Implementation of Simulation Algorithms in FPGA for Real Time Simulation of Electrical Networks with Power Electronics Devices

Julio C G Pimentel
IEEE Industrial Applications Society
Cisco Systems Inc.
Ottawa, Ontario, Canada
pimentel@ieee.org

Abstract

Development of digital simulators for transient analysis of electric power systems has been the subject of many research works during the last four decades. In most studies, the simulation can be done off-line but in some particular situations, usually equipment testing, it may be mandatory to have the simulation executed in real time so that the interaction with real equipments can be studied. Digital Real Time Power System Simulators, DRTPSS for short, became a mandatory tool to ensure the quality of power systems.

Most of the DRTPSSs in use nowadays are based on high performance parallel processors using last generation technology. Recently, some works have proposed new architectures based on the use of FPGAs ("Field Programmable Gate Arrays") as a mean to increase the simulator's processing power without increasing its cost too much. This work presents the implementation in FPGA of a DRTPSS capable of simulating large electrical networks including high frequency power electronic devices.

1. Introduction

A general architecture of a DRTPSS consists in a high performance parallel processor, a simulation engine normally coded in C/C++ and input/output acquisition cards responsible for interfacing the simulator to the external equipment. The simulation engine has a library of device models the simulator can understand and a representation of the power systems network using for example state-space formulation [18]. Although parallel processor architectures present many desirable characteristics such as being readily reconfigurable to study new network topologies and not too difficult to scale it to address complex problems, it is very difficult to reduce the minimum time step at a reasonable cost. The processor-memory hardware paradigm and the inter-processor communication overhead experienced with the present technology set a minimum time step around 20 μs. The load on the main processor bus, the data memory allocation strategy and the data network load and latency cause a bottleneck in present digital systems [3].

In this paper we present the implementation details of a DRTPSS simulation algorithm in FPGA. The simulator is capable of simulating large electrical networks including high frequency power electronic devices. One important feature of the simulator is the use of multiple timesteps so that sub networks with very different natural frequencies can be simulated using the most appropriate timestep. In order to achieve such high performance the simulation algorithm is mapped directly into hardware following a hardware acceleration approach. The real time FPGA simulator can also be integrated to a parallel processor following a hardware and software co-design approach. This way, part of the simulation algorithm runs in software in the parallel processor and part of the algorithm runs directly in hardware as presented in [3]. This combined approach allows the real time simulator to simulate high frequency networks as well as very large networks. Finally, we should also point out this hybrid architecture has higher flexibility than a purely hardware or purely software architecture.

2. Related Works

FPGAs are being used in many areas as a way of accelerating algorithms. References [1] and [9] present FPGA hardware acceleration for the Conjugate Gradient and Jacob iterative methods respectively. These works show the FPGA implementation of the algorithm can achieve higher peak performance than

its implementation in software running in state-of-the-art microprocessors. The results presented show the architecture performs very well for large matrices especially when the solver runs continuously. Because of the real time constraint and the nature of the equations involved, these scenarios are rarely valid for real time simulation of power electronics circuits. The approach proposed here addresses the problem of small and medium size state space matrices where the calculation is bounded by the simulation time step.

The method proposed in [8] is based on the generalized algorithm developed by Dommel and implements each element of the network using their discrete-time equivalent resistances and current sources. This approach models the linear network at very low level as such it can use a lot of the FPGA area. The work presented in [17] is based on a hybrid solution where an FPGA is responsible for sampling the high-speed PWM firing pulses, conditioning them in 32-bit words and send it to two IBM 750GX double-precision RISC processors responsible for the algorithm computing. The FPGA also coordinates the signals the beginning of each small time-step in order to maintain the synchronism of the small time-steps in the two processors. This solution can achieve a reported minimum timestep of 2 μs which is one of the shortest results reported in the literature. We propose a different approach where the simulation is entirely realized in the FPGA. As presented in the following sections, the preliminary results show our approach can achieve a timestep smaller than 0.4 μs.

3. Overview of the DRTPSS using FPGAs

This section presents an overview of the DRTPSS high level architecture so that the reader can understand the constraints guiding the design and implementation in FPGA of each module of the system. Further information about its architecture can be found in [19].

