A Variable Long-precision Arithmetic Unit Design suitable for Reconfigurable Coprocessor Architectures

Alexandre F. Tenca*
tenca@cs.ucla.edu

Miloš D. Ercegovac
milos@cs.ucla.edu

Computer Science Department
University of California, Los Angeles

Abstract

This paper presents the organization of an arithmetic unit for variable long-precision (VLP) numbers suitable for reconfigurable computing. The reconfigurable arithmetic coprocessor (RAC) cooperates with the host computer in the VLP tasks. The main design issues addressed in the paper are: (a) mapping of the most frequent and time consuming operations of the VLP arithmetic algorithms to RAC, and (b) design of VLP algorithms that allow reduced reconfiguration time between arithmetic operations. The VLP arithmetic algorithms proposed cover multiplication, division and square-root. In this paper we present the main building blocks used in the VLP arithmetic circuits, show the similarities of each arithmetic operator and present some implementation results of these circuits in Xilinx FPGAs.

1 Introduction

Reconfigurable architectures have been used to speed up the execution of many computer applications [1, 5, 2, 14]. Part of the application tasks that are implemented in software are transferred to the reconfigurable hardware in order to execute the whole task at a higher performance level by tuning the hardware resources to the algorithm and by avoiding the software overhead typically present in conventional architectures. The speedup is a function of many factors, but the most important one is that the downloaded task must correspond to a significant part of the overall computation. Otherwise, the speedup remains small (Amdahl's Law). This factor forces the careful selection of tasks that can be successfully downloaded to the reconfigurable hardware.

The reconfiguration time of programmable devices is relatively large. So, if it is necessary to make many reconfigurations during a computation, the performance of the reconfigurable hardware is greatly affected. Recent programmable devices allow partial reconfiguration which favors designs that have many common parts and may be easily modified to perform multiple tasks.

Variable Long-Precision (VLP) computation is one application that we believe has properties desirable for a successful implementation in reconfigurable hardware. The kernel of a VLP computation is composed by repetitive and time-consuming operations. Only these operations should be implemented on FPGAs. Other tasks would still be executed at the host level and for this reason we consider using reconfigurable hardware in a host+coprocessor architecture. Since our emphasis is on the arithmetic operations, we call our approach Host+Reconfigurable Arithmetic Coprocessor (HRAC) to stress the cooperation between the two computational resources.

Our investigation of arithmetic circuits for VLP computations suitable for FPGAs lead to the development of arithmetic algorithms that

*the author is also with the University of São Paulo and CNPq, Brazil

satisfy the properties listed above [10]. These algorithms deal with the four arithmetic operations: multiplication, division and square-root. VLP addition was not considered since the host processor is very efficient for integer addition, the main component of VLP addition. The operands for the VLP operations are fixed point numbers. The utilization of the coprocessor VLP operations for long-precision floating-point calculations is accomplished by the host processor, in the HRAC architecture.

We develop the paper starting with the explanation of the proposed VLP arithmetic algorithms. Than we show the organization of the coprocessor and the main differences between the main arithmetic operations. We conclude with some experimental results obtained from the implementation of these circuits using Xilinx FPGAs.

2 VLP Arithmetic

When the precision is larger than the one available in hardware, software algorithms are applied to obtain the desired precision, typically considering the hardware precision as corresponding to a single high-radix digit.

Software algorithms for long-precision computations have been developed by many researchers following basic algorithms such as those presented in [7]. These algorithms have an asymptotic time which is always better than the asymptotic time of algorithms used in the hardware implementation of arithmetic coprocessors for long-precision [8, 6]. The reason is the complexity of implementing fast software algorithms for VLP computations directly at the hardware level. Simpler algorithms will take advantage of the hardware speed and will also result in simple and fast components.

As mentioned above, at the conceptual level all VLP algorithms manipulate a processor word as a high-radix digit. The operations are carried digit by digit, serially. For this reason we propose VLP algorithms based on on-line arithmetic [3, 12] where the inputs and outputs are handled serially, most significant digit first. This feature offers several advantages, discussed later, with respect to the conventional, least significant digit first serial arithmetic. We first review the concepts of on-line arithmetic, and later we present and discuss the hardware organization and VLP algorithms based on the on-line arithmetic.

