory its last two characters to be "`/s`". Also, comments may be included in the SFG listing by appending them to symbol "`#`".

For the case of gain only branches, `sfg2hsp` tool implements each additive contribution by instantiating a voltage controlled voltage source (E element) with the gain determined in the branch description as sketched in the left hand side of Fig. 1. On the contrary, if a gain-integrator branch is found, the gain part is implemented as previously explained. Then, the resulting voltage is "copied" in μA by a voltage controlled current source (G element) and injected into a 1 μF capacitor, therefore achieving the desired integration as Fig. 1 sketches.

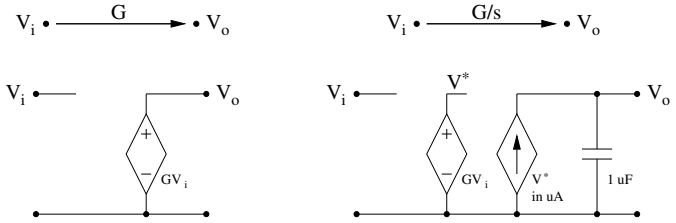

Fig. 1. Example of the structural description used by `sfg2hsp` of gain only branches (left) and gain-integrator branches (right). Note the use of controlled current/voltage sources and capacitors to perform the integration.

Finally, when all the branches have been processed, all the individual contributions $\Delta_k x_j$, are added together using a behavioral voltage source. Once this step is performed, the netlist generated by `sfg2hsp` can be fed into HSPICE [4] and simulated as usual. One interesting feature about the program is that it is capable of managing arbitrary parameters since it treats the input signal flow graph description as a sequence of strings. The netlist can be easily parameterized using HSPICE's `.param` instructions.

The developed tool also provides a listing of DOT commands to be fed into GraphViz [10] to generate the graph representation of the SFG. Of course, the graph layout is automatically decided and may be differently organized as it would be if drawn by hand, although it provides an automatic, efficient and effortless way of generating the graph of large LTI systems. Also, `sfg2hsp` tool provides the listing of node equations computed while parsing the input file. These equations can be fed into a computer algebra system (CAS) to solve for all nodes transfer functions symbolically. Specifically, `sfg2hsp` is instructed to provide the equations to be fed into the Maxima computer algebra system [11]. Such symbolic approach makes the simplification of arbitrary complex SFG a straightforward job avoiding errors and the tedious task of tackling the problem by hand. Fig. 2 summarizes tool's working flow and its capabilities.

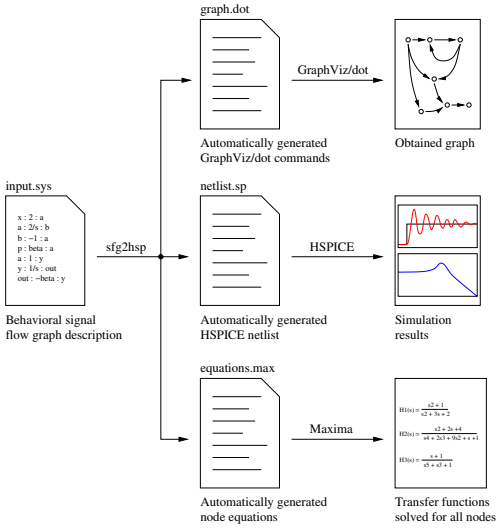

Fig. 2. Tool's working flow showing its automatically generated outputs: HSPICE netlist for electrical simulation, GraphViz/dot commands for generating the graph and Maxima commands for solving node equations.

III. Arbitrary Transfer Function Branches

In previous section, it has been assumed branch gains were scalars or integrators. In order to allow the use of arbitrary transfer functions as branch gains a new tool called `tf2sfg` has been developed as a complement and preprocessor to `sfg2hsp`. It parses a behavioral signal flow graph description containing transfer functions and substitutes them by an equivalent description containing only integrators and gains, hence `sfg2hsp` is able to process the file smoothly as described in previous section.

