Floating-Point Engines for the FPGA-Based Real-Time Simulation of Power Electronic Circuits

Tarek Ould Bachir, Christian Dufour, Jean-Pierre David, Jean Mahseredjian

***Abstract*—The real-time simulation of power electronic circuits is challenging for several reasons. A PC-based simulation can hardly achieve time-steps below 5-10 μs: this yields a limit on the maximal power electronic switching frequencies that can be accurately simulated using standard methods. This paper presents a design methodology for the hardware implementation of high-performance FPGA-based floating-point calculation engines aimed for the real-time simulation of power electronic systems. The power electronic circuits are modeled using the associated discrete circuit technique. A calculation time step of 100 ns is achieved for a boost converter, and the simulation results are validated against the SimPowerSystems library. The paper also discusses emerging paradigms for the FPGA-based floating-point computation that favor optimal performance and offer near double precision arithmetic at a minimal hardware cost.**

***Index Terms*—power electronic circuit, companion circuit model, real-time simulation, on-chip simulator, floating-point arithmetic, mac, FPGA.**

I. Introduction

As speed and density of modern high-end FPGA grow, on-chip real-time simulation of power electronic circuits (PECs) is gaining attractiveness and becoming a real alternative over pure PC-based real-time simulators. Hence, the use of FPGA devices for the purpose of real-time simulation is emerging as a dominant trend in the realm of hardware-in-the-loop simulation (HILS). HILS is an industrial testing practice in such fields as aerospace, automotive and power systems, whose main objective is to keep the development budget at bearable levels while running realistic tests on the actual prototyped hardware [1], [2]. Figure 1 presents a typical HILS configuration: A multi-core computer forms the base computing platform for the mathematical model and is connected to the physical hardware under test by means of filters or power amplifiers, depending on the type of HILS being conducted (Signal-HIL or Power-HIL). The computational power and the fast I/O coupling capabilities of the FPGAs make them an excellent choice for the extension of the processing potential of the simulator. An FPGA affixed to the simulator is accessible to the PC through a low-latency high-speed serial link and serves various purposes such as hardware ADC/DAC interfacing, function generation (sine, resolver, PWM), and the high fidelity simulation of the mathematical model [3]. The latter application is known as on-chip simulation and constitutes the main aim of this paper.

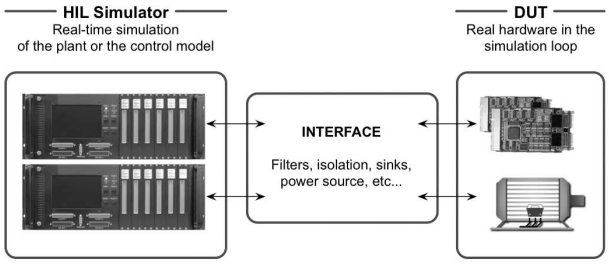

Fig. 1. Typical hardware-in-the-loop configuration.

On-chip simulation is an attractive option for the HILS because of a conjunction of factors such as the very low time-steps it can achieve and its ability to conduct a realistic real-time simulation of rapid PECs, which are frequent in aerospace, automotive and power systems. Indeed, modern general purpose processors (GPP) are reported to achieve no less than 5 μs time-steps in simulating systems of moderate size [2], and can not therefore simulate power electronic circuits using standard methods beyond a certain limit of switching frequencies (> 5 KHz) [4]. On the other hand, FPGA-based simulators attain time-steps in the 100-300 ns range [3], [4], [5], [6] and help overtake this barrier.

Nevertheless, FPGA-based on-chip simulation introduces a number of challenges that necessitate appropriate consideration. The designer must take care of every computational detail; that is, for a given problem, an application-specific processor (ASP) must be conceived from scratch. Thus, the computational regularity in problem formulation is a key feature for a hardware solver. Besides, in the context of FPGA-based real-time simulation, where a system clock is in the 100-200 MHz range, the time budged allowed to perform all the computations within a single time-step is limited. This fact can lead to situations where the pre-calculation of certain computationally-intensive parts of the mathematical model (as matrix inversion for instance) is mandatory [3].

The representation of real numbers is a major concern for on-chip simulation. Two main alternatives are acknowledged in the literature. The first alternative is the fixed-point (FXP) format, which is a machine representation for reals that uses

The author T.O.B. gratefully acknowledges the financial support of Opal-RT Technologies, the National Sciences and Engineering Research Council of Canada (NSERC) and the Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT).

T. Ould Bachir, J.-P. David, and J. Mahseredjian are with École Polytechnique de Montréal, Departement of Electrical Engineering, Montreal, PQ, H3T 1J4, Canada (e-mail: {tarek.ould-bachir, jpdavid, jeanm}@polymtl.ca).