3.1. The Programming Flow

Figure 1 shows the programming flow. The modules in the VHDL library were designed to be compatible with their SimPowerSystem™ counterparts [6]. The VHDL models were developed and tested using the following model-based methodology: 1) The modules are developed and tested using Simulink; 2) After the behavior of the model attained the expected accuracy it is coded in VHDL; 3) the results are compared against the simulink model and then stored in the VHDL system library.

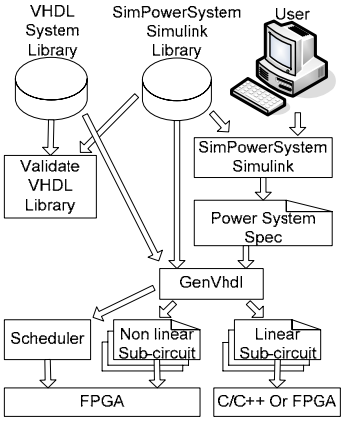

Figure 1. FPGA DRTPSS Programming Flow

A proprietary tool called GenVhdl reads the Simulink power network schematic, the VHDL libraries as well as data provided by the user and automatically generates the VHDL code that implements the real time simulation. The derived VHDL code can be synthesized with general commercial EDA tools using an FPGA design and implementation flow such as Xilinx ISE [7].

3.2. High Level Architecture Overview

The architecture shown in Figure 2 is a coarse grain dataflow architecture of highly specialized processor elements PEs running in parallel and synchronized at time step level. The power system sub-circuits are modeled as concurrent PEs which are interconnected through their input and output ports so that the inputs of a module can only change at the end of a time step. The outputs of the sub-circuits only depend on their inputs and their internal states. At the end of each time step each PE transfer their calculated voltages and currents to the next PE.

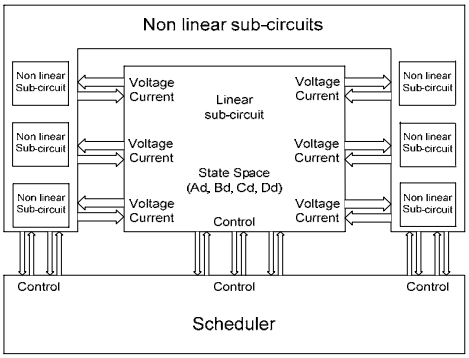

Figure 2. FPGA DRTPSS High Level Architecture

PEs may run at different time step depending on the natural frequency of the sub-circuit it is modeling.

Therefore, the system should provide a way of up sampling and down sampling the PEs inputs and outputs to the same clock domain so that they can exchange information with each other. The scheduler synchronizes the function of all PEs and is responsible for coordinating the following tasks: simulation initialization; generation of the multiple time steps reference clocks; synchronization of the simulation and exchange data between linear and nonlinear.

Decoupling the electrical network in concurrent PEs reduces the overall complexity of model what is highly desirable. However, because the PEs are simulated independently of each other they only exchange information at the end of a time step. It is well known that this strategy can introduce one or two timesteps delay in the feedback loop formed by the linear and non linear sub-circuits affecting the accuracy of the simulation and in certain cases causing it to become instable [16]. Also, the use of iterative methods to solve this problem is not usually possible because of the real time constraint. It should be used some other way to improve the accuracy of the simulation.

4. Distributed Concurrent Modules

Concurrent periodic EFSMs ("Extended Finite Sate Machines") are used in distributed real time systems as a formal representation of concurrent process [4]. We have modeled the PEs as a parallel composition of concurrent periodic EFSMs synchronized by a scheduler. Figure 3 shows their interfaces.

CLK – High frequency clock used as the reference clock for the whole system.
TS1 to TSn –Multiple time step synchronizing signals;
STC_1 to STC_n –The PE inputs are sampled when a STC for the module is asserted;
REG_1 to REG_n –Asserted at the end of each timestep to store the newly computed result at the output registers of the PE;
EOC_1 to EOC_n – Tells the scheduler all PEs completed their processing before the end of the timestep.

