11-23-2010

A Bit Serial Approach to Massively Parallel Floating Point Operations on an FPGA

Duane Marcy
Syracuse University, dlmarcy@syr.edu

Fred Schlereth
Syracuse University, schleret@syr.edu

Parija Kshirsagar
Syracuse University, pkshirsa@syr.edu

Anvith Katte Mahabalagiri
Syracuse University, akattema@syr.edu

Department of Electrical Engineering and Computer Science

Technical Report

SYR-EECS-2010-07 Nov. 23, 2010

A Bit Serial Approach to Massively Parallel Floating Point Operations on an FPGA

Duane Marcy dlmarcy@syr.edu
Fred Schlereth schleret@syr.edu
Parija Kshirsagar pkshirsa@syr.edu
Anvith Katte Mahabalagiri akattema@syr.edu

ABSTRACT: In this paper we discuss the pros and cons of bit serial arithmetic for performing mathematical operations for signal processing and scientific computations on an FPGA. We describe our formulation of the architecture for such massively parallel systems, the advantage being that it requires no parallel programming in the traditional sense. We describe a pseudo floating point bit serial circuit which is less complex than full precision floating point and show that it is suitable for many applications. We conclude with several application examples and show that a bit serial implementation can be competitive with a high speed parallel implementation.

KEYWORDS: Bit Serial, Floating Point, FPGA, Parallelism, State Space Formulation

Syracuse University - Department of EECS,
4-206 CST, Syracuse, NY 13244
(P) 315.443.2652 (F) 315.443.2583
http://eecs.syr.edu

A Bit Serial Approach to Massively Parallel Floating Point Operations on an FPGA

Duane Marcy, Fred Schlereth, Parija Kshirsagar, Anvith Katte Mahabalagiri
EECS Department, Syracuse University

***Abstract* - In this paper we discuss the pros and cons of bit serial arithmetic for performing mathematical operations for signal processing and scientific computations on an FPGA. We describe our formulation of the architecture for such massively parallel systems, the advantage being that it requires no parallel programming in the traditional sense. We describe a pseudo floating point bit serial circuit which is less complex than full precision floating point circuit and show that it is suitable for many applications. We conclude with several application examples and show that a bit serial implementation can be competitive with a high speed parallel implementation.**

***Index Terms* - Bit Serial, Floating Point, FPGA, Parallelism, State Space Formulation.**

I. INTRODUCTION

SIGNAL processing involves operations such as FIR and IIR filtering, modulation/demodulation, each of which has a well defined data flow and can be easily programmed on an FPGA. Major advantages of FPGA implementations are highly parallel operations and data memory that is local to the processors eliminating the power and time needed for data transfers to and from external memory. Many such signal processing operations are well suited for the use of fixed point data types. Scientific computing, on the other hand, requires full precision floating point for tasks such as the inversion of large matrices, solution of boundary value problems requiring many iterations, and signal processing tasks in which it is difficult to predict the dynamic range of the variables. However there is a set of signal processing applications (described below), where partial (pseudo) floating point (pFP) can be very effective. The advantage is that pFP has much lower cost than full precision floating point. However, pFP is suitable only for a subset of the scientific computations.

Our interest in bit-serial processing arises from a desire to construct massively parallel processors (thousands of elements). We demonstrate that this is feasible and cost effective with current FPGA technology using a parallel architecture modeled after analog computers.

In this paper we give some general background showing our approach to the design of FPGA-based computing circuits and systems, a design of a pFP processor and a plan for the design of a full floating point processor. A non-linear pendulum example is described showing our approach to the discretization of such systems. We also describe the problems involving matrix operations, and finally discuss the pros and cons of high speed processing using our FPGA approach to that based on the NVIDIA approach [1], as well as other approaches [2,3,4,6].

