2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Modeling of Generic Transfer Functions in SystemVerilog. Demystifying the Analytic Model.

Elvis Shera, Methodology, Dialog Semiconductor, Nabern, Germany

elvis.shera@diasemi.com

***Abstract*—**with the increased complexity of integrated circuits (IC) and the importance of verifying at system level and running end to end tests to verify customer scenarios, new modeling techniques which are capable of high accuracy and still allow fast simulation are being adopted with increasing pace. SystemVerilog is becoming very attractive for modeling analog circuits after its latest release, SV-DC 2012, introduced user-defined resolution function (UDR) and user-defined data types (UDT). With these additions it has become easier to model resistive sources, both as Norton and Thevenin equivalent circuits, and to resolve the common node value. There are other cases though, where the model remains challenging because of too complex filtering or the presence of feedback loops. An example of such a circuit is an error amplifier which is a critical component in blocks such as LDOs and DC-DC switching converters. One possibility within Real Number Modeling (RNM) in SystemVerilog, is to consider the block behaves linearly and to characterize the behavior with its transfer function. A known approach to this is via the well-known Z-Transform. This is based on sampling the input by considering the overall requirement of the system in which the model is operating. Depending on the function we want to model, such an approach might be very difficult and require some pre-analysis. In this paper, a different technique in modeling the transfer function is presented. This technique is generic and is based on a time domain model. The starting point is the transfer function of the circuit in the s-domain. There is no assumption made on small or large signal models. The validity of the model is related to the validity of the transfer function of reference. Such an approach is mathematically exact, it is not based on sampling, and it shows both, high performance and accuracy in simulations.

Keywords*—*Black-box modeling; SystemVerilog; Real Number Modeling; Mixed Signal Simulation; Event-Driven simulator; Filter; Transfer Function; Laplace and Z-Transform.

I. INTRODUCTION

Modeling is an important discussion in several key areas. A relevant example is architectural exploration, where running many simulations in a short time for the proof of concept of more ideas is fundamental. In fact, implementations of innovative ideas are a consequence of trying as many solutions as possible within an allowed time. Adopting models which allow short simulation loops is a key enabler of innovation. Models can push verification earlier as they can be made available before the design, and can originate from specifications or even just from ideas. This allows bugs to be identified and removed from the design when bug-fixing is cheap. Tests can be developed early and qualified even if blocks are not yet being designed or being finalized. In mixed-signal designs, models (particularly RNM) represent a strong glue element between the digital design teams and the analog teams as they can be simulated in both mixed-signal, and pure event-driven, simulators. This enables sharing of the understanding of the system behavior and an increased confidence in the product.

Not to forget that models always represent an abstraction. They consider only a subset of the features of the real circuit. In most of the cases, focus is put on modeling functionality rather than performance. When focusing on functionality, we are less interested in the details of the implementation of the circuit. With this in mind, what matters is what the circuit does, and not how the circuit does it. This is an important principle, and if adopted, allows for significant reduction of the modeling effort. In order to model the functionality of a circuit, it is required to identify the targeted input/outputs and a good mathematical description of the relations between those inputs and outputs. In some cases, this relation is not obvious but it can be obtained by means of simulation. Simulation (like AC analysis) can provide, for example, the frequency (gain and phase) response of a circuit by characterizing its behavior.

By knowing where poles and zeros are, we can derive the transfer function which represents the wanted mathematical relation between the input and the output. The technique described in this paper is about modeling circuits for which the transfer function is known or can be derived in a more or less easy way.

The mathematics used are not new; the "novelty" is in the way the generation of output samples has been reduced to achieve better simulation performance. The paper also aims to simplify the description of the technique, highlight some of the "gotchas" not described in other papers (see **References**), and illustrate the concept behind using a very simple example.

