Integration, the VLSI Journal xxx (2018) 1–10

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Real-time emulation of block-based analog circuits on an FPGA

Philipp Tertel *, Lars Hedrich

Institute for Computer Science, Goethe-Universität Frankfurt a. M., Germany

A B S T R A C T

In order to provide a real-time emulation platform for analog signal processing circuits, we propose a block based approach with a constant worst case runtime. We evaluate an FPGA-based implementation of this approach by comparing its output for different test cases to a non-real-time SPICE simulation. The implementation runs at a sampling rate of 88.2 kHz and features roundtrip times as low as 0.096 ms (12 bit ADC) and 0.190 ms (16 bit ADC). A completely automated process is introduced for fitting model parameters of filters and various nonlinear blocks. The fitting process is evaluated based on 35 block examples with R^2 values ranging from 0.967 to 0.999. For complex filter structures we were able to replicate the frequency responses predicted by a SPICE AC analysis accurately. Furthermore we compare measured transient responses of the FPGA-based emulation with SPICE and discuss advantages and disadvantages of the approach.

1. Introduction

With verification being a major part in design flows for analog circuits, there is a huge demand for analog simulation. While current simulation techniques are able to produce highly accurate outputs, the time they need to produce the outputs can be impractical when there is a need for larger timescale simulations. Additionally, existing simulators are in general not fast enough to simulate in real-time which prevents their use for real-time applications and in-circuit emulation. Another challenge for real-time application is the latency introduced by using software simulations depending on I/O buffers. We propose an emulation/rapid prototyping technique for analog signal processing circuits that overcomes the mentioned first limitation by producing fixed calculation times for a time step thus enabling real-time usage. Also our technique runs on dedicated hardware and does not need I/O buffers of more than a single sample which results in a very small and fixed latency. This technique can be used to functionally verify designs early in the design flow and can be used alongside digital emulation for mixed signal circuits.

2. Related work

One way of simulating an analog circuit is to create a netlist of the circuit and then use a SPICE-like analog circuit simulator to do a transient simulation. However simulation speed of non-trivial circuits is usually well below real-time, rendering this approach impractical for simulation of long signals. Furthermore, in-circuit emulation of the circuit becomes impossible. There have been numerous efforts to make SPICE simulations faster. As these simulations are based on sparse matrix calculations a natural approach is to use hardware acceleration. For example it is possible to reduce the runtime of the LU factorization part of the SPICE algorithm by using a GPU accelerated algorithm [1,2]. However, the performance increase as indicated by the reported results ranges greatly for different circuits. In many cases the GPU implementations are still outperformed by multithreaded CPU implementations like NICSLU [3]. A similar approach is to use an FPGA to run a parallel SPICE implementation [4]. Here speedups of 2.4× (geometric mean) could be observed when comparing runtimes on a Virtex-6 LX760 with an Intel Core i7 965 running a serial solver. While there is also active research on faster factorization methods for CPUs [5], all of these approaches have in common that they cannot guarantee the convergence of the SPICE algorithm and therefore cannot deliver a real-time emulation environment for signal processing chains.

Another possible approach is to map the circuit to a field-programmable analog array (FPAA). As this can be done based on a high-level block-based Simulink description [6], and consequently results in a real-time emulation of the circuit, it seems like a natural solution for the task. However, even with new academic advances in the area of FPAAs [7,8] which reach 55 and 98 CABs (Configurable Analog Blocks) of one OTA functionality each, respectively, an inherent issue for the emulation of large block-based circuits is the signal-to-noise ratio, the relatively inflexible construction of analog blocks and the restricted number of analog functionality available. Both approaches need a lot of CABs to implement a filter. The latest [7] implements 6 bandpass filters with detection logic in the audio range. Commercial FPAAs [9] have up to 20 CABs each being able to implement one

* Corresponding author.
E-mail addresses: tertel@em.cs.uni-frankfurt.de (P. Tertel), hedrich@em.cs.uni-frankfurt.de (L. Hedrich).

https://doi.org/10.1016/j.vlsi.2018.01.008
Received 29 September 2017; Received in revised form 21 January 2018; Accepted 28 January 2018
Available online XXX

Please cite this article in press as: P. Tertel, L. Hedrich, Real-time emulation of block-based analog circuits on an FPGA, Integration, the VLSI Journal (2018), https://doi.org/10.1016/j.vlsi.2018.01.008

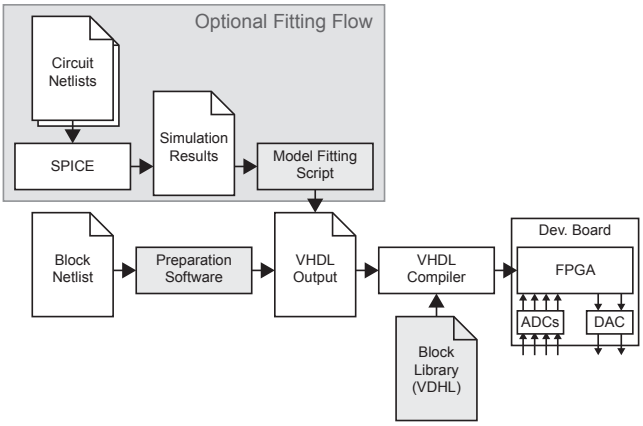

Fig. 1. Flowchart of the proposed emulation methodology.

biquad filter at most. Thus the number of filters that can be implemented is restricted. Some of these approaches have a more flexible set of analog functions including multipliers. The available maximum signal frequency differs depending on the FPAA.

