Hindawi
Mathematical Problems in Engineering
Volume 2017, Article ID 5274517, 14 pages
https://doi.org/10.1155/2017/5274517

Research Article

Implementation of SoC Based Real-Time Electromagnetic Transient Simulator

I. Herrera-Leandro,[1] P. Moreno-Villalobos,[1] S. Ortega-Cisneros,[1] Jorge Rivera,[2] and F. Sandoval-Ibarra[1]

[1] *Cinvestav, Unidad Guadalajara, Del Bosque Av. 1145, El Bajío, 45019 Zapopan, JAL, Mexico*
[2] *Cátedras Conacyt-Cinvestav, Unidad Guadalajara, Del Bosque Av. 1145, El Bajío, 45019 Zapopan, JAL, Mexico*

Correspondence should be addressed to S. Ortega-Cisneros; sortega@gdl.cinvestav.mx

Received 23 September 2016; Revised 23 January 2017; Accepted 29 January 2017; Published 8 March 2017

Academic Editor: Paolo Boscariol

Real-time electromagnetic transient simulators are important tools in the design stage of new control and protection systems for power systems. Real-time simulators are used to test and stress new devices under similar conditions that the device will deal with in a real network with the purpose of finding errors and bugs in the design. The computation of an electromagnetic transient is complex and computationally demanding, due to features such as the speed of the phenomenon, the size of the network, and the presence of time variant and nonlinear elements in the network. In this work, the development of a SoC based real-time and also offline electromagnetic transient simulator is presented. In the design, the required performance is met from two sides, (a) using a technique to split the power system into smaller subsystems, which allows parallelizing the algorithm, and (b) with specialized and parallel hardware designed to boost the solution flow. The results of this work have shown that for the proposed case studies, based on a balanced distribution of the node of subsystems, the proposed approach has decreased the total simulation time by up to 99 times compared with the classical approach running on a single high performance 32-bit embedded processor ARM-Cortex A9.

1. Introduction

Electromagnetic transients (EMT) in power systems [1] involve voltages and currents with very fast variations, which are produced by sudden changes in the power network and may last for only a few microseconds. Switch changes or short circuits are some of the causes that could trigger these phenomena; although they usually last only a few cycles, it is important to gather detailed information about their behavior, for example, for the design and evaluation of new control and protection devices to ensure satisfactory operation before they can be utilized in real systems. Real power systems are not accessible for testing purposes, so offline and real-time simulators represent the best alternative for testing equipment under stress conditions similar to those that they will encounter in real operating environments.

Developing real-time and even offline simulators is not a straightforward task, due to power system size and dynamic behavior of some elements, such as switches that cause modifications in the conductance matrix and nonlinear devices that make it a nonlinear problem. Modifications in the conductance matrix lead to a new matrix factorization, a time consuming process. Moreover, for real-time simulations, the complexity increases significantly when considering the limited time window usually available for processing data, generally on the order of microseconds.

Most proposals for the implementation of real-time simulators deal with the limited window of time available for computing the transient, avoiding the solution of the resulting linear system of equations that relate voltages and currents in the network [2–4], because this process is very time consuming and the computational complexity grows exponentially with the network size. Such approaches are based on prestoring the resulting inverse matrices for all possible combinations of the states of the time variant elements.

Since power electronics systems such as HVDC are widely used in power systems [5], precomputing and storing the corresponding inverse matrices in memory might not be a

suitable option. In general, prestoring the possible inverse matrices is acceptable for a power system with only a few time variant elements, but in the case of many time variant elements, the memory resources can be insufficient.

This work aims to describe the implementation of an alternative offline but also real-time electromagnetic transient simulator based on a System on Chip (SoC). The simulator was designed to keep an almost invariant simulation time step independently of the number of time variant devices in the network, as the simulator performs a complete solution of the system for every simulation time step.

The architecture of the proposed simulator is based on a highly parallel implementation designed to reach multiple levels of parallelism. For the simulation transient algorithm, a topological splitting network is proposed, a technique which permits reaching a deeper degree of fragmentation than that generated by only using the well-known splitting technique using transmission lines [6, 7].

2. Numerical Solution

Power systems are composed of several interconnected electric devices, like generators, transformers, transmission lines, and lumped devices, among others.

2.1. Lumped Elements. Digital simulators are based on synchronous devices, such as CPU, DSP, and other digital components, which all work based on clock signals that synchronize all internal electronics, and therefore their response is limited by the clock period of the system. As a result, digital simulators cannot reproduce the natural analog response of the power network element devices (voltages and currents).

Consider the relations at terminals of an inductor and a capacitor as written in (1) and (2), respectively:

$$v_l(t) = l\frac{di_l}{dt}, \tag{1}$$

$$i_c(t) = c\frac{dv_c}{dt}. \tag{2}$$

In digital simulations, differential equations of lumped components, inductors, and capacitors are transformed into difference equations in the form of (3). For this, most of the software and methods for electromagnetic transient simulation work are based on a numeric integration rule, as the trapezoidal integration method. This method is one of the most widely adopted methods by engineers due to its precision and stability [8].

The resulting difference equation can be taken to a circuit representation, as in Figure 1.

$$i_{km} = g * v_{km}(t) + h(t - \Delta t). \tag{3}$$

In (3), g is an equivalent conductance. This parameter depends on the value of the element that was discretized and the specific time step Δt chosen for the simulation; the term h is a history current source, whose magnitude depends on past values of voltages at element terminals, the current through the element, or both, depending on the specific integration method used [9].

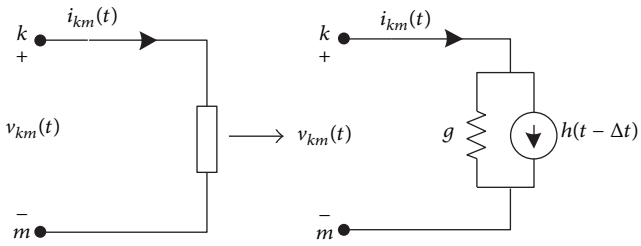

Figure 1: Discretized model of lumped devices.

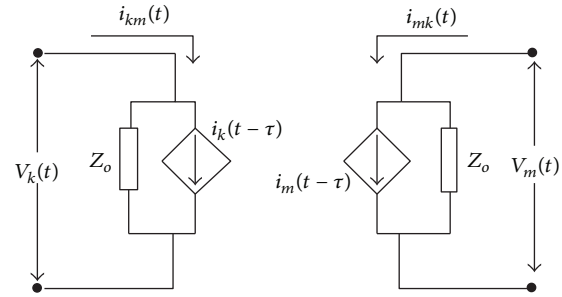

Figure 2: Lossless transmission line two-port equivalent.

2.2. Elements with Distributed Parameters. To include transmission lines in circuit simulators, the Bergeron model is adopted [9]. The equivalent discrete time circuit is shown in Figure 2; in this case, the line is represented with a two-port equivalent circuit, one port at each end of the line. In Figure 2 Z_o represents the characteristic impedance of the line and the history sources $i_k(t - \tau)$ and $i_m(t - \tau)$ are given by (4). τ is the travel time of the line.

$$\begin{aligned} i_m(t - \tau) &= -\frac{1}{Z_o}v_k(t - \tau) - i_{km}(t - \tau), \\ i_k(t - \tau) &= -\frac{1}{Z_o}v_m(t - \tau) - i_{mk}(t - \tau). \end{aligned} \tag{4}$$

2.3. Network Segmentation Based on Distributed Transmission Lines. Since history sources of distributed transmission lines are in terms of currents and voltages computed in the past and specifically a travel time of the line before, the model has the property of uncoupling the solution at both ends of the line, as can be seen in Figure 3 [10]. Most real-time simulators use transmission lines to break up the power system into smaller subsystems and distribute the computational burden among multiple computers or servers. The main limitation of this technique is that segmentation of the power system and distribution of the computation are limited by the presence, topology, and number of transmission lines in the network. On the other hand, it is also important to note that the performance of the simulator is restricted by the biggest subsystem resulting from the segmentation in terms of nodes.