Let us assume $H(s)$ is a generic transfer function of a LTI system relating input $X(s)$ and output $Y(s)$. Let m and n be the degrees of transfer function's numerator and denominator respectively and assume a causal system, $n \geq m$,

$$H(s) = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0} = \frac{Y}{X} \quad (1)$$

Multiplying previous transfer function's numerator and denominator by s^{-n} and equating the result, one can isolate response $Y(s)$ as a linear combination of input and output signal integrations,

$$\begin{aligned} Y = &- \frac{a_{n-1}}{a_n} s^{-1} Y - \cdots - \frac{a_1}{a_n} s^{1-n} Y - \frac{a_0}{a_n} s^{-n} Y \\ &+ \frac{b_m}{a_n} s^{m-n} X + \cdots + \frac{b_1}{a_n} s^{1-n} X + \frac{b_0}{a_n} s^{-n} X \end{aligned} \quad (2)$$

Therefore, the transfer function $H(s)$ can be implemented using integrators and feedback/forward networks as Equa-

tion (2) states and Fig. 3 sketches. Such implementation corresponds to the well known *input feedforward form* [7].

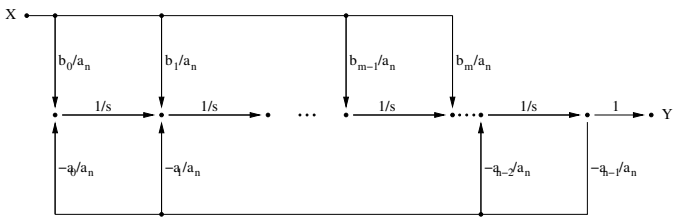

Fig. 3. Signal flow graph description of a causal transfer function using integrators and feedback networks [7]. This implementation corresponds to the so called input feedforward form.

The way of defining a branch containing an arbitrary transfer function is similar to the one described in section II for scalar/integrator branches. The *departure node*, transfer function gain and *destination node* need to be listed and separated by a colon symbol, following this format `xi:[NUMCOEFS],[DENOMCOEFS]:xj`. Transfer function's numerator and denominator are described using comma separated arrays containing their coefficients in descending degree order, in a similar way MATLAB represents polynomials [8].

Fig. 4 sketches the working flow using the `tf2sfg` tool. Regardless of the existence or not of transfer functions between any pair of nodes, the `tf2sfg` and `sfg2hsp` programs can be applied in pipe since `tf2sfg` will keep its input unchanged if no transfer functions are found. If a transfer function is detected during the parsing of the input file, `tf2sfg` will substitute such line by an equivalent signal flow graph using the input feedforward form depicted in Fig. 3.

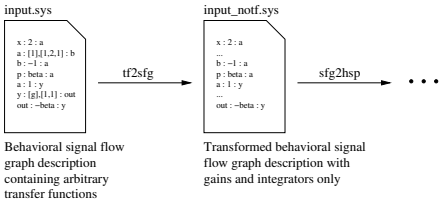

Fig. 4. `tf2sfg` tool working flow showing the parsing of a signal flow graph containing arbitrary transfer functions in branch gains. The generated output can be directly fed into `sfg2hsp` tool.

IV. Case Study: Biquad Filter with Transient Error/Fault Compensation

A. Biquad Filter Structural Description

In order to illustrate the capabilities of the developed tool, it will be used to simulate a Biquad filter provided with the transient error and structural fault compensation scheme proposed in [12]. The filter is described structurally as an HSPICE netlist while the transient error/fault compensation scheme is described behaviorally using a signal flow graph. The chosen system is rather simple for the sake of illustration purposes, although the developed tool may be applied to much more complex systems.

Filter's topology corresponds to a KHN Biquad with three outputs: high-pass, band-pass and low-pass as can be observed in Fig. 5. The filter is a second order linear system and has been implemented using operational amplifiers and passive components. The Biquad filter has been tuned at $f_0 = 100$ kHz with quality factor $Q = 3$ and unity gain.