2.1 On-line algorithms

The basic ideas and algorithms of on-line arithmetic are presented in [3, 12] and a design methodology in [4]. As mentioned above, The result digits are produced serially, most significant digit first, after a few cycles (on-line delay δ).

On-line multiplication ($Z = XY$), division ($Z = X/Y$) and square-root ($Z = \sqrt{X}$) are defined by the recurrence equations shown in Table 2, with the following convention for the digit vectors:

$$X[j] = \sum_{i=1}^{j+\delta} x_i r^{-i},$$

$$Y[j] = \sum_{i=1}^{j+\delta} y_i r^{-i} \text{ and}$$

$$P[j] = \sum_{i=1}^{j} p_i r^{-i}, \text{ where } P \in \{W, Z\}$$

where the variable W is called the *scaled residual* and any digit is in the set $\{-a, ..., -1, 0, 1, ..., a\}$, $r/2 \leq a \leq (r-1)$.

The on-line delays in each expression is slightly different and varies depending on the radix used. As we consider the operations in high radix, the on-line delays are 2, 3 and 3, for multiplication, division and square-root, respectively.

Since the on-line results are obtained in redundant form, they must be converted to conventional representation before being delivered to the program. This task, as explained later, is handled by the host.

The scaled residual is kept inside bounds by subtracting the output digit z_j that is selected based on the equations also shown in Table 2. In order to use simple selection functions, the following conditions are considered:

- *in VLP division:* the divisor is pre-scaled to a value close to 1;
- *in VLP square-root:* the radicand must be in the range $0.25 \leq X < 1$ and an estimate of the output (with two fractional digits) must be provided by the host.

The VLP multiplication was explained in detail in [11]. Traditional on-line designs [9, 13] include parallel multipliers to compute digit by vector products ($y_{j+\delta}X[j-1]$, for example) and are implemented usually in radix 2 or 4.

The variables $Y[j]$,$X[j]$ and $Z[j]$ increase in precision as new digits are received or generated. In previous on-line designs using fixed hardware, append registers store the new digit values forming the growing digit vectors. The append registers have a length that corresponds to the precision of the greatest operand. When working with very long precision computations, it is unlikely that there is enough hardware to design the complete unit for the given precision, and for that reason, the use of the traditional hardware structures for on-line operations are not adequate for VLP arithmetic.

2.2 Overview of VLP Algorithms

The objective of this paper is not to present the details of the algorithms for VLP arithmetic but rather show the main features and ideas behind the utilization of the VLP algorithms in reconfigurable hardware. Details of the algorithms can be found in [10].

The VLP algorithms are separated in two main parts, the tasks executed by the host processor and the tasks executed by the reconfigurable coprocessor. The host is responsible for the following tasks:

1. configure the coprocessor for the required operation (when necessary);
2. adjust the operand range. For division it is necessary to multiply the divisor by an approximation of its reciprocal (scaling factor). In square-root, only bit shift operations are needed to scale the radicand;
3. transfer the operand digits, estimate of the output (square-root) and required precision to the coprocessor;
4. convert the output digits generated by the coprocessor into conventional form (on-the-fly conversion (OFC) at the software level);
5. post-correction of the output (required by square-root only);
6. when working with floating-point numbers, normalize the significand and compute the exponent value.

These tasks are done only once in the VLP arithmetic operation and thus should not be mapped onto the coprocessor.

The coprocessor executes the following steps (generalized for all VLP arithmetic operations):

1. initialize the scaled residual value.
2. for each digit of the input operands:
   (a) execute the recurrence equation shown in Table 2 serially;
   (b) select the output digit;
   (c) update the information in the data vectors.
   (d) (VLP Multiplier) perform recoding of the output digit before sending it to the host;
3. (for VLP multiplication) complete the output digit recoding and transfer the remaining digits in the scaled residual vector to the host, performing digit conversion (redundant to non-redundant form) as the data is transferred.