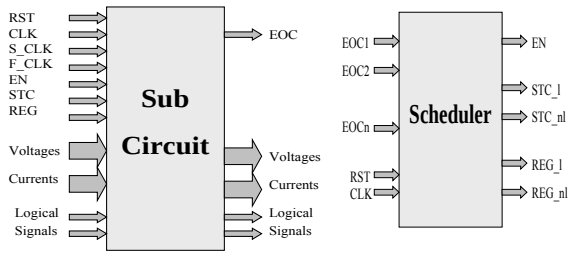

Figure 3. PE and Scheduler Standard Interface

They are defined as 6-tuple where the sixth element of the tuple is a counter holding the elapsed time since the EFSM has moved to a new state [4]. Because the devices modeled here do not use elapsed time we removed it from the PE EFSM representation. In this work an EFSM is defined as 5-tuple (S, I, V, δ, init) where: S is a finite set of states; I is a finite set of events (i.e., transitions); V is a finite set of variables that represent inputs, outputs and parameters; δ is a finite set of transition rules; and init represents the initial state and initial values of all variables. The transition rules are represented as a logical conjunction of integer linear inequalities on the variables. The EFSMs are made periodic by introducing a dummy transition as the last event of a path starting at the initial state so that it returns to the initial state [4]. In order to execute an event *a* the value of the associated transition rule P must be true. Input events are represented as *a*?x and output events are represented as *b*!y{y := E(x)} where x and y denote input and output variables, respectively. b!y@{y:=E(x)}[P] denotes that the output event *b* is executed at the end of the time step satisfying P and its output value y is E(x) [4]. This notation will be used in the following sections to describe PEs.

The EFSM representing a PE model is mapped to hardware such that the EFSM variables (input and output voltages and currents, some logical signals, etc.) are part of the data path implemented using MAC ("Multiply-And-Accumulate") units [1][2][11]. The remaining signals are part of the control path and are intended to send information to or receive information from the scheduler. The EFSM S and (I, δ) tuple elements are respectively the state and the branches of the PE state machine implementing the control path. All PEs follow the same standard interface specification which is based on the following processing model: at every time step sample inputs, process information, and hold computed data at outputs. The PE and the scheduler interfaces use the following main signals:

Special care was taken in the development of the non linear models (described in section 5) as well as in the design of the simulator itself to reduce the occurrence of unrealistic oscillations caused by the decoupling strategy used. Also, notice that the simulation becomes more stable and more accurate as the time step decreases when compared to the circuit natural frequencies. The scheduler can synchronize the PEs to let them run in series or in parallel using the same or different timesteps. Because non linear PEs can run at a timestep that is a fraction of the time required for the linear sub-circuit PEs. Simulating the first in series with non linear PEs do not increase much

the total timestep but reduces the decoupling delay to one timestep instead of two as is the case for parallel simulation. This improves the simulation stability. Also, multiple timestep simulation allows us to optimize the FPGA resource allocation. For example, if the linear sub-circuit requires to be simulated at 10 μs timestep and a PWM converter requires less than 1 μs, we can tell GenVhdl to allocate less hardware for the state space solver in order to allow more hardware for the PWM converter. Figure 4 shows an example of a series and a parallel sequencing.

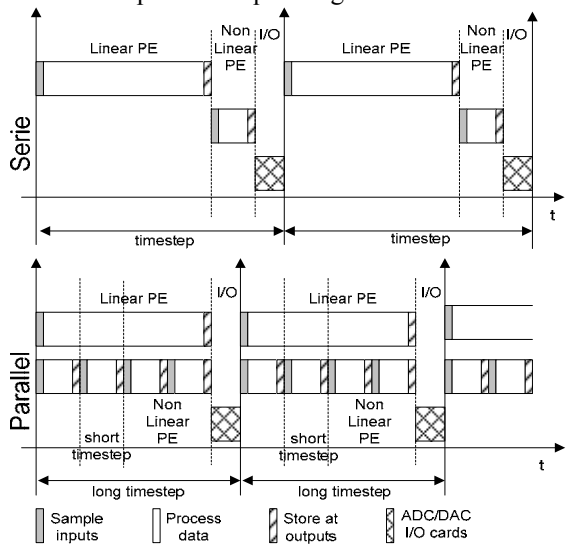

Figure 4. Series and Parallel Scheduler Sequencing

5. Recoupling link model

Sub-circuits running at different timesteps can not immediately exchange information. We should first decrease ("downsample") or increase ("upsample") the signal sample rate to the same reference clock. In order to simplify the design of the architecture we chose to let the PE running at the shortest timestep to up convert and down convert the signals going to PEs running at larger timestep. In this way, the VHDL system library includes a down sampler and an up sampler module and GenVhdl tool instantiates them into the appropriate PE according to their specified sampling rate.