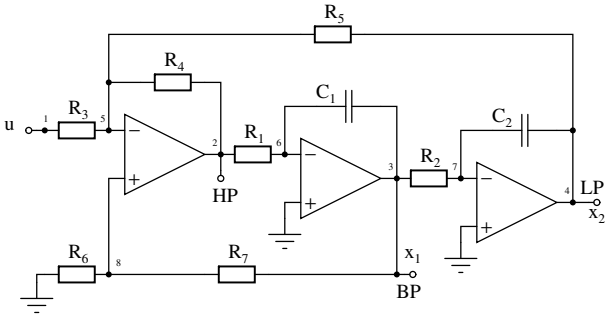

Fig. 5. Schematic of the Biquad filter used as case study. It has been implemented using operational amplifiers and passive components. Filter's characteristic frequency is $f_0 = 100$ kHz and quality factor $Q = 3$.

B. Transient Error/Fault Compensation Scheme Behavioral Description

As mentioned earlier, a transient error/fault compensation scheme using analog checksums for the presented Biquad filter will be used as case study. In [12], [15] the concept of analog checksums, also called real numbers checksums, derived from the real number codes of [16] was first proposed. In [13], [17] the method was applied for checking linear analog circuits using DC signals. The use of analog checksums has also been shown to be effective for checking control systems [18] as well as for achieving almost perfect compensation of transient error/noise signals in linear analog filters [14].

The second order linear filter presented in Fig. 5 can be easily represented by means of state space equations in the form $sX = AX + BU$. Let x_1 and x_2 be the state variables corresponding to filter's band-pass and low-pass outputs. If $u(t)$ represents filter's input signal, one possible state space representation is given by,

$$-\frac{s}{\omega_0}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/Q & -1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \end{bmatrix} u \quad (3)$$

According to the theory presented and developed in [12], [14], the system in Equation (3) can be extended by a redundant state x_r which corresponds to a linear combination of filter's original states. Let $\bar{\alpha} = [\alpha_1, \alpha_2]$ be the so called *coding vector* and $\beta < 0$ the stabilization feedback to be used in the analog checksum generation. Then, the resulting system becomes,

$$-\frac{s}{\omega_0}\left[\begin{array}{c} x_1 \\ x_2 \\ \hline x_r \end{array}\right] = \left[\begin{array}{cc|c} 1/Q & -1 & 0 \\ 1 & 0 & 0 \\ \hline a_1^* & a_2^* & -\beta \end{array}\right]\left[\begin{array}{c} x_1 \\ x_2 \\ \hline x_r \end{array}\right] + \left[\begin{array}{c} -1 \\ 0 \\ \hline b^* \end{array}\right] u \quad (4)$$

Where the extra rows $a^* = [a_1^*, a_2^*]$ and b^* are computed as $a^* = \bar{\alpha}(A + \beta I)$ and $b = \bar{\alpha}B$ respectively. See [12] or [14] for further details and theoretical derivations. According to previous work, an analog error signal can be naturally defined as $E = (\alpha_1 x_1 + \alpha_2 x_2) - x_r$. Since the defined redundant state x_r corresponds to a linear combination of filter's states, error signal E should be zero if no fault or transient error is present. Otherwise, if the Biquad filter presents a fault or any of the states are perturbed by an error/noise signal, E will present a non zero value.

It can be demonstrated that fault or transient error compensation can be achieved by feeding back the error signal to the perturbed state [12], [14]. Let k_1 and k_2 be the gains of such feedback for state x_1 and x_2 respectively. Then, the extended

system represented in Equation (4) can be described by the signal flow diagram sketched in Fig. 6. As can be observed, it performs the described linear combination of the original states and the corresponding integration to get a new redundant state x_r. Error signal E is computed as described earlier by subtracting x_r to $\alpha_1 x_1 + \alpha_2 x_2$. Branches in blue dashed line represent the compensation feedback while branches in red represent the transient error/noise signals upsetting filter states.

