New Computational Modeling for Solving Higher Order ODE based on FPGA

Alireza Fasih[1], Tuan Do Trong[2], Jean Chamberlain Chedjou[3], Kyandoghere Kyamakya[4]

[1, 3, 4] Alpen-Adria University of Klagenfurt – Austria

{alireza.fasih, jean.chedjou, kyandoghere.kyamakya}@uni-klu.ac.at

[2] Hanoi University of Technology, Hanoi – Vietnam

dotuan@mail.hut.edu.vn

***Abstract*—*In this paper we propose a method for solving complex higher order ordinary differential equations (ODE) based on an emulation of the analog computing paradigm on digital hardware platforms. In this case, we mimic real analog system elements by digital discretized models. Due to the flexibility and re-configurability of FPGA and also the possibility of system behavioral modeling through hardware description languages (HDL), we are able to create all fundamental elements that are necessary to both simulating complex systems and the modeling of any ordinary differential equations (ODE) or a system simulation based on ODEs. We therefore propose a novel methodology of solving systems and higher order ODEs. This technique is similar to the analog computing but with the key difference that we possess more flexibility and are able to control at will the precision level wanted/needed. Further features are values scaling of both the results and internal variables.*

***Keywords*—*HDL, ODE, Digital Emulation of Analog Computing, Rössler Equation, FPGA*.**

I. Introduction

Solving ordinary differential equations is essential in most scientific fields [1]. Around the 1950s, analog computing was the only solution/technology to solving the differential equations and simulating complex dynamic systems by using analog electronic components. In the last decades this method has been pushed away by the digital computing revolution. Almost all researchers have switched to numerical and algorithmic methods for solving ODEs. But the digital computing has also its limitations. Thus, recently some researchers are exploring ways to return to the use of the analog computing method again [2], especially in cases where ultrafast speed of the solving process is needed (see realtime simulation needs for example). There are many reasons behind this decision. The first and most important issue is the processing speed. Much of the physical phenomena in the real world are measured/expressed by calculus; therefore with analog computing we can simulate these models very fast.

Due to the inherent process and computing parallelization, analog computers are capable of producing complex solutions in real time. In the early days of the analog computing age (more than 40 years ago) they were facing a series of limitations due amongst others to the use of discrete electronic components of this analog computing paradigm: imperfect connections between elements, limitation in the voltage scaling, variability of elements characteristics during the process due to the temperature, etc. Also the precision of the component characteristics values is a serious technological non-scalable limitation. In a real analog computer, there is a limitation of the voltage scaling. The voltage is limited between the noise level and the high voltage level. Because of this physical limitation one can not reach solutions that are out of this boundary (interval). Since the 1980's up to now, many researchers have been trying to implement analog computation systems for specific problems in VLSI chips [3, 4]. In VLSI chips, one can rescale the voltages within CMOS or TTL ranges. One particular weakness of this approach (i.e., analog VLSI implementation) is that the circuit is not re-configurable and for solving a new problem we have to design another chip, what is a very expensive issue. Another problem is that we are not able to re-scale the time in this case. In some cases, for coupling system components, time synchronization becomes necessary. Due to the limitations of the VLSI approach, we have started thinking of an alternative that consists in the essence of an emulation concept of the analog computer on top of reconfigurable and scalable digital platforms, especially FPGA chips. This paper shows that this emulation has been successful and that we are capable of modeling and solving any type of higher order even nonlinear ODE at an ultra-fast speed on this fully digital structure.

II. HDL description and System Architecture for the Analog Computing Emulation Concept on FPGA

Hardware description languages are similar to the normal programming languages. The main difference between sequential programming and HDL programming is that in

HDL one describes the operations of digital circuits by code at either low level gates or at behavioral level. But in sequential programming we have a flow chart and an algorithm that should sequentially run on a single execution CPU. In contrast to a software programming language, HDL syntax and semantics include explicit notations for expressing time and concurrency, which are the primary attributes of hardware. There are many types of HDL descriptions that can cover from low level gates up to behavioral modeling, such as Verilog, SystemC, VHDL, HandelC and so on. Verilog is the only language that can cover the whole of this domain. It means that for description of system operations there is the possibility to use a mix of gate level programming and behavioral programming. Also, Verilog is easy to understand, and one can describe the functionality of complex circuits by using this language. To get good performance we have to execute operations such that as many operations as possible are in parallel mode. This is due to the low clock rate in FPGA if compared to a dedicated CPU. For implementing a system simulation or solving an ODE by a flow diagram, we need some basic and fundamental arithmetic components, such as summation, subtraction, multiplier, and integrator. For solving ODEs by analog computing, there is no general solution, as each differential equation requires its own unique circuit or bus connections. However, one could in theory create a series of reconfigurable matrix switches that would be rewired for obtaining circuits and bus structures to different arrangements depending on the problem (ODE) at hand. The implementation of these functions on a digital platform (here FPGA) is easy, except that of the analog "*Integrator*". For this we use and adapt an old method called "digital differential analyzer (DDA)" for computing time integrals. In the next section, this technique is presented.