C. Dufour is with Opal-RT Technologies, 1751 Richardson, suite 2525, Montreal, PQ, H3K 1G6, Canada (e-mail:christian.dufour@opal-rt.com).

Paper submitted to the International Conference on Power Systems Transients (IPST2011) in Delft, the Netherlands June 14-17, 2011

integers with a fixed number of digits after and before the radix point. FPGAs offer all the basic arithmetic operators needed for FXP arithmetic (adder, multiplier...), yet the FXP format suffers from a restricted dynamic range that may be cumbersome for the implementation of complex solvers. The other alternative is the floating-point (FP) format, which is a machine representation where a real x is represented by the triplet (s, e, m) such that $x = (-1)^s \cdot 2^e \cdot m$, where s is a sign bit, e the exponent and m the mantissa. The advantage of FP over FXP for modeling purposes is the one-to-one correspondence of all the physical quantities in the offline model and the real-time model. However, FP operators are more complex, occupy more area and have higher latencies than their FXP counterpart. Hence, while FP solves the dynamic range problem and permits the implementation of complex DSP algorithms, it may also bring a speed penalty.

Finally, PEC modeling is an important issue that must be carefully considered for the implementation of our solver. The formulation and solution of network equations fall into two categories: the state-space and the nodal analysis formulations.

The state-space formulation depends heavily on the circuit topology. The inclusion of N switching devices in the network usually implies 2^N different sets of equations [7]. Hence, in the context of FPGA-based PEC real-time simulation, the pre-calculation of these 2^N sets of equations requires a certain amount of on-chip memory that limits the number of switches to 1-10.

The nodal analysis or the modified-augmented-nodal analysis [8] found in todays Electromagnetic Transients Programs (EMTPs) are based on the following. All circuit components are discretized with an appropriate integration rule to form the main network matrix equations. A discretized model is also referred to as the companion circuit model in the literature. The classical nodal analysis uses an admittance matrix, whereas modified-augmented-nodal is based on a more generic A matrix which contains the admittance matrix. The main network equations are solved using sparse matrix techniques. If switching devices are represented using ideal switch models, then it is needed to refactorize the network matrix for each change of switch status. The same problem occurs if a high-low resistance model is used for switches.

The formulation of the associated discrete method proposed in [9], [10] has the advantage of maintaining the admittance matrix fixed irrespective of the switching states, which makes it a good candidate for FPGA implementation [4]. In this paper, we propose a general framework for the effective formulation of hardware FP engines for the real-time simulation of PEC, based on the the associated discrete circuit technique. The paper also discusses emerging paradigms for the FPGA-based FP computation that favor optimal performance and offer near double precision arithmetic at a minimal hardware cost.

The remainder of this paper is organized as follows: Section III proceeds with the presentation of the associated discrete circuit technique used for PEC modeling. It also details the design of the custom FP accumulator used in the FPGA-based PEC simulating engines. Section IV presents experimental results obtained for the FPGA-based simulation of a simple boost converter that help measure the effectiveness of four different engines developed with the proposed approach. Section V concludes this work and forecasts future developments.

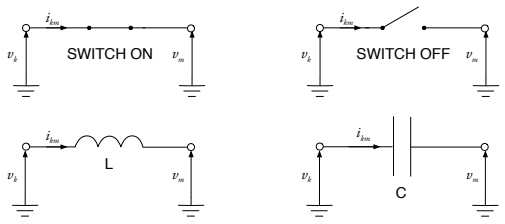

Fig. 2. Companion discrete circuits for the opened and the closed switch.

II. On-chip modeling

A. Associated discrete circuit technique for PEC modeling

The nodal equations are assembled after discretizing all circuit devices with a numerical integration rule such as the implicit trapezoidal method (ITM) or the backward Euler method (BEM). Under switching conditions, the ITM can cause numerical oscillations, thus making the BEM preferable for fast PECs modeling, even if it is less precise.

The companion circuit of a two-pole (k, m) device is a discrete Norton equivalent expressed in the form $i_{km}(t) = I_{\text{hist}}(t - \Delta t).G_{\text{eq}}(v_k(t) - v_m(t))$, where $I_{\text{hist}}(t - \Delta t)$ is a history term, G_{eq} is the equivalent conductance and Δt is the simulation time-step. When BEM is used, the equivalent conductance of the inductance (L) and capacitance (C) are respectively $G_l = \Delta t/L$ and $G_c = C/\Delta t$. We also have $I_{\text{hist}}(t - \Delta t) = -i_{km}(t - \Delta t)$ for the inductance and $I_{\text{hist}}(t - \Delta t) = G_c(v_k(t - \Delta t) - v_m(t - \Delta t)))$ for the capacitance. The resistance has no history term. When all circuit components are represented by a companion circuit, it is possible to write the system equations:

