An Efficient FPGA Matrix Multiplier for Linear System Simulation

Sam Mish, John Zenor and Roy Crosbie
Dept of Electrical and Computer Engineering,
California State University, Chico
jzenor@csuchico.edu

Keywords: FPGA simulation, linear systems, matrix multiplication.

Abstract

FPGAs are increasingly being used as a key component in cost-effective high-performance simulations. Most of these efforts have been application specific requiring a custom design. Coding an FPGA can be very challenging but simple, non-optimal configurations are possible using graphical tools such as the Simulink blockset. Greater complexity and improved performance can be gained using a Hardware Description Language (HDL) such as VHDL or Verilog.

The problem of producing an FPGA-based general-purpose simulation system is considerably more demanding. The design of FPGA solutions that perform complex mathematical operations efficiently is difficult and compile times can be very long. One approach to providing this capability to simulation designers who are not expert FPGA programmers is to use pre-compiled FPGA structures driven by data representing a particular model.

A first step is described which provides for the solution of systems of linear differential equations. Non-linear features will be added later. The key to this first step is a flexible, efficient FPGA implementation of a matrix multiplier. An approach is described that improves on previously published methods; that can handle matrices of any size and up to over 100*100 on a single device; that offers flexibility in the choice of FPGA resources, such as DSP (digital signal processor) slices or LUTs (look-up tables); and is linearly scalable in its use of these resources.

The method is used as part of an implementation of a multi-rate benchmark representing an unmanned underwater vessel. The matrix multiplier is used in the simulation of the 3-phase induction motor drive of the vessel. Non-linear product terms are also handled in this model. Non-linear features will be added in future in the form of libraries of non-linear elements.

1. INTRODUCTION

The potential for using field-programmable gate arrays (FPGAs) for a range of scientific computing applications, including simulation, is becoming increasingly recognized. The FPGA offers a reconfigurable architecture capable of dramatically greater numbers of parallel operations than conventional general-purpose processors (GPPs). Modern FPGAs provide support for floating-point operation unlike their predecessors, and although they use clock frequencies that are approximately an order of magnitude slower than GPPs their flexibility and highly parallel structure make them competitive in many applications.

One application in which they have proved effective is in high-speed real-time simulation for which they can provide a computing environment in which run times can be controlled precisely with simulation frame times in the hundreds of nanoseconds as part of real-time multi-rate simulations [1-3].

Successive families of FPGAs are providing ever increasing functionality and support for using clusters of devices to tackle larger problems. There is however a downside to the use of FPGAs: it is the difficulty of generating efficient code that minimizes both execution times and the utilization of FPGA resources.

One approach is to use fixed program structures which have been carefully optimized to solve a specific class of problems and which are data-driven at run-time using data that is generated off-line and that defines a particular problem. This approach is particularly attractive for simulations of linear systems, which can be reduced to repeated calculation of matrix-vector products. A key requirement in this case is a flexible, scalable, efficient algorithm for performing this operation. Earlier implementations were limited to small (up to 12*12) matrices and were relatively inefficient in the use of resources, but they proved effective in legitimizing the approach. Given the need for greater efficiency and scalability, the algorithm was completely redesigned.

The process began with an analysis of the previous design, to preserve its valuable features while improving it in areas of weakness. One feature to be retained was row-wise parallelism, in which each component of the resultant vector is computed independently. Consequently it is only necessary to design an efficient algorithm for a single, vector inner product (copying the algorithm for each row). This structure is also readily generalized to any number of rows and columns, to accomplish the goal of scalability.

A literature search of hardware designs for computing reductions (inner products are a type of reduction) revealed two papers by Prasanna, Morris and Zhou [4,5] that were particularly useful, since they analyzed a variety of reduction algorithms as applied to FPGAs. Their Fully Compacted Binary Tree (FCBT) method was used as a starting point, and was developed to better suit the specific needs of the Chico research.

2. LINEAR DIFFERENTIAL EQUATION SOLVER

A key factor in the effective use of the FPGA-based linear differential equation solving software is the preparation of the data that drives it. The process begins with either a schematic diagram of the system (such as a Simulink diagram) or the system differential equations, which may include non-linear terms. If a schematic is the source, the Mathematica program [6] first extracts the differential equations. The linear equations are then separated from the non-linear equations and are combined with the selected integration algorithm to produce a set of linear difference equations to be solved. Mathematica allows this to be done symbolically and can produce printable report documentation. Numerical values are entered and the matrix coefficient values are calculated and transferred to a file.