5.1. Down sampling

Some downsampling methods have been proposed by works in multi rate digital filter design. If M is the specified downsampling rate, one method that is largely used consist in simply keeping one sample for every M samples at the input and discarding the remaining (M-1) ones [10]. This method is simple to implement but it has no physical relationship with how power electronics circuit works.

In general, power electronics circuits work based on averaging the current flowing through an inductor or the charge stored in a capacitor. Thus, an averaging based method is the natural choice because it resembles the normal functioning of the circuit. We have studied the moving average, fixed interval average and low pass filtering. The fixed interval average method was found to be the easiest to implement and the one that produced the best results. It consists in deriving the average simulated at each timestep of the fast clock during one timestep of the slow clock. For example, the average current between times Ta and Tb as a function of the fast clock timestep Δt is defined by (1):

$$\overline{i(t)} = \frac{1}{T_b - T_a} \sum_{k=1}^{n} i(T_{a+(k-1)}) \cdot \Delta t \,, \qquad (1)$$

The approach assumes the slow clock period is an integer multiple n of the fast clock period so that (1) can be rewritten as (2):

$$\overline{i(t)} = \sum_{k=1}^{n} \left(\frac{1}{n} \cdot i(T_{a+(k-1)}) \right), \qquad (2)$$

Figure 5 shows the data path of the downsampling module implementation. It uses a MAC unit where one input is the fixed value (1 / n) and the second input is the instantaneous output of the PE running at the fast clock. The MAC accumulator is cleared at the beginning of every slow timestep and the accumulated result is transferred to the next PE at the end of it. Because the numerical representation of (1 / n) is calculated a priori by GenVhdl and passed to the PE, the ratio n do not need to be a power of 2 so that the simulator accepts any integer ratio between the slow and fast timesteps.

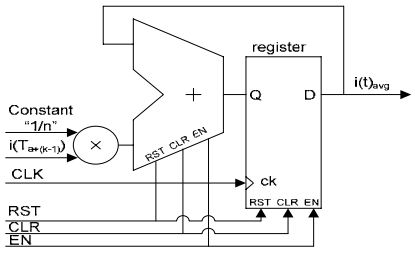

Figure 5. Down Sampling Hardware Implementation

5.2. Up sampling

The fast clock PE inputs are sampled at the beginning of each slow clock timestep. The simplest

method to up sample them is to keep their value constant during the whole slow clock timestep [10]. Because the maximum bandwidth of the input signals can be comparable to the frequency of the slow clock this approach can incur in non negligible simulation errors. A more accurate approach is to use a predictor to extrapolate the value at the next timestep based on the past history of the slow clock sampled data, and update it with the new sampled value at the beginning of the next slow clock timestep. We have considered only lower order predictors for two reasons: higher order methods have smaller region of stability; lower order methods consume lesser FPGA resources. According to our simulation results, Adams-Bashforth-2 produced about half the error of Mid-Point or Euler. Besides, it consumes little hardware so that we chose to implement the Adams-Bashforth-2 extrapolation rule [13][14]. We can rearrange its interactive equation to use only addition, subtraction and shift operations:

$$k_1 = f\left(x(n), y(n)\right) + 0.5 \cdot f\left(x(n), y(n)\right), \quad (3)$$

$$k_2 = k_1 - 0.5 \cdot f\left(x(n-1), y(n-1)\right), \quad (4)$$

$$y(n+1) = y(n) + k_2 \cdot \Delta T, \quad (5)$$

As shown in figure 6, the past samples are used to calculate the extrapolated value y(n+1). Afterwards, we use a linear regression to interpolate the upsampled results between y(n) and y(n+1) at the fast clock timesteps. In hardware, the predictor is implemented by the circuit shown in figure 7.