$$G_t v_t = i_t \tag{1}$$

where G_t is the admittance matrix (G_t is time-variant due to the topological changes caused by the changes in switching devices status), v_t is the vector of unknown nodal voltages and i_t is the vector of known current injections including history terms. The above system is solved at each simulation time-point after updating the vector with history current sources. Some history terms must be computed from nodal voltages. The associated discrete circuit technique used in [9], [10] origins from the Tableau approach [11] and solves simultaneously the voltage and currents terms by composing a larger set of system equations in the form:

$$H x_t = b_t, \tag{2}$$

where x_t is a composite vector of nodal voltages and branch currents, and b_t is a vector of known voltage and current sources. By modeling switching devices as a small inductance when the switch is closed, and as a small capacitance when it is opened, such that $\Delta t = \sqrt{LC}$, i.e. $G_c = G_l$ (Figure 2), H becomes time-invariant.

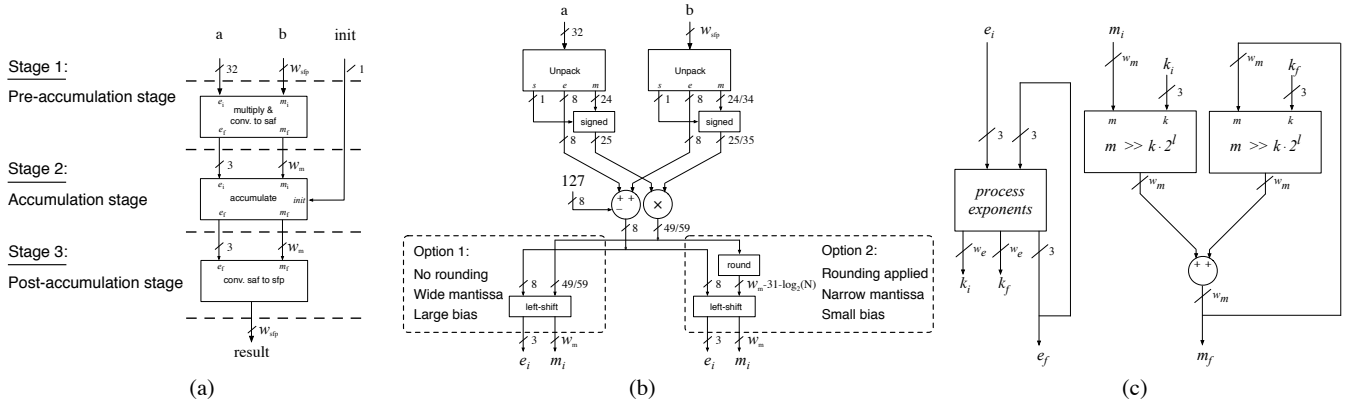

Fig. 3. Main blocs for the floating-point accumulator with design alternatives: (a) Main stages for the FP accumulator; (b) Pre-accumulation stage with the different options considered in the paper; (c) Accumulation stage datapath and control unit.

B. Custom floating-point MAC

A key observation in developing FP calculation engines for the simulation of PECs is the problem formulation in terms of a time-constrained matrix-vector multiplication, which can be efficiently solved by a structure of parallel multiply-accumulators (MACs). However, when considered in the framework of FP arithmetic, such an architecture presents a number of challenges: 1) An FP accumulator is more difficult to implement than its FXP counterpart because of the fundamentally more complex architecture of FP adders; 2) Commercially available FP adders may be considered to perform the accumulation function [3], [6] — however, the introduction of cumulative latencies has a bad impact on the overall simulation time-step; 3) The iterative variables must be fed-back into the solver as fast as possible in order to minimize dead-time cycles during which the MACs remain idle before starting a new iteration.

We propose a solution to this problem by designing a custom FP MAC using a so-called high-radix carry save (HRCS) format for the internal mantissa and by using the self-alignment technique (SAT) for the accumulation function [12]. Our approach guarantees ultra-low latencies, which is a fundamental criteria to achieve short calculation time-steps.

The SAT is a powerful algorithm for the accumulation of FP addends that minimizes the interaction between a new addend and the running sum by shifting its mantissa with respect to a common boundary to all summands rather than using the difference between the exponents of the incoming operand and that of the running sum. Hence, all the necessary shifts of the mantissa are performed outside the critical path of the accumulation loop. Figure 3.a presents the main stages of the MAC for single precision operands. At the first stage (Figure 3.b), the FP inputs a and b are unpacked, their exponents are added (the bias is substracted from the result) and their mantissas converted to two's complement and multiplied. The resulting mantissa contains additional least significant bits that can be discarded after rounding (truncation) to insure a higher precision. We show in Section III that the truncation yields area savings with a minor precision loss. At the accumulation loop stage, the incoming operands are accumulated through out the accumulation cycles. Finally, the post-accumulation stage converts the result to a standard floating-point format.