II. GENERAL BACKGROUND

In our signal processing work we model the circuits in terms of a Signal Processing Object, SPO, as shown in Fig.1. This object is analogous to an analog operational amplifier, and we build parallel systems using SPOs in the same way that analog OPAMPS are interconnected to realize signal processors.[7] E.g., SPOs can be interconnected to solve difference equations in the same way that OPAMPS can be interconnected to solve differential equations. In the case of the former we add a delay to form a multiplier accumulator circuit, and in the case of the latter we add a capacitor to form an integrator. A major advantage is that this approach requires no parallel programming in the traditional sense.

Fig.1 shows a digital circuit (SPO) with enough structure to permit it to be interconnected in large arrays where the interconnections among SPOs are determined by the difference equations for the desired operation. If the goal were to limit the design to standard FPGA cells, then the structure of the SPO would not be so important, because it would be an easy matter to modify the structure to still be useful in a wide range of applications. However as will be seen below, we are also interested in defining an SPO structure that can be committed to a hard IP core. Our experience has shown that the SPO shown in Fig.1 is able to meet virtually all common signal processing applications. .

Some of the features of a fixed point SPO implementation are that all operations are performed in full precision, with rounding just prior to the multiplication. The internal summers provide interconnect for larger arrays of SPOs and a single word memory is provided for data storage. The multiplier provides additional storage because, on an FPGA, a multiplier must have a delay. Another important advantage is that

Manuscript received November 22, 2010.

Dr. Duane Marcy is with the Department of Electrical Engineering and Computer Science at Syracuse University, Syracuse, NY 13244 USA (e-mail: dlmarcy@syr.edu).

Dr. Fred Schlereth is with the Department of Electrical Egineering and computer Science at Syracuse University, Syracuse, NY 13244 USA (e-mail: schleret@syr.edu).

memory is local to the processor eliminating the need for extensive data transfers during processing.

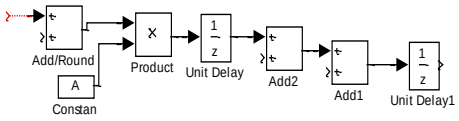

Fig.1 Signal Processing Object (SPO)

The figure below shows an implementation of a second order filter. From this figure it should be clear that large parallel systems can be "programmed" by just "wiring up" groups of SPOs according to the dictates of the difference equation. With this approach to parallelism, there is no traditional programming step, as is required in the implementation of such circuits using DSPs.

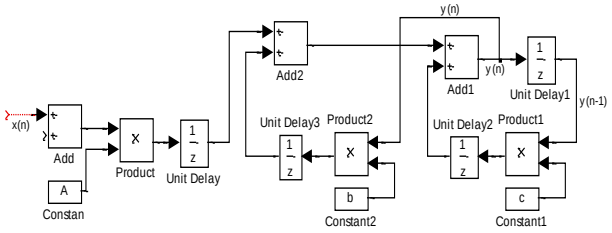

Fig. 2 Second Order Filter
y(n) = a*x(n-1) + b*y(n-1) + c*y(n-2)

Thinking of the implementation of standard filters in this manner provides a convenient and economical way to characterize large systems. It also suggests that for the implementation of very large systems, it would be advantageous to build a hard IP core for SPO, using custom integrated circuits. Of course the SPO could be realized in fixed or floating point. Fig. 3 shows a block diagram of an FPGA with a SPO core.

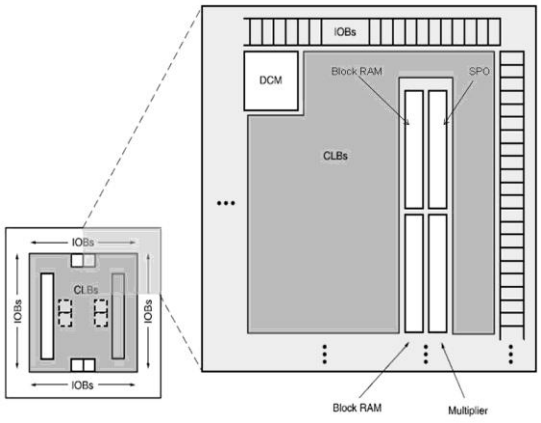

Fig.3 SPO on an FPGA

As mentioned above, our interest in bit serial arises out of the desire to build systems with number of processors which are an order of magnitude greater than possible with present implementations; i.e., thousands rather than hundreds of processors. One particular application of interest is the simulation of large transmission lines on integrated circuits which can include resistance, inductance and capacitance in the model. Another application of interest is modeling a transistor. Using a simple state space formulation we can perform simulations with the accuracy of SPICE [8], but at much higher speed.