Digital Electromagnetic Transient Simulators (DETS) work based on discrete models of the devices that comprise the network. The discrete models are in fact difference equations, which can be easily translated into equivalent resistive circuits, as shown in Figures 1 and 3. Then, the equivalent

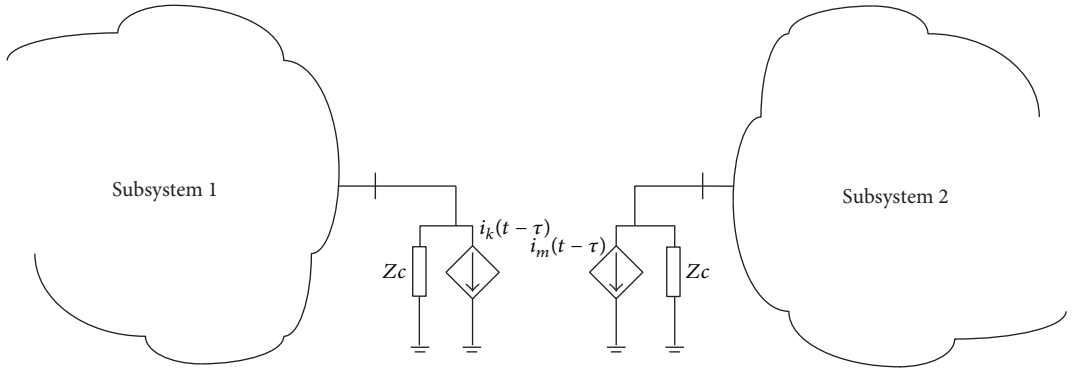

Figure 3: Transmission line between two areas.

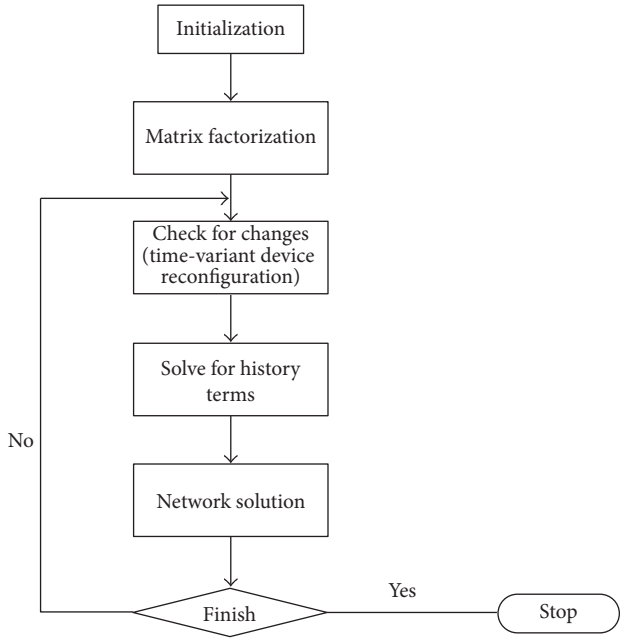

Figure 4: Basic flow diagram of an electromagnetic transient simulator.

circuits can be grouped using the nodal formulation (5) and (6) [11].

$$GV = I, \tag{5}$$

$$I = I(t) - I_{\text{hist}}. \tag{6}$$

Figure 4 depicts a simplified flow diagram for an electromagnetic transient simulator based on the ATP simulator.

3. Proposed Methodology for a Deeper Segmentation of Networks

Since real power systems can contain hundreds of nodes, carrying out network segmentation only by transmission lines can sometimes be insufficient for real-time simulations. Moreover, in networks which do not have transmission lines, distribution of computational burden is not possible. Therefore, an alternative technique for splitting the system is proposed in this work. This approach can be used together with transmission lines for a finer-degree of division or alone for system topologies without lines.

3.1. Network Segmentation Based on the Compensation Theorem. This technique is based on the compensation method theorem [12], originally used to exclude a nonlinear branch from the linear system; in this case, however, it is not intended to exclude a nonlinear branch from the conductance G matrix, but rather a section of the linear network, as illustrated in Figure 5. As a result, two or more subsystems with fewer nodes interconnected by branch currents can be obtained. Not only could this method lead to a reduction in the number of floating point operations for solving the linear system of equations, but also, and most importantly, it could enable a parallel solution flow for the electromagnetic transient solution.

It should be considered that the initial power system network in Figure 5 could be a section already isolated by distributed transmission lines, so the new segmentation in the figure could be the second level of division.

The method performs the solution in three simplified steps: First, the resulting subsystems are solved as if they were independent of each other; then, the branch current i_{km} and voltage v_{km} at the borders are calculated; finally, the effect of the branch currents are incorporated back into the partial solution of voltages obtained in the first step.

Consider an initial power system divided into two subnetworks interconnected at a pair of nodes k and m, as shown in Figure 6. To obtain the border magnitudes, v_{km} and i_{km}, the idea is that both networks see each other as a current injection. First, subsystem "a" sees subsystem "b" as a current branch (Figure 7), so it can be written as follows:

$$v_{km}(t) = v_{\text{tha}} - z_{\text{tha}} * i_{km}(t), \tag{7}$$

where v_{tha} and z_{tha} are the Thevenin equivalent voltage and impedance of section "a" seen from nodes k, m.

Now, a second equation is needed, which relates the variables i_{km} and v_{km}. For this, a reduced model of subnetwork "a" can be considered, as shown in Figure 8. This time subsystem "b" sees subsystem "a" as a current injection.

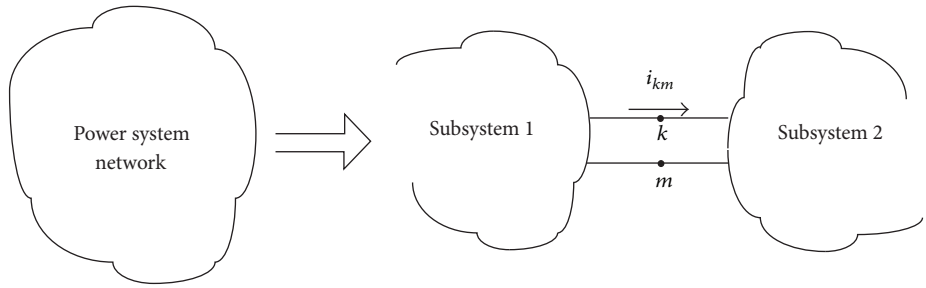

Figure 5: Segmentation of a network using a current branch.

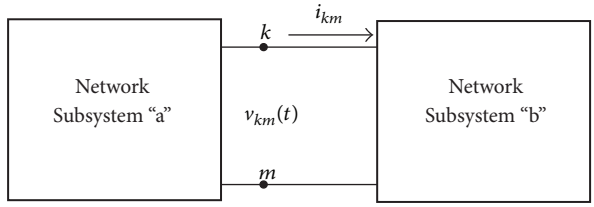

Figure 6: Network equivalent.

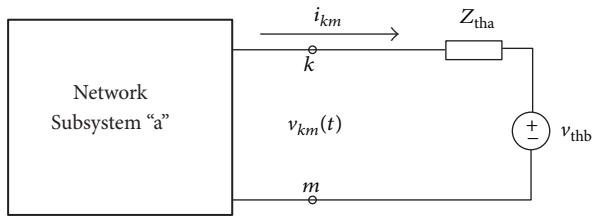

Figure 7: Subnetwork "b" as a Thevenin equivalent.

From Figure 8, the following equation can be found:

$$v_{km}(t) = v_{\text{thb}} - z_{\text{thb}} * i_{km}(t), \tag{8}$$

where v_{thb} and z_{thb} are the Thevenin equivalent voltage and impedance of section "b" seen from nodes k, m. Together, (7) and (8) can be used to solve for i_{km} and v_{km}.

3.2. Simplified Solution Algorithm. The algorithm of the solution can be summarized in the following steps:

(1) Compute the equivalent Thevenin impedance vector Z_{th} of subsystem "a" and subsystem "b"; this can be achieved using (5) and solving for vector V. In this case, vector I is used to inject a current into those nodes that interconnect both subsystems; this vector is filled only with those in the positions corresponding to the nodes that interconnect both systems and zeros for any other element.

(2) Using (5), find "a" and "b" subsystem voltages v_{th} as if they were independent from each other.

(3) Using (7) and (8), solve for the scalar i_{km}.