Up to now, the presented case study contains a mixed-mode description of the target system, i.e. the filter itself is structurally described and the analog checksum compensation scheme behaviorally described using a SFG. In this case, the use of a structural description for the Biquad is mandatory since we are investigating structural faults. Of course, the signal flow graph could also be described using electrical components but it may be tedious to design, specially for large systems, so keeping such high level description will facilitate its analysis and possible optimization. The use of `sfg2hsp` will allow system simulation by means of SFG description.

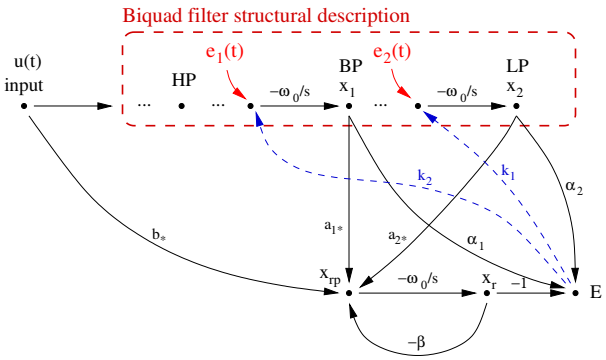

Fig. 6. Signal flow graph describing the analog fault/transient error compensation scheme applied to the structural description of the presented Biquad filter used as case study.

The behavioral signal flow graph describing the analog checksum compensation scheme is listed in Fig. 7. Such listing contains exactly the same information as the graph shown in Fig. 6 according to `sfg2hsp`'s required syntax exposed in section II. Every single line represents a branch in which the *departure node*, *destination node* and the gain between them are specified. As can be appreciated, `sfg2hsp`'s input listing admits the use of arbitrary parameters, i.e. `omega0`, `Q`, `alpha1`, `alpha2` and `beta`, as shown in Fig. 6. Note how integrator branches are denoted according to tool's syntax.

```
# to generate xr (redundant state)
u : -alpha1 : xrp
x1 : alpha1*(1/q+beta)+alpha2 : xrp
x2 : -alpha1+beta*alpha2 : xrp
xrp : -omega0/s : xr
# stabilization feedback
xr : -beta : xrp
# compute error signal
xr : -1 : e
x1 : alpha1 : e
x2 : alpha2 : e
```

Fig. 7. Behavioral description listing of the analog fault/transient error compensation scheme applied to the Biquad filter. Such listing becomes the input to the `sfg2hsp` tool. It corresponds to the description of the SFG branches shown in Fig. 6.

It may seem such text based representation is not as intuitive as its graphical equivalent, but the benefits it presents are clear. First, it is a straightforward way to achieve a precise and quick description of a signal flow graph without the need of drawing block diagrams, performing connections, or specifically inserting adder/subtractor operators as occurs in MATLAB Simulink. The design can be fully parameterized therefore obtaining remarkable generalization capabilities. Or what is more, if graph's structure needs to be modified, it can be done immediately by just changing text. Also, the simplicity of its syntax allows any other tool, script or programming language to generate it and modify it on the fly as desired.