Single precision floating-point	Signed exponent	+3	1.34765625
	Biased mantissa	130	1.34765625
	Binary	10000010	1.010 1100 1000 0000 0000 0000
	IEEE standard	**0** (sign), **10000010** (exponent), **010 1100 1000 0000 0000 0000** (significand)	
Floating-point to SAF(5, 150)	Unpacked	10000010	0 1010 1100 1000 0000 0000 0000
	Exponent reduction	100 00010	0 1010 1100 1000 0000 0000 0000
	Mantissa extension	100 (shift_left(2))	010 1011 0010 0000 0000 0000 0000
	SAF(5, 150)	+4	0x02B20000

Fig. 4. The conversion of an SFP real number to SAF(5,150).

In order to clarify the mechanisms by which the MAC is operated, we define the self-alignment format (SAF) with two associated integral parameters: l, the amount of least significant bits discarded from the FP exponent, and b, the bias used to adjust the dynamic range of the FP value. A FP number x_{fp} is represented in SAF(l, b) by a pair of integral reduced exponent and integral extended mantissa (e, m): $x_{fp} = m \cdot 2^{2^l e - b}$. For instance, if $x_{fp} = 10.78125$ is given in the standard single precision floating-point (SFP) format, we may represent x_{fp} in SAF$(5, 150)$ as illustrates Figure 4. The binary encoding of x_{fp} imposes the insertion of a sign bit and the hidden 1 to form the two's complement standard mantissa. The standard exponent is divided in two parts, e[7:5] (which forms the new exponent for the SAF$(5, 150)$ representation) and e[4:0], which is used to shift the two's complement standard mantissa to the left. The mantissa is then sign-extended up to $w_m = 64$ bits. We thus get $x_{fp} = (4, \text{0x0000 0000 02B2 0000})_{\text{saf}(5,\,150)}$.

It is noteworthy that we arbitrarily set for this example $w_m = 64$ and $l = 5$ ($2^l = 32$). On the other hand, we chose $b = 23 + 127 = 150$ because 23 is the number of

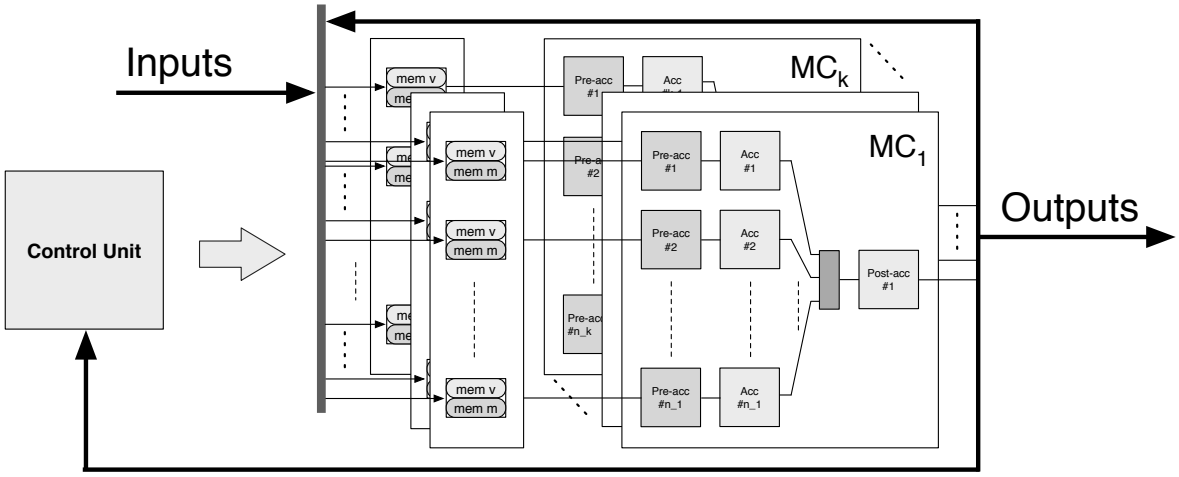

Fig. 5. Calculation engine for the simulation of PECS.

fractional bits in SFP and 127 is the standard SFP bias. The accumulation stage of the MAC adds the feed back operand and the incoming operand by using the SAT addition algorithm (see Figure 3.c): If $x_{fp} = (e_x, m_x)_{\text{saf}(l,\,b)}$ and $y_{fp} = (e_y, m_y)_{\text{saf}(l,\,b)}$, then $r_{fp} = x_{fp} + y_{fp} = (e_r, m_r)_{\text{saf}(l,\,b)}$, is given by $e_r = \max(e_x, e_y)$ and $m_r = \text{shift\_right}(m_x, k_x{\cdot}2^l) + \text{shift\_right}(m_y, k_y \cdot 2^l)$, with $k_x = e_r - e_x$ $k_y = e_r - e_y$. The correctness of this algorithm is guaranteed by the wideness of the mantissa m that is expected to be at least two times w_m^{std} bits wide, where w_m^{std} is standard FP mantissa width ($w_m^{\text{std}} = 24$ in single precision arithmetic).