(4) Calculate the final voltage solution vector V incorporating the effects of i_{km} back into the partial solution

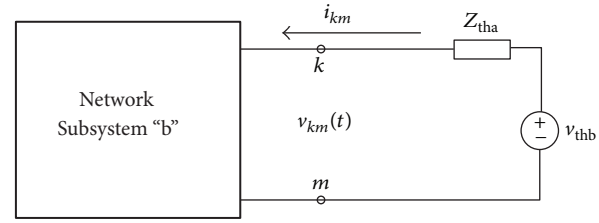

Figure 8: Subnetwork "a" as a Thevenin equivalent.

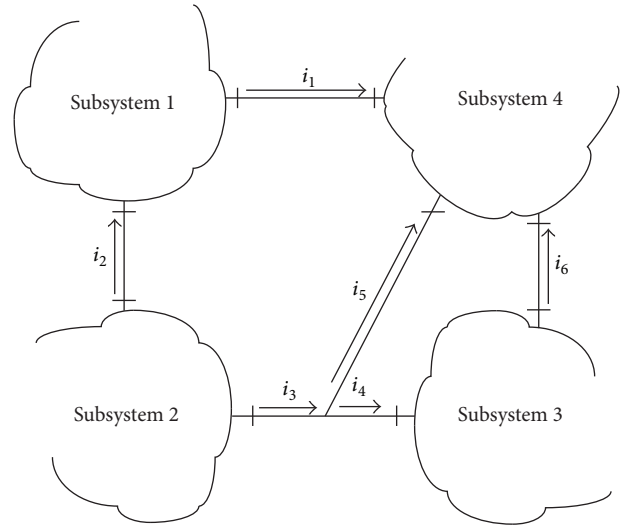

Figure 9: Network divided into multiple subsystems joined through current branches.

V_{th} using (9). This computation is made for both subsystems "a" and "b."

$$V = V_{\text{th}} - Z_{\text{th}} \cdot i_{km}. \tag{9}$$

3.3. Network Segmentation into Multiple Areas. The method for segmenting the system is not limited to splitting a network into only two sections; it can also be applied to dividing a network into several regions using current branches to interconnect the resulting subsystems, as illustrated in Figure 9. In the Figure, a power system is segmented into 4 subsystems connected by i_1, i_2, i_3, i_4, i_5, and i_6 branch currents.

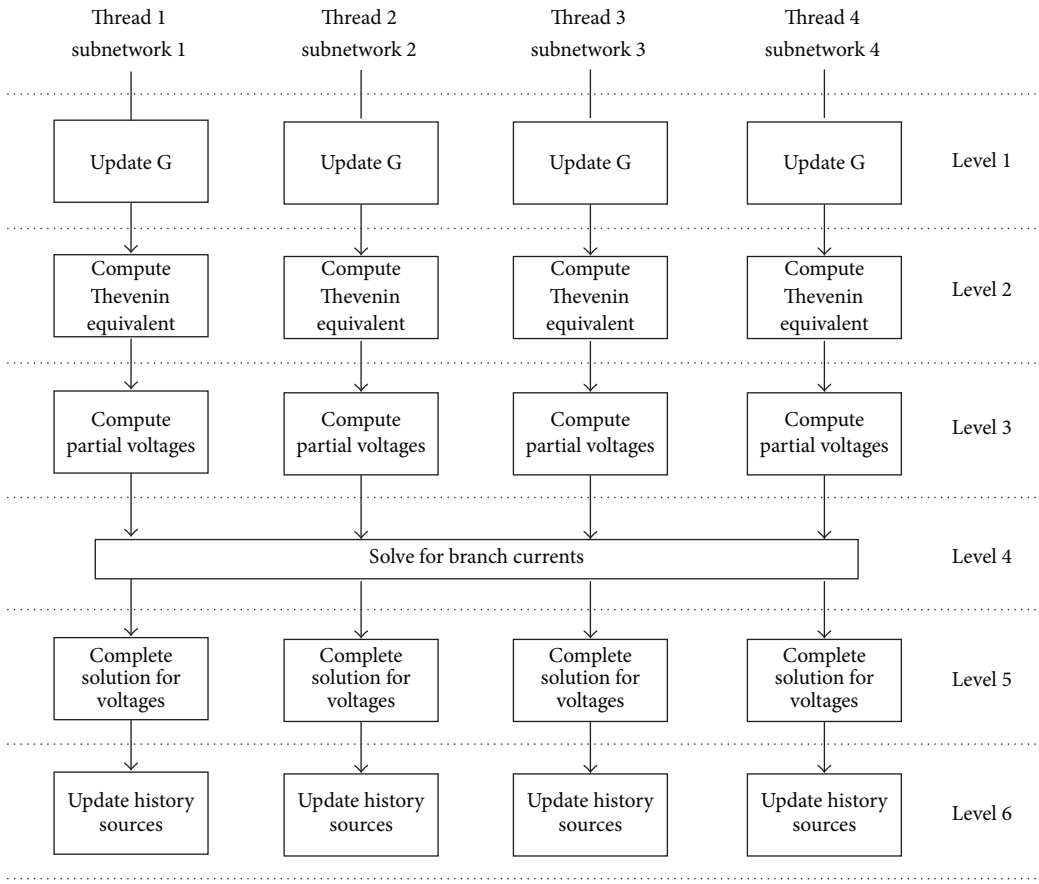

Figure 10: Simplified parallel scheme solution flow for a network divided into 4 subsystems.

The interconnection of multiple subnetworks increases the complexity of the solution, as now all the current branch contributions must be incorporated at the moment of updating the node voltages; in this case, (10) should be changed to the following expression.

$$V = V_{\text{th}} - Z_{\text{th1}} \cdot i_{km1} - Z_{\text{th2}} \cdot i_{km2} - \cdots - Z_{\text{th}n} \cdot i_{kmn}. \quad (10)$$

4. Parallel Solution Scheme

One of the characteristics of the proposal for segmenting the power system is that the method enables a parallel solution scheme, as shown in Figure 10. The diagram corresponds to the solution for a system divided into four areas or sections, as in Figure 9. This method uncouples the solution for each subnetwork generated, with the exception of the calculation of the currents at borders. This is a subsystem level of parallelism, since subsystems are solved in parallel.

It can be seen that the diagram in Figure 10 is composed of 6 sequential simplified levels, most of which are composed of four individual tasks—the same number as subsystems generated. Levels 2 to 5 correspond directly to the four steps described in the previous section, while levels 1 and 6 refer to the update of the conductance matrix G due to possible changes in the time variant elements and the update of the history sources of networks elements, respectively.

The method also allows us other degrees of parallelism. In fact, each individual task of the solution flow has many options for parallel processing. For example, levels 2 and 3 require solving a linear system of equations; even though those subsystems are reduced in size, this can be computationally intensive. In this case, there are parallel techniques that can be used to solve the linear system of equation. Other stages where parallelism can be exploited are on levels 5 and 6, where voltages and history terms are updated by the procedure. The update of history terms requires performing a computation of K independent equations, where K is the number of elements that generate a history term (inductances, capacitances, and transmission lines). In fact, most individual tasks have some degree of parallelism, even those on level 4.

5. Simulator Implementation

The real-time simulator is based on a hardware-software platform supported by a System on Chip (SoC), primarily composed of a hard ARM dual-core processor (software) and a Xilinx FPGA Virtex 7 (hardware). The simulator was designed to link hardware and software advantages. The software part is a user interface that receives the initial network interconnection data (netlist) and setup simulation parameters and handles the software-hardware synchronization. The

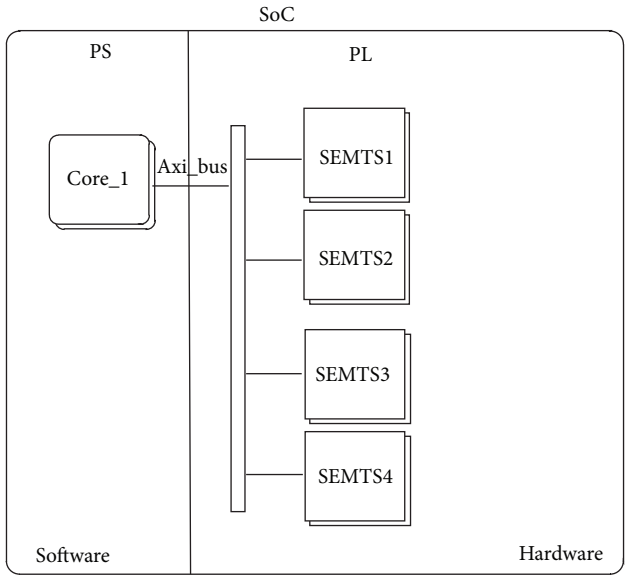

Figure 11: Electromagnetic transient simulator block diagram.