Digit conversions are necessary because fast arithmetic units use redundant digit representations. However, redundant representations use more bits per digit than the non-redundant representations (sign-and-magnitude or two's complement). In order to reduce the number of bits transferred to the host and reduce the computation time during OFC, the coprocessor executes digit conversion as the VLP operation is performed.

When the coprocessor and the host share the same memory, the data transfers are not necessary. The result is left in the shared memory.

The general description of the coprocessor organization is shown in the next section.

3 The Reconfigurable Coprocessor Architecture

The block diagram of the coprocessor is shown in Figure 1. In the diagram we abstract the complexity of the interface between the host and the coprocessor and concentrate on the architectural and design issues of the arithmetic coprocessor. The coprocessor is composed of a control block, data path block and memory elements. The host may be able to initialize values in the local memory and this way simplify the initialization phase. We are omitting details to increase legibility and present only the main concepts.

3.1 Memory Elements

The memory elements are represented as lists. In fact, the data arrays are accessed always linearly. The data values stored in memory elements labeled as X, Y and Z are written only once and not destroyed during the VLP computation. The values on the residual vector W, on the other hand, are rewritten in every iteration (serial computation of the recurrence equation). To avoid I/O limitations, each memory element should be an independent component, that allows read and write operations in each cycle.

The shift operations shown in the recurrence equations (multiplication by r, for example) are done by proper alignment of the data in the memory elements. The alignment must also consider the delays of the data path components. For example, assume that we want to perform serially the computation: $rab + c$, where a, b and c are digit vectors in radix r, and the data path network is composed of a serial multiplier and adder with delays 3 and 1 cycle, respectively. Caused by the multiplier delay and the shift left operation (multiplication by r), the operands a and b should be imposed to the data path 4 clock cycles in advance in order to have the product rab reaching the serial adder at the same time as c. Thus, the digits of c are written 4 digit positions to the right, in respect to the a and b digits. This data organization reduces the control block complexity, which may use the same pointer to read digits from all data vectors.

3.2 Control Block

The control block is reconfigured depending on the type of arithmetic operation being executed in the coprocessor. This unit is responsible for data addressing and read/write control over the memory elements, and also the control over the data registers in the data path. It can be implemented in a FPGA device or a microcontroller. The use of the microcontroller would be justified by the low level of parallelism of the control circuit.

The control block coordinates the actions of the coprocessor according to the steps described in Section 2.2. There are some important pointers that are used and maintained by this block:

- precision of the input operands (m);
- pointer to scan the digit on the memory elements (p). It is the same for all memory elements.
- pointers to digit positions in W (pw) and Z (pz) that should be written in the current iteration;
- pointer to the digit positions in X and Y that are the present input digits (op).

Pointer p starts from zero and is incremented during the serial computation of the recurrence equation, until it reaches $op + disp$, where $disp$ corresponds to the displacement caused by the alignment of digit vectors, as explained before. Pointer pw follows p and the distance between them is a function of the delay in the data path (in clock cycles). By the end of each iteration, the values of pz and op are incremented. The stop condition is detected when $op = m$.

3.3 Data Path

The data path is shown in Figure 2. It has the following basic components: digit-by-vector serial multiplier, on-line adders/subtractors, selection function, converter, and registers. The

digit-by-vector serial multiplier is described in [11]. The main components of this multiplier are: an unsigned 8-bit integer multiplier (without CPA), carry-save to borrow-save converter, sign inclusion and serial adder. The 8-bit integer multiplier uses Booth recoding and an array structure for bit reduction.

In each run for the serial computation of the recurrence equation, one input of the digit-by-vector multiplier has a fixed value and the other input has a different digit in each cycle, the output of the multiplier corresponds to the product of the digit (fixed) and the vector composed by the digits imposed serially to the other input. On-line modules are used to avoid stalling the unit after each iteration, to wait for the output digit selection. Using on-line modules in the data path, the most significant digits of the residual are computed first, and the selection can begin while the other digits of the residual are being generated. This way, the selection circuit can be designed using simple and slow circuits.