C. Solver

Figure 5 presents the proposed calculation engine architecture for the simulation of PECS. The principal constituent blocks of the solver are the k MAC clusters (MC). Each MC_i ($1 \leq \text{i} \leq \text{k}$) disposes of n_i MACs. These MACs are connected to dedicated memory elements (registers/RAM/ROM) to alleviate the routing constraints and maximize throughput. The memory elements store either the iterative variables in mem_v (we clarify what the iterative variables are in Section III) and the inverted values from matrix (H^{-1}) in mem_m. In order to reduce the area occupation, the final stage of the MACs (Figure 3.a), i.e. the SAT to SFP conversion stage, is shared among n_i MACs in each MC_i, since it is active only at the end of the accumulation cycles. Finally, the control unit is in charge of the computation scheduling, input and output registering and updating the state of the switches, the latter explains the feedback path from the operative blocs (MAC clusters) to the control unit.

The minimal time-step for a MAC-based calculation engine (in terms of clock cycles) is given by the sum of the latency of the MAC (l_{mac}) and the number of columns in H^{-1} (q). The actual latency of the solver depends on l_{mac}, the number of rows (p) and columns (q) in H^{-1}, and the solver's processing power — expressed in n_i ($1 \leq \text{i} \leq \text{k}$) and k. The values of n_i ($1 \leq \text{i} \leq \text{k}$) and k are determined by a tradeoff between the calculation time-step that is sought and the size of the FPGA.

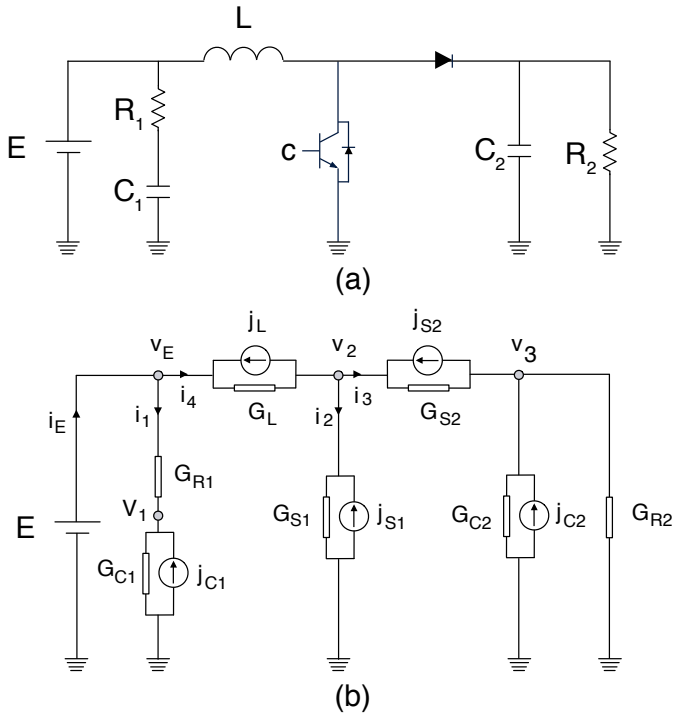

Fig. 6. Boost converter: (a) Original circuit; (b) Companion discrete model.

It is worth noting that Figure 3.b suggests two options for the conversion of the FP product of a and b into SAT(5, 150), namely with or without rounding. Section III shows that the rounding option offers considerable area saving at the cost of a slight precision penalty. Moreover, we propose to considerably improve the precision of the computation by exploiting spare hardware. Indeed, high-end FPGAs from Xilinx (latest Virtex devices) offer DSP blocs embedding asymmetric multipliers of (25×18) [13]. The 25×25 signed multiplication used for the SFP multiplier consumes two DSP blocks. However, 10 bits remain unused since the combination of two DSP blocks