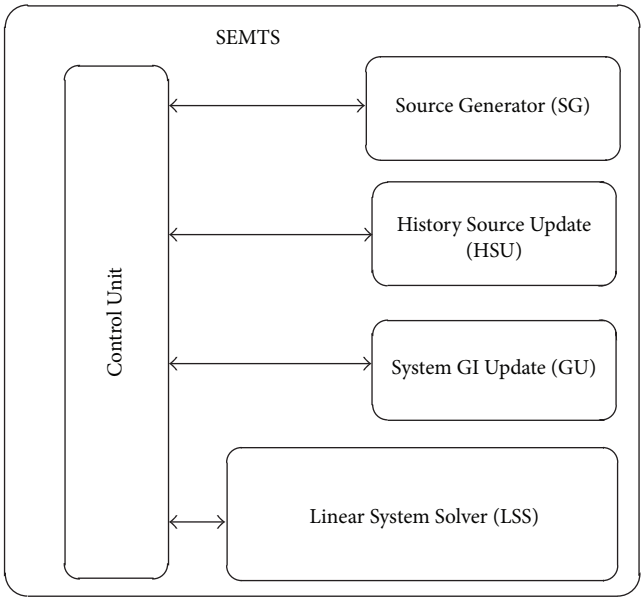

Figure 12: Small Electromagnetic Transient Simulator block diagram.

simulator is implemented almost totally in hardware to avoid spending time on communication between the processor and the FPGA.

The simulator implementation was designed using a custom architecture focused only on solving specific simulation tasks as soon as possible. Techniques such as parallelism and pipeline are widely used in order to comply with the limited time step usually used in electromagnetic transient simulations. The complexity of the simulator is not visible to the final user and remains behind the software layer, so knowledge of the hardware description level or hardware design is not necessary for performing simulations.

To perform the algorithm described in Section 3, using system equivalents and superposition of link currents the simulator implements multiple Small Electromagnetic Transient Simulator (SEMTS) blocks (Figure 11) at a hardware level; each of these modules is capable of solving a small power system. The internal configuration of the SEMTS blocks is shown in Figure 12.

A SEMTS module is composed mainly of 5 submodules.

(1) The Control Unit controls and synchronizes all internal processes inside the SEMTS.

(2) The Source Generator (SG) submodule, in charge of generating the sources waveforms: this module was designed to work based on lookup tables and interpolation of discrete points of the waveform.

(3) The History Source Update (HSU) module was designed to compute the history terms of network elements.

(4) The System GI Update (GU) submodule forms and manages the conductance matrix; changes in time variant elements are handled by this module.

(5) Parallel Linear System Solver (LSS) is designed to solve the linear system of equations.

5.1. Linear System Solver LSS. The heart of the simulator is the Linear System Solver (LSS) accelerator, shown in Figure 13. This module is a hardware version of the Gauss-Jordan elimination procedure, which has been parallelized and designed to rapidly solve the nodal system, almost independently of reconfigurations in the network.

The LSS unit is able to solve a linear system up to 26-by-26, so that any system lower than or equal to 26-by-26 can be solved without any hardware changes, where adjustments are handled in software. The LSS was designed to handle the matrix entity by rows. In the architecture, each row is stored in one individual BRAM memory inside a submodule Core_GJ; this approach allows simultaneous access to all rows of the linear system and also a parallel execution of the elimination procedure.

During the solution of the linear system, the LSS accelerator emulates the procedure presented below, which is also represented in the flow diagram of Figure 14.

Iteration 0

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1N-1} & a_{1N} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2N-1} & a_{2N} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3N-1} & a_{3N} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ a_{N-1\,1} & a_{N-1\,2} & a_{N-1\,3} & \cdots & a_{N-1\,N-1} & a_{N-1\,N} \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{N\,N-1} & a_{N,N} \end{bmatrix}$$

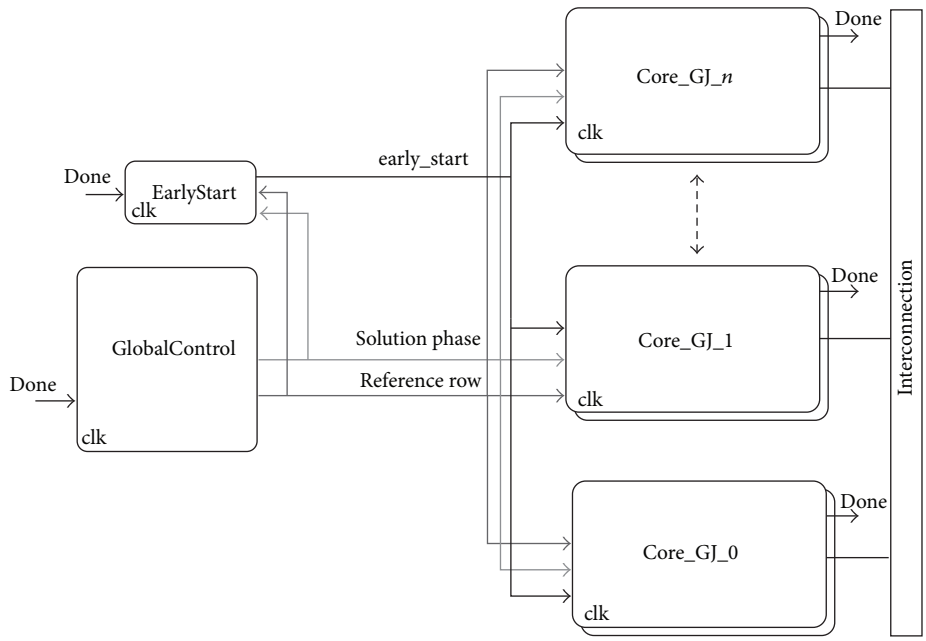

Figure 13: Internal structure of the LSS unit.

$$= \begin{bmatrix} \text{row } 1 \\ \text{row } 2 \\ \text{row } 3 \\ \vdots \\ \text{row } n-1 \\ \text{row } n \end{bmatrix}. \tag{11}$$

Iteration 1

$$A = \begin{bmatrix} 1 & a_{12} & a_{13} & \cdots & a_{1N-1} & a_{1N} \\ 0 & a_{22} & a_{23} & \cdots & a_{2N-1} & a_{2N} \\ 0 & a_{32} & a_{33} & \cdots & a_{3N-1} & a_{3N} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & a_{N-1\,2} & a_{N-1\,3} & \cdots & a_{N-1\,N-1} & a_{N-1\,N} \\ 0 & a_{N2} & a_{N3} & \cdots & a_{N\,N-1} & a_{N,N} \end{bmatrix}$$

$$= \begin{bmatrix} \text{row } 1 \\ \text{row } 2 \\ \text{row } 3 \\ \vdots \\ \text{row } n-1 \\ \text{row } n \end{bmatrix}. \tag{12}$$

Iteration 2

$$A = \begin{bmatrix} 1 & 0 & a_{13} & \cdots & a_{1N-1} & a_{1N} \\ 0 & 1 & a_{23} & \cdots & a_{2N-1} & a_{2N} \\ 0 & 0 & a_{33} & \cdots & a_{3N-1} & a_{3N} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & a_{N-1\,3} & \cdots & a_{N-1\,N-1} & a_{N-1\,N} \\ 0 & 0 & a_{N3} & \cdots & a_{N\,N-1} & a_{N,N} \end{bmatrix}$$

$$= \begin{bmatrix} \text{row } 1 \\ \text{row } 2 \\ \text{row } 3 \\ \vdots \\ \text{row } n-1 \\ \text{row } n \end{bmatrix}. \tag{13}$$

Iteration n

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} = \begin{bmatrix} \text{row } 1 \\ \text{row } 2 \\ \text{row } 3 \\ \vdots \\ \text{row } n-1 \\ \text{row } n \end{bmatrix}. \tag{14}$$

At each iteration, a variable of the system of equations is eliminated simultaneously on all rows by the LSS. The LSS module finds the solution of the system after N iterations.