In many signal processing applications it is advantageous to treat the data word differently than the coefficient word. For example, in a filter, the coefficients determine the location of the poles and zeros, and the data determines the dynamic range. A bit serial implementation allows us to take advantage of this by designing a pFP circuit, where the timing of the multiplier coefficient, relative to the data is used to scale the coefficient. This permits us wide latitude in the values of the coefficients without the need to store leading zeros. This circuit, a pFP, is described next. An estimate of the resources used indicates that several hundreds of such circuits would fit onto a small FPGA.

III. PSEUDO FLOATING POINT BIT SERIAL MULTIPLIER ACCUMULATOR

In applications which require large numbers of multiplier accumulators (MAs) and at the same time can tolerate low speed for each of the MAs, a viable option is to use bit serial arithmetic. This is easily implemented on an FPGA and has the advantage of small size. The slower speed of a bit serial MA is mitigated by smaller area requirements so that in applications requiring large numbers of MAs, tradeoff studies are needed to find the best combination time-area-power product. These studies should consider bit serial MAs using the FPGA cells or a hard IP core for a 'Bit Serial Multiplier Accumulator SPO'.

As a step in that direction, we have implemented a Pseudo Floating Point Bit Serial MA which treats the data and coefficient differently. E.g., if the data word is set to 64 bits, to provide good dynamic range, while the coefficient is limited to 16 bits. Then as the data is passing through the SPO the coefficient is scaled by shifting its bit position relative to the data.

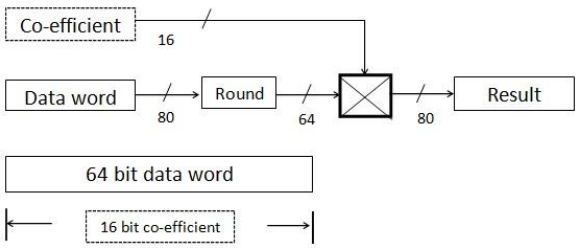

Fig.4 Pseudo Floating Point Bit Serial Multiplier

The advantage is that the dynamic range of the coefficient can be greatly extended without having to store large number of leading zeros in the case of small coefficient values. Fig.4 shows a block diagram of this MA. It uses minimal resources, and preliminary estimates indicate that the number of such circuits that could be available on Vertex-II FPGA if it were realized as hard IP core would be at least several hundred.

IV. EXAMPLES

In the following sections we present examples of our work in the application of bit serial approach to large problems using FPGA implementation. One is the simulation of a simple driven pendulum, another is circuit simulation and the third is GORDIAN.

A. Simple Driven Pendulum - State Space Formulation

MATLAB provides a number of tools to convert among various representations of systems. In our work we have focused on the formulation of standard differential equations, and their implementation on an FPGA. In the following example we use a state space formulation to derive the block diagram for computations on an FPGA [5].

The forced pendulum is one of many examples of chaotic [9] behavior and it is of interest to perform simulations for long periods of time. It is here that the parallel implementation on the FPGA shows its efficacy. Using parallelism, coupled with floating point arithmetic and discrete approximations with guaranteed stability, it is possible to study very long term events.