II. DESCRIPTION OF THE TECHNIQUE

For almost every analog circuit, it is possible to determine an interval where its behavior is linear, or can be approximated as linear. Within this interval, the behavior of the circuit can be described from a transfer function which is represented by the following generic equation:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b0+b_1s^1+....+b_ms^m}{a0+a_1s^1+\cdots+a_ns^n} \qquad \text{Eq.1}$$

Y(s) is the output and X(s) is the input. Based on what described above, inversion of the above equation yields the following function in the time domain:

$$y(t) = L^{-1}\{Y(s)\} = L^{-1}\{H(s)X(s)\} = h(t) * x(t) = \int_{-\infty}^{+\infty} h(t)x(t-\tau)d\tau \qquad \text{Eq.2}$$

The tasks required can then be summarized into three main steps:

1. Inversion of H(s).
2. Inversion of X(s).
3. Perform in the time domain, the convolution integral.

While inversion of H(s) in *h(t)* is the easiest part, and for complex expression can be done by means of partial fraction decomposition, the other two remain a real issue. First of all, inversion of the input X(s) is not trivial as it means that the input waveform is known a priori. This might be possible in few cases but in general *x(t)* is not known. In addition, in SystemVerilog there are no operators which perform the integral, and we need to implement an integration function to perform the convolution integral.

We could for example, consider implementing the trapezoid rule which relies on sampling of the input argument at a certain frequency. This would be an easy method to implement but for very fast signals the sampling frequency of the input need to be high in order to keep the integration error low, and this could create problems with the simulation performance. It will take a longer time to simulate; therefore we need to consider another solution. A first simplification of the problems comes from the fact that, in an event driven simulator, a real value signal is continuous in amplitude but discrete in time.

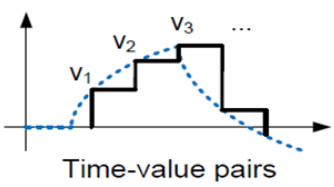

Figure 1: Representation of an analog signal in the discrete domain

Every change on an arbitrary input signal is a new step, with amplitude equal to the current value minus the previous value. So every input signal is then a series of step functions. Let *u(t)* be the unit step function. Our input sequence, as shown in Figure 1, can then be described as:

$$x(t) = V_1 u(t-t_1) + (V_2 - V_1)u(t-t_2) + (V_3 - V_2)u(t-t_3)$$
$$= \sum_{i=t_0}^{T} (V_{t_i} - V_{t_{i-1}})u(t - t_i) \qquad \text{Eq.3}$$

The unknowns are the different instants of times t_i when the changes will happen, and what the amplitude is going to be. This is not very important to know if we do our analysis in between two changes of the input signal.

If the value of the signal is A in the interval, in the s-domain the function will always be A/s which we can call A*U(s)*, with *U(s)* being the Laplace transform of the step function *u(t)*. Our transfer function after the above consideration then becomes:

$$\mathrm{H(s)} = \frac{Y(s)}{AU(s)} = \frac{b0 + b_1 s^1 + b_n s^n}{\frac{A}{s}} \qquad \text{Eq.4}$$

In the s-domain, each output can then be represented from the function:

$$\mathrm{Y(s)} = AU(s)H(s) = \frac{A}{s}H(s) = AG(s) \qquad \text{Eq.5}$$

The time domain model, in the interval where the input is constant, can then be derived from inversion of the previous expressions:

$$\mathrm{y(t)} = L^{-1}\{AG(s)\} = AL^{-1}\{G(s)\} = Ag(t) \qquad \text{Eq.6}$$

The only operations which are now required are the inversion of *G(s)*, and a simple multiplication. No knowledge of the input is needed upfront and there is no longer any need to perform the convolution integral. What is actually a limitation of the event-driven simulator becomes an advantage in simplifying the implementation of a time domain model.

G(s) = H(s)/s might be a complicated function to invert, but the partial fraction decomposition technique which translates H(s)/s in a sum of simple fractions, allows inverse Laplace tables to be used to get the sum in the time domain. Several tools are also available to help get the transformation directly and lower this effort. Some of them are open source or free of charge.