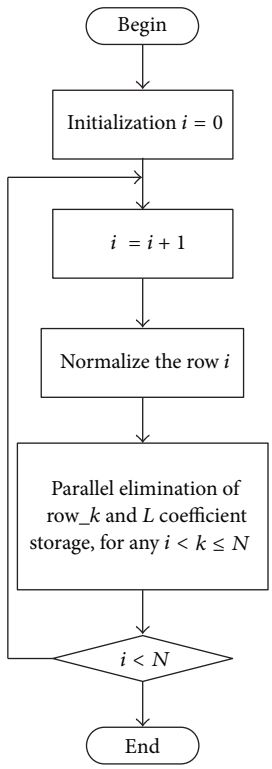

Figure 14: Elimination procedure flow diagram.

Basically, two procedures are carried out to factorize the matrix A, first, a normalization, and then, a parallel elimination, as follows:

(1) Normalization: in this step, a normalized row is calculated; for this, the ith row assigned to the ith Core_GJ is divided by the diagonal element (a_{ii}):

$$\text{normalized\_row} = \frac{\text{row\_}i}{a_{ii}}. \tag{15}$$

(2) Elimination: during the elimination process, the normalized row is used to eliminate the ith variable in the kth equation assigned to the kth Core_lu, for every k different than i.

$$\text{row}_k = \text{row}_k - a_{ki} * \text{normalize\_row}. \tag{16}$$

The LSS unit performs the parallel elimination using four different modules: Core_GJ, Global Control, Early Start, and the interconnection module.

5.1.1. Control Unit. The Control Unit is dedicated for generating the synchronization signals "solution_phase" and "reference_row"; together, these signals provide detailed information about the progress in the solution phase of the linear system. The outputs of the "Control Unit" are used by the Core_GJ modules to decode the current stage of the solution and the instruction to be executed; a Core_GJ responds with the "done" signal to indicate the end of the instruction execution. The Early_start module also depends on the Control Unit for synchronizing the Core_GJ operation. The "solution_phase" signal indicates the current solution phase, which can be initialization, elimination, or return phases. The "reference_row" signal provides the row in turn to be initialized, normalized, or returned.

In Figure 15, a block diagram of the module Control Unit is depicted. There are three different submodules inside: the "sequence controller," the "phase generator," and the "row generator." The submodules "phase generator" and "row generator" were designed to generate the sequence for the solution_phase and reference_row signals. Figure 16 illustrates the dynamic of these signals.

5.1.2. Early Start Unit. Synchronization between Core_GJ modules is not managed internally but rather by external modules called Early_start (Figure 17). EarlyStart modules generate a set of control signals as es_elimination, es_writeBackRE, and so forth that trigger internal Core_lu subprocesses, such as those that perform the elimination, or row update procedures, applicable to their assigned matrix rows. The Early_start module allows Core_lu to know in advance the arrival of the processed data; that is, during the factorization phase this implementation allows Core_lu modules to start their internal elimination processes some clock cycles before the normalized row arrives to them; in this way, while one Core_lu is normalizing its internal row, performing equation (15), other Core_lu can start to fill in their internal pipelines in order to perform the elimination procedure, solving (16).

In the same way, the Early_start module is used to anticipate the arrival of the eliminated row and prepare the lower part of the Core_GJ to save the updated row, as illustrated in Figure 18. First, at time $t1$, the Early_start enables the Core_GJ modules to begin the elimination procedure; then, at time $t2$ and k clock cycles before the elimination procedure is performed, the signal es_lu_writeBack_RE is asserted, indicating to the Core_GJ the start of the process to update its internal row with the recently eliminated row.

The Early_start module can predict and synchronize Core_lu modules based on the latency of the instruction being executed and that of the following instruction; for this, the internal subblock "sequence controller" keeps a constant memory table with the number of clock cycles that must wait before triggering the next subprocess; the subblock "latency cyc counter" is basically a clock cycle counter, which is compared with the constants table in order to determine the exact clock cycle to begin the next subprocess in the solution flow to proceed in synchrony.

5.1.3. Core_GJ. Core_GJ modules are the most complex hardware blocks in the design and consume most of the hardware of the LSS unit. There are 26 Core_GJ blocks in every LSS unit. During the instantiation or automatic generation of the Core_GJ arrangement, each Core_GJ receives a unique ID to differentiate it from the others and indicate its relative position. This ID also identifies the assigned row to the Core_GJ, ranging from 1 to 26 for the last Core_lu in the

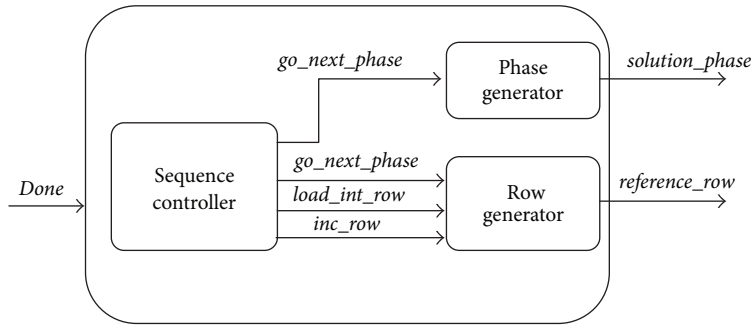

Figure 15: Control Unit block diagram.

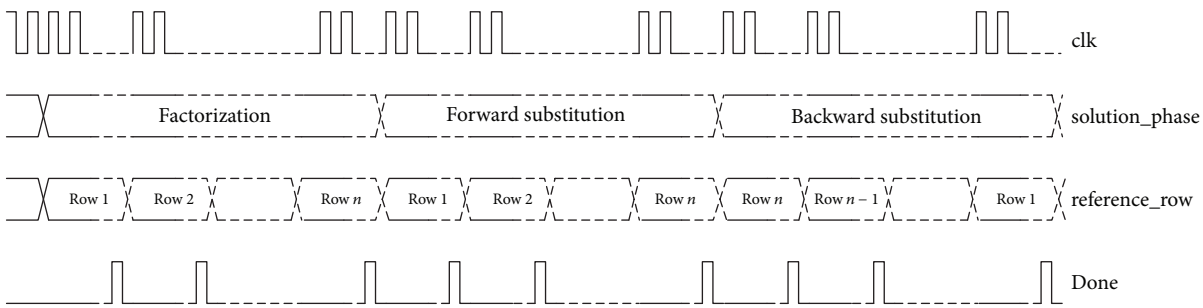

Figure 16: Dynamic of signal solution_phase and reference_row.

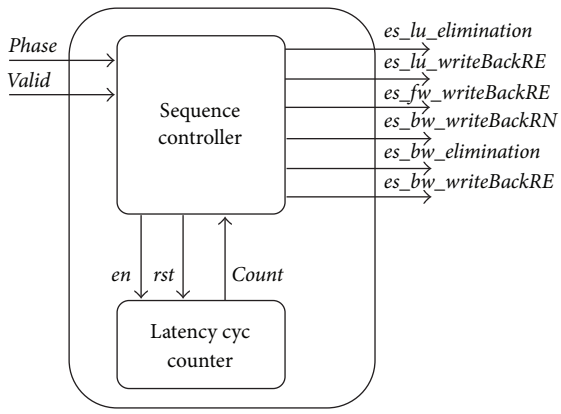

Figure 17: Early Start block diagram.

arrangement. A block diagram with the main components of a Core_lu is shown in Figure 19.

Core_lu accelerators are designed to work in matrix rows (vectors), which reside in BRAM memories; their instruction set supports only a few basic instructions designed mainly to compute (16).

Some supported instructions are as follows.

(1) Move a scalar from memory RAM to a register (num, row, fac, or den).

(2) Multiply/divide the vector by a register.

(3) Add/subtract two vectors.

(4) Update the vector.

(5) Return the vector.

For high performance, the design has a real dual-port BRAM memory, which supports two simultaneous readings or one reading and one writing. Core_GJ also contains double control logic to support the real dual-port memory. Control logic can be divided into two sections: the lower and the upper part; the upper part handles the read-only port of the ram, while the lower control section is assigned to the read-write port. During operation, both sections work simultaneously when possible. A brief description of the internal subblocks of the Core_lu accelerator is given below.