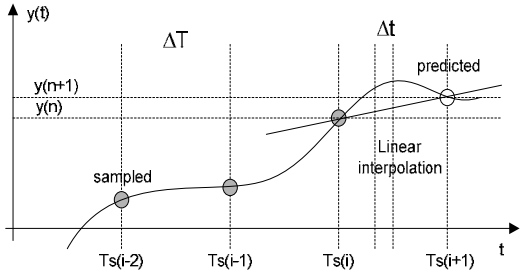

Figure 6. Up Sampling Method

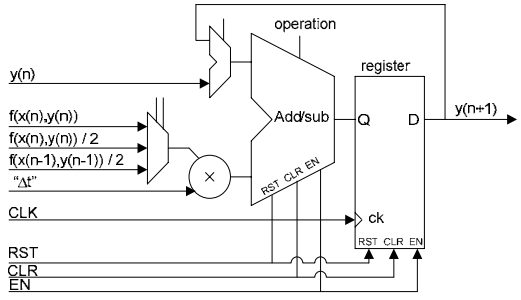

Figure 7. Up Sampling Hardware Implementation

6. Power Electronics Devices

Currently the VHDL system library includes five power electronic devices: diode, MOSFET, thyristor, power switch and power bridge. We present the models of the thyristor and the power bridge in the following sections. The diode, MOSFET and power switch are not presented here because their model can be extrapolated from the two models presented. We should notice that a similar approach can also be used to model others types of non linear devices.

6.1. Modeling of the thyristor PE

Figures 8 and 9 show the thyristor VHDL entity definition and its electrical and EFSM representation respectively. We should notice the module is fully configurable so that it can be used in many different applications. The same is also true for the others modules of the VHDL library. The EFSM configuration variables are passed to the model as VHDL generic parameters while the I/O and control variables are passed as signals [12].

```
entity thyristor is
  generic (Vf : natural:= 16; Ic : natural := 0);
    Ts1 : natural:= 1; Ts2 : natural:= 1;
    Ron : natural:= 16; Lon: natural:= 0;
    NBits: natural:= 32; NBitsRadix: natural:= 8;
  port (
    CLK: in std_logic;
    TS_sync: std_logic;
    RST: in std_logic;
    EN: in std_logic;
    Reg_output: in std_logic;
    Gate: in std_logic;
    STC: in std_logic;
    Vak: in std_logic_vector(NBits-1 downto 0);
    Iak: out std_logic_vector(NBits-1 downto 0);
    Iavg: out std_logic_vector(NBits-1 downto 0);
    EOC: out std_logic);
end thyristor;
```

Figure 8. Thyristor VHDL entity definition

Electrically, the thyristor is modeled as a switch in parallel with a RC snubber [5][6]. The input and output variables are respectively (Gate, Vak) voltages and the averaged and instantaneous Iak currents. When turned off it is an ideal switch so that Iak(n)=0. Otherwise, when it is turned on it is modeled by a series RLV circuit. During the decoupling phase, the RC snubber is placed to be simulated in the linear sub-circuits. The remaining devices in the model are simulated by the thyristor PE. The continuous time transfer function H(s) of the two port RLV network is given by (6):

$$H(s) = \frac{I_{ak}(s)}{V_{ak}(s) - V_f(s)} = \frac{(1/R)}{1 + s \cdot (L/R)}, \quad (6)$$

Using backward Euler discretization rule we can write (7) which is implemented in hardware using the same approach explained in section 5.

$$I_{ak}(n) = A_1 \cdot V_{ak}(n) + A_2 I_{ak}(n-1)), \quad (7)$$

Where: $A_1 = \frac{T_s}{L_{on} + R_{on} T_s}, A_2 = \frac{L_{on}}{L_{on} + R_{on} T_s}$

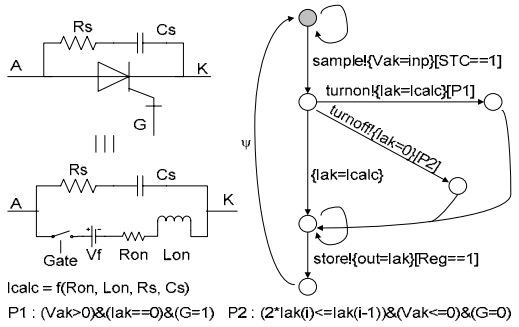

Figure 9. Modeling of the Thyristor

6.3. Modeling of the Universal Power Bridge

As defined in SimPowerSystems™ from Matlab, the Universal Power Bridge ("UPB") block implements a universal three-phase power converter that consists of up to six switches connected in a bridge topology and can be configured to model many different types of power switches [5][6]. Currently, the VHDL library of the FPGA real time simulator includes three types of UPBs: diode, thyristor and MOSFET. This section describes the MOSFET UPB implementation.