The missing bit now is to discuss what happens when there is a change in the input signal. Since the new input is still going to be a constant, and the transfer function is a property of the circuit and will not change (for linear systems) we can, at the moment where the change happens, reset the model variables and see the change as a new initial condition. The model then continues to evolve in the new constant interval in the same manner. The example below, Figure 2, explains this in more details.

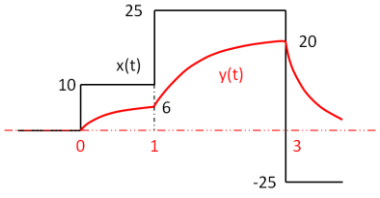

Figure 2: Generic description of input x(t) and output y(t)

If the fundamental time domain model is described from A*f(t)* with A the amount of change of the input, in the interval [0-1], the model equation will be 10*f(t)* as initially all was zero and input changed at time t=0 by an amount of 10. In the interval [1-3] the model equation will be 6 + (25-6)*f(t)*. 6 is the value of the model output a, *y(t)* at the time t=1 and (25-6) is the new delta input the model sees in its input. Finally at the time t=3 the model equation will be: 20 + [-25 – (+20)]*f(t)* = 20 – 45*f(t)*.

III. EXAMPLE OF A LOW PASS FILTER

The presented approach is generic and could be used for complex transfer functions. The example which follows is intentionally very simple so that the technique can be easily understood in all its steps. The related SystemVerilog code will be shown in Section IX.

Let us then consider the following simple RC low pass filter example and apply the presented theory.

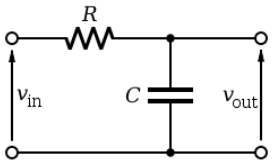

Figure 3: Simple Low Pass RC filter

The filter transfer function can be expressed as:

$$H(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{s + 1/RC} \quad \text{Eq.7}$$

The output of the filter at a certain instant in time can be obtained from:

$$Y(s) = \frac{1}{s + 1/RC} \quad \frac{A}{s} \quad \text{Eq.8}$$

The time functional model is obtained by inverting *Y(s)*:

$$y(t) = A(1 - e^{-t/RC}) + y(t_0) \quad \text{Eq.9}$$

$y(t_0)$ is the initial condition. The behavior of this filter, y(t), is shown in Figure 4:

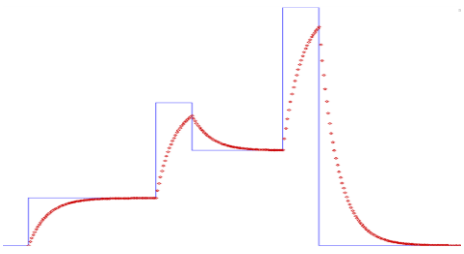

Figure 4: Filter response to multiple steps with different amplitude

At any input change (blue) the filter (red) reacts by recalculating initial conditions and then evolves naturally with the time constant of the system (1/RC). The output might get very close to 100% of the change if the input remains stable for long enough (this is the case between the first and the second change). For any input change happening at time t_i, the model saves the value of the output $y(t_i)$ and calculates the new value of the change which will be equal to $x(t_i)$ - $y(t_i)$.

The time of the model will be reset to 0 so that the model has a new start and evolves with different parameters:

$$y(t) = A'(1 - e^{-t/RC}) + y(t_i) \qquad \text{Eq.10}$$

Note that we are not resetting the simulator time only the time of the model. With this approach, there is no sampling of the input which this translates to better simulation performance. Given a certain initial condition, the system knows how it will evolve over the simulation time.

IV. GOTCHAS

The presented model calculates an output any time the input signal changes. This means, that in the case of a single step, at time $t_0 = 0$ and on a transition 0->1 of the input signal, the model calculates the output exactly at time t_0. If the initial conditions are $y(t_0) = 0$, the calculated output will be 0 (we cannot instantaneously change the voltage at the capacitor). Since there are no other changes of the input, the model will not update the output.