The "Position Decoder" subblock generates the relative position of the core using the input reference row drive by Global_controlas as reference; the output of this block can be equal or different when the equal signal is asserted; this indicates that the ID of the Core_lu is equal to the reference_row; contrariwise the equal signal is not asserted; it means that the ID of the Core_GJ does not correspond to the input reference_row. The relative position is useful for decoding the instruction to be performed by the Core_lu according to the stage of the solution.

The internal block called the "Control Unit" decodes the instruction to be executed based on the input solution_phase and the relative position generated by the "Position Decoder," but the "Vector Instruction Generator" subblock is the real block which generates the synchronized signals to execute the vector instruction.

There are four registers in the diagram block: "num," "row," "fac," and "den"; these registers are temporary locations used together with the "Vector Arithmetic Unit" to perform lightly more elaborate operations on the rows; for example, to execute (16), the term a_{ki} is first saved in the

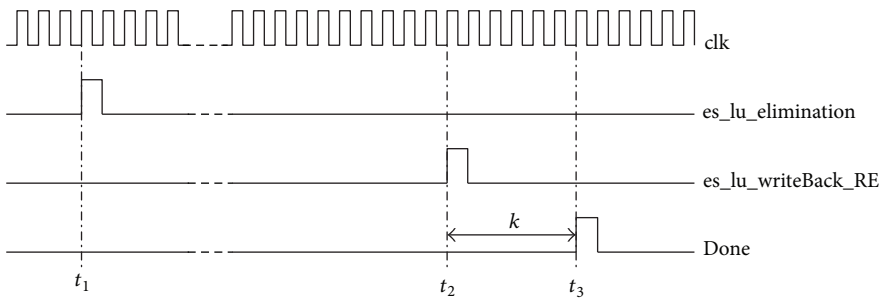

Figure 18: Dynamic of signals es_lu_elimination and es_lu_writeBack_RE.

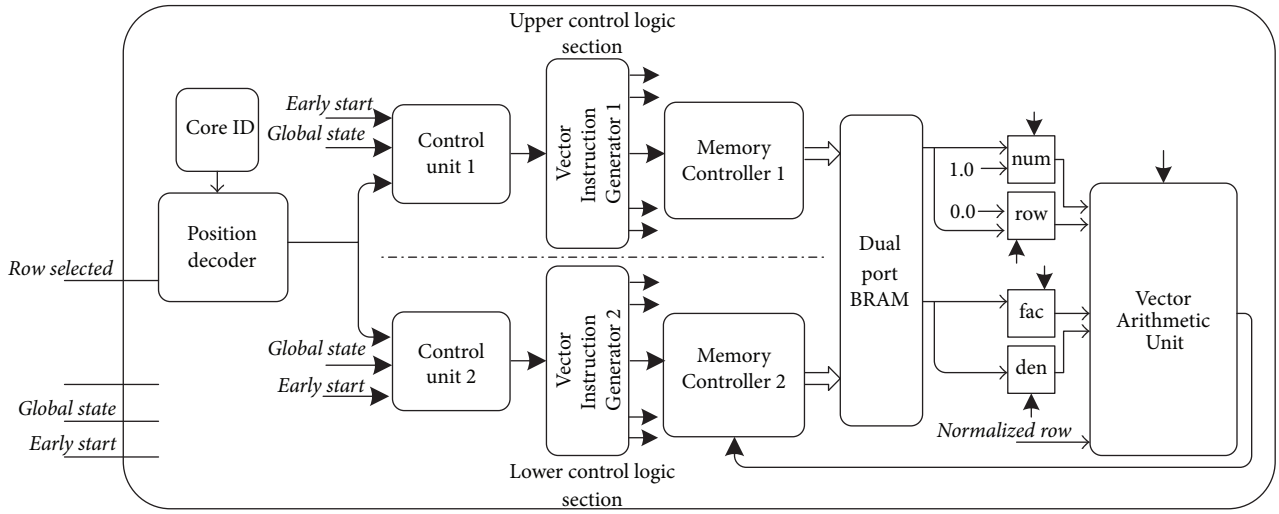

Figure 19: Block diagram of the Core_lu module.

register "fac," and then the normalized vector is multiplied by this factor and added to the internal row of the Core_lu, which is recorded by the register "row."

"Vector Arithmetic Unit" is an arithmetic floating point unit capable of performing a couple of instructions, such as dividing or multiplying a vector by a scalar, adding or subtracting two vectors, and other simple operations.

Finally, the "Vector Instruction Generator" module is the block which generates the required sequence of control signals for the memory controller, the registers, and the Vector Arithmetic Unit in order to execute the instruction in turn.

6. Results

In this chapter, simulation results and measurements of the simulator's performance are presented, where two hypothetical case studies were used for this purpose. All simulations were performed twice, first using only the PS side of the ZYNQ SoC in order to obtain a reference for comparison purposes and a second time using both PS and PL implementing the proposed algorithm. In all cases, PS works at a frequency of 800 MHz, while the logic in PL works at a frequency of 250 MHz, the highest rate directly allowed by PS. In these simulations, the worst case was considered, meaning that a complete reconfiguration of the system was performed.

6.1. Study Case I. The first system to consider is shown in Figure 20. In case, the system under test is a power network with 3 short transmission lines modeled using LC sections; losses are considered zero but they could be included in the system in every LC section. The switch at load 1 is used to simulate a fault at a time of 2.5 mS. The required simulation time step is 20 μS.

The first simulation was carried out using the classical approach based on the algorithm of the commercial ATP program, solving the complete network without any partition technique or hardware implementation. The algorithm was compiled using the highest optimization level. After the first run, the timer measured a maximum total time of 449 μs per simulation step, almost 22 times the required 20 μs. For the second run, the network was partitioned as in Figure 21 and distributed in PL as in Figure 22 using the proposed simulator and partition network technique.

The maximum time step captured by timers in PS was 11.0 μS, which includes the time spent for both software and hardware. Table 1 summarizes the time measured by timers in PL for all hardware blocks.

For this specific case, the maximum time step was reduced from 449 μS to 11.6 μS, an improvement of almost 39 times over the first run. Figure 23 shows the resulting voltage waveform at load 1 yielded by the simulator compared

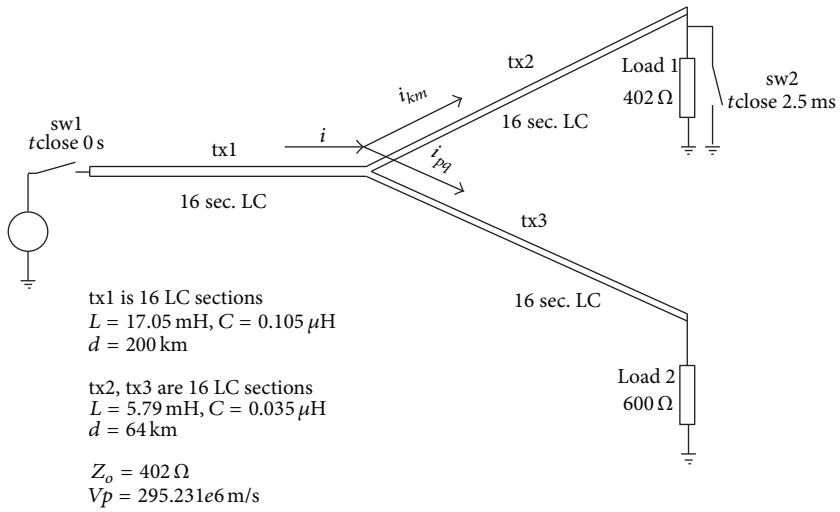

Figure 20: 47-node network under test.

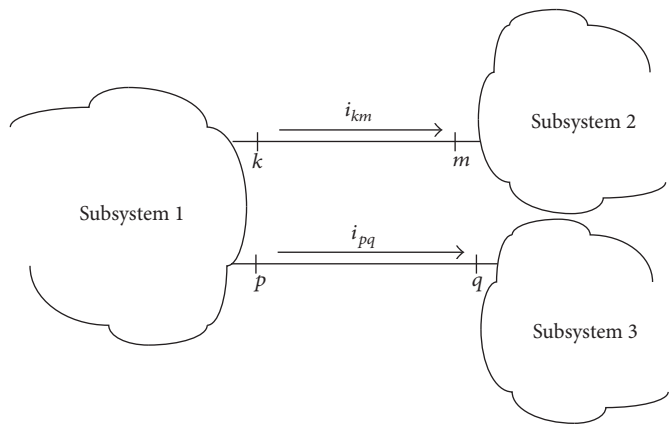

Figure 21: Equivalent three-section network.