Figure 10 shows the electrical model of a single-phase UPB composed of one arm. A three-phase UPB is formed by three arms connected in parallel. Each half of the arm, called a bridge, includes a forced-commutated switch controlled by signal Gx and a naturally commutated flywheel diode. The proper operation of the UPB requires the following conditions to hold: 1) the two halves of an arm can not be turned on at the same time; 2) every time Q1 is on the associated diode D1 is turned off and vice-versa; 3) If Gx=1 then Qx is on; 4) Current Ia can not suffer sudden changes.

During switching, the UPB presents a non linear behavior which combined with one or two timesteps delay originated by the decoupling strategy can cause simulation errors and possibly instability. To mitigate these problems, the UPB EFSM coordinates the data path to ensure current through the UPB devices will be redistributed respecting voltage and current Kirschoff's laws as well as energy conservation. The simplified EFSM representation of bridge #1 is shown in figure 10. Variable B1_state stores the aggregated value of B1_on, the ON/OFF state of bridge, during the current and the last two timesteps. Notice the RC snubber is simulated as part of the linear sub-circuit so the current flowing through it is not readily available to the UPB. However, to accurately model the switching, the UPB should take it into account. The UPB model includes a first order predictor to estimate this current. When the new bridge voltage VQ1 arrives one time step later the new value is used to correct the estimated value. Figure 11 shows the UPB data path.

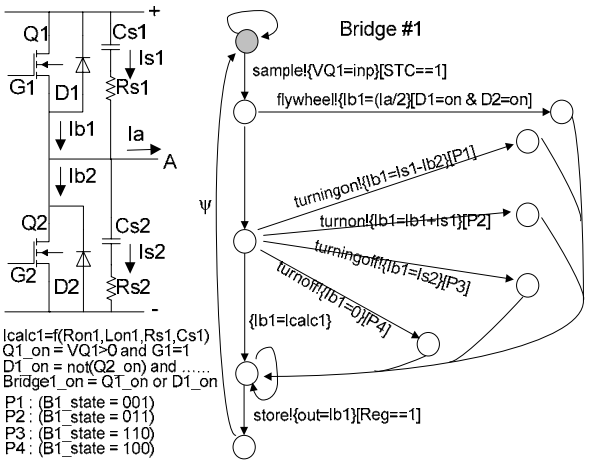

Figure 10. Modeling of the UPB Bridge #1

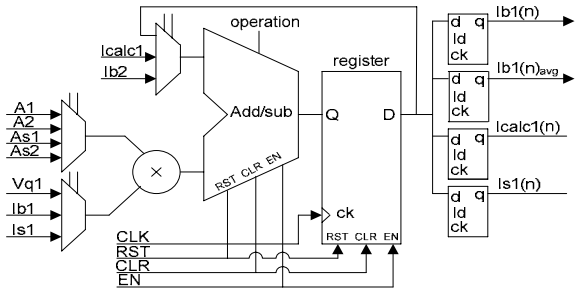

Figure 11. Data path of UPB Bridge #1

7. State Space Solver

Here we present the hardware implementation of the state space solver. Its architecture exploits the characteristics of the state space formulation to allow an efficient mapping to the FPGA resources. The linear sub-circuits are modeled by their discrete-time state space formulation (8) and (9):

$$X(n+1) = A_d \cdot X(n) + B_d \cdot U(n), \quad (8)$$

$$y(n+1) = C_d \cdot X(n) + D_d \cdot U(n), \quad (9)$$

Where n is the timestep number and Ad, Bd, Cd and Dd are the discrete state space matrices. The high level architecture of the state space solver is shown in figure 12. It implements the linear sub-circuit PEs. The control signals MemAddr, SigAddr, doing_X_not_U, etc. are generated by its internal state machine.

Its basic module is the VVM ("Vector-To-Vector Multiplier") which multiplies one column of a matrix (Ad, Bd, Cd or Dd) by a vector (X or U). It includes a MAC unit, a blockRAM memory bank configured as 512 entries x 32 bits, multiplexers and a state machine. The MAC units can run at a minimum clock frequency of 150 MHz with a latency of 6 clock cycles [1][2].

We should notice the matrices can have very different dimensions. Also, the minimum dimension can vary from 1 up to hundreds of states. GenVhdl maps the ODE state space formulation into the FPGA using a priori algorithm. GenVhdl takes into account some parameters such as the number of VVMs to use for the simulation, matrices dimensions, MAC pipeline latency and the number of clocks per timestep to calculate the optimum distribution of VVMs per matrices that minimizes the timestep. The current implementation can solve a state space with 10 states in less than 0.4 μs.