The downstream blocks, to which this model is eventually connected, will see a constant value although simulation time flows. This behavior is highlighted in Figure 5:

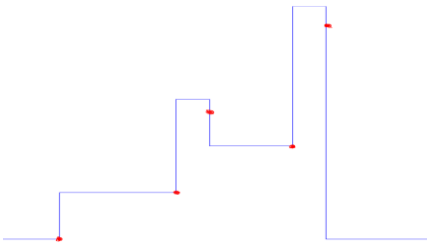

Figure 5: Filter output (red dots) in absence of an internal timer and input change

The red dots represent the output as the calculation is triggered from a change in the input. This might be a real issue, especially if the input does not change or it changes at a very low frequency. We need to show a changing output to the other blocks connected to the output of this filter.

To address this problem, an internal timer has been implemented in the model. Such a timer should serve the purpose of making the model recalculate the next output based on the natural response, and not the forced one due to the input change. We also need to consider how many recalculations are needed, ideally as few as possible to improve simulation performance. One possibility is to relate the number of points to the time constant of the system. We can do this by adding a parameter which specifies how many points per time constant are to be generated. In Figure 6 are examples of 4, 8, and 12 point per TAU (time constant).

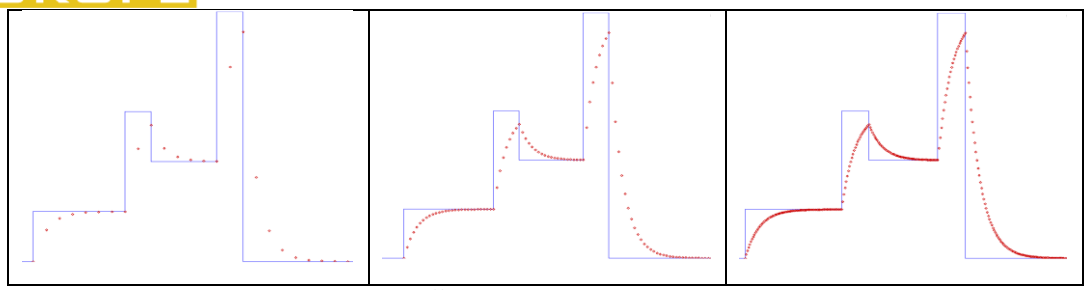

Figure 6: Filter output with different point generated in relation to the its time constant

V. COMPARING DIFFERENT TECHNIQUES

To give an idea of how good the technique is, we could compare it with another implementation of the filter based on the bilinear transform (z-domain filter). Figure 7 left, shows the behavior of the two different implementations and on the right comparison in mixed-signal simulation with analog filter.

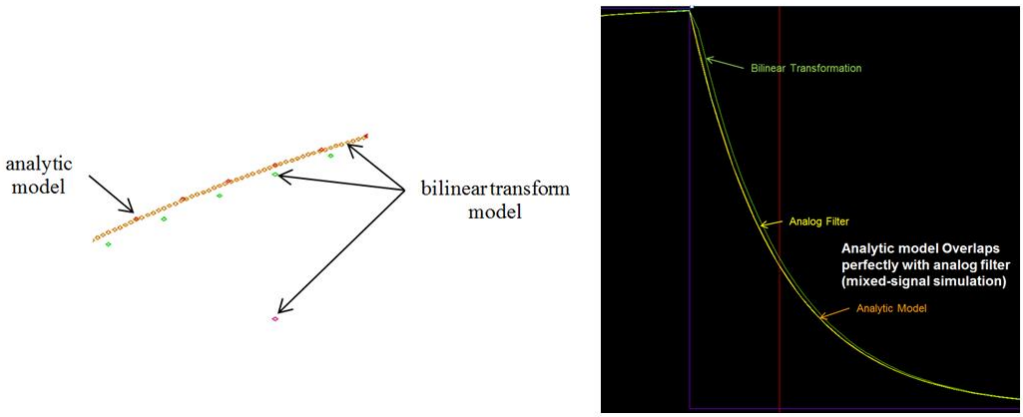

Figure 7: Comparison between different filter representations and with analog filter in mixed signal.