Hardware Description Language (HDL) approaches [10,11] already focused on using VHDL simulators to evaluate high-level and block-level models of analog circuits. However the result was not intended to be real-time or synthesizable for use on an FPGA. They are only used to perform a simulation.

3. Emulation framework design

Analog signal processing chains can be thought of as interconnected blocks which either create signals (oscillators) or transform existing signals (for example filters, rectifiers). We use this block based approach for our emulation framework. As shown in Fig. 1 we use a netlist defining the block instances and their connections as input for a preparation software, which uses the structure defined by the netlist to construct its output. The block netlist is given in SPICE format, using subcircuit commands for each block, while the VHDL output links instances of corresponding simulation blocks. As the considered circuits are made up of blocks with common principle, it is possible to recreate the chain by instantiating implementations of configurable models stored in a library. In our approach we restrict us to directed analog signal processing with blocks having a dedicated input and output. This allows for easier mapping of the analog signal processing chain into digital hardware. Each emulation block can calculate its outputs as soon as its inputs are determined. If we can further ensure that each emulation block has a fixed maximum runtime to calculate a single time step, the calculation of an emulation block should be started when the emulations along the longest path (in terms of runtime) leading up to the block are done. Thus the triggering signals should be routed such that the sum of necessary clock cycles of the triggering path is the highest among all input paths of a block, as depicted in Fig. 2. In our implementation the routing of the trigger signals is done by the preparation software.

3.1. Block emulation models

The implemented block emulation models consist of:

- First order IIR filters
- FIR filters
- Piecewise quadratic functions
- Integrators

By default, voltages are passed between the blocks on 24 bit buses, using fixed point values that represent a range of about ±32 V. The data type used in all block implementations can be customized at compile time making it possible to trade precision for usage of adaptive logic modules (ALM) on the FPGA. An example of a use case where this is necessary is when the simulation becomes part of a mixed-signal emulation and the digital part of the system already occupies a large part of the FPGA's resources.

The IIR model, which is mainly used for emulating RC filters can be either instantiated with specific filter coefficients or with resistor and capacitor values, defining the cutoff frequency. In the latter case, the coefficients are calculated at compile time to fit the specified cutoff frequency.

The IIR filters internally use fixed point arithmetic and evaluate

$$V_{out} = a_1 V_{out} z^{-1} + b_1 V_{in} z^{-1} + b_0 V_{in} \tag{1}$$

where V_{out} and V_{in} are the output and input Voltages, a_1, b_0 and a_0 are the filter coefficients and z^{-1} is the discrete time operator for a unit delay. The computations are done in 2 cycles of the 50 MHz clock. This concept could easily be extended to filters with higher order.

In order to enable more flexible emulation of filters, a generic FIR emulation block was designed, that can implement a linear time invariant system with short impulse response. This is done by convolving the input signal and the impulse response. The implementation uses a buffer with length of the given impulse response to store intermediate results. In each 50 MHz cycle two values of the buffer are updated, which when running an emulation with a sampling rate of 88.2 kHz allows for impulse responses with a maximum length of 1128 samples. As the results necessary for the output are computed first, the implementation triggers the next block after 2 cycles of 50 MHz while continuing to process the rest of the intermediate results. While it would be possible to process more than two buffer entries at a time, and thus theoretically allowing for even longer impulse responses, it would not necessarily be possible to fit such long impulse responses, due to the number of required registers.

While the two aforementioned models show that filters can be fitted, there are many conceivable blocks which do not show the filter-like behavior. Examples are rectifiers and limiters. For a lot of non-filter blocks the output voltage can be described as a function of the input voltage. Often we could separate the linear dynamic from the nonlinear static behavior leading to Hammerstein-Wiener models [12]. E.g. an ideal full-wave rectifier without filtering shows an input/output relationship similar to $y = abs(x)$ function. However even if the output voltage is a function of the input voltage, the behavior of diodes and other components is more complex than the simple *abs*() example. In order to approximate such functions we try to determine appropriate regions in which the function can be described as a second order polynomial. Two examples for the result of such piecewise approximations are given in Fig. 3. These examples show that the success of the approximation is dependent on the behavior of the circuit. While the limiter seems to be faithfully modeled over the whole range of input voltages, the half wave rectifier shows ambiguity for very low positive input voltages. Such behaviors can not be recreated by this model alone. Furthermore the model is limited to second order polynomials in order to keep resource usage minimal. The HDL implementation only has to determine the appropriate set of coefficients by comparing the input voltage with the regions, and subsequently evaluate the polynomial. The coefficients are stored in combination with the maximum of the interval in which they are to be used. Even though the coefficients are stored ordered by this maximum value, the linear search to find the appropriate interval is not canceled as soon as the input voltage is higher than the currently visited maximum value. This is done in order to keep the amount of cycles constant which the block takes before triggering the following. Consequently the piecewise quadratic blocks take $k + 2$ cycles for processing, where k is the number of intervals.