The differences between the data path of the multiplier, divider and square-root are mainly in the selection function and interconnections. In the Figure, we show the different schemes using differently shaded lines. The selection function must also be modified for each operation. The recoder is used only for the VLP Multiplier. which has a simpler selection function than the other two operations. The adder that has the scaled residual and C as inputs may be removed from the data path for VLP Multiplication. The decision on the removal of this component depends on the reconfiguration time and area that can be freed. The area of the on-line adder is very small compared to the multipliers' area, so, it seems reasonable to keep the component configured in the system.

The next table shows the memory elements that are connected to the inputs of the data path.

Operation	A	B	C
VLP Multiplication	X	Y	zero value
VLP Division	Z	Y	X
VLP Square-root	Z	Z	X

4 Reducing Reconfiguration Time

The suitability of the proposed organization to FPGA technology comes from the following features:

- alows adjustment of the data path according to the hardware resources available in the reconfigurable coprocessor.
- the broadcast of signals is reduced by the utilization of digit serial components.
- only simple modifications of the data path are necessary to change from one arithmetic operation to the other (as shown in Figure 1), reducing the reconfiguration time.
- the design avoids allocation of resources that are not required for the operation being executed: e.g., muxes for data bus multiplexing, multiple selections, and complex control.

When using an FPGA chip that does not allow partial reconfiguration, the main components of the data path shown in solid lines in Figure 2 may be mapped to one (or some) FPGA chip(s) and are not reconfigured again for these arithmetic operations (other operations may require reconfiguration of the FPGAs). The components and interconnections shown in dashed lines can be executed outside the chip, on another FPGA chip. This other chip can also implement the control circuit for the particular arithmetic operation.

For FPGAs with partial reconfiguration capability, it is possible to modify the interconnections without affecting the components, or even swap in and out the selection modules and control, depending on the required arithmetic operation. In this way, the reconfiguration time would be reduced compared to the total configuration time.

5 Estimates of Area and Execution Time

The control part of the arithmetic coprocessor uses an area that depends more on the algorithm complexity than the addressing space of the memory elements. The data path, on the other hand can be adjusted to occupy the most of the remaining resources in the chip, constrained only by I/O limitations. In this paper we do not discuss the use of multiple chips, but we believe that the solution for this problem can be found with the allocation of different units in the data path to different chips or the unfolding of iterations in the computation of the recurrence equation. The local connections that are used in such an organization provide good conditions for data transmissions since most of them are point-to-point connections.

The implementation of the VLP multiplier in radix 256 was done for the XC4013 (576 CLBs) and we obtained the following area figures:

Component	Area (CLBs)
Data path	276
Control	209

Part of the area considered in the control part were used for bus interface and memory interface. The data transfer in this implementation was simplified the most and is composed of two phases: (1) the host transfers the operands and triggers the operation, (2) when the coprocessor finishes the task the host reads the results from the local memory. A more sophisticated interface could be used, allowing the coprocessor to work on the input operands as soon as some of the most significant operand digits were available. The host could also start reading the results and perform digit conversion as the output digits are generated. This approach would lead to an higher level of parallelism between the host and coprocessor and improve the performance.

In this implementation of the VLP multiplier for radix 256 in a XC4013-5, the data path has a critical path delay of 69.1ns for a non-pipelined version. This delay sets the maximum clock frequency to 14MHz. The critical path delay is dominated by the propagation time of the digit-by-vector multiplier that is constituted by the following delays: booth recoding, partial product generation, bit reduction using carry save adders (no CPA stage was included), digit code conversion (carry-save to borrow-save form), sign correction of the result, on-line adder critical path and interconnect.

Operation	Cycles
VLP Multiplication	$\frac{m^2+9m}{2}$
VLP Division	$\frac{m^2+23m}{2}$
VLP SQRT	$\frac{m^2+17m+6}{2}$

Table 1: Number of cycles

As the data path is similar for other VLP operations, we expect to have the same clock cycle time for VLP division and square-root.

Since the target FPGA board has a fixed interconnect between FPGA chip and one local memory bank, we perform three cycles for each data path cycle: one to read the operand digits, one to read the residual digit and one to write the residual digit and output digit.

Assuming that there is no pin limitations, i.e., all reads and writes to memory elements are executed in the same cycle, the total number of clock cycles for each of the arithmetic operations is shown in table 1 as a function of m, the number of digits in the input operands. The equations shown were obtained by analysis of the algorithms and simulation results. The time expend in the host operations were not considered.