The red points are calculated with the proposed technique. The brown points are the output of the bilinear transformation method. To be close with the analytic model, the sampling frequency used in the bilinear transform is 10 MHz. As we relax the sampling frequency to improve the performance, we lose accuracy. The green points are coming from 1 MHz and the purple one from 100 KHz. This example makes it clear that the presented technique is far better for modeling purposes.

VI. OPTIMIZATION FOR PERFORMANCE

To further optimize, we could stop the timer once the difference between the output and the input is less than a certain defined value (say 1%). In the following simulation, the previous filters have been compared again to see what is happening when the error is below the specified value.

Figure 8: Absence of new event for the analytic filter in case of small error

There are no more generated points (red squares) once the output of the filter gets close to the value which the model would assume at time t=infinite (in practices very big number) by less than 1%. This is possible to calculate since we know the evolution of the model in time. However, using the bilinear transformation method,

the model continues to generate points penalizing the simulation performance. There is no reliable mechanism to stop generating points as we would have to stop sampling. The possibility of not generating events is another clear advantage of the proposed technique.

VII. EXTENDING TO MORE COMPLEX MODELS

It is very easy to extend this technique to other more complex transfer functions. We could have a template for such models where we would have only to change the time domain equation. Below is an example of a second order filter obtained from cascading the previous one and the relative plot overlaid with that of the 1st order filter:

$$y(t) = A\left(1 - e^{-\frac{t}{RC}} + \frac{t}{RC}e^{-\frac{t}{RC}}\right) + y(t_0)$$

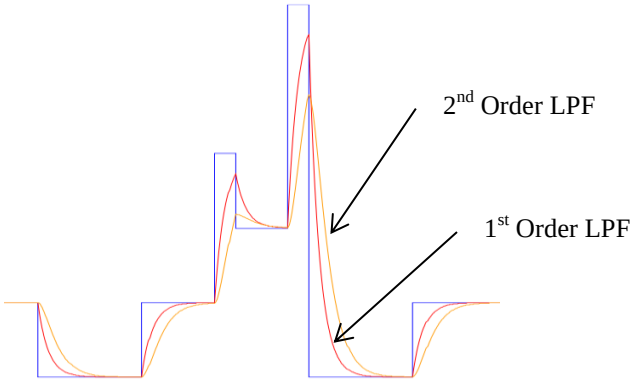

Figure 9: Behavior of 1st and 2nd order low pass

VIII. HIGH FREQUENCY INPUT

The following plot shows the behavior of this approach when a high frequency sinusoid is placed at the input. For simplicity we use the technique on the previous 1st order RC filter which has a cut-of frequency of $1/2\pi RC$.

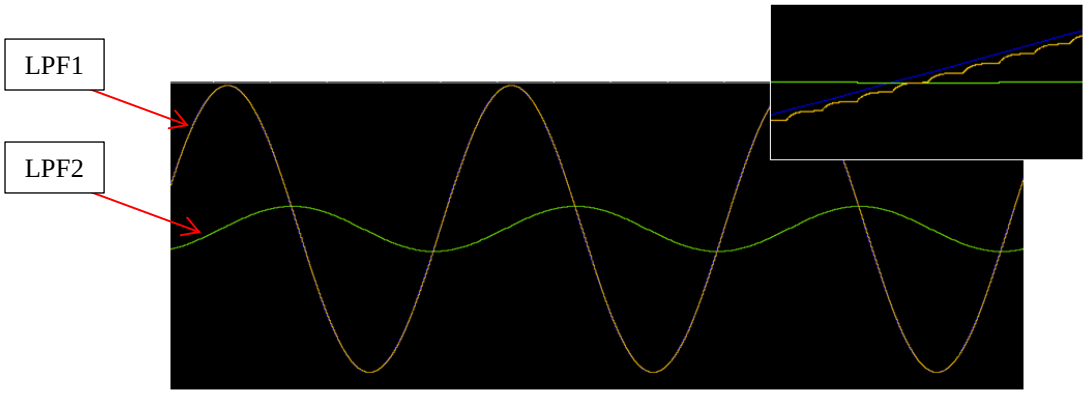

Figure 9: Attenuation of high frequency signals

In Fig. 9, two different filters are feed with the same sinusoid at 10 KHz. LPF1 has a cut-off frequency of 1.6MHz while LPF2 of 1.6 KHz. For LPF1 the input crosses the filter without being attenuated (input and