C. Simulation Results

In order to validate the tool, some HSPICE simulations have been carried out to get the response of the whole system when a component deviates from its nominal value and the compensation scheme is applied. When the SFG behavioral description shown in Fig. 7 is fed into `sfg2hsp` tool it automatically generates the HSPICE netlist listed in Fig. 8. With the aim of debugging and tracing program's output, the tool also reports some comments containing useful information. As can be observed, the branches described in the listing shown in Fig. 7 are transformed into HSPICE component instantiations by means of controlled voltage/current sources and capacitors to perform the required integrations (refer to section II for further details).

```
* Begin parsing
* From: u, Gain: -alpha1, To: xrp
* Node: xrp (1)
e_xrp_1 xrp_1 gnd u gnd '-alpha1'
* From: x1, Gain: alpha1*(1/q+beta)+alpha2, To: xrp
* Node: xrp (2)
e_xrp_2 xrp_2 gnd x1 gnd 'alpha1*(1/q+beta)+alpha2'
* From: x2, Gain: -alpha1+beta*alpha2, To: xrp
* Node: xrp (3)
e_xrp_3 xrp_3 gnd x2 gnd '-alpha1+beta*alpha2'
* From: xrp, Gain: -omega0, To: xr
* Integrator found in current branch
* Node: xr (1)
e_xr_1 xr_1_i gnd xrp gnd '-omega0'
g_xr_1 gnd xr_1 xr_1_i gnd 1e-6
c_xr_1 xr_1 gnd 1e-6
* From: xr, Gain: -beta, To: xrp
* Node: xrp (4)
e_xrp_4 xrp_4 gnd xr gnd '-beta'
* From: xr, Gain: -1, To: e
* Node: e (1)
e_e_1 e_1 gnd xr gnd '-1'
* From: x1, Gain: alpha1, To: e
* Node: e (2)
e_e_2 e_2 gnd x1 gnd 'alpha1'
* From: x2, Gain: alpha2, To: e
* Node: e (3)
e_e_3 e_3 gnd x2 gnd 'alpha2'
* Summing nodes
e_e e gnd volt='+v(e_1)+v(e_2)+v(e_3)'
e_xrp xrp gnd volt='+v(xrp_1)+v(xrp_2)+v(xrp_3)+v(xrp_4)'
e_xr xr gnd volt='+v(xr_1)'
```

Fig. 8. Netlist describing the SFG of Fig. 6 automatically generated by `sfg2hsp` when it is fed with the listing of Fig. 7. The generated netlist can be simulated and parameterized using HSPICE.

Let us focus on the first branch. As can be observed, it corresponds to a gain only branch and therefore it is implemented using a voltage controlled voltage source with gain `-alpha1` between input node `u` and branch's contribution node `xrp_1`, which corresponds to the first found contribution to node `xrp`. On the contrary, to generate state `xr`, an integration is needed. As can be seen in the generated netlist, the tool instantiates a VCVS to apply the desired gain but also a current source to inject the current to a 1 μF capacitor to achieve signal integration as explained in section II. Finally, last three lines add all branches individual node contributions together to nodes `e`, `xrp` and `xr`, respectively.

Fig. 9 shows Biquad filter frequency response when capacitor C_1, presents a 200% deviation from its nominal value. As can be appreciated, such deviation causes a considerable shift in filter's magnitude Bode diagram as the solid traces clearly show. It is obvious all three filter outputs seem to be affected by parametric faults in C_1.

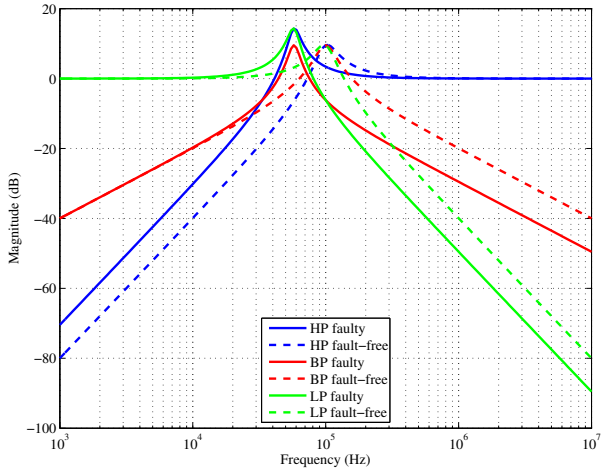

Fig. 9. Frequency response of the whole system when capacitor C_1 triples its value and no compensation is applied. As can be observed, the Bode diagram (solid traces) clearly deviates from its fault-free version (dashed traces).