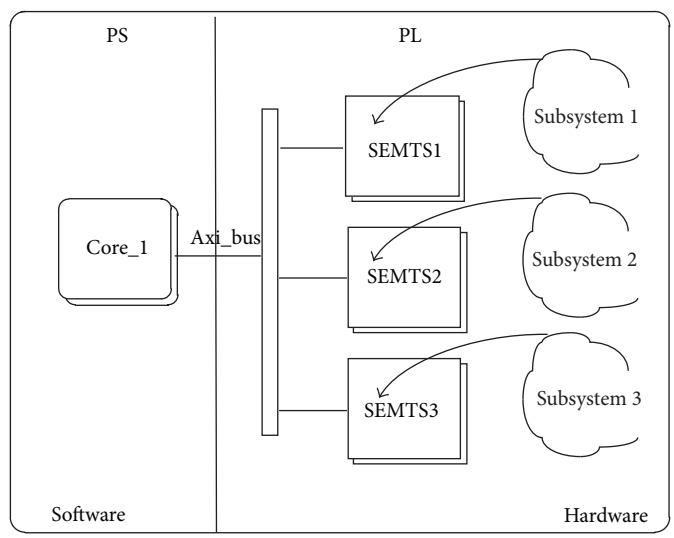

Figure 22: Power network distribution in the simulator.

Table 1: Measured timing for study case I.

Subsystem	Nodes	Time for LSS	Time for UHS	Time for GU	Time for SG	Total
1	17	3.88 μs	0.184 μS	0.432 μS	0.072 μS	4.568 μS
2	17	3.88 μs	0.184 μS	0.360 μS	0.02 μS	4.444 μS
3	17	3.88 μs	0.184 μS	0.304 μS	0.02 μS	4.388 μS

Table 2: Measured timing for study case II.

Subsystem	Nodes	Time for LSS	Time for UHS	Time for GU	Time for SG	Total
1	21	4.8 μs	0.212 μS	0.44 μS	0.07 μS	5.522 μS
2	21	4.8 μs	0.212 μS	0.44 μS	0.07 μS	5.522 μS
3	21	4.8 μs	0.212 μS	0.368 μS	0.02 μS	5.4 μS
4	24	5.9 μs	0.22 μS	0.384 μS	0.02 μS	6.508 μS

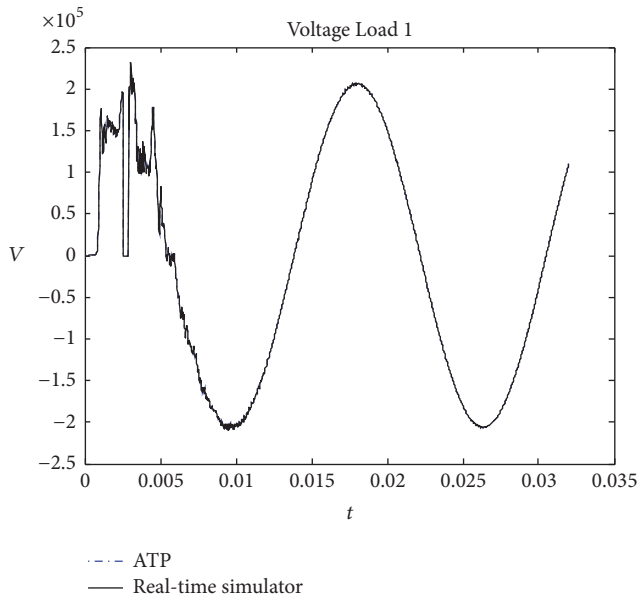

Figure 23: Voltage at load 1.

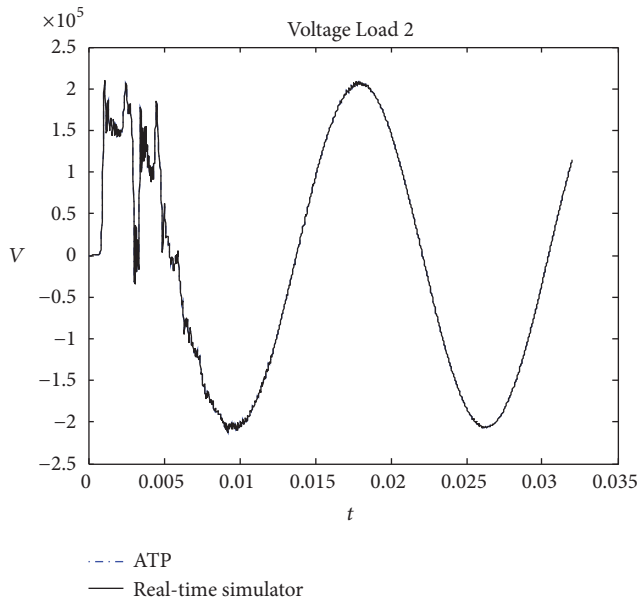

Figure 24: Voltage at load 2.

to the data obtained from the ATP program; the maximum percentage error detected between simulators is 3×10^{-4}. Figure 24 shows the same comparison at load 2, and an error of the same order is presented between waveforms.

6.2. Study Case II. The second power system to be simulated is shown in Figure 25. In this case, the initial system of 82 nodes was partitioned into four subsystems: subsystem 1 with 21 nodes; subsystem 2 with 21 nodes; subsystem 3, with 21 nodes; and subsystem 4 with 24 nodes. Both switches are closed at time $t = 0$ s. The required simulation step is 50 μS.

The power system was segmented and distributed into four subsystems, as in Figure 26. As there is now one extra subsystem, another SEMTS unit was enabled in the PL side. First, the solution was performed only in PS, detecting a maximum of 2300 μS per simulation step—46 times the target—so the processor in PS cannot comply with the simulation timing constraint. For the second run using both PS and PL, the timer in PS captured a maximum total simulation step of 23.2 μs. Table 2 summarizes the measured timing of the SEMTS units, in columns 3 to 6 specifying the individual timing for internals submodules, while in the last column the total time is specified.

The results show that an improvement of 99 times over the classical approach was obtained. Figure 27 shows the resulting real-time simulation waveforms at loads 1, 2, and 3.

Table 3 summarizes the PL resources utilization by module in PL; this summary includes utilization for LUTs, BRAMs, and DSP devices. The relative low percentages of utilization obtained can be noted, which suggests that a larger system can be simulated.

This work deals with the design and implementation of a real-time electromagnetic transient simulator on a SoC. The approach followed in this work allows reaching a deeper segmentation degree than current simulators which work based only on network segmentation through transmission

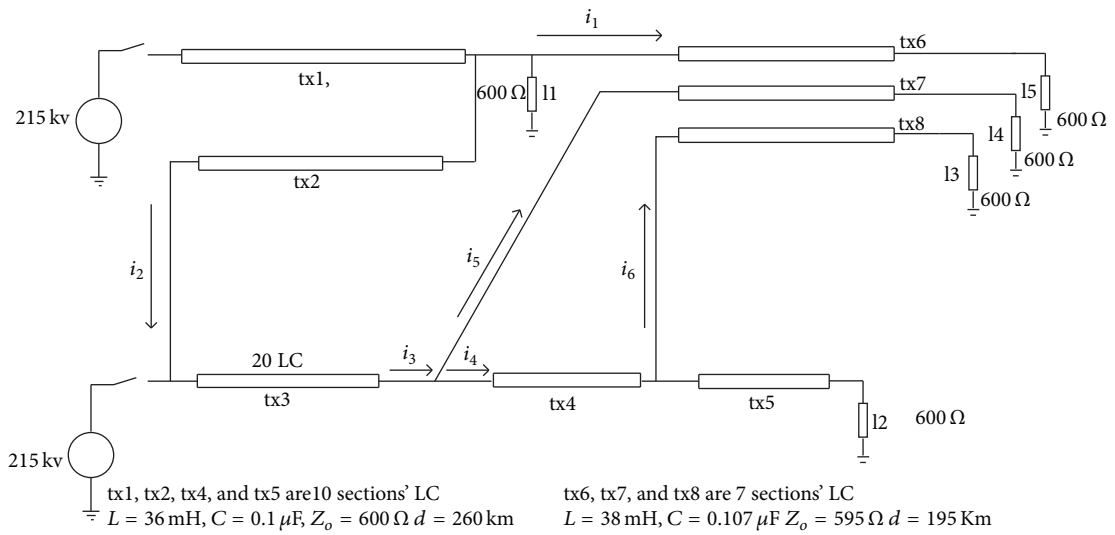

Figure 25: 82-node network under test in study case 2.