The current configuration can execute the simulation in a Matlab/Simulink environment in which case the data is picked up by Matlab and transferred to the FPGA where it is organized and stored in RAM modules by a part of the FPGA code known as the Dense Matrix Module.

The core calculations consist of sum of product calculations of the form

$$\sum A_{jk} * B_k \text{ where } j,k = 1,2\ldots\ldots..N.$$

Processes of this kind in which a stream of data is converted into a single item are known as reductions and the FCBT method offers an efficient way to perform them. In the original FCBT approach, the module used a series of $\log_2(n)$ buffer stages to store intermediate answers, and a scheduling system to manage when and where to keep each piece of data. While this method avoids collisions and produces correct answers, it uses considerable resources and only achieves optimal performance for large problem sizes.

By exploiting the associativity and commutativity of addition, it proved possible to improve on this method in simplicity, performance, and resource usage for the calculations required in a matrix-vector product.

Because the order of additions is inconsequential, it is not necessary to use an elaborate scheduling system, or multiple buffer stages. Instead, by pushing and popping operands in a common buffer, it is possible to develop a simpler control scheme that can achieve optimal performance without collisions. Another consequence is that the necessary buffer size proves to be independent of the problem size, giving an $\Theta(1)$ resource footprint, as opposed to $\Theta(\log_2 N)$ with the FCBT.

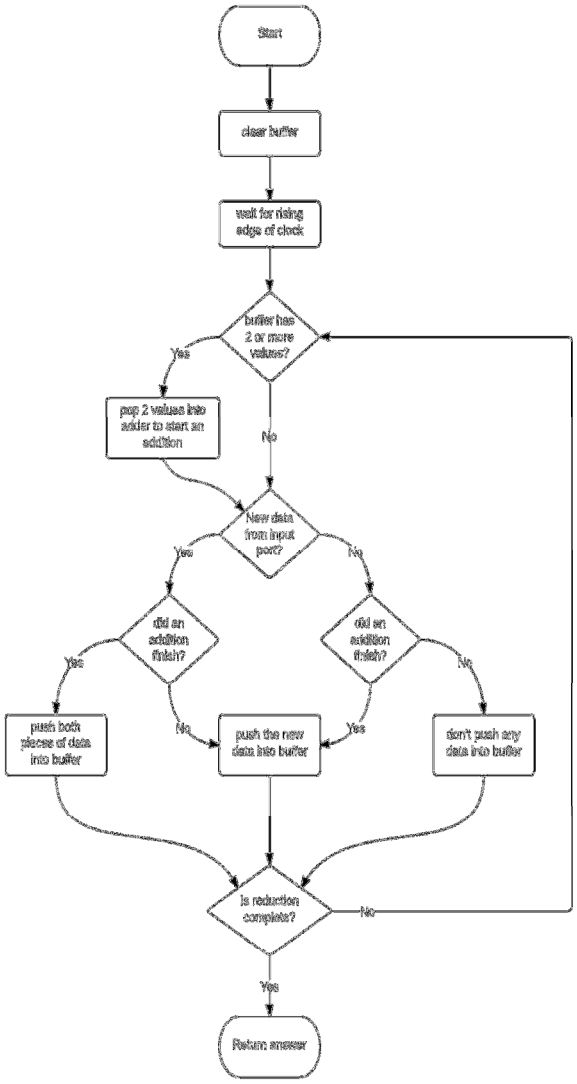

Figure 1: Flow Diagram of Reduction Process

The processing of the data during the execution of the simulation is performed by a module known as the Inner Product Module. This includes one 32-bit floating-point multiply pipeline for each row of the matrix to calculate the scalar products $A_{11}*B_1$, $A_{12}*B_2$ etc in the pipeline for Row 1, $A_{21}*B_1$, $A_{22}*B_2$ etc in the pipeline for Row 2 and so on.