When the analog compensation scheme presented in [15] and summarized in Fig. 6 is applied to state x_1, which is the one being affected by the parametric fault (capacitor C_1), the magnitude Bode diagram shown in Fig. 10 is obtained. In this case, a compensation gain of $k_1 = 50$ has been chosen. As can be observed, the resulting magnitude Bode diagram is compensated as the theory presented in [12] predicts.

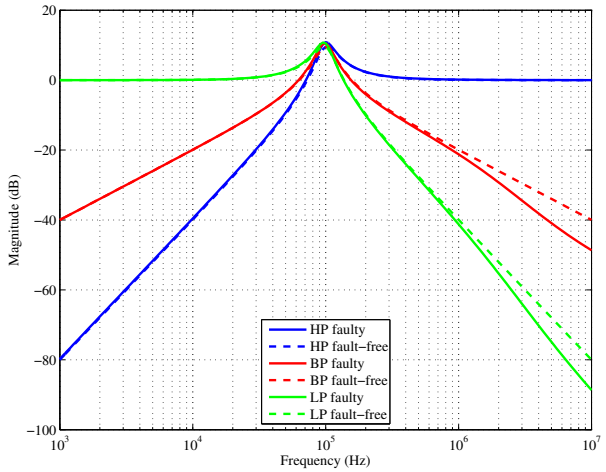

Fig. 10. Frequency response of the whole system when capacitor C_1 triples its value and the compensation scheme is applied. As can be observed, the compensated Bode diagram (solid traces) tends to its fault-free version (dashed traces). If gain k_1 is increased, better fault correction is achieved.

Also, transient simulations have been carried out. Fig. 11 shows filter's response to a unit step input. Again, it can be observed how all three outputs clearly differ from the nominal traces when the same fault is applied to capacitor C_1. Similarly to the results obtained in the frequency domain, when error signal E is fed back to state x_1 with gain $k_1 = 50$, the transient response is almost completely compensated as Fig. 12 illustrates.

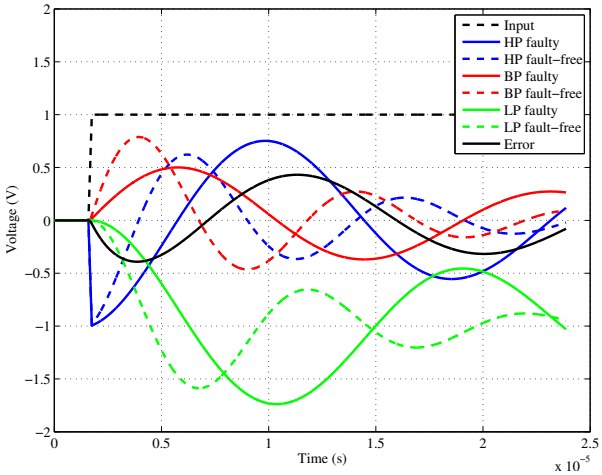

Fig. 11. Step response of the whole system when capacitor C_1 triples its value and no compensation is applied. As can be observed, filter outputs (solid traces) clearly deviate from the fault-free version (dashed traces).

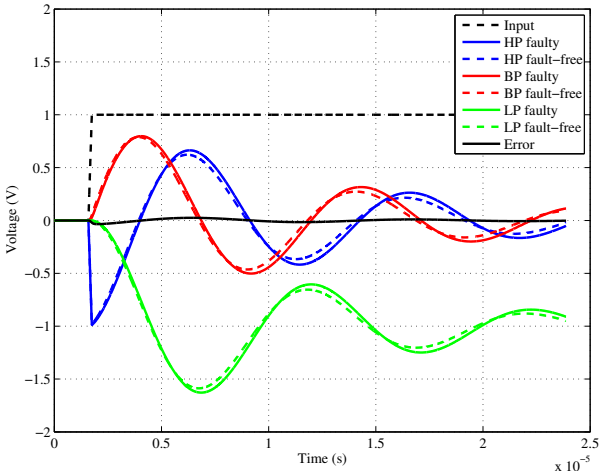

Fig. 12. Step response of the whole system when capacitor C_1 triples its value and the compensation scheme is applied. As can be observed, the compensated responses (solid traces) tend to its fault-free version (dashed traces). If gain k_1 is increased, better fault correction is achieved.