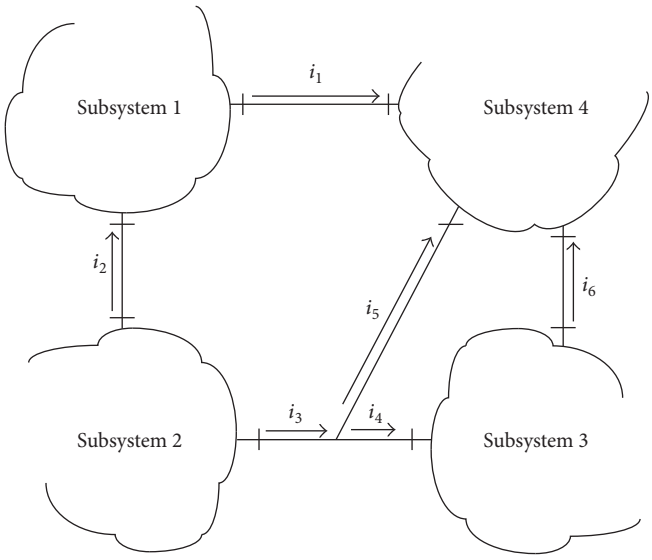

Figure 26: Equivalent four sections' network in study case 2.

lines. The proposed methodology can be used together with transmission lines; in this case, the power system is first segmented into areas by using transmission lines; then the generated areas can be segmented again by using the proposed method in this work, in order to obtain smaller subareas or subsystems in terms of nodes. A specific methodology for dividing the network is not discussed in this document, although it is logical that the best performance should be obtained by the method when the number of nodes is evenly distributed among the subsystems. An analysis to establish a formal method for breaking down the power system could be worthwhile.

Remark 1. The objective of using LC sections to represent transmission lines is only to illustrate how our proposed sectioning methodology distributes the computational burden for large systems.

7. Conclusion

The design of real-time electromagnetic transient simulators is a complex task due to fast transients; this leads to the research of modeling techniques, parallel algorithms, and the use of technologies capable of exploiting them. The real-time

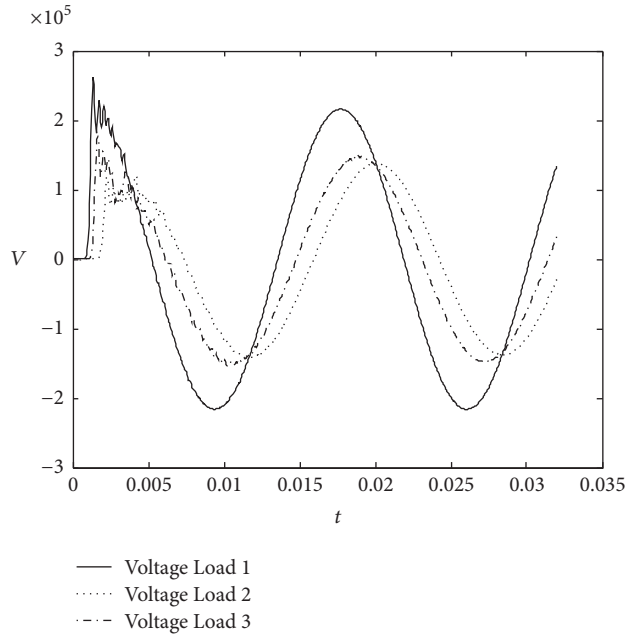

Figure 27: Simulator results for voltages at loads 1, 2, and 3, case of study 2.

Table 3: SEMTS and PL resources utilization.

Unit	LUTs	BRAMs	DSPs
SG	472 (2%)	1 (1%)	12 (8%)
HSU	472 (1%)	3 (1%)	12 (8%)
GU	8759 (3%)	4 (1%)	6 (7%)
LSS	13939 (6%)	13 (2%)	104 (11%)

electromagnetic transient simulator presented in this work has shown an increase of the simulation performance up to 99 times with respect to the ATP algorithm running on the PS in the SoC. The simulator can perform the simulation in real-time in sections with more than 80 nodes using the hardware and architecture currently available. For larger power systems, simulator performance could be improved by using faster FPGAs with more available resources, which can comply with the required simulation timing. Also, the interconnection of multiple FPGAs can provide more resources to implement faster hardware accelerators which also could improve the simulator performance.

The simulator framework was implemented in a SoC; the proposed architecture was designed to provide the necessary conditions for exploiting the parallelism enabled by the new segmentation algorithm by providing sufficient and replicated parallel hardware. In this sense, the simulator architecture is capable, for example, of solving multiple linear systems of equations, computing independently for history sources, or updating the conductance matrix of the generated subsystems simultaneously. The most hardware and time consuming accelerator is the LSS unit which implements a Gauss-Jordan hardware version designed to boost the solution. Hardware implementation allows solving it using multiples levels of parallelism which includes pipeline, parallel elimination procedure, and efficient architecture design.

Simulation results show a proper operation of the proposed simulator compared to the ATP method. The errors detected in the testing cases are negligible and are associated with the precision lost when extracting data from the ATP software. Some issues still remain as the designing of a specific hardware module for solving the link currents (for dealing with fault current). Solving the link currents in hardware will require a hardware which supports pivoting to avoid the numerical round error propagation. Another interesting issue to consider is the implementation of LU factorization in hardware.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] Y. Li, G. Li, Z. Wang, Z. Han, and X. Bai, "A multifeature fusion approach for power system transient stability assessment using PMU data," *Mathematical Problems in Engineering*, vol. 2015, Article ID 786396, 10 pages, 2015.

[2] M. Matar and R. Iravani, "The reconfigurable-hardware real-time and faster-than-real-time simulator for the analysis of electromagnetic transients in power systems," *IEEE Transactions on Power Delivery*, vol. 28, no. 2, pp. 619–627, 2013.

[3] Y. Chen and V. Dinavahi, "FPGA-based real-time EMTP," *IEEE Transactions on Power Delivery*, vol. 24, no. 2, pp. 892–902, 2009.

[4] M. M. A. Matar Bayoumi, *An FPGA-based real-time simulator for the analysis of electromagnetic transients in electrical power systems [Ph.D. thesis]*, University of Toronto, 2012.

[5] Y. Chen and X. Zhang, "Voltage balancing method on expert system for 51-level MMC in high voltage direct current transmission," *Mathematical Problems in Engineering*, vol. 2016, Article ID 2968484, 6 pages, 2016.

[6] U. Karaagac, J. Mahseredjian, I. Kocar, G. Soykan, and O. Saad, "Partial refactorization-based machine modeling techniques for electromagnetic transients," *IEEE Transactions on Power Delivery*, vol. 31, no. 5, pp. 2370–2378, 2016.

[7] J. M. B. Bezerr, A. A. P. Silv, Z. D. Lins, J. C. O. Junior, and E. L. Santos, "Field validation of a new model for uprating transmission lines," *Electric Power Systems Research*, vol. 134, pp. 30–37, 2016.

[8] J. R. Marti and J. Lin, "Suppression of numerical oscillations in the EMTP," *IEEE Transactions on Power Systems*, vol. 4, no. 2, pp. 739–747, 1989.

[9] N. Watson and J. Arrillaga, *Power System Electromagnetic Transients Simulation*, IET, London, UK, 2nd edition, 2007.

[10] H. W. Dommel, *EMTP Theory Book*, Bonneville Power Administration, Portland, Ore, USA, 1986.

[11] K. Yamamoto and A. Ametani, "An optimum adoption of iterative methods for nonlinear simulations on EMTP-type simulators," in *Proceedings of the International Conference on Power System Transients (IPST '01)*, Rio de Janeiro, Brazil, 2001.

[12] H. W. Dommel, "Nonlinear and time-varying elements in digital simulation of electromagnetic transients," *IEEE Transactions on Power Apparatus and Systems*, vol. 90, no. 6, pp. 2561–2567, 1971.