Since all the data for these multiplications is available from the start of the process, the pipelines remain full until all the data has been entered. The products emerge several cycles after entry depending on the pipeline depth, which is selectable by the user. As soon as these products start to emerge from the multiply pipelines they are transferred into a common buffer that feeds the addition pipeline for each row of the matrix. Initially the addition will be performed on individual scalar products (e.g. $A_{11}*B_1 + A_{12}*B_2$) but as the partial products emerge from the addition pipelines they are returned to the input buffer and re-input as quantities to be added. The control scheme ensures that whenever two quantities are available, whether they are simple products or partial sums, they are entered into the adder as soon as possible. At the completion of each step the new values of the system states and inputs are available to be used in the next integration step. A flow chart representing the process is shown in Fig. 1.

2.1. Performance Issues

Performance can be measured in terms of both execution speed and the quantity of FPGA resources required. Limitation of the reduction process to operations that are associative and commutative is a key factor in that it means that the required buffer size is independent of matrix size. The flexibility of design available with FPGA cores means that the length of the pipelines used can be optimized. Execution time is the product of the clock period and the number of clock cycles required

Time = Period*No of cycles = Period/Frequency

Unlike conventional general purpose processors (GPPs) the FPGA clock frequency is adjustable. The possibility of pipeline stalls places restrictions on its maximum value that depend on the depth of the arithmetic pipeline, α, and on the way in which the pipeline is implemented using FPGA resources.

In terms of execution time, two factors must be considered. The greater the pipeline depth, the longer its latency, but deeper pipelines can operate at higher clock frequencies resulting in more but shorter cycles. Figure 2 shows how clock speed depends on pipeline depth or latency for three different designs assuming single-precision, 32-bit data. It shows that clock frequencies can be increased if pipeline depth is increased. For the Virtex 6 this varies between approximately 100 and 450 MHz for latencies between 2 and 12.

Fig. 3 shows that for all but very small matrices the number of clock cycles required rises linearly with problem size. A 50*50 matrix, for example, requires less than 100 cycles. Clock frequencies of 100 to 400 MHz imply execution times for a 50*50 matrix ranging from 250 nS to 1 μS.

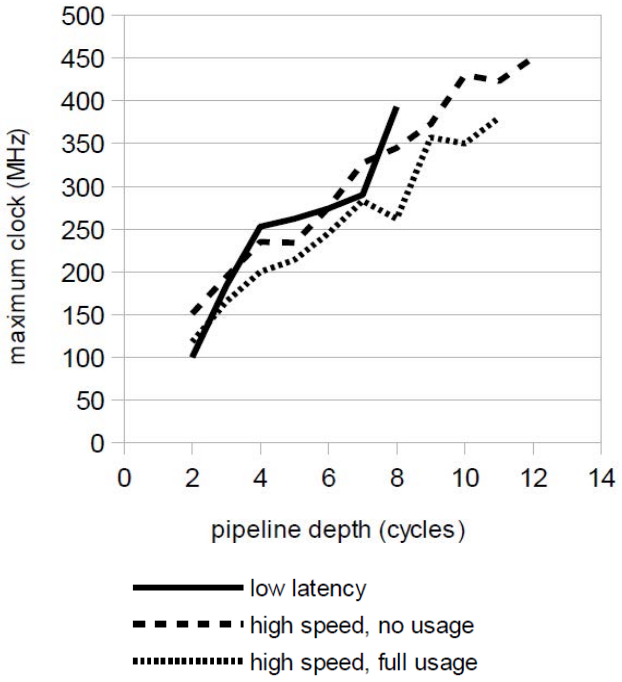

Figure 2. Maximum Clock Speed vs Pipeline depth

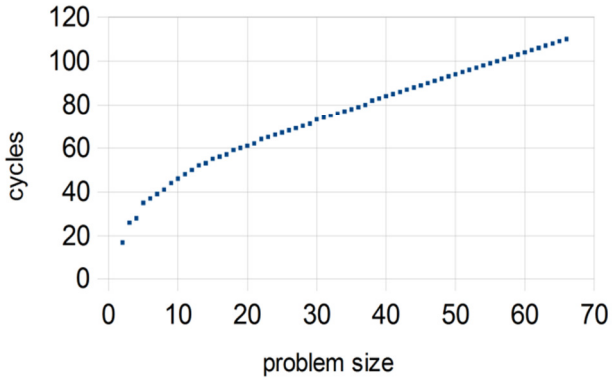

Figure 3: Matrix Multiplication:
Number of Clock Cycles vs Problem Size (Matrix Size)

Combining the data in Figures 2 and 3 for a given problem size of 64 produces the results shown in Figure 4. This shows that for N=64 the expected execution time falls in the approximate range of 200 to 500 nS. Faster execution corresponds to larger pipeline depths and is relatively independent of the way FPGA resources are used, leaving the designer free to make the most convenient choice.