Consider the differential equation for the forced pendulum:

$$mL\frac{d^2y}{dt^2} + \gamma sin(y) + mg\frac{dy}{dt} = Acos\,(bt)$$

Where,
y = angle with respect to the vertical
m = mass of pendulum
L = length of pendulum
γ = damping
g = acceleration due to gravity
A = driving force amplitude
b = driving force frequency

By choosing the scaled units, the equation is simplified to,

$$\frac{d^2y}{dt^2} + sin(y) + a\frac{dy}{dt} = cos\,(bt)$$

let, $x_1 = y$, $x_2 = \frac{dy}{dt}$

$$\frac{dx_1}{dt} = x_2\,,\ \frac{dx_2}{dt} = -\,sin(x_1) - ax_2 + cos(bt)$$

$$\begin{bmatrix}\frac{dx_1}{dt}\\ \frac{dx_2}{dt}\end{bmatrix} = \begin{bmatrix}0 & 1\\ \frac{-sin\,(x_1)}{x_1} & a\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}[cos\,(bt)]$$

Where,

$$A = \begin{bmatrix}0 & 1\\ \frac{-sin\,(x_1)}{x_1} & a\end{bmatrix},\ B = \begin{bmatrix}0\\1\end{bmatrix},\ u = [cos\,(bt)]$$

Here, the 'A' matrix contains a term sin() which illustrates the manner in which the equations will be programmed on the FPGA. A discrete version of this equation is as follows,

$$\begin{bmatrix}x_1(k+1)\\ x_2(k+1)\end{bmatrix} = e^{AT}\begin{bmatrix}x_1(k)\\ x_2(k)\end{bmatrix} + (\int_0^T e^{Av})dvBu[k]$$

$$\int_0^T e^{Av}dv = \frac{e^{AT}-1}{A} \approx \frac{1+AT-1}{A} = T$$

$$\begin{bmatrix}x_1(k+1)\\ x_2(k+1)\end{bmatrix} = e^{AT}\begin{bmatrix}x_1(k)\\ x_2(k)\end{bmatrix} + T * Bu[k]$$

The choice of the approximation for e^{AT} has several important considerations. (1+AT), the forward Euler method, leads to a realizable FPGA implementation in which all state variables depend only on values in the previous time step. However this approximation is unstable for stiff systems of equations. The approximation $(1\text{-}AT)^{-1}$ is unconditionally stable but not realizable unless the inversion is performed prior to the simulation. Herein lye several interesting implementation issues.

If the inversion is performed prior to the simulation, then FPGAs can be easily programmed to provide the simulation. However the A matrix becomes dense and the coefficients will have a large dynamic range. It is here that the pFP circuit is useful.

If, however, the system of equations is nonlinear, then the inversion must be performed at every time step. In this case an iterative solution to the inversion using the following formulation would be feasible. I.e., the A matrix is a function of the dependent variables. In this case A' needs to be computed at every time step. We use an iterative procedure to update A',

$$A'(k) = A'(k\text{-}1) * c * [I - A_{new}{}^{*}A'(k\text{-}1)]$$

Where, $A' = (1 - AT)^{-1}$ and A_{new} is the modified value of A due to the action of the nonlinearities. The iteration should take only a few steps since the changes in A will be small from one time step to the next. However in this operation, it is necessary to use full precision floating point because A'(k) (coefficient) is changing every time step.

Fig.5 shows a Simulink (Xilinx Blockset) model of the pendulum using the state space formulation. Note the use of a ROM to provide sine of the state variable. It is estimated that a table in the order of 256x 8 bit will be sufficient.

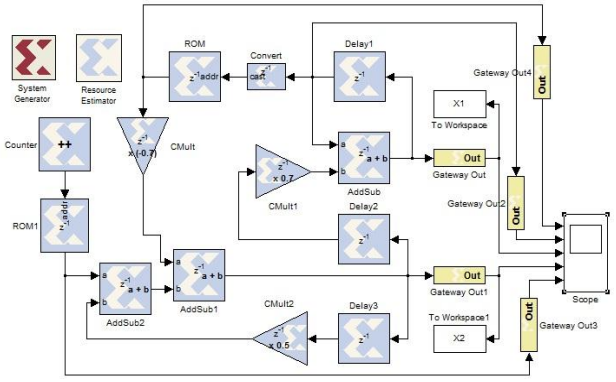

Fig.5 Simulink Model of a Driven Pendulum

B. *Integrated Circuit Simulation*

Another example is Integrated Circuit Simulation. The transmission line is a good model for data lines on sub micron integrated circuits, where RC models for transmission lines are no longer accurate enough. Using this technology we are able to simulate very long lines with high accuracy in a fraction of the time required by a SPICE simulation.