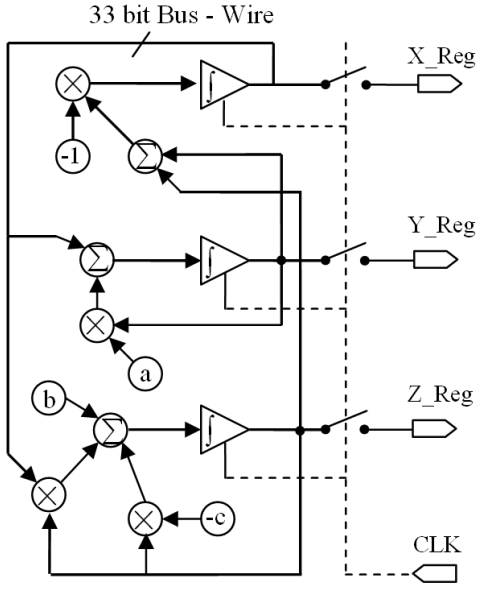

Figure 1. Flow diagram for modeling the Rössler Equation (see Equation (2)) in the analog computing scheme emulated on FPGA

Depending on the accuracy needed either for a given system simulation or for ODE solving, we can define a specific data type for modules and basic elements. For each basic element such as subtraction, addition and multiplier we define a fix-point data type with proper "integer" and "precision" ranges. Depending on the complexity of the equations, we can extend the number of bits used for storing both the "integer" and "precision" values. The MSB (most significant bit) bit is for sign, and for subtraction we use the 2's complement method. The key advantage of a fix-point data coding in this case is the processing time and also the saving of FPGA resources. In a floating point representation alternative data coding we should have to use complex modules and architectures and it would need/require more resources on the FPGA when compared to the fix-point alternative.

Still now, there is no general purpose architecture for solving any type of ODE equations [8], but we propose a method for designing a system and solving ODEs in a straight forward process (flow diagram). The flow diagram is consisting of many basic elements that are coupled together. Fig. 1 shows a flow diagram for solving the so-called "Rössler equation" (see Equation (2)). In this diagram, outputs are registers, and internal connections are bus wires. The data bus width for connecting the components is the same in all parts of the diagram. We use 33 bits for sign registers, sign wires and storing the values and integrators. All components are synchron and the system is operated by a common clock.

III. Digital Differential Analyzer

A digital differential analyzer (DDA), also sometimes called "digital integrating computer", is a digital implementation of the differential analyzer. The integrators in DDA are implemented as accumulators, whereby the numeric results are converted back to a pulse rate by the overflow of the accumulator. The main advantage of the digital integrator, when compared to an analog integrator, is the scalable precision. Also, in a digital integrator based on DDA, we don't have drift errors and noise [6] due to the imperfection of electronic components. By accumulation over time of values in a register we can calculate the integral of signals. The basic digital integrator is expressed by (1).

$$X_{n+1} = X_n + K \cdot S \tag{1}$$

In (1), X_{n+1} denotes the next state of the accumulator used for calculating the integral. The coefficient of K is a constant factor that is less than 1; it is used for time scaling. In this equation S denotes the input signal for integration. We can map this technique on FPGA very easily by writing a behavioral code. After each rising clock pulse, the equation updates the integral value. In this integrator, Rounding or truncation errors are only due to the limitation of registers. Therefore, by increasing the register sizes we have a way to

control/reduce this error. This error is cumulative. Thus for low precision registers a lack of accuracy will be observed after long time The only way to overcoming to this problem is setting proper register sizes.