Implementation of VLP division and VLP square-root is currently in progress and the precise experimental result will be available for the final version of this paper.

6 Previous Work

For the best of our knowledge, the only other system implemented in FPGAs for Variable Long-Precision computations was described in [6].

This system uses the classical algorithm for multiplication and digit-recurrent algorithm for division The multiplication algorithm uses only one multiplier that incurs in larger broadcast delays than our approach. The on-line multiplication is also better when the coprocessor is used to perform multiplication of long-precision significands of floating point numbers, and a product with twice the precision of the input operands is not required. The number of cycles for long-precision multiplication in this system is $2(m^2+m)$. Considering our simple implementation with limited I/O bandwidth, we get $\frac{3(m^2+9m)}{2}$, that is roughly 25% better than the VLP multiplier proposed in [6].

7 Acknowledgements

This work was partially supported by Xilinx and Virtual Computer Corporation through the State of California Micro grant "Variable-Precision Arithmetic Structures and Algorithms for Field-Programmable Gate Arrays." We thank S. Kelem of Xilinx and S. Casselman and J. Schewel of Virtual Computer Corporation for interest and support.

References

[1] A. L. Abbott, P. M. Athanas, L. Chen, and R. L. Elliott. Finding Lines and Building Pyramids with Splash 2. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 155–161, Napa Valley, California, April 1994.

[2] M. Dahl, J. Babb, R. Tessier, S. Hanono, D. Hoki, and A. Agarwal. Emulation of the Sparcle Microprocessor with the MIT Virtual Wires Emulation System. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 14–22, Napa Valley, California, April 1994.

[3] M. D. Ercegovac. On-line Arithmetic: An Overview. In *Real Time Signal Processing VII - 495*, pages 86–93. SPIE, 1984.

[4] M. D. Ercegovac and T. Lang. On-line Arithmetic: A Design Methodology and Applications in Digital Signal Processing. In *IEEE Workshop on VLSI Signal Processing*. IEEE, 1988.

[5] D. T. Hoang. Searching Genetic Databases on Splash 2. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, April 1993.

[6] C.-Y. Hsu. Variable Precision Arithmetic Processor in FPGAs. Master's thesis, University of Toronto, 1996.

[7] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Publishing Co., 1969.

[8] M. J. Schulte and E. E. Swartzlander Jr. Hardware Design and Arithmetic Algorithms for a Variable-Precision, Interval Arithmetic Coprocessor. In *IEEE 12th Symposium on Computer Arithmetic*, pages 222–229, 1995.

[9] A. Skaf and A. Guyot. VLSI Design of On-line Add/Multiply Algorithms. *IEEE Int. Conference on Computer Design*, pages 264–267, 1993.

[10] A. F. Tenca. *Variable Long-Precision Arithmetic (VLPA) for Reconfigurable Coprocessor Architectures*. PhD thesis, UCLA, 1998. in preparation.

[11] A. F. Tenca and M. D. Ercegovac. A High-Radix Multiplier Design for Variable Long-Precision Computations. In *31st Asilomar Conference on Signals, Systems and Computers*, Monterey, Nov. 1997.

[12] K. S. Trivedi and M. D. Ercegovac. On-line Algorithms for Division and Multiplication. *IEEE Trans. on Computers*, C-26(7):681–687, 1977.

[13] P. K.-G. Tu. *On-line Arithmetic Algorithms for Efficient Implementation*. PhD thesis,

University of California, Los Angeles, Sept 1990.

[14] J. E. Vuillemin, P. Bertin, Roncin D., M. Shand, H. Touati, and P. Boucard. Programmable active memories: Reconfigurable systems come of age. *IEEE Transactions on VLSI Systems*, 4(1):56–69, March 1996.