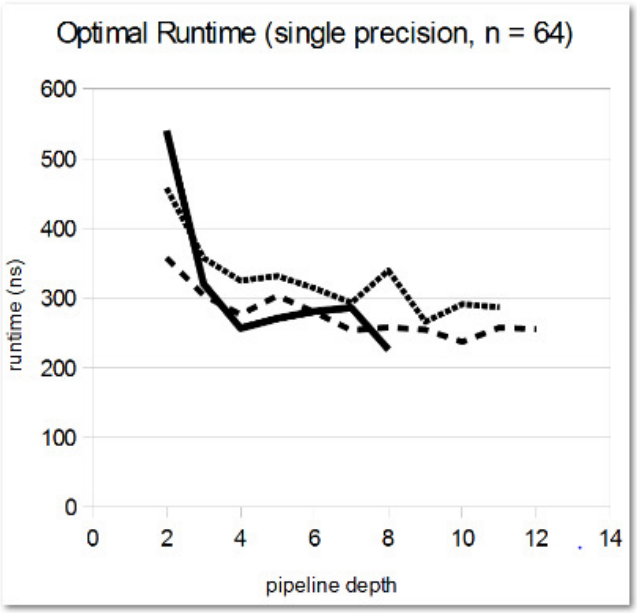

Figure 4: Optimal Runtime for N=64

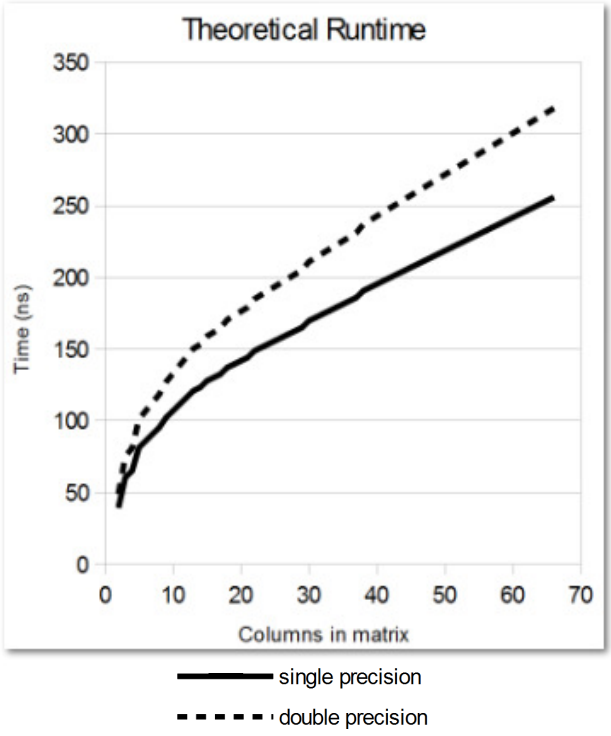

Figure 5: Theoretical Runtimes vs Matrix Size

Figure 5 shows the theoretical minimum runtime for different matrix sizes for both single and double precision. Note that the time penalty for using double precision is relatively small, of the order of 20-30%.

3. EXAMPLE APPLICATION

At the time of writing the algorithm has been checked for correctness both in simulations and in simple test programs in the FPGA. It has also been incorporated in a simple example for demonstration purposes.

The example consists of a model of an induction motor (6 differential equations) with a sinusoidal 3-phase drive voltage and a constant torque load running in a Matlab/Simulink environment. The inputs are currently generated on a PC so timing measurements are not very meaningful. It is planned to introduce a more realistic motor drive including a power electronic converter into the FPGA. Full testing of the algorithm will require a more extensive implementation that supports models involving considerably larger matrices.

4. FUTURE WORK

There is considerable work still to do to fully determine the potential and capabilities of the new algorithm. An exhaustive investigation involving tests with a variety of matrix sizes and design choices is planned.

A linear differential equation solver can only form part of a general purpose simulation system and further work is planned to investigate techniques for combining it with non-linear components.

The Mathematica software is being extended to include a wider selection of integration algorithms and to support non-linear elements.

Efforts so far have been carried out using Virtex 6 FPGAs and it is planned to port the software to the Virtex 7 once its capabilities have been fully investigated.