output are overlapped in the figure) while in the LPF2 there is attenuation. The technique works well also in presence of inputs with high rate of change.

IX. SystemVerilog Code

The following SystemVerilog code shows the part of the filter involved in the calculation which is generating the output:

```
always @(in or timer) begin
    // generate the first output
    if(!done) begin
        out = P__I; // initial value of the output
        din = in;
        done = 1;
    end
    else begin
        dt = $realtime - tlast; t = dt*1e-9; tnorm = t/TAU;
        // check the queue
        wait(inE.triggered | ckE.triggered);
        if(qu.size()) begin
            if(qu[0] == 1) begin    // input has changed
                if(P__N == 1)
                    out = ylast + din * (1 - exp(-tnorm));
                else if(P__N == 2)
                    out = ylast + din * (1 - exp(-tnorm) - tnorm * exp(-tnorm));

                // save values for the next calculations
                din = in - out;
                ylast = out;
                tlast = $realtime;
                element = qu.pop_back();
            end
            else if(qu[0] == -1) begin  // timer has ticked
                if(P__N == 1)
                    out = ylast + din * (1 - exp(-tnorm));
                else if(P__N == 2)
                    out = ylast + din * (1 - exp(-tnorm) - tnorm * exp(-tnorm));
                element = qu.pop_back();
            end
        end
    end
    qu = {}; // clear que in case of multiple events
end
```

X. Conclusions

In this paper, a simple way for modeling a generic transfer function has been described. Some of the "gotchas" were highlighted, and a way for improving the simulation performance has been demonstrated with examples. Simulation plots and code have been provided, and a comparative analysis with other techniques has been made, showing the superiority of this approach for modeling.

References

[1] K. Einwich, G. Krampl, R. Hoppenstock, P. Koutsandreas, S.Sattler. (1999, March) A multi-level modeling approach rendering virtual test engineering (VTE) economically viable for highly complex telecom circuits. User Forum. Conf. on Design, Automation and Test in Europe. [Online]. Available: http://publications.eas.iis.fraunhofer.de/papers/1999/017/paper.pdf (*references*)

[2] K. Karnane and S. Balasubramanian. (2012) Solutions for mixed-signal SoC verification. [Online]. Available: http://www.cadence.com/rl/resources/white papers/ms soc verification wp.pdf

[3] H. Thibieroz, A. Khan, and D. Cronauer, "Extending digital verification techniques for mixed-signal SoCs with VCS AMS," Synopsys, White Paper, Sept 2014.

[4] J.-E. Jang, et al., "True event-driven simulation of analog/mixed-signal behaviors in SystemVerilog: a decision-feedback equalizing (DFE) receiver example," IEEE Custom Integrated Circuits Conf. (CICC), pp.1–4, Sep. 2012.

[5] R. Staszewski, C. Fernando, and P. Balsara, "Event-driven simulation and modeling of phase noise of an RF oscillator," Trans. Circuits and Systems I., vol. 52, no. 4, pp. 723–733, April 2005.

[6] S. Alt, M. Marek-Sadowska, and L. C. Wang, "Circuit partitioning for behavioral full chip simulation modeling of analog and mixed signal circuits," Int. J. Modeling and Optimization, vol. 4, no. 1, pp. 74–80,

[7] E. Jang, M. Park, and J. Kim, "An event-driven simulation methodology for integrated switching power supplies in SystemVerilog," in Proc. DAC, ser. Design Automation Conference, May 2013, pp. 1–7.