IV. Combination of Software and Hardware

The Hardware implementation of systems for solving ODE equations by this method is one issue. Another issue is the communication of this module or system with the physical world for setting parameters, coefficients, and initial conditions. For making a System-on-chip (SoC) we need a central processor, a standard bus, and IO cores that are integrated together. The IBM Processor Local Bus (PLB) is a technology than can support and manage all of these facilities. PLBs are supported slave and master for controlling the IP (Intellectual Property) on the bus. There is a possibility to passing the parameters and values either through simple registers in PLB or by a first in first out method (FIFO) in PLB. In Xilinx Virtex-4 family we can design a system based on PLB and custom IPs. In this case, we integrated a PowerPC 405 (PPC) and Verilog modules as a custom IP by a 32bit PLB. PPC is a hardcore 32bit processor that can provide good performance as a central processor for controlling the custom IP and other peripherals. Through this bus also we are able to send interrupt signals, so Verilog modules can send a signal to PPC 405 as an interrupt for doing some process on the output data. When the output of the module is valid, it sends an interrupt signal through PLB to PPC 405. PLB and everything that is connected to this bus is synchron due to a common clock. By using System ACE (Advanced Configuration Environment) CF (Compact Flash cards) technology we can record results data in files. Then transferring or sending these data files to a computer (PC) through either the normal or high speed serial link is then feasible. On the PC the results will be visualized, saved or used for other purposes.

In some Xilinx family there is a possibility to using soft-core CPU like MicroBlaze, instead of hardware-core (PPC405). It is obvious that for using software-core CPU like MicroBlaze we need more logic slice resources. Therefore, for saving logic slice resources in software integration, we applied a hardware-core CPU (PPC 405).

V. Experimental Results

For checking the accuracy of the developed system, we have implemented a Rössler equation. Equation (2) shows this Rössler equation, which is a set of three highly non-linear ordinary differential equations. This system can, depending on the set of parameters, generate chaotic signals at the output. Equation (3) shows the set of parameters we have used for generating chaotic waves. Traditional methods for solving Rössler equations in digital computers are based on numerical methods, such as Euler or Runge-kutta algorithms. Depending on the initial conditions and the set of parameters, the system can converge to specific orbits, to a point or diverge to infinity. For modeling this equation by HDL we needed 6 multipliers, 3 integrators, and 4 adders. For realizing subtractions, we do apply additions by the 2's complement method. For coupling these components, we have to use wire bus connections. Also, for storing the values of the states we need registers. There is no exact rule for the definition/fixation of the number of bits for both integer and precision parts of the fix-point data coding. Depending on the complexity of the system at hand we can define a safe range for sign register and wires. For Rössler we have set in this work a 33bit bus for connections between modules and accumulating the values in integrators. Further, we have respectively set 1bit for sign, 8 bit for the integer and 24 bit for the fraction parts. With this setting for the variables and Connections, results of Rössler equation (x, y and z) will be limit in maximum range of ±255 (1-bit sign, 8-bit integer and 24-bit fraction). We have implemented this equation on a Xilinx ML405 evaluation board with a 100Mhz clock rate. This board is based on Virtex-4 FX20 family, with 20k logic cells and 32 DSP48 slices.

$$\begin{aligned} \frac{dx}{dt} &= -y - z \\ \frac{dy}{dt} &= x + ay \\ \frac{dz}{dt} &= b + z(x - c) \end{aligned} \tag{2}$$

$$\begin{aligned} a &= 0.41 \\ b &= 1.15 \\ c &= 4.16 \end{aligned} \tag{3}$$

At its most basic level, a DSP48 is a multiplier with a combination of adders and many optional operations. It has a 18-bit sign input signal and a 36-bit sign output result. This result is then sign extended to 48-bits, it can either be fed into the adder or connected directly to the output of a DSP48. Pipeline registers are a unique advantage of the DSP48 block compared to other FPGA DSP architectures [7]. Xilinx has dedicated many DSP48 slice in Virtex-4 and Virtex-5 family for speeding-up of calculations. The number of resources used after synthesis for solving the Rössler equation in terms of DSP48 slices, Flip-Flop slices and 4-Input LUT Slices is shown in the Table 1.

TABLE I. FPGA resources used for implementation of an emulated analog computing solver of the Rössler equation on FPGA

# Components (available)		18bit Rössler	33bit Rössler
Flip Flops Slices	(17088)	1%	1%
4 bit LUTs	(17088)	1%	2%
Bonded IOBs	(320)	1%	2%
GCLKs	(32)	6%	6%
DCM_ADVs	(4)	25%	25%
DSP48a	(32)	25%	62%

Integrated Software Environment (ISE) is the Xilinx® design software suite that allows you to take your design from design entry through Xilinx device programming. By disabling the DSP48 slices in this program, the system tries to synthesize the code and design by normal logic slices. The FPGA synthesis report of Table. 2 for the 2 scenarios with and without DSP48 slices shows what difference there is in terms of FPGA resources use/consumption between the two alternatives.