Here we also use a state space formulation and are able to perform SPICE-like simulations in a fraction of the time needed for simulation on a workstation. We have simulated both linear and nonlinear circuits with good results. This is a very important problem in IC design because it is necessary to have accurate estimates of a pulse waveform at the end of long lines [4, 10]. Fig.6 shows a simplified model.

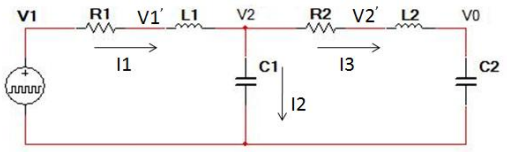

Fig.6 Transmission Line Model

The state space equations are as follows. Inductor currents and capacitor voltages, I_1, I_3, V_0, V_2, respectively, are the state variables.

$$\begin{bmatrix} \dot{I}_1 \\ \dot{V}_2 \\ \dot{I}_3 \\ \dot{V}_0 \end{bmatrix} = \begin{bmatrix} -R/L & -1/L & 0 & 0 \\ 1/C & 0 & 1/C & 0 \\ 0 & 1/L & -R/L & -1/L \\ 0 & 0 & 0 & 1/C \end{bmatrix} \begin{bmatrix} I_1 \\ V_2 \\ I_3 \\ V_0 \end{bmatrix} + \begin{bmatrix} 1/L \\ 0 \\ 0 \\ 0 \end{bmatrix} [V_1]$$

Based on the state equations, the Simulink (Xilinx Blockset) model of the above transmission line can be created following the SPO architecture, as shown in Fig.7. This model can be implemented using bit serial pFP MA, however, if there is any nonlinearity in the circuit, then a full precision floating point MA is needed for its implementation.

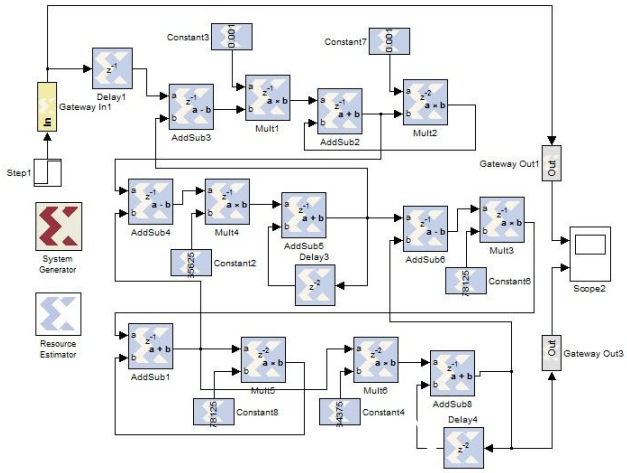

Fig.7 Simulink model of the Transmission Line

C. *GORDIAN Algorithm for VLSI Placement*

Finally we describe GORDIAN algorithm as another application involving large matrix computations, where true floating point is an absolute requirement.

GORDIAN is a widely used algorithm for optimized VLSI module placement [11]. Most of the CAD tools in the VLSI industry are based on this algorithm. The GORDIAN algorithm deals with placement of VLSI modules such that the wire length is minimized. It involves two stages namely Global Optimization and Rectangular Dissection. Our interest lies in the Global Optimization part of the algorithm which involves solving equations with matrices of massive sizes.

The core computation involved in the Global Optimization step is in iteratively solving the equation,

$$\boldsymbol{Z^T C Z x_i^* = -c}$$

Where, $\boldsymbol{Z}$ and $\boldsymbol{-c}$ are constraint matrices, $\boldsymbol{C}$ is the matrix containing distance between movable modules.