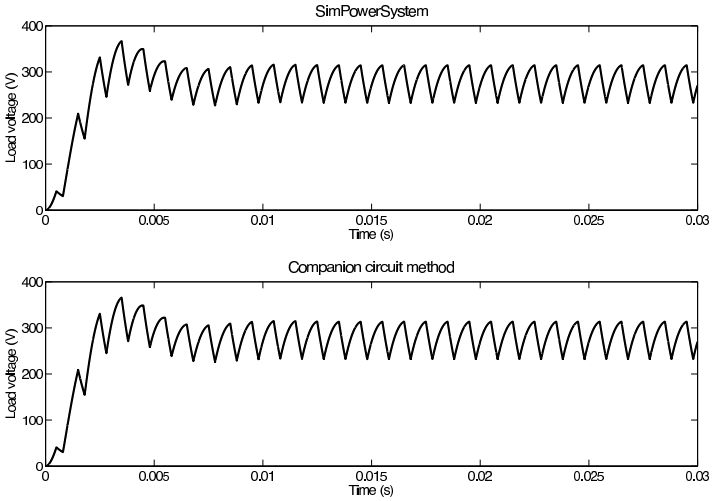

Fig. 7. Load voltage: Model against SimPowerSystem.

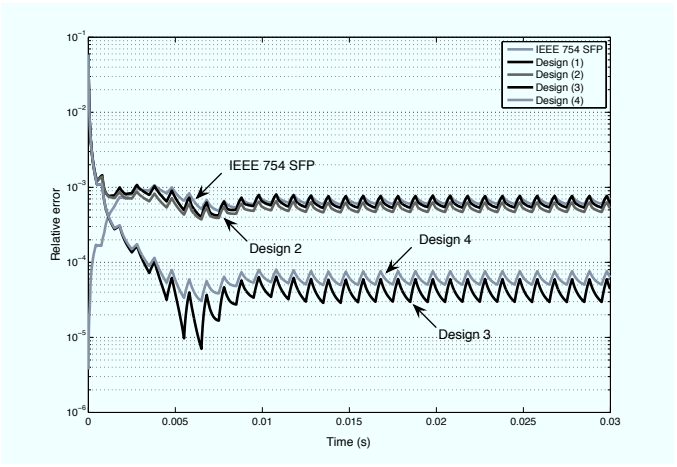

Fig. 8. Relative error for the four designs against double precision arithmetic. Relative error for standard single-precision computation is given as a reference.

forms a 25×35 signed multiplier. We thus propose to represent the iterative variables in a non-standard SFP format: we add the 10 spare bits to the mantissa.

III. Results

In this section, we propose to conduct a study on four different FPGA implementations calculation engines. We consider for that purpose the simple boost converter of Figure 6.a. This circuit is fairly comparable to the example proposed in [9], except for the shunt RC-branche. The associated discrete circuit technique models the boost circuit by the companion circuit of Figure 6.b for which we defined four nodal voltages (v_{E} is known) and four branch currents. The network solution $Hx^{n+1} = b^{n+1}$ for the boost converter is given by Equation (3). The iteration for each calculation time-point consists in repeatedly solving for b^{n+1} then solving for $x^{n+1} = H^{-1}b^{n+1}$. Because H is constant, the solver maintains the pre-calculated matrix H^{-1} rather than H. Moreover, since the first three terms in b are 0, we may consider H^{-1} to be a 9×6 matrix. Finally, from a data-flow point of view, the output's of the solver are selected from x, while the iterative variables are in b.

The history term of the switches are solved by:

$$j_s^{n+1} = \begin{cases} -i^n & \text{if the switch is closed} \\ G_s v^n & \text{if the switch is opened} \end{cases} \tag{4}$$

where G_s is the conductance associated to the switch. The updating rule for the switch state depends on the nature of the device. For the IGBT-diode pair, the current state of the switch is given by the boolean equation:

$$s^{n+1} = c^{n+1} + s^n(i^n \leq 0) + \overline{s^n}(v^n < 0) \tag{5}$$

where c^{n+1} is the current command at the IGBT gate, s^n is the switch state from the previous iteration, v^n and i^n are respectively the last computed voltage and current associated to the switch. On the other hand, the updating rule for the diode is given by the boolean equation:

$$s^{n+1} = s^n(i^n \geq 0) + \overline{s^n}(v^n \geq 0) \tag{6}$$

These state equations are computed by the control unit of Figure 5 because the controller has to decide whether j_s^{n+1} has to be updated to $-i^n$ or $G_s v^n$.