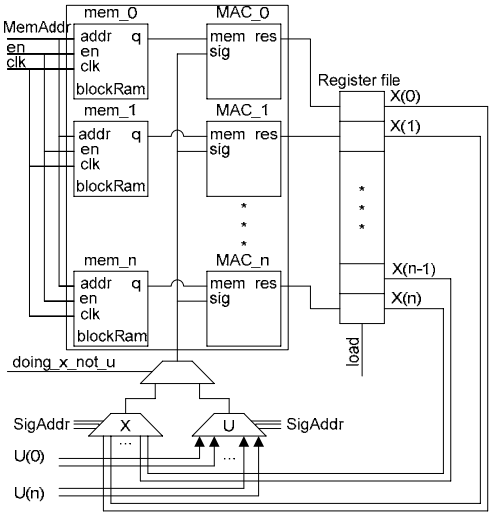

Figure 12. State Space Solver Macro Architecture

8. Others Modules

Besides the modules presented in the previous sections, the VHDL system library includes PEs to realize many others functions such as voltage and current sources, Delta-Sigma and PWM modulators, PI and PID controllers, digital filters, Clark/Park transforms, etc. The implementation of some of these modules and their experimental results are presented in [15] [16].

9. Experimental Results

In this section we present the simulation results using the FPGA simulator on a three-phase DC-AC converter shown in figure 10 [5]. The circuit works on the principle of converting energy from the DC voltage source into a 60 Hz sinusoidal current flowing through the three-phase charge RL according to the configuration parameters set to the PWM modulator. The gates of the MOSFET power switches are controlled by a sinusoidal PWM modulator which had its carrier frequency set to 2 kHz at a modulation index of 0.85. The PWM was configured to generate a 60 Hz sinus at the output when the high frequency carrier is filtered out. The results are comparable to those obtained with SymPowerSystems from Mathworks, which is a commercial power system simulator tool largely used by electrical utilities and research centers [6]. The simulation used a fixed timestep Ts shown in the schematic diagrams.