If any change needs to be made regarding the fault/transient error compensation scheme, it becomes a straightforward task using `sfg2hsp` since only HSPICE `.param` instructions need to be modified. The benefits provided by `sfg2hsp` are clear when simulating behavioral SFG description.

D. Graphical Output

As mentioned earlier, `sfg2hsp` is also capable of generating graphical output commands to be processed by GraphViz/dot utility to draw system's SFG. After rendering the commands generated by the tool, the graph shown in Fig. 13 is obtained. It can be appreciated the layout does slightly differ from the one depicted in Fig. 6. This is so because no indications of desired node position are passed to the tool.

The presented case study graph is extremely simple, but `sfg2hsp` graphical capabilities may be very desirable when dealing with complex graphs which may be hard to be drawn by hand.

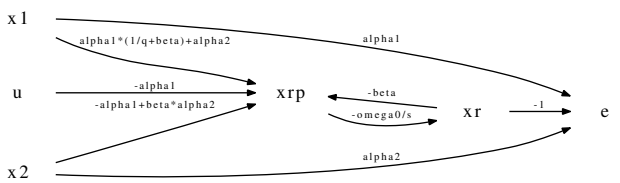

Fig. 13. Automatically generated graph by `sfg2hsp` and rendered by GraphViz/dot utility. It corresponds to the SFG depicted in Fig. 6 although presenting a slightly different layout.

E. Node Equations Solutions

It has been also mentioned the capabilities of `sfg2hsp` to generate node equations as it parses input behavioral SFG description. To that purpose, the developed tool internally stores all branch contributions to all system nodes and writes node equations as a linear system in a syntax understandable by most computer algebra systems.

Once the listing shown in Fig. 7 is parsed by `sfg2hsp`, the corresponding Maxima code to solve the system and get the transfer function of every single output node is generated. For illustration purposes, two transient errors e_1 and e_2 perturbing states x_1 and x_2 have been considered. The CAS system reports the following transfer functions for error signal E and low-pass output x_2,

$$E = \frac{-\alpha_1}{D_1(s)}\, e_1 + \frac{-\alpha_2}{D_1(s)}\, e_2 \tag{5}$$

$$x_2 = \frac{-1}{D_2(s)}\, u + \frac{N_1(s)}{D_1(s)D_2(s)}\, e_1 + \frac{N_2(s)}{D_1(s)D_2(s)}\, e_2 \tag{6}$$

Where $N_1(s)$ and $N_2(s)$ are first and second order polynomials and $D_1(s) = \frac{s}{\omega_0} + \alpha_1 k_1 + \alpha_2 k_2 - \beta$, which corresponds to the extra pole added by the analog compensation scheme as described in [14]. Denominator $D_2(s) = \frac{s^2}{\omega_0^2} + \frac{1}{Q}\frac{s}{\omega_0} + 1$, can be identified as the second order term characterizing the Biquad filter. As can be observed, if compensation gains k_1 or k_2 are large enough, the error signal tends to zero and filter's output tends to its error-free transfer function since the extra pole becomes non dominant, as derived in the theory presented in [14]. The diagram could be simplified by hand but such attempt may be error prone and time consuming. The presented case study is rather simple, but such node analysis capabilities would be very desirable when analyzing large signal flow graphs.

V. Conclusions

A tool for converting a behavioral signal flow graph description into a SPICE netlist has been proposed. Tool's input SFG description can be achieved by just listing the *departure* and *destination* nodes together with the gain between them. Such gain may be an arbitrarily complex causal transfer function which is converted into a subgraph with real gains and integrators by exploiting the input feedforward form of a transfer function. The developed tool generates a SFG equivalent netlist using controlled sources and capacitors. The tool is also capable of generating the required commands to render the graph as well as the node equations which allow to obtain all output nodes transfer functions with respect to system inputs.