We implemented four different solvers for the considered boost problem:

- Design (1); The rounding (truncation) is performed prior to the accumulation ($w_m = 64$) and we use the standard SFP for the iterative variables ($w_m^{\text{std}} = 24$);
- Design (2); The rounding (truncation) is not performed prior to the accumulation ($w_m = 96$) and we use the standard SFP for the iterative variables ($w_m^{\text{std}} = 24$);

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 1 \\ G_{r_1} & -G_{r_1} & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & G_{c_1} & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & G_{r_2}+G_{c_2} & 0 & 0 & 0 & -1 & 0 \\ G_l & 0 & -G_l & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & G_{s_1} & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & G_{s_2} & -G_{s_2} & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} v_E^{n+1} \\ v_1^{n+1} \\ v_2^{n+1} \\ v_3^{n+1} \\ i_E^{n+1} \\ i_1^{n+1} \\ i_2^{n+1} \\ i_3^{n+1} \\ i_4^{n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ E \\ G_{c_1} v_1^n \\ G_{c_2} v_3^n \\ -i_4^n \\ j_{s_1}^{n+1} \\ j_{s_2}^{n+1} \end{bmatrix} \tag{3}$$

TABLE I
IMPLEMENTATION RESULTS OBTAINED FOR THE XILINX VIRTEX 5 FPGA (XC5VSX50T).

Metric	design (1)	design (2)	design (3)	design (4)	available
MAC operator (13 clock cycles latency)					
Number of Slices	372 (5 %)	617 (8 %)	685 (8 %)	464 (6 %)	8,160
Number of DSP blocks	2 (1 %)	2 (1 %)	2 (1 %)	2 (1 %)	288
Minimal combinatorial latency	4.792 ns	4.729 ns	4.938 ns	4.949 ns	N/A
Maximal clock frequency	208.68 MHz	211.46 MHz	202.51 MHz	202.06 MHz	N/A
Solver for the boost converter (100 ns time-step)					
Number of Slices	2,657 (33 %)	4,509 (55 %)	4,397 (54 %)	3,222 (39 %)	8,160
Number of DSP blocks	16 (6 %)	16 (6 %)	16 (6 %)	16 (6 %)	288
Number of BRAMs	7 (5 %)	7 (5 %)	7 (5 %)	7 (5 %)	132
Minimal combinatorial latency	4.969 ns	4.979 ns	4.973 ns	4.973 ns	N/A
Maximal clock frequency	201.25 MHz	200.84 MHz	201.09 MHz	201.09 MHz	N/A
Actual latency	5.000 ns	5.000 ns	5.000 ns	5.000 ns	N/A

- Design (3); The rounding (truncation) is not performed prior to the accumulation ($w_m = 96$) and we do not use the standard SFP for the iterative variables ($w_m^{\text{std}} = 34$);
- Design (4); The rounding (truncation) is performed prior to the accumulation ($w_m = 64$) and we use the standard SFP for the iterative variables ($w_m^{\text{std}} = 34$);

The four solvers necessitate 20 clock cycles to complete one computation cycle: 1 clock cycle is needed to address the registers, 6 clock cycles are needed to feed the MACs with the elements from given rows in H^{-1} (H^{-1} is $p \times q = 9 \times 6$), 13 additional clock cycles are needed to pass through the MACs ($l_{mac} = 13$). There is no need for other arithmetic units besides the MACs because the solver is dedicated towards the computation of the history terms (present in b^n) mainly. As one may observe, for a given b^n, one may first compute $x^n = H^{-1}b^n$ to estimate b^{n+1} for the next time-point. However, there is no real need for x^n to compute b^{n+1}: for instance $G_{c_1}v_1^{n+1}$ (the history term for C_1) can be obtained from $G_{c_1}(H^{-1}[2,:]b^n) = (G_{c_1}H^{-1}[2,:])b^n$. Hence, we should store $G_{c_1}H^{-1}[2,:]$ rather than $H^{-1}[2,:]$ in mem_m. From a more general perspective, we state that only linear combinations of rows from H^{-1} are to be pre-calculated and stored. The history terms in b^n are then used as iterative variables. It should be noted that the history term j_s for each switch necessitates the computation of two distinct history terms ($-i_l$ and $G_c v_s$). Hence, Equations 5 and 6 should be modified accordingly to exploit the available history terms rather than the current and voltage of the considered switch. Finally, it is noteworthy to mention that other variables may be computed during dead cycles of the MACs (the MACs are not processing any data during at least 14 cycles) as, for instance, the outputs of the solver.

The four designs were developed using the RT-XSG toolkit for MATLAB/Simulink from Opal-RT. Each solver disposes of four MCs (k = 4) with two MACs each ($n_1 = n_2 = 2$). With 8 MACs at hand, the solvers are capable of producing 16 different outputs within the 20 clock cycles budget. In our implementation, the four solvers were computing values from x^n for the outside world. The designs targeted an HIL simulator from Opal-RT, equipped with an ML-506 development board powered by the Virtex 5 XC5VSX50T FPGA from Xilinx. We used an Intel Core 2 Duo based Windows PC with 3 Go of RAM to synthesize and implement our architectures with the version 10.1 of the ISE software from Xilinx. The synthesizer was configured to automatically choose to use DSP blocks or not, and to balance the optimization between speed and area occupation. For each design, we imposed a timing constraint of 5 ns latency for the combinatorial logic. Table I outlines the area occupation and speed performance of the aforementioned designs. We clearly see that all the designs successfully met the timing constraint (5 ns) thanks to the HRCS format used for the internal mantissa, thus achieving the target calculation time-step of 100 ns.