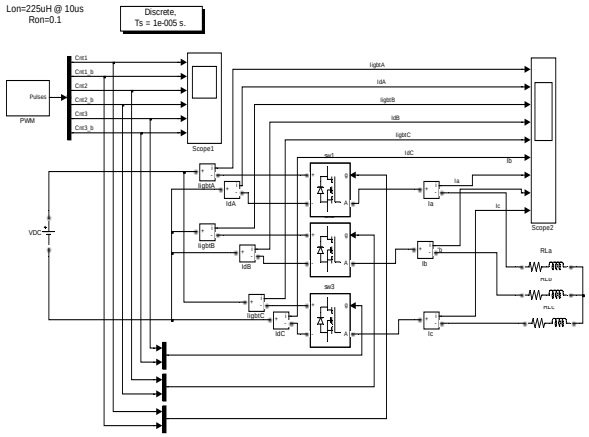

Figure 13. Three-Phase DC-AC PWM converter

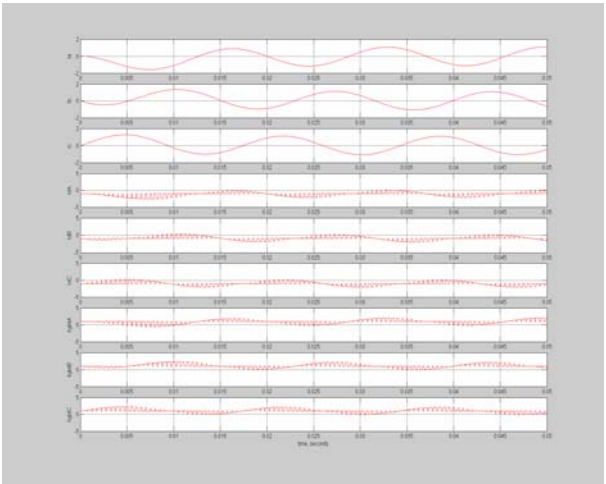

Figure 14. Simulation Results Using FPGASim

10. Conclusion

This work presented the implementation of a DRTPSS which is fully realized in a FPGA. The results demonstrate that modern FPGAs are effective platform for implementing simulation algorithms and can compete favorably with high-performance microprocessors and DSPs in applications where the algorithms can be parallelized. The test environment built consists of an AMD XP2400+ microcomputer and a Digilent Inc. XUP Virtex II Pro Development FPGA Card with a 2VP30-7-FF896 Virtex II Pro FPGA. Currently, we can simulate small and medium size power electronics networks such as DC-AC converters, AC-AC cicloconverters, etc. with a timestep smaller than .4 μs. We should notice that the smallest timestep reported in the literature is around 2 μs. The non linear PEs can run at a time step of about 0.2 μs. Therefore, the state space solver is the system bottleneck. We recently found inefficiencies in the MAC design that should allow us to run it at 200 MHz. Also, we devised a better way of coordinating the VVMs to increase the number of operations per timestep. We estimate these changes combined should allow the linear PEs to achieve a timestep of 0.2 μs.

11. Acknowledgement

J. C. G. Pimentel thanks Xilinx Inc. for its support providing the FPGA development kit used for the simulation and experimental results, and Dr. Yosef Tirat-Gefen at Castel Research Inc., Dr. Guilherme De Souza at Univ. of Missouri-Columbia and Dr. Antonio Mesquita at COPPE/UFRJ for valuable feedback during the preparation of this paper.

12. References

[1] G.R. Morris, V.K. Prasanna and R.D. Anderson, "A Hybrid Approach for Mapping Conjugate Gradient onto an FPGA-Augmented Reconfigurable Supercomputer," FCCM'06, 2006.

[2] X. Wang, M. Leeser, and H. Yu, "A Parameterized Floating-Point Library Applied to Multispectral Image Clustering," 7th MAPLD International Conference, 2004.

[3] J.C.G. Pimentel and H. Le-Huy, "Developing a New Architecture for Digital Real-Time Power System Simulators Based on Pentium II and FPGAs," in ICDS'97 Conference Proceedings, 1999.

[4] T. Kitani, Y. Takamoto, K. Yasumoto, A. Nakata and T. Higashino, "A Flexible and High-Reliable HW/SW Co-Design Method for Real-Time Embedded Systems", Proceedings of the 25th IEEE International RTSS, 2004

[5] T.L. Skvarenina, "The Power Electronics Handbook," CRC Press, 2002.

[6] SimPowerSystems, Matlab Inc., 2005.

[7] ISE Development System, Xilinx Inc., 2005.

[8] M. Matar, M. Abdel-Rahman, A. Soliman, "FPGA-Based Real-Time Digital Simulation," International Conference on Power Systems Transients (IPST'2005), 2005.

[9] G.R. Morris and V.K. Prasanna, "Pipelined Datapath for an IEEE-754 64-Bit Floating-Point Jacobi Solver," 9th High Performance Embedded Computing Workshop, 2005.

[10] J. Franca, A. Petraglia and S. K. Mitra, "Multirate Analog-Digital Systems for Signal Processing and conversión," Proceedings of the IEEE, Vol. 85, No. 2, February 1997.

[11] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs," Oxford University Press, 2000.

[12] D.J. Smith, "HDL Chip Design: A Practical Guide for Designing, synthesizing and simulating ASICs and FPGAs Using VHDL or Verilog," Doone Publications, 8th Ed. 2000.

[13] A. Ralston and P. Rabinowitz, "A First Course in Numerical Analysis," 2nd Ed. Dover Publications Inc., 2001.

[14] L.O. Chua and P.Y. Lin, "Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques," Prentice Hall, 1975.

[15] J.C.G. Pimentel, H. Le-Huy and G. Sybille, "A VHDL Library of IP Cores for Power Drive and Motion Control Applications," CCECE'2000, 2000.

[16] J.C.G. Pimentel, H. Le-Huy and G. Sybille, "An FPGA-Based Real Time Power System Simulator for Power Electronics," 7th MAPLD International Conference, 2004.

[17] T. Maguire and J. Giesbrecht, "Small Time-step (< 2μSec) VSC Model for the Real Time Digital Simulator," International Conference on Power Systems Transients (IPST'2005). 2005.

[18] C. Dufour, J. Bélanger, "A Real-Time Simulator for Doubly Fed Induction Generator based Wind Turbine Applications", Proceedings of IEEE 35th Power Electronics Specialists Conference (PESC 2004), June, 2004.

[19] J.C.G. Pimentel, "A High Performance Architecture for Real Time Simulators Based on FPGA Hardware Acceleration," Real Time Simulation Systems, December, 2006 (submitted for presentation).