The proposed tool has been applied to simulate a transient error/fault compensation scheme applied to a Biquad filter. The tool has proven its capabilities to assist SPICE simulation when structural and behavioral descriptions are used in the same system. The simplicity of the developed software permits the SFG description to be modified easily what makes the tool extremely useful when designing analog circuits and targeting optimum design. The tool is also capable to analyze the given SFG and provide the transfer function of all system nodes as well as the graphical commands to render the graph, what supposes a valuable help when dealing with large linear analog systems and therefore validating the proposal.

Acknowledgments

This research was supported in part by the Spanish Ministry of Economics and Competitiveness (mobility scholarship BES-2011-044129 attached to project TEC2010-18384) and by the US National Science Foundation.

References

[1] L. Nagel and R. Rohrer, "Computer analysis of nonlinear circuits, excluding radiation (CANCER)," *Solid State Circuits, IEEE Journal of*, vol. 6, no. 4, pp. 166–182, August 1971.

[2] L. W. Nagel and D. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)," EECS Department, University of California, Berkeley, Tech. Rep., Apr 1973.

[3] T. Perry, "Donald O. Pederson (electronic engineering biography)," *Spectrum, IEEE*, vol. 35, no. 6, pp. 22–27, June 1998.

[4] Synopsis, "HSPICE User Guide," September 2008, version B-2008.09.

[5] D. Potop-Butucaru, C. Lallement, and A. Vachoux, "VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 24, no. 2, pp. 204–225, February 2005.

[6] K. S. Kundert and O. Zinke, *The Designer's Guide to Verilog-AMS*. New York, USA: Springer Berlin Heidelberg, 2004.

[7] P. Mohan, *VLSI Analog Filters: Active RC, OTA-C, and SC*, ser. Modeling and Simulation in Science, Engineering and Technology. Springer, 2012.

[8] MATLAB and Simulink, *version 7.14.0 (R2012a)*. The MathWorks Inc., 2012.

[9] L. Wall, *Programming Perl*, 3rd ed. Sebastopol, California, USA: O'Reilly & Associates, Inc., 2000.

[10] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software - Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[11] Sourceforge.net, "Maxima, a Computer Algebra System," 2012, 5.27.0 http://maxima.sourceforge.net/.

[12] A. Chatterjee, "Concurrent error detection and fault-tolerance in linear analog circuits using continuous checksums," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 1, no. 2, pp. 138–150, 1993.

[13] S. Banerjee, A. Gómez-Pau, and A. Chatterjee, "Design of Low Cost Fault Tolerant Analog Circuits Using Real-Time Learned Error Compensation," in *Proceedings of IEEE European Test Symposium (ETS)*, Paderborn, Germany, May 2014, pp. 229–230.

[14] A. Gómez-Pau, S. Banerjee, and A. Chatterjee, "Real-Time Transient Error and Induced Noise Cancellation in Linear Analog Filters Using Learning-Asisted Adaptive Analog Checksums," in *Proceedings of International On-Line Testing Symposium (IOLTS)*, Platja d'Aro, Spain, July 2014, pp. 25–30.

[15] A. Chatterjee, "Checksum-based concurrent error detection in linear analog systems with second and higher order stages," in *VLSI Test Symposium*, 1992, pp. 286–291.

[16] V. S. S. Nair and J. A. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *Transactions on Computers, IEEE*, vol. 39, pp. 426–435, April 1990.

[17] A. Chatterjee, B. Kim, and N. Nagi, "DC Built-In Self-Test for Linear Analog Circuits," *Design and Test of Computers, IEEE*, vol. 13, no. 2, pp. 26–33, 1996.

[18] S. Banerjee, A. Banerjee, A. Chatterjee, and J. Abraham, "Real-time checking of linear control systems using analog checksums," in *On-Line Testing Symposium (IOLTS), 2013 IEEE 19th International*, 2013, pp. 122–127.