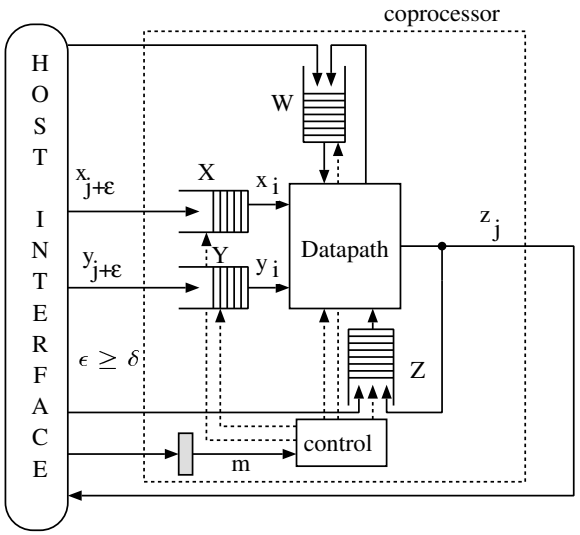

Figure 1: Coprocessor organization

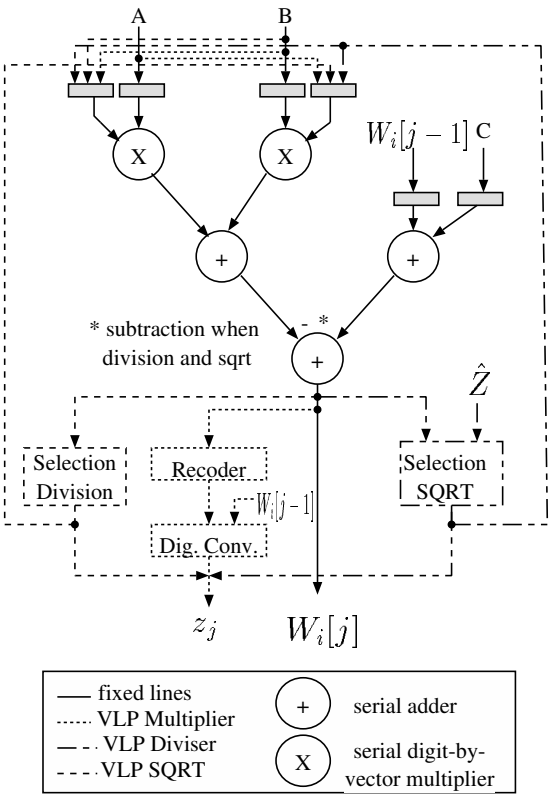

Figure 2: Data path of the coprocessor

On-line Multiplication

$W[j] = r(W[j-1] - z_{j-1}) + r^{-\delta_\times}(x_{j+\delta_\times-1}Y[j] + y_{j+\delta_\times-1}X[j-1])$

$W[0] = X[0]Y[0]$

$z_j = Sel(\hat{W}) = rW_0[j] + W_1[j]$

$\delta = 2$

$\frac{1}{r} \leq X < 1, \frac{1}{r} \leq Y < 1$

On-line division

$W[j] = rW[j-1] + x_{j+\delta-1}r^{-\delta+1} - y_{j+\delta-1}Z[j-2]r^{-\delta+1} - rz_{j-1}Y[j]$

$W[0] = X[0]$

$z_j = Sel(\hat{W}) = \lfloor \hat{W} + 0.5 \rfloor$

$\delta = 3$

$\frac{1}{r} \leq X < 1, 1-\alpha < Y < 1+\alpha$

On-line square-root

$W[j] = r(W[j-1] - z_{j-1}(z[j-1] + z[j-2])) + x_{j+\delta-1}r^{-\delta+1}$

$W[0] = X[0]$

$$z_j = Sel(\hat{W}, \hat{Z}) = \begin{cases} \left\lfloor \frac{\hat{W}[j-1]}{2\hat{Z}} + \frac{1}{2} \right\rfloor & \text{if } 0.5 \leq \hat{Z} < (1 - r^{-1}) \\ \left\lfloor \frac{\hat{W}[j-1]}{2} + \frac{1}{2} \right\rfloor & \text{if } (1 - r^{-1}) \leq \hat{Z} < 1 \end{cases}$$

$\delta = 3$

$\frac{1}{4} \leq X < 1$

Table 2: Parameters and equations for on-line arithmetic (High-radix only)