TABLE II. BENCHMARKING OF RESOURCES USE FOR FPGA SYNTHESIS REPORT FOR IMPLEMENTING A 33-BIT ODE SOLVER FOR THE RÖSSLER EQUATION

# Components (available)	**% of resources used when using DSP48**	**% of resources used, without DSP48**
F-Flop Slices (17088)	1%	2%
4 bit LUTs (17088)	2%	21%
Bonded IOBs (320)	2%	2%
GCLKs (32)	6%	6%
DCM_ADVs (4)	25%	25%
DSP48a (32)	62%	____

In some Xilinx family FPGA, there is no DSP48 slice for implementing Integrators, multiplier and basic elements, so we need more logic cells to implementing these resources. According to Table. 2, 62% DSP48 (20 slice DSP48 in Virtex-4, FX20 FPGA) is equal 21% (3600 slice, 4-bit LUTs in Virtex-4 FX20 FPGA) of basic logic slices.

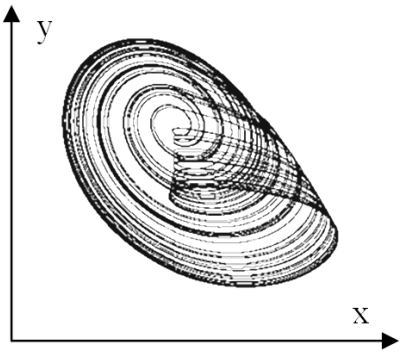

Figure 2. Rössler equation output graph that is generated solving by the analog computing emulation on the FPGA board

Solving the Rössler equation for 6000 iterations in Matlab (time step size selected is 10^-9, on a 3GHZ Dual Core CPU), takes **6 minutes**, but in hardware, by the emulated analog computing on FPGA with 100 MHz clock, it takes **60µs**. The extremely high speed-up of many orders of magnitude (more than a million) is evident. Fig. 2 shows the XY plot of Rössler equation output.

VI. FUTURE WORKS

In the future, we are going to implement a system for generating the HDL code for solving equations by a flow diagram. For modeling an ODE the future program will be node-based. After coupling nodes by either code or a GUI, the program will be able to generate a gate level HDL code for direct programming on FPGA. By this technique we can speed up the design and implementation process of analog computing solvers on FPGA, which will be capable of solving complex ODE equations and simulating complex systems in real time on FPGA.

VII. CONCLUSION

Each 18-bit integrator takes 1% of the FPGA resources, and each multiplier takes 2% resource. In Xilinx Virtex-5 family there is a powerful chip (XC5VSX240T), which does fit very well to the requirements of our analog computing emulation. This chip has 1k DSP48 slices, what is quit good for implementing integrators and multiplexers. The FPGA synthesis program can combine one integrator and one or two multipliers and adders in each DSP48. By this way we can implement many integrators, adder and multiplier on a single FPGA chip. The obtained results are very encouraging, a speed-up or more than a million has been reached when compared to Matlab performance on a normal PC.

REFERENCES

[1] B. F. Mullins, “A Guide to Solving Simple Ordinary Differential Equations (ODE’s)”, Technical report, University of Toronto, Department of Physics. (Last access June 2009: http://www.physics.utoronto.ca/~poptor/Com00/ODEGuide.pdf)

[2] Bruce J. MacLennan, “Review of Analog Computing”. Technical Report, Department of Electrical Engineering & Computer Science University of Tennessee, Knoxville (2007).

[3] G. E. R. Cowan, R. C. Melville and Y. P. Tsividis, "A VLSI analog computer/digital computer accelerator," in IEEE Journal of Solid-State Circuits 41(1), 42-53 (2006).

[4] J. Lee, B. J. Sheu and R. Chellappa, "A VLSI neuroprocessor for image restoration using analog computing-based systolic architecture," in The Journal of VLSI Signal Processing 5(2), 185-199 (1993).

[5] O. A. Reichardt, M. W. Hoyt, W. T. Lee, "The parallel digital differential analyzer and its application as a hybrid computing system element," in Simulation 4(2), 104 (1965).

[6] K. Sivaranjani, J. Venkatesh and P. A. anakiraman, "Realization of a Digital Differential Analyzer using CPLDs," in International Journal of Modelling and Simulation 27(3), 280 (2007).

[7] D. Phanthavong, ”Designing with DSP48 Blocks Using Precision Synthesis“, Mentor Graphics Corporation, Xcell Journal (2005).

[8] A. Stoica, X. Guo, R. S. Zebulum, D. Keymeulen, and M. I. Ferguson, "On-chip evolutionary synthesis of reconfigurable analog computing circuits," Internet Source, 2003. Last Access June 2009. Source: http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/6434/1/03-0281.pdf