The models were successfully validated against off-line SimPowerSystems models as illustrated in Figure 7. Figure 8 shows that our solvers perform better than what is usually expected from standard SFP arithmetic thanks to the use of a custom operator. We measured the significant impact on area occupation ($+65\%$) of avoiding truncation after the FP multiplication at the input stage of the MAC. However, we demonstrated that the rounding (truncation) operated after multiplication of the mantissas has less impact on the overall precision of the computation than the rounding performed at the output of the MAC (needed to match the standard SFP binary encoding). This result is obvious from Figure 8 which shows that the use of a non-standard floating-point format for the computation of the iterative variables enables very precise calculations, comparable to double precision arithmetic with a relative error in the 10^{-4}–10^{-5} range, This solution also leads to important area savings if rounding is performed at the input stage of the MAC (after multiplication) with a minor impact on precision loss.

IV. CONCLUSION

A new framework for the implementation of hardware floating-point engines has been presented and proved effective

for FPGA-based real-time simulation of PECs. The engines exploit custom floating-point MACs. Proper assessment of insightful trade offs between area and precision was proposed for such MACs. The framework guarantees very low latencies and high clock frequencies thanks to the HRCS format and the self-alignment technique. Time-steps well below 1 microsecond are achievable, as demonstrated by the successful FPGA-based implementations of a boost converter. Future work will consider more complex power electronic systems and the implementation of double precision floating-point calculation engines.

REFERENCES

[1] B. Lu, X. Wu, H. Figueroa, and A. Monti, "A low-cost real-time hardware-in-the-loop testing approach of power electronics controls," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 2, pp. 919–931, April 2007.

[2] J. Mahseredjian, V. Dinavahi, and J. Martinez, "Simulation tools for electromagnetic transients in power systems: Overview and challenges," *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1657–1669, July 2009.

[3] T. Ould Bachir, C. Dufour, J.-P. David, and J. Bélanger, "Effective FPGA-based electric motor modeling with floating-point cores," in *36th Annual Conference of IEEE Industrial Electronics Society (IECON 2010)*, Arizona, USA, Nov. 2010, pp. 829–834.

[4] M. Matar and R. Iravani, "FPGA implementation of the power electronic converter model for real-time simulation of electromagnetic transients," *IEEE Transactions on Power Delivery*, vol. 25, no. 2, pp. 852–860, april 2010.

[5] C. Dufour, H. Blanchette, and J. Belanger, "Very-high speed control of an FPGA-based finite-element-analysis permanent magnet synchronous virtual motor drive system," in *34th Annual Conference of IEEE Industrial Electronics Society (IECON 2008)*, Nov. 2008, pp. 2411–2416.

[6] J. Pimentel and H. Le-huy, "Hardware emulation for real-time power system simulation," in *IEEE International Symposium on Industrial Electronics*, vol. 2, July 2006, pp. 1560–1565.

[7] B. De Kelper, H. Blanchette, and L.-A. Dessaint, "Switching time model updating for the real-time simulation of power-electronic circuits and motor drives," *IEEE Transactions on Energy Conversion*, vol. 20, no. 1, pp. 181–186, march 2005.

[8] J. Mahseredjian and F. Alvarado, "Creating an electromagnetic transients program in matlab: Matemtp," *IEEE Transactions on Power Delivery*, vol. 12, no. 1, pp. 380–388, jan 1997.

[9] P. Pejovic and D. Maksimovic, "A method for fast time-domain simulation of networks with switches," *IEEE Transactions on Power Electronics*, vol. 9, no. 4, pp. 449–456, jul 1994.

[10] T. Maguire and J. Giesbrecht, "Small time-step (<2us) VSC model for the real time digital simulator," in *IPST*, June 2005.

[11] G. Hachtel, R. Brayton, and F. Gustavson, "The sparse tableau approach to network analysis and design," *Circuit Theory, IEEE Transactions on*, vol. 18, no. 1, pp. 101 – 113, Jan. 1971.

[12] T. Ould Bachir and J.-P. David, "Performing floating-point accumulation on a modern FPGA in single and double precision," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM 2010)*, 2-4 2010, pp. 1050–108.

[13] Xilinx, *UG192 v5.1: Virtex-5 FPGA User Guide*, 2009.