At every iteration, a global minimum is calculated for $\boldsymbol{x_i^*}$. With the current VLSI technology moving to 12 nm and below, it is evident that the module count on a chip is of the order of tens of thousands. Thus along with the increase in the size of the matrices, the requirement of floating point operation also comes into the picture. Keeping these factors in mind, the computation of the above matrix equation on general purpose microprocessors would require several days of execution. Using the techniques proposed in this paper, such massive amounts of floating point operations can be performed in parallel using an FPGA. However, data flow poses as the major bottleneck in such applications. If proper data flow techniques are designed to assist the architecture proposed in this paper, an exponential decrease in the computation time can be visualized.

V. CONCLUSION

It is interesting to compare the speed of a massively parallel bit serial processor using FPGA technology with a processor based on NVIDIA chips [1]. It is clear that the NVIDIA system will be faster than our FPGA based system. However there are other considerations which, for many applications, may tip the balance in favor of our approach.

1. *Cost:* Tens of thousands vs. hundreds of dollars.
2. *Programming:* Our approach was conceived from the ground up as requiring no special programming to make effective use of massive parallelism. Each of our processors is "wired" into the proper configuration according to the mathematical description provided by the underlying difference equations or state equations. This has the obvious advantage of saving lots of time but there are other advantages. Debugging is more of a mathematics problem as opposed to a coding problem. Another is that designs based on our approach have the advantage of being provably correct [12]. Again, this is because there is no step between the mathematical formulation and the implementation. In fact in circuit simulation it is feasible to go directly from a netlist to a FPGA simulation.
3. *Speed:* The speed of a processor based on a bit serial FPGA implementation can be estimated as follows. Assume 1000 hard IP-core MAs running at a clock speed of 500 MHz with a 100 bit word. The rate in this case is 5 GFlop/sec.
4. *Reconfigurability:* This is a breeze with our FPGA implementation, and with the latest advances in technology, can even be done "on the fly". The bit serial implementation is particularly advantageous in this regard.

At the present time we are working on the implementation of the double precision floating point circuit and continuing with applications to a variety of problems.

VI. REFERENCES

[1] NVIDIA: A white paper on "NVIDIA's Next Generation CUDA™ Architecture: Fermi™,"

[2] "Floating-Point FPGA: Architecture and Modeling", C.H. Ho, C.W. Yu, et al, IEEE Transactions on Very Large Scale Integration Systems, Vol 17 No 12 Dec 2009.

[3] "Digital Signal Processing with Field Programmable Gate Arrays", U. Meyer-Baese, Springer 2007

[4] "Advanced FPGA Design" Steve Kilts, Wiley, 2007

[5] "Digital Control – A State Space Approach", R.J. Vaccaro, McGraw Hill, 1995

[6] "Floating-Point FPGA: Architecture and Modeling", C.H. Ho, C.W. Yu, et al, IEEE Transactions on Very Large Scale Integration Systems, Vol 17 No 12 Dec 2009.

[7] Jackson, Albert S., "Analog Computation". London & New York: McGraw-Hill, 1960. OCLC 230146450

[8] L. W. Nagel and D. O. Pederson, "SPICE (Simulation Program with Integrated Circuit Emphasis)", Memorandum No. ERL-M382, University of California, Berkeley, Apr. 1973.

[9] "Chaotic Dynamics – an introduction", G.L. Baker, J.P Golub, Cambridge, 1990.

[10] "Analysis of On-Chip Inductance Effects for Distributed *RLC* Interconnects, K. Banerjee, A. Mehrotra, IEEE Tranasaction on Computer-Aded design of Integrated Circuits, Vol 21, No 8, August 2002.

[11] "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization', IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 10, Issue 3, pg. 356-365.

[12] "Access Control, Security and Trust" Shui Kai Chin, Susan Older, CRC Press 2010.