5. SUMMARY AND CONCLUSIONS

FPGAs offer many advantages as alternate processing units for cost-effective simulations. The key to their efficient use is the development of user friendly and efficient software and algorithms to ensure that their full potential is delivered to users who are not expert FPGA programmers. An efficient scalable matrix multiplier is one essential component of such a system. The use of double-precision arithmetic appears to be feasible at modest cost in terms of runtime.

6. ACKNOWLEDGEMENT

The authors wish to acknowledge the financial support for this research provided by the US Office of Naval Research.

References

[1] Word, D., J.J. Zenor, and R. Powelson, “Using FPGAs for Ultra-High-Speed Real-Time Simulation” *Proceedings of Conference on Grand Challenges in*

Modeling and Simulation, Edinburgh, Scotland, June 16-19, 2008.

[2] Word, D., R. Bednar, J.J. Zenor, and N.G. Hingorani, "High-Speed Real-Time Simulation for Power Electronic Systems. *SIMULATION: Transactions of the Society for Modeling and Simulation International,* Aug 2008, Vol 84, pp 441-456.

[3] Crosbie, R.E., J.J. Zenor, D. Word, R. Bednar and N.G. Hingorani, *"FPGA-Based Real-Time Simulation of Power Electronic Systems",* Proceedings of 7th Eurosim Congress, Prague, Czech Republic, Sept. 5-10, 2010.

[4] Zhuo, L., G.R.Morris and V.K.Prasanna, "Designing Scalable FPGA-Based Reduction Circuits Using Floating-Point Cores", *Proceedings of 12th Reconfigurable Architectures Workshop,* (RAW 2005), Denver, Colorado, April 4-5, 2005, IEEE-CS

[5] Zhuo, L., G.R.Morris and V.K.Prasanna, "High-Performance Reduction Circuits Using Deeply Pipelined Operators on FPGAs", *IEEE Transactions on Parallel and Distributed Systems,* Vol. 18, No.10, October 2007, pp1377-1392.

[6] Zenor, J.J., R.E.Crosbie, R.Bednar, D.Word and N.G.Hingorani, "Software Support Tools for High-Speed Real-Time Simulations", *Proceedings of the International Conference on Grand Challenges in Modeling and Simulation,* Istanbul, Turkey, July 13-16, 2009, SCS.

Biographies

Sam Mish is an Applied Mathematics and Physics undergraduate student at California State University, Chico. In addition to FPGA development, he is involved in writing finite element simulations for graphical processing units, supervised machine learning and recognition algorithms, and the IEEE Micromouse competition. In his spare time, he volunteers as a sports photographer for the university, and plays brass in some of the school musical ensembles.

Roy Crosbie received his B.Eng and Ph.D. degrees in Electrical and Electronic Engineering from the University of Liverpool, UK. He is a Chartered Engineer (UK). He has experience with the Marconi Co. and Bell Canada. He was on the Electrical Engineering faculty at the University of Salford England for 20 years. He established a Simulation Laboratory at Salford and developed one of the first ever MS programs in Computer Simulation (in 1970). Jointly with the late Dr. John Hay he developed the ISIS, ISIM and ESL simulation languages. In 1968, he helped to found the UK Simulation Council, later the UK Simulation Society and was elected to the Executive Committee of SCS in 1972. He joined CSU, Chico in 1983 and served in the Departments of Computer Science, Computer Engineering and Electrical and Computer Engineering. He was the first Chair of the Dept of Computer Engineering. He helped to create the first center of the McLeod Institute of Simulation Sciences at Chico in 1986. He was President of SCS from 1988-90. He is a holder of the SCS Presidential Award for Service to the Society, a Fellow and Life Member of SCS and a Fellow of the Institution of Engineering and Technology (UK). In 2001 he initiated a research program on high-speed real-time simulation aimed initially at power electronic systems. This research has been supported by the Office of Naval Research since 2004.

John Zenor received a PhD in Geophysical Engineering from the University of Missouri at Rolla, Mo. in 1968, MS in Computer Science, 1966, B.S. in Applied Mathematics in 1963. He spent twenty years at the Naval Air Warfare Center, China Lake, CA in the areas of real-time simulation, software engineering, and tool development and twenty-five years at California State University, Chico, CA where he is currently Professor Emeritus in the Department of Electrical and Computer Engineering specializing in real-time computing and computer networking.